Lições de VBA do Excel

225	225	159
222	222	189
204	204	153
238	238	201

L.A.Bertolo

O **Excel** é um programa que tem um grande potencial, mas a maioria do pessoal o maneja de forma simples, utilizando somente as suas opções básicas. Porém há algo muito importante que tenho a lhes dizer. O **Excel** conta com uma linguagem muito poderosa chamada **Visual Basic**, ou melhor, somente com uma parte da linguagem, mas que permite resolver os problemas mais facilmente. Porém, devemos aprender a programá-lo e para isso é que existe este curso. As pessoas poderiam dizer que este curso é de um nível muito alto, e talvez o seja, mas é fácil de aprender, já que se manipularão as coisas termos simples, e eu gosto de falar com palavras que todo mundo entenda, por isso o torna mais fácil. A programação empregada neste este curso ou as estruturas que aparecerão são criadas pelo próprio aplicativo, já que para manejar a programação de **Visual Basic** do **Excel** é necessário ter muita criatividade, cada pessoa pode criar estruturas diferentes, mas que trabalhem de forma igual. Assim mãos à obra e divirtam-se!

FASES DO CURSO

Fase # 1: Desenvolvendo Procedimentos (Macros) VBA em Excel

Descubra o Gravador de Macros que funcionará para você todas às vezes, mesmo quando você se tornar um expert. Descubra então o ambiente amigável *Visual Basic Editor* onde todas as coisas acontecem no VBA. Aqui temos **9 lições** para fazer você se sentir mais confortável, de modo que você se diverte desenvolvendo procedimentos VBA (macros).

Lição 1: Introdução - Lição 2: Vocabulário das Macros do Excel - Lição 3: O Gravador de Macros Excel - Lição 4: O Visual Basic Editor - VBE - Lição 5: Modificando Macros no Visual Basic Editor - Lição 6: Criando Macros no Visual Basic Editor - Lição 7: Testando Macros no Visual Basic Editor - Lição 8: Eventos VBA - Lição 9: Segurança e Proteção no VBA

Fase # 2: O Código VBA

É quase o Inglês, mesmo assim existem maneiras para falar ao Excel. Nestas **8 lições** você aprenderá trabalhar com o Application (o próprio Excel), com pastas, planilhas, células e ranges, caixas de mensagens, caixas de entradas de dados e erros.

Lição 1: Palavras e sentenças (código) no VBA para o Excel - Lição 2: Trabalhando com o Application - Lição 3: Trabalhando com Workbooks - Lição 4: Trabalhando com Worksheets - Lição 5: Trabalhando com Células e Ranges - Lição 6: Trabalhando com Caixas de Mensagens - Lição 7: Trabalhando com Caixas de Entrada - Lição 8: Trabalhando com Erros

Fase # 3: VBA Avançado para o Excel

Vamos agora nos aprofundar mais . Em **12 lições** você aprenderá como trabalhar com variáveis, declarações e funções. Daí então descobrir os databases no Excel e o código para trabalhar com eles. Depois a parte divertida, trabalhar com userforms e controles. Como criar e/ou modificar Menus? Use os controles ActiveX nas próprias planilhas e gráficos. Vamos trabalhar com barras de ferramentas. Finalmente como criar os Suplementos.

Lição 1: Código VBA do Excel para Variáveis - Lição 2: Trabalhando com Declarações - Lição 3: Trabalhando com Funções - Lição 4: Trabalhando com databases - Lição 5: Criando UserForms - Lição 6: Adicionando Controles aos userforms Lição 7: Código VBA do Excel para UserForms - Lição 8: Código VBA do Excel para Gráficos e Arquivos Seqüenciais – Lição 9: Criando e Modificando Menus – Lição 10: Usando controles ActiveX nas planilhas e gráficos – Lição 11: Trabalhando com Barras de Ferramentas – Lição 12: Os Suplementos

Saudação

Entre no maravilhoso mundo do VBA (macros) com estas **29 lições**. Não somente a linguagem de programação para desenvolver macros (**Visual Basic for Application ou VBA**) é fácil para aprender, mas o seu ambiente de desenvolvimento é extremamente amigável para o usuário. Desde 1997, você pode desenvolver muito mais que comandos de macro (macros) com VBA. O VBA é para todo mundo. Ele é uma linguagem muito simples e projetada para os **USUÁRIOS**. Com o VBA você pode desenvolver pequenos procedimentos (10 linhas ou menos) que lhe pouparão um monte de tempo, eliminando tarefas repetitivas. Você também pode desenvolver programas complexos e poderosos por uma fração do preço que ele poderia lhe custar para conseguí-los desenvolvidos por especialistas em computação. A experiência tem me mostrado em incontáveis vezes que certas análises e relatórios sofisticados **SOMENTE** podem ser desenvolvidos com o Excel e o VBA. Isto é porque o Excel é o programa de análise e relatório mais usado no mundo.

Divirta-se

Fase # 1: Desenvolvendo Procedimentos (Macros) VBA em Excel.

Lição 1: Introdução ao VBA do Excel

O VBA é "*Visual Basic for Application*". Ele é uma linguagem de programação que permite usuários a programarem **macros** para executar tarefas repetitivas ou complexas automaticamente dentro do Excel. Com o VBA do Excel você pode desenvolver pequenos procedimentos (macros) que tornarão a sua vida profissional mais fácil e lhe permitir fazer mais em menos tempo. Mas o VBA também é uma linguagem de programação muito poderosa com a qual você pode desenvolver dentro do Excel programas reais que executem em poucos minutos tarefas muito complexas. Com o VBA do Excel você pode desenvolver um programa que faça **EXATAMENTE** o que você precisa.

Lição 2: Vocabulário das Macros VBA do Excel

Se como eu você for um contador você tem dispensar anos numa faculdade para aprender lançamentos de débitos e créditos. Quando você falar com outros contadores eles entenderão a respeito do que você está falando, mas se você falar com advogados ou engenheiros eles não têm uma pista. Em programação existem certos termos que você precisa aprender sobre eles para descobrir o básico e o mais complexo. Estes termos são "objetos", "propriedades", "métodos" e outros. Aqui está o que eles são e o que eles significam.

Lição 3: O Gravador de Macros do Excel

Quando você iniciar o gravador de macro você pode escolher para anexar a macro a uma tecla do teclado. Quando ela foi gravada você apenas clicou na tecla que você escolheu e a macro por outro lado fez isto novamente. Desenvolva a sua primeira macro Excel com esta lição passo a passo sobre como usar o Gravador de Macros do Excel. Após 12 anos de programação eu ainda uso o Gravador de Macros do Excel diariamente. Ele é o melhor professor e o maior dos assistentes.

Lição 4: O Visual Basic Editor

O Visual Basic Editor - VBE é o subprograma em Excel com o qual você desenvolve, testa e modifica as suas macros, em Excel. Ele é um ambiente muito amigável para o usuário. Você não precisa instalá-lo, ele já está lá toda vez que você abrir o Excel. Aqui está como configurá-lo para tornar a sua experiência tanto quanto possível, fácil, agradável e útil. Desenvolver macros, criar *useforms* com controles (botões de comando, caixa de listagens, caixas de texto, caixas de combinação (listas *drop-down*), etc..). Configure e mude todas as propriedades dos componentes.

Lição 5: Modificando Macros no Visual Basic Editor

Numa lição anterior você gravou uma macro mas agora que você entendeu o fundamento dela você quer fazê-la MAIS. Aprenda como modificar uma dada macro no Visual Basic Editor - VBE.

Lição 6: Escrevendo Macros no VBEditor

Neste capítulo criamos uma nova macro (procedimento VBA) estigmatizada. Escreveremos manualmente as sentenças (códigos) que o Excel entende. Descubra mesmo assim uma linguagem muito poderosa.

Lição 7: Testando Macros no Visual Basic Editor

Agora é a vez de testar a macro passo a passo. Você não quer somente testá-la passo a passo, mas você quer vê-la funcionando no Excel quando você rodá-la lentamente. Aprenda como se desdobra a sua tela para ver o Visual Basic Editor e o Excel ao mesmo tempo. Você clica na tecla F8 para rodar a macro passo a passo e você vê as coisas acontecendo no Excel. Você não gostou do que viu então você volta atrás para modificar o código e vá adiante novamente.

Lição 8: Eventos VBA do Excel

Uma macro inicia quando um evento acontece. Uma macro em Excel usualmente inicia quando você clica num botão como em qualquer outro programa grande. O clicar é chamado de um evento. Existem mais coisas para os eventos do que apenas clicar num botão. Você pode clicar num item de menu, num ícone de uma barra de ferramentas, numa tecla do teclado mas a abertura de uma pasta também é um evento, mover-se para uma certa folha é um evento e apenas mover o cursor sobre um botão pode iniciar uma macro. Aprenda sobre todos os eventos, o mais usual e alguns dos menos usuais. Assim basicamente quando um evento acontecer você chamará uma macro com uma sentença muito simples como "**Call proMyMacro**".

Lição 9: Segurança e Proteção no VBA do Excel

Aprenda como proteger seu computador de vírus transmitidos pelas macros. Aprenda como proteger suas macros (procedimentos VBA) com uma senha (password) e aprenda como usar a propriedade "VeryHidden" da folha para **REALMENTE** ocultar uma planilha contendo salários e preços. Quando ela estiver *very hidden* a folha está lá mas somente **VOCÊ** e uns poucos selecionados que sabe o *password* para a suas macros podem acessá-las.

Para quem serve estas 9 lições?

Estas lições foram projetadas para os contadores, planejadores de produções, supervisores de produção, pessoal de vendas, analistas financeiros e outros analistas de dados de negócios.

O VBA é uma linguagem de programação para os usuários. Ela é simples e fácil para aprender. Qualquer um pode desenvolver macros simples (procedimentos de VBA) e com tempo e interesse você pode chegar a um ponto onde você pode desenvolver muitos procedimentos complexos para efetuar tarefas muito sofisticadas. Ela fornecerá a

você uma ferramenta que vai além daquele feijão com arroz geralmente praticado por muitos todo o dia.

Você pode esperar em desenvolver aplicações muito brilhantes se você:
- conhecer bem os seus dados (financeiros, contábeis, vendas, marketing, estoque, etc..)
- conhecer bem o Excel (SUMPRODUCT, INDEX/MATCH e outras funções importantes do Excel, mais as funcionalidades do database do Excel). Não precisa aprender sobre funcionalidades muito raramente usadas como *pivot tables*, *solver*, *atingir metas*, *cenários*, *group* e *outline* e outras ferramentas especializadas.

É esta maleabilidade que faz com que a análise dos dados tenha uma apresentação realmente profissional e somente vista em versões de programas estatísticos SPSS, etc..

Se você tiver mais do que 10 anos de experiência no seu campo de especialização (Contabilidade, Finanças, Produção, Engenharia, etc...) você deve querer aprender VBA para tornar-se um desenvolvedor VBA do Excel (o mercado e o dinheiro são muito bons). Precisa para isso ter conhecimentos básicos de Excel.

É nossa intenção desenvolver ferramentas que permanecem na obscuridade e abrir a você novas fronteiras na arte de analisar dados.

O que são estas 9 lições?

Por favor, não pule o primeiro capítulo sobre o vocabulário. Algumas das coisas que eu preciso falar para você sobre elas mais tarde soarão como Marcianas. Se você falar Marciano, então, siga em frente....

Daí então eu lhe apresentarei ao gravador de macro, uma característica única que permite a você criar macros enquanto você está fazendo manualmente o que você gostaria que a macro fizesse. Daí então nas próximas 4 lições, eu darei uma olhada no Visual Basic Editor (VBE), o **ambiente** de programação mais amigável do usuário no mercado.

Você então aprenderá sobre os EVENTOS que disparam as macros (lição 8), controles (botões de comando) e os userforms.

Finalmente há a lição 9 sobre segurança e proteção.

Lição #01: Introdução ao VBA do Excel

O VBA é "*Visual Basic for Application*". É uma linguagem de programação que permite os usuários a programarem macros para efetuar tarefas complexas dentro de uma aplicação. Com o VBA do Excel você pode desenvolver pequenos procedimentos (macros e/ou funções) que tornarão sua vida profissional mais fácil e lhe permitir fazer mais em menos tempo. Mas o VBA também é uma linguagem de programação muito poderosa com a qual você pode desenvolver dentro do Excel programas reais que efetuarão em uns poucos minutos tarefas muito complexas. Com o VBA para o Excel você pode desenvolver um programa que faça **EXATAMENTE** o que você precisa.

Visual Basic e VBA

O que é VBA?

Existe o VBA para o Excel, o VBA para o Word, o VBA para o Project, o VBA para o Access. Você pode usar o VBA em Autocad, Oracle e mesmo com o EssBase. Todos os VBA's são semelhantes, mas variam nos objetos com os quais eles trabalham. Todos eles têm a mesma fonte Visual Basic (VB).

Existe um VBA para cada Aplicativo

De onde veio o VBA?

Nos anos “70” uma nova linguagem de computador apareceu chamada **Basic**. Eu a usei em calculadoras avançadas (TI 59). Naquele tempo, a RAM (memória) e a CPU's (o cérebro do computador) eram muito, muito pequenos e você estava limitado no que você poderia fazer com a linguagem. Lembro-me ainda que no final daquela década eu era Estagiário numa Empresa de Construção Civil, no Rio de Janeiro, e tinha programado uma folha de pagamento completa (salários, impostos, benefícios) para 500 empregados na calculadora *Texas Instrument 59*.

Com os computadores de hoje em dia veio o **Visual Basic** (VB). Ela é ainda o Basic, mas, um monte de elementos está pré-programado, tornando a tarefa do usuário muito mais simples. A Microsoft adotou a linguagem e introduziu-a como um componente de todas as suas aplicações.

Deveria eu aprender VBA ou VB?

Digamos que VB é para programadores e VBA é para usuários (ou desenvolvedores). Pessoalmente, eu tive várias horas de treinamento em VB, mas eu faço todo o meu trabalho no VBA para Excel. Você não precisa aprender VB para ser bom em VBA. Eu tenho adotado o VBA para o Excel porque a maioria das funções que eu preciso para organizar e analisar os dados de negócios está pre-programada no Excel. Custa muito menos para desenvolver aplicações financeiras e administrativas no Excel do que re-inventar a roda e fazer tudo em VB.

Qual é a diferença entre VB, VBA para Excel, para Access, para Word, para Project, etc.?

Para responder esta questão eu lhe pediria para imaginar um encontro de cinco pessoas falando Português, um advogado, um contador, um médico, um químico e um psicólogo. Todos eles falam Português, mas quando for à vez de falar a respeito do seu trabalho diário nenhum deles entende totalmente o outro. É o mesmo com o VB e

VBA's. Os objetos, propriedades e métodos variam substancialmente de uma sublinguagem para a outra.

Mas basicamente, se você falar Inglês você pode falar VB ou quaisquer dos VBA's.

O Projeto VBA

Ao iniciarmos uma **Pasta de trabalho** no Excel  é automaticamente criado um **Projeto VBA**  atrelado a ela. Um **Projeto VBA** é o equivalente à Pasta de trabalho no Excel.

O **Projeto VBA** é composto por:

- Pastas de Trabalho
- Planilhas
- Formulários
- Módulos
- Classes

Estes dois elementos são próprios do Excel

Estes três elementos são próprios do VBA

Gostaríamos de salientar que durante o processo de desenvolvimento em VBA, haverá situações onde a resposta será encontrada com muito trabalho. Para assegurar que o esforço não foi em vão, é boa prática documentar o que está sendo feito, e como a solução foi encontrada. Quando todo o trabalho é documentado você assegura um documento de referência para o futuro.

Prática #01: Introdução ao VBA do Excel

Lição 2: Vocabulário das Macros VBA do Excel[1]

No universo do VBA do Excel (Macros) existe algum vocabulário (terminologia) que você poderá querer adotar de modo que outros possam lhe entender e de modo que você possa entender os outros. Isto ocorre em qualquer linguagem de programação. Alguns dos termos são bem conhecidos, e requerem pouca explicação. Outros são termos que você não deve ter encontrado antes. O que segue é uma lista dos **termos chaves** do *Visual Basic* que são cobertos nesta lição:

Objeto
Propriedade
Método
Argumento
Coleções
Procedimento
Subrotina
Função
Macro
Declaração
Expressão

Esta lição dará uma explicação destes termos junto com ilustrações de modo que quando você encontrar estes termos nas outras lições você não ficará confuso. Também, você encontrará que outras lições dão uma cobertura em profundidade destes termos.

O que é uma MACRO?

Macros do Excel: É uma série de passos que se armazenam e que podem ser ativados com alguma *tecla de controle* (Ctrl) mais uma letra. Por exemplo, todos os dias emprego freqüentemente em minhas células os mesmos passos: Tamanho de Fonte (Tamanho da letra), Negrito, Fonte (Tipo de letra) e Cor de Fonte (Cor de Letra). Para não estar repetindo estes passos, posso armazená-los numa macro e, quando executar esta macro, os passos antes mencionados se executarão quantas vezes eu desejar. Enfim, uma macro é uma série de instruções realizadas pelo Excel, ao invés de você.

As macros ou sub-rotinas começam com a palavra Sub, seguida pelo nome da macro. Elas terminam com as palavras End Sub.

O que segue é um exemplo de uma subrotina ou macro:

```
Sub AlôMundo()
    MsgBox "Alô Mundo"
End Sub
```

[1] **NOTA:** Esta lição é sobre o vocabulário de programação. Se você estiver procurando códigos (as palavras usadas para falar ao Excel em VBA) vá para a seção de código VBA (11 páginas deles)..

PROCEDIMENTOS: São conjuntos de declarações que realizam uma tarefa específica. Procedimentos podem ser ou macros ou funções. As Funções serão vistas mais abaixo.

Eventos: Para um procedimento iniciar e ser executado, um evento deve acontecer. O evento é, portanto, a **causa** de uma *ação* (ou método, como veremos abaixo). Um evento que todo mundo conhece é o clique (click) no botão. Outros eventos incluem abertura de uma pasta, ativação de uma folha, modificação do valor de uma célula, etc.

Código: Quando você estiver escrevendo as instruções em VBA, diz-se que você está codificando, ou escrevendo código. Mostre-me seu código e eu lhe mostrarei a mina. Todos os pedaços de códigos estão aqui apresentados na fonte Courier New de tamanho 11.

Componentes dos **Códigos**

Para escrever **código** em VBA para macros você usará 5 tipos de componentes: objetos, propriedades, métodos, funções e declarações:

O QUE SÃO OBJETOS ?

Antes de definir o que é um objeto, observe os exemplos seguintes:

Televisões
Carros
Lâmpadas
Mesas
Lápis
Computadores

Objetos do dia a dia

Não é exatamente o que você esperava ver? Objetos à sua volta são as coisas que você pode *ver, tocar, sentir* ou *mudar*. Em *Visual Basic*, objetos são as mesmas coisas. Eles são basicamente qualquer coisa que você possa selecioná-la. O que segue são alguns dos objetos mais comuns do Visual Basic do Microsoft Excel:

Células
Planilhas
Pastas
Folhas de gráficos
Gráficos encaixados
Qualquer coisa que você desenhar numa folha
Linhas de um diagrama
Legendas de diagramas
Eixos X e Y num diagrama
Barras de ferramentas
Botões
Menus

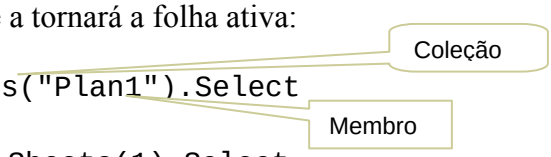
Objetos do VBA

Eles são os blocos de construção dos seus **projetos** Excel e, são (entre outros), o Application (o próprio Excel), as Workbooks (Pastas), as Worksheets (Folhas de Planilhas e/ou Graficos), as Células e os Intervalos (Ranges), os Charts, os Desenhos, os Controles (botões de comando, caixas de texto, caixa de listagens, etc..).

Você deve ter notado que grandes objetos, freqüentemente, incluem objetos menores. Por exemplo, um objeto **pasta** contém objetos *folhas de planilhas*. Um objeto **planilha** contém objetos *células*. Se você olhar ao seu redor, o mesmo se aplica. Um objeto **casa** contém objetos *salas*. E objeto **sala** contém objetos *móveis* e *utensílios*.

SOBRE COLEÇÕES...

Uma **coleção** é um grupo de objetos que são todos do **mesmo tipo**. As coleções estão quase sempre no plural. Exemplos de coleções são pastas, folhas de planilhas, e diagramas. Os membros individuais de uma **coleção** podem ser referenciados especificando a coleção e, daí então, ou o nome do membro, ou um número que indique a posição do membro na coleção. Por exemplo, se a primeira folha de planilha na coleção Sheets é chamada Sheet1, uma das duas declarações seguintes selecionará esta folha e a tornará a folha ativa:




```
Sheets("Plan1").Select
```

ou

```
Sheets(1).Select
```

Um dos modos principais que o Visual Basic funciona é especificando um objeto e daí então uma **ação** a ser imposta ao objeto. É por isso que o *Visual Basic* é chamado de linguagem orientada ao objeto.

SOBRE AS PROPRIEDADES...

Se você observar os objetos ao seu redor, você verá que eles têm propriedades. Por exemplo, uma casa tem uma propriedade **endereço**, uma pessoa tem uma propriedade **nome**, uma sala tem uma propriedade **cor**, e uma televisão tem uma propriedade ativa (ou está ligada ou desligada). Os objetos no *Microsoft Excel* também têm **propriedades**. Por exemplo, uma célula tem uma propriedade **endereço**, uma folha de planilha tem uma propriedade **nome**, e uma fonte tem uma propriedade **cor**.

Como os exemplos acima ilustram, as **propriedades** são características de um objeto. Alguns objetos no Microsoft Excel têm muitas propriedades, outros têm muito poucas. A maioria das propriedades dos objetos é expressa como números (tais como o tamanho da margem), *strings* de textos (tal como um nome de uma folha), ou valores Booleanos (True ou False).

Se você observar os objetos ao seu redor, você pode mudar algumas propriedades de um objeto, mas nem sempre todas as propriedades. Por exemplo, você pode ligar e desligar sua televisão. E você pode pintar uma sala para mudar sua cor. Mas você não pode mudar onde a sua casa está localizada. O mesmo se aplica ao Microsoft Excel. Você pode mudar quase todas as propriedades que você encontrar, mas existem umas poucas que você não pode mudar. Por exemplo, você pode mudar o valor de uma célula (a propriedade Valor), mas você não pode mudar o *número de células* de uma folha.

O *Visual Basic* funciona muito do jeito que você faz no Microsoft Excel. Se você quiser entrar com um valor numa célula, você primeiro seleciona a célula daí então

você digita nela o valor. No Visual Basic, você especificará a célula, digita um **ponto**, especifica uma propriedade Valor, digita um sinal de **igual** e digita o valor para a célula. Por exemplo:

Como entrar com um valor numa célula

```
ActiveCell.Value = 5
```

Neste caso ActiveCell é uma *palavra-chave* que se refere à **célula ativa**, que é a célula destacada pelo cursor. Value é uma *palavra-chave* que representa o **valor** da propriedade de uma célula. Por favor, note que são muitas as maneiras de se referir às células além de ActiveCell. Estas maneiras serão cobertas, em detalhes, nas lições posteriores.

Exatamente como você especifica sua cidade, rua, e número da casa num endereço, você pode fazer o mesmo no Visual Basic para melhor identificar uma célula particular:

Outra maneira

```
Workbooks("Custo.Xls").Sheets("Folha1"). _
                       Range("A1").Value = 5
```

Neste caso, vários objetos (uma pasta, uma folha de planilha e uma célula) são juntados com **pontos** para identificar uma célula particular. Dai então a propriedade (Value novamente neste caso) é ligada com um ponto. Finalmente um sinal de **igual** é entrado seguido por um número. A *string* acima diz ao Visual Basic que ajuste o valor da *célula* A1 da folha de planilha *Folha1* na pasta *Custo.Xls* para **5**. Por favor, note que célula A1 não tem que ser a célula ativa.

Se você quiser **encontrar** o valor de uma propriedade em vez de mudá-la, especifique o *objeto* e a *propriedade*. Normalmente, o valor que é retornado é armazenado numa variável ou usado numa declaração. Por exemplo, a declaração seguinte mostra uma caixa de mensagem que lhe diz o valor da célula ativa:

```
MsgBox ActiveCell.Value
```

Outra maneira é usar a janela local que será discutida na lição sobre depuração de suas macros.

A **declaração** seguinte armazena o nome da *folha de planilha ativa* numa variável chamada X, para uso posterior:

```
X = ActiveSheet.Name
```

Neste caso, ActiveSheet é o **objeto** e se refere à folha de planilha ativa. Name é a **propriedade**. A **variável** nesta declaração é chamada simplesmente de "X". Nomes mais compridos e mais descritivos de variável são permitidos.

Falar em propriedade é pensar então no **verbo ser/estar** ou **ter**.

A seguir, mostramos outros exemplos de propriedades de objetos do VBA:

Exemplos de Propriedade

`Selection.Font.Bold = True` .....A fonte está em negrito.

`Sheets("Talqual").Visible = True`.....A folha de planilha *Talqual* está visível

`ActiveWorkbook.Name` ..... A pasta ativa tem um nome

Exemplos de Propriedade

`Activecell.Value = 10` ou

`Activecell.Formula = "=A1+B2"` .... A célula pode ter um valor ou uma fórmula.

O conjunto de propriedades difere de um objeto para outro. Uma planilha não pode ser negritada e uma pasta não pode ter uma fórmula. Excel dirá quando você estiver tentando usar uma propriedade que não existe para o objeto que você está trabalhando.

SOBRE OS MÉTODOS

Um **método** é um termo do Visual Basic para uma *palavra-chave* que representa uma **ação** que você quer impor a um *objeto*. Por exemplo, Copiar, Recortar, e Colar são todos exemplos de métodos do Visual Basic. Como uma ação está associada com um objeto, o Visual Basic requer que você especifique o objeto, um **ponto** e daí então o método. Um sinal de **igual** não é usado. Na declaração seguinte,

```
Range("B9").Select
```

o objeto é a célula B9 e o método é a palavra-chave Select, que faz o ponteiro de célula mover-se para esta célula.

Na declaração seguinte,

```
ActiveCell.Copy
```

Ação = Método

ActiveCell é a palavra-chave que identifica a célula ativa. Copy é o **método**. Esta declaração diz ao Visual Basic copiar os conteúdos da célula ativa para o *clipboard* para ser usado mais tarde.

O que segue é um exemplo de uso do método Copy sobre uma célula que não é a célula ativa:

```
Range("A9").Copy
```

Falar em método é pensar então no **verbo fazer**. Você poderá querer fazer com que um objeto feche, abra, seja copiado ou colado, etc.. (`ActiveWorkbook.Close`, `Activecell.Paste`). Novamente, o Excel lhe dirá quando você estiver tentando usar um método que não se aplica ao objeto em uso.

Exemplo:

```
Sub proTeste ()
  Range("A2").Value= 2
  ThisWorkbook.Close
  Application.Quit
End Sub
```

Estas quatro linhas constituem um **procedimento** (macro) chamado "proTeste". Eu sempre nomeio minhas macros iniciando com o prefixo "pro" e a primeira letra

maiúscula para qualquer palavra significante compreendida no nome como proInfo, proDadosBrutos, proSempreque. Você verá mais tarde quão importante este prefixo e letra maiúscula costumam ser.

"Range("A2")", "ThisWorkbook" e "Application" são **objetos**, "Value" é uma **propriedade** e "Close" e "Quit" são **métodos**.

Este procedimento VBA será atribuído a um **objeto** botão (controle) e quando o usuário clicar nele (o **evento**) o procedimento VBA rodará (o **método** rodar será aplicado).

Este procedimento VBA diz ao computador que quando o usuário clicar no botão, a célula "A2" recebe um valor 2 e o Excel é fechado.

Aqui vão alguns códigos muito comuns:

Trasladar-se a uma Célula

```
Range("A1").Select
```

Escrever em uma Célula

```
Activecell.FormulaR1C1="Bertolo"
```

Letra em Negrito

```
Selection.Font.Bold = True
```

Letra Cursiva (Itálico)

```
Selection.Font.Italic = True
```

Letra Sublinhada

```
Selection.Font.Underline = xlUnderlineStyleSingle
```

Centralizar o Texto

```
With Selection
.HorizontalAlignment = xlCenter
End With
```

Alinhar à esquerda

```
With Selection
.HorizontalAlignment = xlLeft
End With
```

Alinhar à Direita

```
With Selection
.HorizontalAlignment = xlRight
End With
```

Tipos de Letra (Fonte)

```
With Selection.Font
.Name = "AGaramond"
End With
```

Tamanho de Letra (Tamanho de Fonte)

```
With Selection.Font
.Size = 15
End With
```

Copiar

```
Selection.Copy
```

Colar

```
ActiveSheet.Paste
```

Cortar

```
Selection.Cut
```

Ordenar Ascendente

```
Selection.Sort Key1:=Range("A1"), Order1:=xlAscending,
Header:=xlGuess, _
OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
```

Ordenar Descendente

```
Selection.Sort Key1:=Range("A1"), Order1:=xlDescending,
Header:=xlGuess, _
OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
```

Buscar

```
Cells.Find(What:="Bertolo", After:=ActiveCell,
LookIn:=xlFormulas, LookAt _
:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext,
MatchCase:= _
False).Activate
```

Inserir Arquivo

```
Selection.EntireRow.Insert
```

Eliminar Arquivo

```
Selection.EntireRow.Delete
```

Inserir Coluna

```
Selection.EntireColumn.Insert
```

Eliminar Coluna

```
Selection.EntireColumn.Delete
```

Abrir uma Pasta

```
Workbooks.Open Filename:="C:\Meus documentos\video safe 3.xls"
```

Gravar um Pasta

```
ActiveWorkbook.SaveAs Filename:="C:\Meus documentos\piscis.xls",
FileFormat _
:=xlNormal, Password:="", WriteResPassword:="",
ReadOnlyRecommended:= _
False, CreateBackup:=False
```

Esses seriam alguns códigos muito comuns no **Excel**, mas se você deseja pode gerar mais códigos de outras opções, é uma questão de que os ocupe.

EXERCÍCIOS:

1. Escreva uma declaração para o Visual Basic **colar** o conteúdo do clipboard na célula A21.
2. Escreva uma declaração para o Visual Basic **selecionar** a célula A10.
3. Escreva uma declaração para o Visual Basic **escrever** Bertolo na célula ativa.
4. Identifique nos códigos acima o objeto, a propriedade e os métodos

SOBRE AS FUNÇÕES...

Funções são conjuntos de declarações que retornam **um** valor. Exemplos de funções que são construídas no Microsoft Excel são a função Soma() e a função TIR().

As funções escritas por um usuário são chamadas User Defined Function (UDF's). Tais funções são úteis porque elas podem ser usadas nas células da planilha para fazer cálculos especializados não programados nas funções existentes no Microsoft Excel. Também, você pode usar funções para trocar complexas fórmulas de célula SE. Você pode, também, usar funções em suas macros.

O que segue é um exemplo de uma função UDF:

```
Function TamanhoDaCaixa(X, Y)
    TamanhoDaCaixa = X * Y
End Function
```

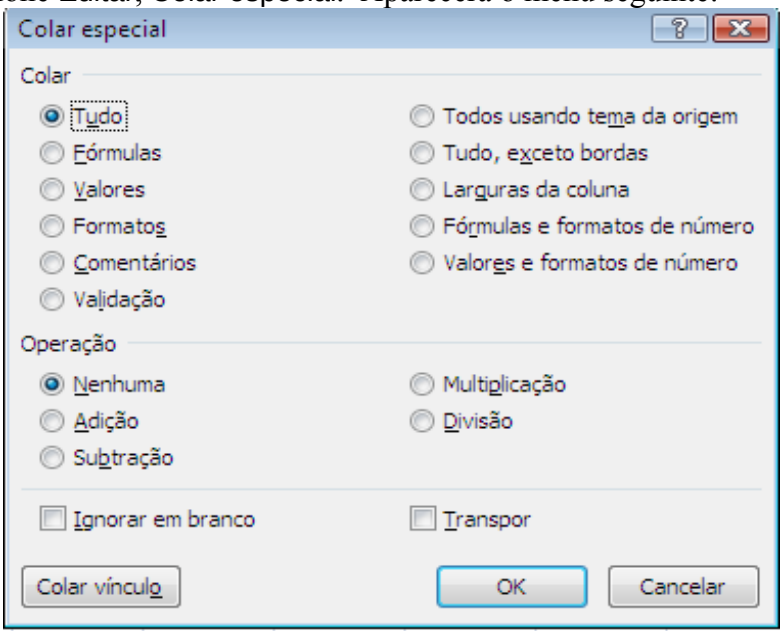
Esta função calcula o tamanho de uma caixa e retorna com **um** número baseado nos valores fornecidos para X e Y. Para usá-la, você deverá digitar = *TamanhoDaCaixa* (*célula referência, célula referência*) numa célula de uma planilha. Por favor, note que as **macros** e as **funções** podem ser muito complexas e poderosas!

Você pode, e você vai usar todas as funções Excel dentro de seu código VBA, mas você também pode usar as funções do VBA como UCASE, LCASE, NOW(), etc..

(`Activecell.Value=NOW()`)

FORNECENDO ARGUMENTOS

Argumentos são valores que são passados às **macros**, aos **métodos**, e às **funções**. Para ilustrar o que é um argumento, primeiro selecione uma célula e selecione Editar, Copiar do menu da barra de ferramentas Padrão do Excel.. Daí então, vá a outra célula e selecione Editar, Colar especial. Aparecerá o menu seguinte:



As diferentes seleções que você pode fazer neste painel representam os **argumentos** para o comando *colar especial*. A declaração seguinte do Visual Basic tem o mesmo efeito que as seleções na caixa acima:

```
ActiveCell.PasteSpecial paste:=xlValues, _
      operation:=xlNone, _
      skipBlanks:=False, _
      transpose:=False
```

Argumentos de um Método

Nesta declaração, as palavras: paste, operation, skipBlanks e transpose são os **argumentos** para o **método** PasteSpecial. Os valores são atribuídos a um argumento por uma combinação de **dois pontos** e um sinal de **igual**. Neste caso os valores ou são numéricos, especificados pelas constantes do Visual Basic (xlValues e xlNone), ou são valores Booleanos que são as palavras: Verdadeiro(True) ou Falso (False). As **constantes** do Visual Basic são apenas nomes para números. Usando nomes para números, torna as declarações do Visual Basic mais fáceis de entender e os valores mais fáceis de lembrar.

SOBRE AS EXPRESSÕES E DECLARAÇÕES

As macros, e as funções, são constituídas de **declarações**, e estas declarações são constituídas de **expressões**. Na função

```
TamanhoDaCaixa = X * Y
```

acima, X * Y é uma expressão e TamanhoDaCaixa = X * Y é uma declaração. Uma declaração pode se estender por múltiplas linhas, ou apenas ser de uma única linha de tamanho. Na macro *AlôMundo*, cada uma de suas linhas são declarações.

Outros exemplos de declarações são: IF..THEN, DO...LOOP, FOR...NEXT, WITH...END WITH, EXIT FOR, EXIT DO, EXIT SUB que veremos em detalhes em lições posteriores.

O PROJETO VBA

Um **projeto** VBA pode compreender 4 tipos de componentes: a *pasta*, as *planilhas*, os *módulos* e os *userforms*.

Componentes de um **Projeto**

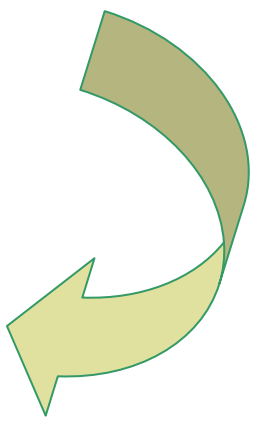
Workbooks (Pasta): A pasta é um arquivo Excel (algo.xls) também chamado de planilha. O objeto "ThisWorkbook" é a pasta para a qual a macro foi criada. O código:

```
ThisWorkbook.Close
```

é um *método* que fechará a **pasta** dentro da qual a macro ativa está rodando.

Worksheets (Planilha): Uma folha de planilha do Excel 2007 que pode ter 1.048.576 linhas e 18.278 colunas.

Módulos: É uma espécie de um arquivo no qual você guarda a maioria de suas macros (= sub-rotinas VBA). Módulos são criados e nomeados no Visual Basic Editor.



UserForms: São folhas de planilhas especializadas que você cria para permitir o usuário submeter parâmetros (valores). São usados extensivamente em bancos de dados (databases), contabilidade, produção e programas de vendas, porque não existem folhas regulares nestes ambientes.
Os **Controles** são os botões de comando, as caixas de verificação, os rótulos, as caixas de texto, as caixas de listagens, os botões de opção e outros utensílios que você colocar nas planilhas ou *userforms*.

REQUISITOS DE CONFIGURAÇÃO (SETUP) DE MÓDULO

Esta lição supõe que esteja ativada a opção Mostrar erros de sintaxe. Para confirmar isto no Visual Basic Editor, selecione Ferramentas, Opções, guia Editor, e verifique esta configuração. Nós recomendamos também que à medida que o seu nível de experiência no Visual Basic tenha aumentado que você desligue esta opção – ela pode ficar bastante irritante.

Também, no *Visual Basic Editor*, selecione Ferramentas, Opções e vá à guia Geral e certifique-se que a opção Interromper em erros não tratados esteja marcada.

SUMÁRIO

Esta lição introduziu a linguagem *Visual Basic*. Quanto mais você usar o *Visual Basic*, mais entendível a linguagem se tornará. A informação nas lições seguintes fornecerá exemplos adicionais dos **objetos**, **métodos**, **propriedades**, **macros**, e **funções** do *Visual Basic*.

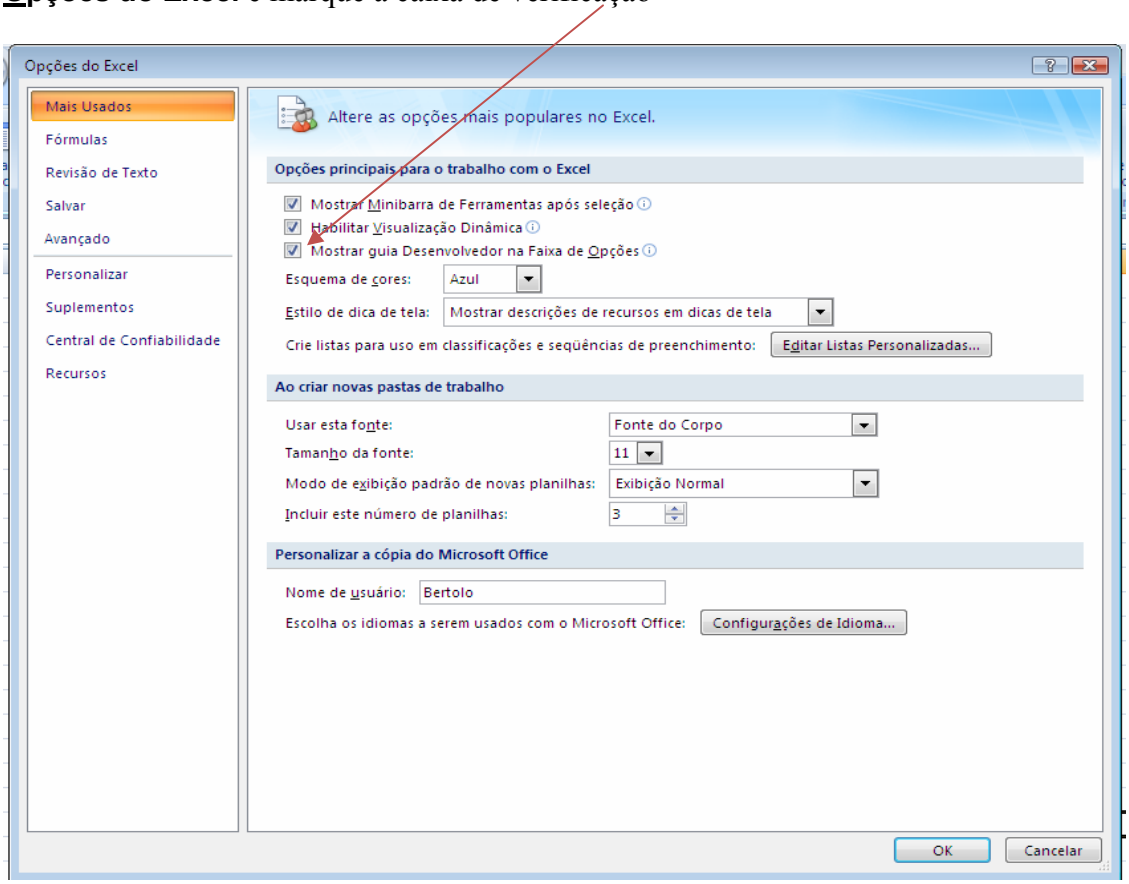
Lição 3: O Gravador de Macros Excel

Com o gravador de macro Excel você não pode desenvolver uma macro que danificará o Excel ou seu computador. Quanto mais corajoso você é em suas tentativas mais você aprenderá.

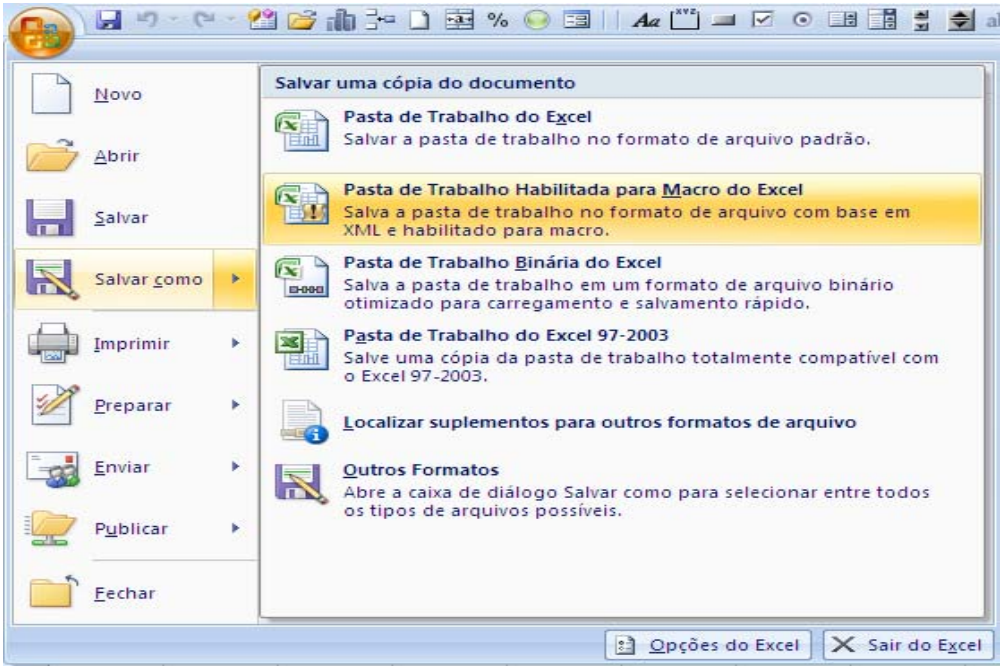
Eu tenho desenvolvido macros por algum tempo, e uso o gravador de macro Excel diariamente. Não para aprender qualquer coisa, mas para escrever **código** (palavras e sentenças VBA) para mim. O gravador de macro Excel é o melhor professor que você pode ter e permanecerá o melhor assistente para o resto de sua vida de desenvolvedor de VBA. Pessoalmente, eu uso muito o gravador de macro Excel para evitar em escrever código e fazer erros de ortografia.

Imprima esta página e siga as suas instruções passo a passo**.**

Antes de mais nada, caso esteja trabalhando com o Office 2007 procure ativar (se você ainda não fez) a guia **Desenvolvedor**. Para tanto vá ao **botão do Office** > **Opções do Excel** e marque a caixa de verificação

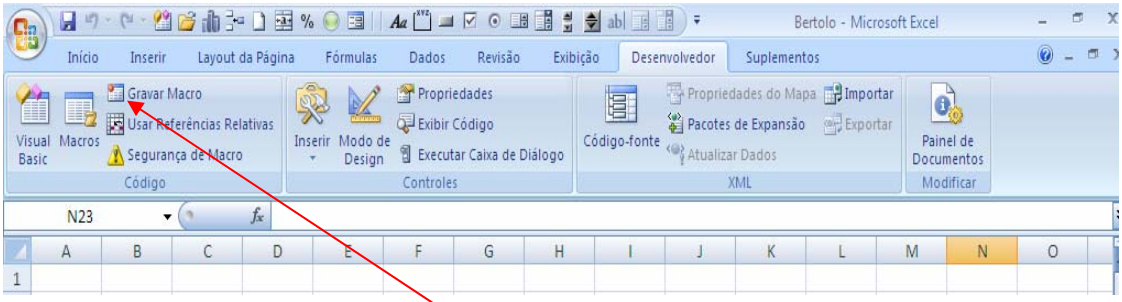


Lembre-se que no **Office 2007** precisamos salvar as planilhas habilitadas para macros. Assim,



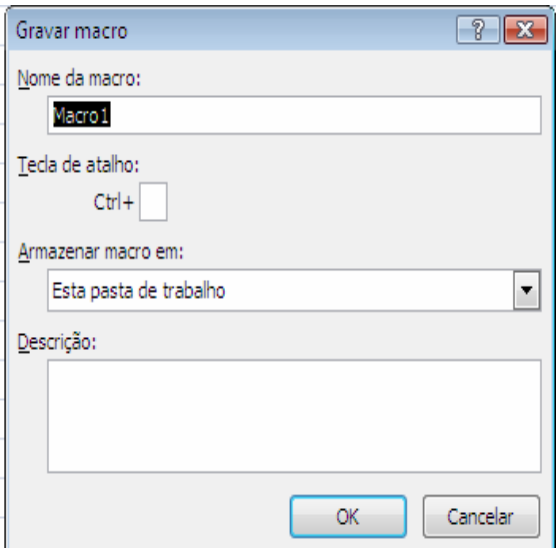
A seguir lhe mostro como **gravar** uma macro, e como **executá-la**:

- Abra o Excel
- Translade-se à célula A1 e escreva o seu nome. Por exemplo, Bertolo e pressione Enter
- Regresse à célula A1, porque quando destes Enter sob a célula A1, o ponteiro muda para outra célula.
- Dê um clique na Guia Desenvolvedor, a sua tela ficará assim:



Ativar-se-ão as ferramentas do *Visual Basic*.

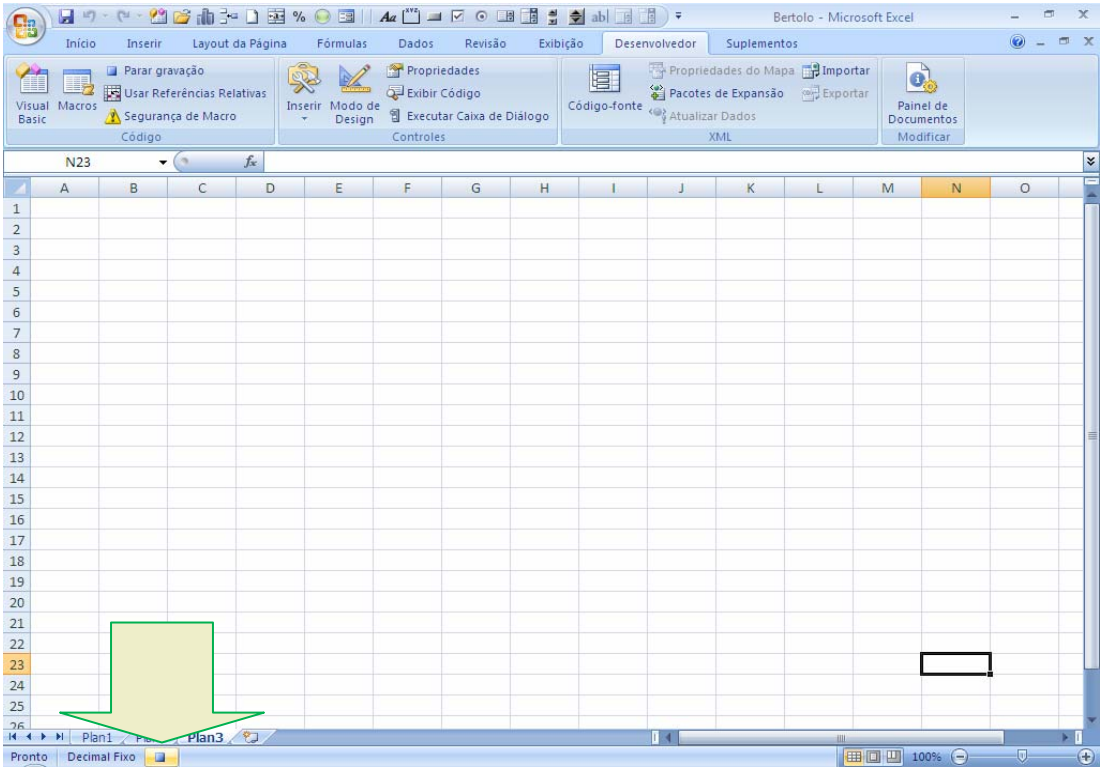
Agora dê um clique no botão Gravar Macro. O Windows ativa a caixa de diálogo Gravar macro, a qual lhe permitirá dar a ela um nome e qual tecla de atalho será usada para executá-la. A tecla de atalho se refere com qual letra se vai ativar a macro, obviamente se ativará com a tecla Controle (Ctrl) e uma letra que você queira, de preferência em minúscula, porque se usar as maiúsculas a macro deverá ser ativada pressionando a tecla Shift + Controle + a letra que você indicou.



- Onde se diz **Nome da macro** já aparece o nome que a macro levará, neste caso Macro1. Se desejares trocar este nome, escreva um novo, mas eu o recomendo que deixes assim.
- Na opção **Tecla de atalho** que aparece e que se ativará com a tecla **Controle (CTRL)** + a **letra que você indicar**, dê um clique no quadradinho e ponha uma letra. Para o propósito deste exercício entre com A (maiúsculo) nesta caixa de texto. Mais tarde você será capaz de rodar a macro apenas mantendo as teclas CTRL e um SHIFT e clicando em A[2].
- Em "**Armazenar macro em**:" está uma lista de todas as pasta que estão abertas, selecione "Esta pasta de trabalho" que é a pasta na qual você está realmente trabalhando.

  Observe que existe uma opção *drop down* nesta caixa de diálogo. Se você selecionar Pasta de trabalho pessoal de macros, as instruções serão gravadas numa pasta chamada "Pessoal.xls". Se esta pasta não existir, o Microsoft Excel a criará. Se o Microsoft Excel a criar, a pasta inicialmente será uma pasta oculta. Você terá que selecionar Janela, Reexibir para mostrá-la. Quando você salvar Pessoal.xls, ela será salva no seu diretório XLSTART. Deste modo, quando você abrir Microsoft Excel, este arquivo é automaticamente aberto e as macros que você criou e armazenou nesta pasta estão imediatamente disponíveis. Se você estiver executando o Excel numa rede, você deve precisar especificar um diretório de início alternativo no painel Ferramentas, Opções, Geral e salvar o arquivo para este diretório na primeira vez. Usualmente a opção Esta pasta de trabalho é melhor.
- Dê um clique no botão OK. O Windows começará a gravar todos os passos na chamada daquilo que você especificou e a anexou na pasta que você está trabalhando, e aparecerá na barra de Status o ícone quadradinho Azul ■ e chamar-se-á Parar gravação

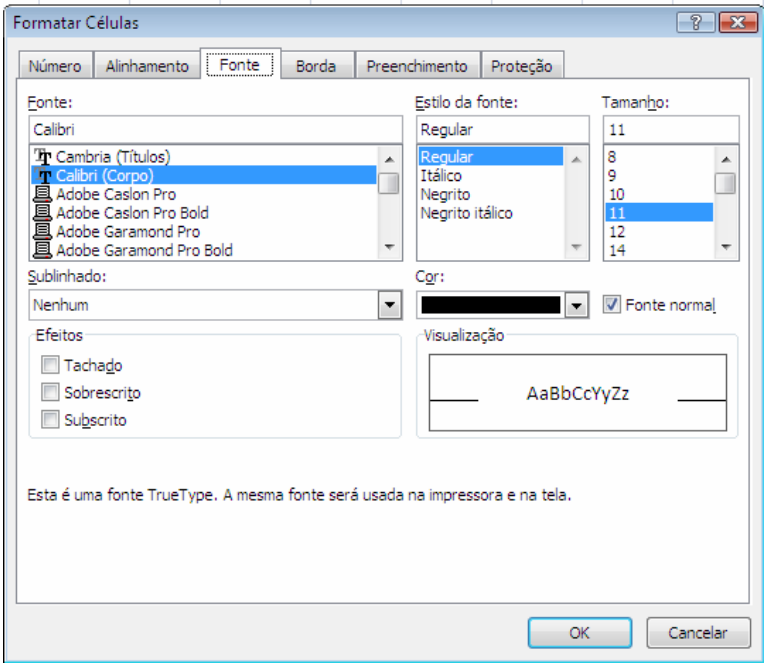
[2] **NOTA:** Sempre selecione a letra maiúscula de modo que você não desative importantes funções que estão já anexadas a alguma letra minúscula pelo Excel.



Quando você gravar a planilha que você está trabalhando, a macro será salva ao mesmo tempo. Também significa que a Macro1 desaparecerá do seu computador quando você deletar a planilha.

De agora em diante você não pode errar, pois será tudo gravado (inclusive os erros).

#1. Clique com o botão direito do mouse na célula e, a seguir, clique no item de menu Formatar Células..., e aparecerá o painel abaixo:



Na guia Fonte troque o **tipo** de letra na caixa Fonte:

#2. Troque o **tamanho** da letra na caixa Tamanho:

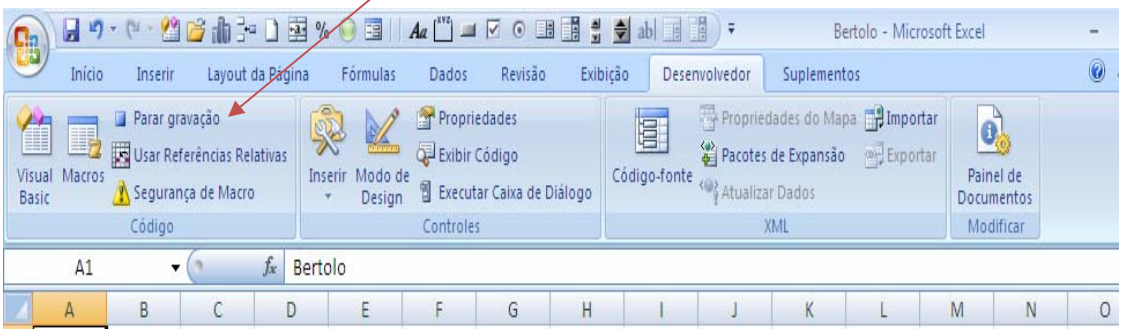
#3. Selecione **Negrito** na caixa Estilo da fonte

#4. Troque a **cor** da letra na caixa Cor:.

Lembre-se que todos estes passos estão sendo armazenados na macro que estamos gravando e, também lembre-se que estes passos estão se efetuando na célula A1**.**

#5. Pressione o botão Parar gravação na barra de Status:

ou então, no ícone Parar Gravação na Guia Desenvolvedor**.**



Assim o Excel guarda os passos na **Macro1** que se ativará pressionado a tecla Ctrl + a **.**

I. Escreva outro nome na célula C1 e pressione Enter**,** depois volte à célula C1**.**

II. Pressione a tecla Ctrl + Shift + A**.** O Windows efetuará todos os passos gravados sobre a célula C1, isto quer dizer que o nome que esta em C1 terá as mesmas caractereísticas do que está em A1**.** Tipo de letra, tamanho, negrito e a cor que indicastes ao gravar a macro[3].

III. Vá para Plan2 e rode a macro novamente. Você pode ver que a macro que você criou rodará em qualquer folha de planilha enquanto a folha de planilha em que ela foi criada estiver aberta

IV. Salve sua pasta como "VBATeste1" para as próximas lições e feche-a. Tente rodar a macro novamente. Ela não estará mais lá, assim você não pode rodá-la se a tecla de atalho estiver livre.

Se você fez isto, você gravou sua primeira macro com o gravador de macro do Excel e a usou, congratulações. Agora recomendo que domines estes passos antes de passares à lição seguinte. Trata-se de criar macros que armazenem passos como esses, grave os passos que te indiquei, e parar a gravação depois de que indicastes estes passos. Repita este exercício tantas vezes quanto desejares necessário para apreendê-lo bem.

[3] Cada vez que presisonares Ctrl + Shift + a, o Excel executará a macro e efetuará os passos na célula que onde você se encontrar. Podes gravar todas as macros que desejares.

Prática I

1. Vamos agora gravar outra macro com o gravador de macro Excel:
   Selecione célula A1 e entre com 34
   Selecione célula A2 e entre com 55
   Selecione célula A3 e entre com a fórmula =A1 + A2
   Selecione célula A2 e mude a cor da fonte para vermelha
   Selecione célula A1 e mude a cor de background (padrão) para azul
   Selecione célula A3 e mude o tamanho da fonte para 24
   Clique no ícone no meio da tela para parar o gravador ou ir para a barra de menu "Ferramentas/Macro/Parar gravação".
   Teste a sua macro na Plan2, na Plan3,

2. Gerar as Macros seguintes:

   Grave uma **Macro** que se ative com **Ctrl+Shift+b** e que esta macro lhe permita **abrir** um arquivo
   Grave uma **Macro** que se ative com **Ctrl+Shift+c** e que esta macro lhe permita **inserir** um WordArt
3, Ver Livros mais exercícios.....

Muitos pensarão que isto não é nada mais do que veremos mais adiante quando mesclarmos os códigos que fora gerado pelo **Excel** com os de **Visual Basic**. Isto será pura **Mentira**.

GRAVAÇÃO ABSOLUTA E RELATIVA

Quando você estiver gravando uma macro, você tem a opção gravar usando endereços de célula **absolutos** ou endereços de célula **relativos**. Além disso, você pode mudar esta opção quando você estiver gravando uma macro. Qual opção escolher é muito importante em como opera a macro que você gravou. Também, declarações diferentes do Visual Basic são gravadas.

Esta opção está disponível no segundo botão na barra de ferramentas que aparece quando você começar a gravação de uma macro:

Se o botão estiver apertado, a gravação está no modo **relativo**. Se não apertado, então referências de célula **absoluta** estão sendo usadas.. O *execício* que segue ilustra a diferença entre gravação relativa e absoluta.

Crie uma folha de planilha como a seguinte de modo que você possa fazer este exercício.

	A	B	C	D	E
1					
2					
3		Números		Fórmulas	
4					
5		1234		1234	
6		5678			
7					

Coloque o ponteiro de célula na célula B3, a qual contém a palavra "**Números**".

Você não pode pre-definir a opção de gravação relativa/absoluta. Principalmente, se você configurá-la após começar a gravação. Se o botão **relativo** estiver apertado, então a gravação é relativa, caso contrário, é absoluta.

Novamente selecione na Barrra de Status o ícone Gravar Macro

Nomeie a macro como "*GravaçãoAbsoluta*" (sem as aspas duplas). Mude a caixa de descrição para apresentar "Macro modelo para demonstrar gravação absoluta". Selecione o botão OK. Tenha certeza de que o botão gravação relativa **não** está apertado.

Até este ponto, suas ações estavam sendo gravadas. Selecione (usando seu mouse) a célula D3, que contém a palavra "Fórmulas".

Pare a gravação clicando no botão parar ■ na Barra de Status.

O que você tem feito foi gravar uma macro usando gravação de endereços de célula absoluta. Eu explicarei mais sobre esta macro mais tarde. O que eu preciso que você faça agora é gravar uma outra macro. Esta uma demonstrará a gravação relativa.

Novamente selecione a célula B3, a célula com a palavra "**Números**" de modo que você esteja nesta célula quando você fizer o próximo passo.

Selecione Gravar Macro na Barra de Status.

Dê à macro o nome de "*GravaçãoRelativa*", e entrar com uma breve descrição na caixa de descrição. Selecionar OK. Clicar no botão gravação relativa para torná-lo pressionado, ligando assim a gravação relativa.

Clique na célula D3 (a célula com a palavra "**Fórmulas**") com o mouse.

Selecione Parar Gravação (o botão ■) na Barra de Status

Até este ponto você tem duas macros, uma chamada "*GravaçãoAbsoluta*", e uma chamada "*GravaçãoRelativa*". Faça o seguinte para testar a macro GravaçãoAbsoluta:

- Coloque o ponteiro de célula na Célula A1
- Selecione Ferramentas, Macro, e execute a macro GravaçãoAbsoluta.
- Repita o passo acima, mas primeiro coloque o ponteiro de célula em diferentes células, Por exemplo, A5, C3, D7, etc..

O que acontece cada vez que você executa a macro é que o cursor vai para a célula D5. Esta é a célula que você selecionou quando você gravou a macro. Assim, gravação de endereço de célula absoluta significa ir para as células que estão selecionadas, não importa onde você esteja na planilha.

Faça o seguinte para tentar a macro GravaçãoRelativa:

- Coloque o ponteiro de célula na Célula A1
- Selecione Ferramentas, Macro, e dê um duplo clique na macro GravaçãoRelativa.
- Repita o passo acima, mas com o ponteiro de célula nas células diferentes, Por exemplo, A5, C3, D7, etc..

O que acontece cada vez que você rodar a macro e que o ponteiro de célula move duas células para a direita. Isto é o que você fez quando você gravou a macro usando gravação relativa. Assim, gravação relativa de célula significa mover o ponteiro da célula relativa para a célula ativa.

Não importa qual meio de gravação você escolheu, relativo ou absoluto, depende da seleção de célula em atividade que você quer gravar. Se a macro que você está gravando não envolver qualquer seleção de célula em atividade, então não faz diferença qual configuração está no lugar. Por exemplo, uma macro que apenas muda a estrutura da página ou que imprime um arquivo não envolve seleções de célula.

O que segue é o que o código de macro das duas macros acima se parece:

```
Sub GravacaoAbsoluta()
    Range("D3").Select
End Sub

Sub GravacaoRelativa()
    ActiveCell.Offset(0, 2).Range("A1").Select
End Sub
```

Quando macros são gravadas usando gravação absoluta, o Microsoft Excel grava as referências de célula na macro que é criada. Se a macro for gravada usando gravação relativa, então é gravada a distância da célula destino em termos do número de linhas e colunas da célula ativa. Como as duas macros acima ilustram, declarações diferentes também são gravadas, por exemplo, os métodos Select e Offset.

As duas macros que gravamos também dão vários comandos adicionais para nós que provarão serem de valor: Range, Select, e Offset. Este é um dos benefícios principais do gravador de macros - o fato que eles mostram a você declarações de macros que provarão úteis para você em suas macros. A discussão deles será deixada por enquanto.

ARMADILHA DA GRAVAÇÃO ABSOLUTA

Há uma armadilha com o uso de gravação **absoluta** que você deverá estar consciente. Eu ilustrarei isto com o seguinte exercício que insere *Jan*, *Fev*, e *Mar* como títulos de colunas.

- Vá para uma planilha em branco numa pasta
- Selecione a célula B2
- Verifique para ver se a gravação é absoluta ou relativa. Faça isto examinando o botão. Se ele não estiver apertado, então a gravação está configurada para Absoluta. Se ele estiver apertado, então a gravação está configurada para Relativa. Clique o botão se ele não estiver apertado
- Selecione Ferramentas, Gravar Macro, Gravar Nova Macro.
- Defina o nome da macro para atravesMeses
- Selecionar OK para começar a gravação. Fique certo de que o botão gravação relativa não esteja pressionado.
- Digite "Jan", selecione C2, digite "Fev", selecione D2, e digite "Mar"
- Selecione B2, a célula contendo "Jan"
- Parar gravação.

Até este ponto você tem uma pequena macro que entra com Jan, Fev, e Mar nas células B2, C2, e D2 para você. Vamos conferir a macro.

- Apagar as B2, C2 e D2
- Coloque o ponteiro de célula em B2
- Execute a macro.

Como você esperava, a macro escreve Jan em B2, Fev em B3, e Mar em B4. Apagae a planilha e repeta o que está acim, mas coloca o ponteiro de célula em B5 antes de executar a macro. O que segue é o resultado que você obtém:

Note que Jan não está na célula B2, mas em B5. Se você repetir o proceso mas começando numa célula diferente, por exemplo A1, a entrada de Jan termina naquela célula. Por que isto aconteceu com a gravação absoluta ligada? Esperar-se-ía a macro colocar Jan na célula B2 como você digitou-a pois a gravação absoluta estava ligada.

O problema acima aconteceu porque a primeira ação que você fez depois de ligar o gravador foi digitar a palavra "Jan". Você não selecionou primeiro a célula B2. O que segue é o que a macro que você gravou se parece:

```
Sub atravesMeses()
    ActiveCell.FormulaR1C1 = "Jan"
    Range("C2").Select
    ActiveCell.FormulaR1C1 = "Fev"
    Range("D2").Select
    ActiveCell.FormulaR1C1 = "Mar"
    Range("B2").Select
End Sub
```

O R1C1 é um resquício da versão original do Excel. Nas versões anteriores do Excel, macros trabalhavam se referindo às linhas e colunas por seus números, por exemplo, Coluna A era C1, coluna B era C2, e assim por diante. Se uma propriedade, neste caso a propriedade fórmula tem R1C1 anexada nela, significa que você deve fornecer os endereços de célula numa fórmula usando a notação R1C1. Se você está fornecendo apenas valores, então não faz diferença. No nosso exemplo, estamos fornecendo valores.

Para gravação absoluta funcionar da maneira que você quer, primeiro você deve selecionar uma célula, depois que você ligou o gravador, e daí digitar a entrada. Se a célula já estiver destacada, clicar nela novamente com o mouse. Isto diz ao gravador para selecionar a célula. Se você clicou na célula B2 mesmo que ela estivesse selecionada, sua macro ficará parecida com esta, e entrará com Jan em B2, etc.. exatamente como você pretendia:

```
Sub atravesMeses()
    Range("B2").Select
    ActiveCell.FormulaR1C1 = "Jan"
    Range("C2").Select
    ActiveCell.FormulaR1C1 = "Fev"
    Range("D2").Select
    ActiveCell.FormulaR1C1 = "Mar"
    Range("B2").Select
End Sub
```

Por favor, note que se você estiver usando gravação relativa e já tiver o ponteiro de célula onde você quer começar suas entradas, você não precisa selecionar a célula primeiro.

Lição 4: O Ambiente Amigável do Visual Basic Editor do Excel

Bem, agora depois de praticar os diferentes exemplos ou seja, as **Macros** desenvolvidas com o Gravador de Macros (GM), vamos, nas próximas lições, observar os códigos que foram gerados para elas. Recomendo a você que salve o **Excel** e volte a entrar, para que trabalhe limpo sem nenhuma macro, e começando a macro1 de novo.

As observações, testes e modificações dos códigos dos procedimentos VBA são feitas numa ambiente de programação conhecido como *Visual Basic Editor* (VBE) do Excel. É um ambiente muito amigável ao usuário. Nele também podemos escrever os nossos procedimentos (macros e funções) a partir do zero, isto é, sem a utilização do gravador de macros.

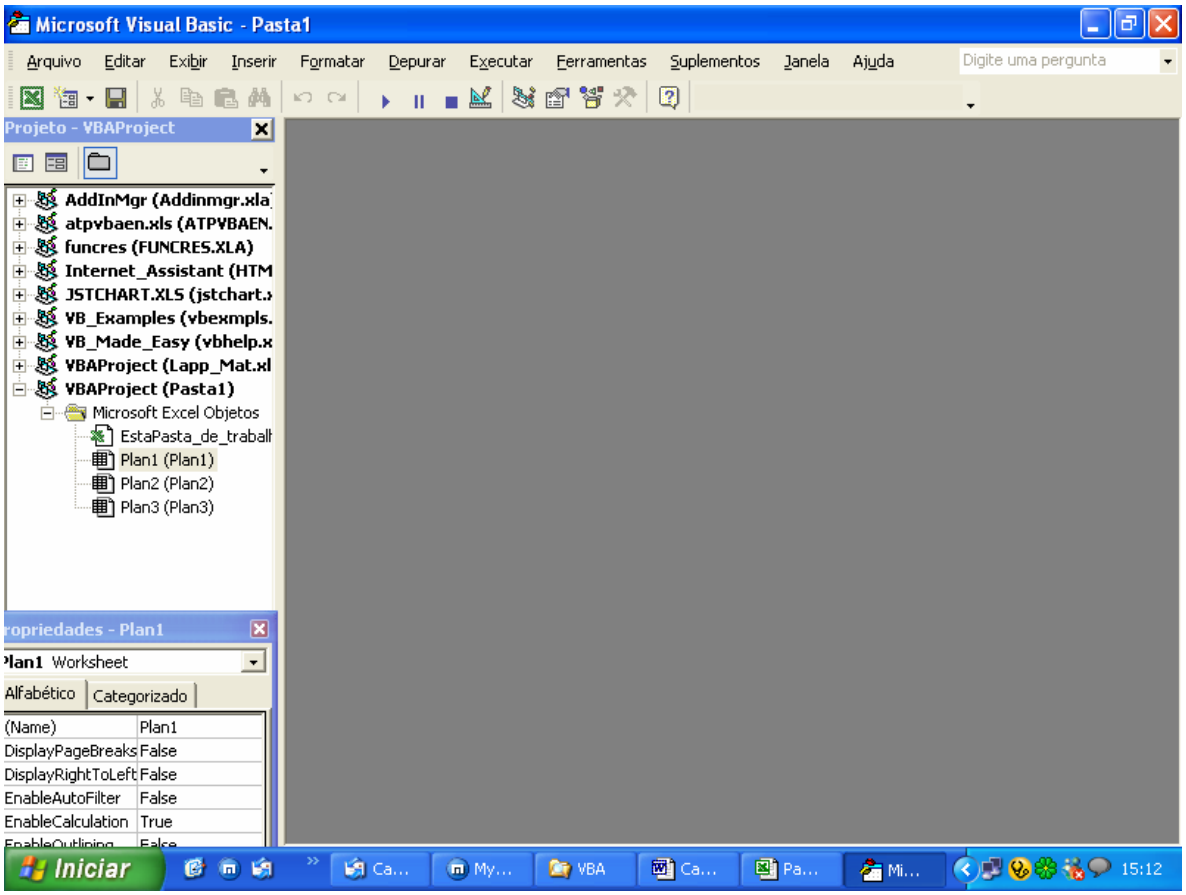
Os procedimentos VBA desenvolvidos no VBE, tornam-se parte da pasta na qual eles foram desenvolvidos, e quando a pasta é salva, os componentes do projeto (macros, módulos, *userforms*) são salvos ao mesmo tempo.

O Visual Basic Editor do Excel (VBE)

O VBE está integrado ao Excel e você pode abrí-lo de três maneiras:

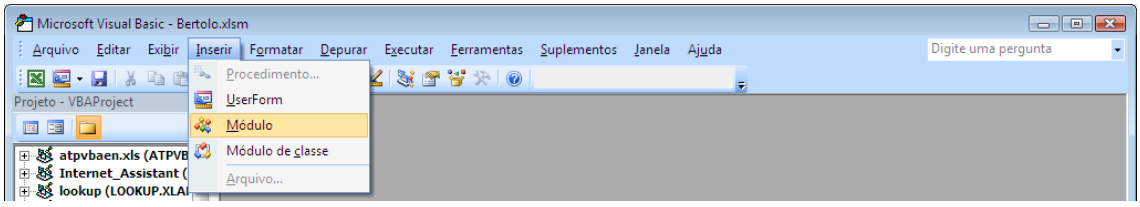
- Selecione a guia Desenvolvedor, Visual Basic Editor, ou
- Pressione ALT-F11, ou
- Clique no ícone  na Barra de Ferramentas de Acesso Rápido do Excel.

O que segue é com o que o Visual Basic Editor se apresenta. Seu editor deve parecer ligeiramente diferente por causa das mudanças que você fez ou porque diferentes versões do Excel ter características diferentes:

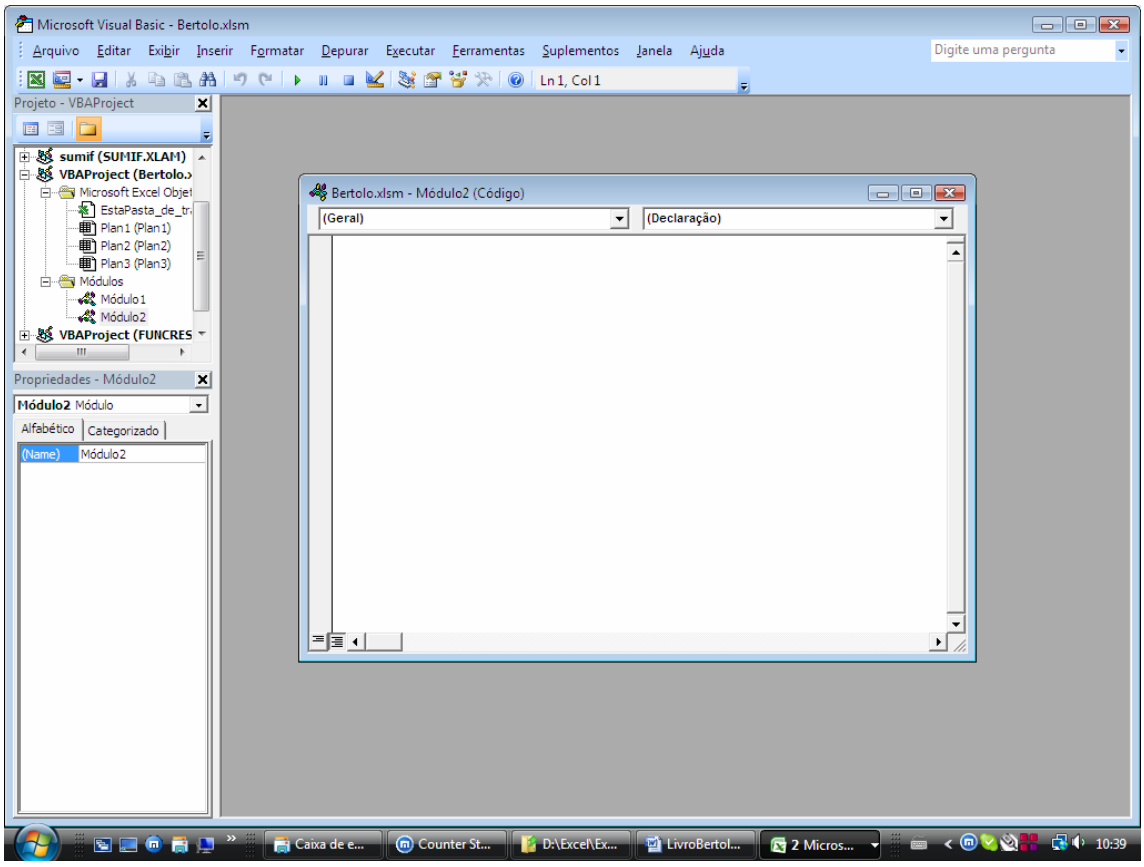


Do VBE você pode ir para o Excel clicando no botão Excel no topo/esquerdo de sua tela  . Assim usando os dois botões  você pode navegar no VBE para o Excel e, vice-versa.

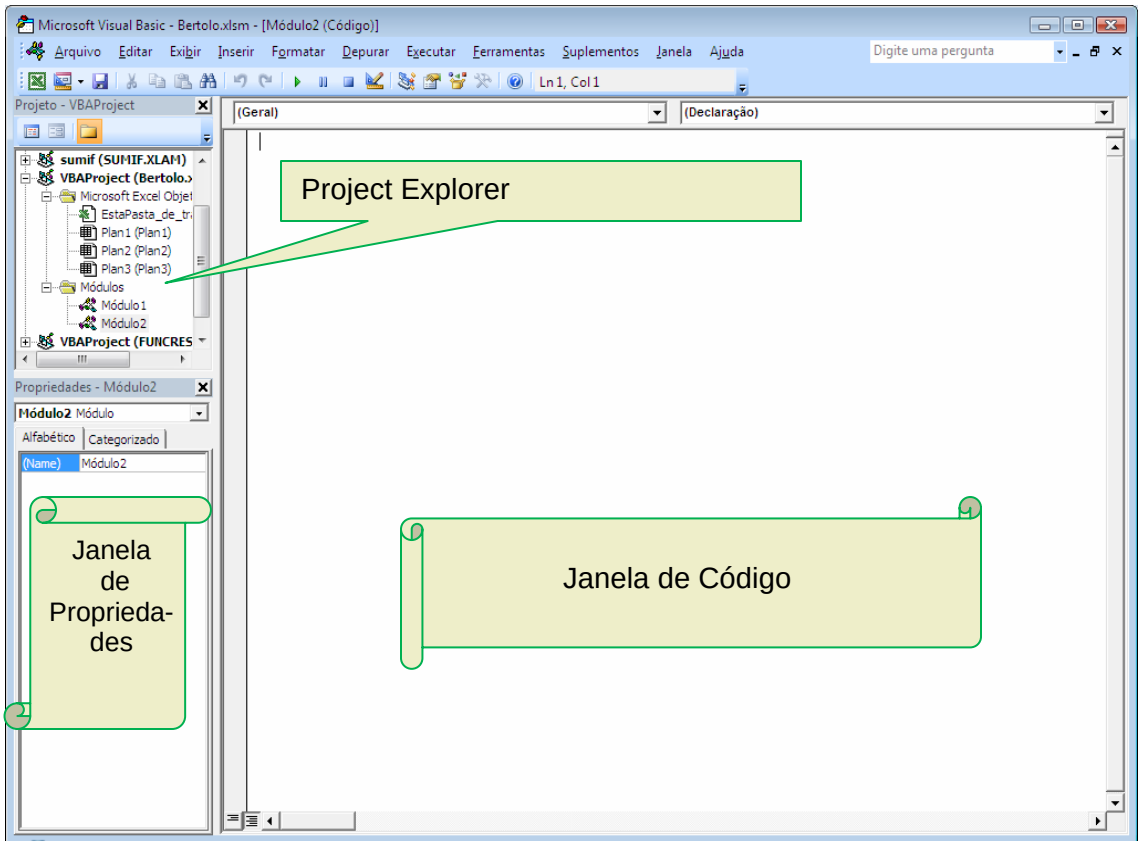
Para inserir um módulo usando os menus do Visual Basic, selecione (no editor do VB) Inserir, Módulo.



Não inserir um módulo de classe. *Módulos de classe* são usados para programação especializada de macros tais como indicar macros para serem rodadas quando eventos tais como substituição de folhas ou pastas acontecerem. Um exemplo de usar módulos de classe é encontrado numa lição posterior. O que segue ilustra um módulo que foi adicionado:



O que está acima pode também parecer com o que segue se o *módulo de código* (a janela do centro) for maximizada:



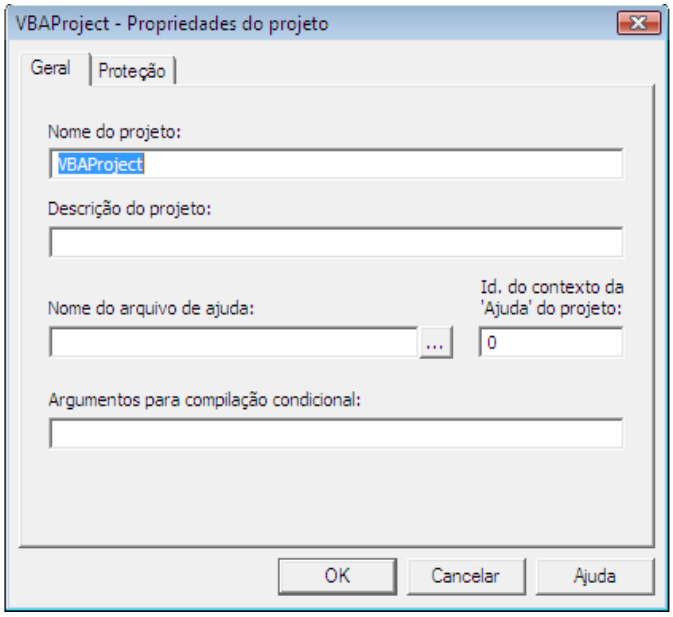
A janela esquerda superior é chamada de janela Project Explorer. Ela mostra a diferentes folhas de cada pasta aberta. A janela esquerda inferior é a janela Propriedades. Ela mostra as propriedades do objeto selecionado ou ativo. A janela do lado direito é chamada de janela de código.

Para mudar o nome de um módulo, clique no nome do módulo na janela Project Explorer. Daí então mostrar a janela Propriedades (pressione F4, ou selecione Exibir, Janela Propriedades, ou clique no botão que mostra a Janela de Propriedades na barra de ferramentas padrão). Na janela propriedades você pode mudar o nome do módulo. Você pode usar somente letras e números para nomear os seus módulos. Não deve ser usado espaço ou caracterees especiais.

NOMES DE PROJETOS

Às folhas e módulos associados com uma pasta é dado o nome **projeto**. O nome *default* é "VBAProject". Se você tiver múltiplas pastas abertas, então você poderia dar a cada uma delas seu próprio nome de projeto. Isto lhe permite usar o Pesquisador de Objeto (discutido mais tarde) para ver somente as macros de uma dada pasta. E, como o Pesquisador de Objeto remove gradualmente nomes de projeto, nomes únicos são úteis.

Para mudar o nome de projeto *default* de "VBAProject", selecione Ferramentas, Propriedades de VBAProject (Se você já tiver mudado o nome de projeto, você verá "xxx" no menu Propriedades, onde "xxx" é o nome do projeto para o qual você mudou..) O diálogo que segue aparece:



Você pode então mudar este nome para um novo nome – contanto que o novo nome não tenha espaços. O campo Descrição projeto do pode ser qualquer texto que você queira, e aparece no Pesquisador de Objeto quando o projeto é selecionado. As outras opções neste diálogo são discutidas posteriormente neste material. Se você estiver curioso agora, você pode selecionar o botão Ajuda para uma explicação.

ONDE COLOCAR AS MACROS

Você pode colocar macros em qualquer pasta que você queira. Muitos usuários guardam suas macros numa pasta chamada PERSONAL.XLS. Esta pasta é normalmente encontrada no diretório de inicialização do Microsoft Excel. Este diretório é chamado XLSTART e é quase sempre um subdiretório de seu diretório Microsoft Excel. O que segue são os caminhos mais comuns para este diretório:

- C:\PROGRAM FILES\MICROSOFT OFFICE\OFFICE\XLSTART
- C:\EXCEL\XLSTART
- C:\MSOFFICE\EXCEL\XLSTART

Se você estiver rodando o Excel numa rede, você deve precisar especificar um diretório de inicialização alternativo em Ferramentas, Opções, painel Geral

Para usuários da Apple e MacIntosh, a pasta de inicialização é quase sempre chamada "EXCEL STARTUP FOLDER" e é normalmente uma pasta na pasta SYSTEM:PREFERENCES.

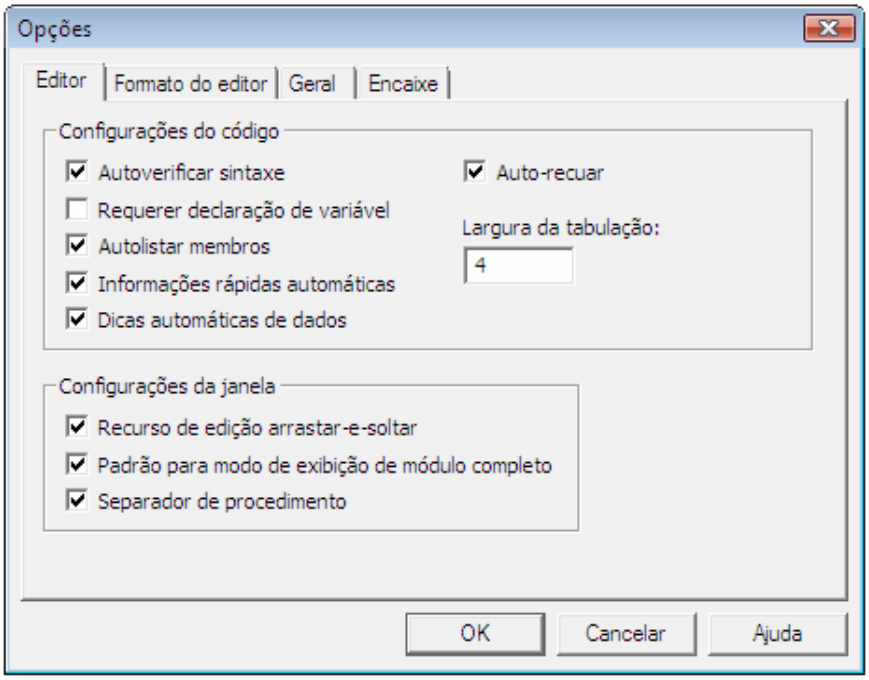
A vantagem de colocar macros na PERSONAL.XLS no seu diretório de inicialização é que este arquivo automaticamente será aberto pelo Microsoft Excel quando você iniciar o Microsoft Excel. Deste modo, suas macros estão imediatamente disponíveis para o seu uso. E, quando você gravar macros, você pode especificar que a gravação seja armazenada neste arquivo, como veremos na lição seguinte.

O MENU DO VISUAL BASIC EDITOR

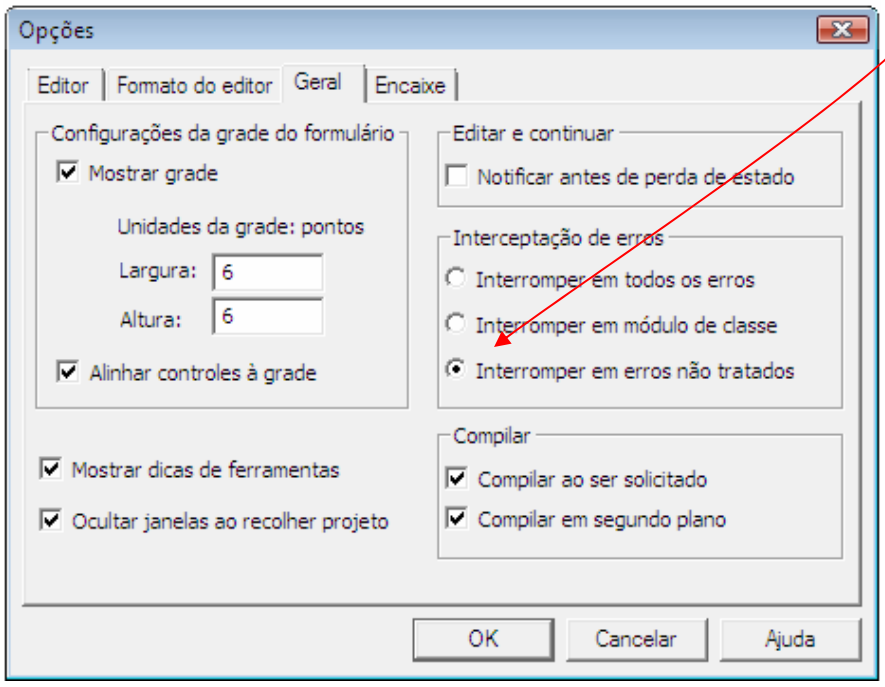
O menu do Visual Basic Editor que estava no Excel 5 e 7 foi removido a partir do Excel 97. A personalização de menu deve agora ser feita com o código.

CONFIGURANDO SEUS MÓDULOS

No Visual Basic Editor, selecionando Ferramentas, Opções, mostra o seguinte diálogo:



O que está acima também mostra o *setup* (configuração) necessário para os exercícios da lição seguinte: Marque as caixas de verificação Auto-recuar , Autoverificar sintaxe e Requerer declaração de variável. A seguir, selecione a guia Geral e certifique-se que a caixa de verificação, Interromper em erros não tratados, está assinalada.



Se o seu *setup* nestas caixas for diferente, você poderá mudá-los para este *setup* para os exercícios iniciais da lição seguinte. O que segue explica em detalhes estas opções:

Auto-recuar: Se o texto que você está digitando é recuado, quando você dar um enter para ir à próxima linha, a nova linha automaticamente será recuada. Recuar o seu código ajuda juntar as declarações relacionadas de modo que elas fiquem visivelmente agrupadas.

Autoverificar sintaxe: Esta opção mostrará uma caixa de mensagem sempre que um erro de sintaxe for detectado na declaração a ser editada e lhe dará uma breve mensagem explicando o problema da sintaxe. Até este ponto, você decidirá qua a coisa certa a fazer é desligar estas mensagens. Não existe problema em se fazer assim. Entretanto, inicialmente deixe esta opção ligada, pois, as mensagens como você verá serão parte da lição seguinte.

Interromper em erros não tratados: Esta opção deverá ser selecionada, ela fará macro parar de rodar sempre que ocorrer um erro não manipulável. Fique certo de que esta opção esteja selecionada. Rotinas de manipulação de erros serão cobertas em detalhes no capítulo 10, Manipulando Erros. A seleção Interromper em erros não tratados encontra-se na alça Geral.

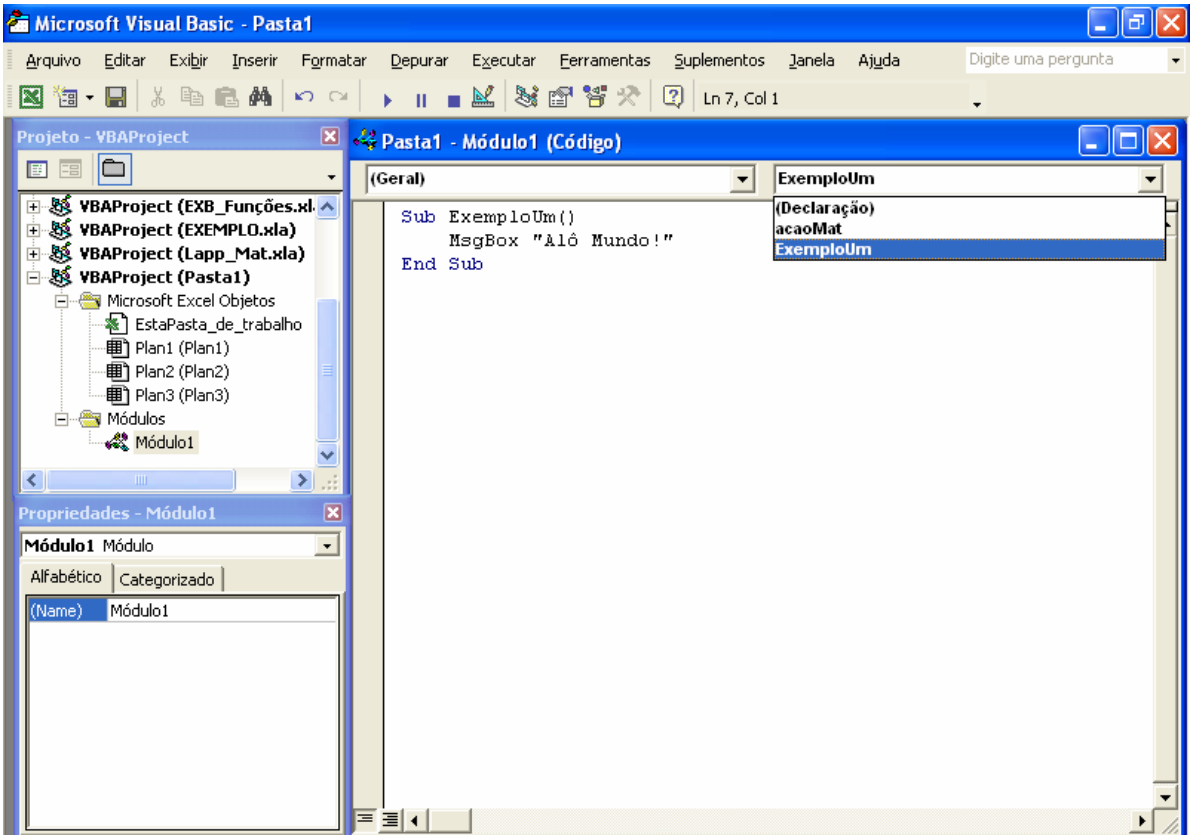
Requerer declaração de variável: Você deverá ligar esta opção. Quando esta opção é ligada, a declaração Option Explicit do Visual Basic é automaticamente colocada no topo de cada nova folha de módulo. Esta declaração simplesmente diz ao Visual Basic que cada variável deve ser declarada antes de funcionar a macro. Para declarar uma variável é muito simples e será coberto mais tarde.

Declarar variáveis tem várias vantagens. Ela ajuda a evitar erros acidentais e resultar em confusão. E ela mantém a aparência da capitalização que você der às variáveis. Quaisquer declarações que você entrar num módulo poderá ser após a declaração Option Explicit.

Os ajustes nas caixas de diálogo acima fazem o seguinte:

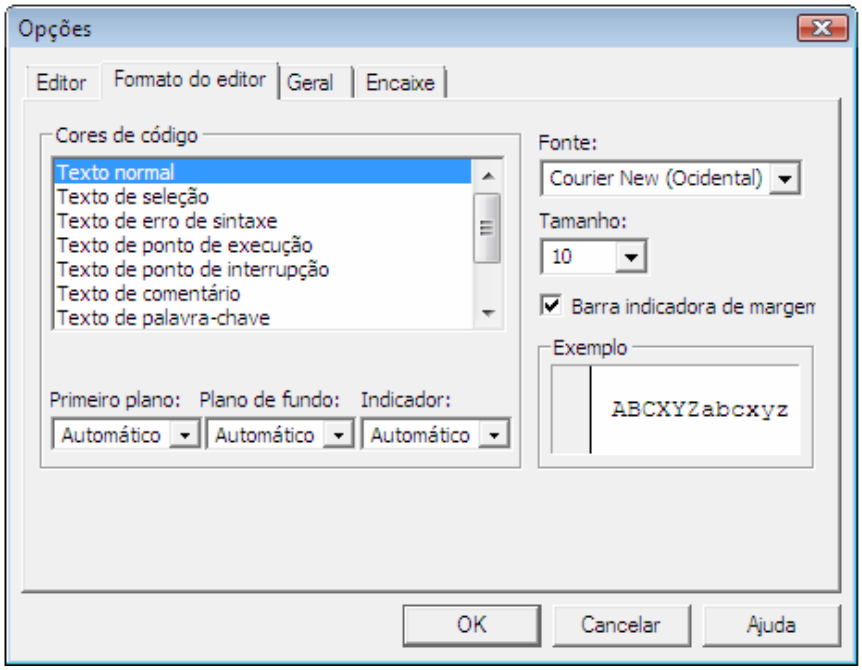
Recurso de edição arrastar-e-soltar – permite-lhe destacar texto e arrastá-lo a uma nova localização no módulo. Você pode também arrastar texto aos painéis imediatos ou de visão (watch) (discutidos mais tarde)

Padrão para modo de exibição de módulo completo – se esta opção está ligada, então todas macros são visíveis num módulo em qualquer momento. Se ela estiver desligada, então você pode ver somente uma macro de cada vez. Se você desligar esta opção, você pode ir a outras macros, clicando na caixa abaixo localizada na janela direita superior de um módulo. Isto mostrará as diferentes macros, de modo que você possa saltá-las:



Separador de Procedimentos – Este desenha uma linha entre os diferentes módulos como ilustrado acima. É muito útil para ajudá-lo separar visualmente uma macro da outra.

Clicando na alça Formato do editor (do diálogo Ferramentas, Opções) mostrará o seguinte painel:



As opções deste painel controlam a aparência do texto nos seus módulos. A fonte *default* é Courier New. Entretanto, a fonte *Fixedsys* dá uma aparência mais ousada e é para muitos, mais fácil de ler que a Courier New. É altamente recomendado que você mude sua fonte para *Fixed sys*. A tabela que segue mostra as cores de código que eu recomendo:

Tipo de Texto	Frente (texto)	Fundo
Texto de seleção	Automático*	Automático*
Texto de erro de sintaxe	Vermelho	Automático (branco)
Texto de ponto de execução	Automático (preto)	Automático (branco)
Texto de ponto de interrupção	Automático (preto)	Amarelo
Texto de comentário	Automático (preto)	Azul celeste
Texto de palavra-chave	Violeta Automático	(branco)
Texto de identificador	Automático (preto)	Automático (branco)
Texto Normal	Azul escuro	Automático (branco)

* Automático em Texto de seleção significa aparência de vídeo reverso.

Texto de seleção é o texto que você destacou com o cursor de modo que você possa recortá-lo, copiá-lo ou deletá-lo. Deixando o ajuste em automático em Primeiro plano e no Plano de fundo, permite o Microsoft Excel, apresentar o texto selecionado texto na aparência de vídeo reverso, o qual é o que a maioria dos processadores de textos fazem quando você seleciona texto.

Texto de erro de sintaxe é o texto que Visual Basic não entende porque ele não está escrito corretamente. Por exemplo, um parêntese pode ter sido deixado de fora. Desde que erros de sintaxe são ruins, colorir o texto em vermelho é o melhor modo de destacar o texto que deve ser corrigido. Este é o ajuste *default*.

Texto de ponto de execução é a próxima declaração que será executada se você estiver caminhando pela macro para depurá-la. Existe um capítulo mais tarde neste material que cobre a depuração de suas macros. O ajuste *default* é maravilhoso quando o Visual Basic também desenha uma caixa ao redor da próxima declaração a ser executada.

Texto de ponto de interrupção é uma entrada que você pode fazer e que pausará sua macro e lhe permitirá depurá-la interativamente. Isto lhe permite caminhar pelos comandos, começando no texto que foi destacado e declarado um ponto de interrupção. Configurar o seu Primeiro plano para automático (preto) e seu Plano de fundo para amarelo serve para enfatizar os pontos de interrupção, mas torna-os bem mais legíveis que os ajustes *default*. (Não se aborreça com os pontos de interrupção. Eles serão tratados em detalhes posteriormente. Apenas lembre-se que eles são úteis).

Texto de comentário é qualquer texto que comece com aspas simples. Ele serve para documentar suas macros. Tenho percebido que usando um Plano de fundo verde coloca os comentários fora das declarações reais do Visual Basic que são executadas.

Texto de palavra-chave são palavras que o Visual Basic identifica como parte da linguagem de programação. Alguns exemplos de tais palavras são If, Then, GoTo, And, Is, Integer, Next, e Error. Ajustando as palavras chave para um texto de cor azul escuro serve para distinguir tais palavras melhor do que o ajuste *default*.

Texto de identificador são os nomes de variáveis que você criou e muitos dos comandos que você usa. Por exemplo MsgBox é um comando do Visual Basic que mostra uma mensagem na tela. Ele é também considerado texto identificador, muito do que você digitar será assim. O ajuste automático mostra MsgBox como texto negro.

Texto normal são os números, texto incluídos em aspas, e símbolos como =, +, *, -,), e (. Ajustando tais textos para azul escuro serve para distinguí-los de outros textos.

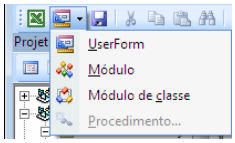
A BARRA DE FERRAMENTAS DO VISUAL BASIC

Existem várias barras de ferramentas no *Visual Basic Editor*. A barra de ferramentas normalmente no topo do Visual Basic editor é chamada de barra de ferramentas Padrão. Ela se parece com o seguinte:

Se você mantiver o ponteiro do mouse sobre um botão, você verá uma dica de ferramenta que descreve a função deste botão.

Os primeiros três botões,  , fazem o seguinte:

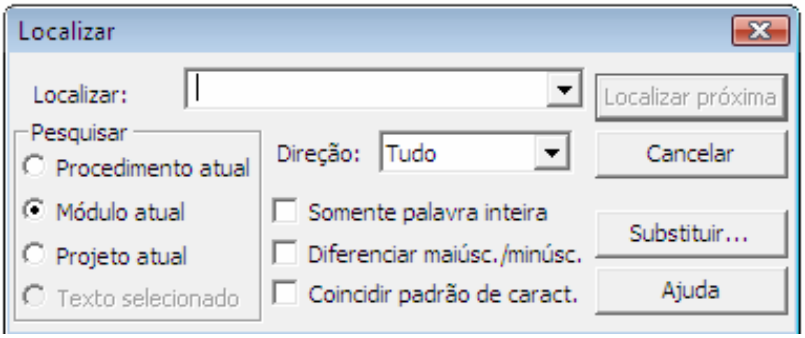
- Retornar ao Excel
- Permite-lhe inserir um novo **módulo**, procedimento (macro), ou a *userform* (diálogo). O pequeno triângulo à direita deste botão mostrar uma lista *drop down* destas escolhas.



- Salvar o Arquivo

Os próximos três botões,  lhe permitem recortar, copiar, e colar textos.

O botão binóculos,  . Clicar nos binóculos para mostrar uma caixa de diálogo Localizar:



Os próximos dois botões,  , ou o *undo* e o *redo* desfaz e refaz a última ação de edição.

Os próximos três botões nesta barra de ferramentas,  lhe permitem fazer o seguinte:

- Rodar o procedimento ou mostrar o userform que você está editando
- Parar ou interromper uma macro que esteja rodando. Parar uma macro que está rodando

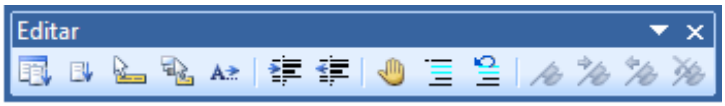
O botão  é o botão modo de **design** (criação). Normalmente você fará muito pouco uso deste botão, e pode ser ignorado.

Os próximos quatro botões na barra de ferramentas padrão,  lhe permitem fazer o seguinte:

- Mostrar o Project Explorer se ele estiver oculto. O Project Explorer é a janela no VB editor que mostra uma lista das diferentes folhas de planilhas de um arquivo.
- Mostrar a Janela Propriedades sempre que um objeto for selecionado na janela *Project Explorer*
- Mostrar o Pesquisador de Objeto (discutido mais tarde)
- Se num userform (diálogo), o último botão mostra a caixa de ferramentas, que lhe permite inserir objetos no formulário.

O último botão na barra de ferramentas padrão mostra o assistente de ajuda.

O Visual Basic Editor tem outras três barras de ferramentas que você pode também usar:



Estas barras de ferramentas podem ser mostradas selecionando Exibir, Barra de ferramentas dentro do Visual Basic Editor. Uma curta descrição de cada botão é mostrada, se você mantiver o cursor sobre o botão. Os botões da barra de ferramentas Userform estão ativos somente se um deles estiver no Userform.

As características úteis da barra de ferramentas Editar são:

- Os botões recuar e não-recuar, que ajudam tornar seu código mais fácil de ler.

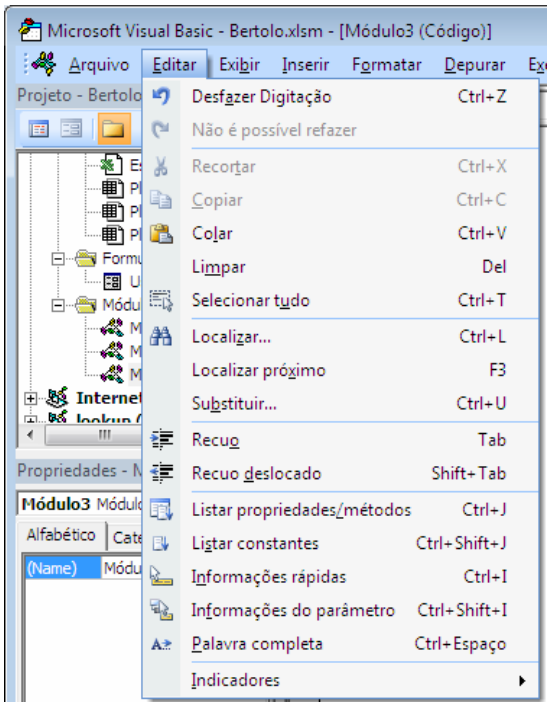
- Os botões bloco de comentário e bloco de não-comentário, os quais mudam as declarações do Visual Basic para comentários e voltam novamente.
- O botão indicador de bookmark (bandeiras), que realmente cria um indicador (*bookmark*) para ajudá-lo na navegação de seu código. Uma vez tendo criado um indicador, os outros botões lhe ajuda retornar para ela.

Os botões úteis na barra de ferramentas Depurar são:

- O botão executar, que roda uma macro sem pressionar quaisquer teclas ou retornar à sua pasta.
- O botão interromper, que pára a sua macro.
- Os botões step into e step over que lhe permitem executar uma única linha de cada vez.
- Os botões de inspeção, que lhe permitem examinar valores de variáveis e outros ítens quando você caminhar através do seu código

EDITANDO UM MÓDULO

Você entra com o texto numa folha de módulo exatamente como num processador de texto. Para adicionar linhas adicionais apenas pressione enter. Use o mouse para andar pela folha de módulo. Pressionando a tecla inserir indicadores entre inserir e editar durante a digitação. Você pode recortar, copiar, e colar usando os botões nas suas barras de ferramentas, os menus de comandos Editar, ou as teclas (CTRL-X para recortar, CTRL-C para copiar, e CTRL-V para colar). Também, se você selecionr texto e daí clicar com o botão direito do mouse, o seguinte menu *pop-up* aparecerá:



Se você tiver destacado o texto, os itens de menu recortar e copiar estarão ativos. A seleção Definição (shift F2) em EXIBIR lhe permite saltar para macros (mais sobre isto, mais tarde). As seleções Adicionar inspeção de variáveis, e Inspeção de variáveis rápida  , são usadas na depuração de uma macro. A opção Indicadores, permite-lhe inserir ou um *ponto de interrupção* ou um *bookmark*.

A tecla característica de **editar** num módulo é o comando desfazer (undo). Este está disponível selecionando os menus Editar, Desfazer, ou usando o botão desfazer na Barra de Ferramentas Padrão do Microsoft Excel. O comando desfazer pode ser usado repetidamente para desfazer múltiplas mudanças de edição. Você pode também ligar o comando desfazer pressionando CTRL-Z no teclado.

Quando você digitar os comandos, você poderá digitar com letras minúsculas. Isto fornece os seguintes benefícios:

- Se você entrou com as declarações corretamente, as palavras identificadas pelo Visual Basic são convertidas para a *case* apropriada quando você for à próxima declaração.

- Quaisquer palavras que não forem convertidas para o case apropriado ou não são soletradas corretamente, ou são palavras que o Visual Basic não reconhece.

Outra técnica que é útil na edição de um módulo é recuar grupos de declarações relacionadas. Você pode facilmente recuar seções de código destacando-o e pressionando a tecla Tab. Para remover recuos, pressione shift Tab. Linhas em branco podem ser usadas para agruparem comandos relacionados. Isto os torna mais fáceis de leitura e seguir as macros. Você verá muitos exemplos destas técnicas através deste material.

Você pode também usar o comando Substituir, disponível selecionando Substituir no menu Editar, para fazer as variações. Entretanto, um cuidado: O ajuste *default* para o comando substituir é substituir todo o texto correspondente, se ele é um texto destacado texto ou não. Você pode mudar estes ajustes e especificar qualquer um do que segue:

- Procedimento (a macro ou função que você está editando)
- Módulo
- Todos os Módulos
- Texto Selecionado

Existem também várias teclas de atalho que você pode usar enquanto edita um módulo. O uso das teclas de atalho pode economizar tempo quando você não tem de manter o uso do *mouse*. O que segue são as teclas de atalho que eu acho mais úteis:

Tecla de Atalho/Mouse	Resultado
Duplo clique	Seleciona a palavra do cursor
CTRL-X	Recorta o texto destacado para o clipboard
CTRL-C	Copia o texto destacado para o clipboard
CTRL-V	Cola o texto destacado do clipboard
Home	Vai para o inicio da linha
End	Vai para o final da linha
CTRL-Home	Vai para o topo do módulo
CTRL-End	Vai para o final do módulo
CTRL-down arrow	Vai para a próxima macro abaixo
CTRL-up arrow	Vai para a próxima macro acima
CTRL-F	Mostra o painel localizar
CTRL-H	Mostra o painel substituir
F1	Mostra ajuda de palavra destacada
F5	Roda a macro onde o cursor está localizado
F8	Caminha através de uma macro linha por linha

Se você mantiver apertada a tecla shift quando você usar um dos comandos relocação acima, isto selecionará todo o texto na direção que o cursor move. Por exemplo, mantendo a tecla shift pressionada quando você pressionar CTRL-End, isto selecionará todo o texto à direita do cursor.

CARACTERÍSTICAS ÚTEIS DE EDIÇÃO

Você também tem as seguintes capacidades:

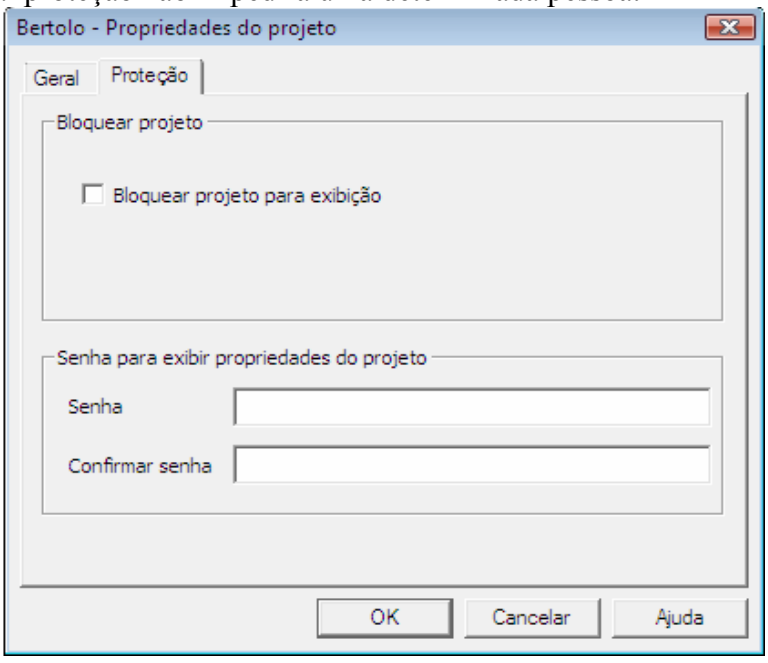
- Auto-complete - o Visual Basic editor sugere propriedades dos seus objetos para você. Pressionando a barra de espaço, um ponto, parêntese esquerdo, sinal de igual ou tab inserirá o nome sugerido.
- Recurso de edição de arrastar-e-soltar.
- Quando você destacar uma seleção e pressionar a guia Tecla, as linhas destacadas são recuadas 4 espaços.
- Você pode editar a macro enquanto estiver no modo depurar. ENTRETANTO, isto se sabe tem derrubado o Excel, tome cuidado.

USANDO TELAS SEPARADAS ENQUANTO EDITA

Se você estiver fazendo uma porção de edição num módulo, você pode separ as telas selecionando Janela, Dividir. Isto lhe permite trabalhar em duas seções separadas de um módulo. E, o tamanho das partições que foram criadas pelo item de menu Dividir pode ser mudado clicando e arrastando a barra de divisão. Você pode também dividir as telas clicando e mantendo-se sobre a barra de divisão logo acima da barra de rolamento.

PROTEGENDO SUAS MACROS

Você pode evitar os outros de acessarem o código selecionando no Visual Basic Editor Ferramentas, Propriedades de VBAProject. No diálogo que aparece, selecione a alça Proteção e ajuste a proteção do jeito que você quiser e, também, entre com uma senha. Por favor, note que *crackers* de senhas existem e que extraem uma determinada senha, assim tal proteção não impedirá uma determinada pessoa.



OPERADORES MATEMÁTICOS E OUTROS OPERADORES

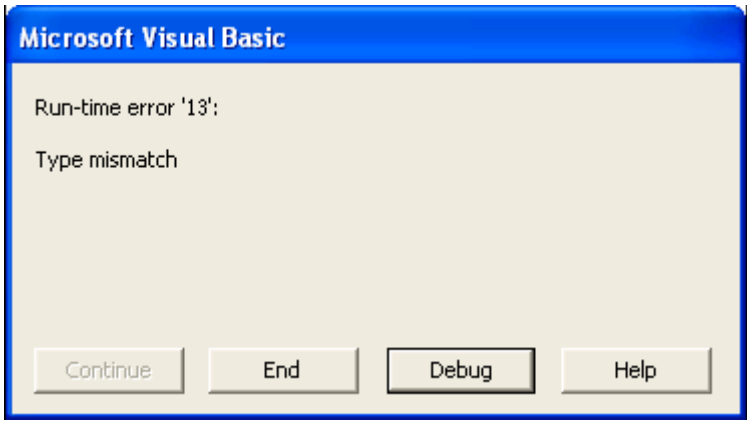
Para fazer cálculos numa macro, você usa os operadores matemáticos normais: +, -, *, e /.

A multiplicação e divisão são feitas primeiro na ordem encontrada. Por exemplo, 10/2*5 é 25. Adição e subtração são feitas a seguir, Também na ordem encontrada. Você pode usar parênteses para controlar a ordem dos cálculos.

Além dos operadores matemáticos acima, você pode também usar os seguintes operadores, nas suas declarações Visual Basic:

Operador	Explicação	/	Exemplo
^	Exponenciação.		3^2 = 9
\	Divisão inteira. Qualquer resto fracionário é descartado, deixando um resultado inteiro.		Por exemplo, 10 \ 3 = 3. Se o numerador e o denominador são frações, eles primeiro serão arredondados para inteiro antes da divisão ser feita.
Mod	Retorna somente o resto inteiro de uma operação divisão. O numerador e o denominador são primeiro arredondados. Exemplos: 5 mod 2 = 1 e 6.8 mod 3.4 = 1		

Se você tentar fazer cálculos usando um valor que o Visual Basic considera ser um texto string, ele dará a seguinte mensagem de erro:



Por exemplo, "Miami" * 3 pode não ser interpretado pelo Visual Basic assim a mensagem de erro acima é mostrada. Você pode determinar qual dos valores numa expressão matemática causou um erro usando o modo depurar do Visual Basic para examinar a expressão para texto strings, que é qualquer valor entre aspas duplas. Depuração será coberta no último capítulo.

Se o operador adição (o sinal de mais) é usado e todos os valores a serem adicionados são textos, o Visual Basic não mostrará erro. Ao invés disso, ele concatenará os valores.

Qualquer coisa, texto ou números, entre aspas duplas é considerado uma string texto. Por exemplo, "3" + "5" + "2" torna-se "352", não o número 10. Se um dos valores na expressão é um número e os outros são textos, uma mensagem de erro do tipo combinação mal sucedida será mostrada.

USANDO & PARA CONCATENAR STRINGS

Existirão inúmeras vezes em que você precisará combinar duas strings de texto juntas. Isto poderia ser feito usando o operador de concatenação, o *ampersand* (&). Por exemplo:

"Trabalho" & "E" & "Mais Trabalho"

torna-se "Trabalho E Mais Trabalho"

Por favor, note que quando você usar o & para combinar *strings*, você deve colocar um espaço na frente dele e depois dele. Se você não colocar, poderá obter um erro. Note que aspas duplas são usadas para cercar o texto. Isto indica que ela é um dado e não instruções para o Visual Basic.

Você pode também concatenar números e texto. Por exemplo:

1008 & 44	torna-se:	"100844"
972 & " Main Street"	torna-se	"972 Main Street"

Note que os resultados acima são agora considerados *strings* (representadas por estarem entre aspas duplas). Se você precisar converter uma *string* para um número, então a função **Val** (discutida posteriormente) fará isto: **Val**("100844") conduz 100844 a número. **Val**("972 Rua Principal") torna 972. O texto é truncado.

SEPARANDO LINHAS COM O CARACTERE CONTINUAÇÃO

Se uma declaração Visual Basic é muito longa par ser mais facilmente vista numa única linha, então um espaço e um caractere sublinhado (_) é colocado no final de uma linha para permitir a declaração continuar na próxima linha. Esta é uma convenção do Visual Basic padrão, e que você pode usá-la quando escrever macros. Você pode usar até nove sublinhados numa declaração única. Você não precisa separar linhas se você não quiser. Eu tenho escrito linhas que são extremamente longas. A desvantagem de se fazer isto é que é mais difícil de lê-las!

O que segue ilustra o uso de um caracteree de continuação:

```
MsgBox = _
     "Isto é um exemplo de continuação de uma declaração."
```

Em alguns lugares o caractere de continuação pode ser usado antes ou após o sinal de igual, operadores matemáticos (+,-, *,/), nomes de variáveis, e seguindo o ponto que junta duas palavras chaves. Você verá muito poucas ilustrações do caractere de continuação neste material, pois as linhas deste material são limitadas nos comprimentos!

Se você usar o caractere de continuação para separar o texto de uma longa mensagem, como mostrado pelo que segue, você obterá um erro.

```
MsgBox = "Isto é um exemplo de como não separar _
          uma longa mensagem"
```

Para separar corretamente esta mensagem, você precisa usar a seguinte abordagem:

```
MsgBox = "Isto é um exemplo de como separar " & _
"uma longa mensagem"
```

Note que existe espaço seguindo a palavra '`separar`' para separá-la da palavra '`uma`'. E, cada conjunto de palavras que estiver entre aspas duplas.

LINHAS DE COMENTÁRIOS

As linhas de comentários nas macros começam com uma aspas simples ('). Para enfatizar mais os comentários nas várias macros ilustradas neste material, eles são negritados e são sublinhados. Por exemplo:

```
'Isto é com o que se parece uma linha de comentário numa macro
deste material
```

Embora você não possa sublinhar comentários num módulo, é possível mudar o *Plano de fundo* e a cor das linhas de comentários como discutido anteriormente neste capítulo para atingir um efeito semelhante nos seus módulos.

Uma linha de comentário como a seguinte:

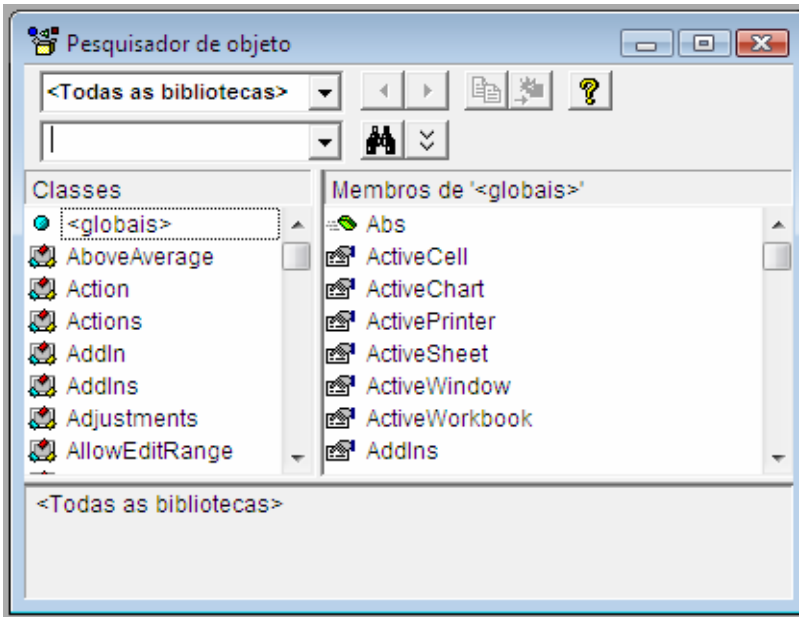
```
'************************************************
```

Freqüentemente é usada para separar uma dada macro de outra, nas ilustrações deste material. Você pode usar a mesma abordagem nos seus módulos. Linhas separadoras são automaticamente adicionadas à tela entre macros. Se por alguma razão você não tiver estas barras separadoras, verifique as opções no VBA Editor. Linhas separadoras não são impressas quando você imprimir seu código.

MOVENDO-SE DE UMA MACRO À OUTRA

Mais cedo ou mais tarde, você descobrirá que têm muitas macros. E que você está gastando uma porção de tempo indo de uma macro a outra. Entretanto, existem várias maneiras muito rápidas de ir de uma macro para outra. O modo mais simples é usar CTRL-seta para cima ou CTRL-seta para baixo se as macros estão no mesmo módulo.

Outra abordagem para se mover de macro a macro é usar o Pesquisador de Objeto. O Pesquisador de Objeto permite-lhe ver uma lista de macros em cada módulo e daí ir diretamente a uma macro, mesmo se a folha de módulo estiver oculta. Para conseguir o Pesquisador de Objeto, você pode clicar no botão Pesquisador de Objeto  na barra de ferramentas Padrão do Visual Basic. Ou, selecionar Exibir, Pesquisador de Objeto enquanto você estiver num módulo. Ou, apenas pressionar a tecla F2. O que segue aparece:



Na caixa Biblioteca/Pasta você pode selecionar qualquer pasta aberta pelo nome do seu projeto. O nome *default* para um arquivo é "VBAProject". Uma vez tendo selecionado um projeto, os módulos daquele projeto/pasta serão listados na janela do lado esquerdo. Quando você selecionar um módulo, as macros e funções do módulo selecionado são mostrados na caixa do lado direito. Se você der um duplo clique sobre uma macro, você pode então ir diretamente a ela clicando sobre o botão Show.

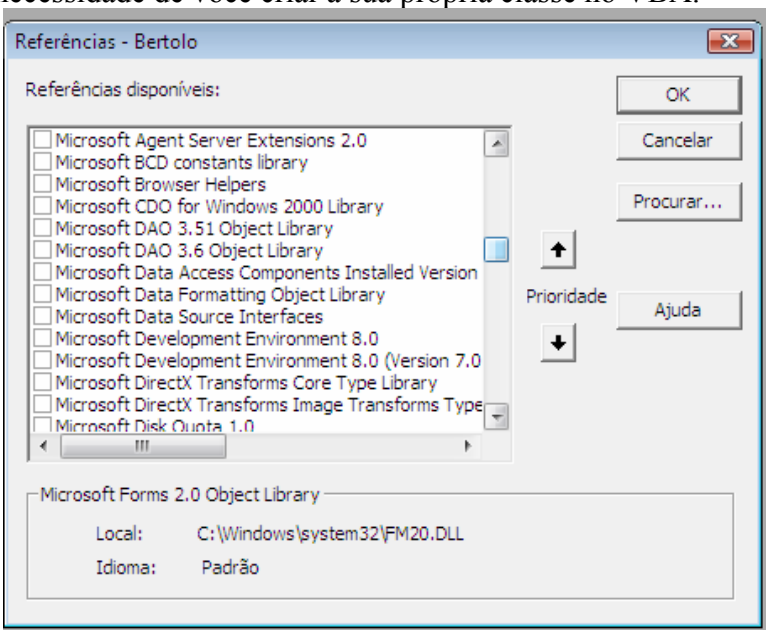
O Pesquisador de objeto fornece, portanto, todas as **classes** disponíveis no VBA com suas respectivas propriedades e métodos. Na janela Membros do objeto temos as funções, os eventos e as propriedades daquele objeto. As *funções* são representadas pelo pequeno "tijolo" em movimento. Os *eventos* são representados pelo "raio" e as *propriedades*, pelo pequeno cartão segurado por uma mão.

Além dos objetos já instalados, o VBA ainda contém referências a bibliotecas de terceiros e da própria Microsoft. Por exemplo, se desejarmos criar uma conexão a um banco de dados Access, na biblioteca inicial não existe o objeto Database ou Recordset. Porém, podemos adicionar referências a outros objetos. Para adicionar referências a outros objetos, clique em Ferramentas/Referência.

Uma caixa de diálogo contendo as referências é aberta. Como exemplo, se desejarmos criar conexões a banco de dados diretamente podemos utilizar a referência ao Microsoft DAO 3.6 Object Library, como mostra a figura abaixo.

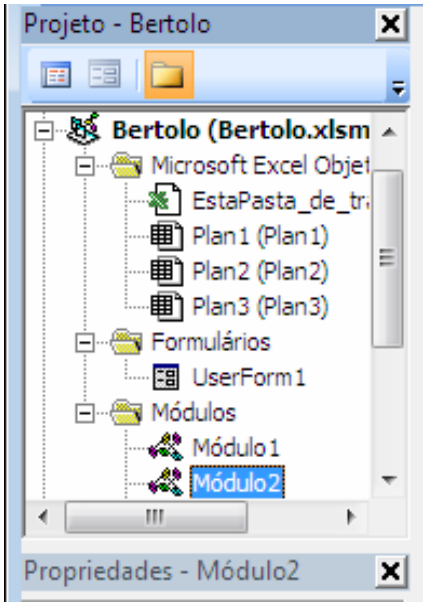
Ao acrescentarmos a referência a esta biblioteca temos à nossa disposição todas as classes da biblioteca adicionada. Isso facilita, em muito, a criação de objetos, como formulários, onde necessitamos de uma conexão direta ao banco de dados.

Na mesma lista da figura abaixo, ainda temos várias outras referências que não foram selecionadas. O número de classes disponíveis é tão grande que é improvável que haja a necessidade de você criar a sua própria classe no VBA.



MOVENDO DE UM MÓDULO A OUTRO

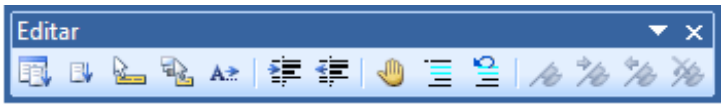
Você move de um módulo a outro com duplo clique num módulo ou *userform* listado no *Project Explorer*:



Se o *Project Explorer* não está visível, clicando no botão *Project Explorer* na barra de ferramentas Padrão ou pressionando Ctrl-R a mostrará.

USANDO INDICADORES

O Visual Basic Editor lhe dá a habilidade de adicionar bookmarks a um módulo. Isto é mais facilmente feito usando os botões na barra de ferramentas Editar. Você pode então saltar de *bookmark* a *bookmark*. Adicionando bookmarks e ir de *bookmark* a *bookmark* é mais facilmente feito usando os últimos quatro botões na barra de ferramentas Editar:



Mantendo o ponteiro do mouse sobre um botão, ele descreve o que o botão faz. Se esta barra de ferramentas não é visível, você deve mostrá-la selecionando Exibir, Barra de ferramentas de dentro do VBA editor.

PROBLEMAS DAS BARRAS DE FERRAMENTAS

Se você tiver barras de ferramentas flutuantes numa pasta, quando você indicar ao Visual Basic Editor usando ALT-F11, as barras de ferramentas flutuantes do Excel podem permanecer visíveis. Isto é um bug claro. E, ele não acontecerá consistentemente. Se a barra de ferramentas do Excel é visível no Visual Basic Editor, clicando nela joga você de volta ao Excel. Existem várias maneiras de contornar:

- Reduza todas as barras de ferramentas flutuantes. Isto lhe permite usar a tecla Alt-F11 para mover para trás e para frente. Entretanto, reduzir deve não ser uma abordagem aceitável.
- Use ALT-TAB para ir ao editor. Se você mantiver a tecla ALT apertada e não liberá-la, você pode confirmar que você está indo para o editor, ou continue a selecionar das aplicações abertas pressionando a tecla ALT.
- Use a Barra de Tarefas do Windows quando quiser permutar do Excel para o Visual Basic Editor

UMA MACRO ILUSTRATIVA

Para ilustrar a edição num módulo, a maior parte da formatação do texto acima, e mostrar-lhe quão simples uma macro pode ser, crie uma nova folha de módulo e entre com as seguintes linhas nela.

```
Sub Alô()
     'Esta macro diz alô ao mundo
     MsgBox "Alô mundo!"
End Sub
```

Se uma caixa de erro aparecer quando você estiver digitando esta macro e a linha estará em vermelho, você tem de redigitar a linha e ela precisa ser corrigida.

As palavras Sub e End Sub são usadas pelo Visual Basic para indicar o começo e o final de uma macro. Alô é o nome da macro. O parênteses são exigidos.

A linha:

```
'Esta macro diz alô para o mundo
```

é uma linha de comentário pois ela começa com aspas simples.
A linha:

```
MsgBox "Alô mundo!"
```

diz ao Visual Basic mostrar a seguinte caixa de mensagem.

Assumindo que você tenha modificado o esquema de cores no painel de formato do módulo pelas minhas recomendações, você verá que Sub e End Sub, estarão coloridas em violeta indicando que elas são palavras chave para o Visual Basic. A linha "Esta macro diz alô para o mundo" é mostrada com um Plano de fundo azul celeste para indicar que ela é uma linha comentário. A palavra MsgBox está em preto, pois ela é uma palavra de identificação. Ela representa uma instrução para o Visual Basic dizendo a ele para mostrar uma mensagem. Finalmente na frase "Alô mundo!", as aspas duplas, e os parênteses são coloridos em azul escuro par indicar que eles são textos normais em oposição a uma instrução ou comando.

Para rodar esta macro é muito simples: apenas coloque o cursor em qualquer linha da macro e então pressione a tecla F5 ou clique no botão da barra de ferramentas do Visual Basic que roda uma macro (o botão com o triângulo azul). Usando uma destas duas abordagens, vá em frente e rode a macro acima.

TROCANDO MACROS

Uma das coisas que finalmente você vai querer fazer é trocar uma macro que você escreveu com alguém. Toda vez que a macro for escrita em Inglês, trocar macros é muito simples. Você deve dar ao usuário uma cópia do arquivo contendo as macros. Se a macro está escrita numa outra linguagem, tal como Alemão ou Francês, e a linguagem da sua versão do Excel é diferente, você ainda poderá trocar arquivos. Entretanto, você precisa ter arquivos especiais do Microsoft que lhe permitam rodar as macros. Os arquivos são chamados de arquivos biblioteca do objeto. Você pode baixar os arquivos do site de Internet da Microsoft ou contatar a Microsoft por eles. Certifique-se também de obter as instruções de instalação.

SUMÁRIO DA LIÇÃO

Este material lhe fornecerá muitos exemplos de macros. Você poderá levar tempo para digitar os exemplos, mesmo se elas são somente três ou quatro linhas, dentro de um módulo e testá-los fora. Isto lhe dará experiência valiosa em usar Visual Basic e em corrigir erros que vem dos erros de digitação! Você encontrará que quanto mais você usar o Visual Basic, mais simples ele se tornará.

Lição 5: Modificando Macros no Visual Basic Editor

Vamos começar esta lição criando uma macro e depois observando e modificando seus códigos:

Para observar os códigos de uma macro devemos de seguir os seguintes passos:

1. Primeiramente translade-se à célula A5 antes de começar a gravação da Macro
2. Pressione o botão Gravar nova macro  na barra de Status do Excel**:**

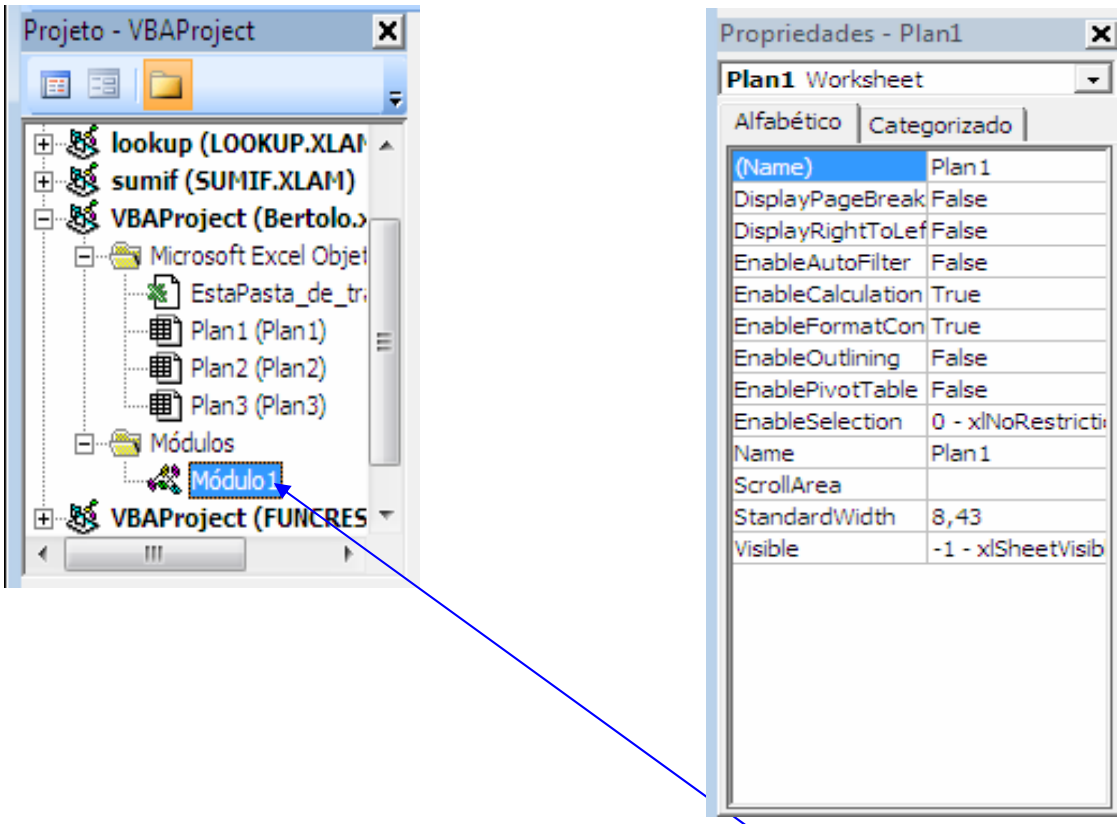
O **Excel** mostra a caixa de diálogo Gravar macro.

3. Na opção **Tecla de atalho** escreva a letra R (em maiúsculo), portanto a macro será chamada com o Ctrl +Shift+ R
4. Pressione o botão OK**.** O **Excel** inicia a gravação da **Macro1**
5. Mude-se para a célula A1 e escreva **Bertolo**, depois pressione Enter para aceitar o valor na célula
6. Pare a gravação da macro pressionando o botão Parar gravação  na barra de Status do **Excel.**

O Excel terá gravado os passos e terá gerado um código. Observemo-lo:

7. Pressione a tecla Alt + a tecla de função F11 (Alt + F11). O Excel nos leva ao **Editor** de Visual Basic.
8. Ative as seguintes caixas ou janelas:
   - Dê um clique no menu Exibir e escolha a opção Project Explorer
   - Dê um clique no menu Exibir e escolha a opção Janela de propriedades

   Estas duas opções devem estar **sempre** ativadas já que delas dependem tudo que vamos fazer.



1. Na janela **Projeto – VBAProject** dê um duplo clique em Módulos ou simplemente pressione o sinal de **+** que aparece na opção Módulos. Sob as janelas de Módulos se ativará a opção Módulo1

2. Dê um duplo clique em Módulo1**.** Será mostrado no **Editor de Visual Basic** o código da macro que gravamos, da seguinte forma:

```
Sub Bertolo()
'
' Bertolo Macro
'
' Atalho do teclado: Ctrl+Shift+R
'
    Range("A1").Select

    ActiveCell.FormulaR1C1 = "Bertolo"

   Range("A2").Select

End Sub
```

O que significa isto, perguntaremos nós assombrados. A seguir damos-lhes uma explicação do que fez o **Excel**:

- Sub e End Sub indicam o início e o final do procedimento da **Macro1**
- Tudo o que aparecer com um apóstrofe ´, indica que não se levará em conta, pois é somente texto ou comentários e esses textos devem aparecer numa cor, ou seja, a cor verde.
- Range**("**A1").Select **-** Indica que a primeiro coisa que fizemos ao gravar a macro foi transladarmos à célula A1.A ordem Range nos permite transladarmos a (ou ativar) uma célula
- ActiveCell.FormulaR1C1 = "**Bertolo"** - Isto indica que se escreverá na célula em que se encontra (célula ativa), o valor de texto **Bertolo**. Tudo o que aparece entre aspas sempre será um valor de texto. A ordem ActiveCell.FormulaR1C1 nos permite escrever um valor na célula ativa.
- Range**("**A2").Select **:**Outra vez indicamos que se translade à célula A2. Isto se deve a que ao escrevermos o nome **Bertolo** em A1 pressionamos Enter e ao dar Enter pulamos para a célula A2**.**

Para compreender melhor, alteraremos o código dentro do **Editor de Visual Basic**.

O que acreditas que se passará com nossa Macro?

```
Sub Macro1()
'
' Macro1 Macro
' Macro gravada em 7/7/2007 por Bertolo
'
' Atalho do teclado: Ctrl+Shift+R
'
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "Bertolo"
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "Catanduva, 07 de Julho de 2007"
    Range("C1").Select
    ActiveCell.FormulaR1C1 = "31-2-47-13"
    Range("D1").Select
    ActiveCell.FormulaR1C1 = "Festa do Peão"
    Range("E1").Select
    ActiveCell.FormulaR1C1 = "FAFICA"
End Sub
```

Assim é terminada alteração do código e quando voltares ao **Excel** e executares a macro com **Crtl + Shift + R,** terás o seguinte:

Em A1 se escreverá **Bertolo**

Em B1 se escreverá **Catanduva, 07 de Julho de 2007**

Em C1 se escreverá **31-2-47-13**

Em D1 se escreverá **Festa do Peão**

Em E1 se escreverá **FAFICA**

Assim que salvarmos do editor, dando um clique no menu Arquivo e elegendo a opção Fechar e voltar ao Microsoft Excel (Alt +Q).

Se não deseja sair por completo, dê um clique no botão **Microsoft Excel**  que se encontra ativado na barra de ferramentas Padrão, e quando desejar voltar ao editor dê um clique no botão **Microsoft Visual Basic**  que se encontra na Barra de Ferramentas de Acesso Rápido.

Agora já que saimos do **Visual Basic** e estamos no **Excel** de novo executemos a macro através de **Ctrl + Shift + r** e vejamos os resultados de nossa modificação.

Parece simples ou não? Claro, necessitamos praticar bastante para dominar isto, antes de passarmos adiante. Outra coisa, não trates de gerar códigos muito complexos em tuas macros porque se enrolarás, pouco a pouco se chegará ao longe.

Prática

- Gere uma **Macro** que escreva um nome em uma célula e o ponha em negrito e observe o **Código**.
- Gere uma **Macro** que escreva um nome em uma célula e o Centralize. Observe o **Código**.
- Gere uma **Macro** que escreva um nome em uma célula e troque o tamanho da letra para 20 pontos. Observe o **Código**.

Vejamos alguns códigos muito comuns:

Trasladar-se a uma Célula

```
Range("A1").Select
```

Escrever em uma Célula

```
Activecell.FormulaR1C1="Bertolo"
```

Letra em Negrito

```
Selection.Font.Bold = True
```

Letra Cursiva (Itálico)

```
Selection.Font.Italic = True
```

Letra Sublinhada

```
Selection.Font.Underline = xlUnderlineStyleSingle
```

Centralizar o Texto

```
With Selection
.HorizontalAlignment = xlCenter
End With
```

Alinhar à esquerda

```
With Selection
```

```
.HorizontalAlignment = xlLeft
End With
```

Alinhar à Direita

```
With Selection
.HorizontalAlignment = xlRight
End With
```

Tipos de Letra (Fonte)

```
With Selection.Font
.Name = "AGaramond"
End With
```

Tamanho de Letra (Tamanho de Fonte)

```
With Selection.Font
.Size = 15
End With
```

Copiar

```
Selection.Copy
```

Colar

```
ActiveSheet.Paste
```

Cortar

```
Selection.Cut
```

Ordenar Ascendente

```
Selection.Sort Key1:=Range("A1"), Order1:=xlAscending,
Header:=xlGuess, _
OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
```

Ordenar Descendente

```
Selection.Sort Key1:=Range("A1"), Order1:=xlDescending,
Header:=xlGuess, _
OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
```

Buscar

```
Cells.Find(What:="Bertolo", After:=ActiveCell,
LookIn:=xlFormulas, LookAt _
:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext,
MatchCase:= _
False).Activate
```

Inserir Arquivo

```
Selection.EntireRow.Insert
```

Eliminar Arquivo

```
Selection.EntireRow.Delete
```

Inserir Coluna

```
Selection.EntireColumn.Insert
```

Eliminar Coluna

```
Selection.EntireColumn.Delete
```

Abrir uma Pasta

```
Workbooks.Open Filename:="C:\Meus documentos\video safe 3.xls"
```

Gravar um Pasta

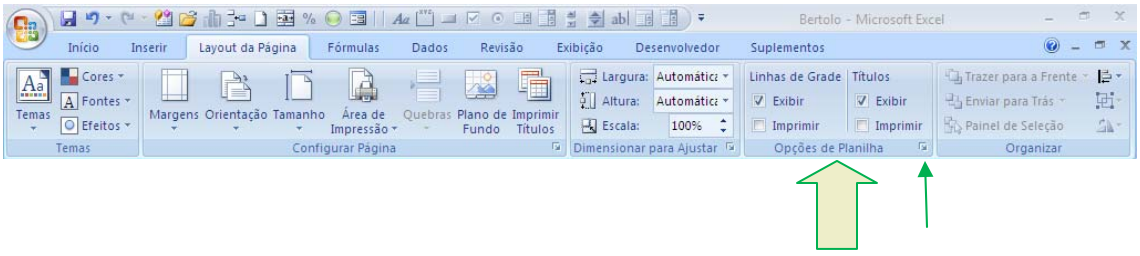
```
ActiveWorkbook.SaveAs Filename:="C:\Meus documentos\piscis.xls",
FileFormat _
:=xlNormal, Password:="", WriteResPassword:="",
ReadOnlyRecommended:= _
False, CreateBackup:=False
```

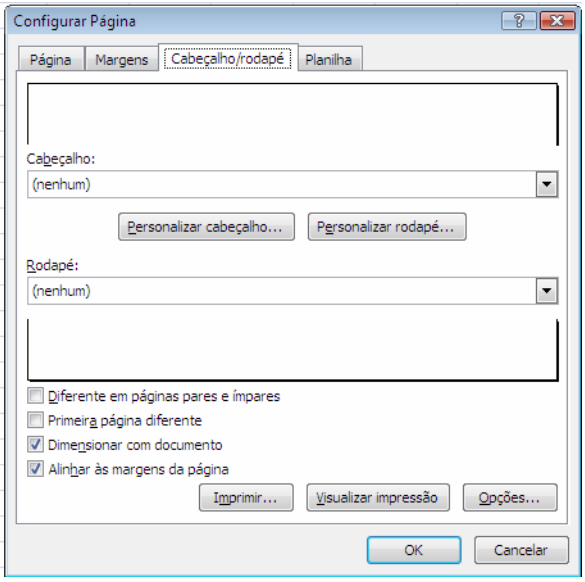
Esses seriam alguns códigos muito comuns no **Excel**, mas se você deseja pode gerar mais códigos de outras opções, é uma questão de que os ocupe.

EXTERMÍNIO DO GRAVADOR DE MACRO

Um dos problemas com macros criados pelo gravador de macro é que as macros resultantes incluem muito mais declarações do que você precisa. E, freqüentemente declarações que você não quer. O exercício que segue ilustra isto gravando uma macro para remover o cabeçalho e rodapé:

- Selecionar o botão Gravar nova macro na barra de Status do Excel. A configuração de referências relativa ou absoluta não afetará esta macro.
- Especificar "mudarConfiguracao" como o **nome da macro** e daí selecionar OK para começar a gravação.
- Selecionar a guia Layout da Página, o grupo Opções de Planilha e clicando na setinha a janela Configurar Página aparece, nela selecionamos a alça Cabeçalho/rodapé





- Mudar o cabeçalho do nome da folha para nenhum. (Ou, selecionar "(nenhum)" na caixa *pull down*, ou selecionar cabeçalho personalizado e deletar o cabeçalho.
- Fazer a memsma coisa para o rodapé
- Selecione OK para sair deste painel, e OK para sair do painel de configuração de página.
- Selecionar o botão Parar gravação  na barra de Status do **Excel.** Vá à macro que você gravou e observe nela o código. Ele se parecerá com o seguinte:

```
Sub mudarConfiguracao()
    With ActiveSheet.PageSetup
        .PrintTitleRows = ""
        .PrintTitleColumns = ""
    End With
    ActiveSheet.PageSetup.PrintArea = ""
    With ActiveSheet.PageSetup
        .LeftHeader = ""
        .CenterHeader = ""
        .RightHeader = ""
        .LeftFooter = ""
        .CenterFooter = ""
        .RightFooter = ""
        .LeftMargin = Application.InchesToPoints(0.75)
        .RightMargin = Application.InchesToPoints(0.75)
        .TopMargin = Application.InchesToPoints(1)
        .BottomMargin = Application.InchesToPoints(1)
        .HeaderMargin = Application.InchesToPoints(0.5)
        .FooterMargin = Application.InchesToPoints(0.5)
        .PrintHeadings = False
        .PrintGridlines = True
        .PrintNotes = False
```

```
        .PrintQuality = 300
        .CenterHorizontally = False
        .CenterVertically = False
        .Orientation = xlPortrait
        .Draft = False
        .PaperSize = xlPaperLetter
        .FirstPageNumber = xlAutomatic
        .Order = xlDownThenOver
        .BlackAndWhite = False
        .Zoom = 100
    End With
End Sub
```

No que está acima você viu vários valores tais como "xlPortrait" ou "xlAutomatic" ou "xlPaperLetter". Os programadores da Microsoft usam esta convenção quando se referem a uma constante numérica do Visual Basic. Por exemplo, se você executou o seguinte código:

```
Sub ShowConstantValue
     MsgBox xlPortrait
End Sub
```

Então uma caixa de mensagem aparecerá com o número 1 na caixa de mensagem.

Mesmo que você faça somente duas mudanças em sua configuração de página, o gravador de macro gerou 34 linhas de código! Isto ocorreu porque o Microsoft Excel grava **todas** as configurações da página e não apenas aquelas que você mudou. Estas definições são as propriedades do objeto **PageSetup**. Quantidades semelhantes de código resultam com muitas outras gravações. Por favor, note que esta quantidade de código em excesso é provavelmente um caso extremo. A maioria das macros que você gravar não será esta coisa feia.

Um lado positivo do gravador de macro gerar muito mais código é que você encontra freqüentemente comandos que você estava inconsciente deles! Por exemplo, a macro acima ilustra o comando With. O comando With evita você de ter de digitar novamente o nome PageSetup em cada linha. Você pode aprender mais sobre o comando With colocando o cursor na palavra With e pressionando F1. Isto o levará à ajuda Visual Basic deste comando. Este comando é também discutido em detalhes em lições posteriores.

Quando você obtiver muito código como resultado da gravação de uma macro, você deverá manter somente os pedaços que você precisar. Por exemplo, o que segue é todo o código que você precisa a fim de deletar os cabeçalhos e rodapés numa folha de planilha:

```
Sub ChangeSettings()
    With ActiveSheet.PageSetup
        .LeftHeader = ""
        .CenterHeader = ""
        .RightHeader = ""
        .LeftFooter = ""
        .CenterFooter = ""
        .RightFooter = ""
```

```
    End With
End Sub
```

Como a macro acima será usada num exercício seguinte, você deverá deletar o excesso de código de modo que sua macro se pareça com a macro acima.

LIMPANDO O CÓDIGO DO GRAVADOR DE MACRO

O que segue é outra macro que foi gravada pelo gravador de macro. Sua tarefa é colocar em várias entradas de texto e colocar caixas ao redor das células à direita destas entradas:

```
Sub Macro1()
    Range("A2").Select
    ActiveCell.FormulaR1C1 = "Descrição de Caso:"
    Range("C2").Select
    Selection.Borders(xlLeft).LineStyle = xlNone
    Selection.Borders(xlRight).LineStyle = xlNone
    Selection.Borders(xlTop).LineStyle = xlNone
    Selection.Borders(xlBottom).LineStyle = xlNone
    Selection.BorderAround Weight:=xlThin, _
            ColorIndex:=xlAutomatic
    Range("A4").Select
    ActiveCell.FormulaR1C1 = "Nome do Arquivo"
    Range("C4").Select
    Selection.Borders(xlLeft).LineStyle = xlNone
    Selection.Borders(xlRight).LineStyle = xlNone
    Selection.Borders(xlTop).LineStyle = xlNone
    Selection.Borders(xlBottom).LineStyle = xlNone
    Selection.BorderAround Weight:=xlThin, _
                ColorIndex:=xlAutomatic
    Range("A6").Select
    ActiveCell.FormulaR1C1 = "Nome da Planilha"
    Range("C6").Select
    Selection.Borders(xlLeft).LineStyle = xlNone
    Selection.Borders(xlRight).LineStyle = xlNone
    Selection.Borders(xlTop).LineStyle = xlNone
    Selection.Borders(xlBottom).LineStyle = xlNone
    Selection.BorderAround Weight:=xlThin, _
          ColorIndex:=xlAutomatic
End Sub
```

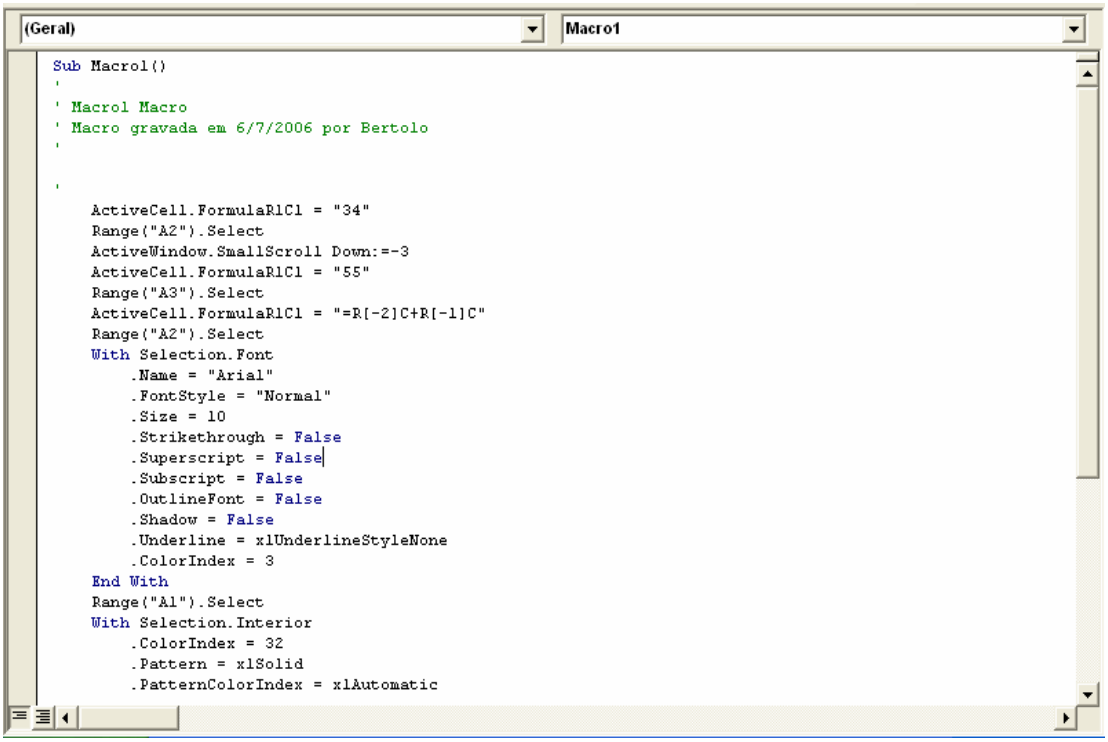
O que está acima consiste de 24 linhas de código. Compare-o à macro seguinte que faz a mesma tarefa com somente 4 linhas de código. Também, note que nenhuma Selects foi usada, o que desacelera suas macros. Reduzindo o número de linhas de código torna suas macros mais legíveis e mais rápidas.

```
Sub A_Shorter_Macro()
   Range("A2").FormulaR1C1 = " Descrição de Caso:"
   Range("A4").FormulaR1C1 = " Nome do Arquivo "
   Range("A6").FormulaR1C1 = " Nome da Planilha "
   Range("C2, C4, C6").BorderAround Weight:=xlThin
End Sub
```

Outro Exercício

É muito importante esta lição. Recomendo que você, através de sua criatividade, invente várias coisas para serem feitas no Excel usando o Gravador de Macros (GM) e daí interpretando os códigos gerados. Acredito que é assim que se aprende esse assunto. Vamos agora modificar um procedimento VBA que você criou com o gravador de macro anteriormente. Abra o Excel, abra o pasta "VBATeste1.xls" e vá para o **Visual Basic Editor** (Alt + F11). Dê um duplo clique no Módulo1 na **janela de projeto** do VBA e o código seguinte aparecerá na Janela de Código.

Clique na janela de código e imprima o módulo para referência futura "Arquivo/Imprimir/ Módulo Atual"



Imprima esta página e siga as instruções passo a passo.

O Gravador de Macros (GM) tem uma maneira muito especial para escrever códigos. O procedimento acima funciona de modo que poderíamos deixá-lo assim mas apenas para torná-lo mais claro, vamos fazê-lo mais simples. Você pode trocar cada linha de código por si mesmo ou copiar/colar por inteiro o novo procedimento abaixo desta *web page* sobre o antigo procedimento no VBE

Quando eu iniciei o Gravador de Macros a célula A1 estava selecionada e eu entrei com 34 nela. O GM escreve:

```
ActiveCell.FormulaR1C1="34"
```

Eu nunca uso e você nunca usará a propriedade FormulaR1C1. Em segundo lugar adicionaremos uma linha de código para dizer ao Excel para iniciar este procedimento na célula A1 em vez disso, 34 será entrado na célula que está selecionada

quando iniciarmos o procedimento. A Célula A1 é a ActiveCell e você pode trocar esta linha simplesmente por:

```
Range("A1").Select
ActiveCell.Value=34
```

Linha 2 e 3 do procedimento para ler como esta:

```
Range("A2").Select
ActiveCell.FormulaR1C1 = "55"
```

Novamente evitaremos uma coisa como FormulaR1C1 e não selecionaremos uma célula apenas e lhe daremos um valor vamos então trocar as linhas 2 e 3 por esta:

```
Range("A2").Value=55
```

Lembre-se que você não precisa selecionar uma célula para dar a ela um valor.
Troque as Linha 4 e 5 do procedimento para ler com esta:

```
Range("A3").Select
ActiveCell.FormulaR1C1 = "=R[-2]C+R[-1]C"
```

Novamente evitaremos a FormulaR1C1. Novamente não queremos selecionar uma célula apena entraremos com uma fórmula vamos então trocar as linhas 4 e 5 por esta:

```
Range("A3").Formula="=A1+A2"
```

As Linha 6 e 7 do procedimento para ler como esta:

```
Range("A2").Select
Selection.Font.ColorIndex = 3
```

Aqui nós não selecionaremos a célula nós apenas especificaremos que nós queremos mudar a cor da fonte vamos então trocar as linhas 6 e 7 por esta:

```
Range("A2").Font.ColorIndex = 3
```

O três está em vermelho. Eu sempre uso o GM nesta situação porque eu não sei como fazer para lembrar todos os códigos de cores.
Linha 8 a 12 do procedimento para ler com esta:

```
Range("A1").Select
With Selection.Interior
        .ColorIndex = 41
        .Pattern = xlSolid
End With
```

O GM usa um monte de "With..End With" mas eu não. O código desenvolvido pelo GM deverá ler num Inglês simples: Selecione a célula A1 (célula A1 torna-se a Seleção) daí então para o Selection.Interior torne a ColorIndex 41 e a Pattern xlSolid. Aqui está uma versão mais simples de trocar as linhas 8 até 12 por isto:

```
Range("A1").Interior.ColorIndex = 41
Range("A1").Interior.Pattern = xlSolid
```

Linha 13 a 25 do procedimento lê assim:

```
Range("A3").Select
With Selection.Font
        .Name = "Arial"
        .Size = 24
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
End With
```

Novamente o GM usa "With...End With" e também modifica TODAS as propriedades da fonte cada vez. Queremos apenas que o tamanho da fonte seja mudado para 24 vamos então trocar as linhas 13 até 25 por isto:

```
Range("A3").Font.Size= 24
```

O procedimento VBA, ou macro chamada "Macro1" agora se parece com isto:

```
Sub Macro1()
' Macro1 Macro
' Macro gravada 27/5/2007 por Bertolo'
    Range("A1").Select
    ActiveCell.Value = 34
    Range("A2").Value = 55
    Range("A3").Formula = "=A1+A2"
    Range("A2").Font.ColorIndex = 3
    Range("A1").Interior.ColorIndex = 41
    Range("A1").Interior.Pattern = xlSolid
    Range("A3").Font.Size = 24
End Sub
```

Teste agora esta macro.

Lição 6: Escrevendo Macros no VBA do Excel

O modo mais fácil para se começar a aprender o *Visual Basic* é escrevendo uma macro. Esta lição lhe ensinará muito do Visual Basic fazendo você criar uma macro que **somará**, **subtrairá**, **multiplicará** ou **dividirá** por qualquer valor que você especificar. E, para ajudá-lo a aprender, eu vou ter que cometer erros! Você aprenderá mais rapidamente com os erros, porque deste modo você ganhará experiência que lhe permitirá controlar os seus próprios erros no futuro. E, esperançosamente, evitar esses mesmos erros!

AS PRIMEIRAS DECLARAÇÕES DE MACRO

O primeiro passo que você precisa fazer é criar um módulo novo. Abra o Excel, renomeie a folha de planilha Plan1 para "Teste" (clicar com o botão direito do mouse na guia, escolher "Renomear"e digitar o nome nela). A seguir vamos criar um novo módulo. Vá para o editor de VB (ALT-F11) e Selecione Inserir, Módulo.

Se você ajustou anteriormente as opções do **Módulo Geral**, a declaração Option Explicit será a linha de topo do novo módulo. Se não, mude as suas opções ou digite Option Explicit no topo do módulo.

Para criar a macro que nós buscamos, primeiro comecemos tentando escrever uma macro estigmatizante que **divida por 100**. Você sempre deverá digitar minúsculas com a exceção de linhas de comentários, que são linhas começando com uma aspa única, e o nome da macro. O *Visual Basic* capitalizará palavras que ele reconhece, e que não estão numa linha de comentário. Entre com o que se segue em seu módulo como o primeiro passo para escrever esta macro:

```
Sub DividirPor100 ()
     'Esta macro divide por 100
     ActiveCell.Value = ActiveCell.Value / 100
End Sub
```

Se uma caixa de erro apareceu quando você foi de uma linha para outra ou a linha é vermelha, você digitou incorretamente a linha e precisa corrigi-la. O "**End Sub**" é adicionado automaticamente quando você entrar na primeira linha no módulo.

A primeira linha no macro, **Sub** `DividirPor100 ()`, começa com o palavra **Sub**. Todos as macros têm que começar com a palavra chave Sub seguida pelo nome da macro. A palavra **Sub** para os programadores de computadores estabelece um "procedimento sub-rotina".

Se você digitar **Sub** em minúsculo, quando você entrar na primeira linha, o Visual Basic mudará isto de sub para Sub quando você for para a próxima linha. Este é o modo do Visual Basic falar para você que ele reconheceu **Sub** como uma palavra chave que ele conhece.

Imediatamente após **Sub** está a entrada "`DividirPor100 ()`". Este é o nome para a macro. Nomes de macro seguem as regras seguintes:

- Um nome de macro tem que começar com uma letra.
- Um nome de macro deverá conter só letras, números e o caráter de sublinhado (_). Não use caracterees especiais como o ponto, asterisco, ponto de exclamação, ou outros caracterees especiais (#, $,%, etc..).
- Um nome de macro não pode incluir espaços.
- Um nome de macro deverá fazer sentido e deveria dizer-lhe o que o macro faz (por exemplo "DB100" realmente não carrega o mesmo significado que "DividirPor100"). Deverá fazer sentido agora e daqui a seis meses.
- Um nome de macro pode conter letras minúsculas e maiúsculas juntas. As palavras em maiúsculo em um nome de macro a torna mais legível (DividirPor100 em vez de dividirpor100).
- Um nome de macro não pode ser o mesmo nome de um módulo ou uma palavra chave do Visual Basic (por exemplo, `ActiveCell ou MsgBox`).
- Um nome de macro pode ser bastante longo, mais de 250 caracterees!

Outra parte do nome de macro são os dois parênteses ao término do nome da macro. Esta é uma exigência de Visual Basic. Você não precisa adicionar os parênteses, pois o Visual Basic os adicionarão automaticamente para você quando você for para outra linha!

Antes de nós darmos uma olhada nas outras linhas dessa macro, cometamos alguns erros no nome de macro. Volte e faça as mudanças seguintes ao nome de macro, e tente ir para a linha imediatamente próxima após cada mudança:

- Escreva o nome usando espaços (`Divida Por 100`)
- Incluir pontos no nome (`Dividir.Por.100`)
- Comece o nome com um número (`100Dividir`)

Nunca se esqueça da cor vermelha! Elas existem para ajudá-lo. Afinal de contas, é melhor saber que você tem um erro do que não saber! Se você não corrigir a linha, você ainda pode mover para outra linha. Porém, a linha vermelha permanece até que o problema seja corrigido. Esteja seguro de corrigir a linha atrás para **Sub** `DividirPor100 ()` senão sua macro não funcionará! Quando você ganhar experiência suficiente em escrever macros e não precisar destas advertências, jogue fora as exibições das opções das sintaxes erro em Ferramentas, Opções.

A segunda linha na que você entrou,

```
'Esta macro divide por 100
```

é uma linha de comentário. Linhas de comentários são linhas que começam com uma aspa simples. A formatação em azul dos comentários neste é feita para enfatizar que a linha é uma linha de comentário. Eles não são sublinhados em um módulo. Nas linhas

de comentário você fornece se a palavra é maiúscula ou minúscula. O Visual Basic capitalizará apenas as palavras que reconhecer e, que estão nas linhas de declaração (não linhas de comentários).

Se você omitir a aspa no começo de uma linha de comentário, você ganhará uma mensagem de erro (a linha fica vermelha) quando você tentar mover à próxima linha. Este é modo Visual Basic de lhe falar que você deu mancada! Por favor, note que o texto realçado sempre não é onde o problema está. Por exemplo, se você digitou a linha de comentário corretamente, volte e remova a aspa simples e vá para a próxima linha. A linha ficará vermelha e o Visual Basic realçará a palavra "divide" na linha de comentário, embora o problema seja a aspa simples perdida. Para consertar a linha de comentário, reponha a aspa simples de volta no começo da linha.

Comentário também pode ser colocado à direita de declarações. O seguinte ilustra isto:

```
Sub DividirPor100()        'Esta macro divide por 100
    ActiveCell.Value = ActiveCell.Value/100  'divide por 100
End Sub             ' Final da macro DividirPor100
```

Também, podem ser postas linhas em branco na macro em qualquer local. Linhas de espaço em branco ajudam freqüentemente as leituras de suas macros.

A próxima declaração em sua macro é a seguinte:

```
ActiveCell.Value = ActiveCell.Value/100
```

Se você digitou esta linha em minúsculas, o Visual Basic capitalizou as palavras que ele reconheceu. Vá para a macro e mude apenas a letra " A " na palavra `ActiveCell` para minúscula. Quando você mover para outra linha, o "a" é mudado de volta para "A" pelo `Visual Basic`.

A melhor aproximação é digitar tudo, salvo linhas de comentário, nome de macro, e texto incluídos em aspas duplas e em minúscula. Deste modo, se o Visual Basic não capitalizar a palavra, você sabe que foi digitado errado.

Para ilustrar isto, mude "`ActiveCell`" para "`activcell`" (torne-a minúscula e apague o **e** dentro de `active`). Quando você faz isto e vai para a próxima linha, a palavra permanece minúscula, e nenhuma mensagem de advertência de qualquer espécie aparece. Por que o Visual Basic não dá uma mensagem de erro devido a ortografia incorreta? O Visual Basic pensa que você está criando que uma variável definida pelo usuário. Uma variável definida pelo usuário é usada para armazenar valores ou outras informações. Esteja seguro em mudar " `activcell` " de volta para " `ActiveCell` ".

A palavra `ActiveCell` diz ao Visual Basic que você está se referindo à célula que é realçada pelo ponteiro de célula. A palavra `Value` depois do ponto diz ao Visual

Basic que você está se referindo ao *Valor* daquela célula. Assim, `ActiveCell.Value` refere-se ao valor da célula ativa. O *valor* de uma célula é uma das muitas propriedades de célula que você pode especificar. Como vimos em lição anterior algumas das outras propriedades de célula que podem ser obtidas e modificadas são:

Propriedade de célula	Retorno
Fórmula	fórmula na célula com um sinal igual
Font.Name	O nome da fonte
Font.Bold	True se estiver em negrito, Falso se não estiver
NumberFormat	Format da célula

Se a expressão `ActiveCell.Value` está à direita do sinal igual, diz ao Visual Basic para retornar com o *valor* da célula ativa. Se a célula contiver um número, o número é retornado. Se a célula contiver uma fórmula, o valor numérico da fórmula é retornado. Se a célula contiver texto, então texto é retornado. Se a célula estiver vazia, então um valor nulo é retornado. Um valor nulo é tratado como um zero nos cálculos. Em equações de texto, um valor nulo é anunciado por "", duas aspas duplas não tendo um espaço entre elas.

Se a expressão que `ActiveCell.Value` está do lado esquerdo de um sinal igual, diz ao Visual Basic para atribuir para a célula o valor do que é calculado à direita do sinal igual. Os exemplos seguintes ilustram isso:

Declaração de macro:	Valor atribuído à Célula:
`ActiveCell.Value` = 45	45
`ActiveCell.Value` = 1/2	0.5
`ActiveCell.Value` = "ABC"	ABC

Note que Visual Basic avalia a expressão à direita do sinal igual antes de atribuí-lo ao valor da célula ativa. Por exemplo, a entrada "1/2" se tornou "0.5".

Com este *background*, nós podemos entender o que faz a declaração `ActiveCell.Value = ActiveCell.Value / 100` em nossa macro. A expressão `ActiveCell.Value` à direita do sinal de igual retorna o valor da célula. Ele é então dividido por 100 e o valor resultante é atribuído à célula.

A última declaração em nossa macro é a seguinte:

```
End Sub
```

A palavra chave **End** quando combinada com a palavra chave **Sub** diz ao Visual Basic que este é o fim da macro. Todos as macros têm que terminar com **End Sub**. Se você se esquecer do **End Sub**, um erro acontecerá quando você tentar rodar a macro. O editor básico visual adiciona um **End Sub** automaticamente quando você digitar a linha de **Sub**.

TESTANDO SUA MACRO

O próximo passo ao escrever esta macro é testar as declarações que você escreveu. Você deveria testar qualquer macro que você escreve muito freqüentemente. Deste modo, se você tiver um problema, é provável que as linhas que acabaram de ser mudadas é que está causando o problema.

Quando você testar uma macro, você precisa testar todas as situações diferentes que podem acontecer quando a macro é rodada. Nesta macro que nós queremos testar é as entradas de célula diferentes que se deveria rodar ao usar esta macro. Elas são:

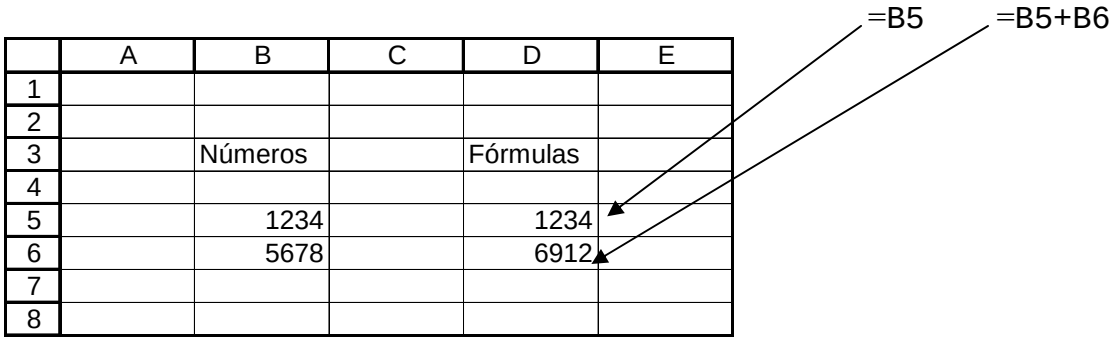
- Células contendo números
- Células contendo fórmulas
- Células contendo texto
- Células em branco

Antes de você poder testar esta macro, você precisará ir para um planilha em branco e fazer as entradas seguintes:

Célula	Entrada	Célula	Entrada
B3	"Números"	D3	"Fórmulas"
B5	1234	D5	=B5
B6	5678	D6	=B5+B6

Por favor, note que foram entrados 1234 e 5678 sem os sinais de igual.

Quando você fizer isto, sua planilha deverá se parecer ao seguinte:

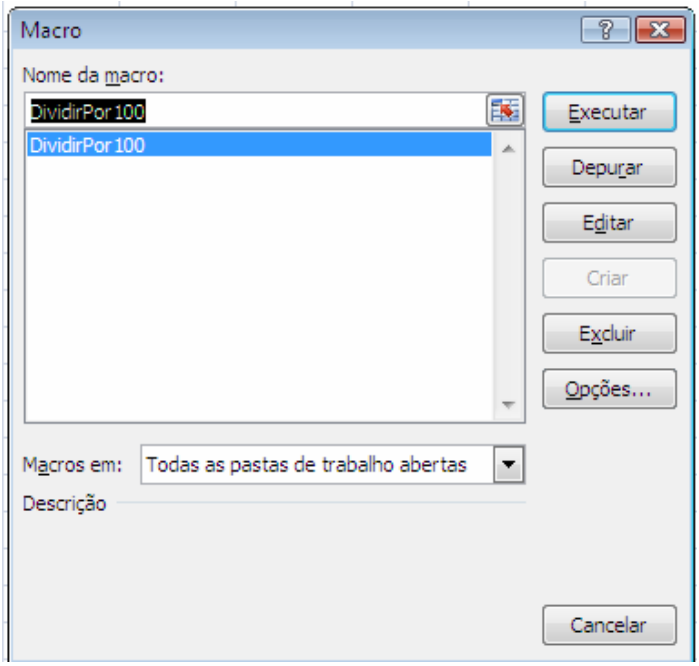


	A	B	C	D	E
1					
2					
3		Números		Fórmulas	
4					
5		1234		1234	
6		5678		6912	
7					
8					

O primeiro passo que você deverá fazer antes de testar uma macro é salvar o seu arquivo. Isto é muito importante. Se você tiver um problema e a macro não funcionar corretamente, você pode restabelecer o arquivo re-carregando-o. Também, se por

alguma razão uma determinada macro travar o Microsoft Excel (muito improvável, mas acontece), você não perderá as declarações de macro que você acabou de escrever.

Para testar a macro, primeiro selecione a célula B5. Esta é a célula que contém o número 1234. Então, selecione Ferramentas do menu da barra de Ferramentas Padrão, e daí Macro neste menu e depois Macros... no sub-menu que aparece. Você também pode exibir rapidamente este diálogo apertando ALT-F8. Isto exibe a caixa seguinte:



Se você tiver outras macros, elas também serão exibidas nesta lista. Quando você clicar uma vez no nome da macro, os botões cinzentos ficam ativos. Realce a macro *DividirPor100* e dê um clique no botão executar. A tela piscará e uma de duas coisas acontecerá:

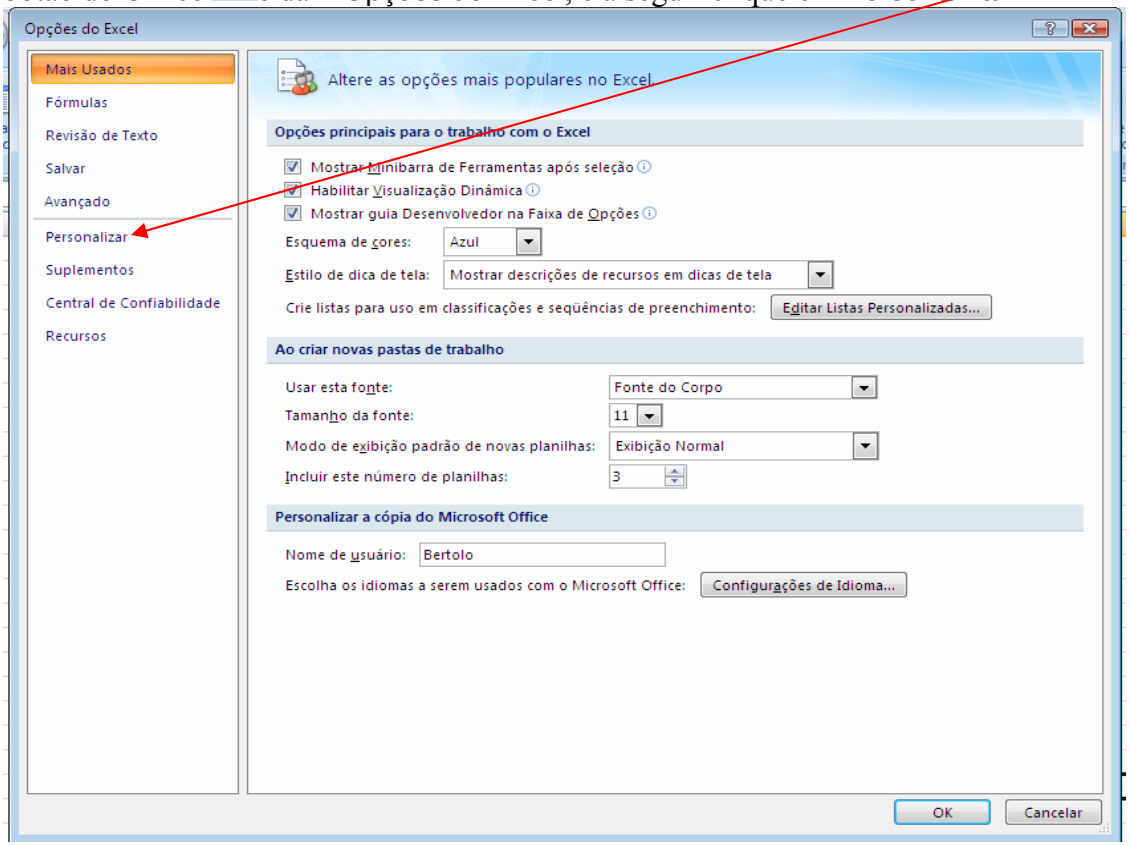
- uma caixa de erro aparecerá.
- ou o valor 1234 em B5 será trocado com 12,34.

Se uma caixa de erro aparecer, significa que você não digitou algo corretamente. A mensagem de erro exata depende do que você digitou! Para fixar, compare o que você digitou a listagem anterior da macro.

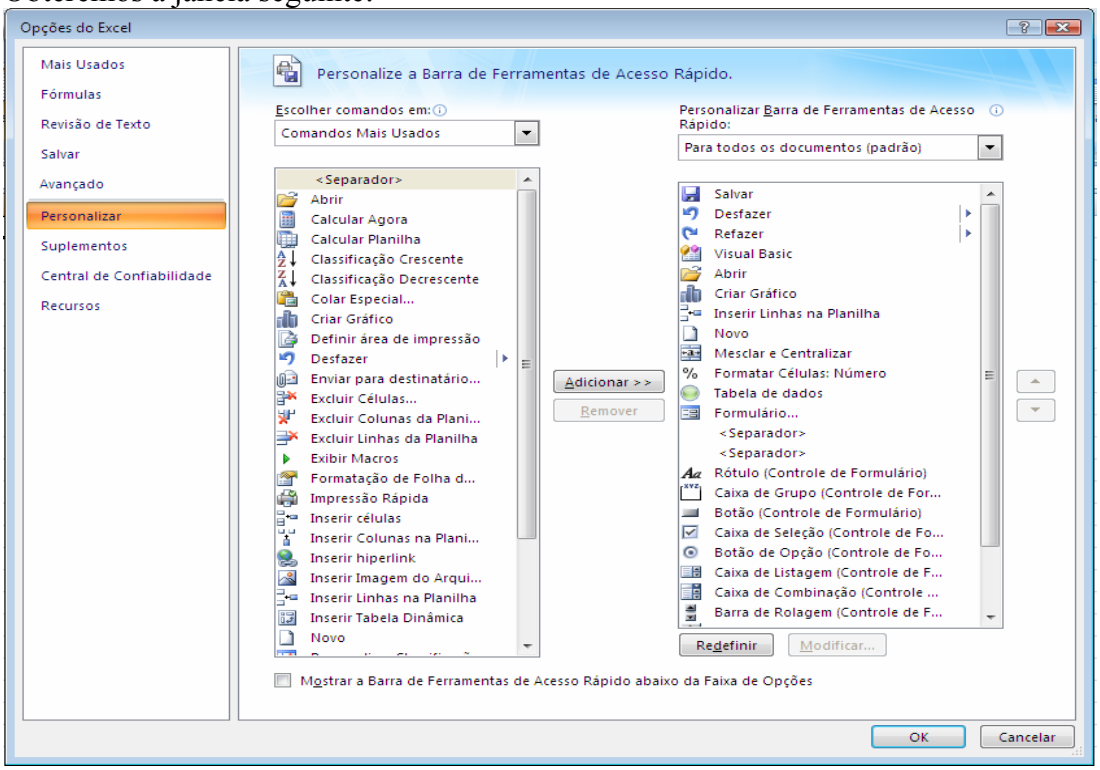
ATRIBUINDO UM BOTÃO À NOSSA MACRO

Neste momento, você tem uma macro que funciona bem com números. Antes de fazer uma prova adicional, montemos uma barra de ferramentas (*toolbar*) com um botão nela que rodará esta macro sempre que o botão for clicado. Isto eliminará os passos de se selecionar Ferramentas, Macros, selecionar a macro e clicar em Executar toda vez que você testar a macro.

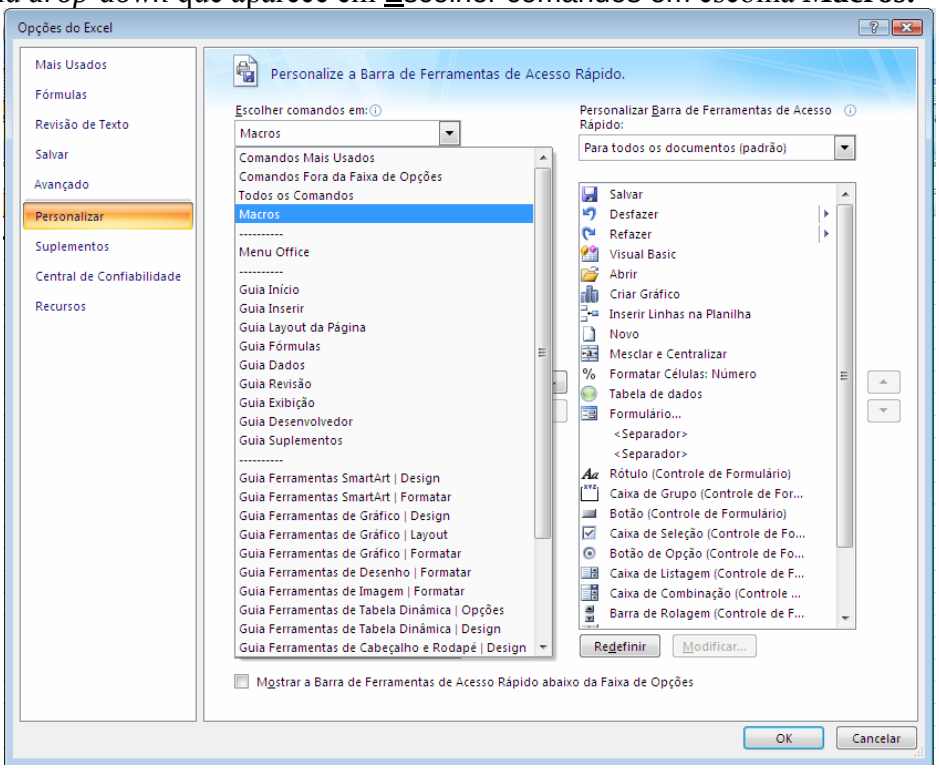
Para criar uma barra de ferramentas com um botão para esta macro, selecione o botão do Office  e daí Opções do Excel, e a seguir clique em Personalizar



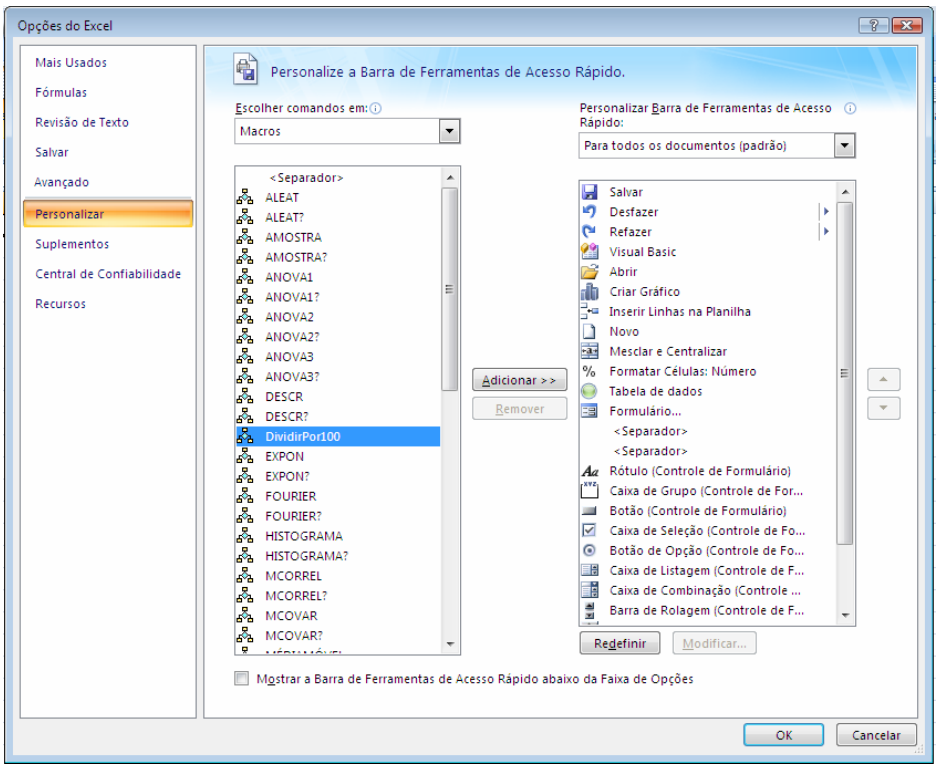
Obteremos a janela seguinte:



No menu *drop-down* que aparece em Escolher comandos em escolha **Macros:**



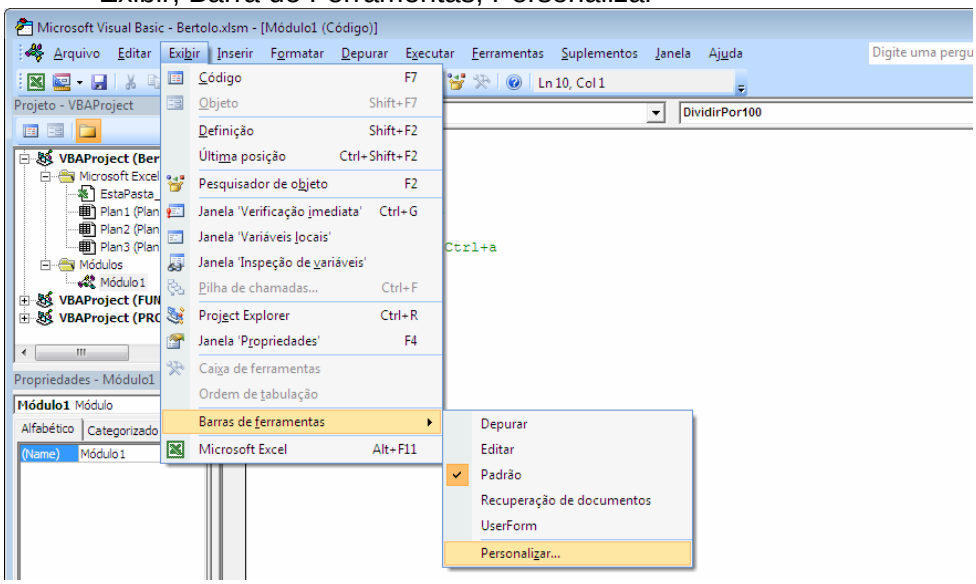
E obtenha a janela:



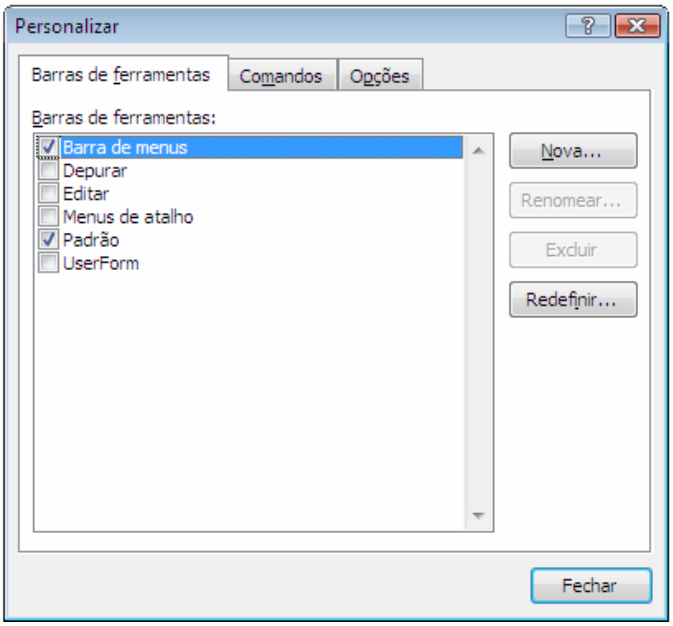
Clique no botão Adicionar e termos a macro na Barra de Ferramentas de Acesso Rápido

Para personalizar a barra de ferramentas do VBE, fazemos o seguinte

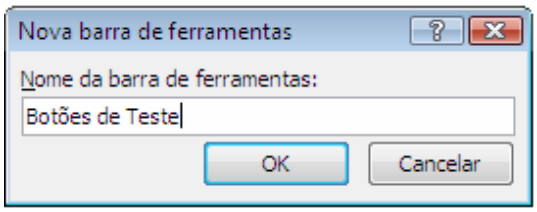
- Exibir, Barra de Ferramentas, Personalizar



Aparecerá a seguinte janela:

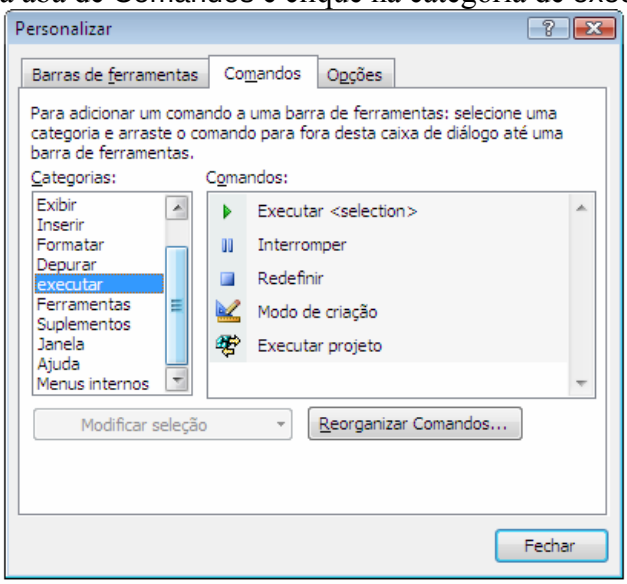


Na guia Barras de ferramentas, selecione o botão Nova... para criar uma barra de ferramentas nova. Isto lhe permitirá criar uma barra de ferramentas nova, em branco. Na caixa de diálogo que aparece dê um nome à barra de ferramentas, por exemplo, "*Botões de Teste*".



Uma nova barra de ferramentas em branco aparecerá: 

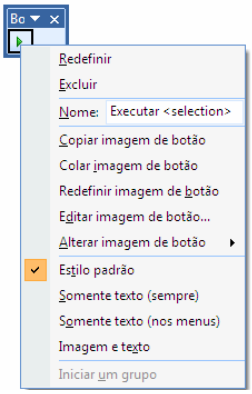
A seguir, selecione a aba de Comandos e clique na categoria de executar:



Clique sobre a face do *Executar* e arraste-o à barra de ferramentas nova 

.

Para mudar o botão a uma nova imagem, clique com o botão direito do mouse sobre a imagem no botão e selecione Alterar imagem de botão.



A seguir, escolha uma nova imagem.

Por favor, note que o Excel freqüentemente torna-se instável quando você criar barra de ferramentas. Sempre que eu crio uma barra de ferramentas nova, eu saio imediatamente do Excel e daí, volto a abri-lo. Eu acho que se eu não fizer isso, o Excel tende a cair mais cedo ou mais tarde.

TESTANDO NAS CÉLULAS DE FÓRMULAS

O próximo passo de teste de nossa macro é testá-la em células que contenham fórmulas. Antes de testar, primeiro tenha certeza de que você repôs os valores nas células B5 e B6 de novo para 1234 e 5678. Os valores nas células D5 e D6 deverão ser respectivamente 1234 e 6912, e elas deverão conter fórmulas, não números.

Para testar uma fórmula, clique sobre D5 e daí clique no botão de teste que você acabou de criar. A notícia boa é que o valor em D5 muda de 1234 para 12,34. A notícia ruim é que ela não é mais uma fórmula, mas um valor 12,34. É preciso encontrar então uma maneira precisa que modifique as células com fórmulas de modo que a fórmula seja retida.

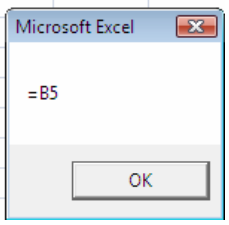
CONSERTANDO A MACRO PARA CÉLULAS DE FÓRMULAS

Para descobrir qual a fórmula que está em uma célula, você precisa usar a *propriedade* Fórmula em vez da *propriedade* Valor. Assim, usando a expressão `ActiveCell.Formula` em vez de `ActiveCell.Value` lhe é devolvido uma fórmula em vez do valor na célula. Mas o que acontece se a célula contém um número em vez de uma fórmula? Para descobrir o que você obtém neste caso com `ActiveCell.Formula`, criemos uma segunda macro que mostre a saída da expressão `ActiveCell.Formula`. Usar pequenas macros, para obter informações ou testar um novo código, é uma técnica muito útil.

Entre várias linhas abaixo da macro anterior *DividirPor100* com esta outra macro que se segue. Esteja certo em digitar minúscula, com a exceção da palavra "*testeMacro*" que é o nome da nosso macro.

```
Sub testeMacro()
     MsgBox ActiveCell.Formula
End Sub
```

A macro acima exibirá uma caixa de mensagem na tela que contém a fórmula da célula ativa. Vá para sua planilha de teste e rode a macro de teste em várias células. A figura seguinte ilustra o que você obterá se você clicar na célula D5 que contém a fórmula " =B5 " e rodar a macro de teste.



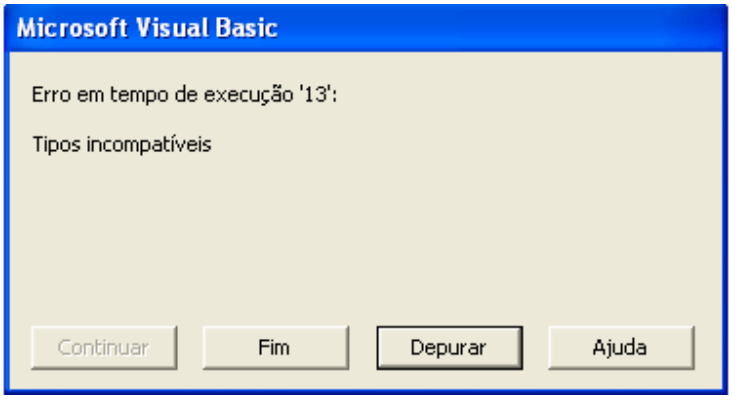
Rode a macro de teste em células que contenham números, numa que esteja em branco, e em uma que contenha texto. O seguinte é os resultados que você adquirirá:

Se a Célula Contém:	ActiveCell.Formula Retorna:
Fórmula	Fórmula com um sinal de igual
Número	O número
Texto	O texto
Nada	Nada

Assim, nós precisamos modificar a macro *DividirPor100* para usar `ActiveCell.Formula` em vez de `ActiveCell.Valor`. Sua macro deverá se parecer com o seguinte quando você tiver terminado:

```
Sub DividirPor100()
' Esta macro divide por 100
     ActiveCell.Formula = ActiveCell.Formula / 100
End Sub
```

Agora volte para sua planilha de teste e teste-a. Teste-a primeiro em números e daí em células de fórmula. Quando você testá-la em números, ela funciona da mesma maneira que antes. O número 1234 se torna 12,34, o número 5678 se torna 56,78. Porém, quando você rodá-la em uma célula de fórmula, as coisas ficam de mal a pior! A caixa de erro seguinte aparece:



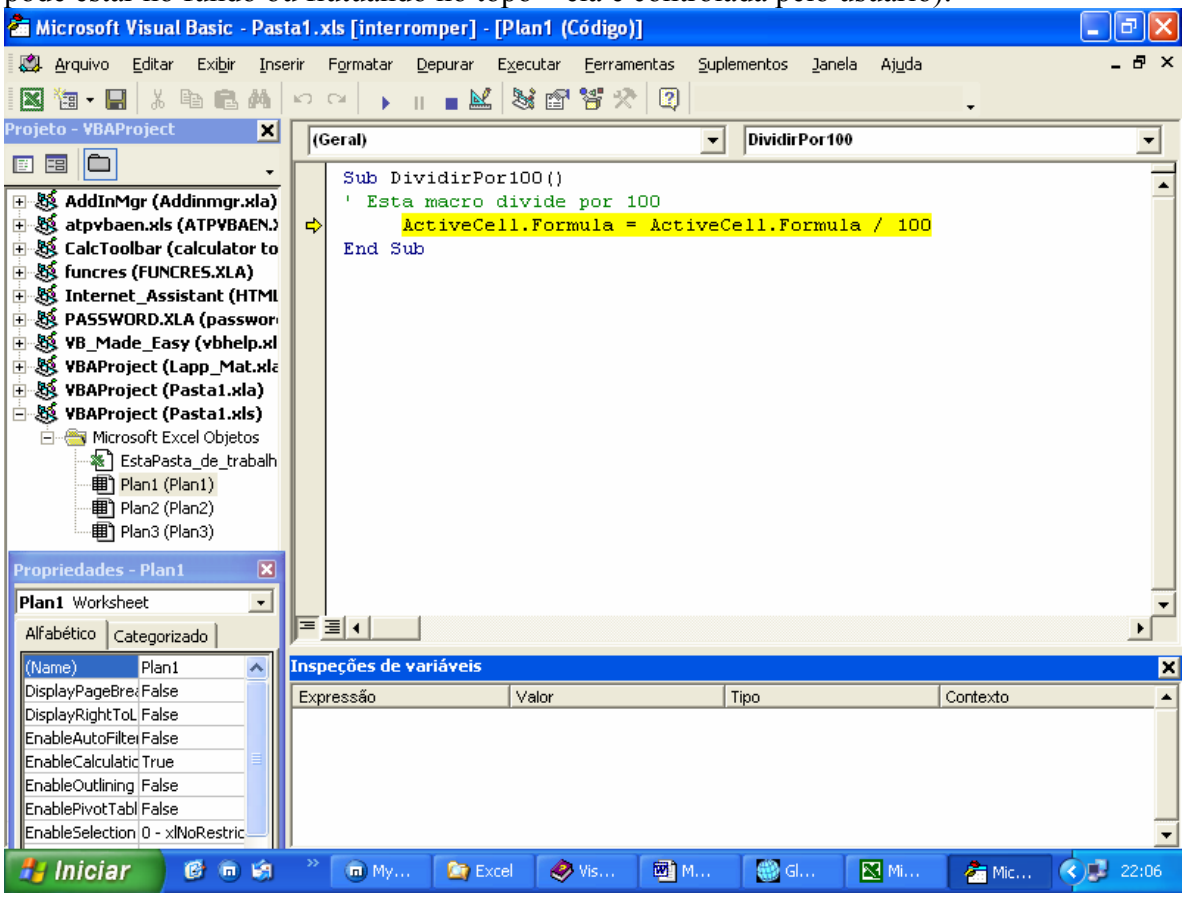
Selecionando o botão Fim lhe é devolvido a pasta, se você executou a macro de uma pasta de planilhas. Se você executou a macro do editor de VB, ele retorna você ao editor.

Para encontrar qual a linha que está causando o problema, selecione o botão Depurar. Isto resultará na linha com o problema começar realçada. Por favor, note que isto o põe no modo DEPURAR no editor de VB. E, você pode corrigir a declaração e clicar no botão executar/continuar para retomar a macro de forma que você possa modificá-la sem correr o risco de travar o Excel. Por ora, selecione do menu Executar, Redefinir. Isto lhe permite parar a macro e então voltar à pasta de planilhas e recomeçá-la do princípio. Embora as correções de declarações possam ser feitas enquanto estiver no modo depurar, mais cedo ou mais tarde fará o Excel e o editor visual basic travar.

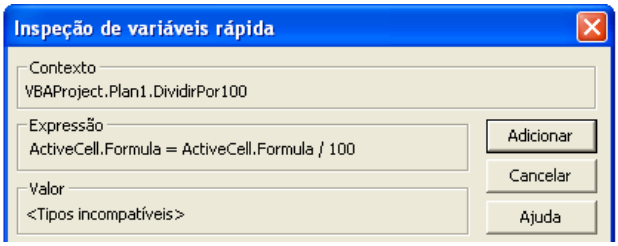
Quando você encontrar um erro numa linha de fórmula, ele normalmente significa que algo está errado do lado direito do sinal de igual. Neste caso, algo está errado com `ActiveCell.Formula/100`. A fórmula mostra-se bem. E, não está em vermelho, o que indica que você digitou algo erradamente.

Para descobrir o que está acontecendo, volte para a célula `D5` que contém a fórmula `"=B5"` e execute a macro novamente. Quando a caixa de erro se aparecer desta vez, selecione Depurar. Daí então, selecione Exibir, Janela 'Inspeção de variáveis' para

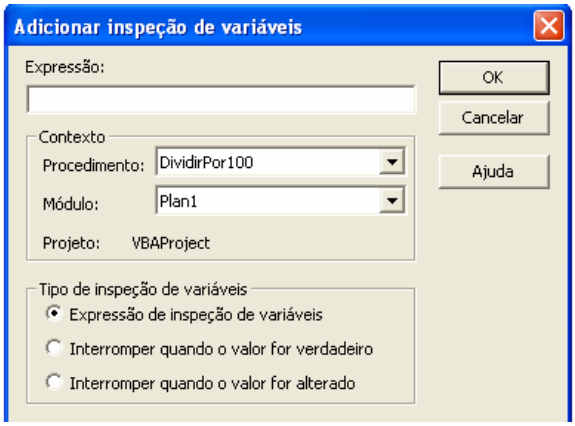
fazer a janela inspeção aparecer. O seguinte ilustra o modo depurar com uma janela Inspeção dentro e a linha que causadora do problema realçada (sua janela de inspeção pode estar no fundo ou flutuando no topo – ela é controlada pelo usuário):



Para usar as características do Depurar, primeiro certifique-se que o painel Inspeção esteja mostrado. Depois, a porção realçada `ActiveCell.Formula/100` da linha do problema. Dai clique na linha realçada e vá para o menu Depurar e procure o item Inspeção de variáveis rápida ou Depurar, Adicionar Inspeção de variáveis. Aparecerá o seguinte, se confirmar sua ação e se você clicar o botão de Inspeção Rápida:

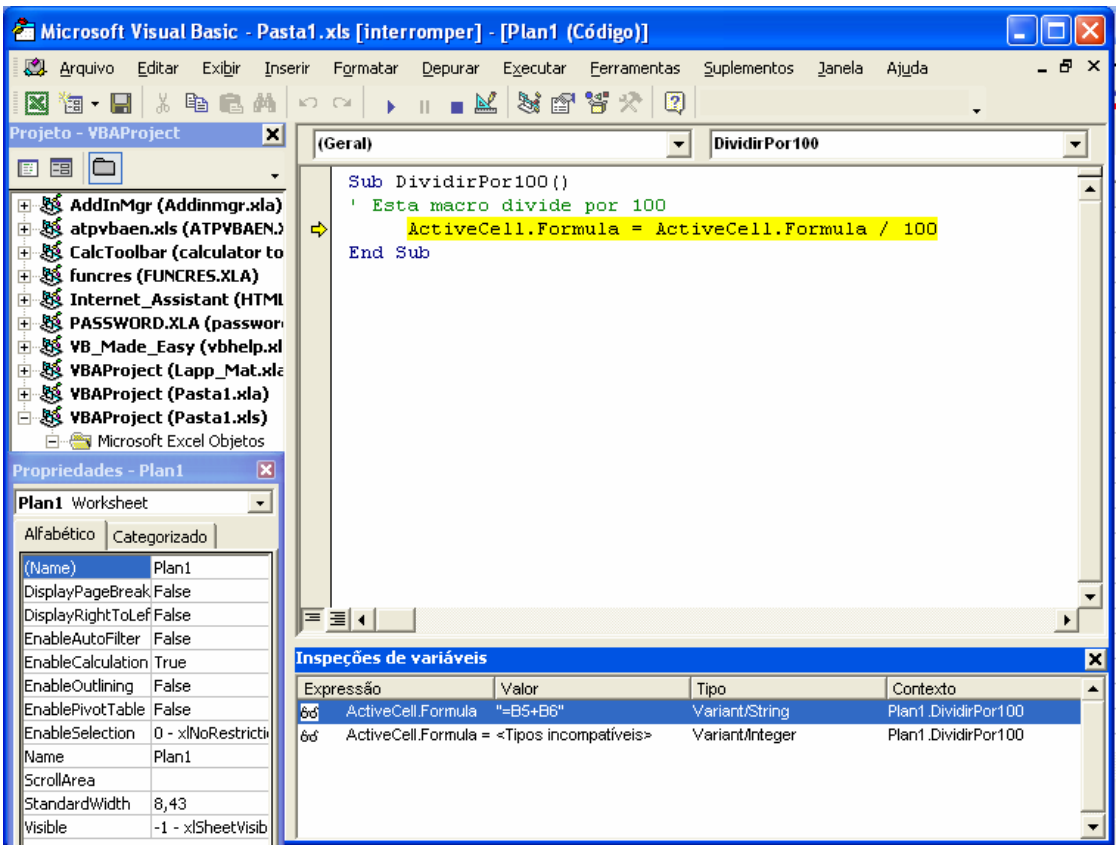


Se você selecionar Depurar, Adicionar inspeção de variáveis aparecerá o seguinte:



Selecione OK na caixa de diálogo. Depois, selecione `ActiveCell.Formula` e repita as mesmas seleções.

O seguinte é o que a tela mostrará após as inspeções terem sido feitas com o ponteiro na célula `D5` da planilha:



Destas "inspeções" você vê que quando o Visual Basic avalia o `ActiveCell.Formula` que está rodando a fórmula na célula, "=B5". Mas quando este é dividido por 100, o Visual Basic retorna "<Tipos Incompatíveis>". Por que isto está ocorrendo? O Visual Basic está dividindo o texto ("=B5") por 100, o que não pode

ser feito. Daí uma mensagem de erro, "Tipos Incompatíveis", porque ele está fazendo uma operação matemática no texto.

O que queremos que aconteça é terminar com "= (B5)/100" na célula. Para fazer isto acontecer, o que você precisa fazer é cercar o "/100" com aspas duplas e concatená-lo com ActiveCell.Formula usando um ampersand (&). Cercar / 100 com aspas duplas diz ao Visual Basic que isto é texto em oposição à expressão matemática que deveria executar. O texto pode ser concatenado usando um ampersand (&). Certifique-se que tenha colocado espaços na frente e após o ampersand (&).

Para ordenar a macro, selecione Executar, Redefinir para sair do modo Depurar. Ou, e ainda mais facilmente, clique no botão quadrado ■ que é o botão parar/redefinir para retornar a edição *normal* e sair do modo depurar. Modifique então a declaração na macro para ficar como o que se segue:

```
ActiveCell.Formula = ActiveCell.Formula & "/ 100"
```

Quando você digitar isto nela, tenha certeza de que você colocou um espaço antes e outro depois do **&**. De outra forma, você obterá uma mensagem de erro, quando o Visual Basic não conseguir entender ActiveCell.Formula&. Tente isto sem o espaço de modo que você também experimente este erro. Você recordará dele quando ele acontecer a você no futuro. Tente também escrever a declaração sem o & para fazer aparecer uma mensagem de erro.

MANIPULANDO CÉLULAS DE EQUAÇÕES COMPLEXAS

Você está ficando mais íntimo! Volte para sua folha de planilha de teste e experimente-a na célula D5 que contém a fórmula "= B5". O resultado é "= B5/100". E se você repetir isto em uma segunda vez nesta célula, você adquire "= B5/100/100". Isto está perfeito! Logo, experimente na fórmula mais complexa em célula D6 que é "=B5+B6 ". Quando você fizer isto, você obtém a fórmula "= B5+B6/100". Isto não é totalmente o que se deseja! A Microsoft Excel dividirá o valor de B6 primeiro antes do 100 e então acrescentará isto a B5 em vez de somar B5 e B6 que dividem junto e então antes das 100. O que você realmente deseja é "= (B5+B6)/100". Se a equação existente não é rodeada por parênteses, as respostas mudam. Por exemplo, 5+100/100 são 5+1 = 6, enquanto (5+100)/100 são 105/100=1.05.

Para consertar este problema, você precisa modificar a declaração de macro de forma a cercar a fórmula com parênteses e então adicionar os " /100 ". Mude a declaração de forma que ela lê:

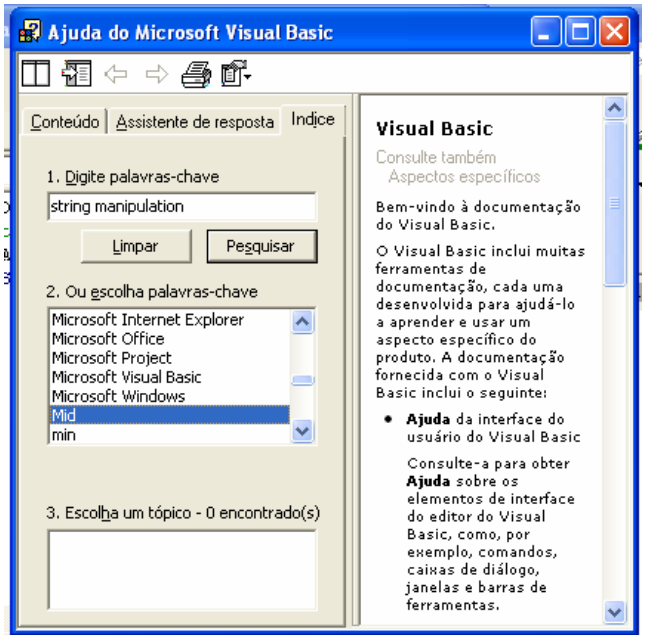
```
ActiveCell.Formula = "(" & ActiveCell.Formula & ") / 100"
```

Volte para sua folha de planilha de teste e para célula D6, e corrija-a de novo a sua fórmula original, "=B5+B6 ". Agora experimente a macro. O resultado é ligeiramente diferente do que é desejado: a célula termina com a entrada de texto

"(=B5+B6)/100" em vez de uma fórmula. O que deve ser feito é selecionar tudo da fórmula que é retornada por `ActiveCell.Formula` com a exceção do sinal de igual. Para fazer isto nós precisamos usar a função Mid que devolverá só parte do texto especificado.

USANDO A AJUDA PARA ENCONTRAR COMANDOS

Se você está desejando saber como eu propus a função **Mid**, eu a achei na ajuda do Microsoft Visual Basic. Selecione Ajuda no menu editor VB e o item de menu "Ajuda do Microsoft Visual Basic". Daí, então, selecione a guia de Índice e digite em "palavras chaves". Selecione "*string manipulation*" e procure a informação de ajuda resultante.



Este texto apresentará muitas tais funções úteis a você para livrá-lo de ter que descobri-las.

Lá apareceu a seguinte informação:

Função Mid

Retorna uma **Variant** (**String**) que contém um número especificado de caracteres de uma seqüência de caracteres.

Sintaxe: Mid(*string*, *start*[, *length*])

A sintaxe da função **Mid** tem os seguintes argumentos nomeados:

Parte	Descrição
string	Obrigatória. Expressão de seqüência da qual são retornados os caracteres. Se *string* contiver Null, será retornado Null.
start	Obrigatória; Long. A posição do caractere em ***string*** onde a parte a ser considerada começa. Se ***start*** for maior que o número de caracteres existentes em ***string***, a função **Mid** retornará uma seqüência de caracteres de comprimento zero ("").
length	Opcional; **Variant** (**Long**). Número de caracteres a ser retornado. Se omitido ou se existirem menos caracteres do que os de ***length*** no texto (inclusive o caractere em ***start***), serão retornados todos os caracteres a partir da posição ***start*** até o final da seqüência de caracteres.

.....

.....

Exemplo da função Mid

O primeiro exemplo usa a função **Mid** para retornar um número especifico de caracteres de uma seqüência de caracteres.

```
Dim MyString, FirstWord, LastWord, MidWords
MyString = "Demonstração da função Mid"    ' Criar a seqüência de texto.
FirstWord = Mid(MyString, 1, 3)    ' Retorna "Dem".
LastWord = Mid(MyString, 17, 6)    ' Retorna "função".
MidWords = Mid(MyString, 1)    ' Retorna "Demonstração da função Mid".
```

USANDO A FUNÇÃO MID

A informação de ajuda nos fala que a função **Mid** precisa ser fornecida a um texto *string*, um número indicando posição do caractere na *string* onde a parte a ser considerada começa, e o número de caracteres a ser retornado. Neste caso o número indicando a posição do caractere onde a parte a ser considerada começa é 2, porque o sinal de igual está na posição de caractere 1. Se nós não fizermos uma entrada para o argumento de comprimento (lenght), o Microsoft Excel assumirá que nós queremos a porção restante do texto, isto é o que nós queremos.

A outra mudança que nós precisamos fazer é pôr um sinal de igual na frente dos primeiros parênteses. Isto se faz para a entrada de uma célula de fórmula em vez de uma entrada de texto. A declaração de macro precisa se parecer com o seguinte depois que você modificá-la:

```
ActiveCell.Formula = _
            "=(" & Mid(ActiveCell.Formula, 2) & ")/ 100"
```

Na declaração anterior você vê um espaço e um sublinhado (_) depois do primeiro sinal de igual, e que a declaração continua na próxima linha. A combinação de um espaço e um sublinhado permite-se continuar uma declaração na próxima linha em vez de fazer a declaração estender além da área visível. Se você tem espaço em sua tela você pode digitar tudo isso em uma linha. Mas se você fizer assim, esteja certo de remover o sublinhado. A razão que eu escrevi declaração anterior em duas linhas em vez de uma é que a área de impressão neste texto não é larga o bastante para exibi-la em uma linha.

TESTANDO A MACRO MODIFICADA

É hora de testar novamente! Volte para a folha de planilha de teste e esteja certo de que as entradas em `D5` e `D6` são respectivamente "`=B5`" e "`=B5+B6`". Quando você rodar a macro em cada destas células, você obtém as fórmulas que nós buscamos:

Célula	Fórmula Era:	Fórmula Tornou-se:
`D5`	`=B5`	`=(B5)/100`
`D6`	`=B5+B6`	`=(B5+B6)/100`

Até aqui tudo bem! Mas, nós fizemos muitas modificações desde que nós testamos a macro por último em uma célula que contém apenas um número em vez de uma fórmula. Vá para a célula `B5` que contém o número 1234 e rode a macro. Os resultados são ligeiramente diferentes do que nós queremos. Nossa macro removeu o "1" de "1234" e entrou aí a fórmula "= (234/100).

USANDO IF E A FUNÇÃO LEFT

Até este momento nós escrevemos uma declaração que possa controlar números mas não pode controlar equações, e nós escrevemos uma declaração que pode controlar equações mas não pode controlar números. O que nós precisamos fazer é usar a declaração que possa controlar números somente em células que contêm números, e usar a declaração que possa controlar fórmulas somente em células que contêm fórmulas. Duas coisas são necessárias para se fazer isto:

- Uma maneira de distinguir entre células que contêm números e células que contêm fórmulas.
- E, uma maneira para dizer a nossa macro que conjunto de instruções usar.

Se você examina células que contêm números ao invés de fórmulas, você vê que a diferença fundamental é o sinal de igual. Se o primeiro caráter for um sinal igual, então nós usaremos a fórmula existente que fica depois do sinal de igual. Se o primeiro caráter não for um sinal igual, então deverá ser um número.

Nós podemos usar a função **Left**(texto, número) do Visual Basic para extrair apenas o primeiro caráter de uma célula de fórmula. Nesta função, você especifica o

texto e depois o número de caráter para retornar. Se o texto é uma equação (Ex: "=B5+B6"), e o número de caráter para devolver é apenas um, então um sinal de igual é retornado.

Para satisfazer a segunda exigência, um modo de dizer à nossa macro que conjunto de instruções usar, nós usaremos uma declaração **If**. O seguinte ilustra a simples declaração **If** que nós usaremos:

Se (If) o sinal igual for encontrado Então (Then)
use a declaração para uma fórmula
Senão (Else)
Use a declaração para um número
Fim do If

Como informação, as declarações **If** serão explicadas em detalhes em numa lição posterior.

Nossa nova macro se parece com o seguinte depois de ter sido modificada para ter a função **Left** () e os testes **If** adicionados a ela.

```
Sub DividirPor100()
' Esta macro divide por 100

  If Left(ActiveCell.Formula, 1) = "=" Then
    'Faça isto desde que a célula contenha uma função
    ActiveCell.Formula = _
              "=(" & Mid(ActiveCell.Formula, 2) & ")/100"
  Else
    'Faça isto desde que a célula contenha um número
    ActiveCell.Formula = _
              "=" & ActiveCell.Formula & "/100"
  End If
End Sub
```

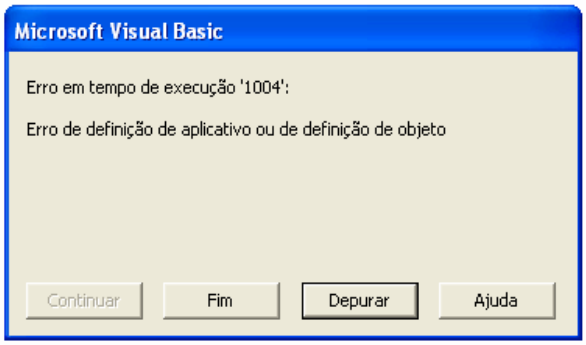
Embora nós mencionamos o sublinhado anteriormente, você pode ter perdido o parágrafo dele. Na macro anterior você vê duas declarações onde eu pus um espaço e um sublinhado (_) e que a declaração continua na próxima linha. A combinação de um espaço e um sublinhado permite-se continuar uma declaração nas próximas linhas ao invés de fazer a linha estender além da área visual. Você não pode dividir palavras usando o sublinhado a menos que você goste de mensagens de erro!

É hora de testar novamente! Fique certo de que suas células de teste estão fixadas novamente às condições originais delas (`B5` e `B6` são os números 1234 e 5678; as células `D5` e `D6` são as fórmulas `"=B5"` e `"=B5+B6"`). Os resultados, em células que contêm números e fórmulas é exatamente o que nós queremos! A célula que contém o número 1234 se torna " = (1234)/100 " e a fórmula `"=B5+B6"` se torna `"=(B5+B6)/100"`. Se nós rodarmos isto em uma segunda vez nestas duas células, os

resultados são "= ((1234)/100)/100" e "= ((B5+B6)/100)/100", também o que nós queríamos!

MANIPULANDO TEXTO E CÉLULAS EM BRANCO

Até agora, quando nós testamos a macro, nós só testamos em células numéricas. Estas são as células que contêm números ou fórmulas que avaliam a um número. Mas se nós rodarmos a macro em uma célula em branco nós encontraremos uma mensagem de erro como a seguinte:

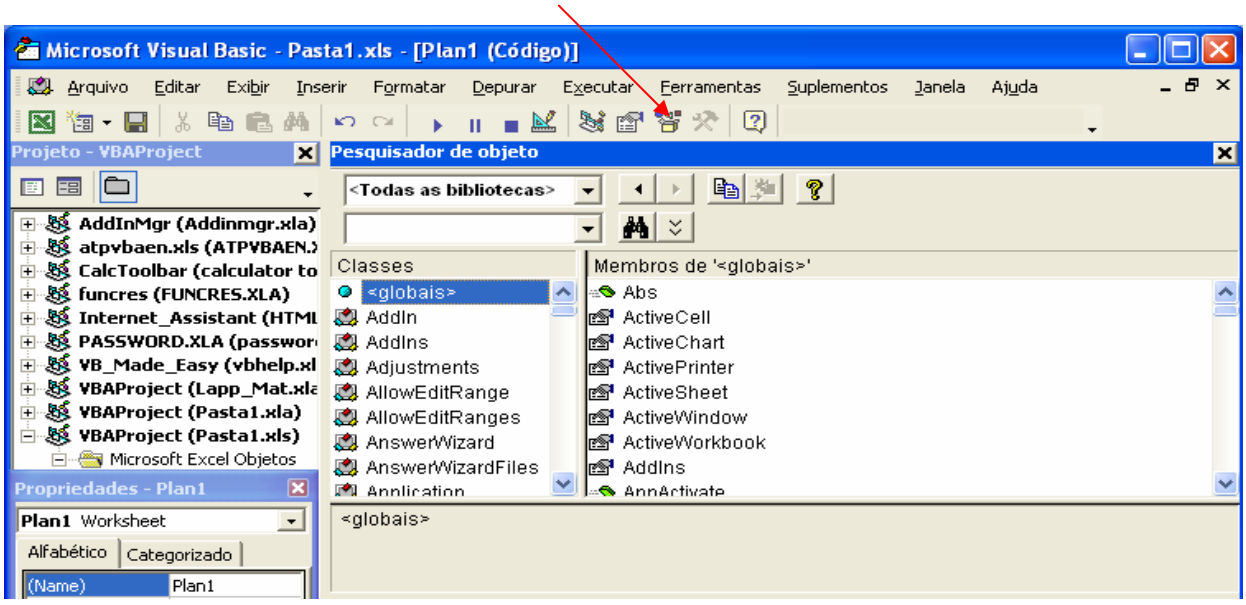


E, se nós a testarmos em uma célula que contém uma entrada de texto, nós obtemos uma fórmula sem sentido. Por exemplo, teste o macro em célula B3 que contém a palavra "Número". O resultado é a equação "= (Número)/100". O que nós queremos é que a macro ignore células que estão em branco ou têm texto.

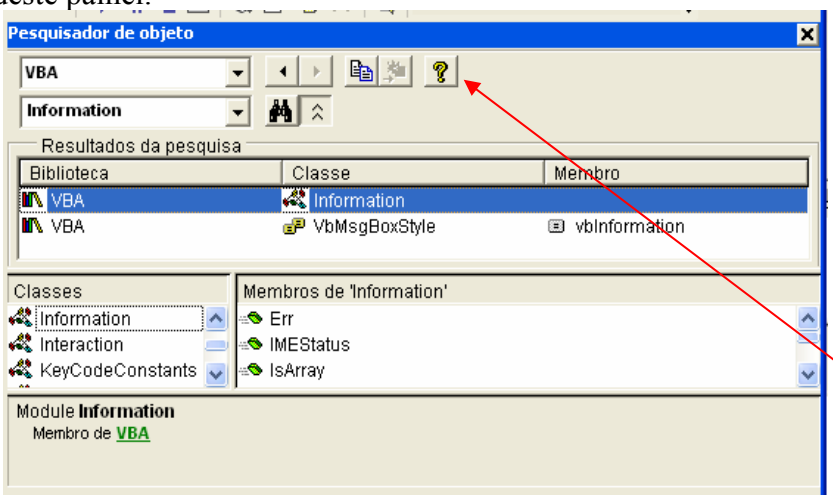
O que nós precisamos é de um modo de dizer se uma célula contém números e equações em vez de texto ou está vazia. Acontece que existe uma função Visual Basic que nos diz se uma célula é uma célula numérica. A função Visual Basic é **IsNumeric (valor)**. Se a célula contiver um número ou uma fórmula numérica, ela retorna com o valor Verdadeiro (True). Caso contrário, retorna com o valor Falso (False). Modificando a macro para usar este teste nela, um teste If, nós poderemos distinguir entre células numéricas e células não-numéricas.

USANDO O PESQUISADOR DE OBJETO (OBJECT BROWSER)

Antes de fazermos uso do **IsNumeric**, lhe mostremos como achá-lo usando o Pesquisador de Objetos. O Pesquisador de Objetos é útil para procurar por comandos. Para fazer isto, vá para o módulo Visual Basic e selecione Exibir, Pesquisador de Objetos (ou F2 ou clicando no botão-ícone). Você verá o Pesquisador de Objetos exibido da seguinte maneira:

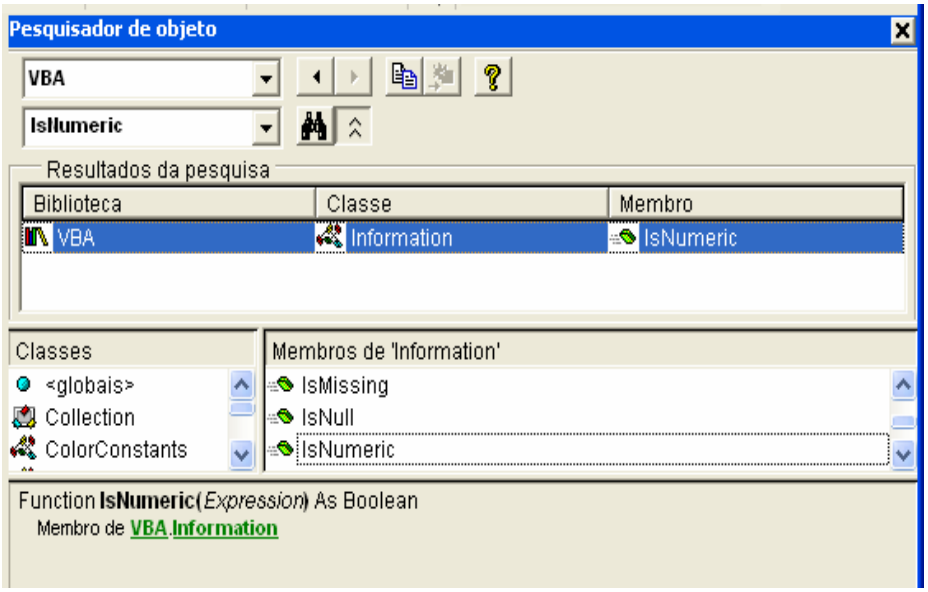


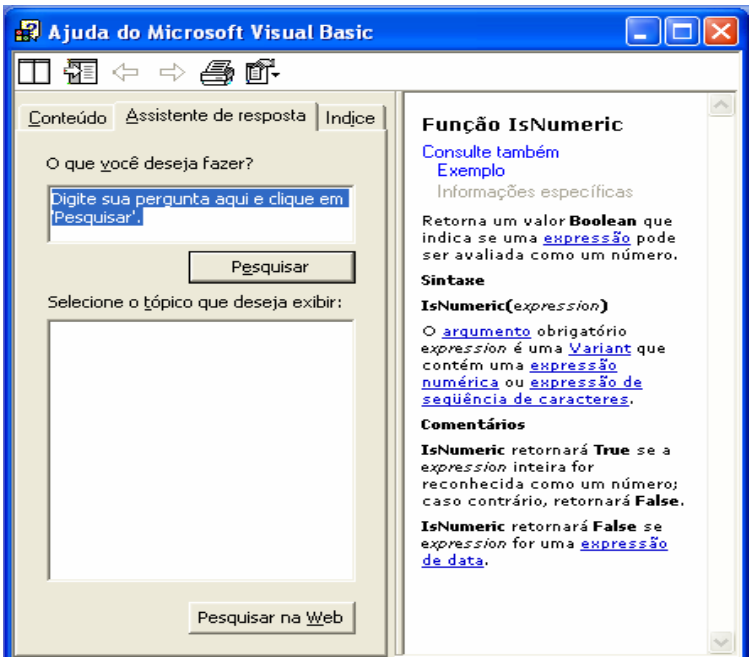
Selecione **VBA** na caixa de listagem de cima esquerda e Information na caixa de listagem de baixo lista direita. Uma descrição curta do item destacado aparecerá no fundo deste painel.



Na caixa Membros de 'Information', na janela acima, você verá as diferentes *propriedades* e *métodos* que podem ser usados para esta **classe de objeto** Information da biblioteca VBA. Se você clicar sobre uma propriedade e daí clicar no ponto de interrogação no canto esquerdo superior do painel acima, a Ajuda do Visual Basic sobre o método ou propriedade selecionada será mostrada. Freqüentemente a ajuda fornecerá vários exemplos usando o método ou propriedade selecionada.

Destaque a palavra **IsNumeric** na caixa **Membros de 'Information'** e clique sobre o botão ponto de interrogação para mostrar a informação de ajuda.





Como você pode ver da informação de ajuda (Comentário), ela retorna True se a célula for numérica e False se não for. Também, a informação de ajuda fornece a sintaxe a ser usada.

Você pode deixar o Pesquisador de Objetos ativo e simplesmente selecionar Janela no menu do editor VB e selecionar a janela que você gostaria de ativar. Você pode também fechar o Pesquisador de Objetos clicando no botão fechar no canto direito superior da janela.

USANDO IsNumeric NA MACRO

O que se segue é macro *DividirPor100* modificada para incluir o teste **IsNumeric**. Note que o uso de identações para agrupar declarações relacionadas. Também, linhas em branco são usadas para melhorar a facilidade de leitura da macro, e comentários foram colocados no final de cada declaração **If**.

```
Sub DividirPor100()
' Esta macro divide por 100

 If IsNumeric(ActiveCell.Value) Then
  'Faça isto desde que a célula seja numérica
  If Left(ActiveCell.Formula, 1) = "=" Then
    'Faça isto desde que a célula contenha uma equação
    ActiveCell.Formula = _
                "=(" & Mid(ActiveCell.Formula, 2) & ")/100"
  Else
    'Faça isto desde que a célula contenha um número
    ActiveCell.Formula = _
                "=" & ActiveCell.Formula & "/100"
  End If     'Fim do teste Se Left
 End If   'Fim do teste IsNumeric
End Sub
```

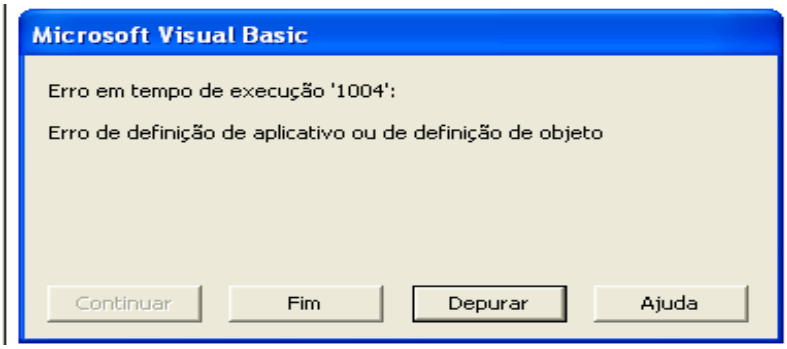
Note que nós testamos a propriedade de **Valor** da célula ativa com a função **IsNumeric**. Se nós tivéssemos testado a propriedade de **Fórmula** da célula ativa, células que contêm fórmulas teriam retornado Falso desde que fórmulas são entradas de texto.

Quando o teste de **IsNumeric** for True, são executadas as declarações imediatamente seguintes. Estas são as instruções que dividem números ou fórmulas por 100. Se o teste é Falso isto indica que a célula não contém uma entrada numérica. Nestes casos, a macro pula o bloco de declarações para a declaração **End IF** que é o fim do teste **If** de **IsNumeric**. Como a próxima declaração depois deste **End IF** é **End Sub**, a macro pára.

Também note que nós pusemos declarações de comentário em cada uma das linhas de declaração **End If**. Sempre que você tem muitas declarações **If** numa macro é melhor documentar as declarações **End If** com um comentário que lhe conte com qual declaração **If** elas estão associadas. Como ilustrado anteriormente, isto é muito fácil fazer. Por favor, note que isso é só porque você rotula **End If** com uma declaração de comentário que diz que é o fim de uma particular declaração Se, isto não funciona assim. Declarações de comentário são apenas aquelas de comentários. Quando você escrever macro mais complexa, você poderá achar que não tem mais que atarraxar às suas declarações **If** os comentários.

Quando nós testamos nossa macro em uma célula que contém texto, como a palavra " Fórmula " na nossa planilha de teste, ela funciona exatamente como nós queremos- ela não faz nada. Isto é porque **IsNumeric**(ActiveCell.Value) retorna o valor Falso quando uma célula contém texto. Isto faz nossa macro pular as declarações

que modificam a célula. Quando nós testarmos em células que contêm um número ou uma fórmula, ela também funciona perfeitamente, modificando estas células a serem divididas por 100. Mas quando nós testamos o macro em uma célula em branco obtemos umas outras das caixas de erro infames do Visual Basic:



Por que isto aconteceu? O teste **IsNumeric**(`ActiveCell.Value`) em uma célula em branco deve ter retornado Verdadeiro (True), não Falso (False). Você pode usar as técnicas de depurar ilustradas anteriormente para confirmar isto. O **IsNumeric** retorna True numa célula em branco porque uma célula em branco em Microsoft Excel é tratada como sendo igual a zero. E desde que zero é um número, o teste **IsNumeric** retorna True. Para provar isto, vá para a planilha de teste e entre com a fórmula seguinte na célula `B12:"=B5*B10".` A célula `B5` contém um número, e célula `B10` é uma célula em branco. O resultado é zero, que é consistente com o tratar uma célula em branco como um zero.

Assim, nós também precisamos de um teste para ver se uma célula está em branco ou não. Vá para o módulo Visual Basic, selecione Exibir, e daí Pesquisador de Objetos. A opção << todas as bibliotecas >> deverá ser selecionada, e Information selecionada no painel Classes. Olhe as várias propriedades. Você achará uma propriedade chamada **IsEmpty**. Selecione-a e dê um clique no ponto de interrogação para descobrir mais sobre ela. Você achará que o teste **IsEmpty** retorna True se a célula está vazia.

Considerando que nós queremos somente rodar as declarações que modificam os conteúdos de células quando a célula não estiver vazia, nós precisamos "sacudir" a resposta que nós queremos. O Microsoft Excel não faz isto para nós. Por exemplo, o teste **Not** retorna True se o teste é falso, e Falso se o teste é verdadeiro. A declaração seguinte na tabela ilustra isto:

	IsEmpty(ActiveCell)	Not(IsEmpty(ActiveCell))
Célula Contém	Retorna	Retorna
Entrada Qualquer	False	True
Nenhuma Entrada	True	False

O uso dos parênteses com a função **Not** é opcional.

Baseado nisto modificamos a macro para ler:

```
Sub DividirPor100()
' Esta macro divide por 100
  If Not IsEmpty(ActiveCell) then
    'Faça isto se a célula não estiver vazia
    If IsNumeric(ActiveCell.Value) Then
      'Faça isto desde que a célula seja numérica
      If Left(ActiveCell.Formula, 1) = "=" Then
        'Faça isto desde que a célula contenha uma equação
        ActiveCell.Formula = _
                "=(" & Mid(ActiveCell.Formula, 2) & ")/100"
      Else
        'Faça isto desde que a célula contenha um número
        ActiveCell.Formula = _
                    "=" & ActiveCell.Formula & "/100"
      End If     'Fim do teste If Left
    End If     'Fim do teste IsNumeric
  End If     'Fim do teste Not IsEmpty
End Sub
```

Quando testarmos as modificações acima ela funciona para todos os casos!

DIVIDINDO TODAS AS CÉLULAS DE UMA SELEÇÃO POR 100

Até este estágio, a nossa macro funcionou somente numa **única** célula por vez. Se tivermos 50 células para processar, teríamos que rodar a macro 50 vezes. Seria muito melhor selecionar um intervalo contendo as células que queremos dividir por 100 e rodar a macro somente uma vez.

É necessário então declarações do Visual Basic que **circule** por todas as células de nossa seleção e rode as declarações em cada uma delas. A declaração **For Each...Next** do Visual Basic faz isto, e tem a seguinte estrutura:

```
For Each  item  In  uma coleção
      declarações
Next
```

Este comando também será coberto em detalhes em lições posteriores.

A coleção que nós desejamos processar são as células no intervalo que nós realçamos. As *células selecionadas* na planilha ativa são identificadas pela de palavra chave Selection do Visual Basic. O comando **For Each** circulará por cada uma das células que foram selecionadas e rodará as declarações que estão entre o **For Each** e o **Next**. A declaração **Next** diz ao Visual Basic para voltar ao topo do laço (loop), onde a declaração **For** está, e processar a próxima célula. Isto continua até todas as células serem processadas.

A célula que está sendo atuada é identificada por qualquer nome de variável que nós usarmos depois das palavras **For Each**. O nome que eu prefiro usar é o nome "célula", em minúscula.

O que segue é a estrutura básica para uma macro que faz um laço através de uma seleção de células e executa uma série de declarações nelas:

```
For Each célula In Selection
        declarações
Next célula
```

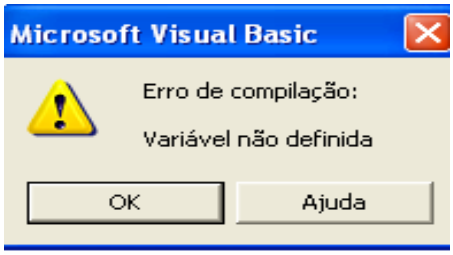
Mudemos nossa macro para usar as declarações anteriores de forma que ela atuarão nas células múltiplas. Sua macro deveria se parecer com o que se segue. (as únicas mudanças são a terceira linha do topo e o segunda linha de baixo para cima).

```
Sub DividirPor100()
    'Esta macro divide por 100
    For Each célula In Selection
    If Not IsEmpty(ActiveCell) Then
        'Faça isto se a célula não está vazia
        If IsNumeric(ActiveCell.Value) Then
            'Faça isto desde que a célula seja numérica
            If Left(ActiveCell.Formula, 1) = "=" Then
                'Faça isto desde que a célula contenha uma equação
                ActiveCell.Formula = _
                "=(" & Mid(ActiveCell.Formula, 2) & ")/100"
             Else
                 'Faça isto desde que a célula contenha um número
                 ActiveCell.Formula = _
                  "=" & ActiveCell.Formula & "/100"
            End If        'Fim do teste If Left
        End If      'Fim do teste IsNumeric
    End If     'Fim do teste Not IsEmpty
    Next célula
End Sub
```

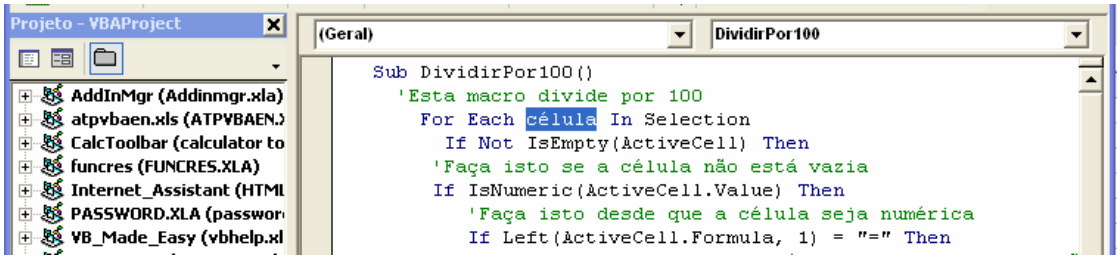
Agora vá testar a planilha e selecione B5 até D6:

	A	B	C	D	E
2					
3		Números		Fórmulas	
4					
5		1234		1234	
6		5678		6912	
7					

Agora rode a macro com todas estas células selecionadas. Em vez da macro funcionar, a seguinte mensagem de erro aparece:

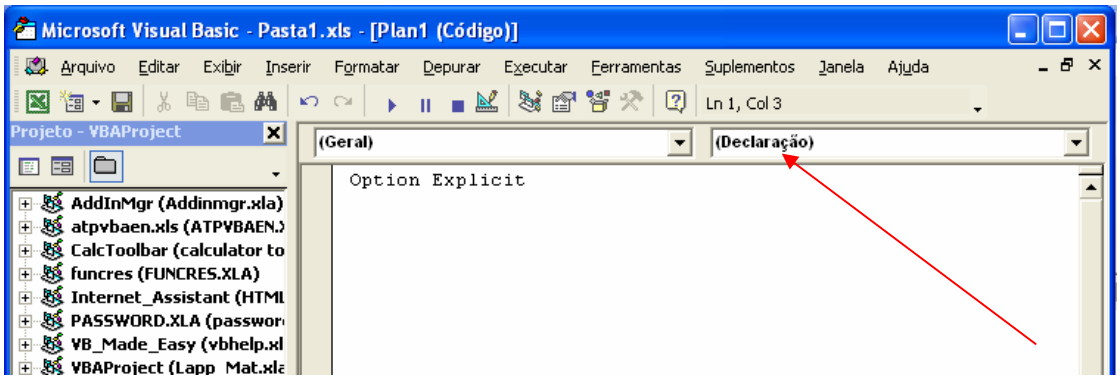


E a palavra "`célula`" na linha "**For Each** `célula` **In** `Selection`" é destacada.



O que aconteceu é que você não declarou no topo da macro que a variável `célula` seria usada na macro. E, desde que a declaração **Option Explicit** está no topo do módulo, o Visual Basic pára a macro, mostra a mensagem de erro, e destaca a variável não declarada.

Se isto não aconteceu, clique no menu *pop up* do VB onde está atualmente *DividirPor100* e acesse o item (Declaração). Se a declaração **Option Explicit** não estiver lá, aquela mensagem não aparecerá e a macro encontrará valores absurdos como resultado.



Declarar uma variável é muito fácil. Adicione apenas a declaração

Dim *nome da variável*

No topo de uma macro. Neste caso,

Dim célula

Seria a declaração para declarar a variável "célula". Dim é abreviação da palavra inglesa *Dimension*, que era usada nas linguagens antigas de programação.

A macro ficaria agora com a seguinte aparência:

```
Sub DividirPor100()
 ' Esta macro divide por 100
 Dim célula
 'restante do código da macro
End Sub
```

Depois que você fizer a correção anterior, volte à planilha de teste, realce as células B5 até D6 e rode a macro. Você notará que só a fórmula em B5 muda. As fórmulas nas outras células não mudam, embora os valores exibidos mudem devido à mudança no valor de B5. Há uma razão simples para isto. Embora a declaração **For Each** está selecionando cada célula no intervalo de seleção, a macro só está agindo na ActiveCell. Há só uma célula ativa em uma seleção. Neste caso que célula é B5. A declaração **For Each** não muda a célula ativa quando ela circula pelas células. Para consertar, modifique a macro mudando cada ocorrência de ActiveCell à palavra "Cell ". A macro corrigida se parece agora com o seguinte:

```
Sub DividirPor100()
  'Esta macro divide por 100
   Dim célula
    For Each célula In Selection
     If Not IsEmpty(célula) Then
     'Faça isto se a célula não está vazia
     If IsNumeric(célula.Value) Then
       'Faça isto desde que a célula seja numérica
       If Left(célula.Formula, 1) = "=" Then
         'Faça isto desde que a célula contenha uma equação
         célula.Formula = _
         "=(" & Mid(célula.Formula, 2) & ")/100"
       Else
         'Faça isto desde que a célula contenha um número
         célula.Formula = _
                 "=" & célula.Formula & "/100"
       End If     'Fim do teste If Left
     End If   'Fim do teste IsNumeric
   End If   'Fim do teste Not IsEmpty
   Next célula
  End Sub
```

Quando você tentar esta macro, você verá que toda célula numérica na seleção é dividida por 100 e as células não-numéricas não são mudadas. Nesta fase você tem uma ferramenta bastante poderosa. Também, por favor, note que a macro anterior não

seleciona nenhuma célula nem muda a célula ativa. Esta aproximação é aquela que você deveria tentar seguir em todas as suas macros.

TORNANDO SUA MACRO SUPER PODEROSA

Realmente seria agradável se a macro que há pouco você escreveu pudesse fazer muito mais que dividir por 100. Pode haver casos onde você quer dividir por 10, 1000, ou algum outro número. Ou, você pode querer fazer multiplicação, adição, ou subtração em vez de divisão. Um modo para resolver isto seria fazer cópias deste macro e modificar o que ela faz. Então, você terá uma macro para cada tarefa que você quer. Isto, obviamente resultaria em muitas macros fazendo tarefas semelhantes. Outra aproximação seria editar a macro sempre que você quisesse fazer algo diferente. Mas isso na verdade não é muito eficiente, e perigoso, pois se você pensar que a macro que foi ajustada para fazer uma coisa é agora ajustada para fazer algo diferente de fato. Isto é especialmente importante, pois não existe capacidade de se desfazer (undo) com macros.

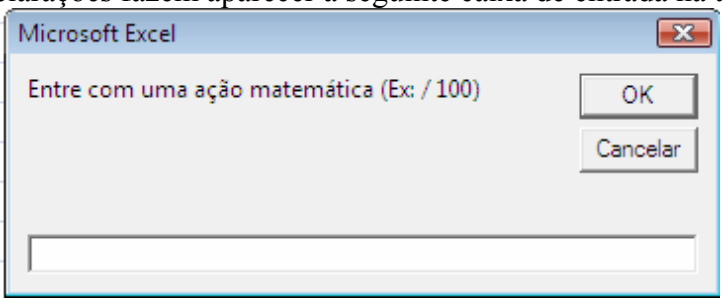
Felizmente, o Visual Basic fornece vários significados para incitar a contribuição do usuário. O mais simples destes é a função **InputBox** (). Esta função permite uma macro mostrar uma caixa de entrada onde o usuário pode digitar em resposta à solicitação.

O que segue são as declarações que precisam ser acrescentadas ao topo da nossa macro (direitamente abaixo da declaração "**Dim** `célula`):

```
Dim açãodesejada
 açãodesejada = _
     InputBox("Entre com uma ação matemática (Ex: / 100)")
If açãodesejada = "" Then Exit Sub
```

A variável `açãodesejada` é uma nova variável de usuário e a primeira letra não é maiúscula. Esta é uma convenção que freqüentemente é usada para nomear variáveis.

Estas declarações fazem aparecer a seguinte caixa de entrada na tela:



Quando estas declarações são rodadas, a entrada na caixa é atribuída à variável de usuário `açãodesejada`. Se o botão Cancelar é selecionado, então a `açãodesejada` é fixada igual a "". A variável `açãodesejada` também é fixada para

"" se nada é posto na caixa de entrada e o botão de OK é selecionado. Se qualquer uma destas duas situações acontecer, então a macro precisa conferir isto e parar. Isso é exatamente o que a declaração **If** `açãodesejada` = "" **Then Exit Sub** faz.

Se o usuário entra com uma ação de matemática, por exemplo "/ 100", ou "* 5", então a variável `açãodesejada` é fixada para este valor. Então, quando nós referenciarmos a `açãodesejada` mais tarde em nossa macro, o Visual Basic substitui o que está armazenado na variável `açãodesejada` para o nome da variável.

A outra modificação que você precisa fazer em sua macro é substituir o " / 100 " com a variável `açãodesejada`. Há duas ocorrências que precisam ser substituídas - uma para o caso da célula conter exatamente um número, e um para o caso em que a célula contém uma fórmula. Naquele contendo a fórmula, os parênteses fechados precisam ser mantidos.

O que segue é o que a macro modificada agora se parece. Note que a variável `açãodesejada` não está inclusa em aspas duplas. Também, o nome da macro foi mudado para *AçõesMatemáticas* para indicar melhor o que o macro faz.

```
Sub AçõesMatemáticas()
  ' Esta macro modifica células baseadas em ações matemáticas entradas pelo usuário
  Dim célula
  Dim açãodesejada
      açãodesejada = _
      InputBox("Entrar com uma ação matemática (Ex: / 100)")
  If açãodesejada = "" Then Exit Sub
  For Each célula In Selection
  If Not IsEmpty(célula) Then
     'Faça isto se a célula não estiver vazia
     If IsNumeric(célula.Value) Then
      'Faça isto desde que a célula seja numérica
      If Left(célula.Formula, 1) = "=" Then
        'Faça isto desde que a célula contenha uma equação
        célula.Formula = _
                      "=(" & Mid(célula.Formula, 2) & _
                       ")" & açãodesejada
      Else
        'Faça isto desde que a célula contenha um número
        célula.Formula = _
                    "=" & célula.Formula & açãodesejada
      End If     'Fim do teste If Left
     End If    'Fim do teste IsNumeric
    End If   'Fim do teste Not IsEmpty
  Next célula
End Sub
```

Se você mudar a capitalização da variável `açãodesejada` em seu comando declaração de variáveis (**Dim** `açãodesejada`), a capitalização muda em todos lugares na macro quando você se mover para fora da linha **Dim**. Mude a capitalização e mova-se para fora da linha para ver isto acontecer.

Se você não salvou o seu arquivo, por favor, faça-o então neste momento! Você deveria adquirir o hábito de salvar muito freqüentemente o seu trabalho.

ANTES DE VOCÊ TESTAR, você precisa mudar o macro atribuindo-a a seu botão da barra de ferramentas. Se você não fizer a seguinte caixa de erro vai estourar na sua frente desde que o nome de *DividirPor100* foi mudado para *AçõesMatemáticas*.

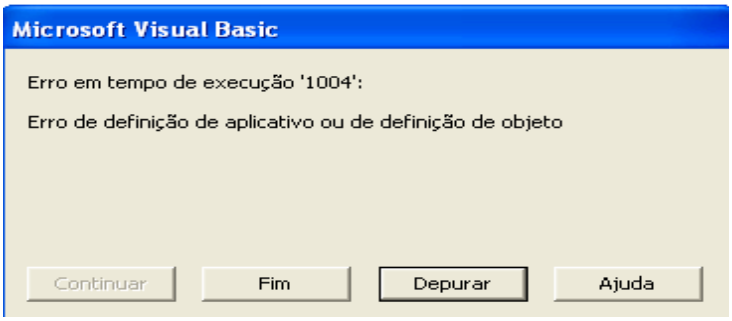
Tente seu botão - o que está acima aparecerá! Para consertar faça o seguinte:

- Selecione Exibir, Barra de ferramentas, Personalizar
- Clique com o botão direito do mouse sobre o botão de sua macro. Selecione 'Atribuir Macro' do painel que aparece.
- Selecione a macro "*AçõesMatemáticas*" e feche todas as caixas.

Uma vez tendo feito as mudanças anteriores, você pode voltar à planilha de teste e testar a macro nova melhorada. Você a achará bastante poderosa. Você não só pode entrar com ações de matemática como " / 100 " ou " * 5 ", como também você pode entrar com ações complexas. Por exemplo, você poderá entrar com "*100 - 456".

MANIPULANDO ERROS

Se você entrar com uma ação de matemática complexa incorreta (Ex: "*/*456"), sua macro explodirá com a mensagem de erro seguinte:



Também explodirá se você tentar modificar uma célula que contém uma fórmula e esquecer de pôr nela o operador de matemática (Exemplo: 1000 em vez de em * 1000).

Seria muito melhor se exibisse uma mensagem que mostrasse a ação de matemática que fora entrada nela e uma mensagem que diga que o Microsoft Excel não pode modificar células que usam aquela entrada de ação de matemática. Para fazer isto, duas mudanças têm que ser feitas na macro. Primeiro, adicione a linha seguinte logo acima da linha **For Each**.

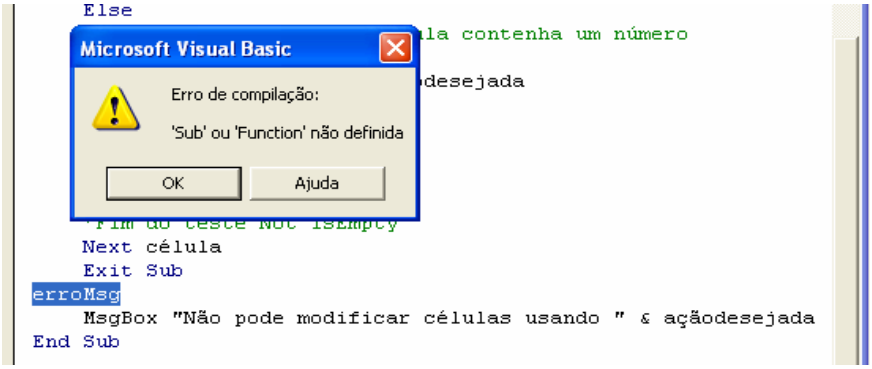
```
On Error GoTo erroMsg
```

E, coloque a seguinte declaração após a declaração "**Next** `célula`".

```
Exit Sub
erroMsg:
  MsgBox "Não pode modificar células usando " & açãodesejada
Exit Sub
```

A frase **On Error** é uma declaração do Visual Basic que monitora sua macro por erros. Ela é chamada de *armadilha de erros*. Quando um deles ocorrer, a declaração que segue **On Error** é imediatamente executada. Neste caso a declaração é "**GoTo** `erroMsg`". **GoTo** é uma declaração do Visual Basic que transfere o controle para a localização especificada após a palavra **GoTo**. Neste caso, o Visual Basic procurará um rótulo (label) na macro chamado "`erroMsg:`". Um rótulo consiste de um nome seguido por dois pontos. Assim, dois pontos são exigidos no final de "`erroMsg`" no segundo grupo de declarações que estão sendo adicionadas.

Se você esquecer de colocar os dois pontos após o nome de rótulo , você obterá a seguinte mensagem de erro quando você rodar a macro:



Nenhum erro ocorrerá se você colocar os dois pontos após o nome de rótulo quando você se referir a ele numa declaração **GoTo**.

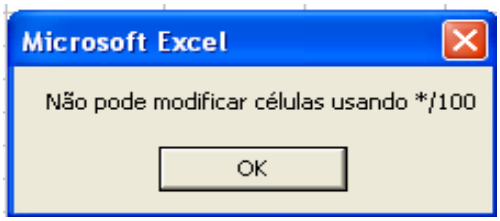
A primeira das duas declarações **Exit Sub** pára a macro se nenhuma mensagem de erro ocorrer. A segunda **Exit Sub** pára a macro se ela saltar o rótulo `erroMsg:`. A macro será sempre parada pela linha **End Sub**, você deverá também colocar uma linha **Exit Sub** após rotinas de erros. Esta é exatamente uma boa técnica de programação.
MsgBox é uma função do Visual Basic que mostra mensagens na tela. O texto a ser mostrado deve estar entre aspas duplas. Se você estiver combinando texto e variáveis, então eles são concatenados usando um símbolo &. Note que espaços são exigidos de cada lado do símbolo &. Caso contrário um erro poderá ocorrer. Também, nomes de variáveis não ficam entre aspas duplas.

O que segue é o que a macro modificada se parece:

```
Sub AçõesMatemáticas ()
     'Esta macro modifica as células baseada em uma ação
matemática que o usuário entra

Dim célula
Dim açãodesejada

'Obtenha uma entrada do usuário
açãodesejada = _
     InputBox("Entrar com uma ação matemática (Ex: / 100)")
If açãodesejada = "" Then Exit Sub

On Error GoTo erroMsg

For Each célula In Selection
  If Not IsEmpty(célula) Then
  'Faça isto se a célula não estiver vazia
    If IsNumeric(célula.Value) Then
      'Faça isto desde que a célula seja numérica
      If Left(célula.Formula, 1) = "=" Then
```

```
        'Faça isto desde que a célula contenha uma equação
        célula.Formula = _
                "=(" & Mid(célula.Formula, 2) & _
                 ")" & açãodesejada
      Else
        'Faça isto desde que a célula contenha um número
        célula.Formula = _
                 "=" & célula.Formula & açãodesejada
      End If     'Fim do teste If do sinal de igual
    End If     'Fim do teste IsNumeric
  End If    'Fim do teste Not IsEmpty
Next célula

Exit Sub

erroMsg:
  MsgBox "Não pode modificar células usando " & açãodesejada
Exit Sub

End Sub
```

Vá agora testar a planilha e testar a macro. Desta vez, coloque a entrada "* / 100" na caixa de entrada ao invés de "/ 100". Quando você fizer isso, a seguinte caixa de mensagem aparecerá:



Finalmente, volte e teste o macro em todos os tipos diferentes de células. Você achará que ela funciona exatamente como nós queríamos, com uma exceção: Se uma célula contém um número e você entra em um número sem um operador de matemática (+. -. *, /), a macro combinará os dois números em um: Por exemplo, se a célula que é modificada contém o número 1234 e você entra em 100 (não /100) na caixa de entrada, você obtém 1234100. Não é totalmente a resposta certa. Para resolver este problema, nós precisamos testar a entrada na caixa de entrada e ver se o primeiro caráter é um operador de matemática. O código a seguir realiza esta tarefa:

```
Private Sub ChecarPrimeiroCaract(entradaParaChecar)
        Dim primeiroCaract
        'checar texto para ver se a entrada é um operador matemático

primeiroCaract = Left(LTrim(entradaParaChecar), 1)
If primeiroCaract = "+" Or _
  primeiroCaract = "-" Or _
```

```
  primeiroCaract = "*" Or _
  primeiroCaract = "/" Or _
  primeiroCaract = "^" Then
  Exit Sub
Else
  'Mostrar mensagem e parar a macro
  MsgBox "Sua entrada não começa " & _
            "com +, -, * ou /.  Atividade interrompida."
  End
End If
End Sub
```

O anterior é de fato uma macro separada que será chamada pela macro *AçõesMatemáticas*. Uma macro que é chamada por outra macro é chamada **sub-rotina**.

Pondo a palavra chave **Private** na frente da declaração de **Sub** desta macro `ChecarPrimeiroCaract`, esta não se apresentará quando você selecionar Ferramentas, Macro dos menus. Isto normalmente é feito em sub-rotinas, pois elas somente são rodadas quando forem chamadas por uma macro.

Para chamar a sub-rotina anterior, insira a seguinte declaração na macro de *AçõesMatemáticas* imediatamente acima da declaração On Error:

```
ChecarPrimeiroCaract açãodesejada
```

Esta declaração diz ao Visual Basic para ir à macro chamada `ChecarPrimeiroCaract` e passar a ela o valor da variável `açãodesejada`. A macro `ChecarPrimeiroCaract` guarda o valor que é passado pra ela numa variável chamada `EntradaParaChecar`. A macro `ChecarPrimeiroCaract` usa então a função **LTrim** e a função **Left** para encontrar qualquer branco na frente do texto armazenado na variável `EntradaParaChecar` e obter o primeiro caracteree. A macro então verifica se aquele primeiro caracteree é um símbolo +, -, *, / ou ^ . Se for, o controle é retornado à macro *AçõesMatemáticas*. Se não for, uma caixa de mensagem aparece dizendo que a entrada está incorreta. Ela então usa a declaração **End** que por si só pára toda a atividade de macro. Se uma declaração **Exit Sub** tivese sido usada na sub-rotina em vez da declaração **End**, o controle teria retornado para a *AçõesMatemáticas*, e a macro *AçõesMatemáticas* teria continuado.

O que se segue são as macros finais. Note que os comentários foram adicionados para facilitar a documentação das macros:

```
Sub AçõesMatemáticas()
'Esta macro modifica as células baseadas nas ações matemáticas
entradas pelo 'usuário
Dim célula
Dim açãodesejada
'Obtenha a ação matemática que o usuário quer introduzir
```

```
açãodesejada = _
    InputBox("Entrar com ação matemática (Ex: / 100)")
'Encerrar se nenhuma ação for entrada o se selecionar Cancelar
If açãodesejada = "" then Exit Sub
'chamar a sub-rotina para confirmar que o primeiro caracteree é
um operador 'matemático
ChecarPrimeiroCaract açãodesejada
'verificar se na ação introduzida  se ocorreu um erro
On Error GoTo ErrorMsg
'circule através de cada célula na seleção
For Each célula In Selection
'faça as ações seguintes se a célula não estiver vazia
If Not IsEmpty(célula) then
  'Faça isto se a célula não estiver vazia
 If IsNumeric(célula.Value) Then
  'Faça isto desde que a célula seja numérica
  If Left(célula.Formula, 1) = "=" Then
   'Faça isto desde que a célula contenha uma equação
   célula.Formula = _
       "=(" & Mid(célula.Formula, 2) & _
          ")" & açãodesejada
  Else
       'Faça isto desde que a célula contenha um número
   célula.Formula = _
        "=" & célula.Formula & açãodesejada
  End If   'Fim do teste de sinal de igual
 End If   'Fim do teste IsNumeric
End If  'Fim do teste Not IsEmpty
'retorne no círculo e faça na próxima célula
Next célula
'terminar a macro quando todas as células forem processadas
Exit Sub
'venha aqui se existir um erro quando a formula é escrita errada
ErrorMsg:
    MsgBox "Não se pode modificar a célula usando" & açãodesejada
  Exit Sub
End Sub

'**************************************************

Private Sub ChecarPrimeiroCaract(entradaParaChecar)
'esta sub-rotina chamada pela AçõesMatemáticas para confirmar se
um operador 'matemático é o primeiro
'caractere no texto que é entrado.
Dim primeiroCaract
'checar o texto para ver se a entrada é um operador matemático
primeiroCaract = Left(LTrim(entradaParaChecar), 1)
If primeiroCaract = "+" Or _
  primeiroCaract = "-" Or _
```

```
    primeiroCaract = "*" Or _
    primeiroCaract = "/" Or _
    primeiroCaract = "^" Then
  Exit Sub        'retorna à macro que a chamou
Else
  'Mostra uma mensagem e pára a macro
     MsgBox "Sua entrada não começa " & _
       "com +, -, * ou /.  Atividade parada"
   End
End If

End Sub

'*************************************************
```

RESUMO DA LIÇÃO

Nesta lição, você criou não só uma macro muito poderosa e útil, você também aprendeu bastante a respeito do Visual Basic e de como escrever macros. Por exemplo, você sabe descobrir o que está em uma célula agora e como mudá-la. Mais adiante, você viu como escrever uma macro que trabalhará em todas as células que você selecionou e não em só numa única célula. E, você viu como perguntar ao usuário pelas entradas e exibir mensagens de volta a ele.

Se você só leu esta lição e não fez os exercícios, você precisa voltar e fazer os exercícios. Só fazendo você irá aprendê-los verdadeiramente.

Finalmente, você encontrará algumas técnicas e comandos adicionais mais tarde que você poderá usar para agilizar e melhorar a macro anterior. Prossiga e faça isto. A macro anterior foi escrita de modo a ilustrar as declarações do Visual Basic ao invés de ser um conjunto bem eficiente de declarações para fazer a tarefa. Por exemplo, você pode querer substituir o InputBox com um userform. Ou, você pode querer modificar a macro de forma que ela se lembre da última entrada.

OUTRO EXEMPLO

Vamos agora criar uma outra macro mais simples. Abra o Excel, abra a pasta "VBATeste1.xls" aquela que você criou na lição.... . Adicione uma folha de planilha à sua pasta e renomeie-a "Teste2" (clicar com o botão direito do mouse na guia, escolher "Renomear" e digitar nela o nome). Na célula A1 a A10 entrar com os 10 primeiros nomes de algumas pessoas e nas células B1 até B10 entrar com os 10 últimos nomes daquelas pessoas.

Vamos agora criar um novo módulo. Vá para o VBE e clique com o botão direito do mouse no **projeto** "VBATeste1" dentro da Janela VBA project e selecione "Inserir/Módulo". Você agora tem um novo módulo. Na janela de propriedades, dê um duplo clique na caixa da propriedade "(Name)" e entre com "modTeste2". Temos agora um novo módulo com um novo nome.

Na janela VBAProject dê um clique na folha de planilha chamada "Teste2". Na janela de propriedades dê um duplo clique na caixa de propriedade "(Name)" e entre com "folTeste2". Você agora tem uma folha de planilha com um *caption* lendo "Teste2" e com o nome "folTeste2".

Na janela VBAProject dê um duplo clique sobre "modTeste2"" e vá para a janela de código para desenvolver uma macro.

Escreva "Sub proTeste2 ()" e clique "Enter". O VBE adiciona uma linha "End Sub". Nós escreveremos nosso código entre estas duas linhas, assim crie alguma coisa para ela definindo o cursor no final de "End Sub" e clicando "Enter" umas poucas vezes.

Logo abaixo de "Sub proTeste2()" escreva seu nome precedido por uma apóstrofe (') ' Luiz Bertolo e clique "Enter". Note que a fonte na linha que você acabou de escrever é verde. Por causa da apóstrofe o VBA considerará o texto como um comentário (remark -REM) e não se incomodará com ele. Você pode adicionar um número qualquer de linhas REM em qualquer lugar para tornar o seu código inteligível. Eu geralmente sempre começo meus procedimentos com três linhas Rem (meu nome, meu número de telefone e o endereço do meu website) de modo que as pessoas possam entrar em contato comigo se elas tiverem problemas com meus procedimentos VBA.

Clique "Enter" novamente par inserir uma vazia linha. Não hesite em inserir linhas vazias em qualquer lugar para tornar seu código mais facilmente legível.

Aqui estão as 6 linhas de código que você estará escrevendo (ou copiar/colar daqui):

```
  folTeste2.Select
    Range("C1").Select
    Do Until Selection.Offset(0, -2).Value = ""
        Selection.Value = Selection.Offset(0, -2).Value & " " &
Selection.Offset(0, -1)
        Selection.Offset(1, 0).Select
    Loop
    Range("A1").Select
```

As primeiras 2 linhas dizem ao VBA onde começar o procedimento. Se você não mencioná-las o VBA começará o procedimento de qualquer célula que estiver selecionada na sua pasta.

As linhas 5 até 7 é a tarefa real que você quer que o VBA realize. Você está dizendo: de alguma coisa para o valor da célula 2 colunas à esquerda da célula

selecionada em vazio (aspas duplas). Com a célula selecionada uma linha após onde estava por último a tarefa encerrará. Se houver um primeiro nome você faz outra vez seu loop (última linha da declaração).

Qual é a tarefa: torne o valor da célula selecionada igual ao valor da célula a 2 colunas à esquerda, (&) um espaço (espaço entre duas aspas duplas) (" "), e o valor da célula a 1 coluna à esquerda da célula selecionada. Daí mova-se para baixo uma célula. Quando a tarefa tiver sido efetuada vá para a célula A1.

Agora volte ao Excel vá para "Ferramentas/Macro/Macros" selecione a macro "proTeste2" e clique "Executar". Apague a coluna **C** e faça isto novamente. Adicione o primeiro e o último nome de pessoas à sua lista e faça isto novamente.

Congratulações por você ter criado o seu primeiro procedimento VBA.
Nós o testaremos passo a passo na próxima lição.

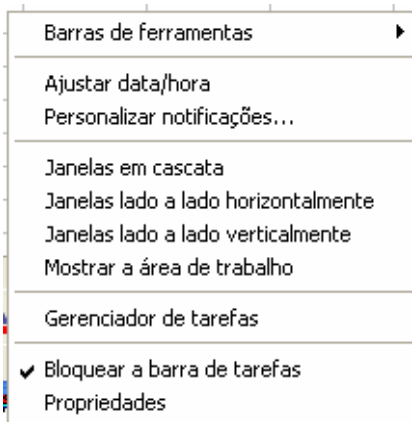
PRÁTICA

Lição 7: Testando Macros no Visual Basic Editor para Excel

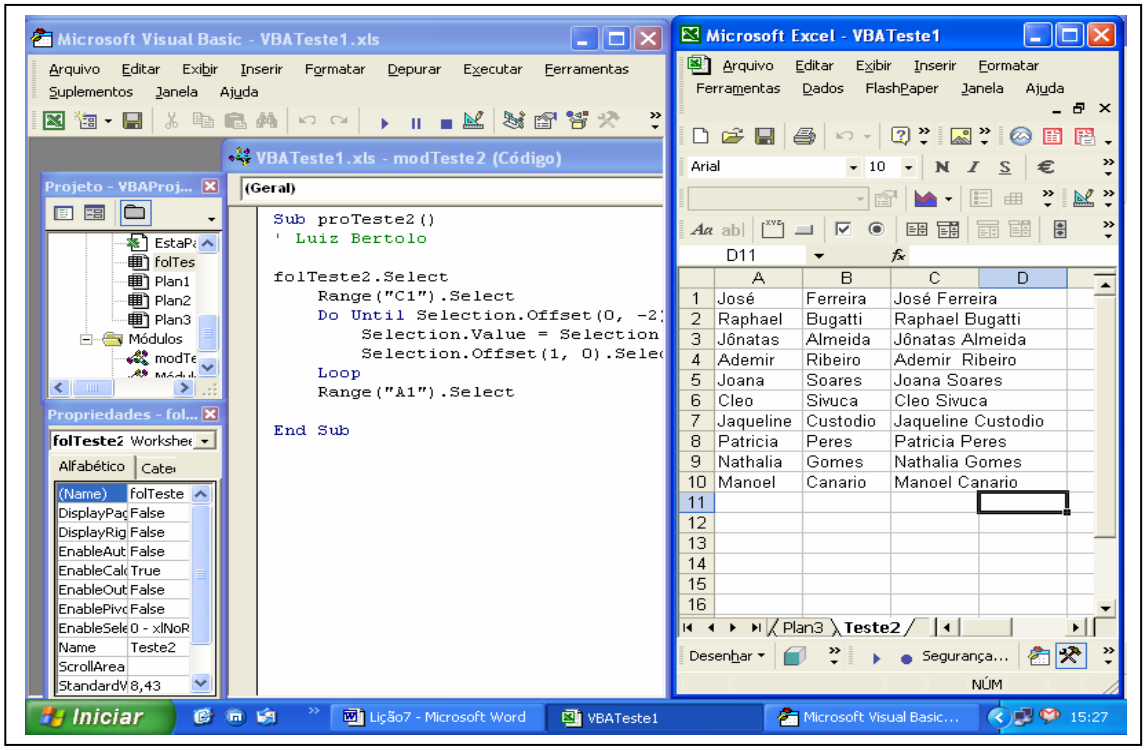
Aqui está uma maneira muito interessante de testar uma macro passo a passo enquanto você a acompanha trabalhando na pasta Excel.

Feche cada um dos programas do seu computador. Abra o Excel e a pasta na qual você criou a macro. Abra o Visual Basic editor. Na barra de status do Window no fundo de sua tela você pode ver que o Excel está aberto e o VBE também. Clique com o botão direito do mouse na barra de *status* no espaço vazio onde onde eu adicionei as estrelas laranjas:

O menu seguinte aparece:

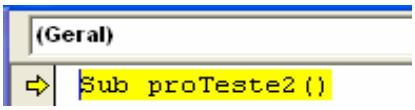


Clique em "Janelas lado a lado verticalmente" e a tela se parecerá com isto:



A pasta Excel ocupa metade da tela e o VBE ocupa a outra metade. Torne a Janela de Código mais larga no VBE de modo que você possa ver todo o código.

Clique em qualquer lugar dentro da macro e pressione a tecla F8 no topo do seu teclado. A primeira linha de código torna-se amarelo e a pequena seta aparece na margem.



Clique novamente em F8 até toda a macro ter sido executada. Apague os valores da coluna B do Excel e inicie a macro novamente. Pare quando a linha "Do" for destacada. Mude "B1 para B5" na linha acima. Clique na pequena seta, segure e arraste a seta de volta para a nova linha "Range("B5").Select". Pressione F8 novamente. Você vê que se você modificar o código e pode voltar um passo sempre você quiser. Você pode mesmo pular certas linhas de código e testar somente parte do procedimento.

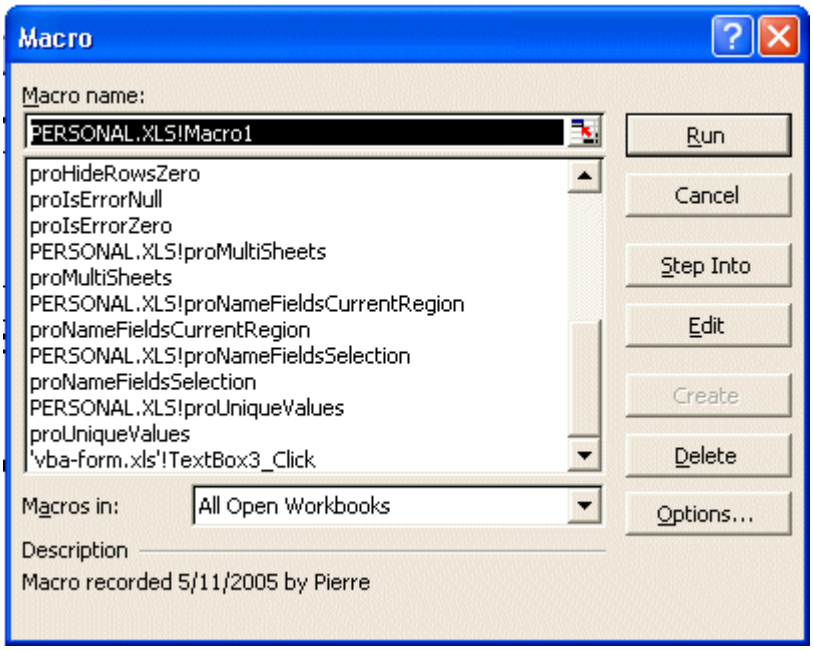
Enquanto você estiver rodando a macro passo a passo você pode parar a execução em qualquer instante clicando no botão parar na barra de ferramentas. 

Lição 8: Eventos VBA No Excel

Quando a macro inicia? Quando um EVENTO acontece. O evento é aquilo que dispara a macro VBA do Excel.

Do Menu

No Excel você pode rodar uma macro VBA do Excel indo ao menu "Ferramentasl/Macro/Macros.." daí selecionar a macro na lista e clicar "Executar".



Abra muitas pastas com macros nelas. Quando você for ao menu "Ferramentas/Macro/Macros.." você notará que você tem acesso a todas as macros de todas as pastas abertas. Isto significa que você pode armazenar TODAS as suas macros úteis do Excel numa única pasta (chame-a myMacros.xls) e tenha acesso a elas enquanto a pasta estiver aberta. Digamos por exemplo que você tenha projetado a macro que multiplica o conteúdo de uma célula por 2. Se "myMacros.xls" está aberta você pode chamar esta macro Excel de qualquer célula de qualquer outra pasta que esteja aberta. Não precisa copiar suas macros essenciais em todas as suas pastas apenas abra myMacros.xls e coloque-as para trabalhar.

Clicando numa Tecla do seu Teclado

Primeiro você precisa programar uma tecla. Para fazer isto vá a "Ferramentas/Macro/Macros.." daí selecione uma macro da lista. Clique nas "Options" e siga as instruções. Uma sugestão, atribua às suas macros teclas maiúsculas ("Shift/A" em vez de "Shift/a por exemplo) para garantir-se de que você não usa uma das muitas teclas minúsculas que já são usadas pelo Excel.

Clicando numa caixa de texto na planilha

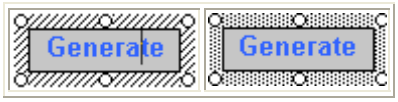
80% das macros Excel que eu desenvolvi são disparadas por um clique numa caixa de texto localizada sobre uma planilha.

Eu prefiro usar caixas de texto ao invés de botões de comando VBA porque elas são muito mais fáceis de manter e permitir muito mais criatividade no design. Você pode usar the fonte que você gosta e a cor de fundo (background) que ajuste às suas necessidades. Se você for um pouco criativo, você pode adicionar efeitos 3D, bordas especiais e as preferências.

Umas poucas notas sobre as caixas de texto do Excel:

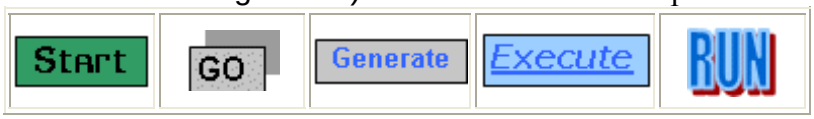
Eu mantenho sempre a barra de ferramentas de desenho visível no fundo da minha tela

Você cria caixas de texto clicando com o botão esquerdo do mouse no ícone  , daí vá à planilha clique com o botão esquerdo do mouse, segurar e esticar a caixa de texto. Quando a borda da caixa de texto ativa for feita de linhas em diagonal você pode trabalhar o texto dentro da caixa de texto. Se você clicar novamente na borda ela torna-se um conjunto de pontos e você pode então trabalhar a própria caixa de texto. Dê um clique com o botão direito do mouse na borda em qualquer dos dois estados e você verá que os menus são diferentes. É no segundo estado que você pode atribuir uma macro à caixa de texto.



Você pode atribuir uma macro VBA para uma caixa de texto e também para um WordArt, uma figura ou qualquer outra forma da barra de ferramentas de "Desenho". Sempre que a um botão (imagem, a palavra arte ou caixa de texto) tiver sido atribuida uma macro ou um *hyperlink* você precisa selecioná-lo com um clique com o botão direito do mouse para modificá-lo.

Download um destes botões (clicar com o botão direito do mouse nele no seu browser e escolher "Save imagem as"). Salve-os no seu desktop:



Inserir a imagem que você tiver importado na primeira folha de planilha "Inserir/Figura/Do Arquivo/Desktop/.....gif". Uma vez a imagem tendo sido adicionada à folha de planilha, clicar com o botão direito do mouse na imagem, selecione "Atribuir Macro" e selecione uma macro da lista. Clique "OK".

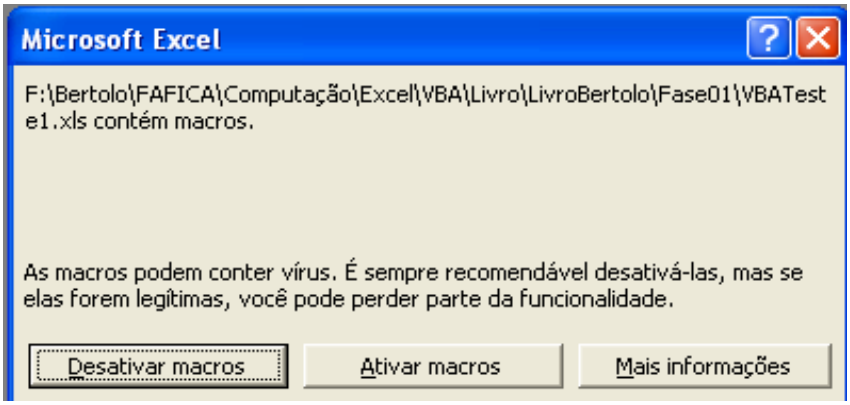
Agora clique na imagem.

Você pode "copiar" todas espécies de botões da Internet ou criar seu próprio da barra de ferramentas "Design" e usá-los para disparar seus procedimentos VBA.

Lição 9: Segurança e Proteção VBA no Excel

Algumas vezes quando você envia uma pasta com macros nela a um colega e ele não consegue trabalhar com elas provavelmente é porque sua definição de segurança está em "High". Logo lhe direi como mudá-la em "Ferramentas/Macros/Segurança".

Um arquivo Excel (.xls) não pode ser infectado por um destes vírus que aparecem regularmente na Internet mas alguém pode desenvolver procedimentos VBA (macros) que possam danificar seus dados e seu computador seriamente. Então defina o nível de segurança do Excel para "Medium" (Ferramentas/Macro/Segurança) e cada vez que você estiver tentando abrir uma pasta que contenha macro a janela de diálogo seguinte aparecerá.



Adotar a mesma atitude quando você tiver com documentos anexados a Emails. Se você conhece a origem do arquivo você pode habilitar as se não clique em "Desativar macros" e você estará inteiramente protegido. Você pode observar a pasta mas os procedimentos VBA (macros) não são operacionais. Você pode ir para quaisquer dos módulos ou outros elementos da pasta que você ver "API Function" significa que o programador está tentando acessar seu computador através do Microsoft Windows ISTO É SUSPEITO.

Protegendo o código

Como um Desenvolvedor VBA-Excel você poderá querer proteger seu código de modo que ninguém deva modificá-lo. No VBE editor vá para "Ferramentas/VBAProject Propriedades/Proteção". Marque a caixa e submeta um password. Verifique se você salvou o password em algum lugar que você se lembrará porque *crackear passwords* VBA é muito custoso.

Protegendo a Pasta

Existem muitos níveis de proteção que você pode configurar para a pasta. Primeiro você deve querer proibir qualquer um de abrir a pasta a menos que eles saibam o password. Para fazer isto no Excel vá ao "Arquivo/Salvar como" e clique em "Ferramentas/Opções Gerais".

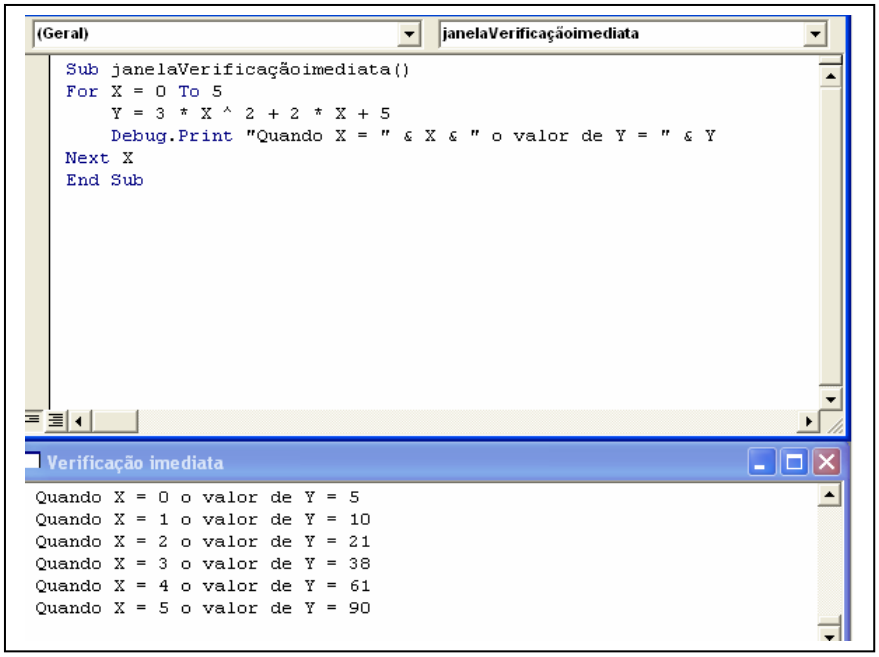
Ocultando Planilhas

Numa de suas pastas você poderá querer ocultar uma folha de planilha que contenha informação confidencial (salários e outros parâmetros). Se você apenas ocultar a folha de planilha os usuários podem reexibí-las. Existe uma maneira no VBA para ocultar uma folha de planilha sem proteger a pasta inteira. Para ver a folha de planilha tem-se que ir ao Visual Basic Editor e mudar a propriedade da folha de planilha. Se o seu código estiver protegido, ele também precisa de um password. Assim a folha de planilha é muito hidden e somente você pode obtê-la para modificar seu conteúdo. Selecione a folha de planilha na janela VBAProject e configure a propriedade visible como xlVeryHidden.

A Janela de Verificação imediata

Na de Verificação imediata (Crtl + G) podemos ver o código sendo executado e o que é retornado pelo código. Veja a figura abaixo.

Ela mostra, passo a passo, os resultados para um loop, por exemplo, conforme eles ocorrem. Neste caso usamos a janela de Verication imediata para ver os resultados de um loop que vai encontrado os valores de **Y** de uma função quadrática ($3x^2 + 2x + 5$), para **x** variando de 1 a 5.



Para vermos o código sendo executado linha por linha, podemos pressionar a tecla F8 para entrar em modo de depuração. Para cad linha a ser executada precisamos acionar a execução através da tecla F8. Este método, porém, pode ser muito eficiente. Vamos supor que neste exemplo acima o loop já tenha sido avaliado e os cálculos/fórmulas são dados como corretos. Não precisamos executar o loop passo a passo. Ao invés, colocamos um ponto de interrupção na parte que desejamos avaliar. Todo o código anterior é executado e somente interrompido no ponto por nós definido. Para isso, selecione a linha e vá em Depurar/Ativar ou desativar pontos de interrupção (F9)

Fase # 2: Código VBA do Excel (macros)

Lição 1: Palavras e sentenças (código) em VBA do Excel - Lição 2: Trabalhando com o Application - Lição 3: Trabalhando com Workbooks - Lição 4: Trabalhando com Worksheets - Lição 5: Trabalhando com Células e Ranges - Lição 6: Trabalhando com Caixas de Mensagens - Lição 7: Trabalhando com Caixas de Entrada - Lição 8: Trabalhando com Erros

Lição 1: Palavras e sentenças (código) em VBA do Excel

Umas poucas notas e dicas úteis ao se escrever código VBA.

Lição 2: Trabalhando com o Application

O objeto "Application" é o Excel, o prório programa. Como congelar a tela enquanto a macro estiver rodando, como evitar mensagens como "Um arquivo já existe com este nome, você quer fazer....", como desativar e reativar cálculos, como modificar o ponteiro, para abrir o "Open" ou "SaveAs" e outras janelas de diálogos etc..

Lição 3: Trabalhando com Workbooks

Propriedades e métodos da "ThisWorkbook" e de outras pastas. "ThisWorkbook" é a folha de planilha onde a macro está você acabou de fechá-la com o código ThisWorkbook.Close e você pode encontrar o diretório no qual ele reside com ThisWorkbook.Path

Lição 4: Trabalhando com Worksheets

Trabalhando com planilhas ativando-as, imprimido-as, copiando-as numa outra folha de planilha...

Lição 5: Trabalhando com Células e Ranges

Movendo-se entre células e ranges, valores, fórmulas e outras propriedades e métodos. Contando o número de linhas e colunas para definir os limites de seus loops (For...Next, Do...Loop, etc..).

Lição 6: Trabalhando com Caixas de Entrada

A ferramenta básica a exigir um pedaço de informação único do usuário. Quando você concretizar seus limites você move para os *userforms* (a parte divertida do VBA)

Lição 7: Trabalhando com Caixas de Mensagens

Um modo fácil para comunicar-se com o usuário.

Lição 8: Trabalhando com Erros

Quando um erro ocorre você não quer que seu usuário fique preso à mensagem de erro do VBA. Você quer dizer-lhe o que está errado e dizer-lhe como fixar o problema. Você não quer que seu procedimento apenas EXPLODA.

Lição 1: Dicas Gerais do Código VBA

Você aprendeu como criar procedimentos VBA (macros) numa lição anterior. Nesta lição você encontrará muitos deles, pequenos, ilustrando os diferentes objetos, métodos e propriedades que você pode usar para fazer as coisas acontecerem no VBA do Excel. Você descobrirá também **declarações**, **funções** e **variáveis**.

Vamos iniciar por meio de umas poucas dicas gerais. Não hesite em usar o gravador de macro para evitar erros de impressão. Escrever seu código em letras minúsculas. Se a ortografia estiver correta, o VBE colocará em maiúsculas as letras necessárias. Se não, verifique sua ortografia. Note:Aspas dentro de aspas devem ser dobradas por exemplo:

`MsgBox "Meu nome é Bertolo"` está OK e resultará em: Meu nome é Bertolo

Mas se você quiser o nome Bertolo fique entre aspas você não pode escrever:

```
MsgBox "Meu nome é "Bertolo""
```

você deve dobrar as aspas:

```
MsgBox "Meu nome é ""Bertolo"""
```

Nomeie TODAS as *células* e *ranges* que você usar no seu código de modo que se os seus endereços mudarem (adicionando ou deletando colunas ou linhas, etc...), seu código não fica invalidado.

Nomeie todas suas folhas de modo que se um usuário mudar o título (*caption*) seu código não fica invalidado. Na janela de projetos do VB Editor, quando você selecionar uma folha a Janela de propriedades lhe permite mudar ambos o nome a usar no procedimento "(Name)" e no caption "Nome" da folha que é usado na guia da folha. Sempre use o "(Name)" para desenvolver seus procedimentos. Por exemplo:

`Sheet1.Select` ou `folProgramacaoNome.Select`

ainda funcionarão mesmo se o usuário mudar *o caption* da folha na guia.

`Sheets("plan1").Select`

será invalidada se o usuário mudar o nome da folha na guia.

Sempre ativar a "Option Explicit". Embora, você seja forçado à declarar variáveis, existem muitas outras vantagens. Se você tiver um erro de ortografia para a sua variável, o VBE lhe dirá. Você deve sempre garantir que suas variáveis são consideradas pelo VBE. Você pode usar Shift/Space para chamar suas variáveis num menu contextual e dar duplo clique nelas para ajustá-las.

Declare todas suas variáveis, (Dim), no começo do procedimento, ele simplificará o teste do seu código.

Lição 2: Código VBA do Excel para o Objeto Application

Objeto Application

O Application é um objeto VBA, ELE É O PRÓPRIO EXCEL. Sempre que você quiser que o Excel faça alguma coisa ou se você quiser mudar as propriedades do Excel, você usará o objeto Application. Através dele, então, podemos configurar o Excel em termos de visualização, execuções e outras funcionalidades.

Este objeto importante possui muitas propriedades e métodos. Vejamos algumas delas:

Propriedade Calculation

Quando você estiver trabalhando com uma pasta na qual existem um monte de fórmulas e você quiser desativar temporariamente os cálculos você usará:

`Application.Calculation = xlManual` (Não esqueça do xl antes de Manual)

CERTIFIQUE-SE de que no final do procedimento você adicionou esta linha de código:

```
Application.Calculation = xlAutomatic
```

Se durante a execução do procedimento você quiser que uma única folha seja calculada você usará:

```
Activesheet.Calculate
```

CutCopyMode

Após cada operação Copiar/Colar, você deverá esvaziar o *clipboard* com a seguinte linha de código para ficar seguro de que a memória do computador não sobrecarregou.

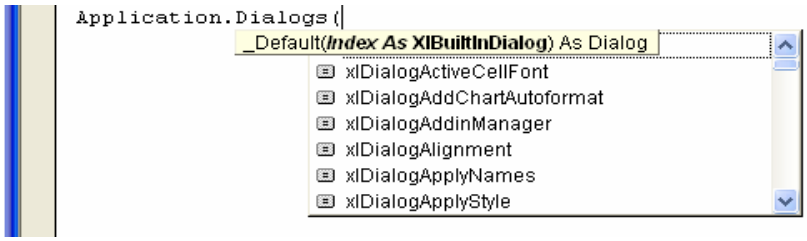
```
ActiveSheet.Paste
Application.CutCopyMode=False
```

Diálogos

Para mostrar quaisquer das janelas de diálogo do Excel você usará o seguinte código (exemplo):

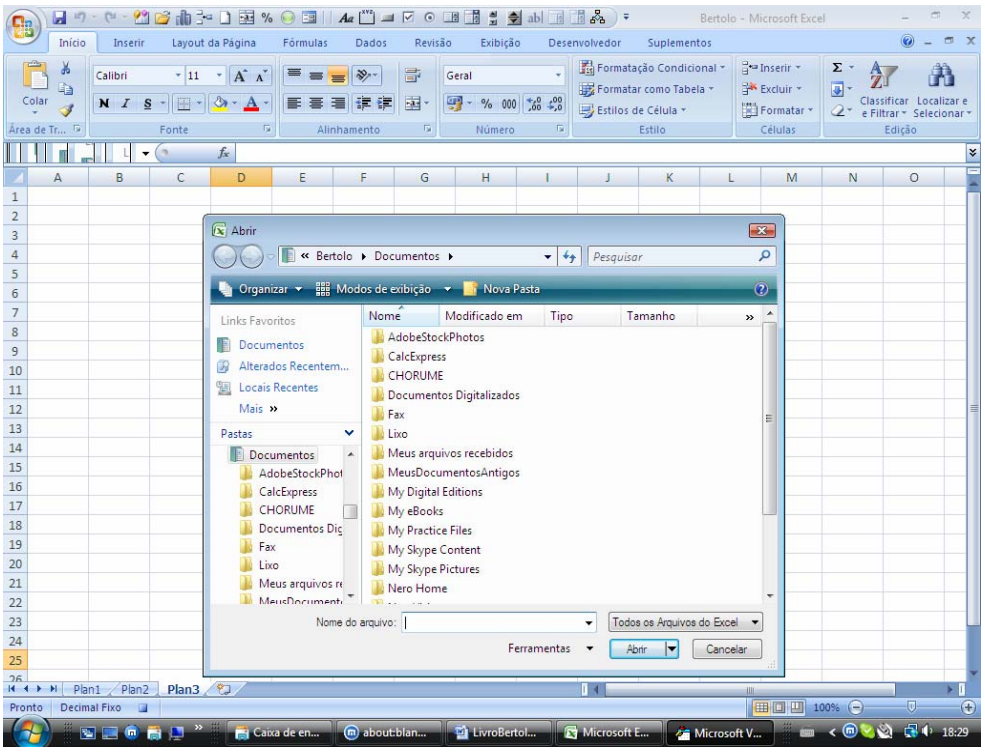
```
Application.Dialogs(xlDialogOpen).Show
```

Note que quando você digitar esta sentença, tão logo você abrir os parênteses o VBE lhe oferece a lista de todas as janelas de diálogo do Excel.



Após digitar o ponto o amigo VBE lhe oferece outra lista para você escolher dentre os itens. Neste caso foi escolhido Show.

Neste exemplo será mostrada a janela de diálogo **ABRIR**, como mostrado abaixo:



Propriedade DisplayAlerts

Habilita ou desabilita a exibição de mensagens de aviso como resposta de várias ações no Excel. True habilita as mensagens e False as desabilita.

Quando você não quiser que o Excel lhe pergunte coisas como "Um arquivo já existe....." ou "Você quer salvar este arquivo..." você usará o seguinte linha de código no começo do seu procedimento VBA.

```
Application.DisplayAlerts = False
```

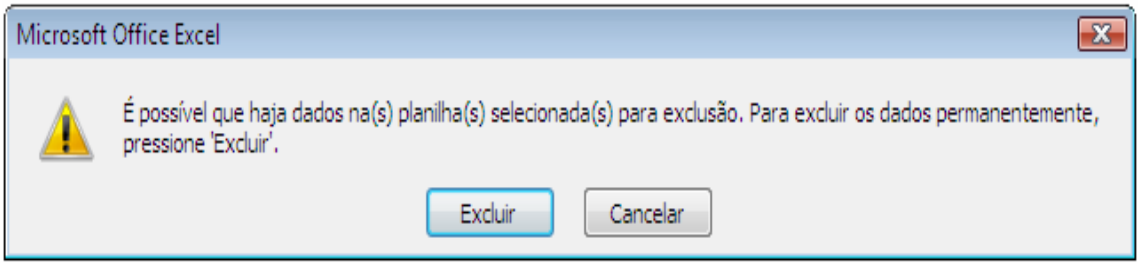
E daí então no final:

```
Application.DisplayAlerts = True
```

Vamos a um outro exemplo:

```
Sub ExcluirFolha( )
      Application.DisplayAlerts = False
      Application.Worksheets(“plan3”).Delete
      MsgBox “A folha de planilha PLAN3 foi excluída sem nenhuma
mensagem...”
      Application.DisplayAlerts = True
      Application.Worksheets(“plan2”).Delete
      MsgBox “A folha de planilha PLAN2 foi excluída com uma
mensagem de aviso...”
End Sub
```

Neste exemplo anterior aparecerá a seguinte mensagem de aviso quando da exclusão da **PLAN2**:



Propriedade DisplayFormulaBar

Exibe ou oculta a barra de fórmulas do Excel como mostra o exemplo a seguir:

```
Public Sub BarraDeFormula ( )
      Dim Resposta As VbMsgBoxResult
      Application.DisplayFormulaBar = False
      MsgBox “A barra de fórmulas está agora desabilitada”
      Resposta = MsgBox (“Você quer habilitá-la novamente?”,
vbYesNoCancel Or vbQuestion,  ”Habilitação da barra de
fórmulas”)
      If  Rsposta  = vbYes Then
            Application.DisplayFormulaBar = True
            MsgBox “A barra de fórmula acaba de ser habilitada
novamente”
      ElseIf Resposta = vbNo Then
            MsgBox “Você clicou em Não. A barra de fórmulas
continuará desabilitada”
      Else
            MsgBox “Você clicou em Cancelar. A barra de fórmulas
continuará desabilitada”
End Sub
```

Propriedade DisplayFullScreen

Exibe o Excel em tela cheia, diferente de maximizar a aplicação.

```
Public Sub TelaCheia ( )
      Application.DisplayFullScreen = True
      MsgBox “A tela está no modo de Tela Cheia”
      Application.DisplayFullScreen = False
      MsgBox “A tela voltou ao modo normal”
End Sub
```

Propriedade EnableAnimations

Essa propriedade ativa animação quando inserimos ou excluimos linhas ou colunas de uma planilha. Podemos ter os seguintes valores:

```
Application.EnableAnimations = True
Application.EnableAnimations = False
```

Propriedade EnableAutoComplete

Esta propriedade habilita ou desabilita o recurso de AutoCompletar textos já conhecidos:

```
Application.EnableAutoComplete = True
Application.EnableAutoComplete = False
```

Propriedade EnableSound

Habilita ou desabilita a emissão do beep dentro do Excel.

```
Application.EnableSound = True
Application.EnableSound = False
```

Método OnKey

O método OnKey é o responsável pela atribuição de teclas de atalho para procedimentois específicos. Esse método é sempre acionado para designar teclas de atalho no momento de criação (design) de uma macro, quando configuramos essa opção.

Sua sintaxe é:

```
Application.Onkey(Tecla,Procedimento)
```

Tecla é uma string que mostra uma tecla ou combinação de teclas que servirão de atalho para um determinado procedimento Sub.

Vamos a um exemplo. O atalho de teclado Crtl + A exibe a caixa de diálogo Abrir dentro de uma pasta de trabalho. Vamos desabilitá-la:

```
Application.OnKey "^a", " "
```

Para fazer com que o atalho volte à sua função normal, fazemos o seguinte:

```
Application.OnKey "^a"
```

O argumento Tecla pode assumir qualquer combinação de teclas. Para tanto, designamos as letras como atalho direto e temos teclas especiais para designar combinações ou outras teclas não-diretas.

Vejamos a seguir uma lista de teclas especiais para o Microsoft Excel. As teclas combinadas com Shift, Ctrl e Alt usam os seguintes prefixos:

Teclas	*Código*
Shift	*+ (sinal de mais)*
Ctrl	*^ (circunflexo)*
Alt	*% (porcentagem)*
Backspace (retorno)	*{BACKSPACE} ou {BS}*
Break (interrupção)	*{BREAK}*
Caps Lock (letras maiúsculas)	*{CAPSLOCK}*

Clear (limpar)	*{CLEAR}*
Delete ou Del (apagar)	*{DELETE} ou {DEL}*
Down Arrow	*{DOWN}*
End	*{END}*
Enter (entrar)	*{ENTER}*
Esc (sair)	*{ESCAPE} ou {ESC}*
Help (ajuda)	*{HELP}*
Home (ponto inicial)	*{HOME}*
Ins (inserir)	*{INSERT}*
Left Arrow (seta a esquerda)	*{LEFT}*
Num Lock (bloqueio numérico)	*{NUMLOCK}*
Page Down (página abaixo)	*{PGDN}*
Page Up (página acima)	*{PGUP}*
Return (retorno)	*{RETURN}*
Right Arrow (seta a direita)	*{RIGHT}*
Scroll Lock (bloqueio de rolagem)	*{SCROLLLOCK}*
Tab (guia para tabulação)	*{TAB}*
Up Arrow (seta para cima)	*{UP}*
F1 a F15	*{F1} a {F15}*

Método GoTo

Para selecionar uma certa célula você pode usar

```
Application.Goto Reference:=Range("V300")
```

ou mais simplesmente

```
Range("V300").Select
```

Mas se você quiser que esta célula seja selecionada e fique como a célula do topo/esquerdo na sua tela, você usará

```
Application.Goto Reference:=Range("V300"), Scroll=True
```

Método Quit

Esse método encerra o Microsoft Excel e dá os procedimentos comuns antes da finalização, com uma caixa de mensagem perguntando se o usuário deseja salvar a aplicação.

A seguinte linha de código fecha Excel no geral.

```
Application.Quit
```

ScreenUpdating

Quando você não quiser ver sua tela siga as ações do seu procedimento VBA, você inicia e termina seu código com as seguintes sentenças:

```
Application.ScreenUpdating = False
Application.ScreenUpdating = True
```

Lição 3: Procedimentos (Procedures) VBA para Pastas (Workbooks)

Um objeto Workbook representa uma pasta de trabalho aberta. Cada objeto Workbook é armazenado em uma **coleção** Workbooks, que faz parte das coleções do objeto Application, visto na lição 2.

Como um objeto Workbook representa uma pasta de trabalho, podemos acessá-lo como um índice da coleção Workbooks da seguinte maneira: Workboobs(1) ou pelo nome do arquivo Workbooks("nome do arquivo").

Para você perceber a diferença entre o objeto Application e o objeto Workbook, monte o seguinte código:

```
Public Sub DiferençaDeObjetos ( )
      MsgBox Application.Name
      MsgBox Workbooks(1).Name
End Sub
```

Retorna o nome da pasta.XLS (ou = o nome-arquivo)

ThisWorkbook

"ThisWorkbook" é um objeto VBA. É a **pasta** dentro da qual os procedimentos rodam.

Vejamos agora alguma PROPRIEDADES do objeto Workbook:

Propriedade FullName

Retorna o Path e o Name do objeto. Digamos que uma pasta chamada Nome.xls fosse salva em C:\Meus Documentos. A declaração:

```
Workbooks(1).FullName
```

Retornará com o seguinte:

C:\Meus Documentos\Nome.xls

Propriedade Path

Retorna o caminho onde está salva a pasta de trabalho desejada. Digamos que uma pasta de trabalho chamada Nome.xls fosse salva em C:\Meus Documentos. A declaração:

```
Workbooks(1).FullName
```

Retornará com o seguinte:

C:\Meus Documentos

Quando você trabalhar com pastas diferentes que estão todas no mesmo diretório que ThisWorkbook você pode mudar o diretório default "Abrir" com esta linha de código:

```
varPath= ThisWorkbook.Path
```

Mas eu prefiro a seguinte abordagem para ficar seguro de que eu abri ou saveAs outras pastas no mesmo diretório que ThisWorkbook

```
Sub proTeste()
Dim varPath as String
```

```
    Workbooks.Open "pasta1.xls"
    varPath=ThisWorkbook.Name
    Workbooks.Open varPath & "/" & "pasta1.xls"
    Workbooks.SaveAs varPath & "/" & "pasta1.xls"
End Sub
```

Propriedade Saved

Indica se a pasta de trabalho já está salva ou não. Se esta propriedade retornar True, é porque a pasta já foi salva.

Se você quiser fechar ThisWorkbook sem salvá-la:

```
ThisWorkbook.Saved = True
ThisWorkbook.Close
```

Propriedade HasPassword

Verifica se a pasta de trabalho possui uma senha de acesso. Em caso positivo, seu retorno é True.

Vejamos agora alguns MÉTODOS deste objeto Workbook:

Método Activate

Este método torna uma pasta de trabalho ativa. Por exemplo,

```
Workbooks(1).Activate
```

Ou

```
ThisWorkbook.Activate
```

Método AddToFavorites

Adiciona um atalho para a pasta de trabalho no menu Favoritos do Windows.

```
ThisWorkbook.AddToFavorites
```

Método Close

Este método fecha a pasta de trabalho desejada. O método Close tem alguns parâmetros que, se omitidos, vão garantir que o funcionamento da ação seja padrão Windows. O método vai verificar se a propriedade Saved do objeto Workbook é True. Se for, fechará a pasta; caso contrário, perguntará ao usuário se deseja salvar a pasta de trabalho.

```
Workbooks(1).Close
```

Para fechar a pasta ativa, basta:

```
ThisWorkbook.Close
```

Métodos Close e Saved atuando juntos

Como vimos acima see você quiser fechar a pasta ativa você escreve:

```
ActiveWorkbook.Close
```

Se você quiser fechar a pasta ativa sem salvá-la:

```
ActiveWorkbook.Saved = True
ActiveWorkbook.Close
```

Se você não quiser dificultar o código com o nome do arquivo, você pode usar os endereços de uma célula na qual você entrou com o nome do arquivo ou uma variável na qual você armazenou o nome do arquivo.

```
Workbooks(Range("A1").Value).Close
Workbooks(varFileName).Close
```

Note a ausência de aspas duplas nas últimas duas sentenças.

Método PrintOut

Imprime toda a pasta de trabalho desejada, ou parte dela, se for especificado. Sua sintaxe:

```
Workbooks(1).PrintOut(From, To, Copies, Preview,
ActivePrinter, PrintToFile, Collate)
```

Serão impressas as páginas que estiverem no intervalo de From até To, com o número total de cópias Copies. Se Preview for configurado para True, você obterá o recurso de visualização antes da impressão. ActivePrinter designa o nome da impressora ativa, enquanto que PrintToFile solicita o nome de um arquivo para ser base da impressão. Collate marcado como True intercala múltiplas cópias.

Método Save

Este método salva quaisquer mudanças na pasta de trabalho.
Você geralmente salva a pasta que está ativa, então aqui está o código:

```
ActiveWorkbook.Save
```

Mas você também pode salvar uma pasta que esteja aberta mas não ativa com:

```
Workbooks("Pasta1.xls").Save
```

Se você não quiser dificultar o código com o nome do arquivo, você pode usar os endereços de uma célula na qual você entrou com o nome do arquivo ou a variável na qual você armazenou o nome do arquivo.

```
Workbooks(Range("A1").Value).Save
Workbooks(varFileName).Save
```

Note a ausência das aspas duplas nas últimas duas sentenças.

SaveAs

Quando você quiser salvar a pasta ativa sob um outro nome ou num outro diretório, você usará o seguinte código:

```
ActiveWorkbook.SaveAs "talqual.xls" or
ActiveWorkbook.SaveAs "C:/talqual.xls"
```

Se você não quiser dificultar código do nome do arquivo você pode usar os endereços de uma célula na qual você entrou com o nome do arquivo ou a variável na qual você armazenou o nome do arquivo.

Você também pode aplicar o método "saveAs" numa pasta que está aberta mas não ativa:

```
Workbooks("Pasta1.xls").SaveAs "C:/talqual.xls"
```

Se você quiser que o usuário escolha um diretório e crie um nome do arquivo na janela de diálogo regular "SaveAs" você escreve este código:

```
Application.Dialogs(xlDialogSaveAs).Show
```

Abrindo Outras Pastas (Workbooks) e Arquivos

NOTA IMPORTANTE: Existem duas maneiras para abrir manualmente a pasta. Você clica no seu nome no Windows Explorer ou você abre o Excel e então na barra de menu "Arquivo/Abrir" você a encontra e abre a pasta. Se você abrir o Excel e então com o item de menu "Arquivo/Abrir" você seleciona a pasta, as funcionalidades "Abrir", "Salvar" e "Salvar como" todas apontam para o diretório na qual reside a pasta que você acabou de abrir. Mas se você abre uma pasta do Windows Explorer as funcionalidades "Abrir" e "Salvar" apontam amabas para o diretório na qual reside a pasta que você acabou de abrir. **MAS, mas** a funcionalidade "Salvar como" aponta para o diretório *default* (usualmente "Meus Documentos"). Lembre-se desta realidade quando você iniciar "**abertura**", "**salvamento**" ou "**salvando como**" de pastas e outros arquivos nos seus procedimentos VBA.

Abrindo Workbooks

Para abrir um outro arquivo (pasta) Excel com um procedimento VBA você simplesmente escreve (note que "Workbooks" é plural):

```
Workbooks.Open "talqual.xls"
```

Você não precisa especificar um diretório (*path*) se a segunda pasta reside no mesmo diretório que "ThisWorkbook"

Você poderá não querer dificultar o código com o nome do arquivo (de modo que se ele mudar você não tem que modificar o código). Entre com o nome do arquivo na célula "A1" da **plan1** por exemplo e escreva isto:

```
Workbooks.Open Sheets("plan1").Range("A1").Value
```

O modo que eu prefiro fazer é armazenar o nome do arquivo numa variável do tipo *string* como isto:

```
Sub proTeste()
Dim varNomeArquivo as String
     varNomeArquivo=Sheets("plan1").Range("A1").Value
     Workbooks.Open varFileName
End Sub
```

Se você quiser que o usuário escolha um arquivo da janela de diálogo regular "Abrir" você escreve este código:

```
Application.Dialogs(xlDialogOpen).Show
```

Se você quiser especificar o diretório escreva:

```
Workbooks.Open "C:\talqual.xls"
```

note que você usa a linha inclinada \ e não a linha inclinada /.

Quando você somente entrar com o *path* e o nome do arquivo numa célula, você pode escrever:

```
Workbooks.Open Sheets("plan1").Range("A1").Value
```

Você também pode escrever o *path* numa célula e o nome do arquivo numa outra célula:

```
Workbooks.Open Sheets("plan1").Range("A1").Value &
Sheets("plan1").Range("A2").Value
```

Você também pode trabalhar com 2 variáveis para o path e o nome do arquivo como mostrado acima:

```
Workbooks.Open varPath & varNomeArquivo
```

Abrindo Outros Arquivos

Quando eu preciso abrir um TXT, CSV ou qualquer outro tipo de arquivo, geralmente eu uso o gravador de macro para fazê-lo e obter algo como este:

```
Workbooks.OpenText Filename:= _
        "C:\Pasta1.txt", Origin:=xlMSDOS, _
        StartRow:=1, DataType:=xlDelimited,
TextQualifier:=xlDoubleQuote, _
        ConsecutiveDelimiter:=True, Tab:=True, Semicolon:=False,
Comma:=False, _
        Space:=True, Other:=False, FieldInfo:=Array(1, 1),
TrailingMinusNumbers _
        :=True
```

Se eu não quiser que o *path* e o nome do arquivo seja difícil de codificar, eu uso a mesma abordagem como mostrada acima para Workbooks.Open usando endereços ou variáveis.

Eu então deleto o TrailingMinusNumbers:=True porque certas versões antigas do Excel não entendem este argumento.

Movendo-se pelas Workbooks

Quando duas pastas estão abertas e você quiser mover de uma para a outra você usará `ThisWorkbook.Activate` e `Windows("OutraPasta.xls").Activate`

Quando eu trabalho com duas pasta geralmente declaro uma variável na qual eu armazeno o nome da segunda pasta e movo de uma para outra com `ThisWorkbook.Activate` e `Windows(varThatWorkbook).Activate`. Aqui está um exemplo:

```
Sub proTest()
Dim varThatWorkbook as String

     Workbooks.Open "pasta1.xls"
     varThatWorkbook=ActiveWorkbook.Name
     ' e daí eu movo de uma para outra até eu fechar a outra
pasta
     ThisWorkbook.Activate
     Windows(varThatWorkbook).Activate
     Windows(varThatWorkbook).Close

End Sub
```

EXEMPLO

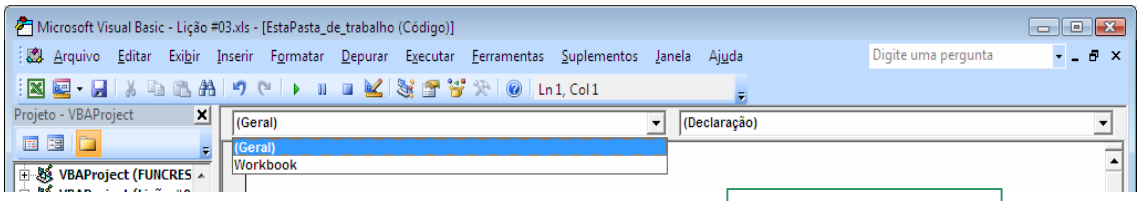
No exemplo que segue, vou dar uma amostra de coisas simples que podemos fazer com o objeto Workbook. Esse é um procedimento que faz uma análise de alguns detalhes da pasta de trabalho ativa e nos dá algumas informações sobre ela e suas planilhas.

```
Public Sub Analise ( )

      Dim Wb As Workbook
      Dim Ws As Worksheet
      Dim Quantidade As Byte
      Dim Plan() As String
      Dim Nome, Mensagem As String

      Set Wb = ActiveWorkbook
      Nome = Wb.Name
      Quantidade = Wb.Sheets.Count
      ReDim Plan(1 To Quantidade)
      For i = 1 To Quantidade
            Plan(i) = Wb.Sheets(i).Name
      Next i

      Mensagem = “Essa pasta de trabalho se chama “& Nome &
Chr(13)
      Mensagem = Mensagem & “Nela há “& Quantidade & “
planilhas”& Chr(13)
      Mensagem = Mensagem & “Seus nomes: “
      For i = 1 To Quantidade
            Mensagem = Mensagem & Chr(13) & Plan(i)
      Next i
      MsgBox Mensagem, vbExclamation, “Análise”
End Sub
```

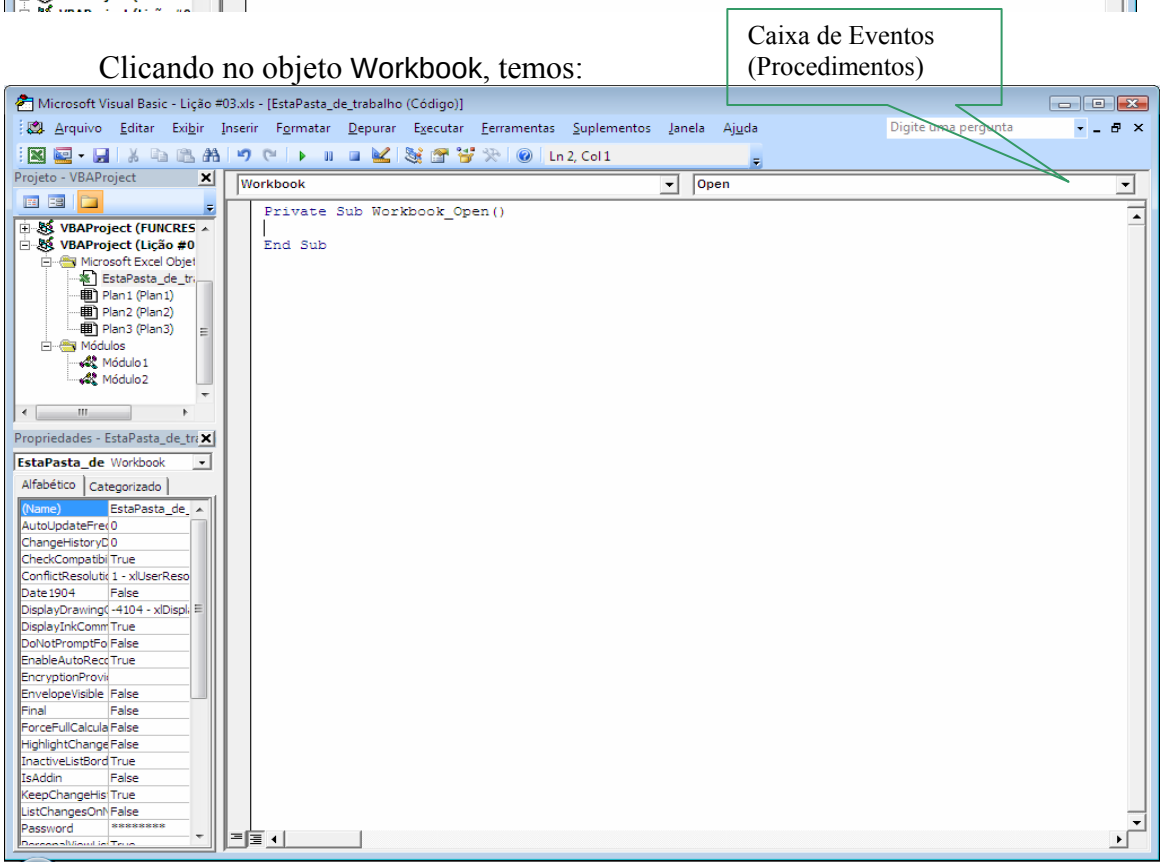
Vejamos agora alguns **EVENTOS** do objeto Workbook. Lembrando que um evento é algo que acontece (causa) com um determinado objeto e que dispara um método (efeito), ou seja uma ação neste objeto. É claro, portanto, que de acordo com o evento que ocorrer será executado um método (que no fundo é um bloco de códigos ou procedimento) apropriado.

O objeto Workbook tem uma série de eventos (em termos de código, uma Sub) que podem ser aplicados a ele.

Para acessar os eventos de um Workbook dentro de um ambiente VBA, dê um clique duplo em EstaPasta_de_Trabalho, que está no seu **Projeto** na Janela VBAProject. Com isso vai aparecer uma janela de código parecida com a do módulo. Clicando na caixa de combinação (combobox) situada no canto superior esquerdo da janela (na qual está aparecendo a opção Geral, também chamada de **caixa dos objetos**), poderemos encontrar o objeto Workbook .



Clicando no objeto Workbook, temos:



Abrindo a caixa de eventos pode-se ver o nome de todos os eventos que o objeto Workbook pode sofrer.

Olhemos alguns mais usados para descobrir o que podem fazer.

Evento Activate

Ocorre sempre que a pasta de trabalho se torna ativa. Isso pode ocorrer de duas maneiras: após a pasta de trabalho ser aberta ou após alternarmos a pasta de trabalho com outra pasta através do menu Janela, ou da barra de tarefas.

Para testar, inclua o seguinte código no evento Activate, da pasta de trabalho, e tente alternar a janela ou reabrir o arquivo.

```
Private Sub Workbook_Activate( )
      Dim Nome As String
      Nome = ThisWorkbook.Name
      MsgBox "A pasta de trabalho "& Nome & _
            " acaba de tornar-se ativa"
End Sub
```

Evento BeforeClose

Ocorre antes que a pasta de trabalho seja fechada. O evento BeforeClose possui um argumento: Cancel As Boolean. Este argumento serve para cancelarmos o evento caso achemos necessário. Para fazermos isso, simplesmente dizemos que Cancel = True.

Para testarmos, podemos elaborar a nossa própria mensagem de saída da pasta de trabalho. Na pasta de trabalho, crie o código que segue e salve a pasta com um nome qualquer. Feche-a e veja a mensagem. Em seguida, reabra o arquivo, faça algumas alterações de texto e torne a fechar. Isso pode ser bem interessante.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)

      Dim WS As Workbook
      Dim Resposta As  vbMsgBoxResult

      Set WS = ThisWorkbook
      If WS.Saved = False Then
      WS.Saved = True
      Resposta = MsgBox(“Gostaria de salvar as alterações”& _
      “nesta pasta de trabalho?”, vbYesNoCancel Or _
      vbQuestion, “Salvar Personalizado”)
            If Resposta = vbYes Then
                  WS.Save
                  MsgBox “Tchau”, vbExclamation
            ElseIf Resposta = vbCancel Then
            End If
            Else
                  MsgBox “Tchau”, vbExclamation
      End If
End Sub
```

Evento Deactivate

É o oposto do Activate. Ocorre sempre que uma folha é desativada. Como exemplo, podemos colocar uma mensagem antes de fechar a pasta de trabalho:

```
Private Sub Workbook_Deactivate()
 MsgBox "A folha está sendo desativada", vbInformation,
"Desativar a folha"
End Sub
```

Evento NewSheet

Ocorre quando uma nova folha é criada ou inserida dentro de uma pasta de trabalho.

Esse evento tem como argumento uma variável Sh, do tipo Object, que vai ser uma referência para a planilha criada e pode ser utilizada dentro do código como um elemento Worksheet.

No exemplo que segue, colocamos em uma mensagem o nome da planilha que está sendo criada, movemos esta para o final da lista de planilhas e solicitamos que o usuário digite um novo nome.

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    Dim Resposta As String

    MsgBox "O nome da nova folha é  " & Sh.Name
    Sh.Move , after:=ThisWorkbook.Worksheets _
    (ThisWorkbook.Worksheets.Count)
    Resposta = InputBox("Qual o novo nome da folha a ser
criada?")
    Sh.Name = Resposta

End Sub
```

Evento Open

Ocorre quando a pasta de trabalho é aberta. Esse é o evento padrão de Workbook, ou seja, quando selecionamos o objeto em sua janela de códigos, é esse evento que se torna ativo. Seja o exemplo a seguir:

```
Private Sub Workbook_Open()
    MsgBox "A pasta de trabalho " & ThisWorkboob.Name & _
    " acaba de ser aberta.", vbExclamation, "Abertura da pasta"
End Sub
```

Para poder ver o exemplo acima em funcionamento, você deve salvar a pasta de trabalho, fechá-la e tornar abrí-la.

Evento SheetActivate

Ocorre quando qualquer uma das folhas contidas na coleção Worksheets da pasta de trabalho for ativada. Esse evento também possui o argumento Sh, que representa a planilha que acaba de ser ativada. Mais um exemplo:

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)

     MsgBox “A folha “ & Sh.Name & _
     “ acaba de ser ativada.”, vbInformation

End Sub
```

Os outros eventos são relacionados às planilha e serão vistos na próxima lição com o escopo unitário da própria planilha, mas que pode ser expandido facilmente para o escopo global. Vejamos quais são eles:

Evento SheetBeforeDoubleClick

Ocorre no momento em que é dado um duplo clique em alguma planilha, antes da ação padrão de duplo clique.

Evento SheetBeforeRightClick

Ocorre no momento em que o botão direito do mouse é clicado em alguma planilha, antes da ação padrão de clique com o botão direito.

Evento Sheet.calculate

Ocorre após qualquer uma das planilhas ser recalculada ou qualquer mudança de dados ser feita em algum gráfico.

Evento Sheet.change

Dá-se quando é executado algum tipo de alteração em qualquer célula das planilhas.

Evento SheetSelectionChange

Ocorre quando muda a seleção de células em qualquer planilha.

Lição 4: Código VBA do Excel para Planilha (WorkSheet)

O **objeto** WorkSheet representa uma planilha especificada da **coleção** WorkSheets.
Por alguma razão obscura quando você clicar num nome de folha na janela VBAProject do Visual Basic Editor (VBE) ela tem duas propriedades chamadas Name. Uma entre parênteses e outra não. Uma planilha tem dois nomes, aquele um que aparece na sua guia no Excel (vamos chamar este de propriedade "*caption*") e aquele um que tem como **objeto** VBA na janela VBAProject do Visual Basic Editor (VBE) (por *default*: Plan1, Plan2....). Assim você pode ativar **Plan1** "cujo *caption* é "Balanço"" com ambas declarações:

```
Sheets("Balanço").Select
Worksheets("Balanço").Select
```

não esquecer os parênteses e as aspas duplas ou

```
Plan1.Select
```

Eu prefiro o segundo método por 2 razões. Primeiro há menos para teclar nele e em segundo lugar, se você, ou seu usuário, sempre mudar o *caption* da folha, não há necessidade para rever e corrigir seu código de acordo. A única desvantagem é que o código VBA não é claro e fácil para ler (O que é "Plan1"?). É por isto que eu renomeio as folhas no Visual Basic Editor (eu seleciono a folha na Janela Project e modifico o seu nome (aquele um com parênteses) na Janela Propriedade). No exemplo abaixo o nome da folha (quando objeto VBA) é "folBalanco" quando para o caption, realmente eu não tomo cuidado:

```
folBalanco.Select
```

Propriedades da Worksheet

Quando a folha for selecionada na janela VBAProject você pode ver 11 propriedades da planilha na Janela de propriedades do VBE, propriedades para as quais você pode definir um valor *default* para começar com ele e que você pode modificar pelo procedimento VBA sempre que você quiser.

Existem 3 propriedades que você usará freqüentemente: a Name (nome dentro de parênteses), a Name (sem parênteses) que é de fato o *caption* aparecendo na guia da folha no Excel e a propriedade visible.

Propriedades (Name)

Como explicado acima você pode mudar a (Name) se você estiver desenvolvendo uma pasta para outros que possam modificá-la no Excel. Você não pode mudar o (Name) de uma folha programaticamente.

Deve-se tomar cuidado de não se usarem caracteres especiais tais como: “:”, “/”, “?”e “*”.

Para mudar o *caption* você pode fazê-lo na janela propriedade do VBE, ou no Excel, clicando com o botão direito do mouse na guia e daí selecionando "Renomear". Programaticamente você pode mudar o *caption* de uma folha com o seguinte código:

```
Sheets("Plan1").Name= "Balanço"
```

Propriedades Visible

A propriedade "Visible" controla a visibilidade de uma planilha e pode tomar 3 valores diferentes. Os primeiros dois são True ou False significando que uma certa folha está ou não está visível que ela está oculta ou não.

```
Sheets("Plan1").Visible= True
Sheets("Plan1").Visible= False
```

Lembre-se que fórmulas em células são calculadas mesmo se a folha estiver oculta mas que antes você pode fazer algo programaticamente na folha você ocultá-la:

```
Sheets("Plan1").Visible= True
Sheets("Plan1").Select
Range("A1").Value=6
Sheets("Plan1").Visible= False
```

O terceiro valor que a propriedade "Visible" pode tomar é muito interessante. Uma folha pode ser very hidden `"Sheets("Plan1").Visible= xlVeryHidden".` Neste estado não somente a folha está oculta mas você não pode ver seu nome quando no Excel você for para "Formatar/Planilha/Reexibir". O valor xlVeryHidden pode somente ser variado programaticamente. O que significa que somente usuários que tenham acesso ao código VBA podem exibir esta folha. Se seu código estiver protegido por um *password* somente usuários com o *password* pode acessar o código e modificar o valor "xlVeryHidden". Você pode usar este valor da propriedade "Visible" para ocultar informação confidencial como salários e preços ou ocultar parâmetros que você não quer que sejam modificados pelo usuário.

Lembre-se que fórmulas nas células são calculadas mesmo se a folha está very hidden mas que antes você pode fazer qualquer coisa programaticamente na folha você deve reexibí-la:

```
Sheets("Plan1").Visible= True
Sheets("Plan1").Select
Range("A1").Value=6
Sheets("Plan1").Visible= xlVeryHidden
```

Lembre-que fórmulas nas outras folhas referindo-se às células de uma folha hidden ou very hidden funciona mesmo se a folha estiver hidden ou very hidden.

Se você quiser ocultar muitas folhas ao mesmo tempo você usará o seguinte código:

```
Sheets(Array("Plan1", "Plan2")).Select
Sheets("Plan1").Activate
ActiveWindow.SelectedSheets.Visible = False
```

Os valores xlSheetHidden e xlSheetVisible permitem, respectivamente, ocultar ou exibí-la novamente.

Propriedades ScrollArea

Essa propriedade limita a área de rolagem e atuação de uma planilha. Por exemplo:

```
ActiveSheet.ScrollArea = “A1:D20”
```

Permite-nos trabalhar apenas no intervalo delimitado.

Propriedades CodeName

Retorna o nome de código de uma planilha, que substitui a declaração Worksheets(index). Podemos usar:

```
Plan1.Cells(1,1).Value = 10
```

Ao invés de :

```
Worksheets(1).Cells(1,1).Value = 10
```

Vejamos agora alguns **MÉTODOS** do objeto Worksheet:

Método Deletar

Você poderá querer deletar folhas. Aqui está o código para fazer isto:

```
Sheets("Balanço").Delete
```

Ou

```
Worksheets(worksheets.count).Delete
```

Método Add

Você poderá também querer adicionar uma folha. Se você usar o seguinte código o VBA adicionará uma nova folha antes da planilha ativa.

```
Sheets.Add
```

Se você quiser ser mais preciso como para onde e quantas você usará em cada um dos seguintes procedimentos:
Inserindo uma folha após a folha cujo *caption* é "Balanço" e cujo nome é shBalance:

```
Sub proTeste()
    Sheets.Add before:=Sheets("Balanço")
End Sub
```

```
Sub proTeste()
   Sheets.Add before:=folBalanco
End Sub
```

Inserindo três folhas após a folha cujo *caption* é "Balanço":

```
Sub proTeste()
    Sheets.Add after:=Sheets("Balanço"), Count:=3
End Sub
```

Inserindo uma folha no começo da pasta. Note a ausência de aspas duplas quando usar a posição (rank) da folha:

```
Sub proTest()
    Sheets.Add after:=Sheets(1)
End Sub
```

Finalmente se você quiser adicionar uma nova folha no final da pasta você precisa contar as folhas com Sheets.Count e usar este valor quando a posição da folha após a qual você quiser adicionar a nova folha:

```
Sub proTeste()
    Sheets.Add After:=Sheets(Sheets.Count)
End Sub
```

Método Select

Este método seleciona uma planilha. A vantagem deste método sobre o Activate é que ele serve para selecionar várias planilhas ao mesmo tempo (poderia ser com o propósito de excluí-las).

Método Copy

Serve para criar uma cópia fiel de uma determinada planilha, em uma posição específica na coleção Worksheets. Por exemplo para criar uma cópia fiel da planilha ativa como a última da coleção Worksheets:

```
Active.Copy After:=Worksheets(Worksheets.count)
```

Algumas vezes você quer fornecer uma única planilha de uma pasta para alguém mas você não quer que todas as fórmulas vão juntas. Aqui está o código para copiar uma folha de uma pasta numa nova pasta, trocar as fórmulas pelos valores e salve a nova pasta:

```
Sheets("Plan3").Select
Sheets("Plan3").Copy
Células.Select
Selection.Copy
Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks _
:=False, Transpose:=False
ActiveWorkbook.SaveAs "novaPasta.xls"
```

Se você quiser fazer a mesma coisa para muitas folhas, você repete o procedimento ou escreve:

```
Sheets(Array("Plan1", " Plan2")).Select
Sheets("Copia2").Activate
Sheets(Array("Plan1", " Plan2")).Copy
Sheets("Plan1").Select
Células.Select
Selection.Copy
Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks _
:=False, Transpose:=False
Sheets("Plan2").Select
Células.Select
Selection.Copy
Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks _
:=False, Transpose:=False
ActiveWorkbook.SaveAs "novaPasta.xls"
```

Veja a lição sobre pastas para gerenciar o diretório (*path*) quando usar o método "SaveAs".

ActiveSheet

A ActiveSheet é a planilha que foi selecionada por último. Assim você pode escrever:

```
ActiveSheet.Visible=True
ActiveSheet.Copy
```

Lembre-se que quando você copiou uma célula ou um grupo de células ou qualquer outro objeto de uma folha você SEMPRE cola-o na ActiveSheet:

```
ActiveSheet.Paste
```

a menos que você esteja fazendo um PasteSpecial no qual caso o objeto é "Selection":

```
Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
SkipBlanks _
:=False, Transpose:=False
```

Se você quiser verificar são os autofilters estão ligados para removê-los você precisa usar o objeto ActiveSheet também como no seguinte procedimento onde eu marco os autofilters para ligados para removê-los ou sair do procedimento:

```
Range("A2").Select
    If ActiveSheet.AutoFilterMode = True Then
        Selection.AutoFilter
    Else
    Exit Sub
End If
```

Você também pode do alguma coisa em cada uma das folhas numa pasta com o seguinte código. Neste exemplo eu defino o valor da célula A1 a 22 em cada planilha. Note que eu primeiro declaro uma variável do tipo Variant `Dim varSheet As Variant` e então o procedimento pode rodar.

```
Sub proTest()
Dim varSheet As Variant

     For Each varSheet In Worksheets
          Range("A1").Value = 22
    Next

End Sub
```

Vejamos agora alguns **EVENTOS** do objeto Worksheet. A maioria dos eventos de um Workbook, como foi visto, afetam globalmente uma aplicação. Os eventos relacionados a um objeto Worksheet afetam apenas a planilha onde o evento ocorre.

Evento Activate

Ocorre quando uma planilha é ativada. Por exemplo:

```
Private Sub Worksheet_Activate( )
     MsgBox “A planilha “ & Me.Name & _
     “ acaba de ser ativada.”,vbExclamation
End Sub
```

Observe o uso da cláusula Me. Ela representa o objeto que está em análise, nesdse caso, a planilha que sofre o evento. Através da cláusula Me, temos acesso a todas as propriedades e métodos da planilha.

Evento BeforeDoubleClick

Acontece quando há um duplo clique na planilha, antes que a ação padrão seja executada.

Esse evento possui dois argumentos: O argumento Cancel, já estudado em outros eventos, que proporciona a possibilidade de cancelá-lo. O outro parâmetro é Target, do tipo Range. Através desse argumento, podemos saber qual a célula recebeu o duplo clique e dar os devidos tratamentos ao evento. Vejamos um exemplo:

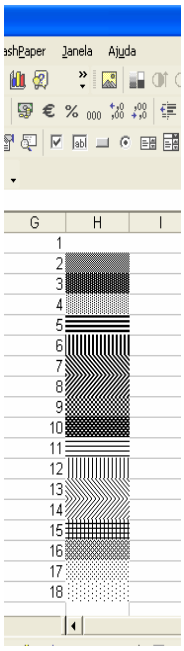
```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range,
Cancel As Boolean)
Dim Resposta As Byte
MsgBox "A célula que recebeu o duplo clique é " & Target.Address
Resposta = InputBox("Digite o número da cor para pintar a
célula:")
Target.Interior.ColorIndex = Resposta
End Sub
```

No exemplo anterior, trocamos a cor da célula que sofreu o evento de duplo clique e, através de cancel = true, evitamos que a ação padrão ocorra, a qual seria colocar a célula em modo de edição.

Evento BeforeRightClick

Esse evento ocorre quando há um clique com o botão direito do mouse na planilha. Assim como o evento BeforeDoubleClick, o BeforeRightClick tem os dois argumentos, Target e Cancel, usados para detectar a célula clicada e cancelar o evento, respectivamente.

Vamos fazer um exemplo que troca o padrão de preenchimento de uma célula que recebe o clique com o botão direito do mouse, de acordo com a vontade do usuário.



Na figura ao lado vemos todos os valores possíveis para esses padrões

Nesta Sub apresentamos um detalhe interessante, o uso de On Error, que faz um tratamento caso algum erro ocorra. Em resposta ao erro, On Error faz um desvio da aplicação para a rotina Trat, que evita o erro e cancela o evento.

Evento Calculate

Ocorre toda vez que um cálculo é executado em uma planilha, alterando valores. Por exemplo, quando montamos uma fórmula e esta executa um cálculo, alteramos os valores das células que são referências para fórmulas (a planilha não está em módulo de cálculo automático) e pressionamos F9, fazendo o recálculo da planilha, entre outros.

A seguir, temos um exemplo do uso do evento Calculate. Antes de implementar o código, crie a planilha que segue. Na coluna D, existe uma fórmula que calcula os valores de entrada, menos os valores de saída. Na célula D14, existe uma fórmula de soma das células que estão acima dela, perfazendo o saldo anual.

	A	B	C	D
1	Mês	Entradas	Saídas	Saldo
2	Janeiro	R$ 3.000,00	R$ 1.000,00	R$ 2.000,00
3	Fevereiro	R$ 2.500,00	R$ 1.500,00	R$ 1.000,00
4	Março	R$ 4.000,00	R$ 4.500,00	(R$ 500,00)
5	Abril	R$ 2.800,00	R$ 3.200,00	(R$ 400,00)
6	Maio	R$ 3.500,00	R$ 2.550,00	R$ 950,00
7	Junho	R$ 4.000,00	R$ 3.500,00	R$ 500,00
8	Julho	R$ 5.000,00	R$ 2.500,00	R$ 2.500,00
9	Agosto	R$ 3.800,00	R$ 4.000,00	(R$ 200,00)
10	Setembro	R$ 2.500,00	R$ 1.555,00	R$ 945,00
11	Outubro	R$ 3.450,00	R$ 2.870,00	R$ 580,00
12	Novembro	R$ 5.600,00	R$ 2.600,00	R$ 3.000,00
13	Dezembro	R$ 10.600,00	R$ 10.600,00	R$ 0,00
14			saldo anual	R$ 10.375,00
15				


```
Private Sub Worksheet_Calculate()

    Dim Valor As Integer

    Valor = Me.Range("d14").Value
    If Valor < 0 Then
        MsgBox "Seu balanço anual está negativo", vbCritical
    Else
        MsgBox "Seu balanço anual está positivo", vbInformation
    End If

End Sub
```

Evento Activate

Este evento ocorre quando uma das células da planilha teve o seu conteúdo alterado pelo usuário ou por um agente externo. Seja o exemplo:

```
Private Sub Worksheet_Change(ByVal Target As Range)
    On Error GoTo Trat

    MsgBox "A célula que teve o seu conteúdo alterado foi " & _
    Target.Address, vbInformation
    MsgBox "O novo valor de " & Target.Address & " é " & _
    Target.Value, vbInformation
    Exit Sub

Trat:
    MsgBox "Você deve ter alterado mais de uma célula ao" & _
    " mesmo tempo", vbCritical

End Sub
```

Perceba que, se você selecionar um conjunto de células e apagar o seu conteúdo, Target.Value retorna um erro, pois só funciona dessa forma quando temos apenas uma célula no Range selecionado. Assim, somos desviados para a rotina de tratamento de erro, que informa o que aconteceu.

Evento Activate

É o evento oposto ao evento Activate. É acionado quando desativamos a planilha.

Evento SelectionChange

Este evento é o evento padrão de uma Worksheet. Ele ocorre toda vez que um Range de células dentro de uma planilha é selecionado. Através do parâmetro de retorno Target, temos acesso a todas as células selecionadas no intervalo e podemos dar o tratamento que desejamos. Seja o exemplo:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)

    MsgBox "O Range de células selecionado é " & Target.Address
    If Target.Count = 1 Then
        MsgBox "Nesse Range existe 1 célula"
    Else
        MsgBox "Nesse Range existem " & Target.Count & "
células"
    End If

End Sub
```

Esses são os sete eventos da coleção Worksheets, que são os mais usados em uma pasta de trabalho. Lembremos que podemos estendê-los ao objeto Workbook de forma global, pois tal objeto possui os mesmos eventos detectados para qualquer planilha da aplicação.

Lição 5: Código VBA do Excel para mover-se entre as Células e Ranges

No Excel o usuário trabalha nas planilhas e precisa conhecer como fazer para ir de uma célula para outra, para uma linha, para uma coluna, como entrar com valores e fórmulas, para copiar/colar etc....

O objeto Range é quem faz o trabalho pesado dentro do Excel. Esse objeto representa uma çelula, até todas as células de uma planilha; por isso, é tão genérico e funcional.

Range, Select, Selection, Value

Quando você quiser entrar com um valor numérico numa célula você escreverá:

```
Range("A1").Select
Selection.Value = 32
```

Note que você não precisa selecionar uma célula para entrar com um valor nela, de qualquer lugar na folha você pode escrever:

```
Range("A1").Value = 32
```

Você pode mesmo mudar o valor da célula ou uma outra folha com:

```
Sheets("AssimEassim").Range("A1").Value = 32
```

Você também pode entrar com o mesmo valor em muitas células, isto é, num intervalo de células, com:

```
Range("A1:B32").Value = 32
```

Se você quiser entrar com um texto numa célula você precisa usar as aspas duplas como:

```
Range("A1").Value  = "Bertolo"
```

Se você quiser entrar com um texto dentro de aspas duplas numa célula você precisa de uma tripla de aspas duplas como:

```
Range("A1").Value  = """Bertolo"""
```

Uma intersecção de intervalos é dada assim:

```
Range(“B1:D4 C2:E6”)
```

Uma união de intervalos é dada assim:

```
Range(“B1:D4, C2:E6”)
```

Pode também ser representado por um conjunto de linhas ou colunas, assim:

```
Range(Columns(1), Columns(3))
```

Range, Formula, Select, Selection

Quando você quiser entrar com uma fórmula numa célula você escreverá:

```
Range("A1").Select
Selection.Formula = "=C8+C9"
```

Note os dois sinais de igual (=) incluindo aquele um dentro das aspas duplas como se você fosse entrá-lo manualmente. Novamente você não precisa selecionar a célula para entrar com uma fórmula nela, de qualquer lugar na folha você pode escrever:

```
Range("A1").Formula = "=C8+C9"
```

Se você escrever o seguinte:

```
Range("A1:A8").Formula = "=C8+C9"
```

A fórmula em A1 será =C8+C9, a fórmula em A2 será =C9+C10 e assim por diante. Se você quiser ter a fórmula exata =C8+C9 em todas as células, você precisa escrever:

```
Range("A1:A8").Formula = "=$C$8+$C$9"
```

Activecell

A Activecell é uma célula muito importante. Ela é a célula selecionada quando somente uma célula está selecionada mas ela é a célula do canto superior esquerdo de um range que está selecionado. Por exemplo, se você selecionar um range como este:

```
Range("A8:G8").Select
```

A Activecell é A8. Assim se você escrever:

```
Activecell.Select
```

célula A8 será selecionada. Mas se você selecionar o range de G8 a A8:

```
Range("G8:A8:).Select
```

Célula G8 torna-se a Activecell.

Vejamos agora as PROPRIEDADES deste objeto Range:

Propriedade Cells

Representa todas as células de dentro do Range. Através dela, podemos obter informações específicas a respeito de cada célula ou do conjunto como um todo.

Propriedades Column e Row

Indicam a coluna e a linha, respectivamente, que formam a **base** de um Range, ou seja, relativas à célula do canto superior esquerdo.

Vamos fazer um teste colocando o código abaixo no evento SelectionChange de uma planilha:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    MsgBox "A célula base do Range está na coluna " & _
    Target.Column & " e na linha  " & Target.Row
End Sub
```

Propriedades Columns e Rows

Veja o **s** no final. Aí está a diferença da propriedade anterior.

Estas duas propriedades agem como **coleções**, que armazenam as informações de colunas e linhas de um Range. Vejamos um exemplo:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    MsgBox "Neste Range existem " & Target.Columns.Count & "
coluna(s) " _
```

```
    & Chr(13) & "Neste Range existem " & Target.Rows.Count & "
linha(s) "
End Sub
```

Propriedade Address

Esta propriedade retorna o endereço de células do objeto Range em questão.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)

    MsgBox Target.Address(RowAbsolute:=False, _
    ColumnAbsolute:=False)

End Sub
```

Nesse exemplo, poderíamos omitir os parâmetros RowAbsolute e ColumnAbsolute; o resultado seria o mesmo endereço, masmem referência absoluta (com os símbolos de $, indicando o travamento de linhas e colunas).

Propriedades ColumnWidth e RowHeight

Essas propriedades medem e definem a largura de uma coluna e altura de uma linha, respectivamente. Quando várias colunas ou linhas estão selecionadas e têm tamanhos diferentes, essas propriedades retornam o valor Null.

A largura de uma coluna é medida em quantidade de caracterees “0” oara o tipo de fonte, enquanto que a altura de uma linha é medida em pontos.

Propriedade Formula

Esta propriedade retorna ou determina qual a fórmula contida em uma célula do Range.

Propriedade HasFormula

Retorna True, se existe uma fórmula em uma célula do Range, ou False, caso não exista.

Propriedade HorizontalAlignment

Retorna ou define o alinhamento horizontal das células contidas no Range selecionado. As constantes podem ser:

xlHAlignRight – Alinhamento à direita

xlHAlignLeft – Alinhamento à esquerda

xlHAlignJustify – Alinhamento justificado

xlHAlignDistributed – Alinhamento distribuído

xlHAlignCenter – Alinhamento centralizado

xlHAlignGeneral – Alinhamento padrão

xlHAlignFill – Preenche todo o Range horizontalmente com o conteúdo da célula.

xlHAlignCenterAcrossSelection – Alinha horizontalmente por sobre uma seleção.

Propriedade Locked

Trava ou destrava as células do Range, sendo equivalente à guia Proteção, em Formatar | Células, no item Travada.

Propriedade Value

Retorna ou define o valor de uma determinada célula do Range. Exemplo:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Dim Célula As Range, i As Integer
    For Each Célula In Target
        i = i + 1
        Célula.Value = i
    Next
End Sub
```

Aqui cada célula selecionada será preenchida com o seu index (posição) no Range.

Vejamos agora os MÉTODOS deste objeto Range:

Método AddComment

Este método adiciona um texto de comentário a uma célula do Range.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Dim R As Range
    Set R = Range(Cells(Target.Row, Target.Column), _
    Cells(Target.Row, Target.Column))
    R.AddComment "Esta é a primeira célula do Range"
End Sub
```

Método AutoFilter

O método AutoFilter habilita o recurso de autofiltro do Excel nas células que envolvem o Range.

Método AutoFit

O método AutoFit ajusta colunas ou linhas para uma melhor disposição do texto dentro das células.

Método BorderAround

É o método usado para acrescentar uma borda ao Range especificado. A sua sintaxe é:

```
Range.BorderAround(LineStyle, Weight, ColorIndex, Color)
```

Onde LineStyle pode ser uma das seguintes constantes:

xlLineStyleNone – Sem estilo

xlDouble – = = = = = = = = = = = = = = = = = =

xlDot – . . . . . .

xlDash – -

xlContinuous – _____

xlDashDot – - . - . - . - . - . - . - . - . - . - . -

xlDashDotDot – - . . - . . - . . - . . - . . - . . -

xlSlantDashDot – O mesmo que xlDashDot, mas entrelaçado

O parâmetro Weight pode ser uma das seguintes constantes:

xlHairLine – Intensidade muito fina

xlThin – Intensidade fina

xlMedium – Intensidade mediana

xlThick – Intensidade forte.

ColorIndex e Color são as constantes de cores, onde ColoIndex é definida por um valor do Index da cor da paleta e Color é definida como um valor de uma função RGB.

Uma função RGB trabalha com três parâmetros, que indicam a quantidade de vermelho, verde e azul na composição de uma cor, variando de 0 a 255. Um exemplo poderia ser a cor vermelha: RGB(255,0,0).

Apenas um componente de cor deve ser definido para o método BorderAround.

Método Clear

Limpa todo e qualquer conteúdo das células dentro de um Range. O conteúdo envolve valores, formatação e fórmulas.

Método Copy

Esse método copia o conteúdo de um Range para um outro Range de destino ou para o *clipboard*. Sua sintaxe:

```
Range.Copy (Destino)
```

Método Delete

Exclui as células do Range especificado.

Método FillDown, FillLeft, FillRight, FillUp

Copia o conteúdo da primeira célula do Range para o resto da seleção, na direção específica.

```
Range(“A1:A20”).FillDown
```

Preencherá da célula A2 até A20 com o conteúdo de A1.

Método Find

Localiza informações específicas em um intervalo, e retorna um objeto Range representando a primeira célula onde essas informações se encontram. Retorna Nothing se nenhuma coincidência for encontrada. Não afeta a seleção da célula ativa.

Vejamos a sua sintaxe:

```
Range.Find(What,After,LookIn,LookAt,SearchOrder, _
```

```
SearchDirection,MatchCase,MatchByte, SearchFormat)
```

What é uma variável do tipo **Variant** necessária. Os dados ou valores a serem buscadoss. Pode ser uma seqüência de caracterees ou qualquer tipo de dados do Microsoft Excel.

After é uma variável do tipo **Variant** opcional. A célula depois da qual você deseja que a pesquisa comece. Corresponde à posição da célula ativa quando uma pesquisa é feita a partir da interface do usuário. Observe que After precisa ser uma única célula no intervalo. Lembre-se de que a pesquisa começa *depois* dessa célula; a célula especificada não é pesquisada até que o método dê a volta e chegue a ela. Se você não especificar esse argumento, a pesquisa começará após a célula do canto superior esquerdo do intervalo.

LookIn é uma variável do tipo **Variant** opcional. O tipo de informação. Pode ser uma das seguintes constantes de busca: **xlValues** (valores), **xlComments** (comentários) ou **xlFormulas** (fórmulas)

LookAt é uma variável do tipo **Variant** opcional. Determina se o valor deve bater com todo o conteúdo da célula, ou parte dele simplesmente. Pode ser uma das seguintes constantes XlLookAt: **xlWhole** (completo) ou **xlPart** (parte).

SearchOrder é uma variável do tipo **Variant** opcional. Define se a busca deve seguir em colunas ou linhas. Pode ser uma das seguintes constantes XlSearchOrder: **xlByRows** ou **xlByColumns**.

SearchDirection é opcional. Indica a direção da pesquisa. XlSearchDirection pode ser uma das seguintes constantes:

xlNext *padrão*
xlPrevious

MatchCase é uma variável do tipo **Variant** opcional. Deve ser definida como **True** para fazer a pesquisa sensitiva, isto é, fazer distinção entre maiúsculas e minúsculas. O valor padrão é **False**.

MatchByte é uma variável do tipo **Variant** opcional. Usado somente se você tiver selecionado ou instalado suporte de linguagem de byte duplo, padrão Excel para o Oriente. **True** para que os caracterees de dois bytes coincidam somente com caracterees de dois bytes. **False** para que os caracterees de dois bytes coincidam com seus equivalentes de um byte.

SearchFormat **Variant** opcional. O formato da pesquisa.

Comentários

As definições de LookIn, LookAt, SearchOrder e MatchByte são salvas toda vez que você usa esse método. Se você não especificar valores para esses argumentos na próxima vez que você chamar o método, os valores salvos serão usados. A definição desses argumentos altera as definições da caixa de diálogo **Localizar**, e a alteração das definições da caixa de diálogo **Localizar** faz com que os valores salvos, usados quando você omite os argumentos, sejam alterados. Para evitar problemas, defina explicitamente esses argumentos toda vez que você usar esse método.

Para localizar células coincidentes com padrões mais complicados, use uma instrução For Each...Next com o operador **Like**. Por exemplo, o código a seguir pesquisa todas as células do intervalo A1:C5 que usam uma fonte cujo nome começa com as letras Cour. Quando o Microsoft Excel encontra uma coincidência, altera a fonte para Times New Roman.

```
For Each c In [A1:C5]
    If c.Font.Name Like "Cour*" Then
        c.Font.Name = "Times New Roman"
    End If
Next
```

Exemplo

Este exemplo localiza todas as células do intervalo A1:A500 na planilha um que contêm o valor 2 e o altera para 5.

```
With Worksheets(1).Range("a1:a500")
    Set c = .Find(2, lookin:=xlValues)
    If Not c Is Nothing Then
        firstAddress = c.Address
        Do
            c.Value = 5
            Set c = .FindNext(c)
        Loop While Not c Is Nothing And c.Address <> firstAddress
    End If
End With
```

Método FindNext e FindPrevious

Esses métodos prosseguem com a busca iniciada no método Find à procura de novas ocorrências do critério dentro do Range.

Exemplo

Este exemplo mostra como o método FindPrevious é usado com os métodos Find e FindNext. Antes de executar este exemplo, certifique-se de que **Plan1** contenha pelo menos duas ocorrências da palavra “Phoenix” na coluna B.

```
Set fc = Worksheets("Plan1").Columns("B").Find(what:="Phoenix")
```

```
    MsgBox "A primeira ocorrência está na célula " & fc.Address
Set fc = Worksheets("Plan1").Columns("B").FindNext(after:=fc)
    MsgBox "A próxima ocorrência está na célula " & fc.Address
Set fc = Worksheets("Plan1").Columns("B").FindPrevious(after:=fc)
    MsgBox "A ocorrência anterior está na célula " & fc.Address
```

Método Insert

Insere células, linhas ou colunas na planilha determinada

Método Merge

É o recurso de mesclar células. Tem a seguinte sintaxe:

```
Range.Merge(Across)
```

Onde Across deve ser definido como True, caso queiramos mesclar as células do Range, gerando uma célula por linha; e False, caso queiramos que todo o Range vire uma única célula.

Método Select

Seleciona o Range especificado.

Método Sort

Classifica os dados de um relatório de tabela dinâmica, um intervalo ou a região ativa se o intervalo especificado contiver somente uma célula. Sua sintaxe é:

```
expressão.Sort(Key1, Order1, Key2, Type, Order2, Key3, Order3,
Header, OrderCustom, MatchCase, Orientation, SortMethod,
DataOption1, DataOption2, DataOption3)
```

expressão - Necessária. Uma expressão que retorna um dos objetos da lista **Aplica-se a**.

Key1 **Variant** opcional. O primeiro campo de classificação, como texto (um campo de tabela dinâmica ou nome de intervalo) ou um objeto **Range** ("Dept" ou `Cells(1, 1)`, por exemplo).

Order1 É opcional. A ordem de classificação para o campo ou intervalo especificado em ***Key1***.

XlSortOrder pode ser uma das seguintes constantes:

xlDescending Classifica ***Key1*** na ordem decrescente.

xlAscending *padrão* Classifica ***Key1*** na ordem crescente.

Key2 **Variant** opcional. O segundo campo de classificação, como texto (um campo de tabela dinâmica ou nome de intervalo) ou um objeto Range. Se você omitir este argumento, não haverá um segundo campo de classificação. Não pode ser usado ao classificar relatórios de tabela dinâmica.

Type **Variant** opcional. Especifica os elementos a serem classificados. Use este argumento somente ao classificar relatórios de tabela dinâmica.

XlSortType pode ser uma das seguintes constantes:

xlSortLabels. Classifica o relatório de tabela dinâmica por rótulos.

xlSortValues. Classifica o relatório de tabela dinâmica por valores.

Order2 **XlSortOrder** opcional. A ordem de classificação para o campo ou intervalo especificado em ***Key2***. Não pode ser usado ao classificar relatórios de tabela dinâmica.

XlSortOrder pode ser uma das seguintes constantes:

xlDescending Classifica ***Key2*** na ordem decrescente.

xlAscending *padrão* Classifica ***Key2*** na ordem crescente.

Key3 **Variant** opcional. O terceiro campo de classificação, como texto (um nome de intervalo) ou um objeto **Range**. Se você omitir este argumento, não haverá um terceiro campo de classificação. Não pode ser usado ao classificar relatórios de tabela dinâmica.
Order3 **XlSortOrder** opcional. A ordem de classificação para o campo ou intervalo especificado em ***Key3***. Não pode ser usado ao classificar relatórios de tabela dinâmica.

XlSortOrder pode ser uma das seguintes constantes:

xlDescending Classifica ***Key3*** na ordem decrescente.

xlAscending *padrão* Classifica ***Key3*** na ordem crescente.

Header **XlYesNoGuess** opcional. Especifica se a primeira linha contém ou não cabeçalhos. Não pode ser usado ao classificar relatórios de tabela dinâmica.

XlYesNoGuess pode ser uma das seguintes constantes:

xlGuess. Deixa o Microsoft Excel determinar se deve haver um cabeçalho, e a determinar onde ele deve estar, se existir um.

xlNo *padrão*. (O intervalo inteiro deve ser classificado).

xlYes. (O intervalo inteiro não deve ser classificado).

OrderCustom **Variant** opcional. Este argumento é um deslocamento de inteiro baseado em um para a lista de ordens de classificação personalizadas. Se você omitir OrderCustom, será usada uma classificação normal.

MatchCase **Variant** opcional. **True** para fazer uma classificação que faz distinção entre maiúsculas e minúsculas; **False** para fazer uma classificação que não faz distinção entre maiúsculas e minúsculas. Não pode ser usado ao classificar relatórios de tabela dinâmica.

Orientation **XlLink** opcional. A orientação da classificação.

XlSortOrientation pode ser uma das seguintes constantes:

xlSortRows *padrão*. Classifica por linha.

xlSortColumns. Classifica por coluna.

SortMethod **XlSortMethod** opcional. O tipo de classificação. Algumas dessas constantes podem não estar disponíveis, dependendo do suporte a idioma (português do Brasil, por exemplo) que você selecionou ou instalou.

XlSortMethod pode ser uma das seguintes constantes:

xlStroke Classificação pela quantidade de pressionamentos de tecla em cada caracteree.

xlPinYin *padrão*. Ordem de classificação do chinês fonético para caracterees.

DataOption1 **XlSortDataOption** opcional. Especifica como classificar texto em tecla 1. Não pode ser usado ao classificar relatórios de tabela dinâmica.

XlSortDataOption pode ser uma das seguintes constantes:

xlSortTextAsNumbers. Trata o texto como dados numéricos para a classificação.

xlSortNormal *padrão*. Classifica dados numéricos e de texto separadamente.

DataOption2 **XlSortDataOption** opcional. Especifica como classificar texto em tecla 2. Não pode ser usado ao classificar relatórios de tabela dinâmica.

XlSortDataOption pode ser uma das seguintes constantes:

xlSortTextAsNumbers. Trata o texto como dados numéricos para a classificação.

xlSortNormal *padrão*. Classifica dados numéricos e de texto separadamente.

DataOption3 **XlSortDataOption** opcional. Especifica como classificar texto em tecla 3. Não pode ser usado ao classificar relatórios de tabela dinâmica.

XlSortDataOption pode ser uma das seguintes constantes:

xlSortTextAsNumbers. Trata o texto como dados numéricos para a classificação.

xlSortNormal *padrão*. Classifica dados numéricos e de texto separadamente.

Comentários

As configurações para Header, Order1, Order2, Order3, OrderCustom e Orientation são salvas, na planilha específica, sempre que você usa este método. Se você não especificar valores para estes argumentos na próxima vez em que chamar o método, serão usados os valores salvos. Defina esses argumentos explicitamente sempre que for usar o método **Sort** , caso escolha não usar os valores salvos.

Seqüências de caracterees de texto que não podem ser convertidas em dados numéricos são classificadas normalmente.

Observação Se não houver argumentos definidos com o método **Sort**, o Microsoft Excel classificará a seleção, escolhida para classificação, na ordem crescente.

Exemplo

Este exemplo classifica o intervalo A1:C20 em Plan1, usando a célula A1 como a primeira chave de classificação e a célula B1 como a segunda chave de classificação. A classificação é feita na ordem crescente por linha, e não há cabeçalhos. O exemplo supõe que existam dados no intervalo A1:C20.

```
Sub SortRange1()

    Worksheets("Sheet1").Range("A1:C20").Sort _
        Key1:=Worksheets("Sheet1").Range("A1"), _
        Key2:=Worksheets("Sheet1").Range("B1")

End Sub
```

Este segundo exemplo classifica a região que contém a célula A1 (a região ativa) em Plan1, classificando pelos dados na primeira coluna e usando automaticamente uma linha de cabeçalho, se existir. O exemplo supõe que existam dados na região ativa, que inclui a célula A1. O método Sort determina a região ativa automaticamente.

```
Sub SortRange2()

    Worksheets("Plan1").Range("A1").Sort _
        Key1:=Worksheets("Plan1").Columns("A"), _
        Header:=xlGuess

End Sub
```

Método Offset

O método Offset é aquele um que você usará mais. Ele lhe permite mover para a direita, para a esquerda, para cima e para baixo. Por exemplo se você quiser mover 3 células para a direita, você escreverá:

```
Activecell.Offset(0,3).Select
```

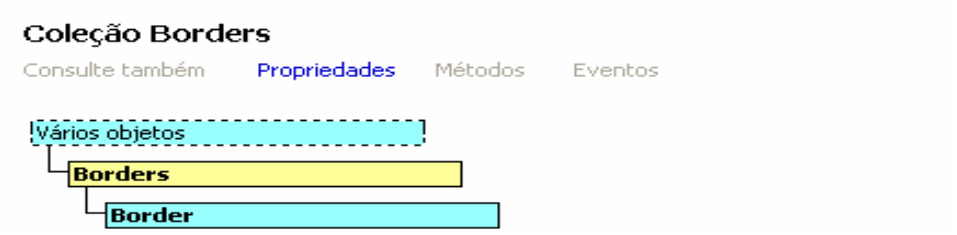
Se você quiser mover três células para baixo:

```
Activecell.Offset(3,0).Select
```

Se você quiser selecionar uma célula e mais uma outra a três linhas abaixo desta:

```
Range(Activecell,activecell.Offset(3,0)).Select
Range("A1",Range("A1").Offset(3,0)).Select
```

A Coleção Borders



A **coleção** Borders é responsável por determinar as caractereísticas das bordas de um determinado Range.

Uma coleção de quatro objetos **Border** representando as quatro bordas de um objeto **Range** ou **Style**.

Usando a coleção Borders

Use a propriedade **Borders** para retornar a coleção **Borders**, que contém todas as quatro bordas. O exemplo seguinte adiciona uma borda dupla à célula A1 na planilha um.

```
Worksheets(1).Range("A1").Borders.LineStyle = xlDouble
```

Use **Borders**(*index*), onde *index* identifica a borda, para retornar um único objeto **Border**. O exemplo seguinte define como vermelho a cor da borda inferior das células A1:G1.

```
Worksheets("Sheet1").Range("A1:G1"). _
    Borders(xlEdgeBottom).Color = RGB(255, 0, 0)
```

Index pode ser uma das seguintes constantes **XlBordersIndex**: **xlDiagonalDown**, **xlDiagonalUp**, **xlEdgeBottom**, **xlEdgeLeft**, **xlEdgeRight** ou **xlEdgeTop**, **xlInsideHorizontal** ou **xlInsideVertical**.

Junto com Borders, são passadas as constantes que representam qual borda está sendo modificada. São elas:

xlDiagonalDown – Diagonal de cima para baixo
xlDiagonalUp – Diagonal de baixo para cima
xlEdgeLeft – Borda esquerda
xlEdgeTop – Borda superior
xlEdgeBottom – Borda inferior
xlEdgeRight – Borda direita
xlInsideVertical – Linha interna vertical
xlInsideHorizontal – Linha interna horizontal.

Exemplos:

```
With Range(“A1:D4”).Borders(xlEdgeBottom)
      .ColorIndex = 3
      .LineStyle = xlDash
      .Weight = xlMedium
End With
```

Propriedades Color e ColorIndex

A propriedade Color especifica,em RGB, a cor da borda, enquanto ColorIndex especifica, em valor inteiro, o index para a paleta de cores. Por exemplo, Color = RGB(255,0,0) equivale a ColorIndex = 3.

Propriedades LineStyle e Weight

Possuem os mesmos parâmetros de LineStyle e Weight do método BorderAround.

O Objeto Font

O objeto Font possui toda a formatação de fonte que podemos fazer na interface gráfica do Excel. Vejamos as principais propriedades:
Bold – Propriedade Booleana que define o estilo negrito da fonte.
Color ou ColoIndex – Define a cor da fonte nos parâmetros anteriormente citados.
Italic – Propriedade Booleana que define o estilo itálico da fonte.
Name – Define o nome da fonte a ser usada.
Size – Define o tamanho da fonte.

O Objeto Interior

O objeto Interior define a formatação visual do fundo de uma célula. Suas principais propriedades são:

Propriedade Color e ColorIndex

Definem a cor de preenchimento de uma célula.

EXEMPLO – Colorindo Células Baseado No Seu Valor

A rotina seguinte colorirá uma célula numa seleção em vermelho se o valor da célula for maior do que 5

```
Sub testeCor( )
Dim celula As Range
For Each celula In Selection
      'testar um valor numérico primeiro
      If IsNumeric(celula.Value) Then
            'testar o valor da célula
            If celula.Value > 5 Then
                  'colorir se for maior do que 5
                  celula.Interior.ColorIndex = 3
            Else
                  'limpará a cor se não for maior do que 5
                  celula.Interior.ColorIndex = xlNone
            End If
End If
Next
End Sub
```

Propriedade Pattern

Define um padrão para ser aplicado no fundo de uma célula. Existem dezoito padrões que podem ser usados.

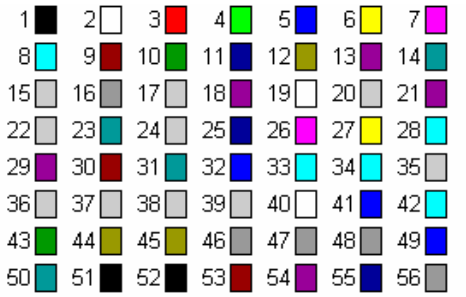
Propriedades PatternColor e PatternColorIndex

Retorna ou define a cor do padrão a ser aplicada no interior da célula como um índice na paleta de cores atual, ou como uma das seguintes constantes **XlColorIndex**: **xlColorIndexAutomatic** ou **xlColorIndexNone**. **Long** de leitura/gravação.

Defina essa propriedade como **xlColorIndexAutomatic** para especificar o padrão automático para células ou o estilo de preenchimento automático para objetos de desenho. Defina essa propriedade como **xlColorIndexNone** para especificar que você não deseja um padrão (isso é o mesmo que definir a propriedade **Pattern** do objeto **Interior** como **xlPatternNone**).

Comentários

A ilustração seguinte mostra os valores de índice de cor da paleta de cores padrão.



Exemplo

Este exemplo define a cor do padrão interior para o retângulo um em Sheet1.

```
With Worksheets("Sheet1").Rectangles(1).Interior
    .Pattern = xlChecker
    .PatternColorIndex = 5
End With
```

Para testar todos os valores, aqui está uma rotina interessante.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
On Error GoTo Trat
Dim Fundo, Padrao, CorPadrao As Integer
Fundo = InputBox("Por favor, digite o valor" & _
" da cor de preenchimento das células (0 - 56)")
Target.Interior.ColorIndex = Fundo
Padrao = InputBox("Por favor, digite o padrão desejado" & _
" para as células (0 - 18)")
Target.Interior.Pattern = Padrao
CorPadrao = InputBox("Por favor, digite o valor da cor" & _
" do padrão da célula (0 - 56)")
Target.Interior.PatternColorIndex = CorPadrao
Exit Sub

Trat:
    MsgBox "Você deve ter escolhido algum valor fora " & _
    "do Range especificado...", vbCritical
End Sub
```

Outro Exemplo

A macro seguinte somará todas as células na seleção que tenha uma cor de fundo(background) em vermelho.

```
Sub SomaDeCores()
      Dim somaInt As Single
      Dim celula As Range
      'inicializa a variável soma variable em zero (não exigido,
mas uma boa prática
      somaInt = 0
      'verifica cada uma das células na seleção que está também
no range usado.
      'Isto evita ter que verificar se há célula está vazia numa
      ' coluna inteira ou linha está selecionada.
      For Each celula In Intersect(Selection,
ActiveSheet.UsedRange)
            'o interior refere-se ao background de uma célula

If celula.Interior.ColorIndex = 3 Then
                  somaInt = somaInt + celula.Value
            End If
      Next celula
      'mostra os resultados
      MsgBox "a soma é " & somaInt
End Sub
```

Mais Um Exemplo – Colorindo Células Para Destacá-las Baseado Num Texto

O que segue mudará a cor de fundo (*background*) de cada linha que tenha o texto "conta" na coluna 1.

```
Sub ColorindoAs()
      Dim Ndx As Long
      For Ndx = 1 To ActiveSheet.UsedRange.Rows.Count
            If LCase(Cells(Ndx, 1).Value) = "conta" Then
                  Rows(Ndx).Interior.ColorIndex = 3
            End If
      Next Ndx
End Sub
```

Como fazer a sua macro esperar uns poucos segundos.

A declaração seguinte:

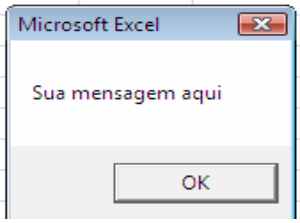
```
Application.Wait Now + TimeValue("00:00:05")
```

esperará por 5 segundos e daí rodará a próxima declaração.

Lição 6: Código VBA do Excel para Caixas de Mensagens

No VBA para Excel a caixa de mensagem é a principal ferramenta para interagir com o usuário. Você pode usá-la para informar, alertar-lhe ou pedir-lhe para escolher um certo path (Sim/Não).
O código no VBA para Excel para gerar a seguinte caixa de mensagem básica é:

```
MsgBox "Sua mensagem aqui"
```



Quando você desenvolver um procedimento que leva tempo para rodar, você encerra seu código com uma caixa de mensagem básica dizendo ao usuário que o procedimento foi executado.

```
MsgBox "O relatório está pronto"
```

Ou

```
MsgBox "O procedimento foi executado você deve agora ir para a folha ""Capa"""
```

Ou

```
MsgBox "O procedimento foi executado você deve agora ir para a célula ""A1"""
```

Note que se você quiser parte da sua mensagem esteja entre aspas você tem de dobrar as aspas dentro da mensagem. Quando a usuário clicar num botão na caixa de mensagem Excel ela dispara um procedimento que fará alguma coisa importante como deletar todos os dados. Sempre iniciar o código com uma caixa de mensagem que pergunte ao usuário se ele tem certeza que ele quer que o procedimento rode. Neste caso use uma caixa de mensagem "Sim/Não".

Quando trabalhando com uma caixa de mensagem " Sim/Não" você precisará coletar a resposta dada pelo usuário.

Passo 1: Você primeiro declara uma variável para armazenar a resposta

"`Dim varResposta as string`".

Passo 2: Você guarda a resposta na variável com o código:

```
 "varResposta = MsgBox("Você está certo que você quer deletar todos os dados?", vbYesNo, "Advertência")".
```

Note os parênteses dentro do qual você submete os três argumentos; a **mensagem**, o **tipo** de caixa de mensagem e seu **título**. Note também as aspas envolvendo a mensagem e o título.

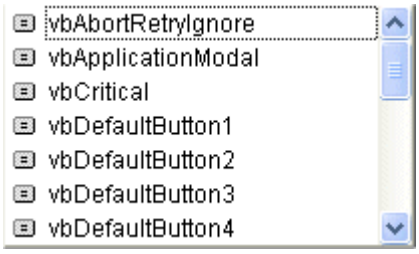
Passo 3: Você trata a resposta com uma declaração "IF" onde se o usuário responder "Não" você sai do procedimento com "Exit Sub"
Aqui está o procedimento inteiro.

```
Sub proProcedimentoPerigoso()

Dim varResposta As String

    varResposta = MsgBox("Você está certo que você quer deletar
todos os dados?", vbYesNo, "Advertência")
    If varResposta = vbNo Then
        Exit Sub
    End if
    Aqui inicia o procedimento perigoso de deletar os dados
End Sub
```

Quando você criar uma caixa de mensagem básica (primeiro exemplo acima), você não precisa usar parênteses. Se você desenvolver uma caixa de mensagem mais complexa você verá que quando você iniciar escrevendo a linha de código (exemplo acima) varAnswer = MsgBox**(,** logo após os parênteses o Excel mostra-lhe o que é esperado como argumentos:

MsgBox(***Prompt***, [*Buttons As* VbMsgBoxStyle = vbOKOnly], [*Title*], [*HelpFile*], [*Context*]) As VbMsgBoxResult

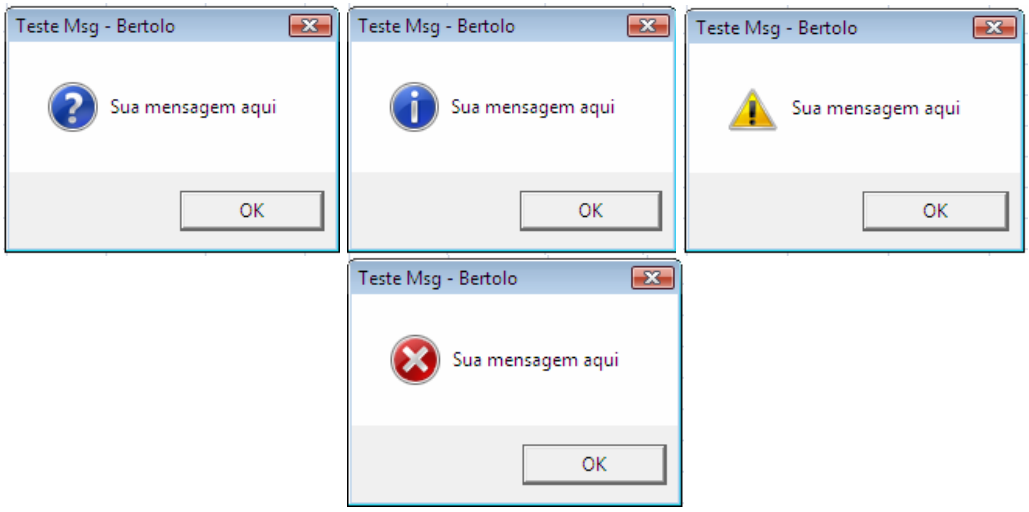
1- O ***Prompt*** não é opcional, é a mensagem que você quer que seu usuário leia na caixa de mensagem e ela deve estar entre aspas.
2- Quando você entrar com vírgula após o prompt, o Excel oferece-lhe uma lista *drop-down* de todos os tipos de caixa de mensagem que você pode criar.



Escolha a sua caixa de mensagem e entre com uma vírgula.
3- O terceiro argumento é o título da caixa de mensagem que aparecerá na faixa azul no topo da caixa de mensagem. Submeta um título entre aspas.

O quarto e quinto argumentos raramente são usados, eles são opcionais e você não precisa entrar com qualquer coisa então...mas feche os parênteses.

Como você pode ver, existem muitos tipos de caixas de mensagens. Aquelas que são mais freqüentemente usadas são a vbYesNo, vbOKCancel, vbRetryCancel. Você deverá também querer usar uma caixa de mensagem mais decorativa como a vbQuestion, a vbInformation, a vbExclamation ou a vbCritical como mostrado abaixo.



Não existem caixas de mensagem no VBA para Excel que você possa mostrar ao usuário enquanto um procedimento roda por um tempo muito longo. Mas há um modo de contornar isto. Adicione uma planilha chamada "Espera" e insira uma figura, um *wordart* ou uma mensagem numa fonte muito grande como "Procedimento rodando" e oculte a folha.

Passo #1: Primeiro você torna a folha que você criou visível e selecione-a (Linhas 1 e 2).
Passo #2: Você deve querer mudar o cursor para a ampulheta (Linha 3)
Passo #3: Você desativa a funcionalidade de atualização de tela (Linha 4)
Passo #4: Você escreve o seu longo procedimento (Linha 5 no exemplo)
Passo #5: Você oculta a folha especial (Linha 6)
Passo #6: Você traz de volta o cursor *default* (Linha 7)
Passo #7: Você re-ativa a funcionalidade atualização de tela

Adicione este código ao seu longo procedimento. Apenas troque a linha "Aqui está seu longo procedimento" pelo seu próprio procedimento.:

```
Sub proProcedimentoLongo()
    Sheets("Espera").Visible=True
    Sheets("Espera").Select
    Application.Cursor = xlWait
    Application.ScreenUpdating=False
    Aqui está o longo procedimento
    Sheets("Espera").Visible=False
Application.Cursor = xlDefault
    Application.ScreenUpdating=True
End Sub
```

Mostrando Caixas de Mensagens

O que segue mostra uma caixa de mensagem que tem apenas um botão **OK** nela.

```
Sub MsgExemplo1()
      MsgBox "Este é um exemplo de uma mensagem."
End Sub
```

O que segue mostra uma caixa de mensagem que mostra os botões **OK** e **Cancelar**, permitindo o usuário fornecer uma resposta:

```
Sub MsgExemplo2()
      Dim iResposta As Integer
      iResposta = MsgBox("Selecione OK ou Cancelar", vbOKCancel)
      'verifique para ver se o botão cancelar foi selecionado
      If iResposta = vbCancel Then
            MsgBox "Você selecionou Cancelar"
      End If
      If iResposta = vbOK Then
            MsgBox "Você selecionou OK"
      End If
End Sub
```

Este exemplo também poderia ter sido escrito de uma maneira muito mais condensada, onde a seleção do botão **Cancelar** pára a execução e a seleção do botão **OK** permite continuá-la:

```
Sub MsgExemplo3()
      If MsgBox("Selecione OK para Continuar") = vbOKCancel Then
End
'Comandos a serem executados se OK for selecionado
End Sub
```

O que segue ilustra uma caixa de mensagem que mostra os botões **Sim**, **Não** e **Cancelar**. Se você quiser somente os botões **Sim** ou **Não**, então use vbYesNo em vez de vbYesNoCancel

```
Sub MsgExemplo4()
      Dim iResposta As Integer
      'mostrar uma mensagem e armazenar uma resposta para
      'avaliação
      iResposta = MsgBox("Selecione um botão", vbYesNoCancel)
      'verifique para ver se o botão Cancelar foi selecionado
            If iResposta = vbCancel Then
            MsgBox "Você selecionou Cancelar"
            End If
      'verifique para ver se Sim foi selecionado
            If iResposta = vbYes Then
            MsgBox "Você selecionou Sim"
            End If
      'verifique e veja se Não foi selecionado
            If iResposta = vbNo Then
            MsgBox "Você selecionou Não"
            End If
End Sub
```

Formatando numa Caixa de Mensagem

O que segue ilustra como formatar entradas numa MsgBox:

```
MsgBox " Equipamento Nº.     " & numEquip & Chr(13) _
     & " Descrição  -   " & Descrição &
 Chr(13) _
     & " Prazo Interno  -   " & Format(partedaData,"ddd-mm")
```

onde as variáveis `numEquip`, `Descrição`, e `partedaData` já foram definidas.

Acima o Chr(13) é a tecla enter, permitindo mostrar a entrada na próxima linha. Outros valores úteis para Chr() são 174 que é um símbolo *copyright* e 34 que é uma aspa dupla.

Usando Aspas Duplas Numa Caixa de Mensagem

O principal uso das aspas duplas no seu código é usá-lo para delimitar *strings* de texto. Por exemplo,

```
MsgBox "Alô Mundo"
```

ou

```
ActiveCell.Value = "Descrição"
```

Se você quiser usar aspas duplas para destacar texto numa caixa de mensagem, então use uma das seguintes aproximações:

```
MsgBox "O mundo" & Chr(34) & "AOOO" & Chr
(34) & "está em aspas duplas"
```

ou

```
MsgBox "O mundo ""AOOO"" está em aspas duplas"
```

O segundo exemplo funciona porque as duas aspas duplas numa linha é interpretada como parte da *string* de texto e, não como delimitação de uma *string* de texto.

Mais exemplos de uso das caixas de mensagens

Aqui estão alguns exemplos de como se usar a função MsgBox:

```
MsgBox "O trabalho está feito !"
' caixa de mensagem com texto e botão OK
MsgBox "O trabalho está feito !", vbInformation
' caixa de mensagem com texto, botão OK e um ícone de informação
MsgBox "O trabalho está feito !", vbCritical
' caixa de mensagem com texto, botão OK e um ícone de advertência
MsgBox "O trabalho está feito !", vbInformation, "Meu Título"
' caixa de mensagem com texto, botão OK, ícone de informação e um
texto de título personalizado.

Resposta = MsgBox("Você quer continuar ?", vbYesNo)
' caixa de mensagem com botões YES e NO,
' o resultado é um inteiro, as constantes são nomeadas vbYes e vbNo.
Resposta = MsgBox("Você quer continuar ?", vbYesNo + vbQuestion)
' caixa de mensagem com botões YES e NO e um ícone de interrogação
Resposta = MsgBox("Você quer continuar ?", vbYesNo + vbQuestion, "Meu
Título")
' caixa de mensagem com botões YES e NO,
' ícone de interrogação e um texto de título personalizado
```

```
Resposta = MsgBox("Você quer continuar ?", vbYesNo + 256 + vbQuestion,
"Meu Título")
' caixa de mensagem com botões YES e NO, ícone de interrogação e um
texto de título personalizado,
' o botão NO é o default
Resposta = MsgBox("Você quer continuar ?", vbOKCancel, "Meu Título")
' caixa de mensagem com botões OK- e CANCEL, o resultado é um inteiro,
' as constantes são nomeadas vbOK ou vbCancel.
```

O resultado da função MsgBox pode ser armazenado numa variável. A variável pode ser do tipo Integer. Esta variável pode ser usada mais tarde num código de macro como este:

```
Resposta = MsgBox("Você quer continuar ?", _
    vbOKCancel, "Meu Título")
If Resposta = vbCancel Then Exit Sub ' a macro termina se o
usuário selecionar o botão CANCEL
```

Ou como este:

```
If MsgBox("Você quer continuar ?", vbOKCancel, _
    "Meu Título") = vbCancel Then Exit Sub
```

Como Formatar Uma Mensagem Numa Caixa de Entrada Ou Caixa de Mensagem

Se você usar Chr(13) no seu texto MsgBox, então isto atua como um alinhamento do papel, colocando o texto seguinte ao Chr(13) numa nova linha numa MsgBox.

```
Sub ExemploMultiLinhas ()
MsgBox "Esta é a primeira linha." & _
               Chr(13) & "Esta é a segunda linha." & _
               Chr(13) & "Esta é a terceira linha"
End Sub
```

Se você quer que uma linha seja indentada (afastada da margem), então use um código como o seguinte:

```
Sub ExemploIndentar ()
MsgBox "Esta é a primeira linha." & _
          Chr(13) & Chr(9) & "Esta é indentada." & _
          Chr(13) & Chr(9) & "Esta é também indentada"
End Sub
```

Adicionando Um Botão de Ajuda A Uma MsgBox

O botão de ajuda numa caixa de mensagem pode somente ser mostrado usando a seguinte aproximação. É possível que erros fixados pelo Microsoft podem consertar isto, mas isto é necessário para usuários que não fizeram. Você tem que adicionar um botão ajuda explicitamente com os argumentos de botões, como este:

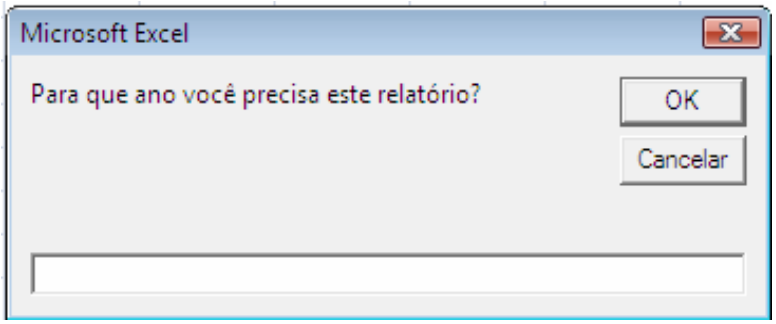
```
MsgBox "Clique Ajuda para ajuda?", _
```

```
            vbCritical + vbMsgBoxHelpButton, "Ajuda",
"runner.hlp", 10
```

Isto mostrará um arquivo ajuda chamado **runner.hlp** que está no diretório Windows (assumindo que ele exista). Por favor note que o número usado como o último argumento é dependente do arquivo ajuda.

Lição 7: Código para Caixas de Entrada VBA do Excel

Caixas de entrada de dados são usadas para exigirem um ÚNICO valor do usuário.

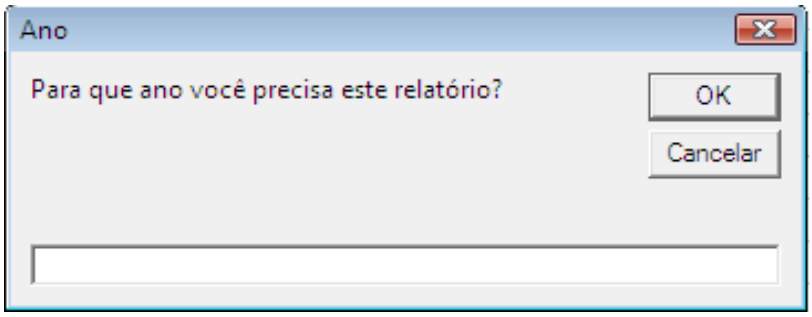


O código básico para um a caixa de entrada de dados é como segue. Com este procedimento sempre que o valor submitido pelo usuário é entrado na célula "A1" de uma folha chamada "Intro".

```
Sub proInput()
    Sheets("Intro").Range("A1").Value = InputBox("Para que ano
você precisa este relatório?")
End Sub
```

Você também pode usar um cabeçalho diferente para a caixa de entrada de dados (como "Ano" neste exemplo)

```
Sheets("Intro").Range("A1").Value = InputBox("Para que ano você
precisa este relatório?",”Ano”)
```



Antes de ir adiante, devo dizer-lhe que não é possível impor um caractere de password (*) na caixa de entrada de dados. Para conseguir isto você deve trabalhar com um userform e uma caixa de texto. Veja como na lição sobre userforms e controles. Se você quiser validar o valor submitido pelo usuário antes de continuar com o procedimento você não armazenará o valor diretamente numa célula mas numa variável. O código deverá então se parecer com este:

```
Sub proEntrada()
Dim varEntrada As Integer

    varEntrada = InputBox("Para que ano você precisa este
relatório?")
    Sheets("Intro").Range("A1").valor = varEntrada

End Sub
```

Uma variável "varEntrada" é declarada Dim varEntrada As Integer para receber o valor submetido pelo usuário. O usuário entra com um valor que é transferido para uma variável e daí para uma célula. Note aqui que a variável é declarada como "Integer" assim se o usuário submete um texto "string" um erro é gerado. Você pode declarar a variável como string Dim varEntrada As String se você estiver esperando um texto ou como date Dim varEntrada As Date se você estiver perguntando uma data.

Com esta abordagem você pode usar um manipulador de erro ou alguma declaração "IF" para validar o valor submetido pelo usuário antes de seguir em frente. Aqui está o código para manipular erros (resposta do usuário cancelada ou inválida) e validar o valor submetido:

```
Sub proEntrada()
Dim varEntrada As Date
    On Error GoTo adErro
    varInput = InputBox("Para que ano você precisa este
relatório?")
     If year(varEntrada)> 2006 then
        MsgBox "Ano inválido"
        Exit Sub
      End If
     Faça isto e aquilo.
     Exit Sub
adErro:
    MsgBox "Isto não é uma data."
End Sub
```

Neste procedimento eu primeiro declaro uma variável do tipo data Dim varEntrada As Date de modo que se o usuário submeter alguma coisa além de uma data lá será um erro VBA a ser manipulado. Eu digo então ao VBA que se um erro ocorrer o procedimento deverá pular para os endereços adErro On Error GoTo adErro**.** Isto significa que se o usuário submeter qualquer coisa além de uma data o procedimento saltará todas as linhas até chegar em adErro. Uma caixa de mensagem dirá então ao usuário que aquilo que ele submeteu não é uma data MsgBox "Isto não é uma data.". Se aquilo que é submetido for de fato uma data eu uso uma declaração "IF" para verificar se o ano da data é maior que 2006. Se ele for há uma outra caixa de mensagem Msgbox "Ano inválido**"** seguido por um Exit Sub que coloca um final ao procedimento. Se o valor submetido for uma data e o ano é válido o procedimento roda completamente até ele atingir o outro Exit Sub logo antes de adErro**:** (com a vírgula no final).

Interrompendo Uma Macro Para Entradas

O que segue é uma macro que se interrompe até que o usuário entre com um valor. O valor é então escrito na célula ativa. Se o usuário pressionar Cancel, a macro entra em loop.

```
Sub ObterEntrada()
     Dim str As String
     Do
          str = InputBox("Entre com alguma coisa")
          If str <> "" Then
```

```
                ActiveCell.Value = str
                Exit Do
            End If
        Loop
End Sub
```

Você pode colocar os seguintes passos diretamente antes da declaração Loop para ver se o usuário quer continuar:

```
If MsgBox("Você quer continuar", vbYesNo, _
                    "Continuar?") = vbNo Then End
```

RESTRINGINDO O QUE É PERMITIDO NUMA CAIXA DE ENTRADA

Se você usar Application.InputBox para mostrar uma caixa de entrada, você pode restringir o que é permitido ao usuário entrar especificando um valor para o tipo de argumento. Por exemplo, o que segue restringe o usuário a fornecer apenas números:

```
Dim entradaResposta As Variant
Dim lNum As Single
'use uma variável variant para obter a resposta da caixa de
'entrada quando a resposta puder ser Boolean (falsa) ou um
'valor
entradaResposta = Application.InputBox( _
                          Prompt:="Entre com um número",
Type:=1)
If entradaResposta = False Then Exit Sub
'armazena a entrada do usuário numa variável numérica.
'Note que a caixa de entrada retorna uma string mesmo se o tipo
'for  1
lNum = Val(entradaResposta)
MsgBox "você entrou com " & lNum
```

O que segue usa Application.InputBox para obter um intervalo selecionado pelo usuário, ajustando o tipo de argumento para 8.

```
'A variável para a resposta da caixa de entrada deve ser do tipo
'range ou então o teste Is Nothing explodirá!
Dim userSelection As Range
'ligue o manipulador de erro no caso do usuário pressionar
'cancel
On Error Resume Next
Set userSelection = Application.InputBox( _
                prompt:="entre com um range", Type:=8)
On Error GoTo 0
If userSelection Is Nothing Then
'mostrar esta menssagem se o botão cancel foi selecionado
     MsgBox "Um range não foi selecionado"
Exit Sub
End If
'mostrar o range selecionado
MsgBox "Você selecionou " & userSelection.Address
(external:=True)
```

```
'para ir ao range selecionado
Application.Goto userSelection, True
```

Os valores mais populares para o tipo de argumento são:

1 Um número
2 Texto (uma string)
4 Um valor lógico (True ou False)
8 Uma referência de célula, como um objeto Range

Para maiores informações sobre a caixa de entrada, destacar a palavra no Visual Basic Editor e pressionar F1 para ajuda.

CONVIDANDO O USUÁRIO A ENTRAR COM UM NÚMERO

Você pode conseguir que o usuário forneça-lhe um número usando o código como o seguinte.

```
Sub GetANumber()
      Dim userInput As Variant
      'mostrar a aplicação caixa de entrada, com o tipo ajustado
      'para 1 a qual permite somente entradas de números
      userInput = Application.InputBox( _
                  prompt:="Por favor entrar com um número",
Type:=1)
      'se o botão cancel for selecionado, o TypeName é Boolean
      If TypeName(userInput) = "Boolean" Then
            Exit Sub
      End If
      'converte a entrada do usuário de uma string para um número
userInput = Val(userInput)
      'mostra a entrada do usuário e o TypeName
      MsgBox userInput & " " & TypeName
(userInput)
End Sub
```

Se o Tipo de argumento não for especificado, então o usuário poderá entrar com qualquer string, não apenas números. Se Application.InputBox não for usado, mas somente a InputBox for usada, não há maneira de distinguir entre um Zero e a seleção cancelar do usuário. Finalmente, toda entrada de uma caixa de entrada é uma *string*, e deve ser convertida para um número se ela é um número. Por outro lado, testes numéricos não funcionarão.

USANDO A FUNÇÃO APPLICATION.INPUTBOX PARA ESPECIFICAR UM NÚMERO

A função Application.InputBox pode ser usada par restringir entradas do usuário apenas a números especificando um tipo de argumento quando você mostrar a caixa de entrada caixa de entrada. O que segue ilustra isto:

```
Sub Obter_Um_Numero()
      Dim RespostaDoUsuario As Variant
```

```
      'mostrar a caixa de entrada
      RespostaDoUsuario = _
            Application.InputBox(Prompt:="Entre com um número", _
                                 Type:=1)
      'verificar e ver se botão cancelar foi selecionado.  Sair
      'se for
      If RespostaDoUsuario = "False" Then Exit Sub
      'mostrar uma mensagem mostrando qual número foi entrado
      'use Val() para converter a um número ao invés de uma
      '"string number"
      MsgBox "O número entrada foi " & Val(RespostaDoUsuario)
End Sub
```

No exemplo acima, a variável que recebe o resultado da declaração Application.InputBox deve ser uma variável Variant. Deste modo, ela pode acitar ou um número ou uma resposta False se o botão cancelar for selecionado.

Uma outra maneira de se determinar se o botão cancelar foi selecionado na Application.InputBox é usar um teste TypeName():

```
If TypeName(RespostaDoUsuario) = "Boolean" Then Exit Sub
```

INPUTBOX PARA PEDIR POR UMA DATA

Tente o código seguinte:

```
Sub ObterUmaData()
      Dim AString As String
      Dim RowNdx As Integer
      Dim AData As Double
      AString = Application.InputBox("Entrar Com Uma Data
Inicial")
      If IsDate(AString) Then
            AData = DateValue(AString)
      Else
            MsgBox "Data inválida"
      End If
End Sub
```

Note que você deveria validar a variável AData para confirmar que ela está dentro do intervalo de data que você quer.

USANDO A INPUTBOX DO VISUAL BASIC PARA RETORNAR UM RANGE

Existem dois tipos de **input boxes** no Excel. Há a função Visual Basic InputBox e ela retorna somente uma *string* texto. Ela é a mais simples para se usar. Há também a Excel Application.InputBox que permite-lhe especificar o tipo de entrada que é retornada. O que segue ilustra o uso da função VB InputBox:

A abordagem mais simples com nenhuma verificação de erro – assume que o usuário não confundirá a seleção ou não sairá se uma caixa de erro aparecer.

```
'declarar uma variável Variant para a saída da InputBox
Dim meuObjeto As Variant
meuObjeto = InputBox("Entrar com uma célula")
Range(meuObjeto).Select
```

Entretanto, você deverá considerar se o usuário acionou a tecla cancelar que retorna False ou entrou com um endereço inválido. O que segue mostra como fazer isto:

```
Sub ExemploInputBox()
      Dim meuObjeto As Variant, celulasSelecionadas As Range
      'retorna uma label no caso de um intervalo inválido ser
      'entrado
ObterUmaCelula:
      meuObjeto = InputBox("Entrar uma célula")
      'Se nenhuma entrada for feita na caixa, mesmo se OK
      'cancelou, sair do procedimento
      If meuObjeto = "" Then
            Exit Sub
      Else
      'ligar error handling no caso da entrada não ser um range
            On Error GoTo ErrorHandler
      'armazenar a entrada numa variável range para usar mais
      'tarde
            Set celulasSelecionadas = Range(meuObjeto)
            'desligar error handling
            On Error GoTo 0
      End If
      'restante do seu código
Exit Sub
      'a sub exit acima evita entrar no seguinte error handler
      'que é usado se a entrada não for um range válido
ErrorHandler:
      MsgBox "Este não era um endereço de célula válido"
      Resume ObterUmaCelula
End Sub
```

COMO OBTER UM ENDEREÇO DE CÉLULA DE UM USUÁRIO

Use o método Application.InputBox, com o argumento Type ajustado para 8:

```
Dim ORange As Range
On Error Resume Next
Set ORange = Application.InputBox _
                (prompt:="Selecione uma célula",type:=8)
On Error GoTo 0
If ORange Is Nothing Then
      MsgBox "Você não selecionou uma célula"
End If
'restante do seu código
```

USANDO INPUTBOXES PARA OBTER UM RANGE DE CÉLULAS

O que segue são dois exemplos que mostram como o usuário conseguir selecionar apenas um intervalo de célula simples. Os dois exemplos após estes dois permitem o usuário selecionar qualquer número de células.

```
Sub ExemploDeEntrada1()
      'Este exemplo mostra uma caixa de entrada e pede ao usuário
      'para selecionar uma célula.
      Dim celulasSelecionadas As Range
      'configurar on error no caso de cancelar ter sido
      'selecionado
      On Error Resume Next
      'mostrar a caixa de entrada e atribuir resultados de
      'seleção para uma variável
Set celulasSelecionadas = Application.InputBox( _
            prompt:="Selecione uma célula simples", Type:=8)
      'desligar o verificador de erro
      On Error GoTo 0
      'verificar se um intervalo foi selecionado
      If celulasSelecionadas Is Nothing Then
            MsgBox "Nenhuma célula foi selecionada"
            Exit Sub
      ElseIf celulasSelecionadas.Cells.Count > 1 Then
            'verificar e ver quantas células foram selecionadas
            'mostrar mensagens dando o resultado
MsgBox "Você selecionou mais do que uma célula"
            Exit Sub
      Else
            MsgBox "Você selecionou " &
                  celulasSelecionadas.Address(external:=True)
      End If
End Sub

Sub InputExample2()
      'Este exemplo mostra uma caixa de entrada e pede ao usuário
      'para selecionar uma célula
      'Ele continua o laço até o botão cancel ser selecionado ou
      'uma célula simples for selecionada
      Dim celulasSelecionadas As Range
      'ajuste o on error no caso do botão cancelar ser
selecionado
      On Error Resume Next
      Do
      'mostrar a caixa de entrada e atribuir o resultado da
      'seleção a uma variável
         Set celulasSelecionadas = Application.InputBox _
          (prompt:="Selecione uma célula simples", Type:
=8)
            'desligar o verificador de erro
            On Error GoTo 0
```

```
            'verificar se um intervalo é selecionado. Parar toda
            'ação se nenhum for selecionado
            If celulasSelecionadas Is Nothing Then

End
            'verificar e ver quantas células foram selecionadas
            ElseIf celulasSelecionadas.Cells.Count > 1 Then
                  MsgBox "Você selecionou mais do que uma célula"
            Else
                  'sair se uma célula for selecionada
                  Exit Do
            End If
      Loop
      'mostrar o valor da célula selecionada
      MsgBox "o valor da célula selecionada é " _
                  & celulasSelecionadas.Value
End Sub

Sub InputExample3()
      'Este exemplo mostra uma caixa de entrada e pede ao usuário
      'para selecionar um intervalo
      Dim cellsSelected As Range
      'configure on error no caso do botão cancelar ter sido
      'selecionado
      On Error Resume Next
      'mostra a caixa de entrada e atribui um resultado da
      seleção 'a uma variável
      Set cellsSelected = Application.InputBox( _
            prompt:="Selecione um intervalo de uma ou mais
células",Type:=8)
      'desligar o verificador de erro
On Error GoTo 0
      'verificar se um range é selecionado
      If cellsSelected Is Nothing Then
            MsgBox "Nenhuma célula foi selecionada"
            Exit Sub
      Else
            'mostrar mensagens dando as células selecionadas
            MsgBox "Você selecionou " &
                  cellsSelected.Address(external:=True)
      End If
End Sub

Sub InputExample4()
      'Este exemplo mostra uma caixa de entrada e pede ao usuário
      'para selecionar um intervalo
      'Ele continua o laço até o botão cancel ser selecionado ou
      'um intervalo ser selecionado
      Dim cellsSelected As Range, cell As Range
      'configure on error no caso de cancel ser selecionado
      On Error Resume Next
```

```
      Do
            'mostra a caixa de entrada e atribui o resultado da
            'seleção a uma variável
            Set cellsSelected = Application.InputBox _
                  (prompt:="Selecione uma célula simples", Type:
=8)
            'desligue o verificador de erro
On Error GoTo 0
            'verifique se um intervalo é selecionado. Pare toda a
            'ação se nenhum for selecionado
            If cellsSelected Is Nothing Then
                              End
            Else
                  'sair quando um range tiver sido selecionado
                  Exit Do
            End If
      Loop
      'mostrar o valor das células selecionadas
      For Each cell In cellsSelected
            MsgBox "o valor da " cell
.Address(external:=True) _
                  & " é " & cell.Value
End Sub
```

UM EXEMPLO DE APPLICATION.INPUTBOX QUE OBTÉM UM INTERVALO

O que segue ilustra como usar a função Application.InputBox.
Para obter uma seleção de intervalo pelo usuário.

```
Dim Rng As Range
On Error Resume Next
Set Rng = Application.InputBox(prompt:="Entrar Com Um Range",
Type:=8)
If Rng Is Nothing Then
      MsgBox "Nenhum Intervalo Selecionado"
Else
      Rng.Select
End If
On Error GoTo 0
```

USANDO A INPUTBOX PARA COLOCAR UM VALOR NUMA CÉLULA

O que segue ilustra como obter um valor do usuário e colocá-lo na célula A3 da planilha ativa:

```
Sub InputBoxExample()
      Dim cellValue As Variant
reShowInputBox:
      cellValue = Application.InputBox("Entrar com o valor e ir
para A3")
      If cellValue = False Then
            Beep
            Exit Sub
```

```
      ElseIf cellValue = "" Then
            Beep
            GoTo reShowInputBox
      Else
            ActiveSheet.Range("A3").Value = cellValue
      End If
End Sub
```

CONVIDANDO O USUÁRIO PARA MÚLTIPLAS ENTRADAS

O que segue é um modo simples de convidar um usuário para uma série de entradas e colocar os valores nas células da planilha ativa. Ele usa uma das rotinas principais que chama novamente uma sub-rotina repetidamente.

```
Sub MainProcedure()
      LoadData "A1", "Entrar com um valor para alguma coisa"
      LoadData "G1", " Entrar com um valor ainda para alguma
coisa"
End Sub

'os argumentos são endereços de células para o valor e a
'mensagem a ser mostrada
Sub LoadData(addr As String, msg As String)
      Dim resposta As Variant, iR As Integer
      'fazer o laço até o usuário entrar ou escolher.
      While response = ""
      'mostra uma caixa de entrada com a msg que foi transmitida
      'à esta rotina
resposta = InputBox(prompt:=msg)
            'se o botão cancelar for selecionado ou nenhum valor
            'for entrado, ver ser o 'usuário quer parar.
            If resposta = "" Then
                  iR = MsgBox("Nenhum valor foi entrado.  " & _
                                    "Você quer parar?",
vbYesNo)
                  'isto pára toda atividade
                  If iR = vbYes Then End
            End If
      Wend
      'isto carrega o valor na célula
      Range(addr).Value = resposta
End Sub
```

Exemplo da função InputBox

Este exemplo mostra várias maneiras de utilizar a função **InputBox** para solicitar ao usuário que digite um valor. Se as posições x e y forem omitidas, a caixa de diálogo será automaticamente centralizada em relação aos respectivos eixos. A variável `MeuValor` contém o valor digitado pelo usuário se ele clicar em **OK** ou pressionar a tecla ENTER. Se o usuário clicar em **Cancel**, será retornada uma seqüência de comprimento zero.

```
Dim Mensagem, Titulo, Default, MeuValor
Mensagem = "Digite um valor entre 1 e 3"    ' Define o aviso.
Titulo = "Demonstração da CaixaDeEntrada"    ' Define o título.
Default = "1"    ' Define o padrão.
' Exibe a mensagem, o título e o valor padrão.
MeuValor = InputBox(Mensagem, Titulo, Padrão)

' Utiliza o arquivo de Ajuda e o contexto. O botão Ajuda é adicionado
' atomaticamente.
MeuValor = InputBox(Mensagem, Titulo, , , , "DEMO.HLP", 10)

' Exibe a caixa de diálogo na posição 100, 100.
MeuValor = InputBox(Mensagem, Titulo, Padrão, 100, 100)
```

Lição 8: Código para Manipular Erros no VBA para Excel

Existem dois tipos de erros que podem ocorrer quando se trabalha com VBA para Excel, o erro VBA e o erro Excel. Os erros VBA são aqueles capturados pelo VBA que disparam a janela de diálogo de erro e que são gerados quando existem erros de impressão ou erros lógicos no código. O erro Excel são aqueles que o VBA não captura e que você manipula com as declarações "IF".

Manipulando Erros no Excel

Um erro Excel acontece quando por exemplo você estiver pedindo a um usuário para submeter um número numa célula A1 de uma planilha e que o número submetido é ou muito pequeno ou muito grande. No seguinte exemplo você está esperando um usuário submeter um número entre 100 e 200. Se o valor for menor do que 100 e, você quiser dizer ao usuário mas você concorda em deixar o procedimento VBA ir adiante. Se o valor for maior do que 200 e, você quiser dizer ao usuário e parar o procedimento. Aqui está o código:

```
Sub proTestExcelErrors()
     If Range("A1").Value < 100 Then
         MsgBox "Valor submetido é muito pequeno."
    ElseIf Range("A1").Value > 200 Then
         MsgBox "Valor submetido é muito grande."
         Exit Sub
    End If
     ' aqui está o resto do procedimento.
         MsgBox "O procedimento vai em frente."
End Sub
```

Erros no VBA para Excel

Um VBA erro acontece quando o VBA não entende ou aceita que você tem escrito. Um erro VBA acontece quando:
- o nome de um objeto, uma propriedade ou um método tem a ortografia incorreta,
- um método inválido ou propriedade são chamados para um certo objeto. Por exemplo você não pode escrever ActiveSheet.Value se a folha não tem um valor. Ela tem um nome, ele pode ser visível ou não mas ele não tem valor.
- um valor inválido é enviado para uma variável. Por exemplo você não pode guardar um texto ("Bertolo") numa variável que você declarou como Integer (Dim varVariable as Integer).
- um valor não é encontrado pelo método Find (Selection.Find...). Por exemplo, você está procurando "Bertolo" na coluna A mas ele não está lá.
- uma variável não foi declarada enquanto trabalhando com a Option Explicit ativada.
- e mais.

É extremamente importante manipular estes erros para programadores que desenvolvem programas enormes ou video games que serão distribuídos numa larga escala. Isto é porque existem enormes equipes de teste para testar e tentar encontrar bugs nos programas antes deles serem lançados para o público. E mesmo assim.....Microsoft Internet Explorer....segurança bugs....

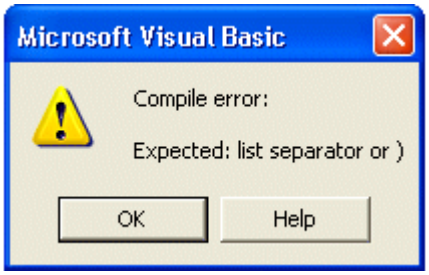
Assim o primeiro passo na manipulação de erros é testar seu programa.

Quando você estiver desenvolvendo procedimentos VBA no Excel para si próprio não é realmente importante manipular erros. Você trabalha com seu programa e corrige-os quando aparecerem. Se você desenvolver programas para umas poucas pessoas com quem tem ligações estretiras você pode esquecer a respeito de erros também. Mas se você estiver desenvolvendo programas para distribuição em grande escala você deve manipular os erros lembrando de adicionar um pouco do custo de sua aplicação. Você pode mesmo formar a sua própria equipe de teste. **Descoberta de Erros VBA enquanto estiver Digitando o Procedimento, enquanto estiver Tentando Rodar o Procedimento ou enquanto estiver Rodando o Procedimento**

Uma porção de erros são capturados pelo Visual Basic Editor (VBE) e você não é alertado quando você escrever o código mas quando você tentar obter a próxima linha enquanto da escrita do código. A fonte da linha torna-se vermelha e a caixa de mensagem lhe diz o que está errado. Outros erros não são detetados quando você escrever o código mas quando você tentar rodar seu procedimento VBA ou quando você rodá-los.

Enquanto digita

O VBE aponta um monte de erros quando você escreve o código. Tente escrever `Range("A1".Value` por exemplo. Você será alertado para o fato que um parênteses está faltando com a caixa de mensagem dizendo "Expected: End of Statement" (imagem abaixo). Você será alertado pela falta de parênteses, aspas duplas, dois pontos, pontos e outro "separadores". Mas o VBE não lhe alertará pelos erros de impressão como `Range("A1").Value` ou qualquer coisa que esteja entre aspas duplas `Range("namedoCampo").Value`. O VBE não lhe alertará se você escreveu de forma errada o nome de uma variável.

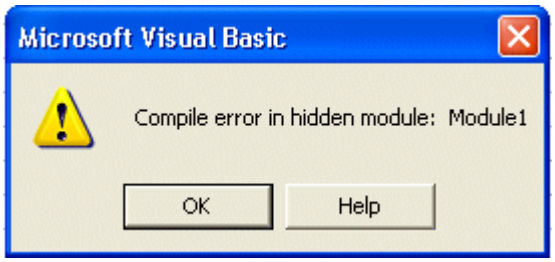


Você corrige estes erros e daí você pode rodar suas macros passo a passo com a tecla F8 ou indo ao menu "Ferramentas/Macro/Macros". Até este ponto outro erroVBA será identificado pelo VBE.

Enquanto tenta Rodar o Procedimento

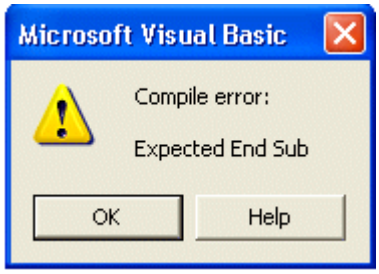
O segundo conjunto de erros é identificado pelo VBE tão logo você tentar rodar o procedimento. Se existirem alguns erros específicos no código, a execução não começará. Uma janela de diálogo diferente aparece dependendo se você estiver rodando o procedimento do VBE do Excel ou do Excel quando o procedimento for protegido com password. O tipo de erro nesta segunda categoria inclui a ausência de `End Sub`, uma falta de `End If`, uma falta de `Next` ou `Loop` uma variável não declarada e alguns outros. Quando estes erros são identificados pelo VBE e a janela de diálogo aparecer

você não terá de se aborrecer sobre se alguns códigos foram executados. A execução não inicia.
Se você estiver tentando executar o procedimento do Excel e aquele procedimento estiver protegido com *password* (proteção ativada) o seguinte tipo de caixa de mensagem aparece.



Clique em "OK", vá para o VBE, desproteja o projetoe rode o procedimento. A primeira linha de código é destacada em amarelo, o problem é destacado em azul e a caixa de mensagem diferente aparece. Fixe o problema, clique na seta azul na barra de ferramentas para reiniciar a execução ou clique no quadrado azul na barra de ferramentas e recomece do VBE ou do Excel..

Se você estiver tentando executar o procedimento do VBE o seguinte tipo de caixa de mensagem aparece. Se você estiver tentando executar o procedimento do Excel e aquele procedimento não estiver protegido com password ou a proteção foi desativad, o VBE abre e o seguinte tipo de caixa de mensagem aparece.



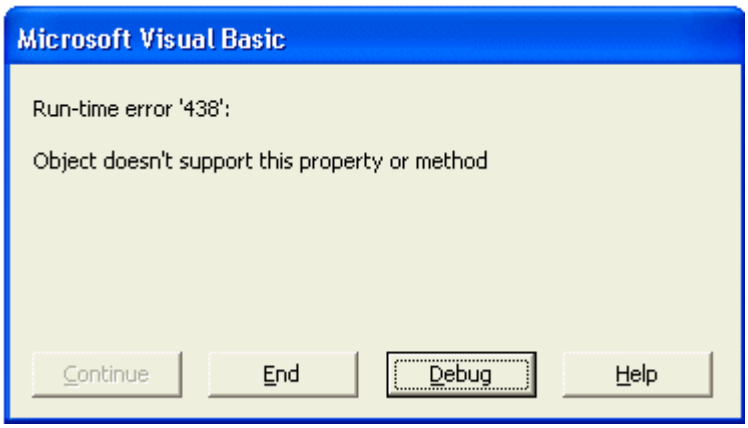
A primeira linha de código na janela de código do VBE é destacada em amarelo e o problema é destacado em azul. Fixe o problema, clicar na seta azul na barra de ferramentas para retornar a execução ou clicar no quadrado azul na barra de ferramentas e reiniciar o VBE ou o Excel.

Enquanto Roda o Procedimento

Um terceiro conjunto de erros é identificado pelo VBE quando ele tenta executar uma certa sentença incorreta no seu procedimento. Este tipo de erro inclui quando você está pedindo para o VBA abrir a pasta que ele não pode encontrar, quando você estiver pedindo ao VBA para ativar uma folha que não saiu ou quando você estiver usando o método Find e o valor que você está procurando não é encontrado.

De onde o procedimento tem sido iniciado a seguinte caixa de mensagem aparece. Se o procedimento estiver protegido por um password e a proteção está ativa o botão "Debug" não está ativo (fonte cinza) neste caso a única solução é encerrar o

procedimento no ponto onde o erro foi encontrado. Isto significa que parte da tarefa foi efetuada sempre que problemas como este forem criados.



Se o procedimento está sendo testado do VBE você pode clicar no "Debug". A sentença falha será destacada em amarelo. Fixe o problema, clique na seta azul na barra de ferramentas para reiniciar a execução ou clique no quadrado azul na barra de ferramentas e reinicie do VBE ou do Excel.

Manipulando Erros VBA

Aqui está o modo básico de manipular um erro. A primeira coisa é você criar um endereço para onde VBA saltará se houver um erro. Neste exemplo o endereço é addJump com os dois pontos **NÃO OPCIONAL** no final. Abaixo **addJump** está aquilo que é suposto acontecer se houver um erro e acima está o procedimento que deveria rodar se não existirem erros. Note que a última linha de código na seção acima **addJump** é **Exit Sub** porque se não existirem erros você não quer que o resto do código seja executado.

Exemplo 1:

```
Sub proTesteManipuladorDeErro()
    On Error GoTo addJump
    Workbooks.Open "xxxxxx"
    Exit Sub
addJump:
    MsgBox "Um erro ocorreu, chamar Bertolo em 1 514-257-0734"
End Sub
```

Exemplo 2:

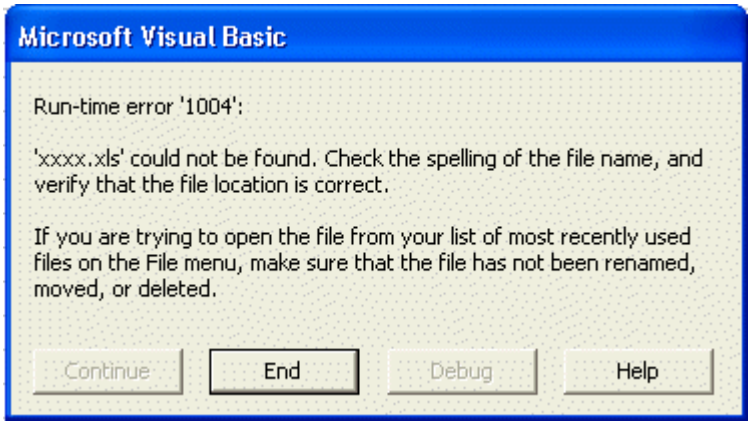
```
Sub proTesteManipuladorDeErroNenhumErro()
    On Error GoTo addJump
    MsgBox "Está tudo perfeito."
  Exit Sub
addJump:
    MsgBox " Um erro ocorreu, chamar Bertolo em 1 514-257-0734"
End Sub
```

Copiar/Colar ambos os exemplos num módulo do seu próprio e rodá-los. O primeiro exemplo gerará uma caixa de mensagem dizendo Um erro ocorreu, chamar Bertolo rm 1

514-257-0734 porque um erro tem ocorrido. A pasta xxxxxx não pode ser encontrada.
O segundo exemplo gerará uma caixa de mensagem dizendo Everything is fine.
Se você rodar o procedimento sem o manipulador de erro:

```
Sub proTesteManipuladorDeErro2()
     Workbooks.Open "xxxxxx"
End Sub
```

Você terminará com a seguinte caixa de mensagem do VBA.



Se você apenas quiser que os erros sejam ignorados você escreve On Error Resume Next no começo do procedimento. Copiar/Colar o seguinte procedimento num módulo para o seu próprio e rodá-lo. Ele gerará uma caixa de mensagem dizendo Um erro ocorreu mas nós temos ignorado-o.

```
Sub proTesteManipuladorDeErroIgnorar()
     On Error Resume Next
     Workbooks.Open "xxxxxx"
     MsgBox " Um erro ocorreu mas nós temos ignorado-o."
End Sub
```

Fase # 3: VBA do Excel Avançado

Lição 1: Código VBA do Excel para Variáveis - Lição 2: Trabalhando com Declarações - Lição 3: Trabalhando com Funções - Lição 4: Trabalhando com databases - Lição 5: Criando UserForms - Lição 6: Adicionando Controles aos userforms Lição 7: Código VBA do Excel para UserForms - Lição 8: Código VBA do Excel para Gráficos e Arquivos Seqüenciais – Lição 9: Criando e Modificando Menus – Lição 10: Usando controles ActiveX nas planilhas e gráficos – Lição 11: Trabalhando com Barras de Ferramentas – Lição 12: Os Suplementos

Lição 1: Trabalhando com Variáveis

Você as cria no VBA e você carrega nelas os valores, números, palavras e tabelas de dados inteiras. O código fica menos difícil, e valores com menos manutenção. Existem mesmo tipos de variável que permite você resolver problemas de cálculos compridos (1.000.000 células calculadas em uns poucos segundos).

Lição 2: Trabalhando com Declarações

Coisas como IF..THEN, DO...LOOP, FOR...NEXT, WITH...END WITH, EXIT FOR, EXIT DO, EXIT SUB

Lição 3: Trabalhando com Funções

Usando as funções Excel com macros, criando novas funções Excel e usando funções VBA como UCASE, LCASE e NOW().

Lição 4: Trabalhando com Databases

As caractereísticas do database em Excel são ferramentas extraordinárias. Aprenda ai a usá-las com VBA (classificação, filtragem de dados).

Lição 5: Criando Userforms

Forms são folhas especializadas que você cria para exigir entradas de um usuário. Quando a caixa de entrada de dados ou a caixa de mensagem não forem suficientes, você desenvolve userforms para exigir um *password* (com os caracterees ****), para exigir um ou muitos parâmetros (datas, números, taxas..., nomes ramificados...).

Lição 6: Adicionando Controles a Userforms

No userform você adiciona controles (botões de comando, caixa de listagens, caixas de texto, caixas de combinação (listas *drop-down*),etc.. Aprenda como criá-las e a configurar suas propriedades de modo que elas façam **EXATAMENTE** o que você quiser que elas façam. Ela é uma forma de ser amigável ao usuário.

Lição 7: Código VBA do Excel para UserForms

O código funciona com userforms e controles (botões de comando, caixas de texto, caixa de listagens, etc...). Adicionar valores aos seus controles quando o form estiver ativado, mostrar e ocultá-lo....

Lição 8: Código VBA do Excel para Gráficos e Arquivos Seqüenciais

O código funciona com controles (botões de comando, caixas de texto, caixa de listagemes, etc...). , validar os valores que os usuários entraram e transferí-los quando o usuário clicar no botão de comando.

Lição 9: Criando e Modificando Menus

O Microsoft Excel tem mais de 30 menus que ajudam você no seu uso do Microsoft Excel. É possível aumentar ainda mais a barra de menus adicionando novos menus e itens de menu a eles. E, você pode adicionar itens de menus que aparecem somente quando um dado arquivo é aberto. De fato, você pode remover os menus Microsoft Excel e trocá-los com menus personalizados se você estiver criando uma aplicação personalizada no Microsoft Excel.

Lição 10: Usando controles ActiveX nas planilhas e nos gráficos

As lições 5, 6 e 7 mostraram-lhe como criar *userforms* e colocar neles objetos tais como as barras de rolamento, as caixas de listagem, e os botões de comando. O Microsoft Excel também lhe permite colocar tais objetos **diretamente** numa planilha ou gráfico. Se o objeto for um botão de comando, uma macro pode ser especificada a executar quando o botão for clicado. Se o objeto for uma caixa de listagem, um botão de rotação ou uma barra de rolamento, então estes podem estar vinculados às fórmulas ou células nas suas planilhas.

Lição 11: Trabalhando com Barras de Ferramentas

É possível com o Microsoft Excel modificar as barras de ferramentas de modo que as ferramentas que você precisar são aquelas que estão nas suas barras de ferramentas. Não somente se pode atribuir qualquer um dos muitos botões ferramenta às suas barras de ferramentas, você pode também atribuir macros aos seus botões que podem ser adicionados a uma barra de ferramenta existente ou a uma nova barra de ferramenta. E, você pode ter macros para criar barras de ferramentas e botões quando uma pasta for aberta e removê-los quando ela for fechada.

Lição 12: Os Suplementos

Se você quiser distribuir macros e aplicações que você escreveu, você provavelmente vai querer considerar em distribuí-las como um arquivo suplemento em vez de um arquivo pasta. A principal vantagem de um arquivo suplemento é que ele é mais fácil para usar funções. Uma outra vantagem, um suplemento pode ser armazenado em qualquer diretório e ajustado para ser carregado quando o Microsoft Excel iniciar. Isto elimina a necessidade de colocar o arquivo no diretório de início do Microsoft Excel (tipicamente o *Excel\Xlinicio*). E, os suplementos não solicitam se serão ou não gravados quando se fecha o Microsoft Excel.

Lição 1: Código VBA do Excel para Variáveis

Uma variável é um objeto que você cria e na qual você pode armazenar texto (STRING), datas (DATE), números (INTEGER, LONG, SIMPLE, DOUBLE) ou qualquer outra coisa também (VARIANT).

Eu crio as variáveis que uso como, e com, contadores. Copiar/Colar o seguinte procedimento num módulo e tentá-lo passo a passo:

```
Sub proTeste()
Dim varContador as Double
Dim varNumero as Double
     Range("A1").Select
     varNumero = 1
     For varContador = 1 to 25
         Selection.Value= varNumero * 3
         varNumero =Selection.Value
         Selection.Offset(1,0).Select
    Next
End Sub
```

Algumas vezes eu contarei o número de linhas, armazeno o resultado numa variável e daí eu faço alguma coisa tantas vezes quantas linhas existirem:

```
Sub proTest()
Dim varContador as Double
Dim varNrLinhas as Double
     'Aqui eu conto o número de linhas
     For varContador = 1 to varNrLinhas
         'Fazer isto e aquilo
    Next
End Sub
```

Quando eu crio uma nova pasta de um procedimento numa outra pasta e quero salvar a nova pasta no mesmo diretório que a pasta que estou trabalhando nela, eu uso uma variável que eu nomeio de "varPath". A razão de fazer isto é que se a pasta original for aberta pelo Excel o diretório *default* é o diretório da pasta aberta, mas se a pasta original tiver sido aberta pelo Explorer, o diretório por default é "Meus Documentos". ASSIM para ficar certo de que a nova pasta está salva no diretório certo, eu processo o seguinte cod:

```
Sub proTeste()
Dim varPath as String
    varPath = ThisWorkbook.Path
    'Eu então crio uma nova pasta e quando é a hora de salvá-la
    Activeworkbook.SaveAs varPath & "\novaPasta.xls"
End Sub
```

Eu uso variáveis em 99% dos meus procedimentos e, você fará o mesmo para reduzir o número de valores codificados dificilmente. A Manutenção torna-se assim muito mais simples.

Option Explicit

Quando você abre um módulo no VBE a sentença "Option Explicit" deverá ser a primeira linha de código no topo. Isto significa que você DEVE declarar todas as variáveis que você usar e existem muitas vantagens desta obrigação.

Para ativar ou desativar esta opção vá a Ferramentas/Options/Editor no VBE e marque o item "Require Variable Declaration". Agora no topo de todas as janelas de código você verá "Option Explicit". Por que deverá esta opção ser ativada? Digamos que você esteja usando uma variável chamada "varMinhaVar" e que você escreva varNinhaVar = 32. Você errou na ortografia do nome da variável e você irá continuar pensando que o valor de varMinhaVar seja 32 e **ele não é**. Se a Option Explict estiver ativada, o VBE lhe dirá que você está usando uma variável não existente " varNinhaVar ". Você então evitará um monte de erros possíveis.

Declaração de Variável

Declare todas as suas variáveis no começo dos seus procedimentos e sempre use o prefixo "var" e um ou muitas letras maiúsculas dentro do nome:

```
Dim varEstaVariavel as STRING
```

se você errar na ortografia do nome da variável dentro do procedimento, o Excel não capitalizará as letras dizendo-lhe que há alguma coisa errada. O prefixo "var" tornará o seu código mais fácil para ler.

Tipos de Dados

Por querer manter as coisas simples, eu essencialmente uso quatro tipos de variáveis:

STRING	Text, até 65.000 caracterees.	Eu uso-o para textos mas também nomes de arquivo, path, nomes de planilha, nomes de pasta, endereços de células.
DOUBLE	Número com ou sem decimais	Eu poderia também usar "Bytes", Integer", "Long" e "Single" mas eles todos tem limites.
DATA	Se você usa o tipo STRING para datas, você não será capaz de realizar cálculos com elas assim use o tipo "Date".	
VARIANT	Eu uso o tipo VARIANT cada vez que eu tenho um grande número de cálculos a realizar (mai do que 2.000 fórmulas). Eu levo o range para um array VARIANT, executo os cálculos e trago o range de volta para a planilha. Eu criei este procedimento muito simples que executa 1.000.000 de cálculos em menos do que 5 segundos.	

Variáveis Public e Private

Quando você declara uma variável com o código Dim varMyVar as String ela pode somente ser usados no procedimento onde ela foi declarada. Quando o procedimento terminar, a variável desaparece.

Se você quer usar uma variável em muitos procedimentos você declará-la no topo de um módulo desta maneira:

```
Public varMinhaVar as String
```

Você deve lembrar que esta variável desaparece somente quando a pasta estiver fechada e que até então ele leva o último valor que você armazenou nela. Para limpar o valor você deve usar o código:

```
varMinhaVar = "" ou varMinhaVar = Empty
```

Eu me sinto desconfortável tornando uma variável public assim eu uso outro abordagem de levar o valor de uma variável de um procedimento ao outro.

Levando o valor de uma variável de um procedimento ao outro

O modo fácil de levar o valor de uma variável de um procedimento a outro é armazenando este valor numa célula qualquer da pasta:

no primeiro procedimento:

```
Range("A3456").Value=Variable1
```

no segundo procedimento:

```
Variable1=Range("A3456").Value
```

Você também pode levar a própria variável de um procedimento para um outro procedimento que você chama do primeiro. Por exemplo:

```
Sub proTeste1()
Dim varMinhaVar As Double
varMinhaVar = 3
    Call proTeste2(varMinhaVar)
End Sub

Sub proTeste2(varMinhaVar)
    varMinhaVar = varMinhaVar * 2
    Range("A1").Value = varMinhaVar
End Sub
```

A variável é declarada somente no primeiro procedimento. Um valor é armazenado nela (3). Um segundo procedimento é então chamado com a variável coimo argumento. Após a execução dos dois procedimentos, o valor do range A1 será 6.

Variáveis Array

Uma VARIÁVEL ARRAY é uma variável multidimensional que você pode ser do tamanho da sua preferência: minhaVariavel (3) é uma variável consistindo de 4 valores diferentes, minhaVariavel (5,10) é uma variável consistindo de 66 valores, 6 linhas e 11 colunas e minhaVariavel (5,10,10) pode tomar 726 valores, etc....

Quando você declara uma variável *array*, o primeiro elemento começa com o número "0". Uma Variável varMinhaVariable(3) inclui 4 elementos de "0" a "3". Se como eu, você fica desconfortável com uma variável varMinhaVariable(0), você pode impor que o primeiro elemento comece com o número "1". Na declaração geral (onde você encontra a Option Explicit), escreva:

```
Option Base 1
```

Nesta situação, minhaVariable(3) contém somente três elementos.

Você pode tomar o valor de uma variável de um procedimento para o outro armazenando este valor numa célula qualquer na pasta. ex:

no primeiro procedimento:

```
Range("A3456").Value=Variable1
```

no segundo procedimento:

```
Variable2=Range("A3456").Value
```

Lição 2: Trabalhando com Declarações

Código VBA do Excel para Declarações

As declarações que eu uso mais freqüentemente nas minhas macros VBA Excel são: If..Then..End If, Do...Loop, For...Next e Select Case

If..Then...End If

Quando há somente uma condição e uma ação, você usará a simples declaração:

```
If Selection.Value > 10 Then
  Selection.Offset(1,0).Value = 100
End If
```

Num Português simples: se o valor da célula selecionada for maior que 10 então o valor da célula abaixo é 100 se não fizer nada[4].

```
If LCase(Selection.Value).Value= "sim" then...
```

Com esta abordagem, seu teste será válido sempre que o case que o seu cliente usa (Sim, SIM ou qualquer outra combinação de cases).

If..Then...End If (múltiplas filas)

Quando existirem somente duas condições que você quiser verificar seqüencialmente, você usará a declaração:

```
If Selection.Value > 10 Then
  If Selection.Value = 12 Then
       Selection.Offset(1,0).Value = 100
  End If
End If
```

Num Português simples: primeiro verifique se o valor da célula selecionada é maior que 10. Se ele não for não faça nada. Se ela é verificada e se o valor da célula selecionada for igual a 12. Se for, defina o valor célula abaixo como 100 e daí não faça nada.

If..Then...And...End If

Quando existirem duas condições inclusivas, você usará a declaração:

```
If Selection.Value >= 10 And Selection.Offset(0,1).Value < 20
Then
  Selection.Offset(1,0).Value = 100
End If
```

Num Português simples: se o valor da célula selecionada é maior ou igual a 10 e meno do que 20 o valor da célula abaixo é 100 caso contrário não faça nada.

If..Then...Or...End If

---

[4] Nota: Testes nas strings são case sensitive de modo que quando você testar uma *string* de caracteres e você não souber se o usuário usará letras maiúsculas ou minúsculas, use a função LCase dentro do seu teste e escreva as *strings* no seu código em letras minúsculas:

Quando existirem duas condições exclusivas e uma ação, você usará a declaração:

```
If Selection.Value = 10 Or Selection.Offset(0,1).Value = 20 Then
  Selection.Offset(1,0).Value = 100
End If
```

Num Português simples: se o valor da célula selecionada é igual a 10 ou igual a 20 então o valor da célula abaixo é 100 caso contrário não faça nada.

If..Then...Else...End If

Quando há somente uma condição mas duas ações, você usará a declaração:

```
If Selection.Value > 10 Then
Selection.Offset(1,0).Value = 100
Else
  Selection.Offset(1,0).Value = 50
End If
```

Num Português simples: se o valor da célula selecionada é maior que 10 então o valor da célula abaixo é 100, caso contrário o valor da célula abaixo é 50.

If..Then..ElseIf...End If

Quando existirem mais do que uma condição vinculada cada uma a diferente ação, você usará a declaração:

```
If Selection.Value = 1 Then
  Selection.Offset(1, 0).Value = 10
ElseIf Selection.Value = 2 Then
  Selection.Offset(1, 0).Value = 20
ElseIf Selection.Value = 3 Then
  Selection.Offset(1, 0).Value = 30
ElseIf Selection.Value = 4 Then
  Selection.Offset(1, 0).Value = 40
ElseIf Selection.Value = 5 Then
  Selection.Offset(1, 0).Value = 50
End If
```

Num Português simples: Se o valor da célula selecionada é 1 então o valor da célula abaixo é 10 mas se o valor da célula selecionada é 2 então o valor da célula abaixo é 20 mas se o valor da célula selecionada é 3 então o valor da célula abaixo é 30 mas se o valor da célula selecionada é 4 então o valor da célula abaixo é 40 mas se o valor da célula selecionada é 5 então o valor da célula abaixo é 50 mas então se o valor da célula selecionada não é 1, 2, 3, 4 ou 5, não faça nada.

Select Case

Digamos que uma variável ou uma célula pode tomar 25 valores diferentes e dependendo daquele valor 25 coisas diferentes deverão acontecer. Você pode ou construir uma declaração IF muito longa ou usar Select Case.

Do..Loop

A declaração Do...Loop faz maravilhosamente muito da mesma coisa que a declaração For..Next mas você não precisa declarar um contador porque o Loop pára quando ele encontrar uma certa condição. Tente o seguinte procedimento primeiro entrando com 1 nas células A1 até A7 ou A27 ou para mais longe quanto você queira ir.

```
Sub proTeste()
Dim varContador as Integer
   Range("A1").Select
   Do Until Selection.Value = ""
         Selection.Value = Selection.Value + 1
         Selection.Offset(1, 0).Select
    Loop
End Sub
```

Num Português simples: começando na célula A1 adicionar 1 ao valor da célula selecionada e mover uma célula para baixo. Faça isto até que o valor da célula selecionada seja nada.
Variação da declaração:

```
Do Until Selection.Value = ""
```

Faça até a célula selecionada estar vazia

```
Do While Selection.Value <> ""
```

Faça sempre que célula selecionada não estiver vazia

```
Loop Until Selection.Value = ""
```

Loop até que a célula selecionada seja vazia

```
Loop While Selection.Value <>""
```

Loop enquanto a célula selecionada não estiver vazia

Exit...

Você pode sair de um FOR..NEXT, DO...LOOP e mesmo de um procedimento a qualquer momento com a declaração EXIT

```
 If Selection.Value > 10 Then Exit For
If Selection.Value > 10 Then Exit Do
If Selection.Value > 10 Then Exit Sub
```

With...End With

Antigamente quando a memória do computador era rara e cara e os computadores não eram muito poderosos, os programadores oferecia-nos um monte de declarações With..End With porque ela era muito menos exigente de memória e das capacidades do computador. Quando você desenvolver no VBA do Excel (programas muito pequenos) a memória não é realmente uma objeção e nossos computadores pessoais são tão poderosos quanto os grandes computadores de ontem. O gravador de macro usa um monte de declarações With..End With mas eu pessoalmente não. De qualquer modo aqui está como ela funciona.

```
Range("A3").Select
With Selection.Font
        .Name = "Arial"
        .Size = 24
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
End With
```

É o mesmo que escrever

```
Range("A3").Select
Selection.Font.Name = "Arial"
Selection.Font.Size = 24
Selection.Font.Strikethrough = False
Selection.Font.Superscript = False
Selection.Font.Subscript = False
Selection.Font.OutlineFont = False
Selection.Font.Shadow = False
Selection.Font.Underline = xlUnderlineStyleNone
Selection.Font.ColorIndex = xlAutomatic
```

Ambos funcionam e você é que escolhe.

For..Next

O loop FOR...NEXT é aquele um que eu uso mais. Ele permite você repetir uma ação um certo número de vezes.

```
Sub proTeste()
    Range("A1") = 10
Range("A2").Select
    For varContador = 1 To 10
        Selection.Value = Selection.Offset(-1, 0).Value * 2
        Selection.Offset(1, 0).Select
    Next
End Sub
```

Num Português simples: Defina o valor da célula A1 como 10 e então selecione a célula A2. Enquanto o contador estiver indo de 1 a 10 (10 vezes em outras palavras) o valor da célula selecionada será duas vezes o valor da célula acima...mover uma célula para baixo. Resultando deste procedimento, Célula A1=10, A2=20, A3=40.....A11=10.240.

Se você escrevesse:

```
For varContador = 1 To 10 Step 2
```

a façanha seria realizada somente 5 vezes. Resultando deste procedimento, célula A1=10, A2=20, A3=40.....A6=320.

Você pode também iniciar debaixo para cima

```
For varContador = 10 To 1 Step -1
```

Quando você usa a declaração For..Next num conjunto de dados é interessante contar o número de linhas e ter o seu contador se movendo de 1 ao número de linhas.

```
For varContador = 1 to varNrLinhas
```

COMANDOS DE CONTROLE

VISÃO GERAL

Para adicionar poder às suas macros e lhe dar controle completo sobre quais ações suas macros fazem, você precisa ser capaz de usar as seguintes declarações do Visual Basic:

- Do..Loop
- For..Next
- For Each..Next
- While..Wend
- GoTo
- With..End With
- IF..Then
- Selecionar Case

Por exemplo, usando declarações Do e For lhe permite repetir um conjunto de declarações novamente, sem que duplicá-las muitas vezes na sua macro. A declaração If é uma dos comandos mais importantes e mais poderosos do Visual Basic. Ele permite o Visual Basic tomar decisões como quais declarações executar baseado nos testes que você especificou. O comando With permite-lhe escrever declarações do Visual Basic de uma maneira curta que minimiza a digitação.

O que segue são os tópicos deste capítulo:

Tests Expressions
Boolean Tests And Numbers
Arithmetical Test Operators
Logical Test Operators
The Object Test Operator
Testing Using The Not Operator
Using Parentheses With Tests
Testing Using The Not Operator
String Tests
If..Then Statements
Do..Loop Statements
For..Next Statements
For Each..Next Statements
Goto Statements
While..Wend Statements
Stopping Macros That Won't Stop
With..End With Statements
The Selecionar Case Statement
Nesting Commands
Condition Statements For Objects

EXPRESSÕES TESTES

Muitos dos comandos de controles do Visual Basic exigem que você construa um teste ou especifique uma condição que retorne um resultado True ou False. Por exemplo, in the seguinte macro the expressão ActiveCell.Value > 0 is a teste:

```
Sub TestExample
If ActiveCell.Value > 0 Then
MsgBox ActiveCell.Value
End If
End Sub
```

Se a célula ativa contém o valor 5, então a caixa de mensagem aparecerá. Se o valor é zero ou negativo a declaração MsgBox seguinte a declaração IF acima será pulada.

Existem várias maneiras básicas que você pode escrever uma expressão teste:

- Expressão 1 operador teste Expressão 2
- Especificar uma propriedade Boolean
- Especificar uma variável Boolean
- Usar um método ou função que retorna True ou False

O que segue ilustra os testes Expressão 1 *operador teste* Expressão 2:

- ActiveCell.Value > 0
- X >= Y
- ActiveCell.Offset(1, 0) = ""

O primeiro teste, ActiveCell.Value > 0, retorna True se o valor na célula ativa é maior do que zero. O segundo teste, X >= Y, retorna True se a variável X é maior que, ou igual, a variável Y. O último teste, ActiveCell.Offset(1, 0) = "", retorna true se a célula uma linha abaixo da célula ativa estiver vazia.

Quando o Visual Basic encontrar um teste, ele primeiro avalia as expressões e daí determina se o teste é ou não True ou False. O que segue mostra um exemplo do uso de um teste para determinar se o botão No é selecionado numa mensagem caixa:

```
Dim response As Integer
response = MsgBox(prompt:="Selecionar Um Botão", Buttons:=vbYesNo)
If response = vbNo Then
MsgBox "Você selecionou o Botão Não"
Else
MsgBox "Você selecionou o Botão Sim"
End If
Exit Sub
```

O teste no exemplo acima é "response = vbNo". Se o usuário selecionar o botão No quando a caixa de mensagem for mostrada, a variável response é definida igual a vbNo.

O Visual Basic vai então a próxima declaração e avalia a expressão teste. Desde que a response é igual a vbNo, o teste avalia para True e a declaração seguinte ao Then é executada. Se você clicar o botão Yes, então o VB pula para a declaração Else e roda o código seguinte a ele.

TESTES BOOLEAN

A propriedade Boolean é uma propriedade que tem um valor True ou False. O que segue são exemplos de propriedades Boolean que podem ser usados como um teste:

Propriedade	If True
ActiveCell.Font.Bold	Célula está negritada
ThisWorkbook.Saved	Arquivo não tiver sido modificado

O que segue ilustra o uso de um valor Boolean como um teste. O teste é a expressão ActiveCell.Font.Bold.

```
If ActiveCell.Font.Bold Then
ActiveCell.Font.Underline = xlSingle
Else
ActiveCell.Font.Underline = xlNone
End If
```

Se a célula é negritada, ActiveCell.Font.Bold, retornará True. Isto torna o teste true e a próxima declaração é executada. Se ela não for negritada, ActiveCell.Font.Bold retorna False e o Visual Basic salta para a declaração Else e executa the código seguinte a ela em vez disso.

Uma variável Boolean é como uma propriedade Boolean no que ela também tem um valor True ou False. Se você declarou uma variável como uma variável Boolean, a ela é dado um valor inicial de False. Geralmente um teste de alguma sorte é usado para definir a variável Boolean para True ou False. Então a variável Boolean é usada mais tarde na macro. Isto está ilustrado na seguinte macro:

```
Dim fazerCorreções As Boolean
Dim célula As Range
For Each célula In Range("A1:A20")
If IsError(célula.Value) Then
fazerCorreções = True
Else
fazerCorreções = False
End If
If fazerCorreções Then
'declarações
End If
Next célula
```

As "declarações" acima podem ser quaisquer declarações que você queira. Por exemplo, elas podem ser declarações que mostrem caixas de mensagem ou mudem os

valores de outras células. Coloque ai sua própria declaração no lugar de "declarações" como um exercício.

Existem dois testes nas declarações acima. O primeiro teste é para ver se a célula é um valor erro. Se ela for a variável Boolean doCorrections é definida para True caso contrário ela é definida para False. Posteriormente na macro o valor doCorrections é testado numa declaração If para determinar se certas declarações adicionais deveriam ser executadas.

TESTES BOOLEAN E NÚMEROS

Testes usando valores Boolean, True ou False, são quase sempre simples e fáceis de se usar. Entretanto, o Visual Basic considera True ser igual a menos um e False ser igual a zero. Se você inadvertidamente comparar um número a um valor Boolean, você pode end up with your macro doing o oposto do que você quer fazer.

Variáveis Boolean podem ser comparadas somente com outras variáveis Boolean ou às variáveis declaradas como Variant. Comparar com uma variável *string* resultaria num erro de execução.

OPERADORES DE TESTES ARITMÉTICOS

O que segue são os operadores de testes aritméticos que você pode usar nos testes de expressões: Expressão1 Operador Expressão2, onde o Operador pode ser qualquer um dos seguintes:

Operador	resultado do Teste
=	True se a primeira expressão for igual à segunda
>	True se a primeira expressão for maior que a segunda
<	True se a primeira expressão for menor que a segunda
>=	True se a primeira expressão maior que ou igual à segunda
<=	True se a primeira expressão menor que ou igual à segunda
<>	True se a primeira expressão não for igual a segunda

Por favor note que o sinal de igual numa expressão teste não é o mesmo que o sinal de igual quando se define uma variável igual a um valor. Quando uma variável estiver sendo definida para um valor, ela está normalmente como uma palavra mais a esquerda numa linha. Se um teste estiver sendo feito, existe uma palavra chave do Visual Basic como If, Do, ou While na frente da primeira variável. No seguinte,

```
If X = 5 Then
'declarações
End If
```

a declaração X = 5 é um teste para ver se X é 5. Não é definir X igual a 5.

Se você escrever os testes >=, <= e <> na ordem errada (=>, =< ou ><), o Visual Basic automaticamente corrigirá a ordem quando você mover para uma outra linha.

OPERATORES DE TESTES LÓGICOS

Os operadores lógicos And, Or, Eqv, Imp, e Xor podem ser também usados para construir testes complexos. Por exemplo, the seguinte uses the And operador to join duas testes. If both testes are true (the ativa célula contains an entry e the célula below it

also contains an entry) the equivalent of pressionar End down arrow para mover-se para o final de um bloco de células é executado.

```
If ActiveCell <> "" And ActiveCell.Offset(1,0) <> "" Then _
ActiveCell.End(xlDown).Select
```

O uso de Offset(row, column) e End(direction) são cobertos no capítulo sobre trabalhando com range de células.

O que segue ilustra o uso mais complexo dos operadores lógico. Neste exemplo, as variáveis Boolean X, Y, e Z já foram definidas anteriormente numa macro.

```
If X And Y And Z Then
declarações
End If
```

No que está acima, a expressão teste

```
X And Y And Z
```

retornará True somente se todas três variáveis Boolean forem True. Se o teste fosse

```
X Or Y Or Z
```

Então ela retornará True se qualquer uma das três variáveis for True.

A sintaxe para uma expressão teste usando um operador lógico é:

expressão 1 *operador lógico* expressão 2

O que segue é o que a expressão acima retorna para cada um dos operadores lógicos
:

Operador Lógico — Resultado doTeste

And Retorna True se a expressão 1 e 2 forem ambas True. Caso contrário False é retornado.

Or Retorna True se a expressão 1 ou 2 forem True. Caso contrário False é retornado.

Xor Retorna False se as expressões 1 e 2 forem ambas True ou ambas False. Retorna True se uma expressão é True e a outra é False.

Eqv Retorna True se as expressões 1 e 2 forem ambas True ou ambas False. Retorna False se uma expressão é True e a outra é False.

Imp Retorna True se a expressão 1 é True e a Expressão 2 é False. Caso contrário False é retornado.

O OPERADOR DE TESTE DE OBJETO

O operador Is é usado para comparar variáveis objeto. Se você usar Is para comparar variáveis não-objeto, você provavelmente travará o Microsoft Excel. Um teste

Is retorna True se ambas expressões referirem ao mesmo objeto. Você também pode usar o operador Is para ver se uma variável objeto fora atribuida a um objeto. Por exemplo, o seguinte usa a expressão teste "cellToUse Is Nothing" para ver se o usuário selecionou uma célula ou um range de células quando a caixa de entrada de dados for mostrada:

```
Dim cellToUse As Range
On Error Resume Next
Set cellToUse = Application.InputBox _
(prompt:="Selecionar a célula range", _
type:=8)
On Error GoTo 0
If cellToUse Is Nothing Then
MsgBox "Nenhum range selecionado"
Exit Sub
End If
'declarações a executarem se um range foi selecionado
```

Isto testa para ver se um range de células fora atribuido a uma variável cellToUse. A palavra chave Nothing representa a ausência de um objeto. Set é usado pois um objeto está sendo atribuido a uma variável.

A razão para a declaração On Error Resume Next (chamada de uma cilada de erro) no que está acima é para impedir uma mensagem de erro de aparecer se cancelar for selecionado na caixa de entrada de dados.

Note que uma declaração On Error GoTo 0 é imediatamente usada para remover esta cilada de erro de modo que ela não se aplica às declarações futuras. Manipuladores de erros é discutido em detalhes num capítulo posterior.

USANDO PARÊNTESES COM TESTES

Quando você construir testes, você pode use combinações dos testes acima. E, você pode usar parênteses para controlar como os testes são avaliados. Sem parênteses, os testes são avaliados da esquerda para a direita. Por exemplo:

Expressão 1 And (Expressão 2 Or Expressão 3)

faria o Visual Basic avaliar a expressão no primeiro parênteses. Sem os parênteses, o Visual Basic avaliaria o primeiro teste

Expressão 1 And Expressão 2

Ele avaliará então este resultado primeiro com o operador Or e Expressão 3. Por exemplo, se Expressão 1 é False, Expressão 2 é True e Expressão 3 é True, o teste acima com os parênteses avaliarão para False. Sem os parênteses ele avaliará para True. Escrever o teste com e sem parênteses e substituir estes valores para provar esta declaração.

TESTANDO O USO DO OPERADOR NOT

O operador Not é usado para converter um resultado True para um valor False e vice versa. Por exemplo, se a variável X tem o valor 7, o teste Not X > 0 retorna False. O teste

```
ActiveCell.Value = ""
```

retorna True se a célula está vazia. O teste

```
Not ActiveCell.Value = ""
```

retorna True se a célula não está vazia. Uma célula pode parecer vazia e conter espaços. Neste caso o que está acima retornará True, pois espaços são os mesmos que letras. Você pode use parênteses with the Not operador. Por exemplo:

```
Not (X = 0 And Y = 0)
```

o Visual Basic avaliará primeiro (X = 0 And Y = 0). Se este avaliar para False, então Not False retorna True. Se ele avaliar para True, então Not True retorna False. Se não existise parênteses no que está acima,

```
Not X = 0 And Y = 0
```

o Visual Basic primeiro avaliará X = 0. Daí então ele aplicará o operador Not ao resultado antes de continuar avaliando a declaração. Se X fosse 1 e Y fosse 1, a expressão com os parênteses avaliará para True. A expressão sem os parênteses avaliará para False.

TESTES DE STRING

Testes não são limitados a testes numéricos. Você também pode fazer testes de string. Entretanto, testsusing strings are mais complex than numeric testes. Os testes numéricos são muito fáceis de se entender pois números são fáceis de se lidar. Entretanto, testes de *string* devem também levar em conta o seguinte:

- Uma capitalização da string.
- . O tamnho da string, o qual inclui qualquer espaço em branco na string ou no final da string.

A definição *default* no Visual Basic é fazer os testes string em case sensitive. Isto significa que os seguinte testes

```
"ABC" = "abc"
"Linda" = "linda"
```

retornam False, não True. Para tornar um teste string case insensitive, você pode fazer uma das duas coisas:

- Colocar a declaração opção Option Compare Text no topo do seu módulo, antes de quaisquer macros ou funções.
- Converter as expressões para o mesmo case usando o UCase, LCase, ou Application.Proper para converter o *case* para maiúsculas, minúsculas ou proper case quando você escrever o teste. Por exemplo:

UCase(string) = UCase(string)

converte as *strings* para maiúsculas e então faz o teste.

Se as strings são do mesmo case, mas de diferentes tamanhos, um teste string também retornará False. Por exemplo o teste seguinte

"ABC" = "ABC "

retornará False pois a primeira expressão é de tamanho 3 caracterees e a segunda é de tamanho quatro caracterees. You will find that many of the text strings given to your macros via caixas de entrada ou read from células include extra spaces. To remove extra spaces, included extra spaces in between palavras, use the function Application.Trim. A Application.Trim remove quaisquer leading e trailing spaces e quaisquer espaços extra numa string texto. Por exemplo:

" May 1 " becomes "May 1". O que segue mostra you could use it to remove extra spaces from user input:

```
textEntered = InputBox("Entrar com a descrição do produto")
textEntered = Application.Trim(textEntered)
If textEntered = "" Then Exit Sub
```

Você pode evitar o uso de variáveis e usar as funções Application.Trim e Ucase diretamente num teste como o teste seguinte ilustra:

```
Application.Trim(Ucase(string1)) = _
Application.Trim(Ucase(string2))
```

Se você usar comparações tais como > ou < num teste de string, o Visual Basic compara a string alfabeticamente. Se as duas strings são as mesmas exceto o comprimento, a string mais curta é considerada menor que a string mais longa. Se as strings forem idênticas exceto o case, strings minúsculas são consideradas maiores que as strings maiúsculas.

O operador Like é usado para comparar strings e ver se um string pattern é encontrada dentro de uma string. A sintaxe de um teste Like é:

string Like pattern

O caracteree "*" pode ser usado para representar múltiplos caracterees na pattern string. O caracteree "?" pode ser usado para representar caracterees únicos e "#" pode ser usado para representar números únicos. O que segue ilustra vários testes Like:

Teste	Resultado
"abc" Like "a*c"	True
"abdc" Like "a?c"	False

O segundo teste é False porque o ponto de interrogação representa um único caracteree. Which means that a?c is just three caracterees e está sendo comparado a quatro caracterees, "abcd"".

Por favor note que os testes serão case sensitive a menos que a declaração Option Compare Text esteja no topo do módulo.

Informação detalhada sobre o operador Like pode ser mostrada colocando o cursor sobre o operador Like e pressionando F1 para mostrar a informação de ajuda do Visual Basic sobre este operador.

DECLARAÇÕES IF..THEN

A declaração If..Then permite uma macro tomar decisões como qual comando deverá ser executado. Por exemplo, você pode querer to carry out uma série de declarações somente se um teste particular for true. Ou parar a macro ou fazer um conjunto diferente de declarações se o teste não for true.

O que segue ilustra a sintaxe da declaração If..Then:

```
Syntax 1 If Test Then declaração se true Else declaração se false
Syntax 2 If Test1 Then
declarações If Test1 True
ElseIf Test2 Then
declarações If Test2 é True
ElseIf testN Then
declarações If TestN é True
Else
declarações se nenhum teste é True
End If
```

As declarações ElseIf e Else são opcionais. O que segue ilustra a primeira sintaxe:

```
If Y = 0 Then X = 0 Else X = 1
```

Sintaxe 1 acima é a forma de linha única da declaração If..Then e é usada para testes simples e curtos. A declaração Else expressão é opcional. O que segue ilustra em linha única as declarações If..Then que não usam a declaração Else:

```
If ActiveCell.Value > 20 Then ActiveCell.Font.Bold = True
ou
If ActiveCell.Value > 20 Then _
ActiveCell.Font.Bold = True
```

Note que a forma de linha única não termina com uma End If, mesmo se um caracteree continuação for usado. Também, o uso de um caracteree continuação não

muda de uma forma de linha única para uma forma de múltiplas-linhas. Um End If não é exigido na forma linha única, e se entrado causará uma mensagem de erro.

O que segue ilustra a other sintaxe para uma declaração If, a forma multi-linha.

```
If teste Then
declarações a executar se o teste acima é true
End If
```

ou:

```
If teste Then
declarações a executar se o teste acima é true
Else
declarações a executar se o teste acima é false
End If
```

ou:

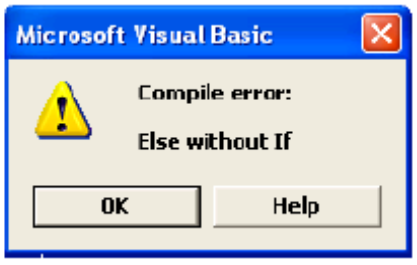
```
If teste Then
declarações a executar se o teste acima é true
ElseIf teste Then
declarações a executar se o teste acima é true
ElseIf teste Then
declarações a executar se o teste acima é true
Else
declarações a executar if nenhum teste for true
End If
```

Na forma de linha múltipla da declaração IF..Then a palavra Then imediatamente segue cada teste e é a última palavra da linha . Você achará que uma declaração multi-linha If..Then é muito mais robusta que a forma linha única e lhe dá muito mais poder para decidir que ações tomar. Você pode tantas declarações ElseIf quanto você precisar . Também, a declaração final Else é opcional.

Se uma declaração seguir o Then numa declaração multi-linha If..Then, então o Visual Basic pensa que você está usando a forma de linha única da declaração If..Then. Quando ela então encontrar um Else, ElseIf ou End If um erro ocorrerá. Por exemplo, the código seguinte resultará num erro porque o Visual Basic não sabe o que fazer com o Else pois a primeira linha é uma declaração completa.

```
If X > 5 Then Exit Sub
Else
declarações
End If
```

O que segue é a mensagem de erro que você verá se você rodar a macro acima:



A ordem em que você escreve uma declaração multi-linha If..Then é muito importante. O Visual Basic executará as declarações para o primeiro teste que é true e

então pulará para a primeira declaração após a declaração End If. Por exemplo, a macro que segue é projetada para negritar células maiores que 20, sublinhar aquelas entre 15 e 20, e mudar a fonte daquelas entre 10 e 15 para azul. Entretanto, ela não funciona corretamente. Por que você verá?

```
Sub IfTestDemo()
Dim X As Variant
X = ActiveCell.Value
'remover qualquer formatação sobre a célula ativa:
ActiveCell.ClearFormats
'Testar e mudar o formato da célula
If X > 10 Then
ActiveCell.Font.ColorIndex = 5
ElseIf X > 15 Then
ActiveCell.Font.Underline = xlSingle
ElseIf X > 20 Then
ActiveCell.Font.Bold = True
End If
End Sub
```

O que acontece é que todas as células maiores do que 10, mesmo aquelas com valores maiores do que 15 ou maior que 20, têm suas cores de fontes mudadas para azul. Entretanto, nenhuma das células são negritadas ou estão sublinhadas. O que acontece é que o primeiro teste If é true não somente para células cujos valores estão entre 10 e 15, mas também para células com valores maiores que 15. Assim os outros dois testes nunca serão atingidos. Somente as declarações para o primeiro teste true são executadas num conjunto de declarações If..End If.

Uma maneira alternativa de se escrever o que está acima to insure qua a ação correta é tomada e que a intenção da macro é facilmente entendida está ilustrada pela seguinte macro. Esta usa testes de ligação pelo operador And.

```
Sub IfTestDemo()
Dim X As Variant
X = ActiveCell.Value
'remover qualquer formatação sobre a célula ativa:
ActiveCell.ClearFormats
'Testar e mudar o formato da célula
If X > 10 And X <= 15 Then
ActiveCell.Font.ColorIndex = 5
ElseIf X > 15 And X <= 20 Then
ActiveCell.Font.Underline = xlSingle
ElseIf X >20 Then
ActiveCell.Font.Bold = True
End If
End Sub
```

DECLARAÇÕES DO..LOOP

Um Do..Loop é usado para repetir uma série de declarações over e over again. Ela começa com a palavra Do e termina com a palavra Loop. Entre as palavras Do e Loop você entra com um conjunto de declarações para realizar as ações que você quiser. O que segue ilustra a forma mais simples de um Do..Loop:

```
Do
declarações
If teste Then Exit Do
declarações
Loop
```

Tal como um Do..Loop repetirá as declarações até que o teste seja true e a instrução Exit Do seja executada para parar o laço. Quando o Exit Do for executado, o comando continua na primeira declaração seguinte à declaração Loop.

O que segue ilustra usando o estilo Do..Loop acima para inserir linhas em branco a cad cinco linhas, até que uma célula em branco seja encontrada:

```
Sub InsertRowsExample1()
Dim cellToUse As Range
Set cellToUse = ActiveCell.Offset(5, 0)
'Set is usado to assign a Range variável to its objeto,
'neste caso a célula 5 rows pertence à célula ativa
Do
If cellToUse.Value = "" Then Exit Do
cellToUse.EntireRow.Insert
Set cellToUse = cellToUse.Offset(5, 0)
Loop
End Sub
```

Você também pode usar as palavras chave While e Until com um Do..Loop para especificar um teste que é checado quando o Do..Loop for rodado. O que segue ilustra o uso desta abordagem:

```
Do While teste é true
declarações
Loop
```

ou:

```
Do Until teste é true
declarações
Loop
```

Na sintaxe acima, o teste é primeiro verificado e então as declarações estando entre as declarações Do e o Loop são rodadas se o teste for true. Se você quiser rodar as declarações entre as declarações Do e o Loop no mínimo uma vez antes de verificar o teste, coloque as declarações While e Until no final do loop. Por exemplo:

```
Do
declarações
Loop While teste é true
ou: Do
declarações
Loop Until teste é true
```

Also, você pode include an If declaração among the declarações between the Do and the Loop declarações that triggers an Exit Do declaração in a Do..Loop that uses a While ou Until declaração. This allows you to exit the Do..Loop based on a diferente teste declaração.

O que segue macro also inserts blank rows every five rows, but uses a While declaração to determine when to stop instead of an If declaração.

```
Sub InsertRowsExample2()
Dim cellToUse As Range
Set cellToUse = ActiveCell.Offset(5, 0)
Do While cellToUse.Value <> ""
cellToUse.EntireRow.Insert
Set cellToUse = cellToUse.Offset(5, 0)
Loop
End Sub
```

DECLARAÇÕES FOR..NEXT

Um dos problemas com o Do..Loops é determinar quantas vezes os comandos farão o laço. A declaração For..Next resolve este problema especificando o número de iterações que ocorrerá. O que segue ilustra uma simples declaração For..Next:

```
Sub ForDemo()
Dim I As Integer
For I = 1 To 5
MsgBox "Alô " & I
Next
End Sub
```

Quando esta macro rodar ela mostrará uma caixa de mensagem que diz alô cinco vezes. A variável "I" atua como um contador. Ela é definida inicialmente para um, e daí através de cada iteração do For..Next ela é incrementada por um. O For..Next pára de fazer o laço depois que I ficar igual a 5 (mas as declarações são rodadas quando I for igual a 5).

O que segue é a sintaxe típica da declaração For..Next

```
For counter = start número To end número
declarações
Next counter
```

Counter é uma variável que serve como um contador de laço. Geralmente, uma variável chamada "Counter", ou uma única letra variável tal como "I", "J", ou "K" são usadas para a variável counter. Colocar o nome da variável counter após a declaração

Next é opcional, mas é uma boa maneira de documentar qual laço For..Next que o Next pertence. Se você colocar o nome counter após a declaração Next, o Visual Basic confirmará que existe um correspondente For que usa aquele nome particular de counter e mostra uma mensagem de erro se não puder encontrá-lo.

O número Start é o número que você gostaria de usar como ponto de partida do laço For..Next. Normalmente, é o número "1". O número End é o valor final para o Counter.

Se você quiser, você pode especificar quanto incrementar o contador usando a palavra chave Step como parte da declaração For:

```
For I = 1 To 10 Step 2
declarações
Next
```

Se Step for deixado de fora, a variável counter é incrementada por um em cada laço. Você também pode usar um valor negativo para o Step iniciar com um grande número inicial para o contador que decresce para um pequeno número final. Por exemplo:

```
For I = 10 To 1 Step -1
declarações
Next
```

Se você quiser, você pode especificar variáveis que tenham sido atribuidas valores para começar com número, terminar com número, e step números ao contrário das entradas de números para estes valores. Ou você pode usar a propriedades que retorna números tais como a propriedade Count ou a propriedade Rows. Desta maneira, sua macro pode determinar o número de vezes a repetir o procedimento For..Next. Por exemplo:

```
iStart = 5
iEnd = 22
For I = iStart To iEnd
declarações
Next I
```

Se você quiser permitir um laço anterior For..Next, então você pode fazer isto usando a declaração Exit For. Geralmente, isto é feito com uma declaração If..Then. Por exemplo:

```
For I = 1 To 10
declarações
If teste is true Then Exit For
declarações
Next I
```

DECLARAÇÕES FOR EACH..NEXT

A declaração For Each..Next lhe permite repetir um conjunto de ações sobre os elementos individuais de uma coleção ou *array*. Por exemplo, se você selecionar um range de células, então este comando lhe permitirá repetir um conjunto de comandos em cada célula no range. Um outro exemplo seria repetir um conjunto de ações sobre as planilhas numa pasta.

O que segue é a sintaxe típica para a declaração For Each..Next:

```
For Each elemento In a coleção
declarações
Next counter
```

Por exemplo um range de células é uma coleção de células. O que segue fará o laço por cada célula na sua seleção e mostrará seu valor numa caixa de mensagem:

```
Sub ShowConstantValue()
Dim célula As Range
For Each célula In Selection
MsgBox célula.Value
Next
End Sub
```

Uma coleção é um grupo de objetos parecidos. Por exemplo, um range de células ou the planilhas numa pasta. Um elemento é um nome variável que é usado para identificar um indivíduo numa coleção. Por exemplo, uma célula ou planilha individual pode ser um elemento. A variável nome que você der ao elemento pode ser descritivo, por exemplo "célula" quando se referir a uma célula individual. Ou o nome pode ser genérico, por exemplo "X" ou "item". Por favor note que certas palavras não podem ser usadas, tais como Sheet ou Workbook pois estas são palavras chaves do Visual Basic. A tabela que segue lista alguns dos nomes que eu prefiro usar para as diferentes coleções:

Nome da Coleção	Membro da Coleção
WorkSheets	wkSheet
range of células	célula
Sheets	oSheet
Charts	oChart ou cht
WorkBooks	wkBook ou oBook
ChartObjects	oChart ou cht

Se você quiser deixar o laço anterior For Each..Next, então você pode fazer isto usando a declaração Exit For. Geralmente, isto é feito com uma declaração If..Then. Por exemplo:

```
If teste é true Then Exit For
```

Quando uma declaração Exit For é executada, o Visual Basic salta para a primeira declaração seguinte à declaração Next.

Se voltar e olhar a macro MathAction do capítulo 4, você verá que ela usou o comando For Each..Next para circular por cada célula na seleção. O que segue ilustra um comando For Each..Next que torna em negrito todas as células que tenham um valor maior do que 20 e desenha uma caixa pontilhada ao redor delas. Se o valor for menor do que ou igual a 20, qualquer negrito e bordas serão removidos.

```
Sub BoldCells()
Dim célula As Range
'fazer o laço por cada célula na seleção
For Each célula In Selection
If célula.Value > 20 Then
'fazer a fomatação se o teste for true
célula.Font.Bold = True
célula.Borders.LineStyle = xlThin
Else
'remover a formatação se o teste for false
célula.Font.Bold = False
célula.Borders.LineStyle = xlNone
End If
Next
End Sub
```

A macro que segue define o zoom em todas as planilhas numa pasta para a mesma definição de zoom, e retorna você à folha que você estava quando a macro começou. Ela também ilustra o uso da coleção de planilhas na pasta ativa.

```
Sub SetZoom()
Dim currentSheet As WorkSheet
Dim desiredZoom
Dim wkSheet As Worksheet
'armazenar a folha ativa atual
Set currentSheet = ActiveSheet
'prompt para o zoom desejado. Mostrar a
'definição do zoom atual na caixa de entrada
'obter o zoom desejado.
desiredZoom = InputBox _
(prompt:="Enter Desired Zoom", _
default:=ActiveWindow.Zoom)
'sair se nenhum valor entrado ou cancelar selecionado
If desiredZoom = "" ou desiredZoom = 0 Then End
'fazer o laço em cada planilha na pasta
For Each wkSheet In ActiveWorkbook.Worksheets
If wkSheet.Visible Then
'faça isto se a folha estiver visível
wkSheet.Selecionar
ActiveWindow.Zoom = desiredZoom
End If
Next
'retornar à folha inicial
currentSheet.Selecionar
End Sub
```

Se você quiser rodar a macro acima numa folha de gráficos, então você muda a linha For Each para para ler:

```
For Each chtSheet In ActiveWorkbook.Charts
```

A macro que segue ilustra o uso de um laço For Each..Next para selecionar cada gráfico que está embutido numa planilha e visualizar o gráfico de modo que você possa decidir se você quer imprimir ou não.

```
Sub PreviewCharts()
Dim oChart As Object
For Each oChart In ActiveSheet.ChartObjects
oChart.Activate
ActiveChart.PrintPreview
Next
End Sub
```

Se você quiser imprimir os gráficos sem visualização, então mude o PrintPreview para o PrintOut. Se você deparar com um problema onde você obtem uma mensagem de erro "Printout Method of Application Class Failed" após imprimir 4 a 5 gráficos, coloque a seguinte declaração dentro do laço:

```
Application.Wait Now + TimeValue("00:00:10")
```

Isto adiciona um atraso de dez segundos entre os gráficos para permitir a limpeza do buffer da impressora..

DECLARAÇÕES GOTO

Uma declaração GoTo permite-se desviar uma macro para qualquer linha daquela macro. Você não pode usá-la para sair de uma macro. A sintaxe da declaração GoTo é muito simples:

`GoTo Line Number`
ou `GoTo Line Label`
Por exemplo:
`GoTo 100`
ou `GoTo NextStep`

Uma linha número é um número único que você usa para identificar um linha no seu módulo. Ela não se refere ao número real de linhas quando você contar do topo. Nem tem as linhas de números que estarem em ordem. Ela é apenas um identificador numérico de uma localização. Uma linha rótulo pode ser qualquer combinação de caracterees (exceto caracterees especiais como *, $, etc..) que comecem com uma letra e terminem com dois pontos. Linhas de rótulo são usadas muito mais freqüentemente do que linhas de números, pois elas conduzem mais significado. O que segue são exemplos de linhas de números e linhas de rótulos:

```
100
200
ErrorMsg:
ExitCommands:
```

Cada linha rótulo e linha de número deve ser única dentro de um módulo. Lihas de rótulos não são *case sensitive* e seguem as mesmas regras de nomeação que as variáveis.

O uso mais comum de uma declaração GoTo é desviar para uma linha de rótulo quando um erro ocorrer. O que segue ilustra este uso de uma declaração GoTo:

```
Sub SuaMacro()
'especificar qual rótulo ir se existir um erro
On Error GoTo ErrorRoutine
'executar as declarações da macro
'sair da macro se nenhum erro ocorrer
Exit Sub
'venha prá cá se um erro ocorrer
ErrorRoutine:
MsgBox "Ocorreu um Erro. Atividade paralizada"
End Sub
```

Rotinas de erros que você escrever podem ser muito mais elaboradas do que aquela acima. Manipulação de erros é discutida em detalhes no último capítulo.

Por favor saiba que muitas declarações GoTo tornarão uma macro difícil de entender e de se removerem erros se alguma coisa estiver errada. Você geralmente ficará numa situação melhor usando declarações If..Then, Do..Loops, e comandos For..Next.

DECLARAÇÕES WHILE..WEND

A declaração While..Wend é muito semelhante ao Do While..Loop. O que segue é a sintaxe para esta declaração

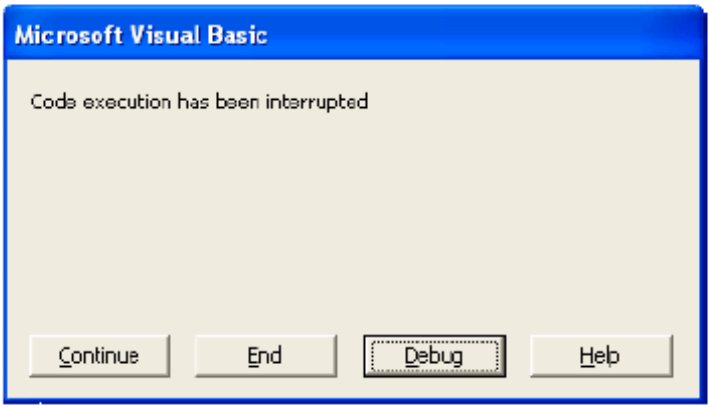
```
While teste
declarações
Wend
```

Para sair de um laço While..Wend, o teste deve tornar-se falso ou uma declaração GoTo deve ser usada. Por exemplo:

```
Sub WhileExample()
While ActiveCell.Value <> ""
ActiveCell.Offset(1, 0).Select
If ActiveCell.Row > 100 Then GoTo nextStep
Wend
nextStep:
'Macro continua daqui
End Sub
```

PARANDO MACROS QUE NÃO QUEREM PARAR

Mais cedo ou mais tarde você escreverá um Do..Loop ou a For..Next que nunca pára a execução do laço. Isto acontece se o teste parar o nunca for encontrado, ou se alguma coisa no laço está reiniciando o contador de laço. Quando isto acontecer, sua macro kfica rodando e rodando até um erro run-time ocorrer ou você interrompê-lo. O modo mais simples de interromper um laço infinito é pressionar a tecla ESC. (Você pode precisar pressionar repetidamente se a macro emitir o comando recalcular). Isto fará a seguinte caixa aparecer:



Até este ponto, você pode:

- . Parar a macro selecionando o End .
- Fazer a macro continuar selecionando o botão Continue.
- Ir para mode debug e ir dando os passos pela macro selecionando o botão Debug.
- Selecionar o botão Goto e ir à linha que era para ser executada quando você parou a macro.
- Mostrar a informação ajuda que realmente não ajudará você com o problema com a sua macro.

Por favor note que se seu laço mostrar continuamente uma caixa de mensagem, você não deverá ser capaz de pressionar a tecla ESC rápido o suficiente para interromper o laço (Eu sei, isto tem acontecido comigo!). Se isto acontecer, você deve fechar o Excel, o qual deverá exigir desligar seu computador. Assim, antes de você testar uma macro contendo um laço, você deverá salvar o seu arquivo. Se você tiver uma macro "run away" que não irá parar de executar, e pressionar o ESC e ele não funcionar, você tem uma alternativa: Se o Visual Basic editor está aberto, você pode ir para ele e clicar o botão reiniciar (aquele um com o ícone quadrado) para parar a macro. Isto paraá a macro. Ele fecha também os user forms que não quer fechar.

DECLARAÇÕES WITH..END WITH

A declaração With..End With é usada para executar uma série de declarações sobre um único objeto sem ter que digitar nome do objeto em cada declaração Tais declarações simplificam o código e torna as macros ligeiramente mais rápidas.

No capítulo 5 você viu as declarações With..End With usadas com as propriedades de PageSetup. Its use avoided having escrever a palavra PageSetup 34 vezes. O que segue é a sintaxe típica de uma declaração With..End With:

```
With objeto
declarações
End With
```

O que segue ilustra como usar uma declaração With..End para especificar a propriedade fonte de uma célula:

```
With ActiveCell.Font
.Name = "Courier New"
.FontStyle = "Bold"
.Size = 12
.Underline = xlSingle
.ColorIndex = 5
End With
```

O que segue atinge o mesmo resultado, mas exige muito mais digitação:

```
ActiveCell.Font.Name = "Courier New"
ActiveCell.Font.FontStyle = "Bold"
ActiveCell.Font.Size = 12
ActiveCell.Font.Underline = xlSingle
ActiveCell.Font.ColorIndex = 5
```

Declarações With..End With podem ser muito mais complexas do que aquelas acima, e podem ter declarações Do..Loops, If e outras declarações entre as declarações With e a End With. Por exemplo:

```
With ActiveCell
If .Value > 1000 Then
.Borders.Weight = xlThin
ElseIf .Value > 100 Then
.Borders.Weight = xlMedium
ElseIf .Value > 1 Then
.Borders.Weight = xlThick
End If
End With
```

Por favor note que você pode embutir um With..End With dentro do outro. Entretanto, você deve ser cuidadoso!

A DECLARAÇÃO SELECT CASE

A declaração Selecionar Case é freqüentemente usada em vez de uma declaração If..Else. A Selecionar Case é um teste If no formato de tabela, e é perfeitamente ajustado para selecionar de um conjunto de opções dependendo do valor de uma única variável. O que segue é a sintaxe típica da declaração Selecionar Case:

```
Selecionar Case Variable Value
Case Value Range 1
declarações
Case Value Range 2
declarações
Case Value Range n
declarações
Case Else
declarações
End Selecionar
```

As palavras Selecionar Case e End Case são exigidas para iniciar e parar uma declaração Selecionar Case. Somente uma declaração Case Simples é exigida. A declaração Case Else é opcional. Declarações múltiplas podem ser colocadas entre cada declaração Case. Por favor note que você não pode misturar valores numa declaração case – se você misturar números e strings, coê obterá uma mensagem de erro "Tipos incompatíveis" quando você executar a sua macro.

Você também pode escrever a declaração Selecionar Case no seguinte formato, se você tiver apenas uma declaração que você queira executar para um dado caso. Note que dois pontos são usados para separar o teste case da declaração da macro que será executada.

```
Selecionar Case Variable Value
Case Value Range 1: Statement
Case Value Range 2: Statement
Case Value Range n: Statement
Case Else: Statement
End Selecionar
```

Usando dois pontos permite você colocar duas ou mais declarações numa linha. O Visual Basic removerá quaisquer espaços na frente dos dois pontos.

A declaração que está acima pode ser muito simples, tal como definir um valor para uma variável. Ou ela pode ser a declaração que chama uma outra macro. Esta segunda abordagem permite você manter a suas declarações Selecionar Case compactas, mas permite também que elas sejam muito poderosas.

A tabela que segue ilustra as declarações de *range* de valor usadas com a declaração Selecionar Case:

Um Range de Valor pode ser por exemplo:
Um único valor - Case 3

Uma série de valores separados por vírgulas - Case 4,5,6
Um range de valores expressos como valor1 Até valor2 - Case 7 To 11
Um range teste expresso como teste Is de operador de valor - Case Is >12
Qualquer combinação destas expressões
Os valores Case podem também estar em ordem alfabética:

```
Case "Level 1"
Case "Level 2B", "Level 3"
```

Testes com valores alfabéticos serão *case sensitive* a menos que você tenha posto uma declaração Option Compare Text no topo do módulo ou você converter todas as strings e valores *case* para o mesmo *case*. Somente o primeiro teste que é a match is that one that será usado pelo Visual Basic, even if a match é possível num teste subseqüente numa declaração Case. A palavra chave Is é exigida para testes range, os quais são testes que usam operadores aritméticos tais como >, <, >=, < =, ou =.

O que segue ilustra using a declaração Selecionar para determinar a multa que se deve pagar por ultrapassar 100 km/h:

```
Sub DetermineFine()
Dim carSpeed As Integer
Dim multa As Integer
carSpeed = Val(InputBox("Entrar com a velocidade do carro"))
If carSpeed = 0 Then Exit Sub
'determinar a multa usando a declaração case
Selecionar Case carSpeed - 100
Case Is < 1: multa = 0
Case 1 To 5: multa = 10
Case 6 To 15: multa = 50
Case 16 To 25: multa = 100
Case Is >= 26: multa = 200
End Selecionar
MsgBox "A multa é $" & multa
End Sub
```

COMANDOS ANINHADOS

Neste capítulo foram ilustrados vários comandos de controles principais do Visual Basic. Entretanto, estas ilustrações tiveram somente um único comando de controle por vez. Você também pode usar múltiplos comandos de controle, aninhados um dentro do outro. Por exemplo:

```
For I = 1 To 5
declarações
For J = 1 To 10
declarações
Next J
declarações
Next I
```

ilustra o aninhamento de um For..Next dentro de outro. O que segue ilustra o aninhamento de declarações If:

```
If Test2 then
If Test2
If Test3
.
.
End If
End If
End If
```

Você pode aninhar tantos comandos dentro de outro quanto você queira. E, você pode aninhar diferentes comandos dentro de cada outro. Por exemplo, você pode aninhar um Do..Loop dentro de um laço For..Next. Quando aninhar comandos, cuidado que o contador para cada loop seja único.

DECLARAÇÕES CONDITION PARA OBJETOS

Volta e meia você precisará escrever uma expressão teste envolvendo um objeto. Um caso típico seria ver se o usuário tem selecionado uma célula ou range de células quando prompted a fazer isto por um controle InputBox ou uma refEdit num userform. Um outro caso típico seria ver se a declaração Find tem retornado uma célula contendo a string procurada.

Comparações de objetos não podem usar operadores de comparação normal tais como =, >, <, etc.. A palavra chave Is deve ser usada em vez disto. E para ver se uma variável objeto tem sido atribuida a um objeto, a palavra chave Nothing deve ser usada.

O que segue ilustra o uso do método Find e um objeto comparação. Você deveria escrever a seguinte macro e testá-la numa planilha contendo várias entradas:

```
Sub Find_A_Match()
Dim textToBeFound As String, msg As String
Dim célula As Range
'mostrar uma aplicação caixa de entrada para obter texto a ser
'encontrado
msg = "Entrar com o texto a ser encontrado"
textToBeFound = Application.InputBox(msg)
'Sair sem uma mensagem se cancelar for selecionado
If textToBeFound = False Then Exit Sub
'determinar se qualquer texto foi entrado
If textToBeFound = "" Then
MsgBox "Nenhum texto foi entrado. Atividade parada."
Exit Sub
End If
'usar o método Find para fazer uma procura. Resultados são
' armazenados na variável chamada Cell
'Note que você não precisa especificar um argumento "After"
Set célula = Cells.Find(What:=textToBeFound, _
LookIn:=xlFormulas, _
LookAt:=xlPart, _
SearchOrder:=xlByRows, _
```

```
SearchDirection:=xlNext, _
MatchCase:=False)
'teste para ver se qualquer texto compatível foi encontrado
'usando um teste IS
If célula Is Nothing Then
MsgBox textToBeFound & " não encontrado."
Exit Sub
Else 'Se o texto for encontrado, vá à célula onde está o texto
célula.Select
MsgBox textToBeFound & " encontrado em " & _
cell.Address
End If
End Sub
```

Pontos chaves sobre a macro acima e o método Find:

- A saída do método Find é um objeto célula. Assim ela pode ser atribuida a uma variável usando o comando Set.
- Para testar se nada foi encontrado, o que significa que a variável objeto célula não for a atribuida, uma comparação usando a palavra chave comparação Is e a palavra Nothing que representa um não atribuido objeto é usado.
- Desde que a saída do método Find é um objeto célula, as propriedades da célula pode ser mostrada como ilustrado na última caixa de mensagem.
- A InputBox Microsoft Excel é usada em vez da InputBox Visual Basic porque a InputBox Microsoft Excel retorna o valor False se o usuário selecionar o botão cancelar. Se o usuário selecionar o botão cancelar da InputBox do Visual Basic, ela retorna uma string de tamanho zero (""). Desde que isto é o mesmo que não entrar com qualquer texto e selecionar OK, a InputBox Microsoft Excel permite a macro diferenciar e aconselhar o usuário do erro. A InputBox Microsoft Excel é chamada pela declaração Application.InputBox. Ambas são cobertas em detalhes no capítulo sobre comandos úteis.
- O método Find tem argumentos adicionais que podem ser mexidos e acima apenas o argumento What. Você poderá normalmente definir todos os argumentos. Para mais informação sobre o método Find, coloque o cursor nele e pressione a tecla F1. Você encontrará também uma cobertura adicional no capítulo sobre comandos úteis. O argumento After célula é opcional

SUMÁRIO

Você encontrará que quanto mais você usar os comandos de controle acima, menores e menores as suas macros vão ficando. Its far mais eficiente e menos trabalho para escrever macros que usam declarações Do..Loops, For..Next, e declarações If than not using such comandos de controle. Adicionando tais comandos às macros que você gravar aumentarão tremendamente para você as suas utilidades. Sempre que você escrever uma macro, você deverá procurar maneiras de usar os comandos acima. Exatamente como é útil usar uma macro para eliminar trabalhos repetitivos, usar um comando de controle é também útil pois ele elimina declarações desnecessárias de macros.

Lição 3: Trabalhando com Funções

Código VBA do Excel para Funções

Três tópicos nesta lição:
- usar funções Excel dentro de macros,
- criar novas funções Excel com VBA do Excel e,
- usar funções VBA dentro de macros.

Funções Existentes no Excel

Existem centenas de funções disponíveis no VBA. A maioria das funções que você encontra no Excel são disponíveis nas macros desta forma:

```
Range("C1").Value=
Application.WorksheetFunction.Sum(Range("A1:A32"))
```

esta sentença soma os valores da célula A1 a A32 e armazena o total na célula C1.
Usar funções do Excel pelo VBA reduz substancialmente o tempo de cálculo e cria folha de planilha sem fórmulas que você pode mais facilmente enviar aos outros.

Novas Funções Excel

Você pode criar novas funções no Excel. Por exemplo, a função criada pelo código abaixo simplesmente multiplicará o valor de uma célula por 2.

```
Function fctDouble(varInput)
  fctDouble = varInput * 2
End Function
```

Uma vez o código desta função esteja num módulo da sua pasta você acessa esta nova função da mesma maneira que você acessa as outras funções do Excel clicando sobre o ícone da função na barra de ferramenta  ou da barra de menu "Inserir/Funções". Na caixa de diálogo selecione a categoria "Definida pelo usuário" e selecione sua nova função ("fctDouble" neste exemplo) e siga as instruções.

Funções VBA

Aqui estão algumas funções VBA que eu uso nas minhas macros Excel:
LCase, Ucase

As declarações IF, a SELECT CASE e DO WHILE são todas case sensitive. Quando você testar uma string de caracterees e você não souber se o usuário entrará com letras minúsculas, use as funções LCase ou UCase dentro de seu teste e escreva a *string* no case apropriado:

```
If LCase(Selection.Value)= "toto" then...
```
ou
```
Select Case LCase(Selection.Value)
```
ou
```
Do While LCase(Selection.Value)<>"toto"
If UCase(Selection.Value)= "TOTO" then...
```
ou

```
Select Case UCase(Selection.Value)
```
ou
```
DO WHILE UCase(Selection.Value)<>"TOTO"
```

NOW()

NOW() é uma função Excel mas também uma função VBA. Com o seguinte código a fórmula Excel NOW() é inserida na célula A1. A célula "A1" mostrará a data do dia e esta data mudará cada vez que a pasta for aberta:

```
Range("A1").Formula = "=Now()"
```

Com o seguinte código, a célula "A1" pegará a data quando o procedimento for executado e manterá este valor até você executar o procedimento novamente. Ela não mudará cada vez que você abrir a pasta.

```
Range("A1").Value = Now()
```

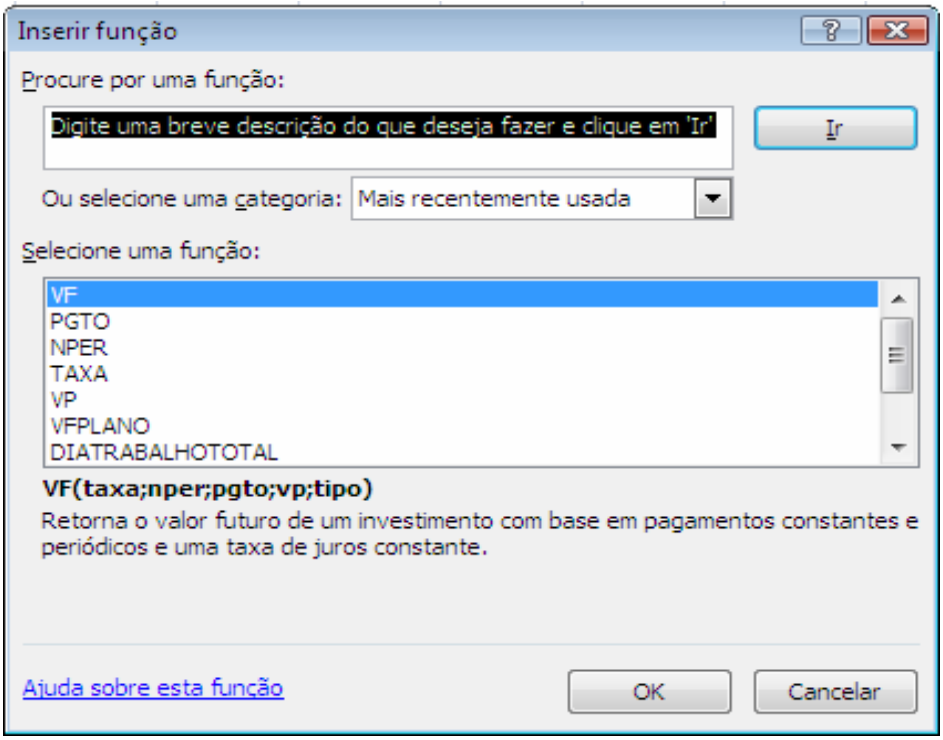
CRIANDO FUNÇÕES

As funções Definidas pelo usuário (UDF = User Defined Functions) são como as macros pois elas consistem de declarações e também são chamadas de procedimentos. Porém, uma função é diferente de uma macro, pois ela retorna com **um valor**. Uma função Definida pelo usuário pode ser usada exatamente como as funções do Microsoft Excel, quer dizer, incluída em suas planilhas eletrônicas e macros para fornecer as respostas que você precisa. Você deverá escrever e usar uma função Definida pelo usuário toda vez que você precisar de uma fórmula complexa que é usada em células múltiplas e está sujeita a mudanças. Uma função Definida pelo usuário é muito fácil de se manter e pode ser documentada com declarações de comentário.

O seguinte são os tópicos deste capítulo:

SOBRE FUNÇÕES

Uma função é um nome para um conjunto de declarações que retornam um valor. Por exemplo, SUM(), AVERAGE(), IRR() são funções **embutidas** que você poderá ter que usar. Uma função é normalmente usada numa fórmula numa célula da planilha. Entretanto, uma função pode ser também usada numa macro ou numa outra função. Se vocÊ for a uma planilha e selecionar Inserir do menu Microsoft Excel, e depois Função, o painel seguinte será mostrado:



Como você pode ver folheando o painel anterior, a Microsoft Excel tem muitas, muitas funções. As funções que você criar são adicionadas na categoria de função Definidas pelo usuário e são chamadas de funções Definidas pelo usuário.

As funções Definidas pelo usuário são escritas em módulos exatamente como as macros. Se uma função Definida pelo usuário for uma função de propósito especial, então você deverá pôr esta função em um módulo na pasta que a usará. Se você tiver várias funções de propósitos gerais, você deverá mantê-las em um módulo no seu arquivo de *Pessoal.Xls*.

FUNÇÕES VERSUS MACROS

A lista seguinte compara e contrasta as *Funções Definidas pelo Usuário* e as *macros*:

· Funções começam com a palavra *Função* e terminam com *End Function*. Macros começam com *Sub* e terminam com *End Sub.*

· Funções podem ser usadas em células de fórmulas. Macros não podem.

· Macros podem ser atribuídas a botões, menus e objetos. Funções não podem ser assim atribuídas.

· Ambas podem chamar outras funções, usar testes **If**, usar os laços **Do..Loops**, **For..Next** e outros comandos do Visual Basic.

· Funções retornam um valor quando elas terminarem de rodar suas declarações. Macros não retornam um valor.

· Funções têm uma variável nelas com o mesmo nome que a função. Seu valor é que é retornado. Macros não.

· Macros podem mudar as propriedades das células e outros objetos, deletar e adicionar objetos. Funções não podem a menos que chamadas por uma macro.

· Ambos macros e funções podem obter propriedades de informações de qualquer objeto, tais como valores de células em qualquer planilha.

· Nomes de Função e nomes de macro seguem as mesmas regras.

· Funções podem ser encerradas a qualquer momento usando a declaração *Exit Function*. Macros podem ser encerradas a qualquer momento, usando a declaração *Exit Sub*.

A exigência básica de uma função é que ela contenha uma variável que tenha o mesmo nome que a função. Seja qual for o valor colocado nesta variável pela função, é o valor que a função retorna.

NOMES DE FUNÇÕES

O nome que você dá uma função segue as mesmas convenções daquelas de nomes de variáveis e macros. Também, você não deveria fazer seu nome de função igual a uma referência de célula. Em algum lugar na sua função você precisa incluir uma linha que fixa uma variável pelo mesmo nome que a função para o valor que você quer que a função retorne. Caso contrário, a função retornará um valor *default* para o seu tipo. (Que seria zero quando as funções forem do tipo Variant a menos que declaradas o contrário).

UM EXEMPLO DE FUNÇÃO DEFINIDA PELO USUÁRIO

Para ilustrar uma função Definida pelo usuário, digamos que você precise calcular o preço de um item, e o preço é dependente da quantidade que ele é comprado, como ilustrado na tabela seguinte:

Quantidade Comprada	Preço
0-100	$10
101-300	$9
301-700	$8
701 ou mais	$7

Uma maneira de manipular isto seria escrever uma fórmula **If** numa célula. Por exemplo, se a quantidade estivesse na célula C7, a fórmula seguinte calcularia o preço para aquela entrada:

```
=If(C7<101,10,IF(C7<301,9,IF(C7<701,8,7)))
```

Esta fórmula seria fácil de manter se você só precisasse isto em uma cela. Mas, se você precisasse determinar preço em várias celas diferentes, ou se a tabela de preço for mais complexa, então você teria dores de cabeça. Especialmente, se você tiver de mudar os preços.

O seguinte é um exemplo de uma função que calcula preço:

```
Function CalcPreço(quantidade As Integer)
        If quantidade < 101 Then
           CalcPreço = 10
    ElseIf quantidade < 301 Then
            CalcPreço = 9
    ElseIf quantidade < 701 Then
            CalcPreço = 8
    Else CalcPreço = 7
    End If
End Function
```

Para usar esta função, você apenas digitará =CalcPreço(quantidade) numa célula. Quantidade pode ser um número ou uma referência de cela. Insira um módulo em branco e crie a função anterior e daí teste-a em uma folha na mesma pasta. Como você

vê, você obtém o mesmo resultado que a declaração Se, mas a entrada é mais compacta. E, você poderia usar esta função em muitas células da planilha apenas precisando manter seu código no módulo.

A função anterior pode também ser escrita usando uma declaração Select Case, que também foi discutida em um capítulo anterior. A declaração Select Case é um ajuste melhor para funções tal como esta. O seguinte é o que a função se pareceria se fosse escrita com uma declaração Select Case:

```
Function CalcPreço(quantidade As Integer)
     Select Case quantidade
         Case   0 To 100: CalcPreço = 10
         Case 100 To 300: CalcPreço = 9
         Case 300 To 700: CalcPreço = 8
         Case Is >=  701: CalcPreço = 7
     End Select
End Function
```

SINTAXE E DESIGN DE FUNÇÃO

Segue a sintaxe típica para uma função definida pelo usuário:

```
Function nomedafunção (lista de argumentos) As Type
      declarações
      nomedafunção = expressão
End Function
```

Uma função tipicamente tem uma lista de argumentos que ela espera receber. Os argumentos estão separados por virgules e entre parênteses. Os nomes dos argumentos são também os nomes das variáveis que a função usará nos seus cálculos.

Como ilustra a sintaxe acima, em algum ponto na função você assume uma variável que tenha o mesmo nome que a função para um valor. Não há restrições no valor que você pode ter para a função retornar. Por exemplo, você pode escrever a função que retorna um número, uma outra que retorne um texto string e uma outra que retorne True ou False. Uma função pode mesmo retornar uma referência a objeto.

Para usar a função numa planilha, você digitaria um sinal de igual, o nome da função, e fornecer valores para os argumentos entre os parênteses. Você não pode nomear os argumentos na equação. Os valores fornecidos devem estar na ordem que a função aguarda recebê-los. Você pode usar referências de células para fornecer os valores. Também, você pode usar a função como parte de uma fórmula complexa.

Quando você usar a função numa macro, você deverá escrever a declaração que faz uso do valor retornado. Por exemplo, você pode ajustar a variável igual ao valor retornado, usar o valor retornado num cálculo, ou usar o valor retornado como uma expressão teste.

Para usar a função num procedimento você digita o nome da função seguida por parênteses. Se você precisar passar valores para a função, então aqueles valores deverão estar dentro dos parênteses e separados por virgulas. Se os parênteses não são usados, a função não retornará um valor. A ordem em que você passa valores a uma função deve ser a ordem que a função aguarda recebê-los a menos que você especifique os nomes dos argumentos. Se você quiser, você pode especificar os nomes dos argumentos seguidos por dois pontos e um sinal de igual (:=). Por exemplo, você pode passar uma única célula, uma seleção de células, um valor numérico, ou um valor string. O que segue ilustra a chamada de funções de uma macro:

```
FunctionName(12, 88, 33)
FunctionName(primeiroNome:="João", idade:=35)
```

Por favor, lembre-se que você não pode misturar as duas aproximações de fornecer valores a uma função quando usada em suas macros. E, se você fornecer valores e não precisar fornecer todos os valores, você deve usar virgulas como placeholders a menos que você forneça os nomes do argumento.

USANDO VARIÁVEIS E ARRAYS EM FUNÇÕES

As funções não são restringidas ao uso das variáveis declaradas na lista de argumentos da função. Você pode declarar e usar tantas variáveis e *arrays* que você precisar numa função, exatamente como você faz em macros. E, funções podem usar nível de módulo e variáveis de nível global. A lista de argumentos é apenas um modo de passar valores para certas variáveis numa função. Você pode também ter *arrays* na lista de argumentos de uma função. O que segue ilustra isto:

```
Function FaçaAlgumaCoisa(X As Integer, _
     listadeCustos(1 To 5) As String, _
     infoNovas() As String, _
     doTeste As Boolean)
Dim Preços(1 To 12), J As Integer, respostadoUsuário As Boolean
'declarações
End Function
```

No exemplo acima, dois *arrays* são partes da lista de argumentos da função, listadeCustos e infoNovas. O *array* listadeCustos tem declarado a sua dimensão. Isto significa que a macro solicitante deva usar um *array* que tenha o mesmo número de elementos, neste caso 5. O array infoNovas não tem suas dimensões declaradas. Isto significa que a macro solicitante pode usar um *array* de qualquer tamanho quando ela chamar esta função. Há também um terceiro *array* nesta função, Preços, o qual é criado pela função juntamente com duas variáveis a serem usadas na função, J e respostadoUsuário.

Tome cuidado de não usar uma palavra chave restrita numa lista de argumentos da função. O Excel a detectará como um erro de sintaxe. Por exemplo, a seguinte função declaração (a qual usa a palavra chave do tipo restrita) será destacada usando a fonte de erro de sintaxe (normalmente vermelho)

```
Function ChecarTipo(tipo)
```

A menos que você perceba que o argumento seja uma palavra chave restrita, você se convencerá que Visual Basic está quebrado! Se você ligar a checagem de sintaxe e re-digitar parte da declaração, o Visual Basic mostrará a mensagem de erro "Expected: identifier" para esta mensagem e destacar a palavra "tipo". Ligando a checagem sintaxe quando você não puder compreender um problema de sintaxe da expressão é um modo de se obter conhecimento adicional ao problema de sintaxe.

Quando uma função permite uma macro solicitante passar um array de qualquer tamanho a ela, então ela pode usar as funções UBound e LBound para determinar o tamanho do *array*. A sintaxe é:

UBound(arrayname, número de dimensão)
LBound(arrayname, número de dimensão)

Por exemplo, UBound(NewInfo, 1) retorna o número de elementos da primeira dimensão de um *array* chamado NewInfo. LBound(NewInfo, 1) retorna o número de elementos iniciais do array. Se um número de dimensão não é fornecido, é assumido ser um.

ESPECIFICANDO UM TIPO PARA UMA FUNÇÃO

Você pode declarar a função, e assim o valor que ela retorna ser um tipo particular (Boolean, Integer, Long, String, etc..) se você quiser. Para fazer isto, usar a notação Quando tipo é seguido de parênteses. Declarar um tipo para uma função é opcional. Se você não declarar um tipo para uma função, seu tipo é considerado como Variant. O que segue, ilustra a declaração de uma função tipo:

```
Function PreçoComputador(quantidade) As Integer
```

ou

```
Function ChecarValor(nomedoEmpregado) As Boolean
```

Declarar um tipo para uma função não só é uma boa prática como também ela ajuda evitar erros ao se usar uma função. Também, ela ajusta o valor inicial para a função baseada no tipo declarado. Por exemplo, a função seguinte:

```
Function ChecarData(valorData) As Boolean
     If valorData > 100 Then ChecarData = True
End Function
```

retornará False, a menos que o teste seja true. A variável ChecarData não precisa ser ajustada para False pois a função é declarada como Boolean e o valor inicial da variável ChecarData é então False.

Por favor, note que se você tiver uma função retornar uma data, então você deverá declarar o tipo da função como Date.

ESPECIFICANDO TIPOS PARA ARGUMENTOS DE FUNÇÃO

É uma boa idéia declarar os tipos dos argumentos de uma função. Declarar os tipos para os argumentos numa função é uma prática muito boa, pela mesma razão que você declara os tipos de variáveis numa macro. Entretanto, não deverá usar Range como um dos tipos se você estiver usando a função numa planilha. Você obterá um #NAME? como resultado de sua função se você usá-la numa planilha. Ao investir nisso, usar Object ou Variant como o tipo para uma variável que é realmente uma variável range.

O seguinte declara as variáveis na lista de argumento da função ChkValues:

```
Function ChkValues(testResults As Boolean, _
      Y As Integer, _
      Z As String) As Boolean
```

Note que a declaração Function pode ser declarada através de linhas múltiplas usando um espaço seguido pelo caracteree sublinhado, (_). Também, o tipo da função por si só é declarado.

Se uma outra macro chamar a função e as variáveis usadas para fornecer valores à macro solicitante são de tipos diferentes daqueles dos argumentos da função, a mensagem de erro seguinte aparecerá:

Para resolver, você precisa ou tornar os tipos iguais ou prefixar o argumento da função com a palavra chave ByVal se a variável tem o valor propriedade. Por exemplo:

```
Function TestarValor(ByVal qualquerNúmero As Single)
```

TORNANDO OS ARGUMENTOS DA FUNÇÃO OPCIONAL

Se você quiser tornar um argumento opcional, inclua a palavra chave Opcional na frente do argumento na lista argumento. Por exemplo:

```
Function TipoCusto(orderNum As Integer, _
      Optional costCode As Variant)
```

Uma vez tendo definido a variável como ótima, todas variáveis subseqüentes deverão ser declaradas opcionais também. Também, todas variáveis opcionais devem ser do tipo Variant.

Para determinar se um argumento optional foi fornecido, você deverá usar a função IsMissing (argumento). Se o argumento não foi fornecido, IsMissing retorna True. Se ele foi fornecido, IsMissing retorna False. O que segue ilustra o teste para um argumento opcional.

```
If IsMissing(costCode) Then costCode = "Unknown"
```

USANDO ARGUMENTOS DO TIPO USUÁRIO

O Visual Basic permite você usar tipos como argumentos numa função usuário. Por exemplo

```
Function DoMore(tireCenter As infoClientes)
```

declara a variável *tireCenter* como do tipo infoClientes, a qual é uma do tipo usuário, criada usando uma declaração Type.

MANIPULANDO UM NÚMERO DESCONHECIDO DE ARGUMENTOS

Se você não souber quantos argumentos uma função receberá, você pode tornar o último argumento de uma função um *array*, e colocar a palavra chave ParamArray na frente do nome *array*. Também, você deve declará-la como sendo do tipo Variant. Por exemplo

```
Function CheckData(descriçãodaData, ParamArray aData() As
Variant)
```

Declara o último argumento, aData, como um *array*. Ele é preenchido com quaisquer valores que forem fornecidos seguindo a descrição que está armazenada na variável descriçãodaData. Se ParamArray é usada, você não pode especificar nomes de argumento quando você chamar a função. Você deve fornecer valores para que a função aguarda receber os valores.

O que segue ilustra a chamada da função acima:

```
dataOk = CheckData("Info Junho", 23, 44, 55, 21, -43)
```

Neste exemplo, à variável dataDescrip é atribuído o valor " Info Junho". Ao *array* aData é atribuído os valores 23 até -43, e acaba sendo um *array* de uma única dimensão com cinco elementos.

Note que o tamanho do *array* não foi declarado. O *array* resultante será um *array* de uma única dimensão. Para determinar o tamanho do array, usar as funções LBound e Ubound (discutidas anteriormente neste capítulo). Por favor, note que se você tem uma declaração Option Base 1 no topo do seu módulo, o Visual Basic assumirá o primeiro valor do *array* como o elemento 1. Entretanto, o primeiro elemento no *array* ainda será 0, e ele não será assumido como um valor. Isto pode causar problemas a menos que você escreva suas declarações para manipular esta situação.

Ilustrações do uso ParamArray é encontrada no capítulo anterior sobre macros.

RETENDO VALORES APÓS RODAR UMA FUNÇÃO

Se você quiser ter uma variável exceto um argumento retendo seu valor após a função rodar, declare-a com a palavra chave Static em vez da palavra chave Dim.

Se você colocar a palavra Static na frente da palavra Function, isto faz quaisquer variáveis declaradas na função reter seus valores entre chamadas da função. Por favor, note isto aplica às variáveis declaradas na função, não aos argumentos. O que segue ilustra isto:

```
Static Function CheckData(X As Integer)
```

Existem várias situações que fazem as variáveis serem reset numa função que usa a palavra chave Static para reter valores de variável:

· A declaração End por si só é usada para parar a atividade da macro.

· O código Visual Basic na pasta é editado.

· A pasta é fechada.

TORNANDO AS FUNÇÕES PRIVATE

Colocando a palavra Private na frente da palavra Função, você pode restringir a função de modo que ela pode somente ser usada no módulo onde ela está armazenada. Funções private não podem ser usadas em planilhas nem aparecerão no coringa de funções. O que segue ilustra o uso da palavra Private:

```
Private Function CheckData(X, Y)
```

Você pode colocar ambas Static e Private na frente da Função. Private vai primeiro, mas se você esquecer, o Visual Basic automaticamente re-arranjará as palavras na ordem certa! O que segue ilustra uma tal declaração:

```
Private Static Function CheckData(Y, Y)
```

TORNADO AS FUNÇÕES VOLÁTEIS - INSERIR RECALCULATION

Se você escrever uma função que obtenha um ou mais valores indiretamente, tal como referir a uma célula via a declaração na função, a função normalmente não recalculará quando aquelas células mudarem. O que segue ilustra a função que obtém um valor indiretamente:

```
Function PreçodeDesconto(preçoInicial) As Single
      If Range("A1").Value = 0 Then
            PreçodeDesconto = preçoInicial * 0.9
      Else
            PreçodeDesconto = preçoInicial * Range("A1").Value

End If
End Function
```

Na função acima, a referência indireta é a célula A1. Se a seguinte fórmula for colocada na célula G5

```
= PreçodeDesconto (E5)
```

A função acima recalculará sempre que célula E5 é mudada. Isto é porque o E5 é referenciada diretamente pela função. Entretanto, se a célula A1 é mudada a função não recalculará. Isto é porque a função se refere à célula A1 indiretamente. Qualquer referência a uma célula através da lista de argumento é uma referência direta. Qualquer outra referência é uma referência indireta.

Para ter a função recalculate quando qualquer célula numa pasta variar, você precisa incluir a seguinte declaração no topo da função, logo abaixo da declaração da função:

```
Application.Volatile
```

Escreva a função acima num módulo e teste-a com e sem a declaração Application.Volatile seguinte à função declaração.

DETERMINANDO A CÉLULA SOLICITANTE PARA UMA FUNÇÃO

É possível determinar qual célula chamada (usada) a função usando uma das seguintes expressões:

Chamando o objeto célula: Application.Caller
endereço de célula somente: Application.Caller.Address
endereço com arquivo & folha: Application.Caller.Address(External:=True)

A segunda forma acima retornará "$B$4" se a função estivesse na célula B4. A terceira forma retornará "[Book1.XLS]Sheet1!$B$4" se a célula B4 estiver na pasta Book1.Xls e na folha Sheet1.

DETERMINANDO UM ARGUMENTO DE CÉLULA

Declarando um argumento de uma função como do tipo Object ou tipo Variant, você pode usar o argumento para não somente obter valor dos argumentos de célula, mas também usá-lo para obter outros valores nas células relacionadas. A seguinte função ilustra isto:

```
Function DataDescrip(qualquerCélula As Object)
     Application.Volatile
     DataDescrip = qualquerCélula.Value & " " & _
                   qualquerCélula.Offset(0,1).Value
End Function
```

Se a célula A1 contém o nome "João" e célula B1 contém "Eduardo", =DataDescrip(A1) retornará "João Eduardo". Note que a função foi declarada Volatile de modo que ela recalcula quando referências indiretas são mudadas. A referência indireta acima é qualquerCélula.Offset(0,1).Value.

CHAMANDO FUNÇÕES DE UMA OUTRA PASTA

Se a função que você quer usar numa planilha célula está numa outra pasta, você deve colocar o nome da pasta da função (tipo e nome de arquivo) na frente do nome da função. Um ponto de exclamação separará o nome da pasta do nome da função. Por exemplo, se a função CalcPreço estivesse localizada na sua pasta Personal.XLS e você precisa to usá-la numa pasta diferente, você poderá entrar com o seguinte numa célula de planilha:

```
=Personal.Xls!CalcPreço(valor)
```

se o nome do arquivo for comprido, então o nome do arquivo ficaria entre aspas. Se você usar esta aproximação e o arquivo que contém a função não estiver aberto (ou pegar fechado), então a função retornará #NAME?.

Um modo de se evitar a declarar a localização do arquivo de uma função seria criar o que é chamado de um arquivo add-in que contenha a função. Criação de add-ins é discutida no último capítulo. Um outro modo de se evitar a declarar o nome da pasta é ajustar uma referência à pasta contendo a função. Isto elimina também o problema #NAME? se o arquivo da função está fechado.

Para ajustar uma referência:

· Vá para a pasta solicitante
· Vá para qualquer módulo naquela planilha. Se nenhum existir adicione um.
· Enquanto na folha de módulo, selecione Ferramentas, Referências. Isto mostrará o seguinte menu (Por favor, note que suas referências podem ser diferentes daquelas mostradas abaixo).
· Clique na caixa ao lado do arquivo que você quer se referir a ele . Uma marca (checkmark) aparecerá. Se a função está no seu arquivo Personal.XLS, então você clicará neste um. Se o arquivo contendo a função não está carregado no Excel, então usar o botão Browse para especificar o arquivo.
· Feche o painel selecionando OK

Se você re-locate ou renomear um file que fez uma referência, você precisará to re-establish a referência ou else Microsoft Excel será incapaz de avaliar as funções pois ele não pode encontrá-las e retornará o valor erro #NAME?. Veja a descrição minuciosa no Capítulo xxxx para mais detalhes sobre como manipular referências.

FUNÇÕES QUE ATUAM COMO MACROS

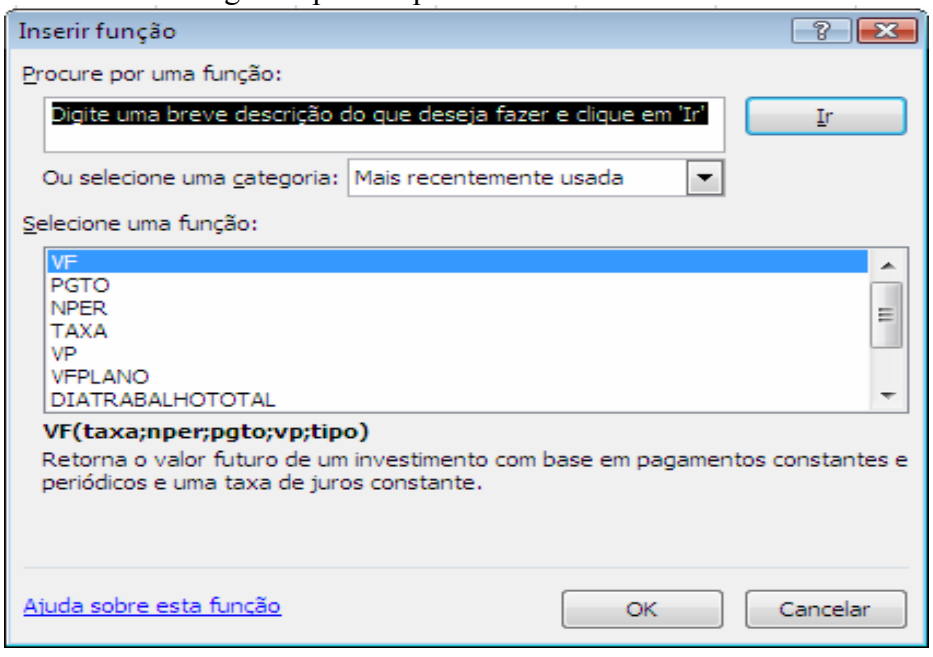
Funções que são usadas numa célula de planilha não pode modificar a planilha. Por exemplo, elas não podem formatar a célula, nem mudar o valor numa célula. Entretanto, funções que são chamadas por uma macro podem modificar as planilhas e outros objetos no Excel. Por exemplo, elas podem deletar linhas, mudar a folha ativa, e selecionar uma célula diferente. Na essência, se a função foi chamada por uma você pode ter que fazer qualquer coisa que uma macro possa fazer.

A habilidade para ter funções que atuam como se fossem macros adiciona um poder tremendo ao Visual Basic. Você pode escrever uma função que faça um número de tarefas tais como adicionar ou deletar linhas, abrir planilhas, e validar dados. E quando a função é feita, ela pode retornar um valor à macro solicitante para indicar seu sucesso ao término de suas tarefas. Por exemplo, ela pode retornar True para todas as tarefas que foram feitas com sucesso, ou False se um problema ocorreu. Também, a função pode parar a atividade da macro usando a declaração End. Se você não quiser uma função para retornar um valor, não coloque os argumentos nos parênteses. Se uma função é usada sem parênteses, então ela é tratada como uma declaração stand alone exatamente como uma chamada a uma macro.

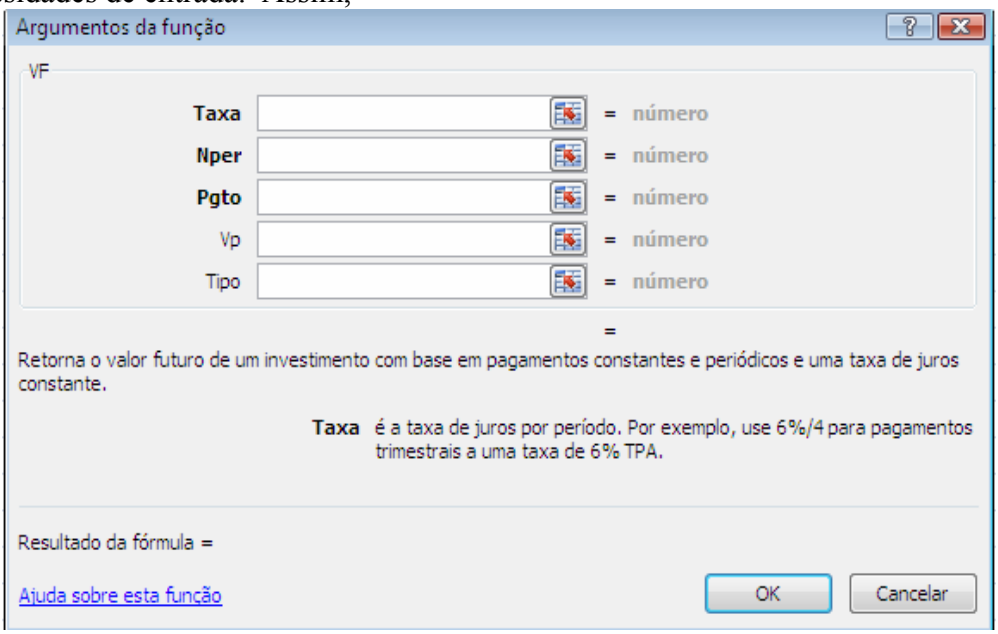
Uma vantagem de se usar funções que atuam como macros é que elas não aparecem na lista de macro quando você selecionar Ferramentas, Macro. Entretanto, elas aparecem na lista de funções definidas pelo usuário.

USANDO O ASSISTENTE DE FUNÇÕES

Se você criou uma função e não a declarou Private, você a encontrará listada sob as funções definidas pelo usuário no assistente de Funções. Se você estiver numa planilha, você pode acessar o assistente de funções selecionando Inserir, Função dos menus Microsoft Excel. O seguinte painel aparecerá:



Quando você selecionar Definida pelo usuário na caixa *pop up* categoria, aparecerão as funções que você criou em qualquer pasta ou suplemento (pasta ou add-in). Quando você selecionar uma função de usuário, o assistente levará você às necessidades de entrada. Assim,



Uma vez preenchidos todos os campos de entrada exigidos, selecione OK. O Microsoft Excel inserirá a fórmula para a função selecionada na caixa de edição. Por favor, note que quando você entrar com os dados nos campos de entrada, o valor mostrado no canto superior direito do painel acima par a função é alterado. Initialmente, ela é configurada como #VALUE.

RECALCULANDO FUNÇÕES

Se você modificar a função, você precisará re-calcular sua planilha para que a modificação seja refletida em quaisquer fórmulas de células, mesmo se o cálculo é ajustado para ser automático. Faça isto pressionando Ctl-Alt-F9 (não apenas F9) todas ao mesmo tempo. Esta aproximação também funciona se você abrir um arquivo contendo células com função fórmulas que foram corretamente calculadas quando você fechou o arquivo, mas elas mostrarão #VALUE! Quando você reabrir o arquivo.

UM MONITORAMENTO DE FUNÇÃO

O que segue é uma função que eu escrevi par monitorar um valor chave em uma das minhas planilhas. Eu não quero mudar o valor do lado de fora de um intervalo estreito. Como eu poderei usar esta função em outro lugar, eu a escrevi de um jeito genérico, de modo que eu posso usá-la para monitorar outras células chaves. Quando a função detectar uma condição for a dos limites, uma caixa de mensagem pops up, diz-me que eu tenho um problema. A razão porque eu uso a função em vez de uma macro é que uma função é re-avaliada pelo Microsoft Excel toda vez que os valores célula monitoradas variarem (se calcular for ajustado para automático).

O que segue é a função:

```
Function EncontrarValor(CelulaObservada As Object, _
                    valorSuperior As Single, _
                    valorInferior As Single, _
                    MsgDeAdvertencia As String)
    EncontrarValor = "Nos Limites"
    If CelulaObservada.Value > valorInferior Or _
      CelulaObservada.Value < valorInferior Then
      EncontrarValor = "Fora Dos Limites"
      MsgBox MsgDeAdvertencia
    End If
End Function
```

Para usar a função para auditor a célula B5, eu entrarei com a seguinte fórmula numa célula próxima:

```
=PESSOAL.XLS! EncontrarValor(B5, 80, 90, "Célula B5 está
Fora dos Limites")
```

Eu sugiro que você escreva a função acima num módulo e prove-a com um valor teste. Ela pode ser uma função que você pode fazer uso dela! Por favor, note que você somente obterá a mensagem de advertência quando o valor variar e quando ele está fora dos limites.

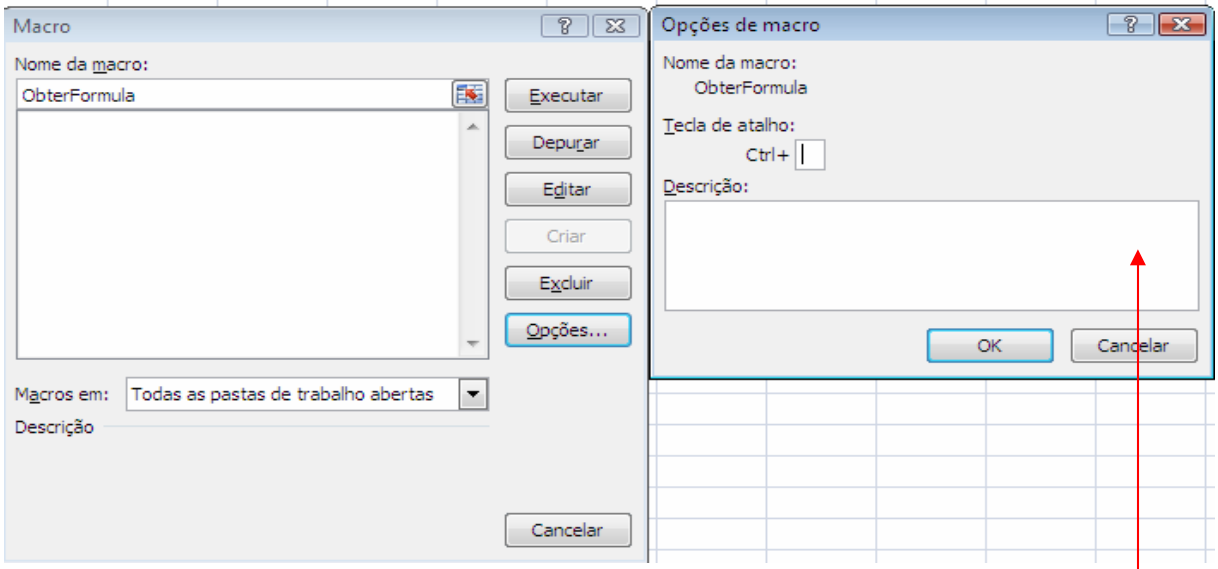
. Se outras variações na pasta não mudarem o valor que está sendo observado, mensagens de advertências adicionais não aparecem, mesmo quando o valor esteja fora dos limites.

Como Fazer Aparecer a Função Criada Na Caixa de Diálogo Inserir Funções do Excel

Como já sabemos , as funções podem ser adicionadas à caixa de diálogo Inserir Funções. Também sabemos que podemos construir uma função, escrevendo os seus códigos no VBEditor. Neste caso elas são chamadas pela sigla inglesa UDF (*user-defined function*)

Entretanto, observe que uma função não é a mesma coisa que uma sub-rotina quando o assunto é macro executável. Portanto, na lista da janela Macro quando escolhemos Ferramentas/Macro/Macros, não aparecem as funções criadas pelo usuário, as UDF. Mas lá na caixa de diálogo Inserir função elas estão presentes na categoria UDF. Como editá-las, então?

Ao digitar o nome da função na janela Macro abaixo, porém, o Excel reconhece e habilita as opções relacionadas à função que digitamos.



Clicando em Opções uma nova janela de diálogo (a da direita) é aberta. Nesta janela você tem a opção de inserir a descrição de sua função.

Para adicionarmos as funções à caixa de diálogo Inserir Funções, por meio de códigos, basta inserir a seguinte linha de comando no seu código:

```
Application.MacroOptions Macro:="MinhaFunção", Category:=1
```

A `category:=1` refere-se à categoria "Financeira", por exemplo. Uma outra característica importante refere-se à **descrição** da função. A descrição auxilia o usuário quanto ao objetivo da função. Na janela da direita da figura acima é só entrar com a descrição.

Para fazer isto por meio de código, fazemos:

```
Application.MacroOptions Macro:="MinhaFunção", _
Desciption:="Esta função soma os itens selecionados.",
Category:=1
```

Para que o código acima funcione é importante que ele esteja em uma sub-rotina que seja executada antes de utilizarmos a função. Se ele for colocado diretamente na função nada ocorrerá.

RESUMO

Você achará que escrever funções para usar me suas planilhas adiciona uma nova dimensão no seu uso do Microsoft Excel. Em vez de escrever complexas expressões If que você deve manter em muitas células numa planilha, você pode agora criar e manter funções para suas planilhas. Funções têm também benefícios semelhantes quando usadas em macros. Isto é um outro significado do poder de adicionar e simplificar as suas macros.

Lição 4: Trabalhando com Databases

Para realmente conseguir a maioria das coisas do VBA funcionando com databases você deve habilitar estas funcionalidades no Excel. Visite o website on Excel e estude os capítulos sobre databases e funcionalidades do databases. Quando você trabalhar num database Excel você deve primeiro verificar que todos os filtros estão desligados. Para este fim você iniciará seu procedimento com estes dois conjuntos de linhas de código. Primeiro selecione qualquer célula dentro do database.

```
Range("A3").Select
   If ActiveSheet.AutoFilterMode = True Then
        Selection.AutoFilter
    End If
   If ActiveSheet.FilterMode = True Then
        ActiveSheet.ShowAllData
    End If
```

Sabendo que um database é um conjunto de linhas e colunas conectadas você pode selecioná-las todas com:

```
Range("A3").Select
     Selection.Currentregion.Select
```

Uma vez feito isto, você pode contar o número de linhas (registros) e o número de colunas (campos) com o seguinte código:

```
varNbRows=Selection.Rows.Count
varNbColumns=Selection.Columns.Count
```

De fato o número de registros é o número de linhas menos um (a linha de título) e aqui está o código:

```
varNbRecords=Selection.Rows.Count – 1
```

Classificando Dados com VBA

Existe um pedaço de código que o Gravador de Macros pode escrever para você e é para classificação de dados. O GM escreverá alguma coisa como esta:

```
Range("J3").Select
    Range("A2:U3").Sort Key1:=Range("J3"), Order1:=xlAscending,
Header:= _
        xlGuess, OrderCustom:=1, MatchCase:=False,
Orientation:=xlTopToBottom, _
        DataOption1:=xlSortNormal
```

A linha 2 é a linha de título e a linha 3 é aquela uma de muitos registros. O GM está somente filtrando o primeiro registro porque eu não selecionei todos eles. Para corrigir esta situação, eu modifiquei o código para isto

```
Range("J3").Select
    Selection.CurrentRegion.Sort Key1:=Range("J3"),
Order1:=xlAscending, Header:= _
        xlGuess, OrderCustom:=1, MatchCase:=False,
Orientation:=xlTopToBottom, _
        DataOption1:=xlSortNormal
```

Eu poderia também mudar a ordem de classificação e eu geralmente modifico o argumento cabeçalho e removo o DataOption porque versões mais antigas do Excel cometerão erros com ela. Assim o código final é:

```
Range("J3").Select
    Selection.CurrentRegion.Sort Key1:=Range("J3"),
Order1:=xlDescending, Header:= _
        xlYes, OrderCustom:=1, MatchCase:=False,
Orientation:=xlTopToBottom
```

Lição 5: Criando Formulários do Usuário (Userforms)

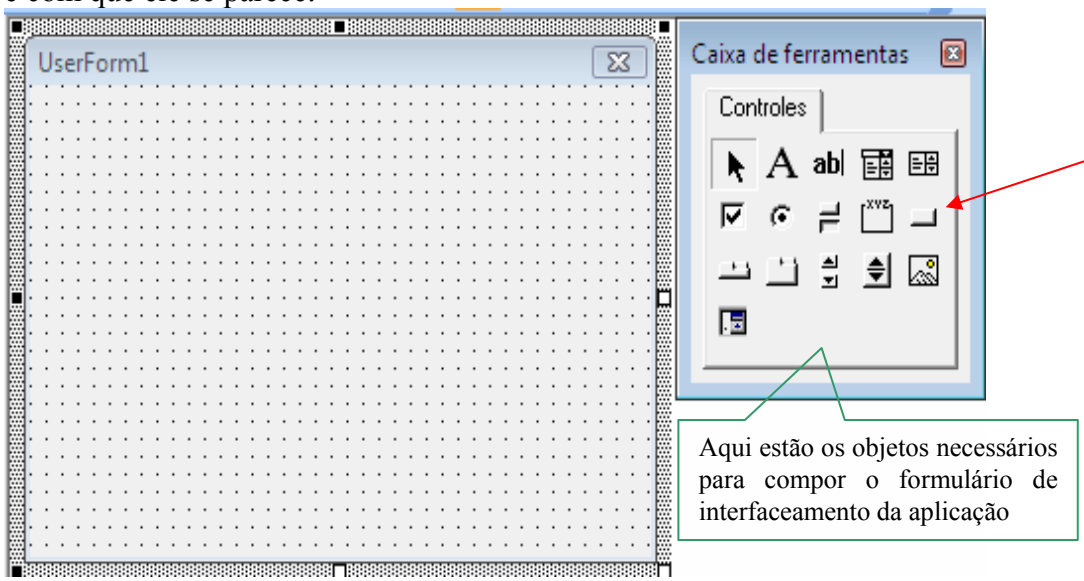
O **formulário** (form) ou userForm é também conhecido como um GUI (Graphical User Interface). O form é usado para requerer *valores*, *parâmetros* e *informação do usuário* para empregar nos procedimentos VBA. Isto é são usados para fazer o interfaceamento entre a pasta de trabalho Excel e o usuário da aplicação, visando facilidade e controle.

Através de um userform podemos fazer com que o usuário da nosa aplicação trabalhe apenas em um ambiente personalizado, sem precisar digitar nas planilhas. A vantagem disso é que podemos criar pequenos e inteligentes sistemas que solicitem dados aos usuário, trate esses dados e devolva os resultados, armazenando informações necessárias em planilhas Excel. Em outras palavras, servem muitas vezes para despoluir as planilhas.

Doze controles diferentes podem ser adicionados ao userform. Eles são chamados: Label, TextBox, ComboBox, ListBox, CheckBox, OptionButton, ToggleButton, Frame, CommandButton, TabStrip, MultiPage, ScrollBar. Eles estão mostrados na Caixa de ferramentas abaixo

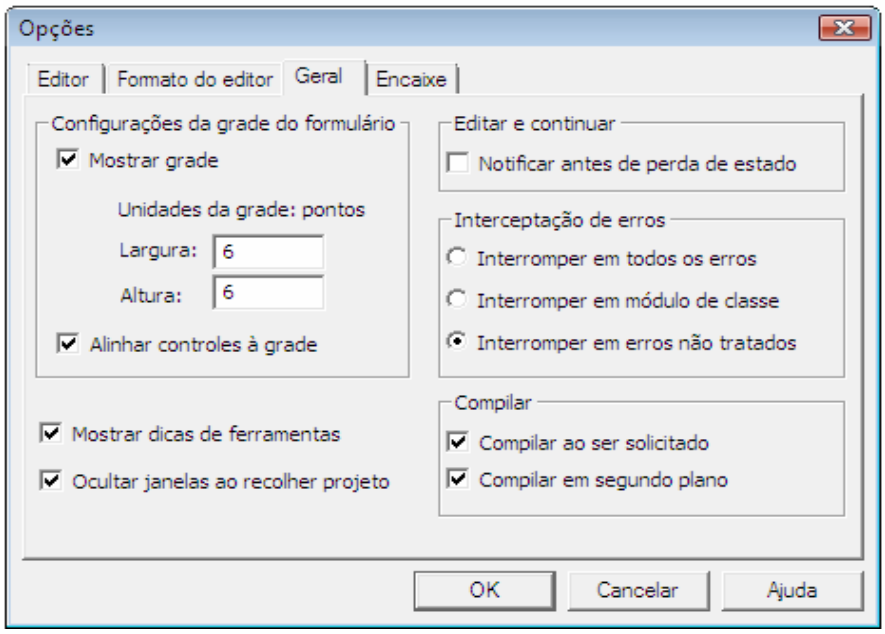
Criando um userForm no VBA do Excel

Para criar um userform, vá ao VB editor (Alt+F11 se você estiver no Excel), selecione o **projeto** (pasta) na janela de projetos (*projects window*) (se não estiver visível, selecione Exibir, Project Explorer ou clicando no ícone na Barra de Ferramentas Padrão do Microsoft Visual Basic) onde você quer armazenar o *userform*, e daí selecione Inserir, UserForm. Você também pode criar um *userform* usando o botão Inserir da barra de ferramentas padrão. Um *userform* é então adicionado, e o que segue é com que ele se parece:



Na lista de componentes do seu projeto o nome do seu *userform* aparece toda vez que você quiser trabalhar no seu userform você dá um duplo clique no seu nome na janela VBAProject.

Como você pode ver, não há controles num *userform* novo. Os objetos são adicionados ao *userform* usando a barra de ferramentas chamada Caixa de Ferramentas, discutida posteriormente. A configuração *default* para um *userform* é aquela de um instantâneo em forma de grade. Quaisquer objetos tais como botões ou listas que você adicionar aí ficarão ancoradas na grade do formulário. Para diálogos simples, isto é ótimo. Entretanto, se você tiver de colocar um grande número de objetos sobre um *userform*, você pode ou desligar a grade, ou mudar o seu tamanho, e/ou ocultá-la, mas manter o instantâneo habilitado. Para fazer qualquer coisa desta, selecione Ferramentas, Opções, e a guia Geral:



Use as configurações da grade do formulário para mudar o tamanho da grade ou para removê-la. Uma unidade de grade pequena, como 3, permite uma colocação muito boa de objetos. E, você pode desligar a grade, e ainda reter o alinhamento na propriedade grade (Alinhar controles à grade). Isto é o melhor de ambos mundos.

Você pode fechar a barra Caixa de ferramentas clicando no seu "**X**" e chamá-la de volta clicando no ícone da caixa de ferramentas na barra de ferramentas 

Para deletar um *userform*, primeiro selecione o form (dê um duplo clique no seu nome lá no Project Explorer). Daí, selecione Arquivo, Remover UserForm. Se você mudou o nome do *userform*, o nome aparecerá como parte do item de menu remover no lugar da palavra UserForm.

Testando o Userform

Toda vez que você quiser ver seu userform (produto acabado ou funcionando em progresso) em ação ou quando você quiser testá-lo esteja seguro de que o userform está selecionado (e não um dos controles) clique no botão "Executar" na barra de ferramentas ou tecle F5. Seu userform se mostrará com o Excel no fundo (background). Para retornar ao VB Editor apenas clique no "**X**" do userform.

Tab Order

Quando as pessoas usam um form elas precisam mover de um controle para o próximo entrando com um valor num deles e clicando "Enter" ou "Tab". Para se assgurar que o usuário move de um controle ao próximo você precisa definir a tab order. Para fazer isto, clicar com o botão direito do mouse no próprio form e selecionar o item "Tab Order". Siga as instruções. O primeiro controle na lista será aquele um que está ativo (cursor brilhante) quando o form está ativado. Levando os controles que não serão usados pelo usuário para o final da lista. Na seção "Controles e suas Propriedades" abaixo ver como desativar um controle "tab ".

Carregando um Userform

Para carregar um formulário em sua planilha, precisamos gerar uma macro que tenha os comandos de inicialização do userform, e devemos executar essa macro pelas formas já vistas anteriormente (por *botões de controles* e *eventos*).

Para carregarmos um formulário, usamos o comando Load e, para exibí-lo, usamos o método Show. Vejamos o exemplo a seguir, que carrega o Userform1 na inicialização da pasta de trabalho xxxxx.xls. Para isso, usamos o **evento** Open da Pasta de trabalho:

```
Private Sub Workbook_Open( )
     Load UserForm1
     UserForm1.Show
End Sub
```

Da mesma forma, podemos ocultar um formulário pelo método Hide e descarregar um formulário da memória com o comando Unload. Não é necessário ocultar um formulário antes de descarregá-lo da memória.

Poderíamos fazer um outro exemplo em que o formulário é carregado e exibido através de botões de controle, por exemplo o botão de comando .

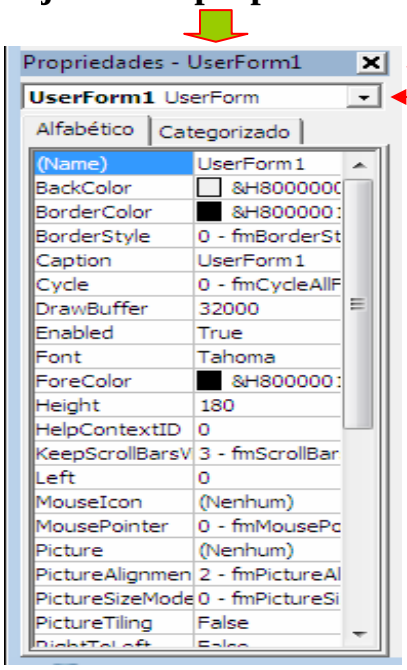
As Propriedades do Userform

Como um **objeto** Excel, um userform possui propriedades, métodos e eventos.

Quando você der um duplo clique no nome do form na janela VBAProject do VBE o form aparece e a janela Propriedades lhe mostra as propriedades do form.

Para mudar o nome ou outra propriedade de um *userform*, selecione o form e pressione F4, ou selecione Exibir, Propriedades. Um diálogo como o que segue aparecerá. Clicando no seu botão direito superior fecha o diálogo.

janela de propriedades



Este *drop down* permite você mudar para outros objetos e rever e/ou mudar suas propriedades.

Do Excel 2000 em diante passou a existirem duas propriedades de *userform* adicionais: ShowModal e RightToLeft. A propriedade ShowModal pode ser configurada para False, o que permite você mostrar um *userform* e permite que a sua macro continue a rodar. Por favor note que se você usar estas propriedades e trocar a macro com alguém que esteja rodando o Excel 97, um erro ocorrerá.

A janela de propriedades acima mostra a lista Alfabético, isto é, em ordem alfabética. Clicando na guia Categorizado ela agrupa as propriedades relacionadas e lhe permite comprimir ou expandir as propriedades que são mostradas, usando o pequeno + ou caixas que aparecem à esquerda das categorias.

Propriedade (Name)

Você pode mudar o nome "(Name)" do form. Quando você nomeia um form sempre use o prefixo "frm" como em "frmDatabase" e seja tão descritivo para você quanto possa ser de modo que seu código será fácil para ler. Sempre use uma ou mais letras maiúsculas no nome. Quando você escrever "frmDatabase.Show" em letras minúsculas, o Excel colocará em maiúsculo quaisquer letras deixando você saber que o nome está com ortografia errada.

Por favor note que os espaços não são permitidos.

Propriedade BackColor

Essa propriedade define a cor do fundo (background) do formulário. Abrindo a flecha da caixa de combinação, aparece as cores da paleta do sistema.

Propriedade BorderColor

Define a cor da borda do formulário.

Propriedade BorderStyle

Determina o estilo da borda.
Pode ser : frmBorderStyleNone ou frmBorderStyleSingle.

Propriedade Caption

O nome que vai aparecer na faixa azul no topo do formulário.

Propriedade Enabled

Ativa ou desativa o formulário. Desativar um formulário significa não poder fazer coisa alguma com ele; portanto, tome cuidado com essa propriedade.

Você pode desabilitar o "X" do form com a propriedade "Enable" definida como "False".

Propriedade Font

Exibe a caixa padrão de fontes do sistema, configurando como os textos serão exibidos nos outros objetos adicionados ao formulário.

Propriedade ForeColor

Define a cor daquilo que será impresso no formulário.

Propriedade Height

Determina a altura do formulário.

Propriedade Left

Demarca a distância do formulário ao canto esquerdo da tela.

Propriedade MouseIcon

Define um ponteiro do mouse personalizado quando estiver sobre o formulário. Para que esse cursor fique ativo, na propriedade MousePointer, você deve escolher a opção 99 – frmMousePointerCustom.

Propriedade MousePointer

Você pode ter o ponteiro do mouse mudado quando estiver sobre o form com esta propriedade.

Propriedade Picture

Opta por uma figura como fundo do formulário.

Propriedade PictureSizeMode

Determina a forma que a figura será disposta no formulário

Propriedade StartPosition

Indica a posição inicial do formulário. Por default o userform aparece no centro da tela. Se você quiser ver ele mostrado em algum outro lugar defina a "Start" para "0-Manual" e use as propriedades "Top" e "Left" para definir a nova posição.

Propriedade Top

Define a distância entre o topo do formulário e o topo da planilha.

Propriedade Width

Delimita a largura do formulário.

Propriedade Zoom

Aumenta ou diminui a distância de visualização dos componentes do formulário.

A seguir apresentadaremos os MÉTODOS a serem usados para este **objeto** Userform:

Método Hide

Como dissemos anteriormente, ele oculta um formulário sem descarregá-lo da memória.

Método Show

Exibe um formulário quando já carregado na memória.

Método PrintForm

Imprime a imagem do formulário na impressora padrão do sistema.

A seguir apresentadaremos os EVENTOS a serem usados para este **objeto** Userform. Os eventos que serão mostrados na seqüência servem para uma gama de outros controles e, portanto, podemos, mais à frente, apenas expandir o conceito.

Evento Activate

Ocorre quando o formulário for ativado.

Evento Click

Dá-se toda vez que o formulário for clicado.

Evento DoubleClick

Acontece sempre que o formulário receber um clique duplo.

Evento Deactivate

Ocorre quando um formulário for desativado.

Evento Initialize

Sucede-se toda vez que um formulário for carregado para a memória.

Evento KeyDown

Ocorre quando uma tecla é pressionada. Sua declaração possui dois argumentos: KeyCode As MSForms.ReturnInteger e Shift As Integer. KeyCode retorna o código da tecla pressionada; Shift retorna 1 caso a tecla Shift esteja pressionada e 0, caso contrário.

Evento KeyUp

Ocorre quando uma tecla é solta. Os argumentos se comportam como no evento KeyDown.

Evento KeyPress

Ocorre quando uma tecla é presionada também, mas retorna o código da tabela ASCII da tecla.

Evento MouseDown

Acontece quando um botão do mouse é pressionado. Possui os seguintes argumentos: Button As Integer, Shift As Integer, X As Single, Y As Single, onde Button representa o botão clicado; Shift representa o estado da tecla Shift; X e Y representam a posição atual do mouse.

Evento MouseUp

Dá-se quando um botão do mouse é solto. Possui os mesmos parâmetros de MouseDown.

Evento MouseMove

Ocorre quando o mouse movimenta-se pelo formulário.

Evento QueryClose

Realiza-se antes que o formulário seja fechado. Possui o argumento Cancel para cancelar o evento caso seja desejado.

Evento Resize

Ocorre sempre que o formulário seja redimensionado

Evento Terminate

Sucede-se após o formulário ser fechado.

Evento Zoom

Ocorre quando há uma mudança da propriedade Zoom do formulário.

Lição 6: Adicionando Controles aos UserForms

A Caixa de Ferramentas do Userform

A Caixa de Ferramentas para *userforms* é chamada barra de ferramentas Caixa de ferramentas. Ela se parece com o que segue:

O que segue são o que os diferentes controles fazem:

Linha Um: Selecionador de Objetos
Inserir um Rótulo (Label)
Inserir uma Caixa de Texto
Inserir uma Caixa de Combinação (Combo)
Inserir uma Caixa de Listagem
Linha Dois Inserir uma Caixa de Verificação (Check)
Inserir um Botão de Opção
Inserir um Botão Alternância
Inserir um Moldura (Frame)
Inserir um Botão
Linha Três Inserir uma Tab Strip
Inserir uma Multi-página
Inserir uma Barra de Rolamento
Inserir um Botão de rotação
Inserir uma Imagem
Linha Quatro Inserir um ReferenceEdit
Caixa

Adicionando Controles

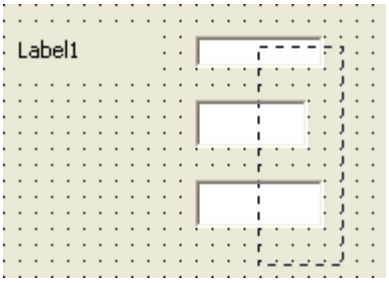
Para adicionar controles ao form você clica com o botão esquerdo do mouse sobre aquele controle que você quiser na caixa de ferramentas, segure o botão apertado e arraste o controle sobre o form. Você pode agora expandir o controle para o tamanho desejado. Uma vez que todos os seus controles estejam no form você clica com o botão esquerdo do mouse neles e na janela propriedades você pode mudar as propriedades do controle selecionado. Você também pode clicar com o botão direito do mouse neles e selecionar "Propriedades".

Gerenciando Controles

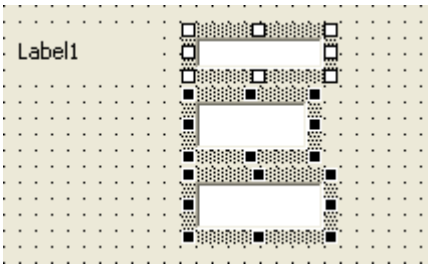
Você pode mover os controles clicando neles segurando-os e movendo-os ao redor. Você pode redimensioná-los selecionando-os e usando os diferentes

identificadores ao redor deles. Você pode copiá-los e colá-los clicando com o botão direito do mouse sobre eles e escolhendo o item de menu certo.

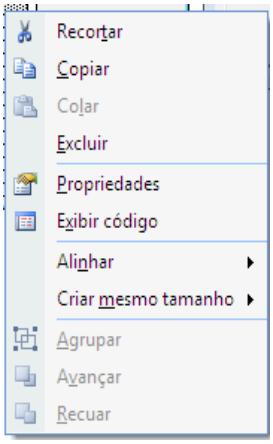
Uma vez tendo adicionado seus controles você poderá querer alinhar alguns deles ou redimensionar alguns de modo que eles fiquem todos do mesmo tamanho. Para fazer isto você primeiro precisa selecionar muitos controles ao mesmo tempo. Para fazer isto clique com o botão esquerdo do mouse no form próximo a um dos controles que você quiser selecionar. Segure e arraste desenhando uma moldura que inclua muitos controles.



Quando soltar o botão todos os controles que você tocou pela moldura estarão selecionados.



Clique com o botão direito do mouse em qualquer um dos controles e este menu contextual aparece:



Você pode então alinhar os controles ou torná-los do mesmo tamanho

Os Controles e suas propriedades

Existem 12 diferentes controles que podem ser adicionados ao userform ou a uma planilha regular. Aqui estão aqueles mais importantes e suas propriedades.

Quando você selecionar um controle sobre o userform suas propriedades estão mostradas na Janela de propriedades. As propriedades e o número de propriedades diferem dependendo do tipo de controles que você selecionou. Estas propriedades

podems ser mudadas na Janela de propriedades do VB editor ou pode ser mudada programaticamente num procedimento VBA.

Defina a propriedade "Name" de todos os controles que você se referir nos seus procedimentos VBA. Eu o encorajo a usar prefixos e algumas letras maiúsculas nos seus nomes (cbxCitx, txbNomeCidade). Seja tão descritivo quanto possível para tornar seu código mais claro. Os prefixos que eu uso são: botão de comando (cmb), rótulos (lbl), caixas de combinação (cbx), caixas de texto (txb), caixa de listagens (lbx), caixas de verificação (check boxes) (ckb), botão de rádio (rdb), botões de alternância (toggle) (tbt), molduras (frames) (fra), tab strips (tsp), multi páginas (mpg), barras de rolamento (scroll bars) (scb), botões de rotação (spin) (spb), imagens (img) e ref edits (rfe). Os controles que particularmente precisam ser bem chamados são os controles cujos valores eventuais você estará usando nos seus procedimentos como as caixas de texto, as caixas de listagens, as caixas de combinação, os botões de opção e as caixas de verificação. Por exemplo:

```
Range("A1").Value = txbNomeCidade.Value
```

tomará o valor entrado pelo usuário na caixa de texto chamada tbxNomeCidade e entrará com ele na célula A1 da folha ativa.

Para a maioria dos controles existem estas propriedades gerais que lhe permitem definir a fonte, a cor da fonte, a cor do fundo (background), o tipo de background, o tipo de borda e outras caracteísticas de *design*.

A propriedade "Caption" contém o texto que é mostrado num rótulo, num botão de comando, numa check box, num botão de opção, num frame, numa tab strip ou numa multi página.

Abaixo estão as outras propriedades interessantes que eu uso com diferentes controles.

Rótulos

Os rótulos são controles passivos significando que o usuário nunca realmente atua nele. Está lá para informar ao usuário e para rotular outros controles como caixas de texto, caixas de combinação ou caixa de listagens. As outras propriedades interessantes do rótulo são:

- TabStop para tornar o controle invisível para as teclas "Tab" e "Enter" (Ver Tab Order acima).
- WordWrap para habilitar a escrever mais que uma linha num rótulo.

Botões de Comando

Os botões de comando são geralmente colocados no fundo do form e servem para completarem a transação para a qual o formulário foi criado. Os *captions* dos botões são geralmente "Ir", "Executar", "Submeter", "Cancelar".

As outras propriedades interessantes do botão de comando são:

- WordWrap para ser capaz de escrever mais que uma linha sobre um botão,
- ControlTipText que gera uma pequena caixa de comentário quando o usuário mover o mouse sobre o controle. Você pode usar esta propriedade para dar explicações e instruções sobre o botão de comando,

- Enabled e Visible são propriedades que você pode mudar programaticamente para desabilitar ou desenhar um botão comando invisível seguinte a uma seleção anterior num outro controle do userform,
- TabIndex é a propriedade que você muda pela functionalidade "Tab Order" como mostrado acima na seção "Tab Order".

Caixas de Texto (Text Boxes)

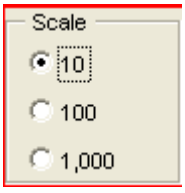
A caixa de texto é o controle mais simples para exigir uma entrada do usuário. O usuário digita alguma coisa nela e este valor pode então ser usado no seu procedimento VBA.

As outras propriedades interessantes das caixas de texto são:
- WordWrap habilitar a escrever mais do que uma linha num botão,
- ControlTipText que gera pequenas caixas de comentário quando o usuário move o mouse sobre o controle. Você pode usar esta propriedade para dar explicações e instruções a respeito do botão de comando,
- Enabled e Visible são propriedades que você pode mudar programaticamente para desbilitar ou desenhar um botão de comando invisível seguido de uma seleção prévia num outro controle do userform,
- TabIndex é uma propriedade que você muda pela functionalidade "Tab Order" como mostrado acima na seção "Tab Order".
- PasswordChar aquela que lhe permite escolher um caractere para os usuários submeterem *passwords*,
- MaxLength para limitar o número de caracterees entrados pelo usuário,
- Value ou Text que é o texto mostrado na caixa de texto quando o userform for ativado ("Entrar com seu Nome" por exemplo)

Molduras (Frames)

Os Frames são também um controle passivo. Frames são usados para melhorarem o *layout* do userform. Você pode usá-los ao redor de um grupode botões de opção ou caixas de verificação ou ao redor de um grupo de caixas de texto ou caixas de combinação que tenham alguma coisa em comum.

Se você tiver dois conjuntos de 3 botões de opção num userform e você não colocá-los dentro de um frame eles todos funcionanrão juntos e você pode escolher somente um dos seis. Se você colocar cada conjunto num frame você pode escolher um dos três em cada conjunto.



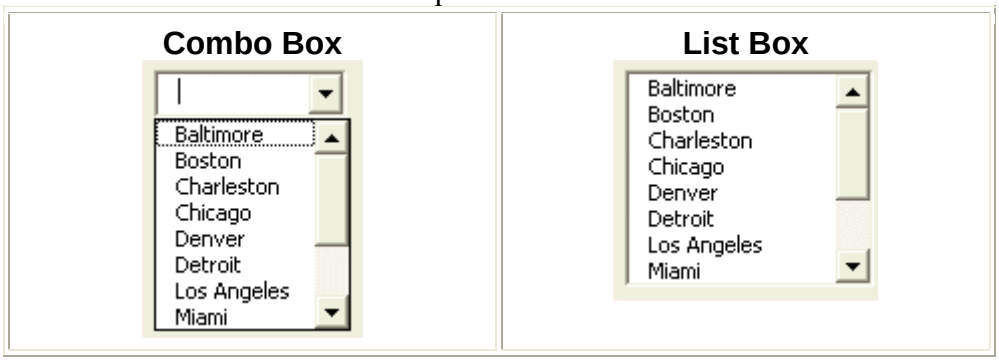
Caixas de Verificação (Check Boxes) e Botões de Opção (Option Buttons)

As caixas de verificação e os botões de opção são ambos usados para oferecer ao usuário uma escolha. A principal diferença entre caixas de verificação e botões de opção é que se você tiver 5 de cada num form um usuário pode marcar todas as 5 caixas de verificação mas pode somente selecionar um dos botões de opção. Veja a nota acima sobre frame e botões de opção. Se você não quiser usar frame para criar grupos de

botões de opção você precisará usar a propriedade "GroupName" dos botões de opção. Todos os botões de opção dentro do mesmo GroupName funcionam juntos.
As outras propriedades interessantes das caixas de verificação e botões de opção são:
- WordWrap ser capaz de escrever mais do que uma linha na caption,
- ControlTipText a qual gera uma pequena caixa de comentário quando o usuário mover o mouse sobre o controle. Você pode usar esta propriedade para dar explicações e instruções a respeito do botão de comando,
- Enabled e Visible são propriedades que você pode mudar programaticamente para desabilitar ou renderizar invisível um botão de opção ou uma caixa de verificação seguida de uma slelção prévia num outro controle do userform,

Caixas de Combinação (Combo Boxes) e Caixas de Listagens (List Boxes)

A diferença entre a caixa combo e a caixa de listagem é que a caixa combo é uma lista drop-down e o usuário pode submeter um único valor um dos valores da lista *drop-down* ou qualquer outro valor. A caixa de listagem mostra um certo número de valores com ou sem a barra de rolamento (scroll bar) e o usuário pode selecionar nela um ou mais valores mas não um valor que não está na lista.



Caixas Combo

As propriedades interessantes das caixas de combinação são:

- RowSource Os valores que deverão aparecer na lista drop-down da caixa combo são submetidos na propriedade RowSource. Por exemplo Sheet1!A1:A12 contratará a lista com os valores residindo nas células A1 até A12 da folha com o Caption "Sheet1". As regras para submeter a propriedade RowSource é a *caption* da folha onde a lista reside seguida por um ponto de exclamação (!), os endereços da primeira célula, dois pontos e os endereços se a última célula.

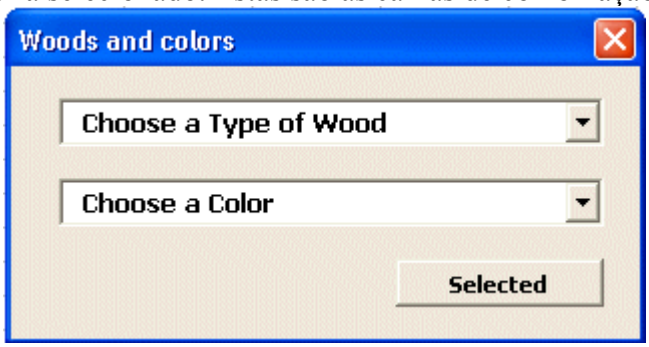
NOTA IMPORTANTE: Se houver um espaço ou um caractere especial na caption da folha onde a lista reside você deve cercá-lo com aspas simples como em 'Esta folha'!A1:A12.
- ListRows é o número de valores mostrados na lista *drop-down*. Se você mostrar menos do que a lista complete uma barra de rolamento (*scroll bar*) é adicionada automaticamente.

- MatchRequired é por default definida para false significando que o usuário pode

submeter qualquer valor na caixa combo. Se você quiser que o usuário fique limitado aos valores da lista defina esta propriedade para True.
- Text deverá conter um valor mostrado na caixa combo quando o userform estiver ativado (Selecione uma Cidade, por exemplo).
- ControlTipText que gera uma pequena caixa de comentário quando o usuário mover o mouse sobre o controle. Você pode usar esta propriedade para dar explicações e instruções a respeito da caixa combo.
- ColumnCount é o número de colunas de valores que você quiser mostrar na drop-down lista. Por exemplo se você quiser mostrar parte número e parte nome na lista você submeterá uma RowSource como Sheet1!A1:B12 com a parte números na coluna A e a parte nomes na coluna B
- ColumnWidth é a largura de todas as colunas quando mostrada na lista drop-down da caixa combo.
- BoundColumn é a coluna da qual o valor é desenhado para o valor final da caixa combo. Por exemplo se a parte número estiver na coluna A do RowSource e a parte nome estiver na coluna B do RowSource quando o usuário selecionar um valor somente a coluna A ou coluna B será igual ao final valor da caixa combo. Assim se você definir o valor de BoundColumn para 1 a parte número torna-se o valor final. Se você definir BoundColumn para 2 a parte número torna-se o final valor.

Caixas Combo em Cascata

Aqui está um simples form exigindo do usuário selecionar um tipo de madeira para seu revestimento de chão e daí uma cor. Nem todas as cores estão disponíveis para todos os tipos de madeira. Quando o tipo de madeira for selecinado na primeira caixa combo o usuário deverá somente ser capaz de selecionar uma cor que está disponível para o tipo de madeira selecionado. Estas são as caixas de combinação em cascata.



Você define a lista de valores para a primeira caixa combo na propriedade RowSource como descrito acima. Para a lista de valores na segunda caixa combo alguma programação é necessária. Com o código apropriado você pode mesmo evitar o botão "Selected". O procedimento move tão logo o usuário tenha selecionado uma cor.

Você também pode validar a entrada para ficar certo que o usuário selecionou ambos um tipo de madeira e a cor.

Veja o capítulo sobre o código VBA para userforms e controles.

Caixas de Listagens

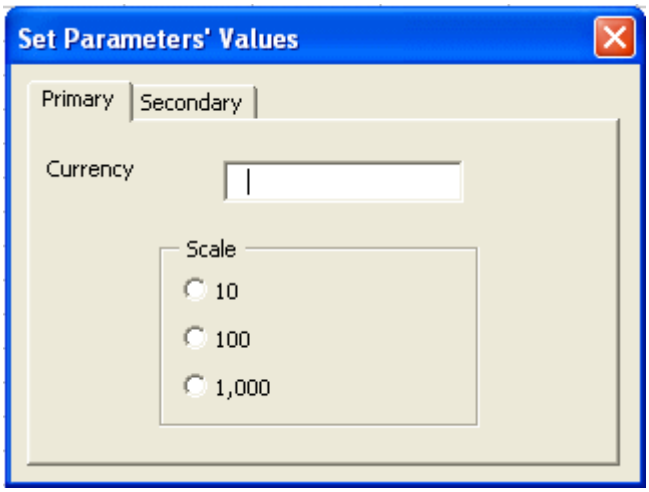
As propriedades interessantes das caixas de listagens são:
- RowSource Os valores que deverão aparecer na lista drop-down da caixa combo são submetidas na propriedade RowSource. Por exemplo Sheet1!A1:A12 empregará a lista

com os valores residindo nas células A1 até A12 da folha com o Caption "Sheet1". As regras para submeter a propriedade RowSource é a caption da folha onde a lista reside seguida por um ponto de exclamação (!), os endereços da primeira célula, dois pontos e os endereços se a última célula.

NOTA IMPORTANTE: se houver um espaço ou um caractere especial na caption da folha onde a lista reside você deve cerça-lo com aspas simples como em 'Esta folha'!A1:A12.

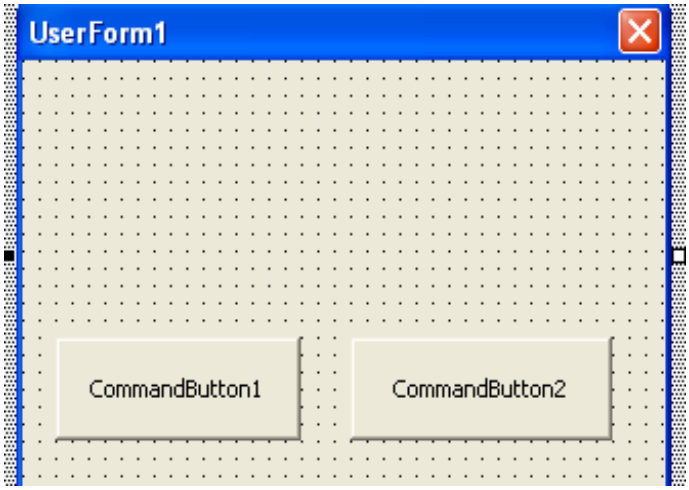
- MultiSelect é definida para 1 se você quiser que o usuário seja capaz de selecionar muitos valores da lista.
- Height O número de valores mostrados na lista dependerão da altura da caixa de listagem. Você pode definir a altura aqui ou no próprio userform esticando-o. Se o número de valores no seu RowSource for maior que aquele que pode ser mostrado na caixa de listagem uma barra de rolamento é adicionada automaticamente.
- Text deverá conter o valor mostrado na caixa combo quando o userform estiver ativado (Selecione uma Cidade, por exemplo).
- ControlTipText que gera uma pequena caixa de comentário quando o usuário move o mouse sobre o controle. Você pode usar esta propriedade para dar explicações e instruções a respeito da caixa combo.
- ColumnCount é o número decolunas de valores que você quiser mostrar na caixa de listagem. Por exemplo se você quiser mostrar parte número e parte nome na lista submeterá umaa RowSource como Sheet1!A1:B12 com a parte números na coluna A e a parte nomes na coluna B
- ColumnWidth é a largura de todas as colunas mostradas na lista *drop-down* da caixa combo.
- BoundColumn é a coluna da qual o valor é desenhado para o valor final da caixa combo. Por exemplo se a parte número está numa coluna A do RowSource e a parte nome está na coluna B do RowSource quando o usuário selecionar um valor somente a coluna A ou coluna B será iguala o valor final da caixa combo . Assim se você definir o valor da BoundColumn par 1 a parte número torna-se o valor final. Se você definir BoundColumn para 2 a parte número torna-se o valor final Multi Páginas

O controle multi páginas é usado para desenvolver userforms mais elaborados onde o usuário pode ver um diferente conjunto de controles em cada uma das multiplas páginas. No exemplo abaixo o usuário pode escolher um conjunto de parâmetros primários numa página e a conjunto de parâmetros secundários na segunda página.

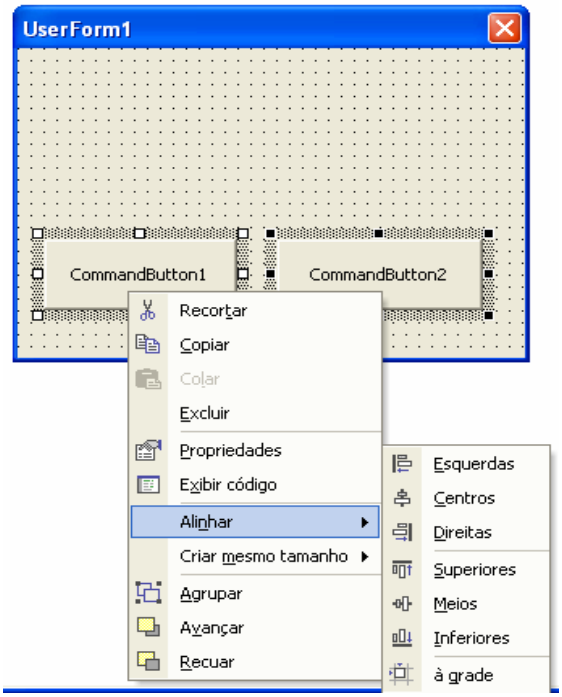


BOTÕES DE COMANDO NOS USERFORMS E MOSTRAR/OCULTAR USERFORMS

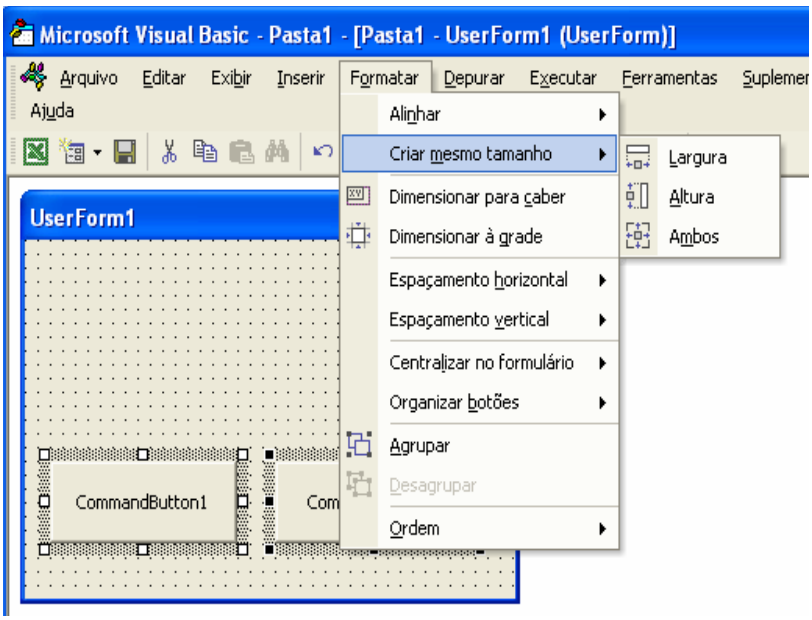
Para demonstrar como mostrar um *userform* e como os botões funcionam, crie um *userform* em branco e desenhe dois botões nele usando a ferramenta botão na barra de ferramentas caixa de ferramentas. Quando você fizer isto, o diálogo deverá se parecer como este:



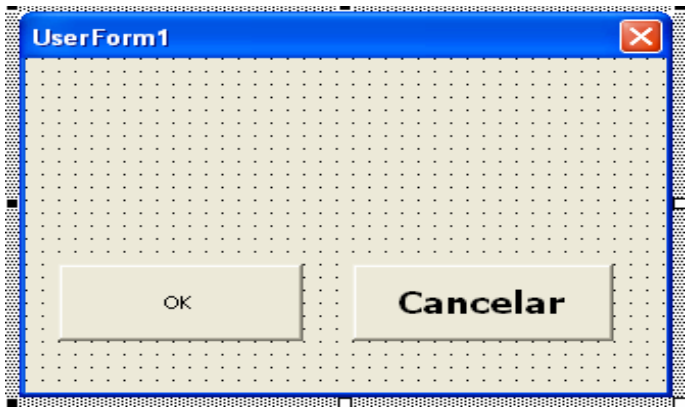
Se os seus botões não estiverem alinhados, clique num botão e pressione F4 para mostrar a janela de propriedades. Daí então verifique os valores da posição. Use o *drop down* no topo para saltar entre os botões. Ou, clique no botão **flecha** na barra de ferramentas e selecione ambos os botões clicando e desenhando uma caixa de seleção ao redor dos botões. Então, com o botão direito do mouse selecione Formatar, Alinhar e escolha as margens de alinhamento.



Você pode também selecionar Formatar, Criar mesmo tamanho para o tamanho dos objetos.



O próximo passo é mudar o texto nos botões para OK e Cancelar. Você pode fazer isto ou clicando na caixa, ou mudando o texto da Caption na janela de propriedades. Após mudar os nomes, mudar a fonte em cada caixa selecionando um botão de cada vez, mostre a janela propriedades (F4), e dê um duplo clique na propriedade fonte. Isto mostra um diálogo que lhe permite mudar a fonte. Mude a fonte para 14 pontos, negrito. O que segue é agora com o que o *userform* se parece:



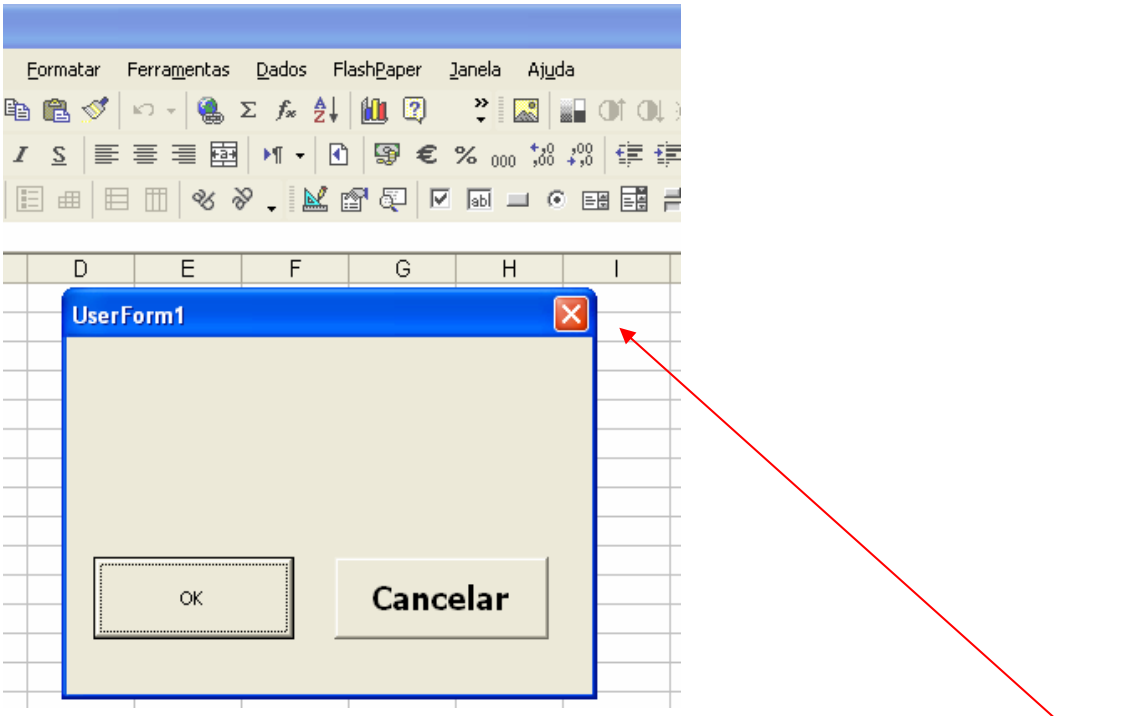
A seguir, mude os nomes dos botões na caixa propriedades para Botão_OK e Botão_Cancelar. Isto os torna mais simples de se trabalhar ao invés dos seus nomes *default*.

O próximo passo é criar um módulo que mostrará o diálogo. Para fazer isto selecione Inserir, Módulo no menu do VB editor. Inserir o seguinte código neste módulo:

```
Sub Mostrar_Meu_UserForm()
UserForm1.Show
'código que armazena a informação do userform
Unload UserForm1
'o que está acima remove o userform da memória
End Sub
```

Se o nome do seu *userform* não for "userform1", use seu nome dado ao *userform* no código acima. Você pode mudar o nome de um *userform* daquele que fora atribuido a ele pelo VB editor selecionando o *userform*, mostrando a janela de propriedades e mudar a propriedade Name. Nenhum espaço ou caracteree especial são permitidos.

Execute esta macro pressionando F5, ou clicando no botão Executar. O *userform* será mostrado.



Porém, os botões OK e Cancelar não funcionam! Isto é porque nenhuma macro fora atribuida a eles. Para liberar o userform neste caso, você deve clicar no botão fechar (o pequeno botão X) no canto superior direito do diálogo mostrado. Você pode também fechar um *userform* indo ao Visual Basic editor e clicando no botão Redefinir (o quadradinho em azul na barra de ferramentas do VB editor se ela estiver aberta).

Para fazer os botões OK e Cancelar responderem, você precisa fazer o seguinte:

- Dê um duplo clique no botão OK. O Excel designará uma macro ao botão chamada de macro " Botão_OK_Click", e coloca você no módulo contendo a nova macro de modo que você possa editá-la. (Este módulo é diferente de um módulo normal, ver discussão no Código de Módulo abaixo).

Coloque o seguinte código nesta macro:

```
Sub Botão_OK_Click ()
'Definir uma variável pública para indicar que botão foi clicado
bResposta = True
'ocultar o form
UserForm1.Hide
End Sub
```

- Dê um duplo clique no botão Cancelar. O Excel atribuirá uma macro ao botão chamada de macro " Botão_Cancelar_Click". Coloque as seguintes declarações nesta macro:

```
Sub Botão_Cancelar_Click ()
'definir a variável pública para indicar qual botão foi clicado
```

```
bResposta = False
'ocultar o form
UserForm1.Hide
'Você pode também usar a declaração " Me.Hide"
End Sub
```

- Modique a macro Mostrar_Meu_UserForm para o seguinte:

```
Sub Mostrar_Meu_UserForm()
UserForm1.Show
'remover o form da memória (se valores do forem
'necesários você primeiro consultará o objeto do form para
'definir a propriedade e daí remover o form.
Unload UserForm1
If Not bResposta Then
MsgBox "Selecionou Cancelar"
End
End If
If bResposta Then
MsgBox "selecionou OK"
End If
End Sub
```

- Finalmente, coloque a seguinte declaração no topo do módulo contendo a macro Mostrar_Meu_UserForm. Ela poderá ficar logo abaixo de quaisquer declarações Option.

```
Public bResposta As Boolean
```

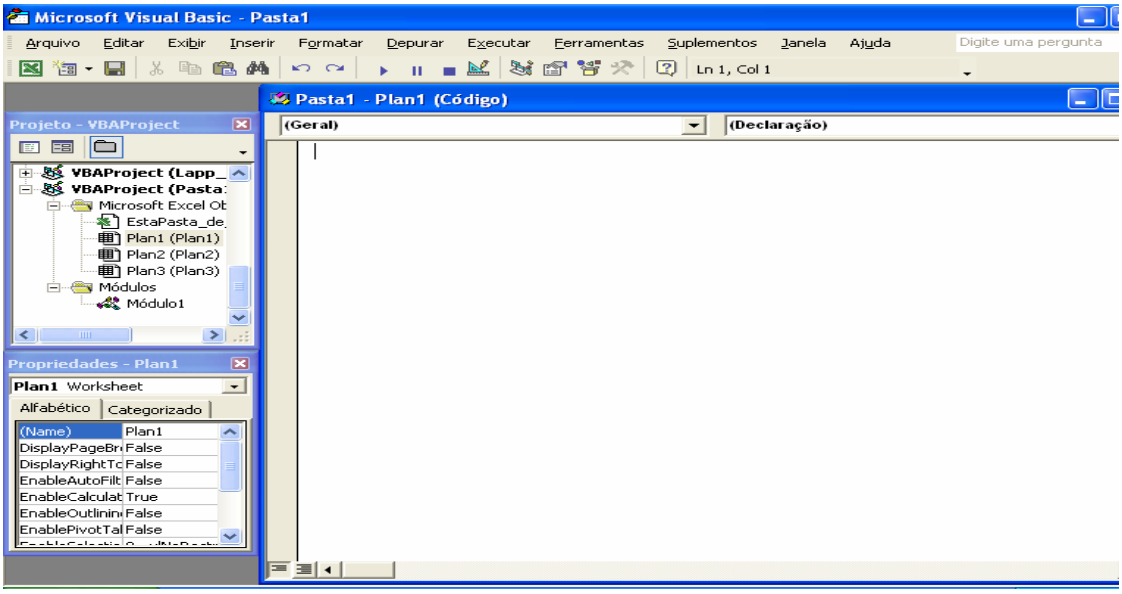
Agora, quando você rodar a macro Mostrar_Meu_UserForm e clicar no botão OK ou Cancelar, o código associado com o clique do botão executará. Neste caso, o botão oculta o *form* e ajusta a variável para pública. Na macro principal, o *userform* é removido da memória e o código determina qual ação tomar baseado no valor da variável pública

Um Exemplo

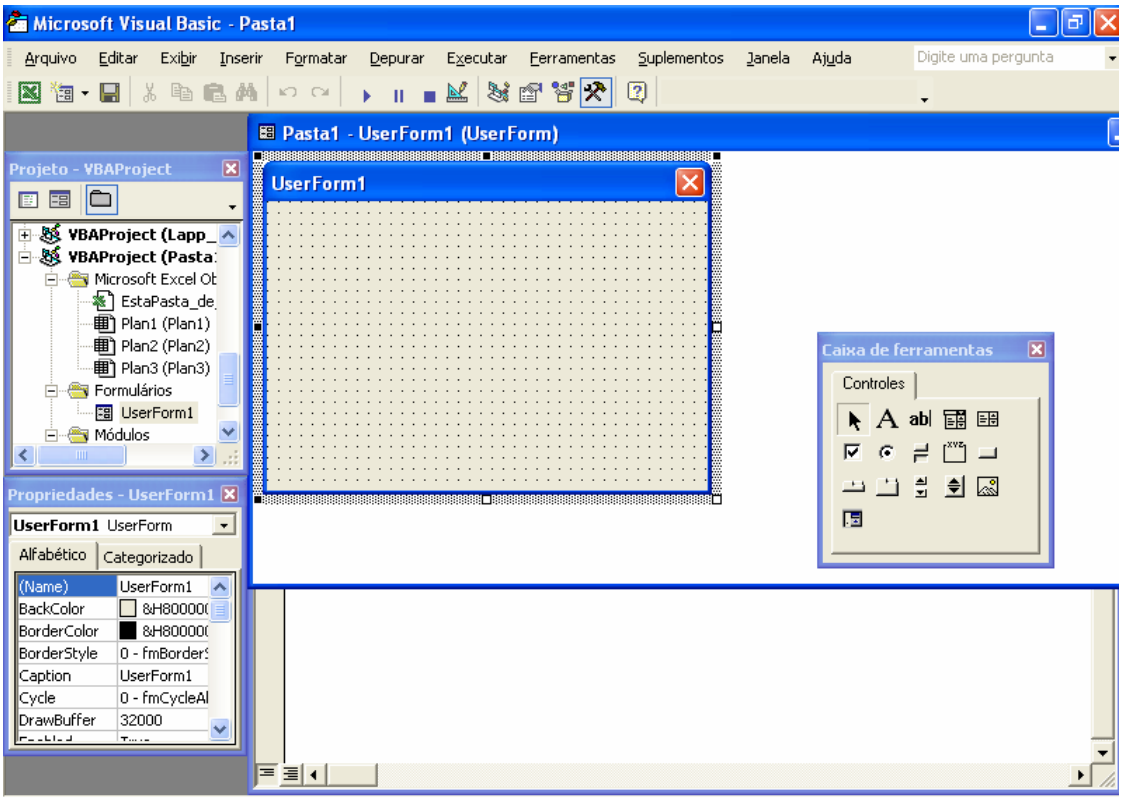
A seguir mostramos como criar um formulário e como programá-lo:

1. Pressione as Teclas Alt + F11, para entrar no editor de **Visual Basic**.
2. Ative as seguintes opções:

- Dê um clique no menu Exibir e escolha a opção Project Explorer
- Dê um clique no menu Exibir e escolha a opção Janela propriedades

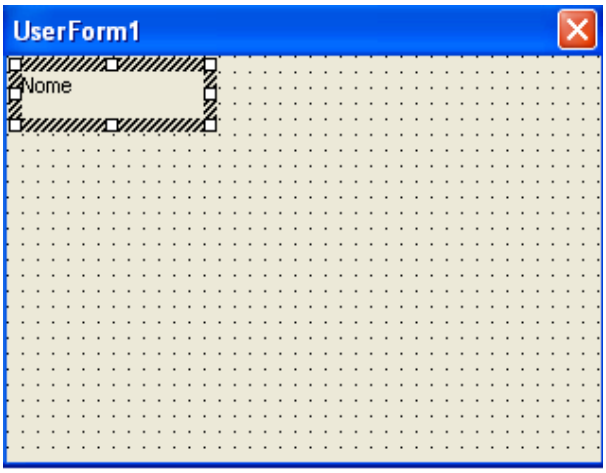


2. No menu Inserir escolha a opção UserForm**.** Isto insere o Formulário que programaremos com controles. No Project Explorer se observará que se inseriu o UserForm.

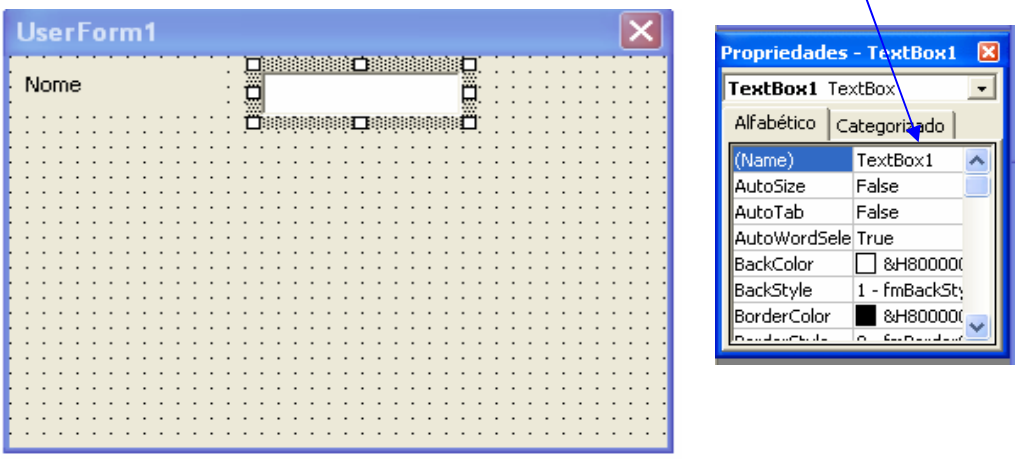


Também dê um clique no Formulário **USERFORM1** que deverá ativar a Caixa de Ferramentas**,** se ele não se ativar dê um clique no menu Exibir e escolha a opção Barras de ferramentas e clique na opção Caixa de ferramentas de controle**.**

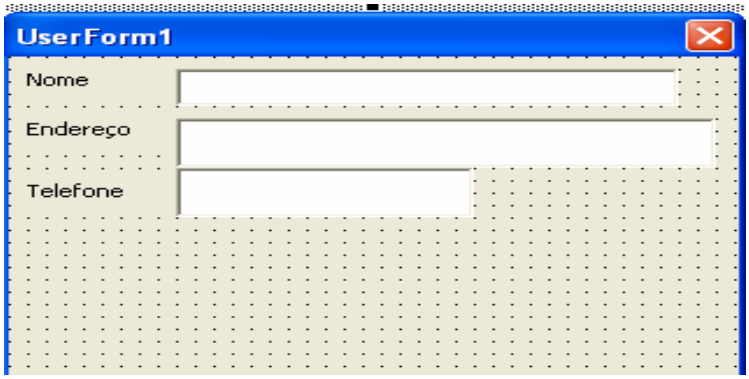
1. Escolha na Caixa de Ferramentas o Controle Rótulo, aquele tem um **A,** e arraste-o desenhando no Formulário **USERFORM1** o rótulo. Ele ganhará o nome Label1, depois dê um clique no rótulo desenhada e poderá modificar o nome de dentro e coloquemos aí **Nome.** Se por erro deres um duplo clique no rótulo, você será enviado à janela de programação do rótulo, daí então dê um duplo clique em **UserForm1** que se encontra no Project Explorer.



2. Escolha na Caixa de ferramentas o controle Caixa de texto, o que tem **ab** e arraste-o desenhando no formulário **USERFORM1** a caixa de texto ao lado do rótulo **Nome**. A caixa de texto deve de estar vazia e seu nome será **Textbox1**, o nome somente aparecerá na janela de propriedades do controle.



3. Faça os dois passos anteriores igualmente, colocando **Endereço** na **Label2** e **Telefone** na **Label3** e também redimensione a sua **Textbox**. Isto ficará assim depois de fazer isto



Se tiveres algum problema ao desenhar os rótulos ou as caixas de texto, somente troque o nome do **Rótulo** ou da **Caixa de texto** na Janela propriedades. A opção se chama **(Name).**

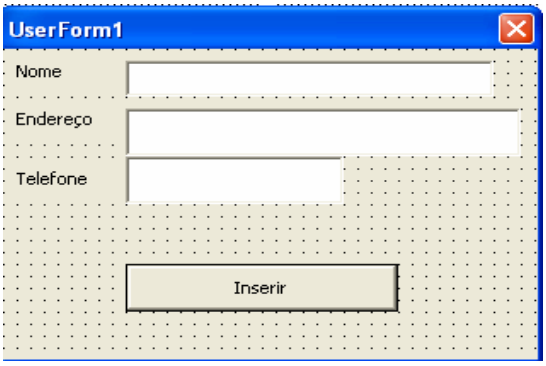
O Erro que marcar pode ser **Nome Ambíguo,** mas se se trocar o Nome do controle se resolverá o erro. Podes por qualquer nome no lugar de Label1.

Somente altere isto se marcar erro, se NÃO deixe assim.

Os controles como as Rótulos e Caixas de textos podem modificar algunas opções na Janela Propriedades. Para fazer isto é necessário ter conhecimento sobre as propriedades dos controles. Não altere as propriedades se não as conhece.

4. Escolha na Caixa de ferramentas o controle Botão de Comando e arraste-o desenhando no Formulário **USERFORM1** um botão. Será criado aí um botão de comando rotulado CommandButton1. Depois dê um clique no nome do botão desenhado e poderá modificar este nome. Coloquemos aí **Inserir.** Se por erro deres um duplo clique no botão, serás enviado à janela de programação do botão, para voltar, somente dê um duplo clique em **UserForm1** que se encontra no **Project Explorer.**

   Assim ficará o Formulário formado pelos controles:



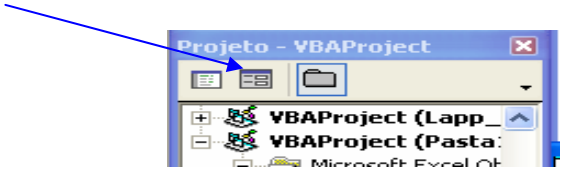
Agora dê um duplo clique sobre o controle **Textbox1** para programá-lo e depois inserir o seguinte código:

```
Private Sub TextBox1_Change()
Range("A9").Select
ActiveCell.FormulaR1C1 = TextBox1
End Sub
```

Isto indica que se vá à A9 e escreva o que há no **Textbox1**

Nota.-O que esta em azul (Claro e escuro) foi gerado automáticamente pelo Excel, você somente escreverá o que esta em Negrito.

Para voltar ao **Formulário** e programar o Textbox seguinte dê um duplo clique em **UserForm1** que se encontra no **Project Explorer,** ou simplemente dê um clique em Exibir Objeto no mesmo **Project Explorer**.



Agora dê um duplo clique sobre o controle **Textbox2** para programá-lo e depois inserir o seguinte código:

```
Private Sub TextBox2_Change()
Range("B9").Select
ActiveCell.FormulaR1C1 = TextBox2
End Sub
```

Isto indica que se vá à B9 e escreva o que há no **Textbox2**

Para voltar ao **Formulário** e programar o Textbox seguinte dê um duplo clique em **UserForm1** que se encontra no **Project Explorer,** ou simplemente dê um clique em Exibir Objeto no mesmo **Project Explorer**.

Agora dê um duplo clique sobre o controle Textbox3 para programá-lo e depois inserir o seguinte código:

```
Private Sub TextBox3_Change()
Range("C9").Select
ActiveCell.FormulaR1C1 = TextBox2
End Sub
```

Isto indica que se vá à C9 e escreva o que há no **Textbox3**

Para voltar ao **Formulário** e programar o Botão de Comando ***Inserir*** dê um duplo clique em **UserForm1** que se encontra no **Project Explorer,** ou simplemente dê um clique em Exibir Objeto no mesmo **Project Explorer**.

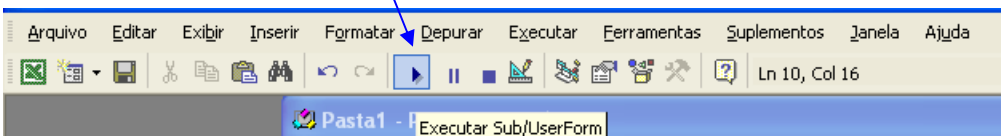
Agora dê um duplo clique sobre o controle **Botão de Comando** para programá-lo e depois inserir o seguinte código:

```
Private Sub CommandButton1_Click()
Rem inserir uma  linha
Selection.EntireRow.Insert
Rem o Empty Limpa os Textbox
TextBox1 = Empty
TextBox2 = Empty
```

```
TextBox3 = Empty
Rem A declaração Textbox1.SetFocus Envía o cursor ao Textbox1
para voltar a capturar os dados
TextBox1.SetFocus
End Sub
```

Nota.-O comando **Rem** é empregado para colocar comentários dentro da programação, o comando **Empty** é empregado para esvaziar as Textbox.

Agora pressione o botão Executar Sub/UserForm que se encontra na barra de ferramentas ou simplemente a tecla de função F5.



Ativar-se-á o **Userform1** e tudo o que se escrever nas Textbox se escreverá no Excel e quando pressionares o botão Inserir, se inserirá uma linha, esvaziarão as Textbox e depois se mostrará o cursor no **Textbox1**.

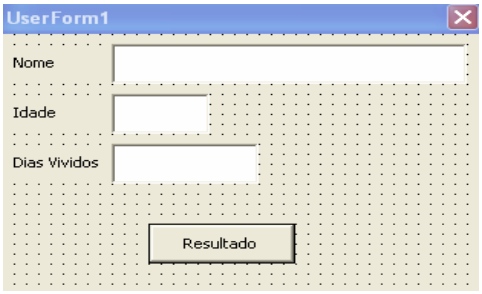
TRABALHANDO COM FÓRMULAS

É de suma importância saber aplicar **Fórmulas** em **Macros** do **Excel**, já que a maioria das folhas de cálculos as envolvem, por exemplo os *Inventários*, as *Normas* ou qualquer outro tipo de folha as levam, é por isso que nesta lição se mostra como manejar **Fórmulas** em **Macros de Excel**.

Pressione as Teclas Alt + F11**,** para entrar no editor de **Visual Basic**.

1. Ative as seguintes opções:
   - Dê um clique no menu Exibir e escolha a opção Project Explorer
   - Dê um clique no menu Exibir e escolha a opção Janela propriedades
1. No menu Inserir escolha a Opção **UserForm.** Isto insere o Formulário que programaremos com controles. No Project Explorer se observará que se inseriu o **UserForm**.

Agora criarás um formulário com o seguinte aspecto:



o formulário terá:

- Três rótulos
- Três Textbox
- Um Botão de Comando

Os dados que se perguntarão serão **Nome** e **Idade**, os **Dias Vividos** serão gerados automaticamente quando se inser a idade. A seguir se mostra como se devem de programar estes Controles:

Programação dos Controles:

```
Private Sub CommandButton1_Click()
Selection.EntireRow.Insert
TextBox1 = Empty
TextBox2 = Empty
TextBox3 = Empty
TextBox1.SetFocus
End Sub

Private Sub TextBox1_Change()
Range("A9").Select
ActiveCell.FormulaR1C1 = TextBox1
End Sub

Private Sub TextBox2_Change()
Range("B9").Select
ActiveCell.FormulaR1C1 = TextBox2
Rem aquí se criará a Formula
```

```
TextBox3 = Val(TextBox2) * 365
Rem O Textbox3 guardará o total da multiplicação do Textbox2 por
365
Rem O Comando Val permite convertir um valor de Texto a um Vaor
Numérico
Rem Isto se deve a que os Textbox não são Numéricos e devemos de
Convertê-los
End Sub

Private Sub TextBox3_Change()
Range("C9").Select
ActiveCell.FormulaR1C1 = TextBox3
End Sub
```

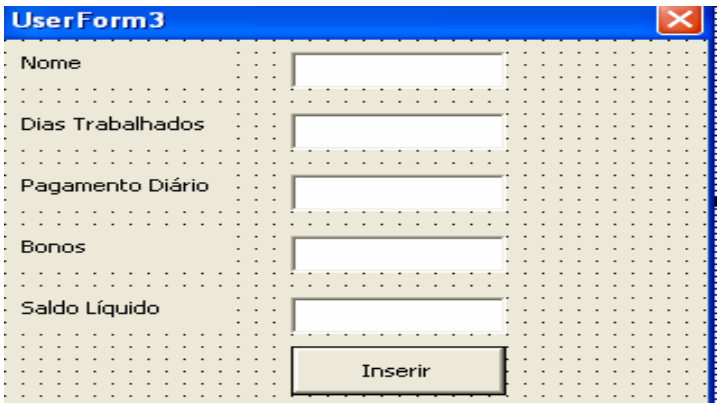
Isto vai permitir que quando se executar o formulário e se der a idade o resultado dos dias vividos aparecerá no **Textbox3** e se escreverá também em **Excel**. O comando **Val** é um comando de **Visual Basic** que te permite converter um valor de texto a um valor numérico. Lembrem-se que o Comando Rem se utiliza para colocar comentários únicamente e não afeta à programação.

Este Arquivo desta **Macro** se chama **Macros de Idade** e está incluido aquí.

Geraremos outro exemplo, crie o seguinte Formulário com os seguintes dados:

- 5 Rótulos
- 5 Textbox
- 1 Botão de Comando

Os dados que se perguntarão serão **Nome**, **Dias Trabalhados**, **Pagamento Diário**, **Bonos** e **Saldo Líquido**.



Gere o seguinte código:

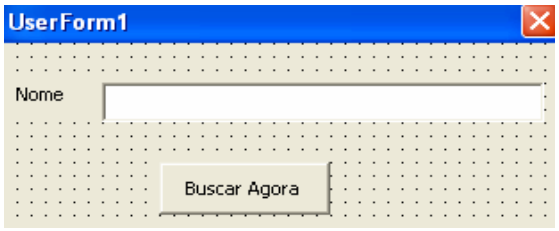
```
Private Sub CommandButton1_Click()
Selection.EntireRow.Insert
TextBox1 = Empty
TextBox2 = Empty
TextBox3 = Empty
TextBox1.SetFocus
End Sub
```

```
Private Sub TextBox1_Change()
Range("A9").Select
ActiveCell.FormulaR1C1 = TextBox1
End Sub

Private Sub TextBox2_Change()
Range("B9").Select
ActiveCell.FormulaR1C1 = TextBox2
End Sub

Private Sub TextBox3_Change()
Range("C9").Select
ActiveCell.FormulaR1C1 = TextBox3
End Sub

Private Sub TextBox4_Change()
Range("D9").Select
ActiveCell.FormulaR1C1 = TextBox4
Rem aqui se criará a formula
TextBox5 = Val(TextBox2) *Val(TextBox3) + Val(TextBox4)
Rem O TextBox5 guardará o total
End Sub

Private Sub TextBox5_Change()
Range("E9").Select
ActiveCell.FormulaR1C1 = TextBox5
End Sub
```

Quando se introduzir os bonos, automaticamente se gerará o Saldo Líquido.
Este exemplo vem no Arquivo **Macros** de **Saldo Líquido**

USANDO INFORMAÇÃO COM UMA TEXTBOX

Pode-se buscar informação com uma Textbox programando-a da seguinte forma:



Desenhe uma **Rótulo**, um **Textbox** e um **Botão de Comando** e agregue o seguinte Código:

```
 Private Sub TextBox1_Change()
Range("A9").Select
ActiveCell.FormulaR1C1 = TextBox1
End Sub
```

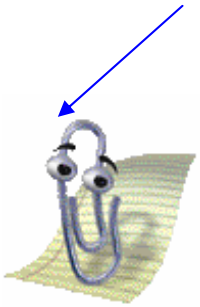
```
Private Sub CommandButton1_Click()
 Cells.Find(What:=TextBox1, After:=ActiveCell,
LookIn:=xlFormulas, LookAt _
:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext,
MatchCase:= _
False).Activate
End Sub
```

Observe que foi incluido na programação do botão **Buscar Agora** que buscará o que está no **Textbox1** na hora que se pressionar.

TRABALHANDO COM O ASSISTENTE

O assistente é o personagem do **Office** que se ativará e nos ajudará com suas janelas que podemos manipulá-las, por exemplo, pode-se dar animação, mover-se, fazer perguntas, etc.

Assistente do Office - chamado Clip



A seguir se mostram alguns códigos do Assistente:

Este código permite tornar visível o ajudante ou seja mostrá-lo. Se desejares ocultá-lo somente troque a opção True por False.

```
Assistant.Visible = True
```

Este código permite **Mover o Asistente** a um novo lugar, somente troque os valores numéricos e trocará de posição.

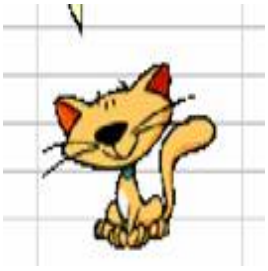
```
Assistant.Move 430, 230
```

Este código permite ativar um efeito de animação, quando escrever o sinal de **Igual** depois de Assistant.Animation =, aparecerá um menu com diferentes efeitos de animação

```
Assistant.Animation = msoAnimationListensToComputer
```

Este exemplo permite criar um **Novo Asistente** para poder manipular com uma pergunta e que tu a contestes. A variável **t** guardará o valor da resposta, se o valor é -3 significa que é Sim e portanto apagará a linha.

Outro Assistente do Office - chamado Mimi



```
Sub BalãoDoOffice()
Assistant.On = False
If MsgBox("Habilitar o Assistente do Office?", _
    vbYesNo, "Assistante está Desligado") = vbYes Then
    Assistant.On = True
    Assistant.Visible = True
    Assistant.Animation = _
        msoAnimationGetAttentionMajor
        End If
With Assistant.NewBalloon
.Text = "Desejas Apagar este Registro?"
.Button = msoButtonSetYesNo
.Heading = "Advertência"
t = .Show
```

```
End With
If t = -3 Then
Assistant.Animation = msoAnimationEmptyTrash
Selection.EntireRow.Delete
End If
End Sub
```

Outros **Assistentes** (Pingo, F1, Office Logo, Merlin, Natureza, Rex) encontram-se na **Galeria de Assitentes do Office**.

Lição 7: Código VBA do Excel para UserForms

Olá amigos, estamos de novo aquí para mostrar como se manipularão **Consultas nos Formulários,** acessos às **Macros** do **Excel** sem necesidade de entrar no **Visual Basic** e alguns métodos mais fácil de trabalhar.

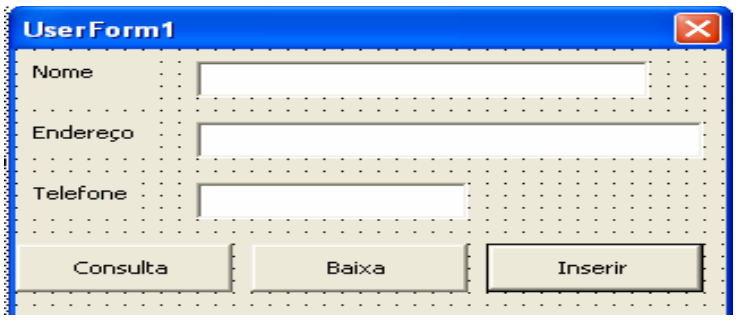
ELABORANDO UMA CONSULTA

Todo **Registro de informação** deve de ter sua própria **Consulta, Baixa e Modificação**, é por isso que neste novo capítulo nos concentraremos nele, primeiramente em poder **consultar a informação** que já se escreveu na **Folha** do **Excel**, obviamente desde uma **Macro** combinada com **Visual Basic**, observemos o seguinte exemplo:

1. Pressione as Teclas **Alt + F11,** para entrar no editor de **Visual Basic**.
2. Ative as seguintes opções:
   - Dê um clique no menu Exibir e escolha a opção Project Explorer
   - Dê um clique no menu Exibir e escolha a opção Janela propriedades

   No menu Inserir escolha opção **UserForm.** Isto insere o Formulário que programaremos com controles. No **Project Explorer** se observará que se inseriu o **UserForm**.

   Agora criarás um formulário com o seguinte aspecto:



o formulário terá:

- Três rótulos
- Três Textbox
- Três Botões de Comando

Os dados que se perguntarão serão **Nome, Endereço e Telefone**. Os três botões nos servirão para o seguinte:

Consultar - consultará a informação que tenhamos inserido com o botão inserir. **Baixa** - poderá eliminar algum dado que se consultou e não o queremos. **Inserir** - terá a fun-ção de inserir os registros, é como os exercícios anteriores. A seguir se mostra como se devem de programar estes Controles:

Programação dos Controles:

BOTÃO DE CONSULTA

```
Private Sub CommandButton1_Click()
Cells.Find(What:=TextBox1, After:=ActiveCell,
LookIn:=xlFormulas, LookAt _
:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext,
MatchCase:= _
False).Activate
ActiveCell.Offset(0, 1).Select
```

```
TextBox2 = ActiveCell
 Rem linha que contém o ActiveCell.Offset(0, 1).Select permite
mover-se uma coluna à direita, portanto depois da busca das
primeiras linhas com Cell.Find se encontra o Nome da pessoa se
move à seguinte coluna e a linha TextBox2 = ActiveCell Permite
capturar o valor da célula ao Textbox2 e asim mostrar o dado da
célula no TextBox2.
 ActiveCell.Offset(0, 1).Select
TextBox3 = ActiveCell
 Rem Cada vez que se escrever linha ActiveCell.Offset(0,
1).Select significa que se tem que mover-se uma coluna à
direita.
Rem Se o nome que tratas de consultar não se encontra poderia
gerar um erro porque falharía o Cell.Find isto pode ocorrer no
Word 97, eu trabalho com o Word 2007 e não tenho esse problema.
Mas isto se solucionaría com uma rotina de erro.
 End Sub
```

BOTÃO BAIXA

```
Private Sub CommandButton2_Click()
Selection.EntireRow.Delete
Range("A9").Select
TextBox1 = Empty
TextBox2 = Empty
TextBox3 = Empty
TextBox1.SetFocus
End Sub
```

BOTÃO INSERIR

```
 Private Sub CommandButton3_Click()
Range("A9").Select
Selection.EntireRow.Insert
TextBox1 = Empty
TextBox2 = Empty
TextBox3 = Empty
TextBox1.SetFocus
End Sub
```

CAIXAS DE TEXTO

```
 Private Sub TextBox1_Change()
Range("A9").FormulaR1C1 = TextBox1
 Rem esta primeira linha trocará estas duas…… que te parece
todavía mas curta
 Range("A9").Select
ActiveCell.FormulaR1C1 = TextBox1
 End Sub

 Private Sub TextBox2_Change()
Range("B9").FormulaR1C1 = TextBox2
End Sub

Private Sub TextBox3_Change()
Range("C9").FormulaR1C1 = TextBox3
```

```
End Sub
```

Se com o **Botão Consulta** tens um erro quando não encontra à pessoa, então terás de agregar isto a teu código do **Botão Consultar**

BOTÃO DE CONSULTA

```
Private Sub CommandButton1_Click()
On Error Goto aoencontro
Rem esta linha gera uma rotina de erro se Excel encontrar um
erro se o disser que se vá à rótulo aoencontro que está definida
mais adiante no código. Não use a janela de erro se não tiveres
problemas à hora que não encontrares a pessoa. Lembre-se se você
cometer qualquer erro o Excel se dirigirá ao rótulo a noencontro
e ignorará qualquer erro,até um que você cometer na programação.
 Cells.Find(What:=TextBox1, After:=ActiveCell,
LookIn:=xlFormulas, LookAt _
:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext,
MatchCase:= _
False).Activate
 ActiveCell.Offset(0, 1).Select
TextBox2 = ActiveCell
ActiveCell.Offset(0, 1).Select
TextBox3 = ActiveCell
 Rem Também se pode utilizar este código para ler a informação
das células ou que está em azul. A diferença é que se atribuam
os valores às variáveis e depois se descarregam os TextBoxs.
ActiveCell.Offset(0, 1).Select
Endereço = Activecell
ActiveCell.Offset(0, 1).Select
Telefone = Activecell
TextBox2 = endereço
TextBox3 = Telefone
aoencontro:
Rem Aquí se esquiva o erro
End Sub
```

O que lhe aparece é incrível como uma Macro combinada com Visual Basic pode fazer até o impossível

Bom já que temos elaborado um exercício de consultas de dados, agora acessaremos ao formulário do **Excel** sem necesidade de entrar no **Editor de Visual Basic**.

Para realizar este exercício devemos permanecer dentro do **Editor de Visual Basic** para poder introduzir o código em um **Módulo**, portanto deverás seguir os seguintes passos:

- Dê um clique no menu Inserir e escolha a opção Módulo
- Escreva dentro do Módulo o nome do módulo em este caso **Sub Entrada**

Quando você escrever Sub Entrada aparecerá da seguinte manera:

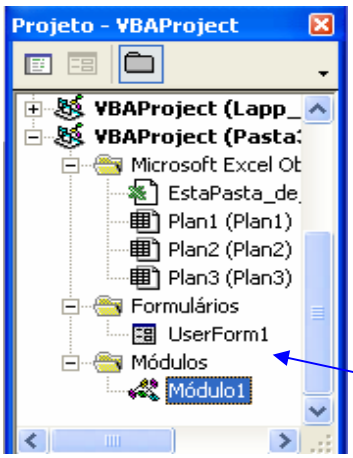
```
 Sub Entrada()
 Load UserForm1
UserForm1.Show
End Sub
```

Você deverá escrever as duas linhas que estão no meio que são:

```
Load UserForm1
UserForm1.Show
```

A primeira linha significa que carregue na memória o formulário que se chama **UserForm1**, a segunda linha significa que o mostre, isto quer dizer que no módulo estamos escrevendo o código de uma macro que permitirá carregar o formulário do Excel sem necessidade de entrar no **Editor de Visual Basic**.

Veja se no Project Explorer aparece o **Módulo** que criamos.



Se quisermos voltar ao formulário somente dê um duplo clique em **UserForm1**
Bom já está visto, agora salvamos do **Editor de Visual Basic e** voltemos ao **Excel.**

- Dê um clique no menu Arquivo do **Editor de Visual Basic**
- Escolha a opção Fechar e voltar ao Microsoft Excel

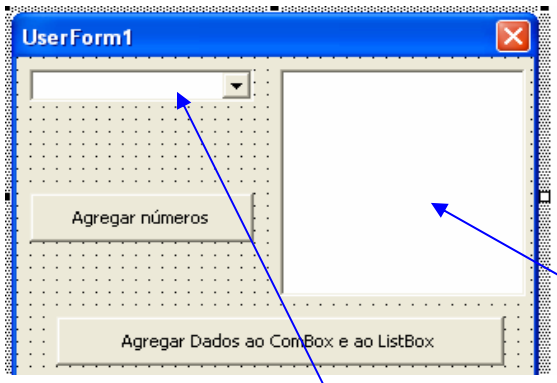
Agora que estamos no **Excel**, podemos inserir uma imagem, ou um botão, ou qualquer gráfico, por exemplo:

- Dê um clique no menu Inserir
- Escolha a opção Figura, seguido por Figura Do Arquivo
- Inserir qualquer Figura e dê a ela o tamanho que você desejares.
- Dê um clique no botão direito do mouse sobre a Figura
- Escolha a opção Atribuir macro
- Dê um clique na **Macro** que se chama **Entrada**, é fácil, foi a única que fizemos
- De um clique em OK
- Dê um clique fora da imagem em qualquer célula e dai se pressionar a imagen carregará o formulário.

COMO AGREGAR INFORMAÇÃO A UMA CAIXA DE COMBINAÇÃO E UMA CAIXA DE LISTAGEM

Bem, começaremos com como agregar informação a uma **Caixa de Combinação** (ComboBox**)** e a uma **Caixa de listagem** (ListBox**)**.

Primeiramente deverás criar o seguinte formulário dentro de **Visual Basic**.



Lembre-se de que do Excel se utiliza a tecla ALT + F11 para entrar no **Visual Basic**, seguido do menu Inserir e depois **Userform**, bem creio que já o sabes.

Insira uma **Caixa de Combinação** (Combox), uma **Caixa de Listagem (**Listbox) e um Botão.

Agora que já criastes a interface, vamos programar ao botão, e veremos como se lhe podem agregar informação por meio de códigos a estes dois novos controles.
Dê um duplo clique no **botão** e escreva as seguintes linhas dentro do procedimento.

```
Private Sub CommandButton1_Click()
ComboBox1.AddItem "João José"
ComboBox1.AddItem "Pedro da Fonte"
ComboBox1.AddItem "Salvador da Luz"
ListBox1.AddItem "João José"
ListBox1.AddItem "Pedro da Fonte"
ListBox1.AddItem "Salvador da Luz"
End Sub
```

Bem vamos analizar o significado destas linhas:

```
ComboBox1.AddItem "João José "
```

A opção **AddItem** significa que vais agregar um dado de texto, portanto se entende como vais agregar a João José ao **Combobox1**, logo eu posso agregar os dados que quiser a um **Combobox** ou a uma **Listbox** com a opção **AddItem**, então ao se pressionar o botão aparecerão os dados que se encontram escritos e poderás selecionar qualquer um deles, lembre-se que a informação vai se agregar segundo tuas necessidades.

Agora se desejares agregar números a um Combobox ou ListBox escreva o seguinte código em um botão:

```
Private Sub CommandButton1_Click()
For X=1 to 50
Listbox1.AddItem str(x)
Next
End Sub
```

A instrucão For-Next é um ciclo contador que te permite contar de um número até um outro. Por exemplo, digo-lhe que conte desde o 1 até 50 e o que se encontrar dentro do ciclo For-Next, o executará o que estiver dentro o número de vezes, a **X** é uma variável numérica em que se guarda o valor, cada vez que o ciclo dá uma volta aumenta um número, portanto **X** vai valer desde 1 até 50, e a instrução **Str** é para converter o

valor numérico da **X** em valor de texto, já que a opção **AddItem** guarda somente texto, claro esta que também pode funcionar sem esta instrução em alguns casos.

Portanto o **Listbox1** vai guardar os número do 1 ao 50, sem necesidade de ires pondo de um por um, imagine.

```
Listbox1.AddItem “1”
Listbox1.AddItem “2”
Listbox1.AddItem “3”
```

Já te quero ver no código para que chegues ao 50, hahahahahahaha.

Bem, isto é para se introduzir os dados a um **ListBox** e **ComboBox**, mas como posso usar estes dados para enviá-los para uma célula? No exemplo seguinte eu te explico:

Dê um duplo clique na **Listbox** e escreva o seguinte código:

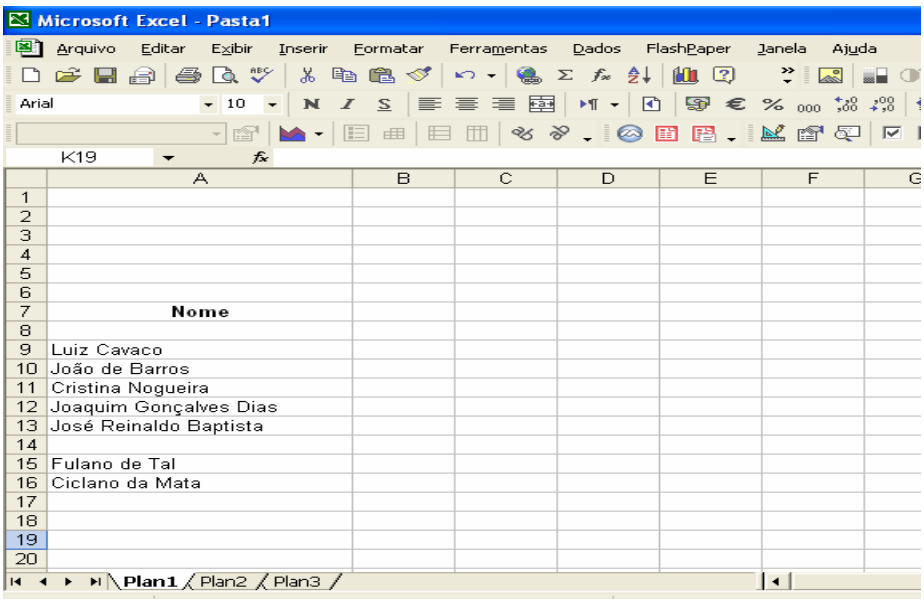
```
Private Sub ListBox1_Click()
Range("A9").Select
ActiveCell.FormulaR1C1 = ListBox1
End Sub
```

Assim é fácil, cada vez que escolher um dado que se encontre em um **Listbox1** ele o enviará à célula A9, escrevendo-o aí. Se desejares, podes fazer em um **Combobox**, somente trocando a **Listbox1** por **Combobox1** e se acabou.

Agora se desejares agregar os dados ao **Listbox** ou **Combobox** sem nenhum botão a pressionar, escreva o seguinte código:

```
Private Sub UserForm_Activate()
ComboBox1.AddItem "João Jose"
ComboBox1.AddItem "Pedro da Fonte"
ComboBox1.AddItem "Salvador da Luz"
ListBox1.AddItem "João José"
ListBox1.AddItem "Pedro da Fonte"
ListBox1.AddItem "Salvador da Luz
End Sub
```

A chave está no procedimento **UserForm_Activate(),** isto quer dizer que quando se ativar o formulário carregará o que tu o indicares, neste caso procures introduzir os dados ao **Listbox1** e **Combobox1** automaticamente, veja o que te aparece.



Agora se desejares tomar informação de uma célula e enviá-la a um **Combobox** ou **Listbox** escreva o seguinte código num Botão:

```
Private Sub CommandButton1_Click()
 Range("A9").Select
 Do While ActiveCell <> Empty
ActiveCell.Offset(1, 0).Select
ListBox1.AddItem ActiveCell
Loop
End Sub
```

Percebestes bem, primeiramente movo ao *range* (célula) A**9** porque aí está o início de minha informação, depois a linha Do While **Activecell<> Empty** significa Fazer enquanto as células não se encontrem vazias, a seguinte linha que é **ActiveCell.Offset(1, 0).Select,** significa abaixar uma Linha, a seguinte linha **ListBox1.AddItem ActiveCell,** agrega a informação desta nova célula ao **Listbox1** e a linha Loop é parte do ciclo Do While**,** ela sempre encera o ciclo, como o For-Next**.** Portanto todos os nomes que estiverem depois de A9 serão enviados ao **Listbox1** e quando o ciclo topar com a célula A14 que se encontra vazia, a condição do Do While parará a execução de seu código. Isto funciona caminhando linhas abaixo, mas se desejares mover à direita, por colunas, somente trocará a linha **ActiveCell.Offset(1, 0).Select,** por **ActiveCell.Offset(0, 1).Select,** quer dizer que se mova por coluna, não por linha.

ActiveCell.Offset(Range, Column).Select .

Se se troca o 1 por outro número se moverá o numero de vezes que tu indicar, por exemplo, se quero baixar 10 linhas num golpe:

```
ActiveCell.Offset(10, 0).Select
```

Se quiser mover-me 20 colunas à direita

```
ActiveCell.Offset(0, 20).Select
```

Assim funciona isto.

COMO SE EXECUTA UMA MACRO NA HORA DE ABRIR UMA PASTA

Agora veremos como se executa uma macro na hora de abrir uma pasta
Primeiramente insira um **Módulo** do menu Inserir dentro do **Visual Basic** e escreva o seguinte código:

```
Sub Auto_open()
Load UserForm1
UserForm1.Show
End Sub
```

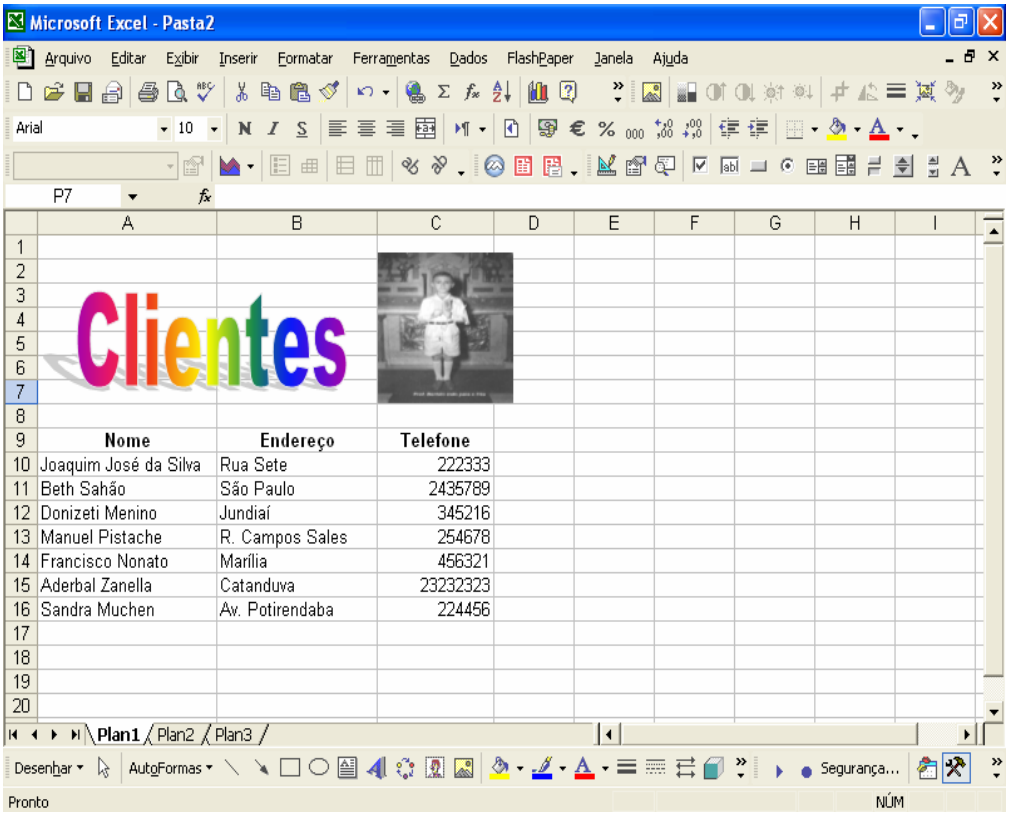
A magia está no procedimento **Auto_open()** que permite executar automaticamente o que se encontrar dentro, quando abrires uma pasta que contenha este código, neste exemplo quando se abre o pasta deve-se ativar o formulário 1 que programamos. Assim tudo o que agregares dentro deste procedimento se executará automaticamente quando abrires uma pasta, o que achas?.

A seguir veremos como ordenar uma informação por ordem alfabética ascendente, é um código muito completo e bom que te permite localizar os dados e ordená-los, sem passar uma linha em branco.

Observemos o seguinte exemplo e aprendamos dele:

Percebe-se na janela seguinte que tenho dados em uma folha que começa na linha A10 e termina na C16, o seguinte código detectará donde se deve parar para poder

ordenar os dados. É necessário criar o código para ordenar dados, mas aqui eu te mostro:



Programar isto num botão1

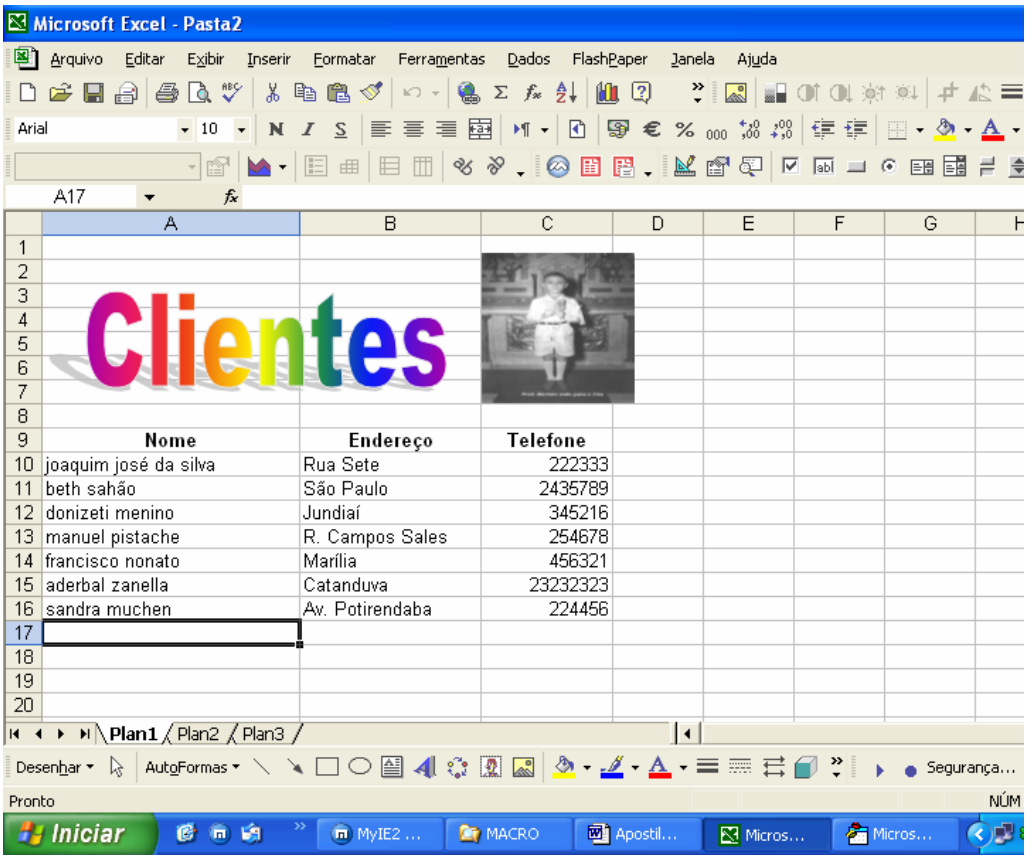
```
Private Sub CommandButton1_Click()
Rem este código localiza o último registro por meio da região
Range("A10").Select
Do While ActiveCell <> Empty
ActiveCell.Offset(1, 0).Select
Loop
Rem chega até o A17 donde não há informação e se volte uma
região para ser exato com a seguinte linha.
ActiveCell.Offset(-1, 0).Select
Rem este código localiza a última coluna do último dado
Do While ActiveCell <> Empty
ActiveCell.Offset(0, 1).Select
Loop
ActiveCell.Offset(0, -1).Select

Rem esta linha guarda a variável célulaativa na célula exata
donde está o último dado da última coluna de informação, neste
caso C16.

  célulaativa = ActiveCell.Address
```

```
Rem este código toma o range desde A10 donde começa a
informação, até onde encontrou o último dado C16, que o guardará
variável célulaativa. Seleciona de A10 até C16.

  Range("A10:" + célulaativa).Select

Rem este código ordena os dados norden ascendente, o código foi
gerado em Excel, assim que se não sabes gerá-lo apenas copie-o
daqui.

  Selection.Sort Key1:=Range("A10"), Order1:=xlAscending,
Header:=xlGuess, _

OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom

End Sub
```

Assim é como funciona este código de Macros do Excel ordenando exatamente desde A10 até onde estão os dados finais.

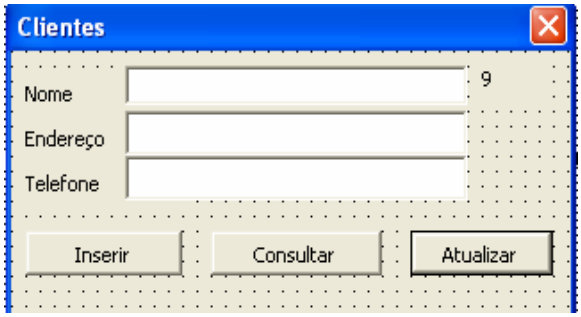
Bem, agora para converter a informação a Minúscula ou Maiúscula, o código é muito parecido, somente observá-lo:

```
Private Sub CommandButton2_Click()
Range("A10").Select
Do While ActiveCell <> Empty
ActiveCell.FormulaR1C1 = LCase(ActiveCell)
ActiveCell.Offset(1, 0).Select
Loop
End Sub
```



Assim é a magia de **Lcase** que converte à Minúsculas e **Ucase** à Maiúsculas, começa em A10 e até que não encontra dados deixa de converter a Minúsculas.

O seguinte Formulário e código mostrará força de como se pode consultar e modificar o dado que se encontrou.



Crie a seguinte Interface, 4 Rótulos, 3 Caixas de Textos (Textbox) e 3 Botões
No **Rótulo 4,** escreva o número 9 dentro dele.
Copie o seguinte código:

```
Private Sub CommandButton1_Click()
Rem não se escreva nada nos Textboxs na hora de inserir escreva
Não Tem
If TextBox1 = EmptyThenRange("A9").FormulaR1C1 = "Não tem"
```

```
If TextBox2 = EmptyThenRange("B9").FormulaR1C1 = "Não tem"
If TextBox3 = EmptyThenRange("C9").FormulaR1C1 = "Não tem"
Range("A9").Select
Selection.EntireRow.Insert
TextBox1 = Empty
TextBox2 = Empty
TextBox3 = Empty
TextBox1.SetFocus
End Sub
Private Sub CommandButton2_Click()
On Error GoTo noencontro
Rem Código para buscar, já o conhecemos
Cells.Find(What:=TextBox1, After:=ActiveCell,
LookIn:=xlFormulas, LookAt _
:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext,
MatchCase:= _
False).Activate
ActiveCell.Offset(0, 1).Select
TextBox2 = ActiveCell
ActiveCell.Offset(0, 1).Select
TextBox3 = ActiveCell
Rem o rótulo 4 toma o valor da região ativa e permite modificar
a informação que encontrou, já que modifiques a informação
pressionas o botão atualizar.
Label4 = ActiveCell.Row
noencontro:
End Sub

Private Sub CommandButton3_Click()
Rem Volte a indicar a região 9 para escrever nos Textboxs
  Label4 = "9"
Range("a9").Select
TextBox1 = Empty
TextBox2 = Empty
TextBox3 = Empty
TextBox1.SetFocus
End Sub

Private Sub TextBox1_Change()
Rem se nos damos conta o rótulo 4 serve para levar região donde
introduzimos os dados os modificamos, assim que cada textbox que
programemos deve levar estas linhas.
Range("A" + Label4).FormulaR1C1 = TextBox1
End Sub

Private Sub TextBox2_Change()
Range("B" + Label4).FormulaR1C1 = TextBox2
End Sub
Private Sub TextBox3_Change()
Range("C" + Label4).FormulaR1C1 = TextBox3
End Sub
```

MÓDULOS DE CÓDIGOS DE USERFORM

Quando você der um duplo clique num objeto sobre um *userform*, o VB editor atibuirá automaticamente uma macro para o objeto, e criará o que é chamado um módulo de código que contém as macros atribuidas para aqueles objetos do *userform*. Este módulo de código não está mostrado na janela do Explorer. Você pode acessar o módulo de código por um duplo clique num objeto sobre o *userform*, dê um clique com o botão direito do mouse sobre o *userform* ou em um objeto sobre ele e selecione ver código, ou dê um clique com o botão direito do mouse sobre o *userform* mostrado na janela de projeto e selecione ver código. Para retornar ao *userform*, dê um clique com o botão direito do mouse sobre a janela de código e selecione ocultar. Você pode também alternar para trás e para frente selecionando o menu Exibir e selecionando ou o Código ou o Objeto.

Por favor note que embora você possa criar variáveis públicas dentro um módulo de código de *userform*, estas variáveis públicas não estão disponíveis para os outros módulos. Para uma variável pública ficar disponível a outras macros e módulos, elas devem ser criadas num módulo normal.

PONTOS PRINCIPAIS SOBRE USERFORMS E BOTÕES

A seção anterior ilustra os seguintes pontos principais:

- Para ver previamente um *form*, vá à folha de *userform* e pressione a tecla F5
- O método Show quando usado com um *userform* não retorna True ou False como faz com uma folha de diálogo. Se você tiver tentado a seguinte declaração,

```
bResponse = userform1.Show
```

A declaração terá sido destacada como contendo um erro de sintaxe desde que um *userform* não retorna com um valor.

- As macros atribuidas a um botão são armazenadas em separado daquelas macros que você criou para os seus módulos.
- Você não pode atribuir macros a botões; por outro lado o VB editor faz isto para você, e dita o nome que será usado. Se você mudar o nome do botão, então a macro atribuida anteriormente não é mais atribuida ao botão.
- Você poderá minimizar o código nas macros atribuidas aos botões e objetos. O que você deverá fazer é usar uma variável global, tal como **bResponse** na seção acima, para armazenar a resposta do usuário. Daí então, quando o controle retornar da chamada da macro, aquela macro determina que ação deverá ser tomada.

- Um *userform* é mostrado usando o método Show. Entretanto, ela é liberada usando o comando Unload. Note que Show está presa ao nome do *userform* com um ponto, e que Unload é uma palavra solitária, com o nome *userform* seguindo-a (o que segue assume o *userform* ser chamado de "userform1":

```
userform1.Show
'declarações que obtém informações do userform
Unload userform1
```

- Se você fosse usar a declaração userform1.Hide, isto ocultaria o diálogo, mas não remova-o (*unload*) da memória. Se sua macro é posta em ação como um resultado de seleções sobre o *userform*, então você poderá usar Hide ao invés de Unload. Isto é discutido mais tarde neste capítulo.

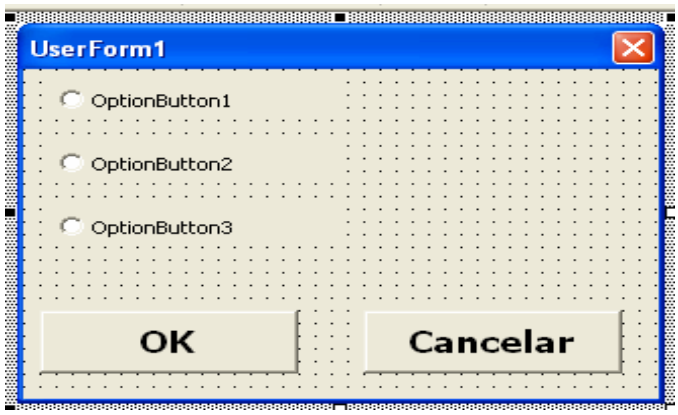
USANDO FRAMES NUM USERFORM

Um *frame* sobre um *userform* é usado para agrupar botões de opções, de modo que somente um botão num grupo possa ser selecionado. E também é usado para fornecer alguma ordem ao *userform*, enquadrando objetos relacionados no *userform*.

Se você colocar um frame num *userform* após você ter colocado outros objetos, o frame fica na frente, e ele oculta os outros objetos. O modo mais fácil para ocultar os objetos visíveis é arrastá-los para fora da área do **frame**, e então voltar completamente sobre o frame. Se somente parte do item está no frame, então somente a não coberta é visível. Mudar o **frame** de lugar é freqüentemente exigido para se ver os objetos ocultos.

USANDO BOTÕES DE OPÇÕES NUM USERFORM

Para ilustrar os botões de opção num *userform*, expanda o *userform* criado anteriormente para ter três botões de opções:



Os Botões de Opção podem ter um valor True ou um valor False. Sabendo isto, o que segue é a versão modificada da macro Show_Meu_UserForm que ilustra o uso dos botões de opções:

```
Sub Show_Meu_User_Form()
'defina os valores iniciais do botão de opção
UserForm1.OptionButton1.Value = True
UserForm1.Show
If Not bResponse Then End
'determina qual botão foi selecionado
With UserForm1
Select Case True
Case .OptionButton1.Value:
MsgBox "Botão 1 selecionado "
Case .OptionButton2.Value:
MsgBox "Botão 2 selecionado "
```

```
Case .OptionButton3.Value:
MsgBox "Botão 3 selecionado"
End Select
End With
Unload UserForm1
End Sub
```

Pontos chaves no uso dos botões de opção:

- A macro sabe quais botões de opções estão sendo questionados pois eles são qualificados com o objeto chamado UserForm1. A declaração With...End With é usada para minimizar a digitação e tornar o código mais eficiente vs. qualidade de cada botão de opção com "UserForm1".

O valor inicial de um dos botões de opção é configurado pela macro. Isto é uma boa técnica de programação.

- Numa aplicação real, você pode mudar o texto do botão de opção, sua fonte, seu nome, e quaisquer de suas outras propriedades.

USANDO CAIXAS DE LISTAGEM (LISTBOXES) NOS USERFORMS

Uma **caixa de listagem** (*listbox*)  é uma grande maneira de mostrar uma lista de seleção a um usuário que terá de selecionar um ou mais itens da lista. Se a um usuário é permitido pegar somente um dos itens de uma *listbox*, então você pode obter o texto e a posição na *listbox* do item selecionado (menos um quando o primeiro item for **0**, o segundo **1**, etc.) usando o código como o que segue:

```
Dim listText As String
Dim listNumber As Integer
If UserForm1.ListBox1.ListIndex = 1 Then
MsgBox "Nenhuma seleção foi feita"
Else
listText = UserForm1.ListBox1.Value & "selecionado"
listNumber = UserForm1.ListBox1.ListIndex
MsgBox listNumber & " " & listText
End If
```

Se a um usuário for permitido pegar múltiplas seleções numa lista (configure-as mudando a propriedade MultiSelect da **listbox**), daí o que segue é como você retornaria os itens selecionados e suas posições na caixa de listagem:

```
Dim listText As String
Dim listNumber As Integer
Dim I As Integer
UserForm1.Show
With UserForm1.ListBox1
For I = 0 To ListCount1
If .Selected(I) Then
listText = .List(I)
listNumber = I
MsgBox listNumber & " " & listText
```

```
End If
Next
End With
Unload UserForm1
```

Existem várias maneiras para especificar a lista-fonte a uma **listbox**:

- Selecione a caixa de listagem no Visual Basic editor e mostre a janela de propriedades. Daí então configure a propriedade RowSource para referir a um range na pasta contendo o *userform*. Se você estivesse especificando a linha fonte numa pasta diferente, então você teria de estabelecer um *link* para aquela pasta, o que pode ser bastante irritante. Também é muito difícil especificar um range manualmente. Você precisa digitar nas declarações como "'Plan1'!A1:A12".
- Especificar um range numa planilha como lista usando a declaração no seu código.

Por exemplo:

```
UserForm1.ListBox1.RowSource = "'Plan1'!A1:A12"
```

Ou

```
UserForm1.ListBox1.RowSource = Selection.Address(External:=
True)
```

A segunda abordagem é bem mais fácil e engloba que os nomes da planilha e da pasta estejam incluidos na string endereço passada a RowSource. Também, você pode usar qualquer *range*, ele não tem de ser selecionado.

- Se você quiser fornecer a RowSource manualmente na janela de propriedades, o modo mais fácil é usar um nome do *range* para especificar a lista. Este modo, você pode mudar o *range* e não precisa modificar o RowSource. Por favor note que você não pode deletar um item de uma lista se ela está configurada por uma referência a um *range*. Você precisará mudar os conteúdos do *range* e então especificar a propriedade RowSource se ela está no estilo A1.
- Se você especificar a propriedade RowSource e configurar a propriedade ColumnHeads para True, a linha imediatamente acima da RowSouce torna-se uma linha de título na caixa de listagem.
- Você pode especificar uma propriedade RowSource que é a largura das colunas múltiplas. Se você fizer isto, você também precisará especificar o número de colunas na propriedade ColumnCount. E, você pode mudar o tamanho das colunas que são mostradas digitando nelas números seguidos por uma virgula na propriedade ColumnWidths. A propriedade BoundColumns especifica o valor coluna que é retornado para uma seleção de lista. Por exemplo, você pode mostrar apenas uma coluna, mas ter outros valores de colunas retornados pela seleção da lista. Para mais ajuda em como estas propriedades funcionam, selecione qualquer uma destas propriedades e pressione a tecla F1.
- Use o método Additem na sua macro:

```
With UserForm1.ListBox1
.AddItem ("Jan")
```

```
.AddItem ("Fev")
.AddItem ("Mar")
End With
```

- Crie um array, e daí especifique que o *array* é para a lista:

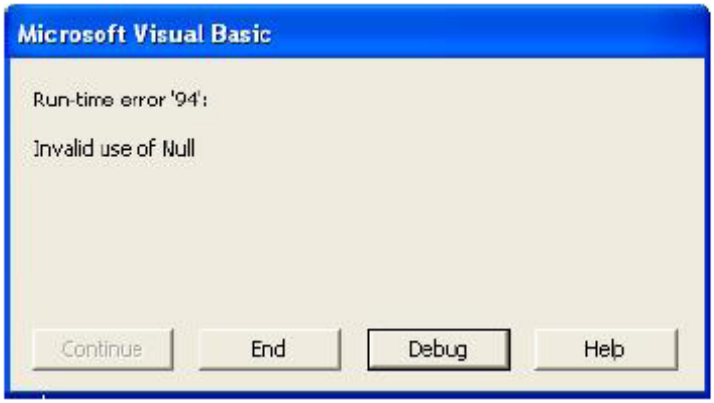
```
Dim MeuArray(1 To 3)
MeuArray(1) = "Jan"
MeuArray(2) = "Fev"
MeuArray(3) = "Mar"
UserForm1.ListBox1.List() = MeuArray
```

Se você usar a abordagem *array*, este pode ser um *array* uni ou bidimensional. Se for um *array* unidimensional, o resultado é uma simples lista. Se for um *array* bidimensional, o primeiro elemento do *array* determina o número de linhas. O segundo elemento do *array* determina o número de colunas. Por favor note que ele está numa aparência somente; você pode somente selecionar linhas inteiras na lista.

Você pode se quiser especificar manualmente o *range* lista quando criar uma caixa de listagem. Você deverá fazer isto primeiro selecionando a caixa de listagem e e daí mostrando a janela de propriedades. Selecione a propriedade RowSource e digite no endereço do *range*. O endereço poderá identificar folha e a pasta. Entretanto, se você mais tarde mudar o nome da pasta ou da planilha, a entrada manual não é atualizada, e a lista não será atualizada.

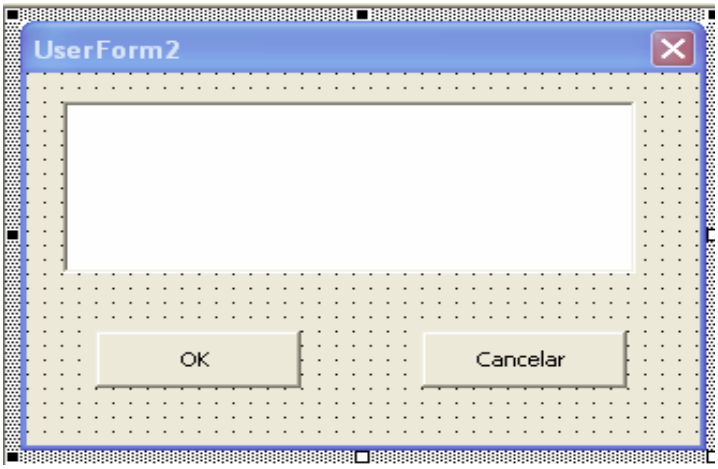
Por favor note que você poderia limpar a caixa de listagem antes de você preenchê-la. Por outro lado, a lista anterior pode ser retida e a nova lista adicionada à antiga. O método Clear limpa um *userform*. Por exemplo, UserForm1.ListBox1.Clear

Para determinar o número de items in a caixa de listagem, use a propriedade ListCount. Para determinar o número do item selecionado numa caixa de listagem, use a propriedade ListIndex. E para determinar o texto do que foi selecionado, use a propriedade Value. ENTRETANTO, você deve obter os valores do diálogo antes de você removê-lo da memória. Se você não fizer, você obterá a seguinte mensagem de erro quando você tentar obter um valor do *userform*:



Portanto, você poderá primeiro liberar o diálogo com um botão que usa a propriedade Hide. Daí então, depois de você ter obtido os valores do diálogo, use a propriedade

Unload para remover o diálogo da memória. O que segue ilustra isto, usando o seguinte *userform*:



Neste diálogo as macros seguintes são atribuidas aos botões OK e Cancelar. Note que o botão OK agora usa o método Hide, não o método Unload.

```
Private Sub Botão_Cancelar_Click()
bResponse = False
Unload UserForm1
End Sub

Private Sub Botão_OK_Click()
bResponse = True
UserForm1.Hide
End Sub
```

O que segue é a macro que **preenche** a caixa de listagem e daí mostra os valores selecionados pelo usuário. Por favor note que o valor ListIndex é **1** se nada for selecionado, **0** se o primeiro item selecionado e o número de items menos um se o último item for selecionado

```
Public bResponse As Boolean
'a declarção acima deve estar no topo do módulo
Sub User_Form_Demo1()
'limpar e preencher a caixa de listagem
UserForm1.ListBox1.Clear
With UserForm1.ListBox1
.AddItem ("Jan")
.AddItem ("Fev")
.AddItem ("Mar")
.AddItem ("Abr")
.AddItem ("Mai")
.AddItem ("Jun")
.AddItem ("Jul")
.AddItem ("Ago")
```

```
.AddItem ("Set")
.AddItem ("Out")
.AddItem ("Nov")
.AddItem ("Dez")
'mostrar o userform
UserForm1.Show
'determine se OK foi selecionado
If Not bResponse Then
MsgBox "cancelar selecionado"
Exit Sub
End If
'confirmar que aquela seleção foi feita
If .ListIndex = 1 Then
MsgBox "Nenhuma seleção feita"
Exit Sub
End If
'mostrar o número e o texto da seleção
MsgBox "Número do item selecionado na lista: " & _
.ListIndex + 1
MsgBox "texto do item selecionado: " & .Value
End With
Unload UserForm1
End Sub
```

Se a propriedade MultiSelect da caixa de lista é configurada para ou Multi ou Extended, então múltiplos itens podem ser selecionados na lista. Entretanto, quando isto é feito, então se deve usar a propriedade Selected (número) para ver se um item da lista foi selecionado. E, dai a Lista (número) para retornar a descrição do item. O que segue ilustra isto.

```
Public bResponse As Boolean
'Se este exemplo for copiado para a mesma pasta, declare
'a variável bResponse somente uma vez, no topo do módulo
Sub User_Form_Demo2()
Dim I As Integer
Dim bSelectionMade As Boolean
'limpe e preencha a caixa de listagem
UserForm1.ListBox1.Clear
With UserForm1.ListBox1
.AddItem ("Jan")
.AddItem ("Fev")
.AddItem ("Mar")
.AddItem ("Abr")
.AddItem ("Mai")
.AddItem ("Jun")
.AddItem ("Jul")
.AddItem ("Ago")
.AddItem ("Set")
.AddItem ("Out")
.AddItem ("Nov")
.AddItem ("Dez")
```

```
'mostrar o userform
UserForm1.Show
'determina se OK foi selecionado
If Not bResponse Then
MsgBox "cancelar selecionado"
Exit Sub
End If
'retorna o número total de itens na caixa de listagem
MsgBox .ListCount
'mostra o item selecionado
For I = 0 To .ListCount 1
If .Selected(I) Then MsgBox .List(I)
bSelectionMade = True
Next
End With
'remove o userform da memória
Unload UserForm1
if Not bSelectionMade Then
MsgBox "Nenhuma seleção foi feita"
Exit Sub
End If
End Sub
```

ADICIONANDO RÓTULOS (TEXTO) NUM USER FORM

Para adicionar texto a um *userform* é muito simples: apenas clique no botão Rótulo (*Label*) (o botão "A") na barra de ferramentas Caixa de ferramentas, desenhe uma caixa do tamanho que você precisar, e digite o texto. Se você tiver a caixa de rótulo selecionada, você pode mostrar a caixa de propriedades e mudar as propriedades. Por exemplo, você pode mudar o tamanho e cor da fonte, a cor de background e muitas outras propriedades. A melhor maneira para aprender mais acerca de adicionar rótulos é tentar as diferentes propriedades no *userform* teste. Rótulos nos *userforms* são tipicamente entradas de texto que dão direções ao usuário. Você pode ter o texto empacotado na caixa de rótulo, e você pode especificar a fonte a ser usada no rótulo.

O que segue são algumas das principais propriedades de uma caixa de rótulo:

Caption - O texto que é entrado na caixa. Este pode ser configurado quando você criar manualmente a caixa de edição, ou pode ser mudado interativamente:

```
UserForm1.Label1.Caption = "123"
```

Se a mudança for feita por uma macro, o título (*caption*) ou texto que aparece no *userform* no VB editor não muda. Entretanto, o texto do rótulo que está mostrado é aquele especificado pela macro.

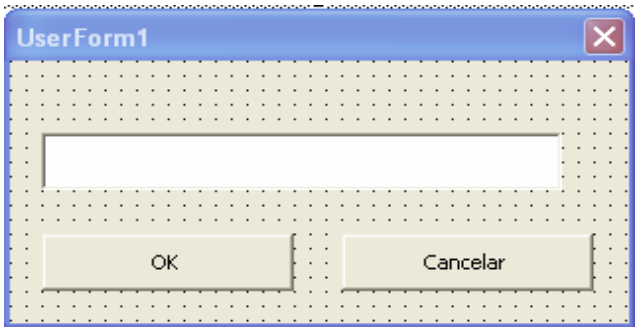
WordWrap - Permite empacotar palavras na caixa de texto

Visible - Permite o rótulo ocultar se configurado para False, mostrado se configurado para True.

USANDO CAIXAS DE TEXTO (TEXT BOXES) NOS USERFORMS

Uma **caixa de texto** (text box) é um modo rápido e fácil de se obter as entradas do usuário. O que segue ilustra uma caixa de texto onde o usuário deve entrar com um número válido, que neste caso é definido como um número entre um e dez. Se o usuário não entrar com um número válido, então uma mensagem dizendo que a entrada não foi válida é mostrada e daí então o diálogo é reapresentado.

O *userform* se parece com isto. A caixa de texto (textbox) foi adicionada usando o botão caixa de texto da barra de ferramentas Caixa de ferramentas (ele é o botão que se parece com "ab").



Os dois botões foram chamados de Botão_OK e Botão_Cancelar, e a eles foram atribuídos as seguintes macros:

```
Private Sub Botão_Cancelar_Click()
Unload UserForm1
End
End Sub

Private Sub Botão_OK_Click()
UserForm1.Hide
End Sub
```

O que segue é a macro principal que mostra o diálogo *userform* e valida a entrada.

```
Sub Obter_Entrada_Usuario()
Dim entradaUsuario As Long
'Limpar a caixa de texto antes de mostrar pela primeira vez
UserForm1.TextBox1.Value = ""
Do
UserForm1.Show
entradaUsuario = Val(UserForm1.TextBox1.Value)
'uma entrada em branco torna-se um valor zero
If entradaUsuario > 0 And entradaUsuario <= 10 _
Then Exit Do
MsgBox "O valor que você entrou não estava " & _
"entre 0 e 10."
Loop
Unload UserForm1
MsgBox entradaUsuario & " foi entrado."
```

```
End Sub
```

Como quase todos os objetos *userform* que se pode usar, existe um tremendo número de propriedades para uma caixa de texto. Par mostrar as propriedades, clique no botão **janela de propriedades**  da barra de ferramentas Padrão do Microsoft Visual Basic, ou selecione Exibir, Janela 'Propriedades', ou pressione F4. Para ver a informação de uma dada propriedade, clique na propriedade e dai pressione o botão F1. As várias propriedades principais que são úteis são:

Value A entrada é feita na caixa
Multiline Permite linhas múltiplas de dados serem entradas.
Scrollbars Mostra barras de rolamento se são feitas entradas longas.
PassWordChar Especifica o uso de placeholders em vez de entradas reais quando um *password* está sendo entrado.
Visible Mostra ou oculta a caixa. Útil se você somente quiser editar a caixa para aparecer se outras condições são encontradas.
Font Pemite-lhe especificar o tamanho e a fonte a serem usadas.

A maioria das propriedades de uma caixa de edição de texto é destinada a serem configuradas quando você está desenvolvendo a caixa de edição de texto e permanecem com aquelas configurações. Algumas, tais como as propriedades Value e Visible são destinadas a serem configuradas interativamente pelo seu código.

COMBO EDIT NO USERFORM, CAIXAS DE LISTAGENS DROP DOWN

O que segue ilustra o preenchimento de uma caixa de combinação  , lista *dropdown* com os meses do ano e daí então determina o que foi selecionado:

```
Sub combo_box_demo()
'limpa os conteúdos da combobox
UserForm2.ComboBox1.Clear
'preenche a caixa de um range de uma planilha
UserForm2.ComboBox1.RowSource = _
ThisWorkbook.Sheets("plan1") _
.Range("a1:a12").Address(external:=True)
'mostra o userform
UserForm2.Show
'mostra a entrada de usuário ou a seleção
With UserForm2.ComboBox1
MsgBox "item número " & .ListIndex & " selecionado." & _
" Ele é " & .Text
End With
End Sub
```

Se o valor da ListIndex é 1, então isto indica que o usuário fez uma entrada na caixa de texto ao contrário de pegar um valor da lista. Também, a ListIndex começa no zero para o primeiro item na lista, não um.

A lista numa caixa de combinação *drop down* é exatamente como uma caixa de listagem regular. Isto significa que você pode usar *arrays* para atribuir valores à lista, e o *array* pode ser bidimensional.

BOTÕES DE ALTERNÂNCIA E CAIXAS DE VERIFICAÇÃO DE USERFORM

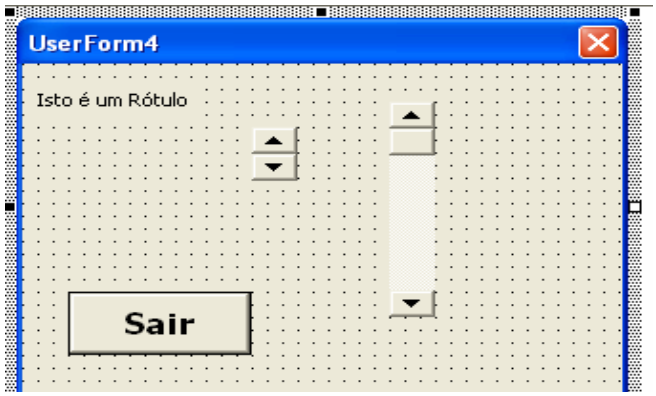
Quando um botão de alternância for clicado, ele alterna de selecionado (aparência pressionado) para não selecionado (aparência de botão normal). Ele, como uma CheckBox, tem um valor ou True ou False. Para configurar o valor inicial destes objetos numa macro, especifique o *userform*, o nome do controle, e o valor a ser usado. Por exemplo:

```
UserForm1.ToggleButton1.Value = True
UserForm1.CheckBox1.Value = False
```

Para determinar o valor do botão alternância, peça seu valor antes de descarregar o *userform*.

BARRAS DE ROLAMENTO (*SCROLLBARS*) E BOTÃO DE ROTAÇÃO (*SPINNERS*) NO USERFORMS

O melhor modo de mostrar como as Barras de rolamento (*Scrollbars*) e Botão de rotação (*Spinners*) funcionam é por meio de um exemplo. O *userform* seguinte contém um rótulo, um botão de rotação, uma barra de rolamento, e um botão de comando.



Neste exemplo, quando você usar ou o botão de rotação ou a barra de rolamento, o valor na caixa de texto (*textbox*) muda. Limites têm sido impostos de modo que o valor fique limitado entre 0 e 100. As propriedades seguintes da barra de rolamento foram configuradas manualmente selecionando a barra de rolamento e mostrando a janela de propriedades:

```
Propriedade     valor
LargeChange     10
Min             0
Max             100
```

Configurando estas propriedades manualmente reduz a necessidade de se fazer via código.

O que segue é o código que foi colocado num módulo regular:

```
'Declarar no topo do módulo. Usado pelo código userform
```

```
Public spinValue As Integer
Sub ScrollAndSpinnerDemo()
'Descarrega o form apenas no caso da macro crashed anteriormente
'e ela não fora descarregada
UnLoad Userform1
'Inicializa a variável pública, a barra de rolamento, e o rótulo
spinValue = 50
UserForm1.ScrollBar1.Value = spinValue
UserForm1.Label1.Caption = spinValue
'Mostra o form, e daí descarrega quando o controle retornar à
'macro
UserForm1.Show
Unload UserForm1
End Sub
```

O código seguinte foi colocado no módulo de código do *userform* (acessível por um duplo clique sobre o *userform*). Clicando no botão de comando apenas fecha o form. Clicando nas setas para cima ou para baixo do botão de rotação, aumenta-se ou diminui-se os valores da spinValue. E, então o rótulo é alterado e o valor da barra de rolamento alterado. Mudando a barra de rolamento, clicando ou na área de rolamento, sobre as setas para cima ou para baixo, ou arrastando a barra de posição muda o valor da spinValue e também altera o rótulo.

```
Private Sub CommandButton1_Click()
Me.Hide
End Sub

Private Sub ScrollBar1_Change()
'Trata de clicar na barra ou topo ou fundo
spinValue = ScrollBar1.Value
Me.Label1.Caption = spinValue
End Sub

Private Sub ScrollBar1_Scroll()
'Trata de mover a barra de rolamento por si só
spinValue = Me.ScrollBar1.Value
Me.Label1.Caption = spinValue
End Sub

Private Sub SpinButton1_SpinDown()
'Trata de clicar parte de baixo do botão de rotação
spinValue = spinValue - 1
If spinValue < 0 Then spinValue = 0
Me.ScrollBar1.Value = spinValue
Me.Label1.Caption = spinValue
End Sub

Private Sub SpinButton1_SpinUp()
'trata de clicar na parte superior do botão de rotação
spinValue = spinValue + 1
If spinValue > 100 Then spinValue = 100
```

```
Me.ScrollBar1.Value = spinValue
Me.Label1.Caption = spinValue
End Sub
```

USANDO UMA CAIXA REFEDIT NUM USERFORM

Uma caixa RefEdit é usada para permitir o usuário especificar um range. Entretanto, você não pode selecionar Janela e mudar de uma janela para outra o usuário deve usar a tecla CTL+Tab para se mover de uma pasta a outra. Para superar esta limitação, adicione uma caixa de listagem (listbox) no seu *userform* que permita o usuário ativar qualquer pasta. O que segue mostra como fazer isto:

```
Sub ExampleWBRangeSelection()
Dim wb As Workbook
For Each wb In Workbooks
UserForm1.ListBox1.AddItem wb.Name
Next
UserForm1.Show
Unload UserForm1
End Sub
```

No módulo código para o form, coloque o seguinte código, que limpa a caixa refedit quando uma pasta for selecionada.

```
Private Sub ListBox1_Change()
Workbooks(Me.ListBox1.Value).Activate
Me.RefEdit1.Text = ""
Me.RefEdit1.SetFocus
End Sub
```

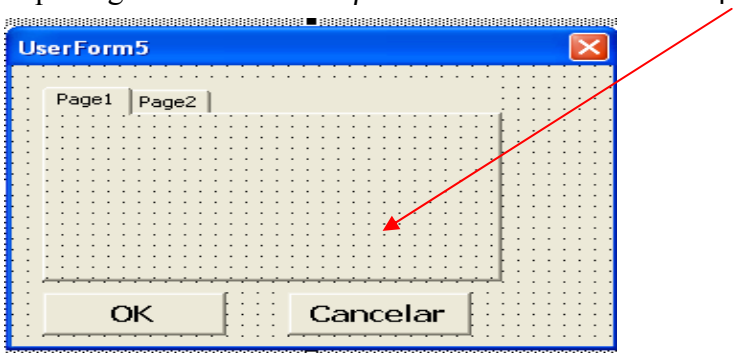
O código seguinte ilustra como configurar o valor de uma caixa RefEdit, e como perguntar seu valor. Por favor note que Application.ScreenUpdating deve ser configurado para True para o usuário selecionar um range.

```
Sub RefEdit_Demo()
'define a referência inicial na RefCaixa de edição
UserForm1.RefEdit1.Value = Selection.Address
'mostra o userform contendo a RefCaixa de edição
UserForm1.Show
'mostra a seleção do usuário
MsgBox UserForm1.RefEdit1.Text
'Define um objeto para se referir à entrada do userform:
dim anyR As Range
'o que segue assume que o usuário fez uma seleção
Set anyR = Range(UserForm1.RefEdit1.Text)
anyR.Select
'Remova o Userform somente após a informação ser armazenada
'pelo último usuário.
Unload UserForm1
End Sub
```

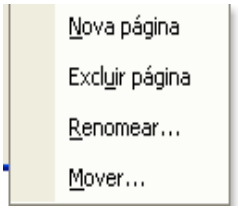
Por favor note que você precisará de um botão de comando no seu *userform* que oculte o *userform*.

CRIANDO UM USERFORM MULTI-PÁGINA

O que segue ilustra um *userform* com um controle Multi-página nele:



Você cria um diálogo Multi-página criando um *userform*, clicando num botão Multi-página na Caixa de Ferramentas, e daí desenhando-o no *userform*. Daí você pode adicionar objetos a cada uma das páginas. Você pode também ter mais do que duas páginas num *userform* Multi-página. O truque para adicionar uma nova página é primeiro clicar na página, e daí então dar um clique com o botão direito do mouse na guia da página. Isto faz com que o diálogo seguinte apareça:



Se você não clicar na guia, um *popup* diferente aparecerá. Use o item de menu Nova página no *popup* acima para **adicionar** uma nova página. Você pode também usar as opções deste *popup* para **deletar**, **renomear**, ou **mover uma página**. Finalmente, um objeto Multi-página não precisa incluir o diálogo todo. Você pode colocá-lo em apenas uma parte do diálogo se você quiser.

USANDO UMA GUIA STRIP NUM USERFORM

A guia nua está essencialmente na parte superior de um objeto multipáginas. Basicamente, você tem um conjunto de objetos, tais como caixas de listagens (ListBoxes) e botões de opção no seu *userform*. Quando você clicar sobre uma dada guia da guia strip, a macro atribuida a strip determina qual guia está ativa agora, e ativa as propriedades dos objetos do *userform*. Por exemplo, você pode ativar quatro guias e atribuir a cada uma delas uma região diferente de um país. Clicando numa guia altera a caixa de listagem (*listbox*) contendo os clientes daquela região.

Para adicionar guias adicionais à guia nua é ligeiramente manhoso. Você deve clicar uma vez na guia nua, esperar cerca de um segundo, e clicar novamente. Se bem sucedido, a banda de seleção ao redor da guia nua fica mais escura. Daí então, dê um

clique com o botão direito na guia nua, e um popup como aquele para um objeto Multi-página aparece.

O código seguinte ilustra uma macro que poderia ser atribuida à guia strip para fazer a ação de atualizar a informação sobre uma *userform* quando uma guia for selecionada:

```
Private Sub TabStrip1_Change()
UserForm2.Label1.Caption = _
"Guia " & UserForm2.TabStrip1.Value & " selecionado."
End Sub
```

O código acima muda o rótulo num *userform* para estabelecer o número da guia selecionada. Você pode também escrever o código no lugar de mudar qualquer objeto *userform* e seu conteúdo. Por favor note que a primeira guia é **0**, a segunda é **1**, e assim por diante. Também, em vez de combinar o código na macro atribuida à guia nua, você pode ter atribuido uma macro para chamar a macro do seu módulo. Isto torna mais fácil de manter este código, acessá-lo, e imprimí-lo. Você pode usar uma declaração IF ou uma declaração Select Case para fazer isso. Por exemplo:

```
Select Case UserForm2.TabStrip1.Value
Case 1: macro1
Case 2: macro2
Case 3: macro3
End Select
```

INSERINDO IMAGENS NUM USERFORM

Para colocar uma figura num *userform*, simplesmente clique no botão Caixa de Ferramentas aquele que parece uma cena de montanha  e desenhe uma caixa sobre o *userform*. Daí então, mostre janela de propriedades e usando a propriedade figura, selecione uma figura para o interior da caixa. Você pode mudar outras opções, tais como o tamanho da figura para ajustá-la na caixa. Você pode também atribuir uma macro à imagem por um duplo clique nela. Uma vantagem de se usar uma figura é que você pode colocar um *design* complexo na sua folha, tal como uma *trademark* ou *logo*.

CONFIGURANDO A GUIA ORDEM

Para configurar a guia ordem dos objetos num *userform*, simplesmente selecione o *userform* e daí então selecione Exibir, Tab Order. No diálogo que aparecer, você pode mover os controles para cima e para baixo na guia order. Se você não quiser mudar a guia order dos controles no *UserForm*, use o método SetFocus para especificar que objeto terá foco quando você mostrar o UserForm. Por exemplo, se você quiser uma caixa de listagem chamada ListBox1 como tendo o foco quando você mostrar o UserForm, use o seguinte modelo de código para o evento Initialize do *UserForm*:

```
Private Sub UserForm_Initialize()
ListBox1.SetFocus
End Sub
```

Este código é colocado na folha de código do *userform*, o qual você pode acessar selecionando o *userform* e selecionando Exibir, Código. Daí então selecione Initialize no *drop down* do lado direito com o *userform* selecionado no *dropdown* do lado esquerdo. Quando você mostrar o UserForm, o ponto de iserção aparece dentro da caixa de listagem, a despeito da guia order no *userform*

USANDO VARIÁVEIS PARA SE REFERIR AOS OBJETOS DO USERFORM

No seu código você pode se referir aos objetos sobre um *userform* pela combinação do Nome do *userform* e o nome do objeto. Isto reduz a necessidade de se criar variáveis de objeto para *userform* objetos. Entretanto, existirão instances onde variáveis de objeto são úteis. Qualquer das abordagens seguintes lhe permite declarar a variável e configurála igual a um objeto num *userform*:

```
Dim myObject
Dim myObject As Variant
Dim myObject As MSForms.objeto tipo
```

A última abordagem acima é melhor pois ela especifica o tipo de objeto. Por exemplo

```
Dim lBox As MSForms.ListBox
Set lBox = UserForm1.Listbox1
```

Como informação, se você não usar MSForms para qualificar o tipo de objeto, você obterá um erro de sintaxe quando você tentar executar sua macro. Se você estiver editando o código num módulo de *userform*, você pode usar a variável Me para se referir ao *userform*. Isto simplifica grandemente a escrita do código.

USANDO DUPLOS CLIQUES PARA FECHAR UM USERFORM

Dando um duplo clique num objeto *userform* tal como um botão opção ou um item numa lista não fecha o diálogo. O *default* é não fazer nada i. e., o *userform* permanece mostrado. Entretanto, você pode atribuir uma macro a cada objeto sobre um *userform* que você queira dando um duplo clique nele. Por exemplo, Se você quiser que o *userform* feche quando um usuário der um duplo clique num botão opção. Cada objeto deve ter a sua própria macro. Para criar macros que fecham quando num objeto é dado um clique duplo, você deve primeiro ir ao módulo que mantém o código para objetos num *userform*. O modo mais fácil de obter este módulo é dar um duplo clique em qualquer objeto no *userform*. Por exemplo, se você der um duplo clique num botão opção, aparecerá o código como o seguinte:

```
Private Sub ListBox1_Click()
End Sub
```

A menos que você já tenha criado o código do objeto que você deu o duplo clique nele, a macro estará vazia. O próximo passo é selecionar no *drop down* do lado esquerdo da folha de macro o objeto para o qual você quer atribuir uma macro duplo clique. No *drop down* do lado direito, selecione a propriedade para a qual você quer atribuir uma macro. Neste caso, a propriedade é a propriedade "DblClick". Quando você clicar nele, o código seguinte aparecerá:

```
Private Sub OptionButton1_DblClick _
(ByVal Cancel As MSForms.ReturnBoolean)
End Sub
```

Assumindo que seu *userform* seja chamado de "UserForm1", você deverá modificá-lo para incluir a declaração seguinte:

```
Private Sub OptionButton1_DblClick _
(ByVal Cancel As MSForms.ReturnBoolean)
UserForm1.Hide
End Sub
```

Este código fará o *userform* ficar oculto. Isto lhe permite determinar as seleções do usuário no *userform*. Por favor note que você não libera o *userform* no código acima. Se você liberar o *userform*, você não será capaz de determinar qualquer das seleções do usuário no *userform*.

CENTRALIZANDO USERFORMS NA TELA

A propriedade que centraliza um *userform* é a propriedade StartUpPosition. Se ela for configurada para 1CenterOwner ou 2CenterScreen, então você obterá resultados de centralizações. Se você tiver o Excel configurado como uma janela na sua própria tela do computador, então CenterOwner centralizará o diálogo na janela do Excel. O CenterScreen centralizará o diálogo no meio da tela, não importa para qual tamanho você configurou para a janela do Excel. A configuração *default* é 1CenterOwner.

SOBRE AS FOLHAS DE DIÁLOGOS DO EXCEL 5/7

Excel 5/7 usa folhas de diálogos em vez de *userform*. As Folhas de diálogos são um pouco diferentes dos *userforms*. Poe exemplo,não há módulos de código ou código associados com objetos. Não há vantagem de se usar folhas de diálogos do Excel 5/7 versus *userforms*. E, em muitos casos, são mais difíceis de se usarem.

SUMÁRIO

Existem várias chaves de trabalhar com objetos num *userform*. Uma delas é reconhecer que cada objeto tem um conjunto de propriedades que você pode configurar, mudar e interrogar. O número de propriedades de objeto tem aumentado dramaticamente. O melhor modo de entender as propriedades de um objeto é mostrar a janela de propriedades do objeto e mudar as propriedades e observar o resultado. Um outro ponto para os objetos é que você não precisa dizer tudo ao VB, qual coleção um objeto pertence. Por outro lado, o VB sabe qual coleção um objeto pertence, assim simplifique seu trabalho. Finalmente, embora existam muitos objetos e propriedades, é importante manter seu uso de *userforms* simples e focado. Evite *userforms* complexos e manipulações complexas dos objetos sempre que possível.

Lição 8: Código VBA do Excel para Gráficos e Arquivos Seqüenciais

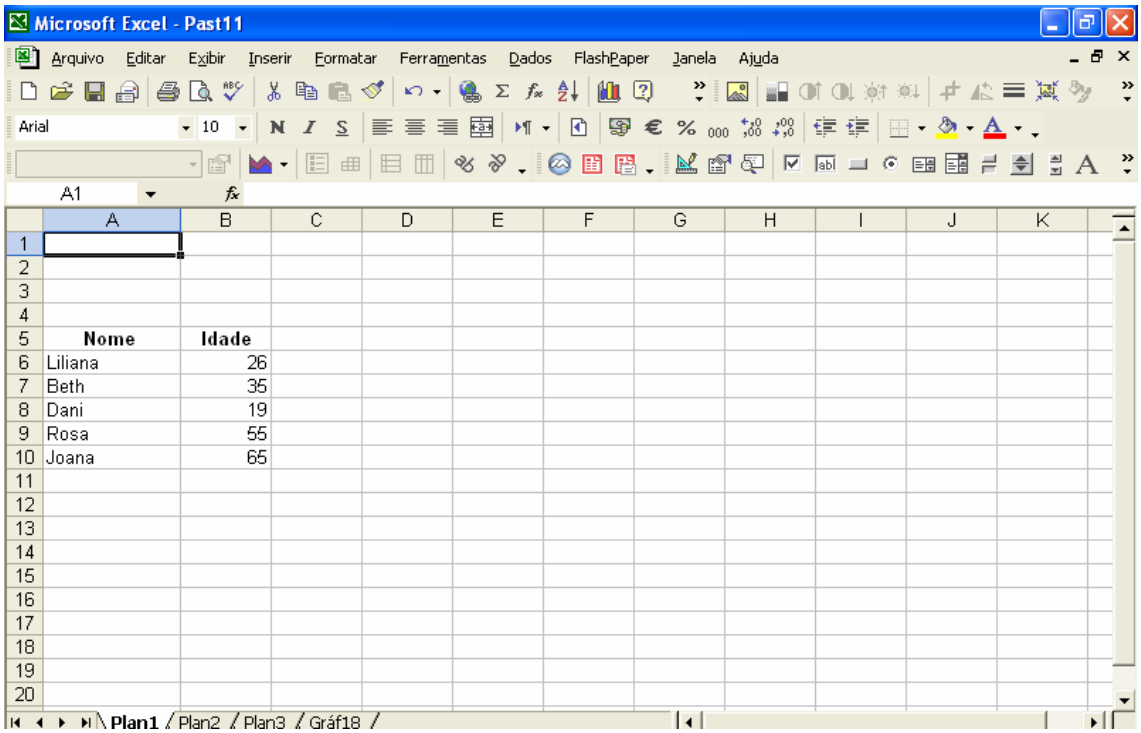
Gráfico no Excel

Bem vindos amigos a mais esta lição de código VBA do Excel, agora nos toca a aprender mais uma coisa interessante sobre macros, trata-se dos gráficos em Excel que trabalharemos agora.

Um dos poderes do Microsoft Excel é a habilidade de se criar gráficos. Com a versão 5 e a introdução de assistentes de gráfico, criar gráficos tornou-se muito fácil. Você pode criar gráficos nas suas próprias folhas ou embutir os gráficos nas planilhas. A abordagem de embutir tem a vantagem de agrupar gráficos relacionados numa única localização.

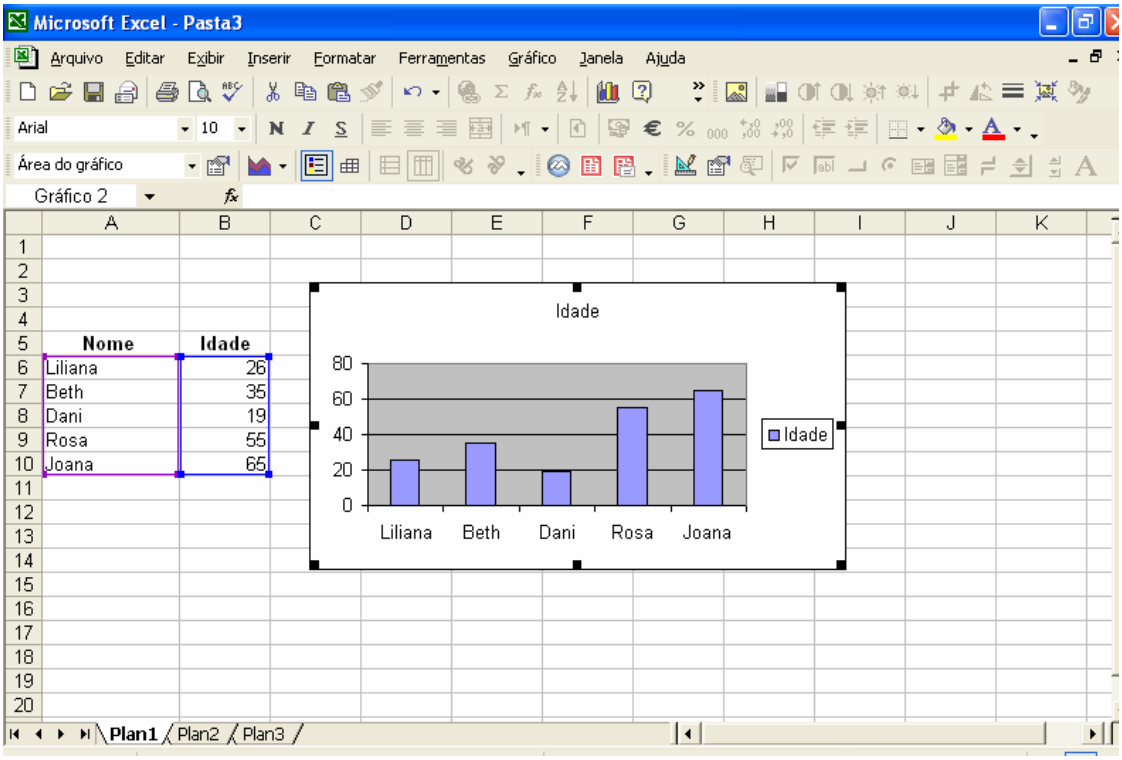
Quanto mais gráficos você criar, mais trabalho manual você deve ter que fazer para modificar cada um deles. E, se os gráficos forem gráficos embutidos vs. folhas de gráfico separadas, você deve selecionar cada gráfico embutido antes de você poder imprimí-los individualmente. Estas tarefas podem facilmente serem automatizadas por uma macro.

O propósito desta lição é mostrar-lhe como trabalhar com ambos, folhas de gráfico separadas e gráficos embutidos. E, esta lição também fornecerá várias macros de aplicações a gráficos.



	A	B
5	**Nome**	**Idade**
6	Liliana	26
7	Beth	35
8	Dani	19
9	Rosa	55
10	Joana	65

Se observarmos os dados que vamos a graficar nos damos conta que na coluna **A** se encontram os **valores dos eixos (X)** e na coluna **B** os **valores séries (Y)**, estes dados são necessários para efetuar uma gráfico que poderá ficar assim



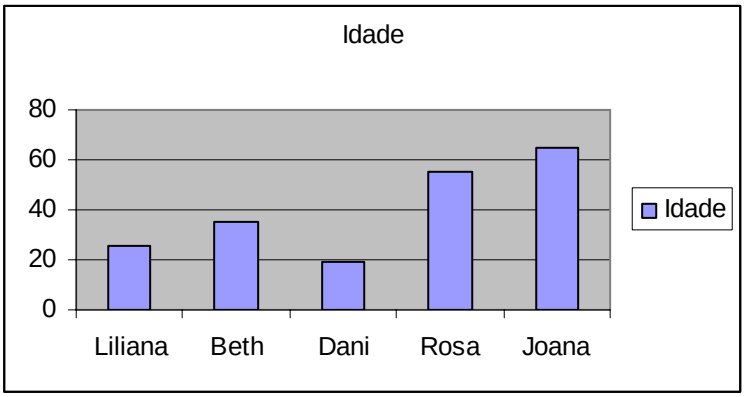
Este gráfico mostra as idades de 5 pessoas, os nomes estão no **eixo valores** e a idade no **eixo séries**, agora veremos como se pode detectar estes dados por meio de uma Macro

Ao graficar estes dados se gerou o seguinte código:

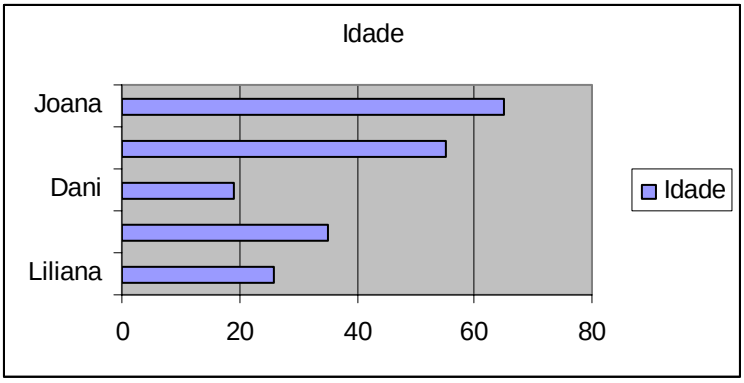
```
Sub Macro1()
   Range("A5:B10").Select
   Charts.Add
   ActiveChart.ChartType = xlColumnClustered 'Esta é a linha 3
   ActiveChart.SetSourceData
   Source:=Sheets("Plan1").Range("A5:B10"), _
      PlotBy:= xlColumns
   ActiveChart.Location Where:=xlLocationAsObject, Name:="Plan1"
End Sub
```

1. A primeira linha indica o range donde estão os dados, **eixos de valores** e **series de valores**,
2. A segunda linha indica que se agrega um gráfico
3. A terceira linha indica o tipo de gráfico que se deseja
4. A quarta linha indica como se acomodam os dados no gráfico
5. A quinta linha indica onde se mostra o gráfico, se na mesma folha ou em uma folha separada.

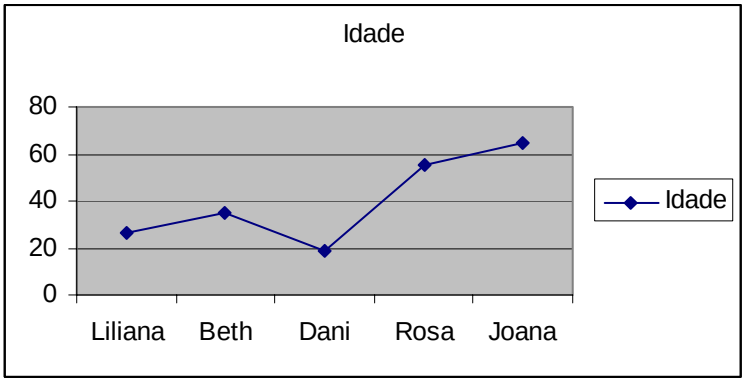
A seguir se mostram alguns dos diferentes tipos de gráficos **Línha 3**:



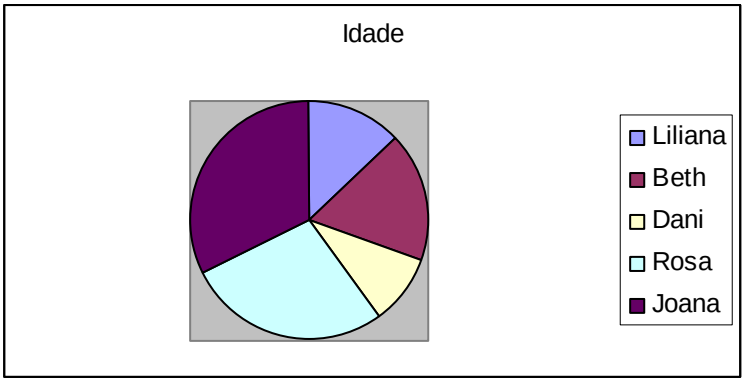
ActiveChart.ChartType = xlColumnClustered



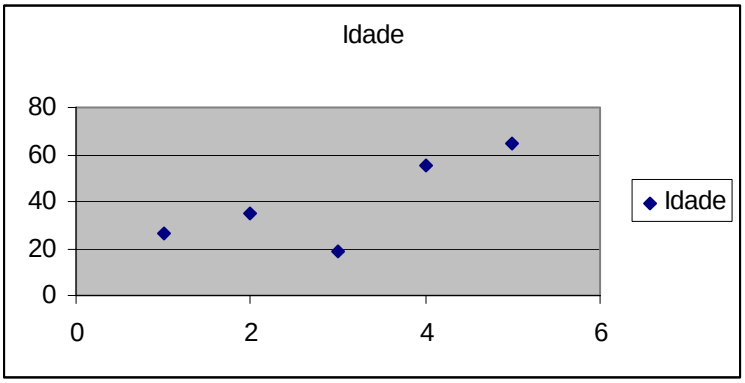
ActiveChart.ChartType = xlBarClustered



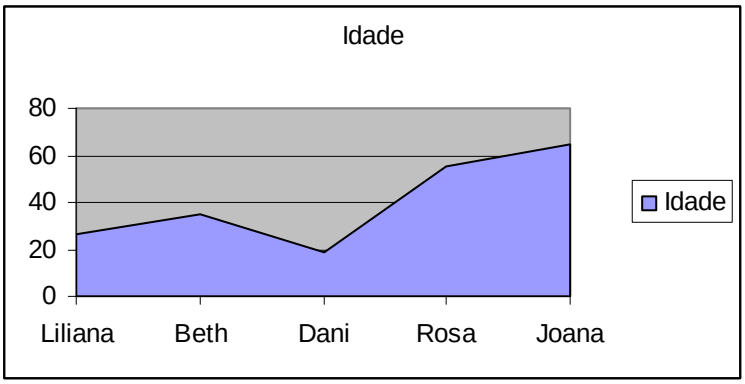
ActiveChart.ChartType = xlLineMarkers



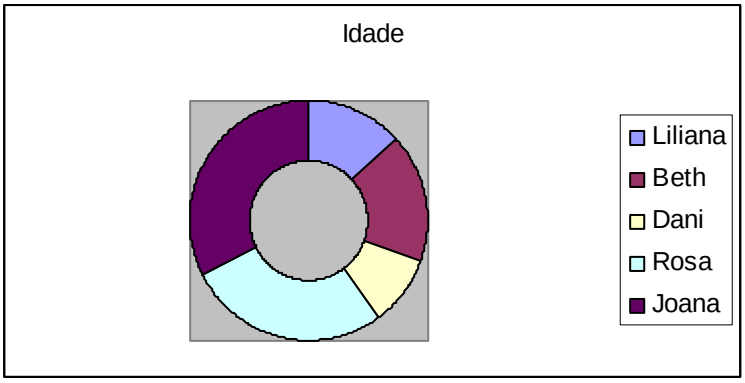
ActiveChart.ChartType = xlPie



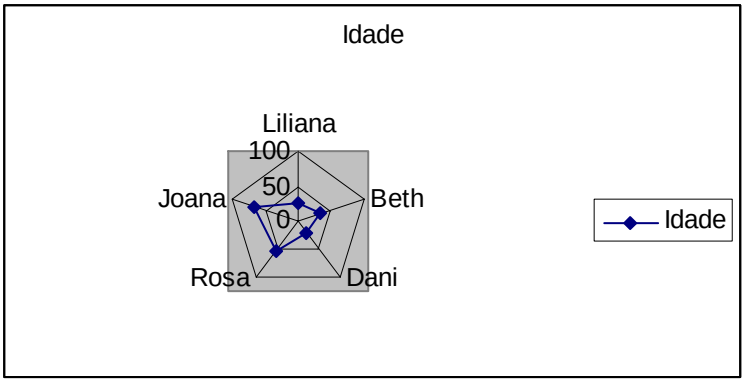
ActiveChart.ChartType = xlXYScatter



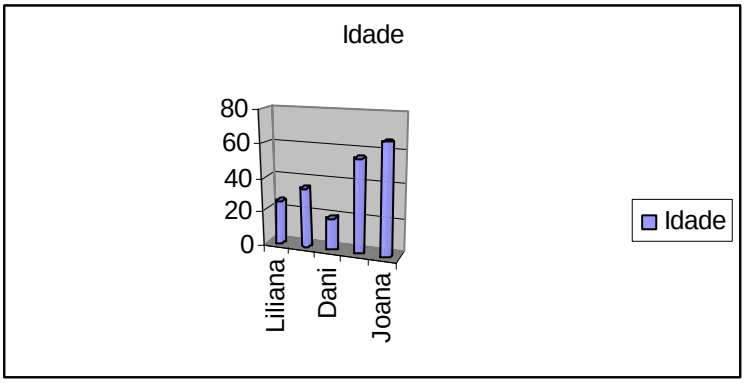
ActiveChart.ChartType = xlAreaStacked



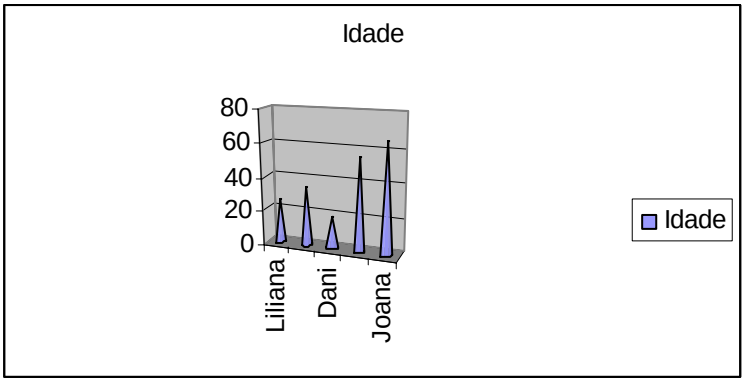
ActiveChart.ChartType = xlDoughnut



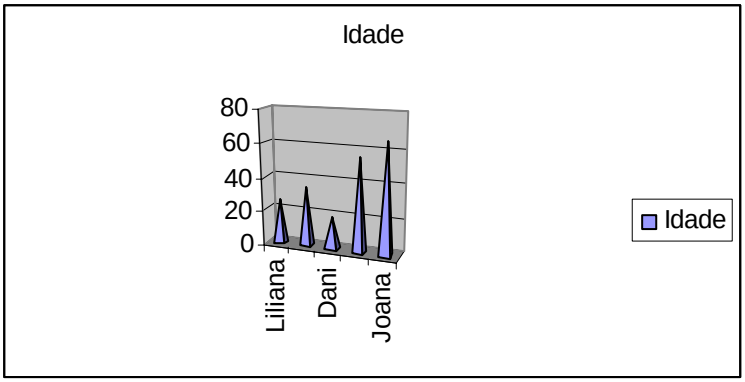
ActiveChart.ChartType = xlRadarMarkers



ActiveChart.ChartType = xlCylinderColClustered



ActiveChart.ChartType = xlConeColClustered

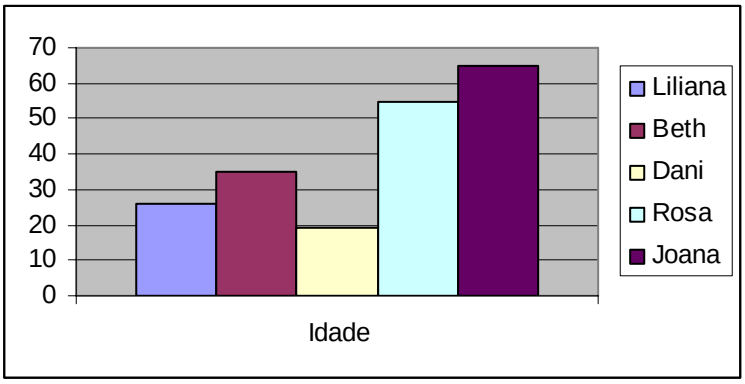


ActiveChart.ChartType = xlPyramidColClustered

Se tu agregas ao final do código principal alguma linha do tipo de gráfico que gostas, este se ativará, por exemplo:

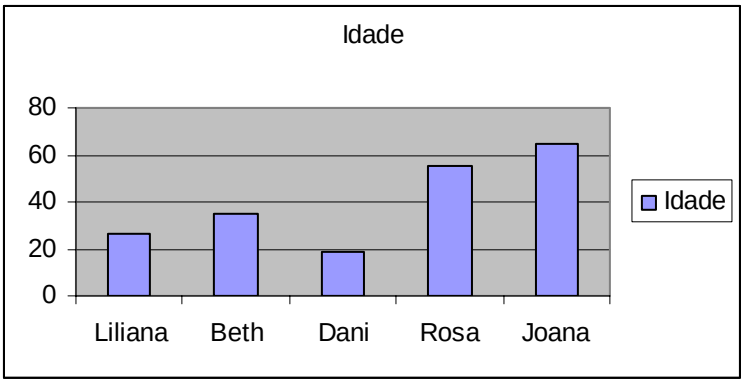
```
Sub Macro1()
Range("A5:B10").Select
Charts.Add
ActiveChart.ChartType = xlColumnClustered
ActiveChart.SetSourceData
Source:=Sheets("Plan1").Range("A5:B10"), PlotBy:= xlColumns
ActiveChart.Location Where:=xlLocationAsObject, Name:="Plan1"
ActiveChart.ChartType = xlPyramidColClustered
End Sub
```

Este código se pode programar num botão ou qualquer outro controle do **Visual Basic**.
A seguir se mostra como se acomodam os dados **Línha 4:**



ActiveChart.SetSourceData Source:=Sheets("Plan1").Range("A5:B10"), PlotBy:= _
xlRows

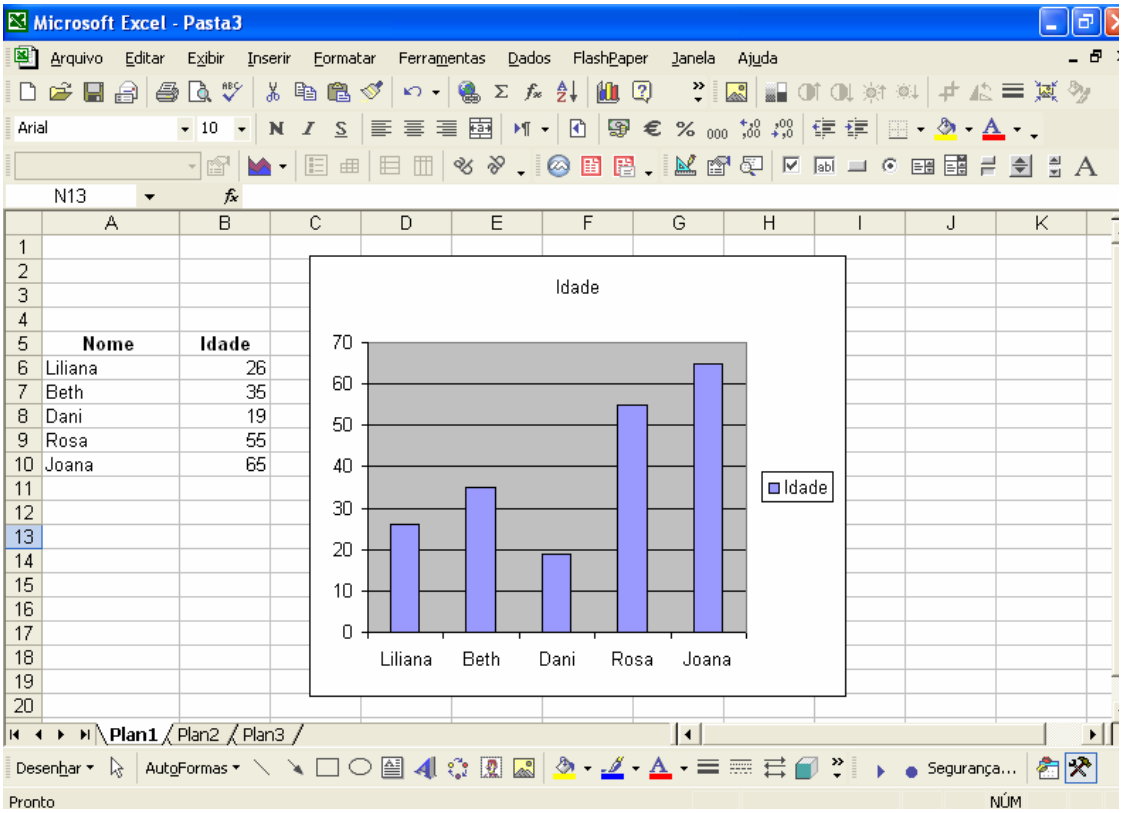
Nesta linha se mostra o gráfico por **Intervalos** (Range)



ActiveChart.SetSourceData Source:=Sheets("Plan1").Range("A5:B10"), PlotBy:= _

xlColumns

Nesta linha se mostra o gráfico por **Coluna**
Esta é a forma em que se mostram os dados do que fala a **linha 4.**



A **linha 5** fala de que se o gráfico ficar na mesma folha ou simplemente pega uma folha separada para ela, por exemplo:

ActiveChart.Location Where:=xlLocationAsNewSheet, Name:="Gráfico 1"

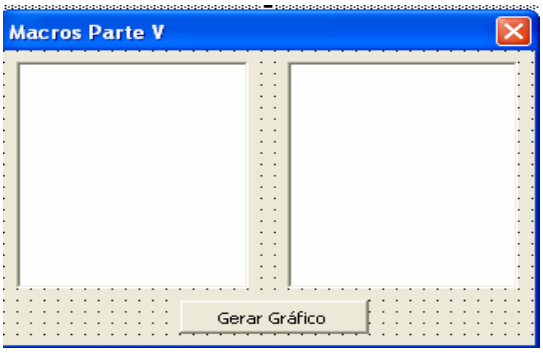
Esta linha indica que o gráfico tenha sua própria folha e que seu nome seja **Gráfico 1.**

Neste exemplo executo um código com cada uma das caractereísticas explicadas nas **5 linhas.**

```
Range("A5:B10").Select
Charts.Add
ActiveChart.ChartType = xlColumnClustered
ActiveChart.SetSourceData
Source:=Sheets("Plan1").Range("A5:B10"), PlotBy:= _xlColumns
ActiveChart.Ocation Where:=xlLocationAsObject, Name:="Plan1"

ActiveChart.ChartType = xlPyramidColClustered
ActiveChart.SetSourceData
Source:=Sheets("Plan1").Range("A5:B10"), PlotBy:= xlColumns
ActiveChart.Location Where:=xlOcationAsNewSheet, Name:="Gráfico
1"
```

- Tipo de Grafico
- Como se acomodam os dados
- Como se mostra o gráfico, neste caso em uma só folha

Elabore o seguinte formulário com o seguinte código, para observar os diferentes tipos de gráficos e a forma em que se acomodam os dados:

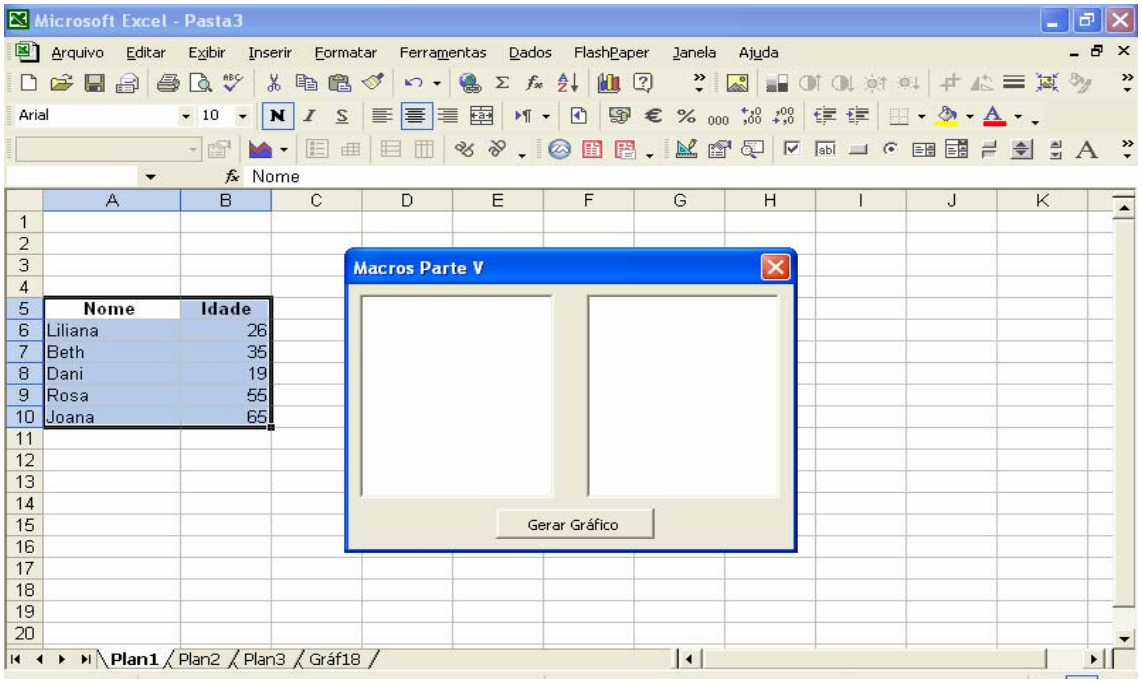


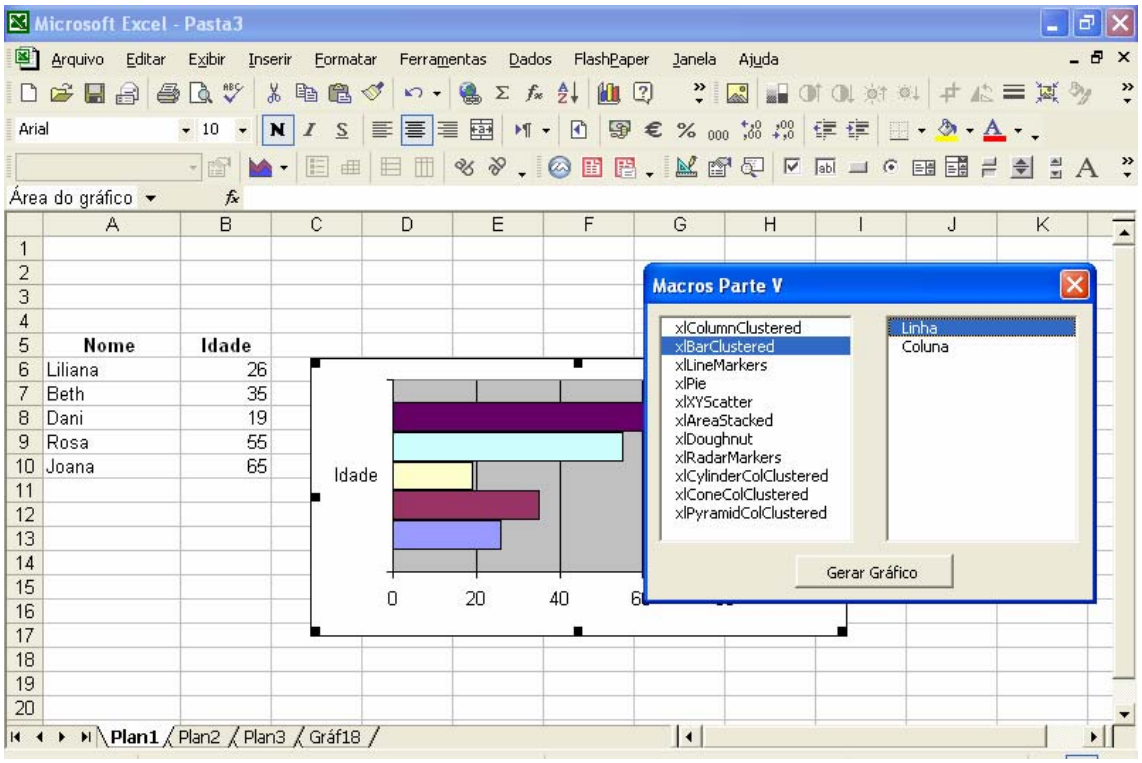
Desenhe duas **Listbox** e um **Botão** e cole o código seguinte dentro do formulário:

```
 Private Sub CommandButton1_Click()
Rem este código gera o Grafico na plan1
Range("A5:B10").Select
Charts.Add
ActiveChart.ChartType = xlColumnClustered
ActiveChart.SetSourceData
Source:=Sheets("Plan1").Range("A5:B10"), PlotBy:= _
xlColumns
ActiveChart.Location Where:=xlLocationAsObject, Name:="Plan1"
Rem agregue os diferentes tipos de gráficos à Listbox1
ListBox1.AddItem "xlColumnClustered"
ListBox1.AddItem "xlBarClustered"
ListBox1.AddItem "xlLineMarkers"
ListBox1.AddItem "xlPie"
ListBox1.AddItem "xlXYScatter"
ListBox1.AddItem "xlAreaStacked"
ListBox1.AddItem "xlDoughnut"
ListBox1.AddItem "xlRadarMarkers"
ListBox1.AddItem "xlCylinderColClustered"
ListBox1.AddItem "xlConeColClustered"
ListBox1.AddItem "xlPyramidColClustered"
Rem Agregue as diferentes formas de acomodar os dados à Listbox2
ListBox2.AddItem "Linha"
ListBox2.AddItem "Coluna"
 End Sub

Private Sub ListBox1_Click()
Rem este código dá o tipo de gráfico ao dar clique na Listbox1
If ListBox1 = "xlColumnClustered" Then ActiveChart.ChartType =
xlColumnClustered
If ListBox1 = "xlBarClustered" Then ActiveChart.ChartType =
xlBarClustered

If ListBox1 = "xlLineMarkers" Then ActiveChart.ChartType =
xlLineMarkers

If ListBox1 = "xlPie" Then ActiveChart.ChartType = xlPie
```

```
If ListBox1 = "xlXYScatter" Then ActiveChart.ChartType =
xlXYScatter

If ListBox1 = "xlAreaStacked" Then ActiveChart.ChartType =
xlAreaStacked

If ListBox1 = "xlDoughnut" Then ActiveChart.ChartType =
xlDoughnut

If ListBox1 = "xlRadarMarkers" Then ActiveChart.ChartType =
xlRadarMarkers

If ListBox1 = "xlCylinderColClustered" Then
ActiveChart.ChartType = xlCylinderColClustered

If ListBox1 = "xlConeColClustered" Then ActiveChart.ChartType =
xlConeColClustered

If ListBox1 = "xlPyramidColClustered" Then ActiveChart.ChartType
= xlPyramidColClustered

End Sub

Private Sub ListBox2_Click()
If ListBox2 = "Linha" Then
ActiveChart.SetSourceData
Source:=Sheets("Plan1").Range("A5:B10"), PlotBy:= _
xlRows
End If
If ListBox2 = "Coluna" Then
ActiveChart.SetSourceData
Source:=Sheets("Plan1").Range("A5:B10"), PlotBy:= _
xlColumns
End If
End Sub
```

Antes de executar esta **Macro** tenhas os dados anteriores na Plan1 do **Excel**





Espero que o código acima seja de utilidade.

ESPECIFICANDO CHARTS

A maneira de você especificar um gráfico numa macro depende da sua localização. Se um gráfico estiver na sua própria folha você pode especificá-lo usando a seguinte sintaxe, pois ele é um membro da **coleção** Charts:

```
Charts("nome da folha de gráfico")
ou Charts(número)
```

Se o gráfico está numa pasta diferente daquela pasta ativa, você deverá também especificar a pasta. Por exemplo:

```
Workbooks("MeusGráficos.XLS").Charts("Preço de Ações")
```

O nome da folha de gráfico é o nome que aparece na guia da folha. O número índice é um número do um ao número de gráficos na planilha. O número índice refere-se às folhas de gráfico baseadas em sua ordem na Pasta. Você pode also especificar uma folha de gráfico como um membro da coleção Sheets. Por exemplo:

```
Workbooks("MeusGráficos.XLS").Sheets("Preço de Ações")
```

Se o gráfico está embutido numa planilha, então você deverá usar a seguinte sintaxe para especificá-lo:

```
Workbook.sheet.ChartObjects("nome do gráfico ")
ou Workbook.sheet.ChartObjects(número)
```

Na sintaxe acima, você deve fornecer uma folha de referência. Exemplos típicos de folhas de referências são:

- ActiveSheet
- Worksheets("nome da folha ")
- Worksheets(número)
- Sheets("noda da folha")
- Sheets(número)

A pasta objeto referência é opcional se o gráfico embutido estiver numa pasta ativa. Por exemplo:

```
Worksheets("Reps Vendas").ChartObjects("Gráfico 1")
Sheets("Reps Vendas").ChartObjects("Gráfico 2")
```

Se você gravou uma macro que funciona com gráficos embutidos, você encontrará que o Microsoft Excel se refere aos gráficos como membros da **coleção** DrawingObjects. Como gráficos embutidos numa folha de planilha são também considerados parte da coleção DrawingObjects, eles podem também ser especificados usando a seguinte sintaxe:

```
Workbook.sheet.DrawingObjects("nome do objeto")
ou Workbook.sheet.DrawingObjects(número)
```

Por favor note que a **coleção** DrawingObjects inclui qualquer objeto embutido numa planilha. Por exemplo, caixas de textos, retângulos, flechas, círculos, etc.. Se você

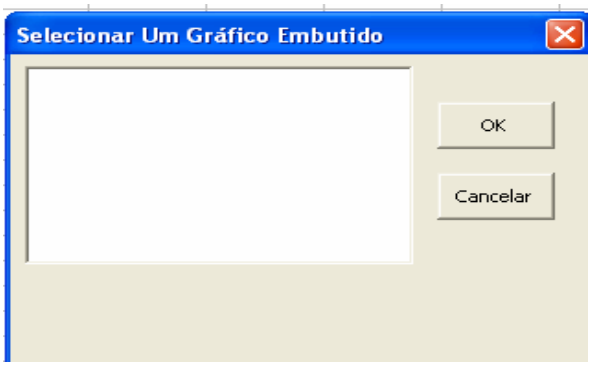
quiser se referir a apenas gráficos embutidos, então use a coleção ChartObjects, não a coleção DrawingObjects.

NOMEANDO CHARTS EMBUTIDOS

Quando você criar um gráfico embutido, o Microsoft Excel atribui a ele um nome default tal como "Chart 1", "Chart 2", etc.. Uma coisa que você deve não ter percebido é que você pode mudar o nome dos gráficos embutidos. Para fazer isto, primeiro mostre a barra de ferramentas de desenho (Exibir, Barra de ferramentas). Daí então selecione os objetos com a ferramenta seleção clicando no botão que parece com a ponta de uma flecha. Quando você clicar num gráfico embutido com esta ferramenta, você pode então clicar na caixa de nome e mudar o nome do gráfico.

UMA MACRO PARA SELECIONAR GRÁFICOS EMBUTIDOS

Se você tiver vários gráficos embutidos em uma folha de trabalho, a macro seguinte mostrará uma lista de seleção para você selecionar com qual gráfico embutido você quer trabalhar. Para criar esta macro, primeiro crie um *userform* (o nosso é chamado UserForm1). Ele deverá ter uma caixa de listagem (chamada *ListBox1*) junto com os botões OK e Cancelar. O que segue é o que esta folha deverá se parecer:



A propriedade caption do userform foi mudada para dizer "Selecionar Um Gráfico Embutido". Para você fazer isto selecionar o userform e daí eles são CommandButton1 e CommandButton2:

```
Private Sub CommandButton1_Click()
Me.Hide
bResponse = True
End Sub

Private Sub CommandButton2_Click()
Me.Hide
bResponse = False
End Sub
```

O que segue é a macro que você precisa entrar. Por favor note que a variável global é chamada no topo do módulo, e que ela é usada para armazenar o valor Boolean indicando qual botão foi clicado.

```
Global bResponse As Boolean
```

```
Sub SelectAnEmbeddedChart()
'Declarar variáveis:
Dim I As Integer
Dim chartName As String
Dim response As Boolean
'certificar-se que o userform está descarregado.
Unload UserForm1
'Ou, apenas limpar a listbox:
UserForm1.ListBox1.Clear
'atualizar a lista listbox
For I = 1 To ActiveSheet.ChartObjects.Count
chartName = ActiveSheet.ChartObjects(I).Name
UserForm1.ListBox1.AddItem chartName
Next
'tornar o 1º gráfico da lista a seleção inicial
UserForm1.ListBox1.ListIndex = 0
'mostrar o form
UserForm1.Show
'mostrar o diálogo e selecionar o gráfico
If bResponse = False Then
'descarregar e sair se cancelar for selecionado
Unload UserForm1
Exit Sub
End If
'Obter o nome da listbox do item selecionado
chartName = UserForm1.ListBox1.Value
'remover o userform da memória
Unload UserForm1
'Ativar o gráfico
ActiveSheet.ChartObjects(chartName).Activate
End Sub
```

Pontos chaves sobre a macro acima:

- . Esta macro deverá ser executada de uma folha contendo gráficos.
- A declaração "ChartObjects.Count" é usada para obter o número de gráficos embutidos na folha.
- Como cada ChartObject é identificado por ambos, um número e um nome, a macro circula pela coleção ChartObject e usa o número de um objeto gráfico para determine seu nome.
- A expressão UserForm1.ListBox1.Value é a string texto na caixa de listagem do item selecionado.
- O comando Activate é usado em vez do comando Select na macro acima. O comando Activate torna o gráfico no objeto embutido ativo. Deste modo, você pode usar a declaração ActiveChart.PrintOut para imprimir o gráfico. Se você somente selecionar o objeto embutido, você precisará ainda ativar o gráfico de modo que você possa trabalhar com ele ou imprimí-lo.

Se você quiser modificar a macro acima para primeiro classificar a lista dos objetos embutidos em ordem alfabética você precisará de:

- Armazenar os nomes dos gráficos numa array
- Classificar o array em ordem alfabética como ilustrado no capítulo de arrays.

- Definir a propriedade List da caixa de listagem igual ao array classificado.

IMPRIMINDO E VISUALIZANDO GRÁFICOS

Uma das coisas freqüentes que as pessoas fazem com gráficos é imprimí-los! O método que se usa para imprimir um gráfico é chamado PrintOut. Se você quer visualizar um gráfico, então você deverá usar o método PrintPreview. Se a folha de gráfico for uma folha ativa, então você pode usar a seguinte declaração para imprimir um gráfico:

```
ActiveChart.PrintOut
```

Se a folha de gráfico não for uma folha ativa, você pode usar a seguinte declaração para imprimir o gráfico sem ativar a folha de gráfico:

```
Workbook.Charts("gráfico sheet nome").PrintOut
```

O objeto referência da pasta não é exigido se a pasta contém o gráfico é a pasta ativa. Um número de folha de gráfico pode ser usado em vez do nome. Por exemplo, a declaração que segue imprime uma folha de gráfico rotulada de "Previsão de Vendas" localizada na pasta 1996Fcst.XLS:

```
Workbooks("1996Fcst.Xls").Charts("Previsão de Vendas").PrintOut
```

Quando você usar o comando acima, você notará que o Microsoft Excel ativará a folha de gráfico, imprimirá o gráfico, e daí então ativará a folha que estava ativa anteriormente ao comando ser emitido.

Esta mesma aboragem não funciona para gráfico embutidos. Se você tentar usar a sintaxe acima para imprimir um gráfico embutido, você obterá uma mensagem de erro. Em vez disso, você precisa primeiro ativar o gráfico embutido e daí então enviar uma declaração do Visual Basic para imprimir o gráfico. O que segue ilustram as declarações do Visual Basic exigidas:

```
Workbook.sheet.ChartObjects("nome").Activate
ActiveChart.PrintOut
```

Você também tem a opção de entrar com um número para o ChartObject em vez de um nome. A macro que segue ilustra como você imprimirá todos os gráficos localizados em folhas de gráfico individuais:

```
Sub PrintCharts()
Dim chtSheet As Object
For Each chtSheet In Charts
chtSheet.PrintOut
Next
End Sub
```

A macro acima imprimirá todos os gráficos localizados em folhas de gráfico individuais porque eles são membros da coleção Charts. Cada pasta tem sua própria coleção Charts.

Se você quiser imprimir todos os gráficos embutidos numa planilha, então você deverá usar a coleção ChartObjects em vez da coleção Charts. Também, você deverá ativar cada gráfico embutido antes de imprimir. Por exemplo:

```
Sub PrintEmbeddedCharts()
Dim embeddedCht As Object
For Each embeddedCht In ActiveSheet.ChartObjects
embeddedCht.Activate
ActiveChart.PrintOut
Next
End Sub
```

Note que ChartObjects tem sido qualificado como um objeto folha. Neste caso ActiveSheet foi usada. Entretanto, você não tem que estar na folha ativa para ativar e imprimir o gráfico embutido. Por exemplo, você poderia ter usado Workbook("Forecast.Xls").Sheets("gráfico Sheet") em vez de ActiveSheet:

```
Sub PrintEmbeddedCharts()
Dim embeddedCht As Object
Set containingSheet = _
Workbooks("Forecast.Xls").Sheets("gráfico Sheet")
For Each embeddedCht In containingSheet.ChartObjects
embeddedCht.Activate
ActiveChart.PrintOut
Next
End Sub
```

Entretanto, quando as declarações são executadas, você acabará numa folha contendo os gráficos, mesmo se você não estivesse nesta folha originalmente. E, o último gráfico embutido na coleção será o objeto selecionado, em oposição a uma célula na planilha.

As macros acima imprimirão os gráficos na ordem que o Microsoft Excel os têm em suas coleções. Se você quiser controlar a ordem de impressão, então você precisa usar declarações print individuais para cada gráfico ou objeto embutido:
Para folhas de gráfico você deverá usar a seguinte sintaxe:

```
pasta.Charts("gráfico 1 nome").PrintOut
pasta.Charts("gráfico 2 nome").PrintOut
```

O objeto pasta lhe permite especificar uma folha de gráfico numa pasta diferente.
Para gráficos embutidos você deverá usar a seguinte sintaxe:

```
sheet.ChartObjects("gráfico embutido 1 nome").Activate
ActiveChart.PrintOut
sheet.ChartObjects("gráfico embutido 2 nome").Activate
ActiveChart.PrintOut
```

Uma referência de folha deve qualificar ChartObjects para gráficos embutidos. Ela pode identificar uma folha em uma pasta diferente daquela da folha ativa.

OBJETOS E PROPRIEDADES DE UM CHART

Existem muitos objetos diferentes num gráfico. Por exemplo existem linhas, bordas, backgrounds, etc.. O modo mais fácil, e freqüentemente o melhor modo, para se determinar o nome do objeto gráfico e as propriedades que você quer trabalhar com elas é usar o gravador de macro. Examinando a macro resultante você pode facilmente determinar o nome do objeto gráfico e suas propriedades. Para obter informações adicionais sobre um objeto gráfico e suas propriedades, coloque o cursor na palavra chave e pressione a tecla F1. Isto mostrará a ajuda do Visual Basic sobre este assunto.

Se você precisar fazer a mesma modificação a uma série de gráficos, uma macro é uma maneira fácil de fazer isto. O modo mais fácil de se criar uma tal macro é usar o gravador de macro e gravar as mudanças do gráfico para cada. Then, you would edit the resulting macro, eliminando o código extra que o Microsoft Excel grava, e usar um For Each..Next para fazer o laço por todos os gráficos. A macro que segue loops through all the gráfico embutidos numa planilha e muda a linha 1 em cada gráfico para uma linha pontilhada:

```
Sub ChangeEmbeddedCharts()
Dim E As Object
For Each E In ActiveSheet.ChartObjects
E.Activate
'selecionar a series 1
ActiveChart.SeriesCollection(1).Select
'modificar para uma linha fina pontilhada
With Selection.Border
.ColorIndex = 1
.Weight = xlThin
.LineStyle = xlDot
End With
Next
End Sub
```

SOBRE AS SÉRIES NO GRÁFICO

O objeto mais importante num gráfico são as séries que estão sendo plotadas. Séries são referidas com a seguinte sintaxe:

Chart.SeriesCollection(número)

A referência Chart é exigida. Freqüentemente você deverá usar ActiveChart como a referência gráfico. Para determinar o número de uma série, clique na série e o nome da série aparecerá na caixa nome. O nome será como S1, S2, etc.. O número representa o número que você deverá usar para identificar a série usando a plavra chave seriesCollection.

Para modificar uma série (linha, barra, etc..), você deverá definir a propriedade Formula da linha igual a uma nova fórmula usando a fórmula Series. A fórmula Series deve ter a seguinte sintaxe:

"=Series(series nome, X axis categories, Y axis values, series número)"

Por exemplo:

=SERIES(Sheet3!$C$8, Sheet3!$D$6:$G$6,Sheet3!$D$8:$G$8, 1)
ou =SERIES("Test Data", Sheet3!$D$6:$G$6,Sheet3!$D$7:$G$7, 2)

Se você clicar numa série num gráfico você verá a fórmula Serie na caixa de edição. Qualquer fórmula que você construir via uma macro deve se parecer com as fórmulas que você vê na caixa de edição. Os pontos principais sobre a sintaxe Series numa macro:

- O sinal de igual e aspas são exigidos.
- O nome series e categorias do eixo X são opcionais. Os valores do eixo Y e número série são exigidos. Se o número de série for o mesmo que aquele de uma série existente, então a série existente é tocada.
- Vírgulas são usadas como placeholders.
- O nome da série pode ser ou uma referência a uma célula única na planilha ou uma entrada de texto entre aspas duplas.
- As categorias eixo X são ou os valores de X para gráficos de scatter charts ou rótulos do eixo X para todos os outros gráficos.
- As entradas para as categorias do eixo X e valores do eixo Y são normalmente entradas de endereços no estilo A1. Por exemplo, Sheet1!$C$3:$G$3
- Se você estiver criando uma entrada de endereço, você pode usar a propriedade Address para obter as referências de células, e daí então concatenar as entradas, incluindo quaisquer aspas simples necessárias, pontos de exclamação, parênteses, e colchetes

As declarações Visual Basic que seguem mostram como fazer isto para uma série de células que tenham sido destacadas e são necessárias para especificar series 2 num gráfico chamado "Key Data". Por favor note que é um gráfico numa folha separada, e não é um gráfico embutido.

```
Sub AddressExample()
Dim yCells As String, sheetName As String
'Esta macro assume que você a tenha executado de uma planilha
'e tenha destacado o novo range Y
'Obter o endereço do address para a seleção
yCells = Selection.Address
'Obter o nome da folha
sheetName = ActiveSheet.Name
'Ir ao gráfico - note que o nome foi definido pelo usuário
Charts("Key Data").Select
'especificar a new formula for the series
ActiveChart.SeriesCollection(2).Formula = _
"=SERIES(,,'" & sheetName & "'!" & yCells & ",4)"
End Sub
```

Note que na construção da declaração Formula acima que aspas simples foram usadas para cercar o noma da folha. Esta é uma exigência se o nome da folha tem espaços, e não causar problema se o nome não tiver qualquer espaço. Se você precisar adicionar uma nova série a um gráfico, então você deverá usar o método Add.
A sintaxe é:

```
gráfico.SeriesCollection.Add _
source:= range address, _
RowCol:= xlRows or xlColumns, _
seriesLabel:= True or False, _
categoryLabel:= True or False, _
replace:=True or False
```

Se o gráfico for um gráfico embutido, você deve primeiro ativar o gráfico e daí então usar ActiveChart para se referir ao gráfico na declaração acima. Para certificar de que você obteve o conjunto correto de declarações, use o gravador de macro.

O que segue são os argumentos do método Add para a SeriesCollection:

- fonte - exigido. Isto é normalmente umendereço de range no text form dos dados a serem plotados. O que segue ilustra este argumento: Source:= "Sheet1!$C$4:$G$5". Se X labels and a series label forem necessários, eles estão incluidos nesta *string*.
- rowCol - opcional. Isto indica se os dados estão em linhas (xlRows) ou colunas (xlColumns). Se omitido, o Microsoft Excel tenta compreender a resposta correta.
- seriesLabel - opcional. Se a primeira célula no range especificado range contém o nome da série de dados, este parâmetro é definido como True. Se não for, então ele deverá ser definido como False. Se omitido, o Microsoft Excel tenta compreender a resposta correta.
- categoryLabel - opcional. Se a primeira célula no range especificado range contém os valores X, este parâmetro é definido como True. Se não for, então ele deverá ser definido como False. Se omitido, o Microsoft Excel tenta compreender a resposta correta.
- replace - opcional. Se os rótulos da categoria são fornecidos e esta variável é definida como True, então os novos rótulos trocam aqueles já em uso no gráfico.

Caso contrário os rótulos deverão ser rótulos adicionais.
Para deletar uma série num gráfico, você deverá primeiro selecionar o gráfico e dai então especificar o ActiveChart e usar o método Delete. Por exemplo:

```
Sheets("Chart3").Select
ActiveChart.SeriesCollection(2).Delete
```

UM EXEMPLO DE MACRO QUE DESENHA GRÁFICO A PARTIR DE DADOS

Se sua base de dados consistir de apenas umas poucas linhas de dados, ela , é bastante fácil de criar gráficos para cada linha de dados. Mas se seus dados estiverem em muitas linhas para tornar a criação de gráficos individuais uma possibilidade

razoável, então você pode usar uma macro para plotar uma linha selecionada (ou múltiplas linhas) de dados no comando.

O truque para ser capaz de plotar seletivamente uma linha de dados é o design da sua planilha. Você precisará de três folhas:

- Uma folha de gráfico
- . Uma planilha que mantenha apenas os dados que estão sendo graficados
- Uma planilha que tenha seus dados.

Para ilustrar uma tal macro, vamos assumir que seus dados representem shipments por cliente. O que segue ilustra a planilha de dados:

	A	B	C	D	E	F
1		1ST	2ND	3RD	4TH	
2		QTR	QTR	QTR	QTR	
3						
4	IBM	172	188	184	195	
5	APPLE	45	40	51	52	
6	DELL	22	34	12	21	
7	SONY	111	127	233	134	
8						

A folha acima deverá ser chamada "The Data". Por favor note que você pode usar nomes diferentes se você mudar a macro. Apesar da ilustração acima ter somente 4 linhas de dados, por favor assuma que existam centenas de clientes. Também, esta abordagem funcionará se os períodos forem meses, anos, etc..

A segunda folha na sua planilha deverá consistir de duas linhas: os rótulos que você quer usar para o eixo X, e uma linha de dados. O que segue ilustra esta folha:

	A	B	C	D	E	F
1		1Q	2Q	3Q	4Q	
2	XYZ Customer	44	55	33	22	
3						

Esta folha deverá ser chamada "Chart Data". Por favor note que o que está acima são amostras de dados. Mais tarde, a macro os trocará com os dados reais.

O próximo passo é criar o gráfico que será usado. Isto é feito simplesmente destacando as células acima (A1:E2) na folha "Chart Data" e selecionar Inserir, Chart, As New Sheet. O que segue é o gráfico que eu construi dos dados acima. Ele é um gráfico de linha usando format 4, o qual dá linhas pontilhadas para cada valor do eixo Y.

O nome que a macro usará para a folha de gráfico acima é "Customer Shipments".

A macro que nós escreveremos será executada da planilha contendo os dados.

Sempre que a célula for a ativa, a macro determinará a linha de dados a serem plotados. A macro selecionará a linha de dados incluindo o nome do cliente, e o copiará para a folha "Graph Data". Ela então atualizará o título do gráfico e mostrará o gráfico. O que segue é o código da macro que alcança este resultado:

```
Sub PlotData()
Dim customerNameCell As Object, endCell As Object
'tenha certeza de estar na folha correta
If ActiveSheet.Name <> "The Data" Then
MsgBox "Você não está na planilha correta."
End
End If
'definir a variável objeto igual a primeira célula na linha, a
qual está o nome do cliente
Set customerNameCell = Cells(ActiveCell.Row, 1)
'Confirmar se ela tem uma entrada
If customerNameCell = "" Then
MsgBox "Você não está numa linha de dados."
End
End If
'determinar a última célula, começando da célula nome do
'cliente
Set endCell = customerNameCell.End(xlToRight)
'selecionar e copiar os dados para ser plotados
Range(customerNameCell, endCell).Copy _
destination:=Sheets("Chart Data").Range("A2")
'recalc Microsoft Excel no caso calc é manual.
'Isto atualiza o gráfico se calc é ajustado para manual
If Application.Calculation = xlManual Then Calculate
'Ir para o gráfico e fixar o título
Sheets("customer Shipments").Select
ActiveChart.ChartTitle.Text = _
customerNameCell.Value & " CARREGAMENTOS"
End Sub
```

Para testar sua cópia da macro acima, selecione qualquer célula contendo dados na folha chamada "The Data" e execute a macro. Se você quiser, você pode ocultar a folha "Chart Data" desde que ela nunca esteja selecionada. Uma coisa que você deve querer fazer é to montar um botão na folha "The Data" que automaticamente chame a macro. Isto eliminará de ter que selecionar manualmente Ferramentas, Macros para executar a macro.

Para criar um botão na planilha de dados:

- Selecionar Exibir, Barra de ferramentas e mostrar a barra de ferramentas Drawing. Sair dos menus.
- Clicar no botão "Criar Botão".
- Ir a célula A1 e desenhar um botão nas células A1:A2. Quando prompted para a macro rodar, selecione a macro entitulada "PlotData".
- Mude o título do botão para a palavra "Plot".

Se você precisar fixar o rótulo do botão por qualquer razão, apenas mantenha apertada a tecla CTRL e clique no botão para selecioná-lo. Se você precisar checar ou mudar a macro, dê um clique com o botão direito e selecione atribuir uma macro.

O último item que você poderia fazer é congelar as linhas e colunas de cabeçalho na folha de dados. Para congelá-los, selecione a célula B3 e selecione Janela, Congelar painéis. Isto mantém os títulos sem deslocar na tela, e mantém também o botão visível todo o tempo. O que segue é o que a folha agora se parece:

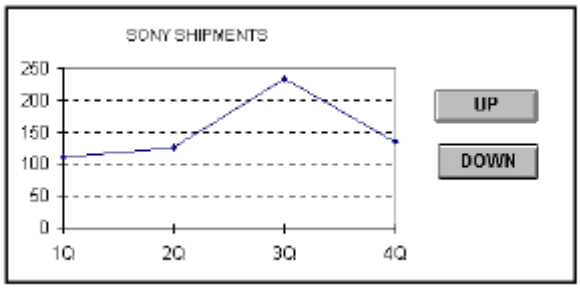
	A	B	C	D	E	F
1	Plot	1ST	2ND	3RD	4TH	
2		QTR	QTR	QTR	QTR	
3						
4	IBM	172	188	184	195	
5	APPLE	45	40	51	52	
6	DELL	22	34	12	21	
7	SONY	111	127	233	134	
8						

Um dos problemas com a abordagem acima é que você tem de retornar continuamente à folha de dados da folha de gráfico para plotar o próximo cliente. Uma maneira de contornar isto é criar duas macros adicionais e colocar botões que as chamem da folha de gráfico . Uma move uma linha para baixo e chama a macro PlotData acima. A outra move uma linha para cima e chama a macro. Se também se encontrar uma linha em branco, ela dá a mensagem que se está no topo ou no fundo dos dados. O que segue são estas duas macros:

```
Sub MoveDown()
'ir para a folha de dados. Activate é usado no caso da folha
estar oculta.
Sheets("Os Dados").Activate
'mover para baixo uma linha e chamar a macro plotdata
ActiveCell.Offset(1, 0).Select
If Cells(ActiveCell.Row, 1) = "" Then
'retorne ao plot se estiver no fundo
ActiveCell.Offset(-1, 0).Select
Charts("customer Shipments").Select
MsgBox "Você está no fundo dos dados"
Exit Sub
End If
'plot a próxima linha chamando a macro PlotData
PlotData
End Sub
'**************************************************
Sub MoveUp()
'ir a folha de dados. Activate é usado no caso da folha estar
oculta.
Sheets("The Data").Activate
'mover para cima uma linha e toda macro plotdata macro
ActiveCell.Offset(-1, 0).Select
If Cells(ActiveCell.Row, 1) = "" Then
'retornar ao plot se estiver no topo
ActiveCell.Offset(1, 0).Select
```

```
Charts("customer Shipments").Select
MsgBox "Você está no topo dos dados"
Exit Sub
End If
'plot a próxima linha chamando a macro PlotData
PlotData
End Sub
'*****************************************************
```

Da mesma maneira que você adicionou um botão à folha de dados para chamar a macro PlotData, você poderá adicionar dois botões à folha de gráfico para chamar as macros acima. Isto lhe permitirá rodar pelos dados sem ter que retornar à planilha de dados. O que segue é com que aquela folha de gráfico modificada se parece:



Por favor note que os botões na ilustração acima são grandes em relação ao gráfico para ilustrar botões num gráfico. Na sua aplicação real, eles podem ser muito pequenos. Também, você pode ocultar uma folha de dados quando as macros MoveUp e MoveDown usarem Activate em vez de Select para ir à folha de dados. O método Activate permite-lhe ir às folhas ocultas e trabalhar com elas como se elas estivessem visíveis. Como se mencionou anteriormente, a folha "Chart Data" pode também ocultar quando ela nunca for selecionada.

Uma outra melhoria que você poderia adicionar seria um botão that mostra uma lista das linhas de dados, permitindo o usuário escolher qual linha de dados mostrar no gráfico.

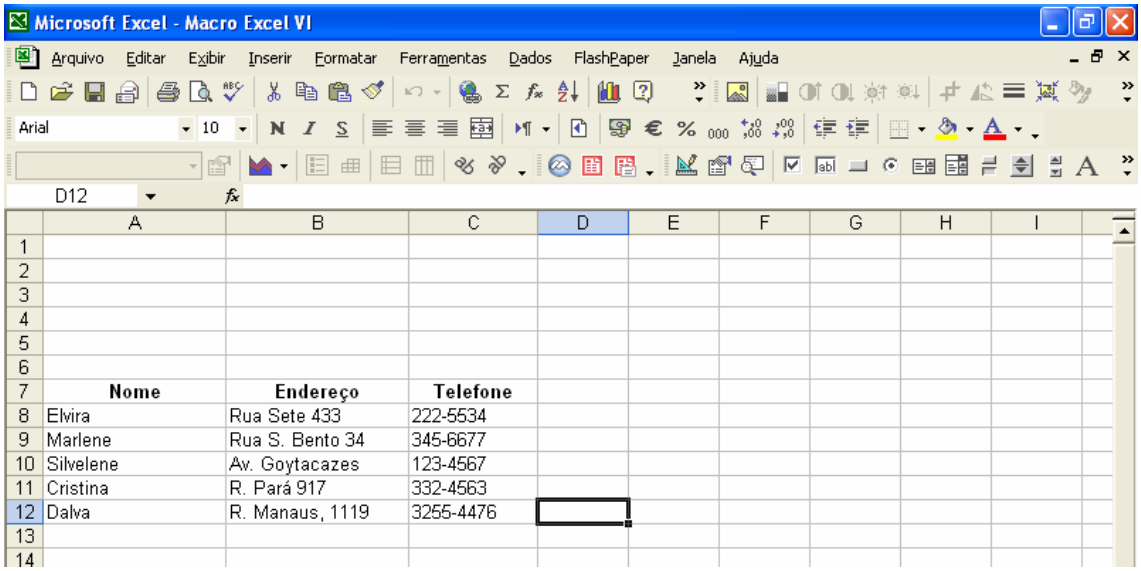
COMO EVITAR A SELEÇÃO DE UM GRÁFICO EMBUTIDO

Meus agradecimentos a *Tom Ogilvy* pelo exemplo que ilustra como evitar selecionar uma ativação de um gráfico embutido. Evitar a seleção e ativação torna seu código mais eficiente e mais fácil de escrever.

```
Sub ChartExample()
Dim cht As Object
Set cht = ActiveSheet.ChartObjects("Chart 1").Chart
With cht.PlotArea
With .Border
.Weight = xlThin
.LineStyle = xlAutomatic
End With
.Interior.ColorIndex = xlNone
End With
End Sub
```

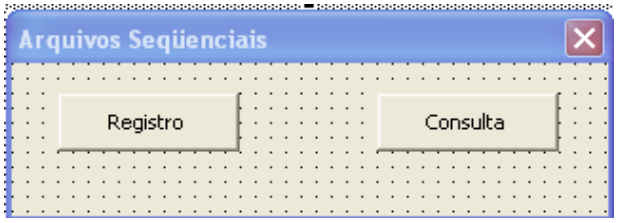
APRENDENDENDO A TRABALHAR COM ARQUIVOS SEQÜENCIAIS NO VISUAL BASIC

Bem vindos amigos à sexta parte de **Macros em Excel e Visual Basic**, estamos prontos para ver mais sobre este interessante curso, neste caso veremos como se pode arquivar os dados de uma folha em um arquivo aparte. Aprenderemos a trabalhar com **arquivos seqüenciais** em **Visual Basic**. Os **arquivos seqüenciais** são aqueles que ao registrar seus dados estes entram numa seqüência, por exemplo se registro 5 nomes, levarão uma ordem de 1 ao 5, em troca existem também os **arquivos aleatórios**, mas eles não respeitam a seqüência, por exemplo os 5 nomes poderiam ficar em qualquer posição do 100 em diante, do 300 em diante, do 10 em diante, de onde quiseres pô-los, tu indicas aonde queres que fiquem os 5 nomes, podem ficar até separados e não respeitar uma seqüência.
O problema dos **arquivos seqüenciais** é que ao se introduzirem alguns símbolos na busca podem alterar o arquivo e este não funcionar corretamente, por isso se recomenda filtrar os dados com algum código ou simplemente não procurar por símbolos.



	A	B	C	D	E	F	G	H	I
1									
2									
3									
4									
5									
6									
7	**Nome**	**Endereço**	**Telefone**						
8	Elvira	Rua Sete 433	222-5534						
9	Marlene	Rua S. Bento 34	345-6677						
10	Silvelene	Av. Goytacazes	123-4567						
11	Cristina	R. Pará 917	332-4563						
12	Dalva	R. Manaus, 1119	3255-4476						
13									
14									

Nesta folha podemos observar 5 nomes, a intenção será arquivá-los em separado e fazê-los desaparecer da folha, para depois voltarmos fazer aparecer na folha. A isto se chamará Registro de dados e Consulta de dados.



Iremos ao **Visual Basic** com **Alt+F11** e Inseriremos um **UserForm,** no qual desenharemos dois botões, um com o nome de **Registro** e o outro com o Nome de **Consulta**.

Agora vamos programar o botão Registro, para poder arquivar os nomes.

```
Private Sub CommandButton1_Click()
Rem se translada à célula A8
Range("A8").Select
Rem se não há nenhum dado na A8 que não se arquive de novo
If ActiveCell = Empty Then GoTo salte
Rem abre um arquivo na unidade c com o nome de dados.txt
Rem em forma de Temporal (Output) na área de armazenamento #1
Open "c:\dados.txt" For Output As 1
Rem ativa um rótulo para poder regressar
volte:
Rem escreva o dado da célula ativa no arquivo
Write #1, ActiveCell
Rem apaga o dado da célula
ActiveCell = Empty
Rem baixa um range para o nome seguinte
ActiveCell.Offset(1, 0).Select
Rem se a célula esta vazia que não regresse agora
If ActiveCell = Empty Then GoTo salte
Rem volte a escrever o seguinte nome no arquivo
GoTo volte:
salte:
Rem acabou-se
Rem fechar o arquivo
Close #1
End Sub
```

Os dados ficarão arquivados na unidade e serão devolvidos quando pressionares o botão consulta. O que a seguir se mostra:

Agora vamos programar o botão consulta:

```
Private Sub CommandButton2_Click()
Rem transladar à célula A8
Range("A8").Select
Rem Abre um arquivo na unidade C com o nome de dados.txt
Rem na forma de Leitura (input) na área de armazenamento #1
Open "c:\dados.txt" For Input As 1
Rem isto significa laço, embora não seja final do arquivo
Rem isto quer dizer que não deixe de ler os dados
Rem até que não se chegue ao último deles
Do While Not EOF(1)
Rem Lê um dado
Input #1, nome
Rem escreve na célula
ActiveCell.FormulaR1C1 = nome
Rem baixa um range para o seguinte nome
ActiveCell.Offset(1, 0).Select
Rem ativa o ciclo Do While - que regressa até
Rem que se cumpra a condição.
Loop
Rem fecha o arquivo
Close #1
```

```
End Sub
```

O que você acha de arquivar os dados em separado sem que ninguém possa observá-los? Esta é a magia dos arquivos seqüenciais.

Este exemplo vem indexado em um arquivo com o nome de **Macros VI**.

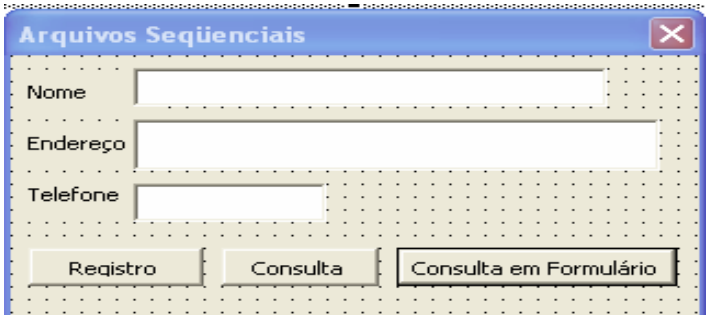
O seguinte código arquiva o **nome**, o **endereço** e o **telefone** no arquivo, cria um formulário igual com dois botões.

```
Private Sub CommandButton1_Click()
Rem trasladar à célula A8
Range("A8").Select
Rem se não há nenhum dado em A8 que não se arquive de novo
If ActiveCell = Empty Then GoTo salte
Rem abre um arquivo na unidad c com o nome de dados.txt
Rem em forma de añadir Temporal(output) no área de armazenamento
#1
Open "c:\dados.txt" For Output As 1
Rem ativa um rótulo para poder regresar
volte:
Rem captura o nome em uma variável
nome = ActiveCell
Rem apaga o dado da célula
ActiveCell = Empty
Rem move-se uma coluna à direita
ActiveCell.Offset(0, 1).Select
Rem captura o endereço em uma variável
endereço = ActiveCell
Rem apaga o dado da célula
ActiveCell = Empty
Rem move-se uma coluna à direita
ActiveCell.Offset(0, 1).Select
Rem captura o telefone em uma variável
telefone = ActiveCell
Rem apaga o dado da célula
ActiveCell = Empty
Rem escreva os dados nome, endereço e telefone no arquivo
Write #1, nome, endereço, telefone
Rem baixa um range para o seguinte nome
ActiveCell.Offset(1, 0).Select
Rem retrocede duas colunas
ActiveCell.Offset(0, -2).Select
Rem se a célula esta vazia que não regresse ainda
If ActiveCell = Empty Then GoTo salte
Rem volte a escrever o nome seguinte no arquivo
GoTo volte:
salte:
Rem se acabou
Rem fecha o arquivo
Close #1
End Sub

Private Sub CommandButton2_Click()
Rem transladar à célula A8
Range("A8").Select
Rem abre um arquivo na unidade c com o nome de dados.txt
```

```
Rem em forma de Ler (input) na área de armazenamento #1
Open "c:\dados.txt" For Input As 1
Rem isto significa laço embora não seja final do arquivo
Rem isto quer dizer que não deixe de ler os dados
Rem até que não se chegue ao último deles
Do While Not EOF(1)
Rem lê os dados
Input #1, nome, endereço, telefone
Rem escreva na célula o nome
ActiveCell.FormulaR1C1 = nome
Rem move-se uma coluna à direita
ActiveCell.Offset(0, 1).Select
Rem escreva na célula o endereço
ActiveCell.FormulaR1C1 = endereço
Rem move-se uma coluna à direita
ActiveCell.Offset(0, 1).Select
Rem escreva na célula o telefone
ActiveCell.FormulaR1C1 = telefone
Rem baixa um range para o seguinte nome
ActiveCell.Offset(1, 0).Select
Rem retrocede duas colunas
ActiveCell.Offset(0, -2).Select
Rem ativa o ciclo Do While - que regresse até
Rem que se cumpra a condição.
Loop
Rem fecha o arquivo
Close #1
End Sub
```

Este exemplo vem no arquivo **Macros VI-2.**

Também se pode consultar sem necessidade de ler os dados na folha, isto quer dizer lendo direto do arquivo e trazendo os dados ao formulário, no seguinte exemplo, se programa o botão consulta em formulário.



Desenhe o formulário acima, os dois primeiros botões é o mesmo código anterior, mas o terceiro botão inclui o seguinte código:

```
Private Sub CommandButton3_Click()
Open "c:\dados.txt" For Input As 1
Do While Not EOF(1)
Input #1, nome, endereço, telefone
If nome = TextBox1 Then
TextBox2 = endereço
```

```
TextBox3 = telefone
End If
Loop
Close #1
End Sub
```

Este exemplo vem no arquivo **Macros VI-3**

Somente rode o formulário e escreva o nome que deseja consultar e pressione o terceiro botão.

Você poderá consultar qualquer dos nomes que se encontrem dentro do arquivo, sem necessidade de que existam na folha, claro está que primeiro é necessário pressionar o botão registro para arquivá-los, mas depois se podem manipular.

Bem espero que seja de seu agrado esta parte e que pratiquen muito os arquivos seqüenciais.

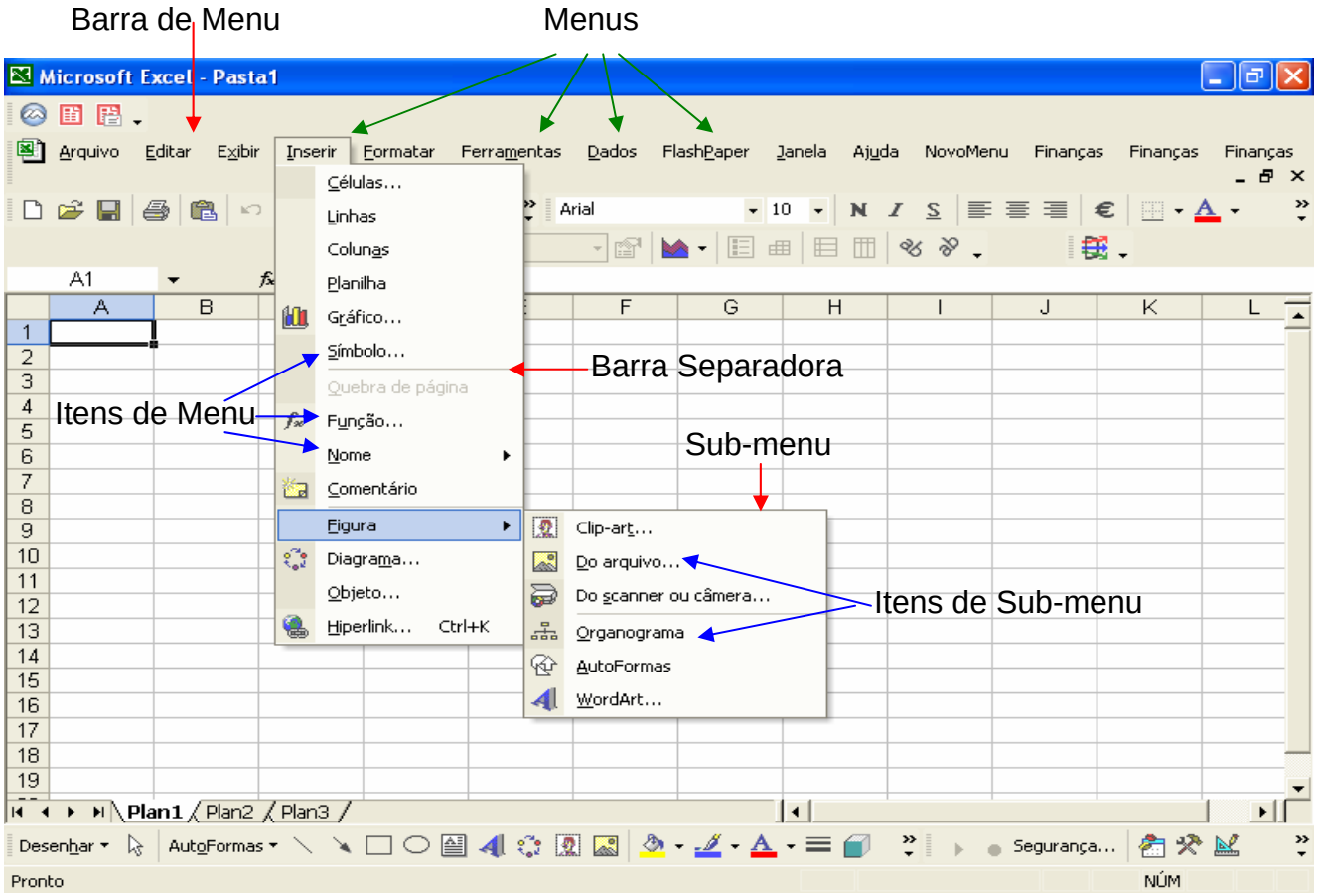
Lição 9: Criando e Modificando Menus

O Microsoft Excel tem mais de 30 menus que ajudam você no uso do Microsoft Excel. É possível aumentar ainda mais a barra de menus adicionando novos menus e itens de menu a eles. E, você pode adicionar itens de menus que aparecem somente quando um dado arquivo é aberto. De fato, você pode remover os menus Microsoft Excel e trocá-los com menus personalizados se você estiver criando uma aplicação personalizada no Microsoft Excel.

O que segue são os tópicos deste texto:

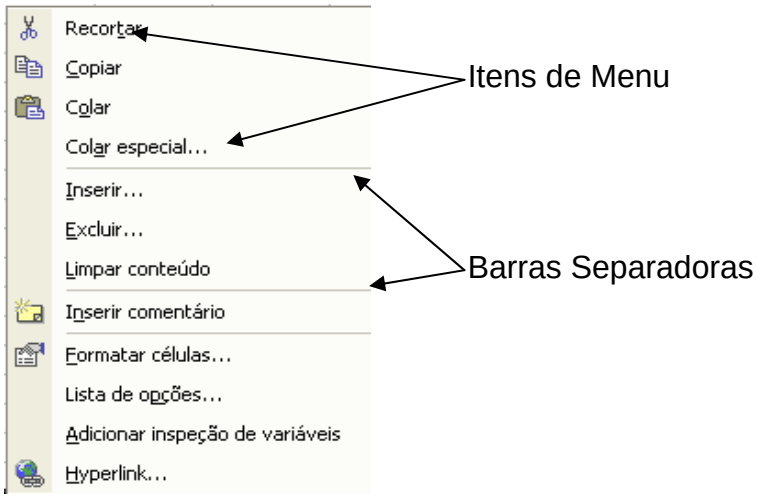
SOBRE OS MENUS EXCEL

Existem dois tipos básicos de sistema de menu do Microsoft Excel: Sistemas de menu que mostram uma barra de menu e menus *pop-up* (que não mostram uma barra de menu) que aparecem quando você clica no botão direito do mouse. O que segue ilustra os diferentes componentes de um sistema de menu do Microsoft Excel que tem na barra de menu:



A barra de menu mantém os nomes dos menus. Estes nomes são chamados *captions (títulos)*. Os menus em fila mantêm os nomes dos itens de menu. Um item de menu pode mostrar um sub-menu, rodar um comando Microsoft Excel ou rodar uma macro. Itens de sub-menus são usados para rodar comandos do Microsoft Excel ou macros. As barras separadoras atuam para separar visivelmente os menus em grupos lógicos.

O que segue ilustra um menu *pop-up* e seus componentes:



É possível fazer o seguinte a um sistema de menu do Microsoft Excel:

- Criar novos sistemas de menu que tenham sua própria barra de menus, menus, itens de menu, e sub-menus.
- Ocultar ou mostrar sistemas de menu que tenham barra de menus
- Deletar ou resetar (un-delete) sistemas de menu criados pelo usuário
- Adicionar ou delete menus numa barra de menu
- Adicionar ou deletar itens de menu num menu
- Habilitar e desabilitar itens num menu
- Colocar marcas de verificação para itens num menu
- Adicionar barras separadoras a menus
- Criar menus *pop-up*

O que segue ilustra itens de menu que foram adicionados ao menu Inserir da planilha. O >> que aparece na frente de cada adição era parte do texto que foi entrado para o item de menu:

FIGURA

USANDO MACROS PARA SE REFERIR AOS MENUS

O sistema de menu no Microsoft Excel que tem barra de menus são também partes da coleção CommandBars. Você também pode usar CommandBars("Worksheet Menu Bar") para referir ao sistema de menu da planilha. Os sistemas de barra de menu que

você criou pode ser referenciado pelo seu nome. Por exemplo, se você criou um chamado Especial, então você se referirá a ele como CommandBars("Especial").

Os menus de atalho, que não tem barra de menus, são parte da coleção commandbars. CommandBars("célula") faz referência ao menu *popup* célula.

AS COMMANDBARS

O que segue é a lista dos nomes commandbars:

Worksheet Menu Bar	Nondefault Drag e Drop	Order
Chart Menu Bar	AutoFill	Nudge
Standard	Button	Align ou Distribute
Formatting	Dialog	Rotate ou Flip
PivotTable	Series	Lines
Chart	Plot Area	Connectors
Reviewing	Floor e Walls	AutoShapes
Forms	Trendline	Callouts
Stop Recording	Chart	Flowchart
External Data	Formula Bar	Block Arrows
Auditing	PivotTable Context Menu	Stars & Banners
Full Screen	Query	Basic Shapes
Circular Reference	Query Layout	Shapes
Visual Basic	AutoCalculate Inactive	Chart
Web	Object/Plot	Excel Control
Control Toolbox	Title Bar (Charting)	Curve
Exit Design Mode	Layout	Curve Node
Drawing	WordArt	Curve Segment
Query e Pivot	Picture Pictures	Context
		Menu
Workbook tabs	Shadow Settings	OLE Object
Column	3-D Settings	ActiveX Control
Row	Borders	WordArt Context
		Menu
Cell	Chart Type	Rotate Mode
Ply	Pattern	Connector
XLM Cell	Font Color	Built-in Menus
Document	Fill Color	System
Desktop	Line Color	

Os nomes acima são usados para se referirem aos diferentes menus, barras de ferramentas e menus *pop-up*. Por exemplo,

```
CommandBars("Célula")
```

Refere-se ao menu *pop-up* célula. Infelizmente, nem todas as *commandbars* têm um título que aparece quando você mostrá-las, e você deve deduzir o nome da *commandbar* pela ação que ela fornece e da lista acima.

ADICIONANDO, MOSTRANDO E DELETANDO SISTEMA DE MENU

Para criar um sistema de menu, você poderá usar o método Adicionar. A sintaxe é:

```
CommandBars.Adicionar "nome do menu"
```

Por exemplo, o comando

```
CommandBars.Adicionar "Pessoal"
```

Adicionará um sistema de menu. Uma vez criado uma nova barra de menu, você pode então adicionar menus e itens de menu a ela. E, você pode também adicionar botões à barra de ferramentas, como discutido no texto sobre barras de ferramentas.

Para mostrar um sistema de menu personalizado que você criou, use a seguinte sintaxe:

```
CommandBars("nome domenu").Visible = True
```

Você tem a opção de trocar o sistema de menu ativo, ou fazer seu novo sistema de menu aparecer e não trocar o sistema ativo.

Para retornar aos menus da Microsoft Excel normais você deve ativar o menu padrão que será mostrado para a folha ativa ou então fechar o Microsoft Excel. Assim se a folha ativa for uma planilha, você poderá enviar o comando

```
MenuBars(xlWorksheet).Activate
```

Se a folha ativa for um gráfico, você poderá enviar o comando

```
MenuBars(xlChart).Activate.
```

Para deletar um sistema de menu que você adicionou, você poderá usar o seguinte tipo de declaração:

```
CommandBars("menu name").Delete
```

Por exemplo, se você criou um sistema de menu chamado Pessoal, então você poderá deletá-lo usando a declaração MenuBars("Pessoal").Delete.

MAIS SOBRE COMMANDBARS

Como mencionado anteriormente, os menus são realmente parte da coleção CommandBars. Isto basicamente significa que um commandbar é um recipiente que pode conter menus, ou botões, ou uma mistura dos dois.

O que segue é a sintaxe completa para se criar uma commandbar:

```
CommandBars.Add Name:="nome da barra de comando", _
Position:=vbconstant
MenuBar:= True ou False,
Temporary:= True ou False
```

Todos os argumentos são opcionais, e se não usados, os *defaults* serão usados.

O que segue são os valores do argumento *Position*:

Constante	Descrição
msoBarLeft, msoBarTop, msoBarRight, msoBarBottom	Indica as posições à esquerda, topo, à direita e fundo da nova barra de comando
msoBarFloating	Indica que a nova barra de comando será uma barra flutuante
msoBarPopup	Indica que a nova barra de comando será um menu de atalho

Se o argumento *MenuBar* é ajustado para *True*, então ele troca a commandbar ativa. Se o argumento *Temporary* é ajustado para True, então a commandbar é suposta removida quando o arquivo que a criou é fechado. Entretanto, a prática atual mostra que esta característica não funciona. Você deverá ter a sua macro para remover o menu quando fizer um.

ADICIONANDO UM NOVO SISTEMA DE MENU

O seguinte ilustra como adicionar um novo sistema de menu com um menu, itens de menu, e sub menus. Você pode modificar este exemplo quando necessário para criar e adicionar sua própria estrutura e adições de menu.

```
Sub Cria_Um_Novo_Sistema_De_Menu()
    Dim Meu_Menu As CommandBar, novoControle, novoItem, subMenu
   'remove menus personalizados se existirem
    On Error Resume Next
    CommandBars("Sistema Novo Menu").Delete
    On Error GoTo 0
```

```
    'cria e mostra um novo menu
    Set Meu_Menu = CommandBars.Adicionar(Name:="Sistema Novo
Menu", _ Position:=msoBarFloating, _ MenuBar:=False)
    Meu_Menu.Visible = True
    'adiciona um menu à nova CommandBar
    Set novoControle = _ Meu_Menu
      .Controls.Adicionar(Type:=msoControlPopup)
          novoControle.Caption = "Menu1"
           'adicionar um item de menu ao novo menu
          With novoControle
              Set novoItem =
      .Controls.Adicionar(Type:=msoControlButton)
              Set subMenu =
      .Controls.Adicionar(Type:=msoControlPopup)
          End With
          With novoItem
               .Caption = "Isto Diz Hello"
               .OnAction = "Alô"
          End With
            'adicionar um sub menu ao novo menu e adicionar itens
     a ele
          With subMenu
                .Caption = "Escolhas Adicionais"
               Set novoItem =
      .Controls.Adicionar(Type:=msoControlButton)
               novoItem.Caption = "Examinar Pescaria"
               novoItem.OnAction = "FishingStatus"
                Set novoItem = _
                  .Controls.Adicionar(Type:=msoControlButton)
               novoItem.Caption = "Examinar Jogo de Golfe"
               novoItem.OnAction = "GolfingStatus"
          End With
           'adicionar um item de menu que restaurará os menus
     originais
          Set novoItem = _
             novoControle
.Controls.Adicionar(Type:=msoControlButton)
    With novoItem
          .Caption = "Remove the Sistema Novo Menu"
          .OnAction = "RemoveCustomMenu"
           'Esta próxima declaração adiciona uma barra separadora
          .BeginGroup = True
    End With
    'adiciona um menu à nova CommandBar
    Set novoControle = _
           Meu_Menu
.Controls.Adicionar(Type:=msoControlPopup)
    novoControle.Caption = "Menu2"
    Set novoItem = _
           novoControle
.Controls.Adicionar(Type:=msoControlButton)
    With novoItem
```

```
        .Caption = "Diga Tchau"
        .OnAction = "DigaTchau"
    End With
End Sub

Sub SayHello()
     MsgBox "Alô Mundo"
End Sub

Sub SayGoodBye()
   MsgBox "Tchau!"
End Sub

Sub fishingStatus()
     MsgBox "Pescar é maravilhoso em todo momento!!!"
End Sub

Sub golfingStatus()
     MsgBox "Quem se importa? Seria melhor estar pescando!"
End Sub

Sub RemoveCustomMenu()
     CommandBars("Sistema Novo Menu").Visible = False
End Sub
```

Pontos chaves sobre a adição de menus, itens de menu, e sub menus:

- Itens de menu são adicionados usando um *Type* de msoControlButton
- Menus e sub-menus são adicionados usando um *Type* de msoControlPopup quando o método Adicionar é usado com o método *Controls*.
- O texto de menu é ajustado pela propriedade *Caption*.
- Uma macro é atribuída a um item de menu usando a propriedade *OnAction*.
- Ajustando a propriedade *Visible* da commandbar para False, a commandbar fica oculta. Para deletar a commandbar, use o método Delete. Você poderá ocultar uma commandbar se você quiser torná-la visível mais tarde.
- Ajustando a propriedade *BeginGroup* de um item de menu para True adiciona uma barra separadora acima do item de menu.

UM EXEMPLO DE ADIÇÃO/REMOÇÃO DE ITEM DE MENU

O que segue são duas macros que adicionam e removem um item de menu com uma barra separadora do menu Ferramentas do Excel.

A primeira macro é chamada "Auto_Open" e adiciona o item de menu quando o arquivo for aberto. A segunda é chamada "Auto_Close" e remove o item de menu quando o arquivo for fechado. As macros com os nomes Auto_Open e Auto_Close são rodadas automaticamente pelo Excel. Elas serão cobertas em mais detalhes num texto posterior. Para evitar as macros de rodarem quando o arquivo está aberto ou fechado, mantenha pressionada a tecla *shift* quando o arquivo é aberto ou fechado.

```
Sub Auto_Open()
    'isto adiciona a item de menu ao menu Ferramentas da
planilha quando o arquivo é aberto
    Dim mItem, novoItem
    With CommandBars("Worksheet Menu
_Bar").Controls("Ferramentas")
        For Each mItem In .Controls
          'se o item de menu já está presente, pule a mensagem
          If mItem.Caption = "NOVO ITEM DE MENU" Then _
                    GoTo displayMessage
        Next
        'adiciona um novo item de menu
        Set novoItem =
.Controls.Adicionar(Type:=msoControlButton)
    End With
    'ajusta caption, atribui macro, adiciona barra separadora
    With novoItem
        .Caption = "NOVO ITEM DE MENU"
        .OnAction = "Calcular_TIR"
        .BeginGroup = True
    End With
    'mostra a caixa de mensagem dizendo ao usuário como usar
displayMessage:
   MsgBox "Para usar a NOVO ITEM DE MENU, selecione 'NOVO ITEM
DE MENU' " & _
          "sob o menu Ferramentas"
End Sub

Sub Auto_Close()
   'isto remove o item de menu adicionado pela macro acima
quando o arquivo for fechado
   Dim mItem, I As Integer
   With CommandBars("Worksheet Menu Bar").Controls("Tools")
       For Each mItem In .Controls
          I = I + 1
          'verifique o item de menu; delete se encontrado
          If mItem.Caption = "NOVO ITEM DE MENU" Then
            .Controls(I).Delete
            'saia do laço quando o item for removido
            Exit For
          End If
       Next
```

```
    End With
End Sub
```

PROPRIEDADES e APARÊNCIA do ITEM DE MENU

Um item de menu tem três propriedades que você pode mudar após o item de menu ter sido adicionado a um menu. Elas são:

- *Caption*
- *Enabled*
- *State* (usado para adicionar ou remover marcas de verificação pelo item de menu)

As propriedades *Caption* permitem-lhe entrar com um texto de descrição diferente para o item de menu. A propriedade *State* pode ser ajustada para msoButtonDown para mostrar um *checkmark* (visto) pelo item. Se ajustado para msoButtonUp, que é o default, não há checkmark. Se a propriedade *Enabled* é ajustada para True, que é o default, o item de menu rodará a macro associada com o item. Se a propriedade *Enabled* é ajustada para False, então o item de menu aparecerá prateado e não rodará a macro associada.

Por exemplo, o que segue adiciona dois itens de menu ao menu Data. Ela também desabilita o Segundo item de menu de modo que somente o primeiro pode ser rodado:

```
Sub Menu_Item_Exemplo()
      Dim oMenu, novoItem
     'adiciona o primeiro item de menu ao menu data
     With CommandBars("Worksheet Menu Bar").Controls("Data")
         Set novoItem =
.Controls.Adicionar(Type:=msoControlButton)
     End With
    'atribui nome e macro ao item de menu.  Também, coloca uma
    'linha separadora acima dele ajustando o BeginGroup para
True
     With novoItem
         .Caption = "Load Data - Step 1"
         .OnAction = "Load_Data_Step1"
         .BeginGroup = True
     End With
    'adiciona um segundo item de menu, mas desabailita-o
     With CommandBars("Worksheet Menu Bar").Controls("Dados")
         Set novoItem =
.Controls.Adicionar(Type:=msoControlButton)
     End With
     With novoItem
         .Caption = "Load Data - Step 2"
         .OnAction = "Load_Data_Step2"
         .Enabled = False
     End With
```

```
End Sub
```

Você pode então usar as declarações seguintes na macro Load_Data_Step1 chamada pelo primeiro item de menu para colocar um *check mark* no seu item de menu, desabilitar o item de menu, e habilitar o passo 2 do item de menu:

```
  With CommandBars("Worksheet Menu Bar").Controls("Data")
            .Controls("Load Data - Step 1").Enabled = False
            .Controls("Load Data - Step 1").State =
msoButtonDown
            .Controls("Load Data - Step 2").Enabled = True
     End With
```

Se você precisar fazer a propriedade *Enabled* mudar dependendo de qual folha está ativa, então você pode criar uma macro que roda cada vez uma folha diferente seja ativada. Esta macro poderá então ser ajustada nas propriedades *Checked* e *Enabled* dos itens de menu após ela determinar qual folha está ativa. A sintaxe par este comando é:

```
Application.OnSheetActivate = "nome da macro"
```

A macro especificada será também rodada se você trocar de uma pasta para outra. Para desabilitar este comando, ajuste-o igual a uma string vazia, (duas aspas duplas, "")

```
Application.OnSheetActivate = ""
```

ITENS DE MENUS ESPECIAIS

Você colocará três novos itens de menu ou botões na sua barra de ferramentas ou menus. Estes novos itens são uma caixa de edição, uma caixa de lista *drop-down*, e a lista de edição *drop-down* combinada. Elas comportam-se exatamente como os mesmos itens sobre um formulário de usuário (*userform*). Por exemplo, você pode atribuir valores, ajustar macros OnAction, e você pode pedir pelos ajustes e conteúdos. Isto é ilustrado melhor no seguinte exemplo de macros:

```
Sub New_Menu_Item_Exemplo()
Dim newBar, newMenu, novoItem
'remove a antiga command bar se existir
On Error Resume Next
CommandBars("Exemplo Novo Menu").Delete
On Error GoTo 0
'cria nova barra
Set newBar = CommandBars.Add("Exemplo Novo Menu")
newBar.Visible = True
'Adicionar um item de menu a ela
Set newMenu = newBar.Controls.Add(Type:=msoControlPopup)
newMenu.Caption = "Exemplo Novo Menu"
'Adicionar um item de menu caixa de edição ao menu e atribuir
uma macro OnAction
```

```
Set novoItem = newMenu.Controls.Add(Type:= _
msoControlEdit)
novoItem.OnAction = "Display_Edit_Box_Entry"
novoItem.Caption = "Entre com a Descrição: "
'Adicionar um item de menu dropdown, atribuir valores, e uma
macro OnAction
Set novoItem = newMenu.Controls.Add(Type:=msoControlDropdown)
With novoItem
.Caption = "Selecione Um Item"
.OnAction = "Display_Drop_Down_Selection"
.AddItem Text:="Primeiro Item", Index:=1
.AddItem Text:="Segundo Item", Index:=2
.AddItem Text:="Terceiro Item", Index:=3
.AddItem Text:="Quarto Item", Index:=4
.AddItem Text:="Quinto Item", Index:=5
.AddItem Text:="Sexto Item", Index:=6
.DropDownLines = 5
.DropDownWidth = 75
.ListHeaderCount = 2
End With
    'Adicionar um item de menu combo edição drop-down, 'atribuir
    valores, e uma macro OnAction
    Set novoItem = newMenu.Controls.Add(Type:= _
    msoControlComboBox)
    With novoItem
    .Caption = "Selecionar ou Entrar com Um Mês"
    .OnAction = "Display_Combo_Box_Selection"
    .AddItem Text:="Janeiro", Index:=1
    .AddItem Text:="Fevereiro", Index:=2
    .AddItem Text:="Março", Index:=3
    .AddItem Text:="Abril", Index:=4
    .AddItem Text:="Maio", Index:=5
    .AddItem Text:="Junho", Index:=6
    .DropDownLines = 5
    .DropDownWidth = 75
    .ListHeaderCount = 0
    End With
End Sub

Sub Display_Edit_Box_Entry()
   'mostra os conteúdos da caixa de edição
    MsgBox CommandBars("Novo menu Exemplos") _
             .Controls(1).Controls(1).Text
End Sub

Sub Display_Drop_Down_Selection()
    'mostra o conteúdo da caixa drop down
     With CommandBars("Novo menu Exemplos") _
                    .Controls(1).Controls(2)
        MsgBox "Item " & .ListIndex & " selecionado.  " & _
           "Seu texto é: " & .List(.ListIndex)
     End With
```

```
End Sub

Sub Display_Combo_Box_Selection()
    'mostra os conteúdos da caixa de edição drop-down
     With CommandBars("ExemploNovo Menu") _
                    .Controls(1).Controls(3)
        If .ListIndex = 0 Then
            MsgBox .Text
        Else
            MsgBox "Item " & .ListIndex & " selecionado.  " & _
                   "Seu texto é: " & .List(.ListIndex)
        End If
     End With
End Sub
```

COMO OCULTAR TODAS AS COISAS - MENUS, BARRAS DE ROLAMENTO, ETC..

Menus são partes da coleção commandbar. Você pode ajustar a propriedade *Enabled* de uma commandbar para False par ocultá-la. Se o menu que é mostrado é a Worksheet Menu Bar, o que segue a ocultará:

```
CommandBars("Worksheet Menu Bar")
.Enabled = False
```

Ou você pode usar um comando **For..Next** e laçar todas as commandbars e ajustar a propriedade *Enabled* para False, ocultando ambos os menus e barras de ferramentas. Ajustando a propriedade *Enabled* para True mostrará quaisquer menus e commandbars que eram visíveis inicialmente.

Outras ações que você pode querer que sua macro faça são:

- Ocultar a barra de fórmula
- Ocultar a barra de status
- Ocultar as alças das folhas
- Ocultar as barras de rolamento
- Ocultar os cabeçalhos de linhas e colunas
- Ajustar a visão de tela cheia para ocultar a barra de títulos da aplicação, ou atribuir seu próprio nome à barra (Application.Caption é a propriedade)

Se você ajustou a Application.DisplayFullScreen para True, você poderá também ajustar ThisWorkbook.Protect Windows para True para eliminar o botão minimizar, maximizar, restaurar e a linha que os mantém.

Se você ocultar estes itens, você precisa armazenar os ajustes originais de modo que você possa restaurar quando você quiser. É melhor armazenar os ajustes numa planilha (como é feito com as macros em ocultar as barras de ferramentas). Este modo, se uma de suas macros encontrar uma declaração End, os ajustes não serão perdidos. Também,

ele faz a depuração e edição de seu código mais facilmente, desde que a edição reajusta os valores de quaisquer variáveis do Visual Basic usadas para armazenar ajustes.

Você também pode querer ocultar todas as linhas e colunas que você não quer que o usuário tenha acesso. E, você pode querer ocultar todos os itens no menu *pop-up* célula e desabilitar todas as combinações de teclas CTRL atribuindo-as a uma macro muda que não faça nada.

O que segue ilustra os conceitos acima, e usa as macros para ocultar barra de ferramentas discutidas num texto anterior. Por favor ela deve ser rodada de uma planilha ou um erro ocorrerá.

```
Sub DemoRun()
   StoreOptionSettings
   HideOptionItems
   HideCommandbars
   Application.ScreenUpdating = True
   MsgBox "Está Tudo Oculto"
   ShowCommandbars
   ShowOptionItems
End Sub

Sub StoreOptionSettings()
     With ActiveWindow
         Cells(1, 2) = .DisplayHorizontalScrollBar
         Cells(2, 2) = .DisplayVerticalScrollBar
         Cells(3, 2) = .DisplayWorkbookTabs
     End With
     With Application
         Cells(4, 2) = .DisplayFormulaBar
         Cells(5, 2) = .DisplayStatusBar
     End With
     Cells(6, 2) = ActiveWindow.DisplayHeadings
End Sub

Sub HideOptionItems()
    Application.ScreenUpdating = False
    ActiveWindow.WindowState = xlMaximized
    ThisWorkbook.Protect Windows:=True
    With ActiveWindow
        .DisplayHorizontalScrollBar = False
        .DisplayVerticalScrollBar = False
        .DisplayWorkbookTabs = False
    End With
    With Application
        .DisplayFormulaBar = False
        .DisplayStatusBar = False
    End With
    ActiveWindow.DisplayHeadings = False
End Sub
```

```
Sub ShowOptionItems()
    ActiveWindow.WindowState = xlMaximized
    ThisWorkbook.Protect Windows:=False
    With ActiveWindow
        .DisplayHorizontalScrollBar = Cells(1, 2)
        .DisplayVerticalScrollBar = Cells(2, 2)
        .DisplayWorkbookTabs = Cells(3, 2)
    End With
    With Application
        .DisplayFormulaBar = Cells(4, 2)
        .DisplayStatusBar = Cells(5, 2)
    End With
    ActiveWindow.DisplayHeadings = Cells(6, 2)
End Sub

Sub HideCommandbars()
    Dim cBar As CommandBar
    For Each cBar In CommandBars
         cBar.Enabled = False
    Next
    Application.DisplayFormulaBar = False
End Sub

Sub ShowCommandbars()
    Dim cBar As CommandBar
    For Each cBar In CommandBars
         cBar.Enabled = True
    Next
End Sub
```

SUMÁRIO

Embora este texto forneceu alguns comandos muito complexos , você precisa so-mente usar aqueles que vão de encontro às suas necessidades. Por exemplo, freqüentemente eu uso apenas o editor de menu editor para adicionar um novo item de menu ao menu. Desde que eu use o editor de menu, este novo item aparece somente quando a pasta em que ele foi criado é aberta.

Exemplos de CommandBar e Menus

Usando Diálogos Built-In do Excel

Microsoft Excel tem mais de 200 diálogos embutidos (built-in) que você pode usar em suas macros. Para mostrar um diálogo embutido (built-in), você usará uma declaração como a seguinte:

```
        Application.Dialogs(vb constant).Show
ou      resposta = Application.Dialogs(vb constant).Show
```

Por exemplo, o que segue mostra o diálogo de seleção de impressora embutida:

```
        Application.Dialogs(xlDialogPrint).Show
ou      resposta = Application.Dialogs(xlDialogPrint).Show
```

A primeira declaração apenas mostra a caixa. A segunda declaração mostra a caixa e determina à variável resposta o valor True se o botão OK foi selecionado, e o valor False se o botão Cancel foi selecionado. O que está acima não somente mostra a caixa de diálogo, mas emprega a ação que a caixa foi planejada a fazer se OK é selecionado na caixa. Por favor note que nem todas as constantes "xlDialog…" mostrarão um diálogo embutido.

Ambas declarações usam uma constante Visual Basic para indicar qual caixa de diálogo mostrar. As constantes do Visual Basic que iniciam com "xlDialog" são aquelas para se usarem.

Por favor note que nem todas as constantes "xlDialog..." mostrarão um diálogo embutido. Para ver uma lista das constantes, faça o seguinte:

- Mostre o Object Browser, selecione "<All Librarys>". Na lista Classes, selecione "xlBuiltInDialog". Uma lista das constantes aparecerá na caixa de listagem membros.

Por exemplo, se você quiser que a caixa de diálogo apareça quando você gravar um novo arquivo, você poderá usar a seguinte declaração:

```
Application.Dialogs(xlDialogOpen).Show
```

Para mostrar o diálogo Salvar Como e garantir que o seu uso salvou o arquivo, você poderá usar as seguintes declarações:

```
Do
  resposta = Application.Dialogs(xlDialogSaveAs).Show
Loop Until resposta = True
```

Exemplos Adicionais do uso de Salvar Como e diálogos embutidos DialogOpen são encontrados na seção arquivos.

O que segue é uma lista de apenas umas poucas constantes do Visual Basic e a caixa de diálogos que elas mostram.

Constante	Caixa de Diálogo
xlDialogOpen	A caixa para abrir arquivo
xlDialogSaveAs	A caixa Salvar Como
xlDialogSetPrintTitles	A caixa set print titles (do Excel 4)
xlDialogChartWizard	O assistente (wizard) de diagramas
xlDialogCreateNames	A caixa para criar nomes
xlDialogFont	A caixa fonte de célula
xlDialogGoalSeek	A caixa goal seek
xlDialogUnhide	A caixa para ocultar um arquivo
xlDialogZoom	A caixa de zoom

Um cuidado ao se usar a caixa de diálogos Microsoft Excel: Se você tentar mostrar uma caixa de diálogo embutido numa folha onde ela não pode ser usada, ela falhará. Por exemplo, a caixa zoom não pode ser usada numa folha de módulo. Mas ela funciona bem se a folha ativa é uma planilha ou uma folha de diagramas.

CommandBar.Add Produz Err 91 na Workbook_Open

Quando se referir ao objeto CommandBars num procedimento evento tal como o código abrir pasta num módulo de objeto pasta, você precisa precedê-lo pelo Application:

```
Set cbarMine = Application.CommandBars _
           .Add("My Toolbar", msoBarTop, True, True)
```

se você não fizer, você pode obter uma mensagem Err 91.

Adicionando Um Ítem de Menu A Um Menu

O que segue são quatro macros que adicionam e removem um item de menu com uma barra separadora de um menu. A primeira é chamada "Auto_Open" e chama uma rotina chamada AddMenuItem que adiciona um novo item de menu a um menu nos menus de planilha. A segunda é chamada "Auto_Close" e chama uma rotina chamada RemoveMenuAddition que remove o item de menu quando o arquivo é fechado. Você pode chamar AddMenuItem e RemoveMenuAddition repetidamente para adicionar múltiplos ítens de menus. Macros com os nomes Auto_Open e Auto_Close, são rodadas automaticamente pelo Excel, quando um arquivo é aberto ou fechado.

```
Sub Auto_Open()
```

```
        'Chama a rotina que adiciona o item, especificando o menu da planilha para
        ' o novo item de menu, o nome a aparecer no menu, a macro a ser rodada,
        'e se adiciona uma barra separadora acima ou não ao novo item
        AddMenuItem "Ferramentas", "Converter Arquivos", _
              "Main_Procedure_For_Converting_Files", True
End Sub

Sub Auto_Close()
   'Chama a rotina que remove um item de menu do menu da planilha
   'especificando o nome do menu e o nome do item de menu.
      RemoveMenuAddition "Ferramentas", "Converter Arquivos"
End Sub

Sub AddMenuItem(menuName As String, itemName As String, _
      macroName As String, bAddSeperator As Boolean)
      Dim mItem, newItem
      'isto remove item se ele estiver num menu
      RemoveMenuAddition menuName, itemName
      'adiciona novo item de menu; setting uma propriedade temporária para
      ' verdadeiro (true) garantindo que o item de menu desaparece quando o Excel
      ‘é  fechado
      With CommandBars("Worksheet Menu Bar").Controls(menuName)
            Set newItem = .Controls.Add(Type:=msoControlButton, _
            temporary:=True)
      End With
      'configura o caption, associa uma macro, adiciona barra separadora
      With newItem
            .Caption = itemName
            .OnAction = macroName
            .BeginGroup = bAddSeperator
      End With
End Sub

Sub RemoveMenuAddition(menuName As String, itemName As String)
      Dim mItem, I As Integer
      'isto remove o item de menu adicionado pela macro acima quando o arquivo
      ’é fechado
      With CommandBars("Worksheet Menu Bar").Controls(menuName)
            For Each mItem In .Controls
                  I = I + 1
                  'verifica um item de menu; delete se encontra
                  If mItem.Caption = itemName Then
                  .Controls(I).Delete
                        'sai do laço quando o item tiver sido removido
                        Exit For
                  End If
            Next
      End With
End Sub
```

Adicionando Um Menu e Sub Menus ao Menu Da Planilha

O que segue ilustra como adicionar um novo menu ao menu de planilha, e daí então como adicionar item de menus e sub menus ao novo menu. Por simplicidade, todos os ítens de menus que tem uma propriedade OnAction são configurados para rodar a mesma macro, "AlôMundo". Na sua aplicação, você pode usar diferentes OnActions.

```
Sub AddingMenusAndSubMenus()
    Dim c
  'deleta o novo menu se ele existir - chama todas as sub-rotinas listadas abaixo
  Remove_New_Menu
  'adiciona o novo menu ao menu da planilha antes do menu ajuda
  With Application.CommandBars(1).Controls. _
                   Add(msoControlPopup, , , 9, True)
    .Caption = "MeuMenu"
  'adiciona um item de menu ao novo menu
    Set c = .Controls.Add(msoControlButton)
c.Caption = "Item 1"
    c.OnAction = "AlôMundo"
  'adiciona um sub menu ao novo menu
  With .Controls.Add(msoControlPopup)
        .Caption = "Item 2"
  'adiciona um item de menu ao menu sub
    Set c = .Controls.Add(msoControlButton)
        c.Caption = "Sub 1 Item 1"
        c.OnAction = "AlôMundo"
  'adiciona um sub menu ao menu sub
    With .Controls.Add(msoControlPopup)
.Caption = "Sub 1 Item 2"
    'adiciona três itens de menu ao sub menu mais baixo
      Set c = .Controls.Add(msoControlButton)
            c.Caption = "Sub 2 Item 1"
            c.OnAction = "AlôMundo"
            Set c = .Controls.Add(msoControlButton)
            c.Caption = "Sub 2 Item 2"
            c.OnAction = "AlôMundo"
            Set c = .Controls.Add(msoControlButton)
            c.Caption = "Sub 2 Item 3"
      c.OnAction = "Alô Mundo"
         End With
    'adiciona a menu item to first sub menu
    Set c = .Controls.Add(msoControlButton)
        c.Caption = "Sub 1 Item 3"
        c.OnAction = "AlôMundo"
    End With
  'adiciona a menu item to the menu
  Set c = .Controls.Add(msoControlButton)
    c.Caption = "Item 3"
    c.OnAction = "AlôMundo"
```

```
End With
End Sub

Sub AlôMundo()
MsgBox "alô mundo"
End Sub
```

Quando a pasta é fechada, o que segue roda e remove o novo menu.

```
Sub Auto_Close()
     Remove_New_Menu
End Sub

Sub Remove_New_Menu()
     Dim c
     'deleta o novo o novo menu se ele existir
     For Each c In Application.CommandBars(1).Controls
          If c.Caption = "MeuMenu" Then c.Delete
     Next
End Sub
```

Como Adicionar Uma Nova Barra de Menu Semelhante À Barra de Menu da Planilha

O que segue adiciona uma barra de menu como a barra de menu da planilha, exceto que os menus são chamados Menu1 e Menu2.

```
Sub Create_A_New_Menu_System()
    Dim My_Menu As CommandBar, newControl, newItem, subMenu
        'remove menus personalizados se existirem
  On Error Resume Next
    CommandBars("New Menu System").Delete
    On Error GoTo 0
        'cria um novo menu e mostra-o
  Set My_Menu = CommandBars.Add(Name:="New Menu System", _
            Position:=msoBarTop, _
            MenuBar:=False)
    My_Menu.Visible = True
     'adiciona um menu à nova CommandBar
  Set newControl = My_Menu.Controls.Add(Type:=msoControlPopup)
    newControl.Caption = "Menu1"
     'adiciona um item de menu ao novo menu
  With newControl
        Set newItem = .Controls.Add(Type:=msoControlButton)
        Set subMenu = .Controls.Add(Type:=msoControlPopup)
    End With
        With newItem
        .Caption = "Isto diz Alô"
         .OnAction = "SayHello"
```

```
    End With
    'adiciona um sub menu ao novo menu e adiciona items a ele
  With subMenu
        .Caption = "Escolhas Adicionais"
        Set newItem = .Controls.Add(Type:=msoControlButton)
        newItem.Caption = "Conferir a Pesca"
        newItem.OnAction = "FishingStatus"
        Set newItem = _
                .Controls.Add(Type:=msoControlButton)
        newItem.Caption = "Conferir o Jogo de Golfe"
       newItem.OnAction = "GolfingStatus"
    End With
    'adiciona um item de menu que restaura os menus originais
  Set newItem = _
            newControl.Controls.Add(Type:=msoControlButton)
    With newItem
        .Caption = "Remover o novo sistema de menu"
        .OnAction = "RemoveCustomMenu"
        'Esta próxima declaração adiciona uma barra separadora
        .BeginGroup = True
    End With
    'adiciona um menu à nova CommandBar
Set newControl = My_Menu.Controls.Add(Type:=msoControlPopup)
    newControl.Caption = "Menu2"
    Set newItem =
newControl.Controls.Add(Type:=msoControlButton)
    With newItem
        .Caption = "Diz Tchau"
        .OnAction = "DizTchau"
    End With
End Sub
Sub SayHello()
     MsgBox "Alô Mundo"
End Sub

Sub DizTchau()
     MsgBox "Tchau!"
End Sub

Sub fishingStatus()
     MsgBox "Pescar é maravilhoso o tempo todo!!!"
End Sub

Sub golfingStatus()
      MsgBox "Quem se importa? Seria melhor pescar!"
End Sub

Sub RemoveCustomMenu()
     CommandBars("New Menu System").Delete
End Sub
```

Botão Como Controle Sobre Um Menu

O que segue cria uma barra de comando flutuante que tem botão como item de menu além de um menu de item *drop down*:

```
Sub Floating_New_Menu_System()
      Dim newMenu As CommandBar, newControl, newItem, subMenu
            'remove menu personalizado se ele existir
      On Error Resume Next
      CommandBars("New Menu System").Delete
      On Error GoTo 0
            'cria novo menu e mostra-o
      Set newMenu = CommandBars.Add(Name:="New Menu System", _
                  Temporary:=True, _
                  MenuBar:=False)
            newMenu.Visible = True
            'adiciona um menu ao novo CommandBar
      Set newControl =
newMenu.Controls.Add(Type:=msoControlPopup)
            newControl.Caption = "Menu1"
            'adiciona um item de menu ao novo menu
      With newControl
            Set newItem = .Controls.Add(Type:=msoControlButton)
      End With
            With newItem
            .Caption = "This Says Hello"
            .OnAction = "SayHello"
      End With
            'adiciona um segundo controle que atua como um botão
      Set newItem = newMenu.Controls.Add( _
                  Type:=msoControlButton, Temporary:=True)
      With newItem
            .Caption = "Control Text"
            .Style = msoButtonCaption
            .TooltipText = "Control Tool Tip"
            .OnAction = "SayHello"
      End With
End Sub

Sub SayHello()
      MsgBox "Hello"
End Sub
```

Ocultando O Menu Da Planilha

Você pode ocultar o menu de planilha Excel usando a seguinte declaração

```
Application.CommandBars(1).Enabled = False
```

Entretanto, se você pressionar a tecla ALT e daí as teclas setas, o menu é mostrado novamente. Existe um modo fácil para resolver o acesso às teclas ALT para desabilitar menus:

```
Application.CommandBars(1).Enabled = False
MenuBars.Add.Activate
```

A segunda linha evita a tecla ALT de funcionar criando e ativando uma barra de menu em branco. Como ela não tem menus, ela não é visível ou acessível.

Para recuperar os menus, use as declarações seguintes:

```
Application.CommandBars(1).Enabled = True
MenuBars(xlWorksheet).Activate
```

Colocando Um DropDown Numa Barra de Comando (CommandBar)

O que segue cria uma barra de comando com um dropdown nela e responde com a seleção do usuário.

```
Sub CommandBarDemo()
      Dim cBar As CommandBar
      Dim I As Integer
            'delete a barra se ela existir
      On Error Resume Next
      CommandBars("Combo Bar").Delete
      On Error GoTo 0
      'cria a barra de comando e a torna visível
      Set cBar = CommandBars.Add("Combo Bar", msoBarFloating)
      cBar.Visible = True
      'adiciona um controle dropdown
      With cBar.Controls.Add(msoControlDropdown)
      'atribui uma macro ao drop down
            .OnAction = ThisWorkbook.Name & "!DropDownOnAction"
            'make wider:
            .Width = 200
            'adiciona items à caixa drop down
            For I = 1 To 10
                  .AddItem "item drop down " & I
            Next
      End With
End Sub

Sub DropDownOnAction()
      'isto mostra o que foi selecionado
      With CommandBars.ActionControl
```

```
                MsgBox "Você selecionou " & .Text
      End With
End Sub
```

Criando Um Menu Que Aparece Somente Quando Uma Pasta Particular Está Ativa

O que segue code ilustra como ter um único menu adicionado ao menu de planilha sempre que uma pasta particular está ativa. Para fazer isto, coloque o código seguinte no módulo de código da pasta:

```
Private Sub Workbook_Deactivate()
      'chama a rotina que remove o menu para esta workbook
      Remove_Workbook_Menu
End Sub

Private Sub Workbook_Activate()
      'chama a rotina que adiciona o menu para esta workbook
      Add_Workbook_Menu_And_Items
End Sub
```

Você pode acessar o módulo de código da workbook clicando o botão direito sobre o objeto workbook no Project Explorer e selecionando exibir código

Num módulo de código regular, coloque o seguinte código:

```
Sub Remove_Workbook_Menu()
      'isto remove o menu se ele estiver presente
      On Error Resume Next
      CommandBars("Worksheet Menu
Bar").Controls("OPTIONS").Delete
      On Error GoTo 0
End Sub

Sub Add_Workbook_Menu_And_Items()
      Dim newMenu
      Dim newMenuItem
      'deleta o menu se ele existir chamando esta subroutina
      Remove_Workbook_Menu
      'adiciona um novo menu ao menu de planilha.  O menu é temporário e
      'desaparecerá quando o Excel fechar
      With CommandBars("Worksheet Menu Bar")
            Set newMenu = .Controls.Add( _
                        Type:=msoControlPopup, _
                        before:=.Controls("Window").Index, _
                        temporary:=True)
      End With
      'dá um nome ao novo menu
      newMenu.Caption = "OPTIONS"
```

```
      'adiciona um item de menu ao novo menu
      Set newMenuItem =
      newMenu.Controls.Add(Type:=msoControlButton)
      'dá um nome ao novo menu e atribui uma macro a ele
      newMenuItem.Caption = "Menu Item 1"
      newMenuItem.OnAction = "MenuItem1OnAction"
      'adiciona um segundo item de menu ao novo menu
      Set newMenuItem =
newMenu.Controls.Add(Type:=msoControlButton)
      'dá um nome ao novo menu e atribui uma macro a ele
      newMenuItem.Caption = "menu Item 2"
      newMenuItem.OnAction = "MenuItem2Onaction"
End Sub
```

Finalmente, cria uma macro Auto_Close que chama a macro Remove_Menu quando a pasta é fechada e uma macro Auto_Open que mostra o menu quando a pasta é aberta inicialmente.

```
Sub Auto_Close()
      Remove_Workbook_Menu
End Sub

Sub Auto_Close()
      Add_Workbook_Menu_And_Items
End Sub
```

Adicionando Um Menu E Itens de Menu Ao Menu da Planilha

O que segue adiciona um menu no final do menu da planilha, e daí adiciona itens de menu e sub menus a ele.

```
Sub AddMenu()
      With Application.CommandBars(1).Controls _
      .Add(msoControlPopup)
            .Caption = "NovoMenu"
            .Controls.Add(msoControlButton)
            .Caption = "Item 1"
            With .Controls.Add(msoControlPopup)
                  .Caption = "Item 2"
                  .Controls.Add(msoControlButton)
                  .Caption = "Sub1Item2"
                  With .Controls.Add(msoControlPopup)
                  .Caption = "Sub2Item2"
                        .Controls.Add(msoControlButton) _
                              .Caption = "Sub1Sub2"
                        .Controls.Add(msoControlButton) _
                              .Caption = "Sub2Sub2"
                        .Controls.Add(msoControlButton) _
                              .Caption = "Sub3Sub2"
                  End With
```

```
                .Controls.Add(msoControlButton)
                     .Caption = "Sub3Item2"
          End With
          .Controls.Add(msoControlButton)
          .Caption = "Item3"
     End With
End Sub
```

Adicionando Um Novo Menu Ao Menu De Planilha

O código que segue é um outro exemplo que adiciona um novo menu personalizado ao menu da planilha, exatamente antes do menu Ajuda, e coloca dois comandos nele. A propriedade OnAction especifica a macro que cada comando roda:

```
Public Sub AddCustomMenu()
     Dim barWS As CommandBar
     Dim mnuCustom As CommandBarControl
     Dim HelpIndex As Integer
     Set barWS = CommandBars("Worksheet Menu Bar")
     HelpIndex = barWS.Controls("Ajuda").Index
     Set mnuCustom = barWS.Controls.Add(Type:=msoControlPopup, _
          Before:=HelpIndex)
     With mnuCustom
          .Caption = "&Personalizado"
          With .Controls.Add(Type:=msoControlButton)
               .Caption = "&Mostrar o Formulário de Dados"
               .OnAction = "MostrarFormularioDados"
           End With
           With .Controls.Add(Type:=msoControlButton)
               .Caption = "&Imprimir Lista de Dados"
               .OnAction = "ImprimirListaDados"
           End With
     End With
End Sub
```

Desabilita o Menu Salvar Como

O que segue desabilitará o item de menu Salvar Como no menu da planilha:

```
Sub DesabilitaItemMenuSaveAs()
      With CommandBars("Worksheet Menu Bar")
            With .Controls("Arquivo")
                  .Controls("Salvar Como...").Enabled = False
            End With
      End With
End Sub
```

Colocando de volta a propriedade Enabled para True ela liga o item de menu Salvar Como.

Recompondo Os Menus

O que segue restaurará os menus da planilha:

```
MenuBars(xlWorksheet).Reset
```

Por favor note que isto removerá toadas as modificações feitas por quaisquer add-in.

Protegendo Commandbars

Para proteger uma commandbar de modificação, use uma declaração como a que segue:

```
CommandBars("Worksheet Menu Bar").Protection = msoBarNoCustomize
```

Para remover a proteção, use:

```
CommandBars("Worksheet Menu Bar").Protection =
msoBarNoProtection
```

O valor da Protection pode ser feito com uma das ou soma das seguintes:

MsoBarNoProtection
MsoBarNoCustomize
msoBarNoResize
msoBarNoMove
msoBarNoChangeVisible
msoBarNoChangeDock
msoBarNoVerticalDock
msoBarNoHorizontalDock.

Como Adicionar Uma Barra Separadora de Item de Menu

Configure a propriedade BeginGroup para True para adicionar uma barra separadora antes de um novo item de menu ou um botão na barra de comando.

O que segue ilustra como adicionar uma barra separadora acima de um item de menu adicionado ao menu Ferramentas:

```
Dim menu_Item
      'adiciona um item de menu temporário ao menu Ferramentas
      'ele desaparece quando o Excel é fechado mas não quando o
      'arquivo é fechado a menos que a macro remover abaixo seja chamada.
      With CommandBars("Worksheet Menu
Bar").Controls("Ferramentas")
            Set menu_Item = _
                  .Controls.Add(Type:=msoControlButton,
Temporary:=True)
      End With
      'atribui nome e macro ao item de menu.  Também, coloca uma linha
      'separadora acima dele configurando BeginGroup para True
      With menu_Item
            .Caption = "Isto Diz Alô"
            .OnAction = "SayHello"
            'this puts the barra separadora above the item de menu
            .BeginGroup = True
      End With
```

Determinando Qual Botão Foi Clicado Numa Barra de Ferramentas

O que segue ilustra como determinar qual botão foi clicado numa barra de ferramentas. Cria uma barra de ferramentas personalizada chamada TestBar1, adiciona três controles personalizados a ela, e configura seus captions para One, Two, e Three. Atribui esta macro a cada botão:

```
Sub WhichButtonWasPressed()
      With CommandBars("TestBar1")
            'retorna o caption ou nome do botão que foi
            Select Case CommandBars.ActionControl.Caption
                  'adapta o caption à declaração case
                  Case "One"
                        MsgBox "Você pressionou Um"
                  Case "Two"
                        MsgBox "Você pressionou Dois"
                  Case "Three"
                        MsgBox "Você pressionou Três"
            End Select
      End With
End Sub
```

Por favor note que as declarações Case acima são *case sensitive* a menos que você coloque a Option Compare Text no topo do seu módulo ou converta todo o texto à mesma *case*.

Um outro modo de se determinar qual botão foi clicado é usar a propriedade Tag ou Parameter do controle:

```
MsgBox CommandBars.ActionControl.Parameter
```

CommandBars E Números de Controles

O que segue ilustra como modificar um menu usando o número de controle de menus ao invés de seu nome. Os nomes são específicos da linguagem.

```
Sub Add_Menu_Item()
      Dim ctlMenu As CommandBarControl
      Dim ctlMenuItem As CommandBarControl
      'remove antes de adicionar!
      RemoveMenuItem
      '
      'o número 30007  é o menu Ferramentas
      '
      Set ctlMenu = _
            Application.CommandBars(1).FindControl(, 30007)
      Set ctlMenuItem = _
            ctlMenu.Controls.Add(Type:=msoControlButton)
      ctlMenuItem.Caption = "QuickTable"
      ctlMenuItem.OnAction = ThisWorkbook.Name & "!QuickTable"
End Sub

Sub RemoveMenuItem()
      Dim ctlMenu As CommandBarControl
      On Error Resume Next
      Set ctlMenu = _
            Application.CommandBars(1).FindControl(, 30007)
      ctlMenu.Controls("QuickTable").Delete
End Sub
```

Como Adicionar Um Atalho de Menu

Com respeito à sintaxe, você acessa atalhos de menus exatamente da mesma maneira que você adiciona-os aos menus regulares. O truque é conhecer qual deles se referir, pois existem 40 deles. O que artigo que segue com base no *MS Knowledge* é de aproximadamente 19 páginas, e é uma boa referência sobre as barras de comando em geral. A página 12 contém uma lista dos nomes das barras de atalhos embutidas.

"XL97: WE1183 "Customizing Menu Bars, Menus and Menu Items"
http://support.microsoft.com/support/kb/articles/q166/7/55.asp

Caixas de Texto Na CommandBars

Caixas de Textos sobre CommandBars funcionam exatamente como caixas de texto normais. O modo mais fácil par ler o texto no procedimento de você atribuir ao controle caixa de texto é colocar o código seguinte:

```
Dim szText As String
```

```
szText = CommandBars.ActionControl.Text
```

Listagem Dos Atalhos de Menus

O que segue macro cria a lista na folha ativa de todos os menus de atalho e itens de menus em cada um.

```
Sub List_Short_Cut_Command_Bars_And_Menus()
      Dim cell   As Range
      Dim R    As Integer
      Dim c    As Integer
      Dim ctlBar  As CommandBar
      R = 2
      Range("a1").Value = "Index #"
      Range("b1").Value = "Commandbar Name"
      Range("c1").Value = "item de menu captions"
      For Each ctlBar In CommandBars
            If ctlBar.Type = msoBarTypePopup Then
            Cells(R, 1) = ctlBar.Index
                  Cells(R, 2) = ctlBar.Name
                  For c = 1 To ctlBar.Controls.Count
                        Cells(R, c + 2) = _
                         ctlBar.Controls(c).Caption
                  Next
                  R = R + 1
            End If
      Next ctlBar
      Cells.EntireColumn.AutoFit
      MsgBox "All done"
End Sub
```

Códigos de Menu Disponíveis Na Internet

Se você já ouvir falar sobre o *Baarns web site*, você pode pegar o seu arquivo *Developer Jumpstart*. Ainda mais para muitos outros modelos de códigos, este arquivo contém uma tabela de driven para construir CommandBar. Tudo o que você tem a fazer é preencher a tabela e incluir nela um módulo de código extra no seu projeto, ela construirá qualquer das commandbars que você especificou na tabela automaticamente. Ele construirá menubars, barras de ferramentas e menus popup todos da mesma tabela.

Você pode encontrar vários exemplos de fazer de códigos de menu no site de *John Walkenbach*,

http://www.j-walk.com/ss

Procure a seção *Excel Developer Tips*. Também, examine a seção *downloads*. Vários dos arquivos incluem códigos que adicionam um item de menu ao manu Ferramentas.

Artigos na Internet Sobre Como Mudar Os Menus

Para maiores informações de como mudar os menus Excel, *download* e rodar os seguintes arquivos. Quando você rodar os arquivos, eles instalarão um documento *word* que você pode então abrir e ler.

Customizing Menu Bars, Menus, e Menu Items

Como Evitar Customization de Menus e Toolbars

Desabilitando a Personalização da Commandbar

Se você estiver usando o Excel 2002, é possível evitar que um usuário modifique as commandbars. A declaração é:

```
CommandBars.DisableCustomize = True
```

Para habilitar a personalização use:

```
CommandBars.DisableCustomize = False
```

Se a sua aplicação ainda precisar rodar sob uma versão anterior, use a seguinte abordagem (*approach)*:

```
Dim cBars As Object
  Set cBars = Application.CommandBars
  If Val(Application.Version) >= 10 Then
      cBar.DisableCustomize = True
  End If
```

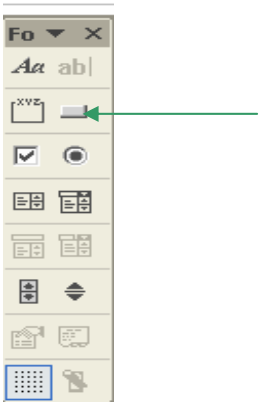
Lição 10: Usando Controles ActiveX nas Planilhas e Gráficos

As lições 5, 6 e 7 mostraram-lhe como criar *userforms* e colocar neles objetos tais como as barras de rolamento, as caixas de listagem, e os botões de comando. O Microsoft Excel também lhe permite colocar tais objetos **diretamente** numa planilha ou gráfico. Se o objeto for um botão de comando, uma macro pode ser especificada a executar quando o botão for clicado. Se o objeto for uma caixa de listagem, um botão de rotação ou uma barra de rolamento, então estes podem estar vinculados às fórmulas ou células nas suas planilhas.

CRIANDO BOTÕES E USANDO A BARRA DE FERRAMENTAS FORMULÁRIO

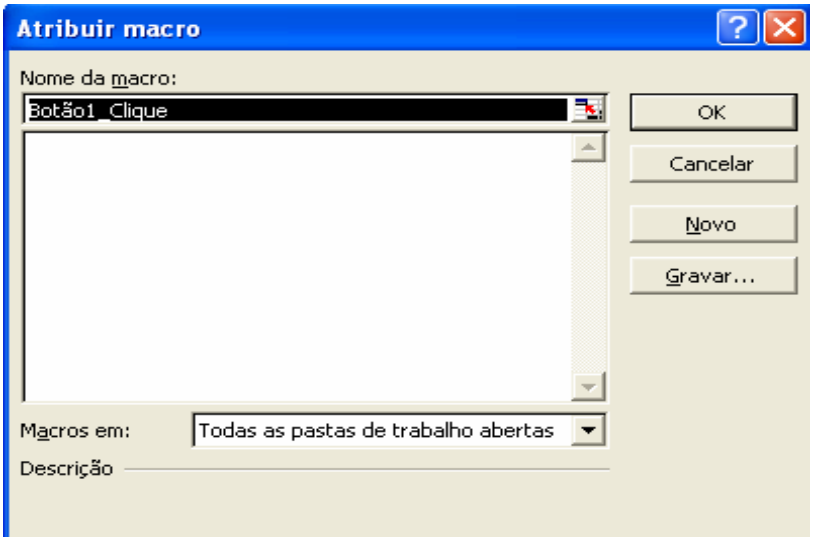
O modo mais fácil de se criar um botão e atribuir uma macro a ele é usar o seguinte botão da barra de ferramentas Formulário.

Para mostrar a barra de ferramentas Formulário, selecione Exibir, Barra de ferramentas, e clique na barra de ferramentas Formulário. Você pode ocultar esta barra de ferramentas quando acabar. Os botões criados deste modo podem ser colocados nas planilhas e nos gráficos.



Quando você clicar neste botão, um *serrilhado* aparecerá e lhe permitirá desenhar o botão. Simplesmente clique com o botão da esquerda do mouse onde você quer o canto superior daquele botão que estamos criando, mantenha o botão do mouse apertado, e arraste para onde você quiser o canto inferior e solte.

Você pode fazer o seu botão tão grande quanto você queira. Quando você soltar o botão do mouse, a caixa de diálogo Atribuir Macro seguinte surge e solicita-lhe especificar a macro a ser associada ao botão.

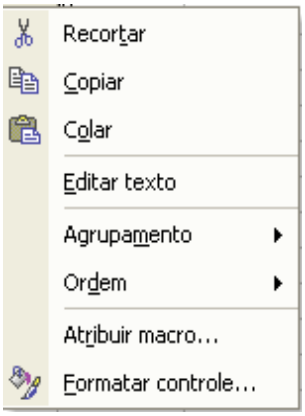


As macros aparecerão em ordem alfabética, qualificadas pelos nomes das pastas se elas existirem em pastas diferentes.

Depois de você selecionar OK, a caixa acima fecha e seu botão é mostrado. É dado um nome default como "**Botão 1**". Também, você notará que o botão está selecionado, como islutrado pela aparência seguinte:

Quando o botão está selecionado, você pode clicar no botão e editar o texto que aparece nele. Se você pressionar enter enquanto você estiver digitando o texto no botão, você irá para a próxima linha. Quando você estiver terminado de digitar o texto, deselecione o botão, apenas clicando em qualquer célula na folha. Para selecionar novamente um botão, mantenha apertado a tecla CTRL e clique nele usando o botão esquerdo do mouse.

Se você clicar num botão com o botão direito do mouse, o botão será selecionado e o painel seguinte é mostrado:



O item de menu Formatar controle... permite-lhe mudar a aparência da fonte. Por exemplo, você pode negritar o texto e mudar a sua cor e o seu tamanho. A mudança afeta todo o texto, não apenas parte dele. O item de menu Atribuir Macro... permite-lhe mudar a macro que foi atribuida ao botão.

Uma vez assumida uma macro para o seu botão e editado o texto no botão para descrever sua função, clique em qualquer célula na planilha para de-selecionar o botão. Agora, quando você clicar no botão, a macro atribuida a ele é executada.

Se você quiser que o botão seja visível não importando onde você rolar na planilha, então coloque o botão na célula A1, selecione uma célula cujo canto esquerdo superior esteja logo abaixo e à direita do botão, e então selecione Janela, Congelar painéis.

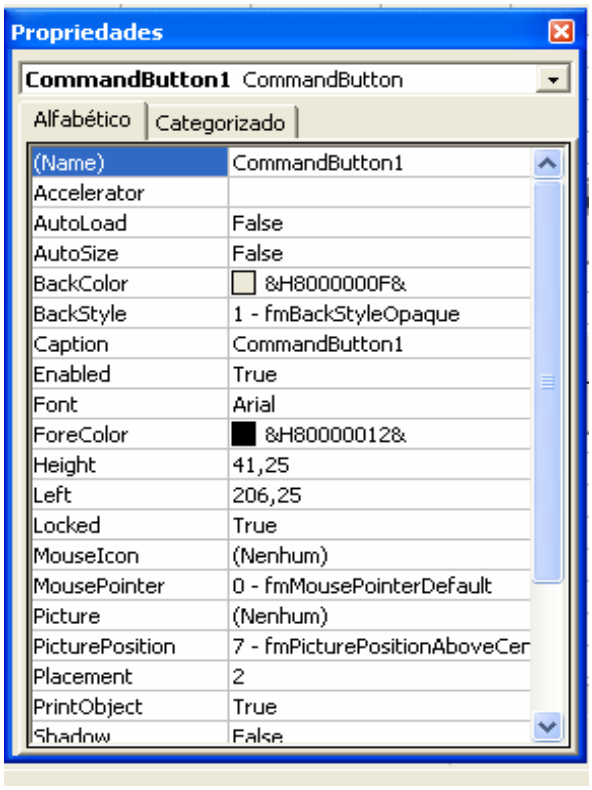
CRIANDO UM BOTÃO ActiveX

O outro modo de se criar um botão numa barra de ferramentas é fazer o seguinte:

- Selecione Exibir, Barras de ferramentas e mostre a barra de ferramentas de Controle. A barra de ferramentas seguinte aparecerá:

O 6º botão da direita tem uma dica de ferramenta "**Botão de comando**". Você pode clicar neste botão e daí desenhar um botão na planilha. Entretanto, você não pode desenhar botões de comando ou qualquer outro controle desta barra de ferramentas nas folhas de gráfico. **Você deve usar os objetos da barra de ferramentas** formulários **em vez disso.**

Depois que você desenhar um botão na planilha, você notará que o botão está selecionado, e que o primeiro botão da barra de ferramentas acima está destacado. Isto significa que você está no **modo *design***, e pode mudar as propriedades do botão. Para mudar as propriedades, clique no segundo botão, cuja dica de ferramenta é "**Propriedades**". Por exemplo, você pode mudar o título (caption**)** do botão, a fonte, e o background.



Você deverá quase sempre mudar a propriedade do botão TakeFocusOnClick para False. O valor default é True. Entretanto, isto causa problemas freqüentemente, quando o código freqüentemente se referir às células, e não existem células num botão!

Para atribuir código ao botão, dê um duplo clique nele enquanto estiver no modo *design* (primeiro botão selecionado/destacado). Isto colocará você no módulo de código para a planilha, com o evento clique do botão pronto par editar. Por exemplo, se o nome do botão for CommandButton1, você verá o seguinte código:

```
Private Sub CommandButton1_Click()
End Sub
```

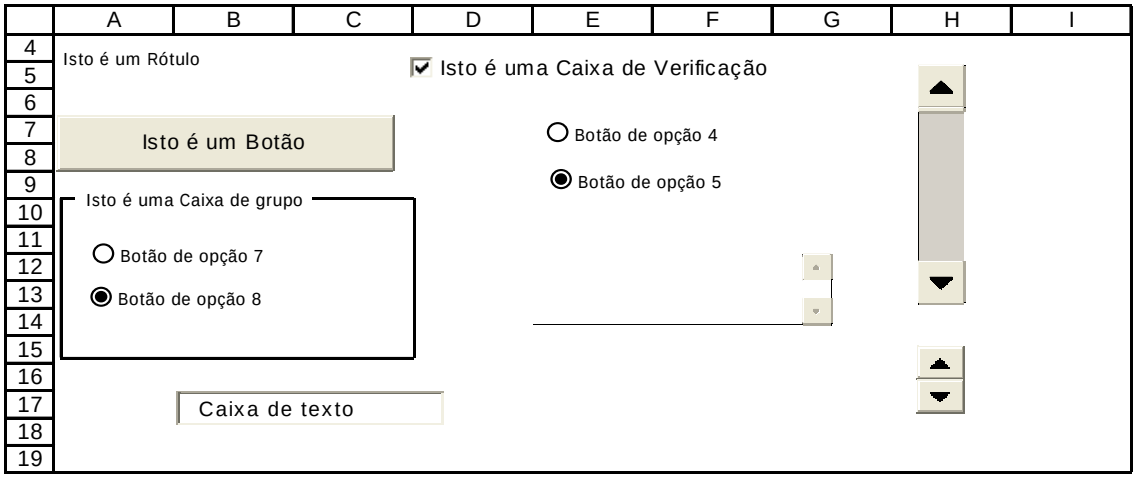
Você deverá encaixar seu código na sub-rotina acima, ou entrar com o nome da sub-rotina num módulo regular a ser rodado. A segunda é a abordagem preferida.

Quando você terminar de criar um botão, clique primeiro no botão na barra de ferramentas para desligar o modo *design*. Isto lhe permite ao clicar no botão, fazer o seu código ser executado. Se você precisar editar o botão no futuro, apenas mostre novamente a barra de ferramentas de Controle e clique primeiro no botão que o colocará de volta ao modo *design*.

Se você precisar editar o código de um botão no futuro, você pode apenas ir diretamente ao código clicando o botão direito do mouse na guia da folha e selecionar Exibir Código.

USANDO OUTROS OBJETOS DA BARRA DE FERRAMENTAS FORMULÁRIO

O que segue ilustra uma folha povoada com muitos objetos que podem ser colocados numa planilha, ou gráfico, usando a barra de ferramentas Formulários:

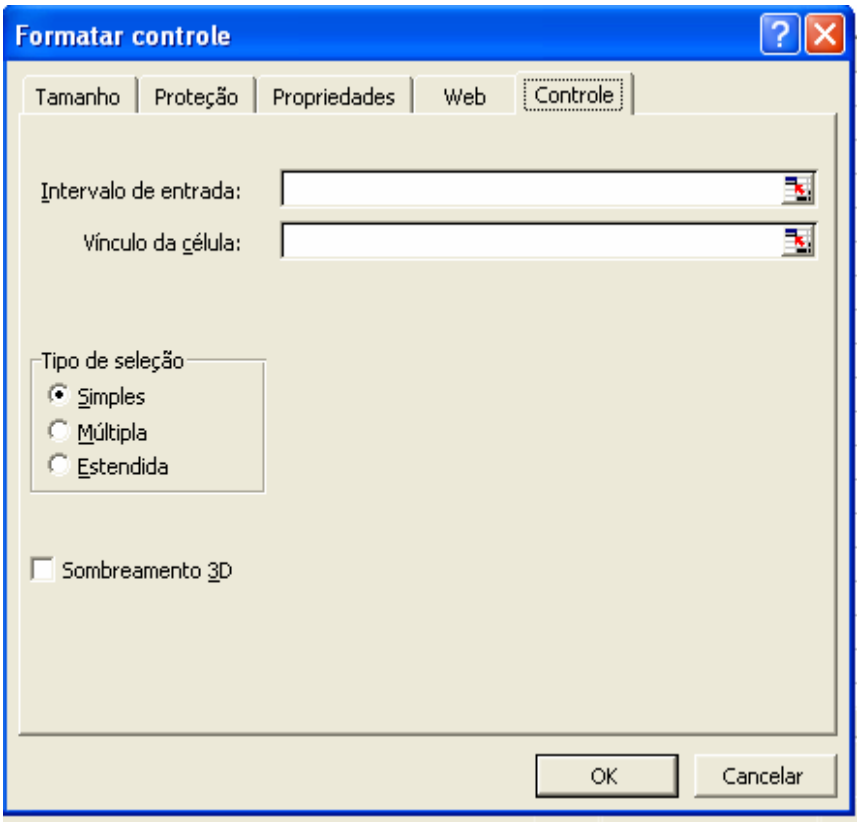


Note que os botões de opção podem ser incluidos numa caixa de grupo ou colocados na planilha do lado de fora de uma caixa de grupo. Você terá de criar a caixa de grupo primeiro.

Para selecionar um objeto que tenha sido inserido numa planilha da barra de ferramentas Formulários, mantenha apertada a tecla de controle e clique com o botão esquerdo do mouse no objeto. Isto lhe permite então editar seu texto e dimensionar o tamanho e rodar a macro atribuida. Geralmente, você não vai querer atribuir uma macro aos objetos de diálogo como a caixa de listagem, pois isto faria a macro ser executada

no momento que você clicar nela, provavelmente impediria você de usar o objeto de diálogo para a sua real função.

Se você clicar com o botão direito num objeto de uma folha e selecionar Formatar controle, seu painel de controle é mostrado. Você pode também mostrar este painel selecionando o objeto e daí selecionando Formatar, Controle (CTRL+1) do menu Microsoft Excel. O que segue ilustra o painel de controle para uma caixa de listagem criada da barra de ferramentas Formulários:



O *range* que você especificar num *range* de entrada de caixa de listagem controla o que está mostrado na caixa de listagem. O número da seleção é retornado na célula vinculada. Você deve também ter notado que o controle da caixa de listagem acima tem uma **caixa de verificação Sombreamento 3D**. Usando o valor que um dado objeto retorna à sua célula vinculada, você pode ter a sua ação em quaisquer dos objetos de diálogo refletida na sua pasta. Por exemplo, você poderia usar um botão de rotação ou uma barra de rolamento vinculada à célula mudando o valor de uma célula chave na sua folha de modo que você possa ver o efeito de diferentes valores sem ter que digitar os números. E outras fórmulas na sua planilha podem referir-se à célula vinculada, afetando assim seus valores. Um outro modo de se usar células vinculadas é com a **função índice** (index). A função índice seleciona um dado valor num *range* baseado na

sua posição no *range*. Por exemplo, digamos que você tenha uma planilha contendo os dados seguintes:

	A	B	C	D	E
1	Ano	2004	2005	2006	2007
2					
3	Vendas	344	472	407	422
4	Carregamentos	563	745	601	655
5	Preço	0,61	0,63	0,68	0,64

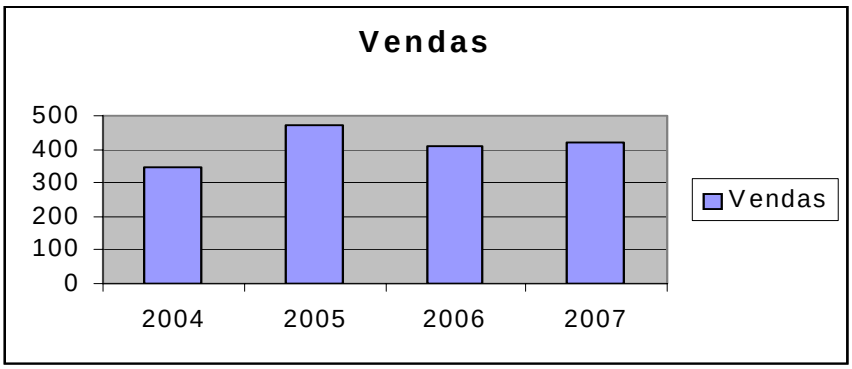
Se você usar a fórmula =ÍNDICE(B3:B5, número) numa célula, a fórmula retornará 344 se o número for 1, retornará 563 se o número for 2, e retornará 0,61 se o número for 3. Se o número for realmente de uma célula de referência, e o valor na célula referência for controlado por um controle do userform, tal como uma caixa de listagem, a ação seleção no userform muda o resultado da fórmula. Para ilustrar esta abordagem e como ela pode ser usada para graficar diferentes dados, entre com os dados acima numa folha. Daí então com as várias linhas abaixo, construa o seguinte:

	A	B	C	D	E
8	1				
9	Ano	2004	2005	2006	2007
10	Vendas	344	472	407	422

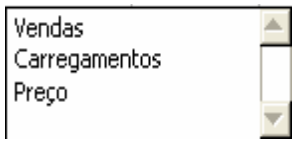
onde as seguintes fórmulas ou entradas são usadas:

Célula	Entrada
A8	1
A9..E9	2004..2007 (ou fórmulas referindo-se a A1..E9)
A10	= ÍNDICE (A3:A5;$A$8)
B10	= ÍNDICE (B3:B5;$A$8)
C10	= ÍNDICE (C3:C5;$A$8)
D10	= ÍNDICE (D3:D5;$A$8)
E10	= ÍNDICE (E3:E5;$A$8)

Note que quando você mudar o valor da célula A8, que o rótulo da célula A10 e os valores das células B10 até E10 mudam para refletirem os diferentes dados das vendas, carregamentos e preço de tabela. O próximo passo é construir um gráfico destacando as células A9 até E10. O gráfico pode ser um gráfico encaixado ou uma folha de gráfico única. Quando você fizer isto usando o assistente de gráfico, certifique-se de especificar que a primeira linha de dados é para ser usada para a categoria de rótulo X. Caso contrário, você obterá os números 1,2,.. como os rótulos de X em vez de anos. Também, certifique-se de especificar a primeira coluna de dados como a legenda de texto. O que segue é o gráfico resultante. Note que eu deletei a legenda para somente uma série ser plotada.



O último passo é inserir uma caixa de listagem na folha contendo o gráfico encaixado ou na folha de gráfico se você escolheu usar uma folha de gráfico ao invés disso. A caixa de listagem deverá usar as células A3:A5 para o *range* de entrada (estes são as vendas contendo as palavras Vendas, Carregamentos, Preços). E ela deverá usar a célula A8 como a célula vinculada. O que segue é com o que caixa de listagem terminada se parece:



Agora, quando você selecionar um dos itens na caixa de listagem, o gráfico é automaticamente atualizado, incluido seu rótulo de título.

USANDO OUTROS CONTROLES ACTIVEX

Se você quiser, você pode usar os controles ActiveX que estão na Barra de ferramentas Controle para criar controles na sua planilha ao invés de usar os objetos da barra de ferramentas Formulários. Somente objetos da barra de ferramentas Formulários podem ser colocados num gráfico. Se você fizer isto, você deverá quase sempre mudar a propriedade de botão TakeFocusOnClick para False. O valor *default* é True. Entretanto, isto frequentemente causa problemas, como o código frequentemente se referir às células , e não existirem células no objeto controle!

Você deverá editar as propriedades controle enquanto estiver no modo *design* (primeiro botão está apertado) selecionando o objeto e dai clicando no botão Properties. Para editar o código atrás do objeto, primeiro dê um duplo clique no controle. Isto cria a sub-rotina default do controle e coloca você no módulo de código da planilha. Se você precisar editar o código do controle no futuro, você pode apenas ir diretamente ao código clicando com o botão direito do mouse na guia da folha e selecionando Exibir Código. Os exemplos de códigos no capítulo sobre user*forms* ilustram comocriar código para os diferentes objetos. Por favor note que você não pode criar uma caixa refEdit na planilha nem colocar qualquer controle ActiveX num gráfico.

USANDO UMA MACRO PARA MUITOS BOTÕES

Se você colocar uma porção de botões de comando numa folha (usando o botão barra de ferramentas formulários), você deve querer fazer o seguinte:

- Atribuir a mesma macro a todos os botões de comando
- Usa Application.Caller para determinar quais botões foram selecionados.
- Baseado no botão selecionado, determine que ação deverá ser tomada.

Isto tem a vantagem que somente uma macro precisa ser mantida em vez de muitas. Também, as macros chamadas por esta macro podem ser do tipo Private, assim não aparece na lista Ferramentas, Macro.

Para ilustrar esta abordagem, crie a seguinte macro:

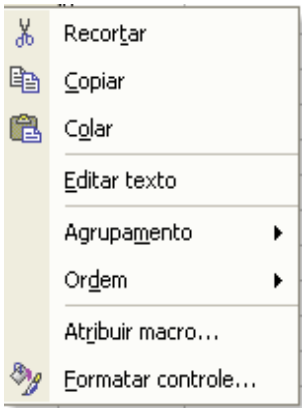
```
Sub ExemploDeBotão()
Dim NomeDoBotão As String
NomeDoBotão = Application.Caller
MsgBox "Botão " & NomeDoBotão & " selecionado."
End Sub
```

Daí adicione um número de botões de comando numa folha, e atribua a todos eles a macroa cima. Quando você clicar num botão, a macro acima mostra o nome do botão. Por favor note que o nome não é o texto que aparece na face do botão, mas o nome que aparece na Caixa Name quando o botão for selecionado (mantendo a tecla shift apertada quando você clicar nele). Você pode mudar o nome do botão editando o nome na Caixa Name. Também, você pode determinar que botão está selecionado usando testes If ou a declaração Select Case e rodar as macros baseadas nestes testes.

ATRIBUINDO MACROS A OUTROS OBJETOS NUMA FOLHA

Acima foi ilustrado a inserção de um botão numa planilha ou num gráfico e atribuir uma macro ao botão. É possível atribuir uma macro a qualquer objeto que se colocar numa planilha ou folha de gráfico. Por exemplo se você colar uma figura numa folha, você pode atribuir uma macro a executar quando a figura for clicada.

Para atribuir uma macro a um objeto, primeiro clique com o botão direito do mouse no. O seguinte menu então aparecerá:



Selecionando Atribuir macro... permite-lhe especificar a macro a atribuir ao objeto.

SUMÁRIO

Adicionar objetos às suas folhas Excel, especialmente botões de comando que ligam macros ou barras de rolamento que mudam valores das células, adiciona um poder tremendo. Por exemplo, se você é solicitado a criar uma planilha que modele preço e volume e seu impacto nos rendimentos, pense na reação que você obteria se adicionasse um gráfico com barras de rolamento que automaticamente mudam o preço e o volume vs oferecer apenas uma planilha cheia de equações!

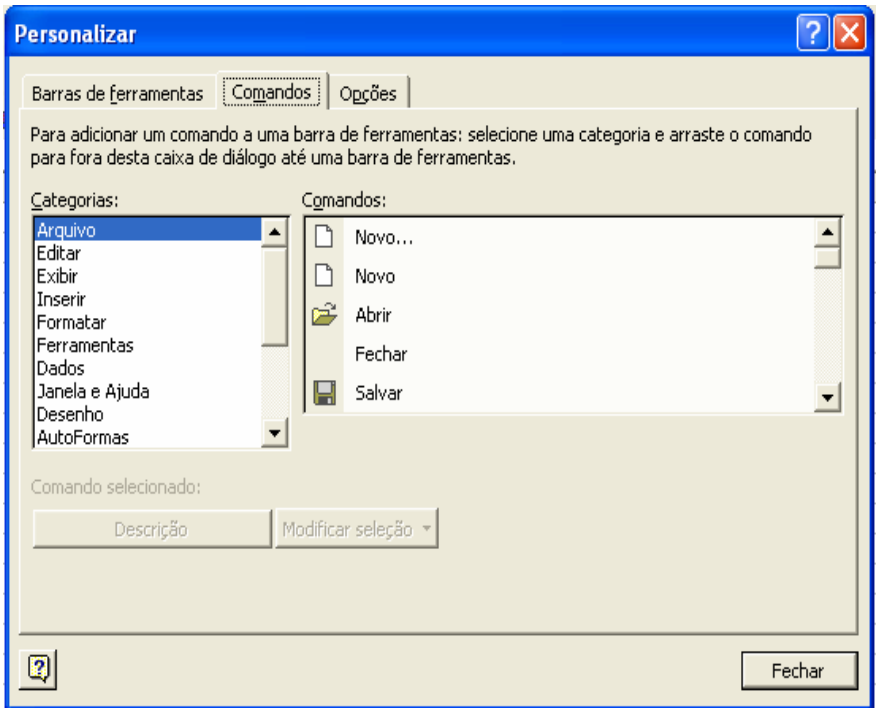
Lição 11: Trabalhando com Barras de Ferramentas

É possível com o Microsoft Excel modificar as barras de ferramentas de modo que as ferramentas que você precisar são aquelas que estão nas suas barras de ferramentas. Não somente se pode atribuir qualquer um dos muitos botões ferramenta às suas barras de ferramentas, você pode também atribuir macros aos seus botões que podem ser adicionados a uma barra de ferramenta existente ou a uma nova barra de ferramenta. E, você pode ter macros para criar barras de ferramentas e botões quando uma pasta for aberta e removê-los quando ela for fechada.

Esta lição compartilha você com técnicas e macros para executar as adaptações acima.

PERSONALIZANDO BARRAS DE FERRAMENTAS

Para personalizar suas barras de ferramentas, escolha Exibir, Barras de ferramentas, Personalizar do menu Microsoft Excel. Dai então selecione a guia Comandos:

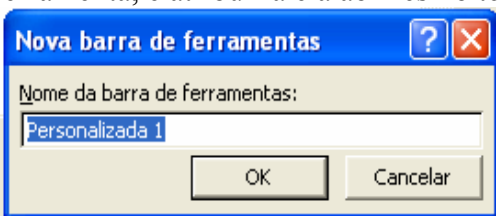


Nas muitas categorias que estão disponíveis nestas caixas, existem mais de 200 botões diferentes disponíveis. Para ver o que um botão faz, apenas clique no botão para ver uma descrição daquele botão. A descrição do botão aparecerá no fundo do diálogo.

Para adicionar um botão a uma barra de ferramenta, clique no botão e, enquanto mantiver clicado pressione o botão esquerdo do mouse para baixo, arraste o botão à localização desejada na barra de ferramenta e libere o botão do mouse. Se você quiser mover um botão para uma localização diferente, clique e arraste-o. Para remover um botão da barra de ferramenta, clique no botão e arraste-o para fora da barra de

ferramenta e libere-o. Ele desaparecerá. Se você quiser adicioná-lo de volta, apenas encontre-o entre os botões disponíveis na caixa acima eadicione-o de volta.

Para atribuir um botão à sua própria barra de ferramenta, você deve primeiro selecionar a primeira alça ("Barras de ferramentas") e clicar no botão "Nova..." para criar uma nova barra de ferramenta. Isto mostrará o diálogo seguinte que lhe permite criar uma nova barra de ferramenta, e atribuir a ela ao mesmo tempo um nome:



Isto cria uma barra de ferramenta vazia. Você pode então adicionar botões a ela.

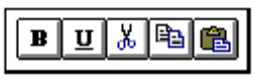
Para ocultar uma barra de ferramenta, você pode fazer qualquer uma das coisas seguintes:

- Se a barra de ferramenta é uma barra de ferramenta flutuante, clique no pequeno botão no canto esquerdo superior da barra de ferramenta.
- Se a barra de ferramenta está ancorada às outras barras de ferramentas suas, arraste-a para fora que a tornará uma barra de ferramenta flutuante. Daí então faça a ação acima.
- Se você souber o nome da barra de ferramenta, selecionar Exibir, Barras de ferramentas e de-selecionar a caixa de verificação barra de ferramenta.

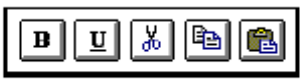
As Barras de ferramentas que você criar podem ser deletadas. Para deletar uma barra de ferramenta, selecionar Exibir, Barras de ferramentas, Personalizar. Selecione a primeira alça, rotulada de "barras de ferramentas". Clique sobre a barra de ferramenta que você quiser deletar e daí então sobre o botão delete.

As barras de ferramentas que vem com o Microsoft Excel não podem ser deletadas. Entretanto, elas podem ser reconfiguradas ao seu projeto original selecionando Exibir, Barras de ferramentas, Personalizar e clicando no botão reset depois de você ter selecionado a barra de ferramenta.

Em muitas das barras de ferramentas, os botões estão muito próximos uns dos outros, o que torna o seu uso de certa forma difícil. O que segue ilustra isto:

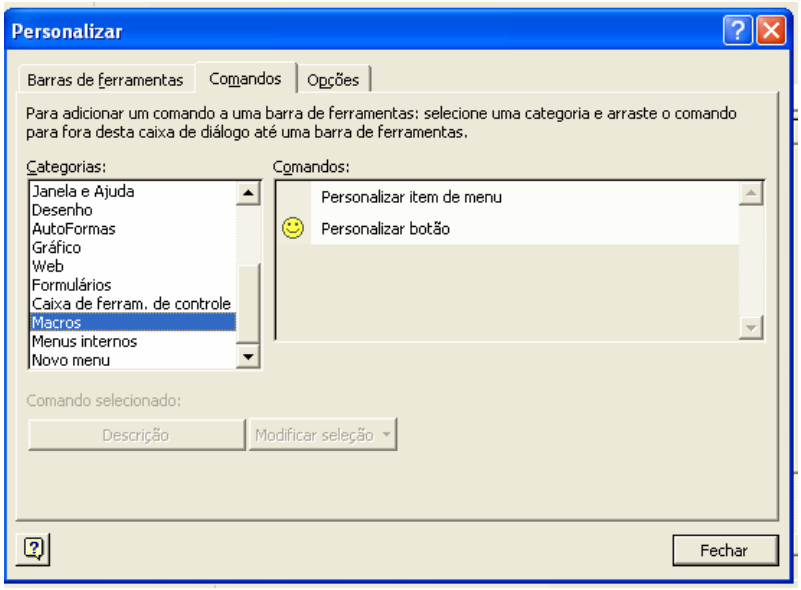


Entretanto, é possível separar os botões clicando e arrastando cada botão à direita um pouco. O que segue ilustra o espaçamento melhorado que resulta:



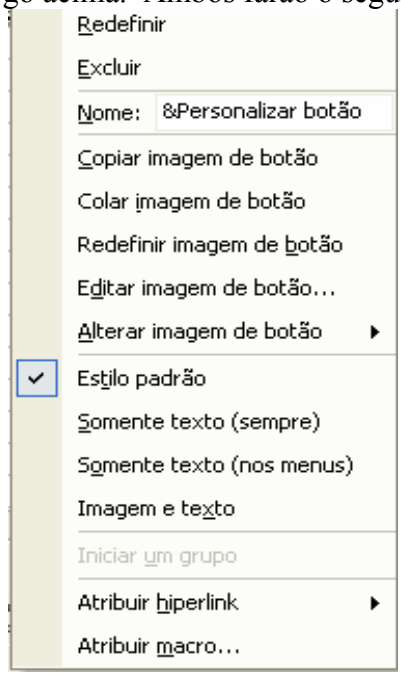
ATRIBUINDO MACROS AOS BOTÕES DA BARRA DE FERRAMENTA

Para atribuir macros a um botão, selecionar Exibir, Barras de ferramentas, Personalizar e selecionar a alça Comandos. Role para baixo a seleção Categorias até você chegar aquela uma chamada "Macros". Selecionando-a, o seguinte lhe será mostrado:



Selecione e arraste o item da lista Comandos entitulado "Personalizar Botão" à barra de ferramenta de sua escolha.

Para atribuir uma macro a este botão, ou mudar a sua face, basta clicar com o botão direito do mouse no botão ou selecionar o botão e daí então usar o botão "Modificar seleção" no diálogo acima. Ambos farão o seguinte pop-up aparecer:

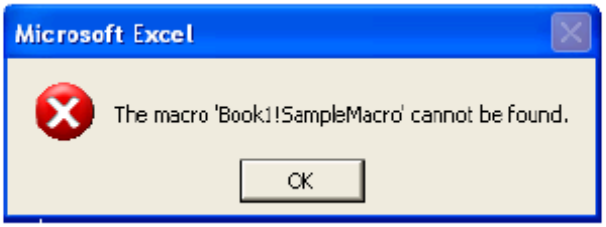


Para atribuir uma macro ao botão, selecionar o item de menu "Atribuir Macro". Para mudar a face do botão, ou use o item de menu "Editar imagem de botão" ou o item "Alterar imagem de botão". Este segundo item mostrará a seguinte lista de seleção das faces.

Selecione a face que você quer usar e daí então saia do painel. Com esta informação, o Excel não gravará a face que você usou se você estiver gravando uma macro para criar uma barra de ferramentas.

RESOLVENDO O PROBLEMA MACRO CANNOT BE FIND

Se você atribuiu uma macro a um botão, clicando no botão automaticamente abrirá o arquivo macro se o arquivo não estiver aberto. Se você renomeou a macro ou moveu ou renomeou o arquivo, você obterá uma mensagem como a seguinte quando você clicar no botão:

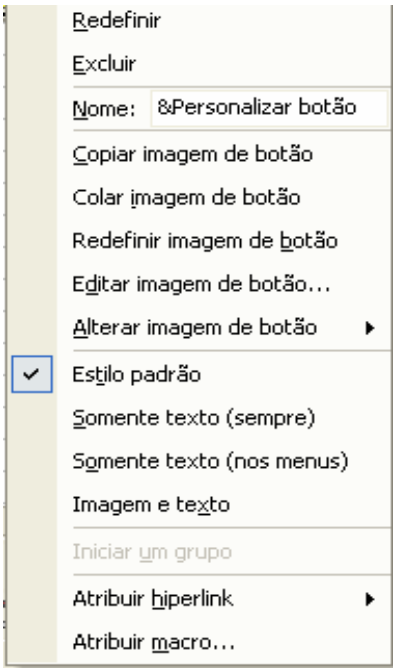


Para resolver este problema, você terá de atribuir novamente a macro ao botão. Par fazer isto, selecionar Exibir, Barras de ferramentas, Personalizar. Depois clicar no botão. Neste momento você pode selecionar Ferramentas, Atribuir Macro e escolher uma macro para o botão se o arquivo contendo a macro estiver aberto. Você pode also clicar com o botão direito do mouse no botão e selecionar Atribuir Macro do menu de atalho que aparece.

EDITANDO BARRA DE FERRAMENTA IMAGENS DE BOTÃO

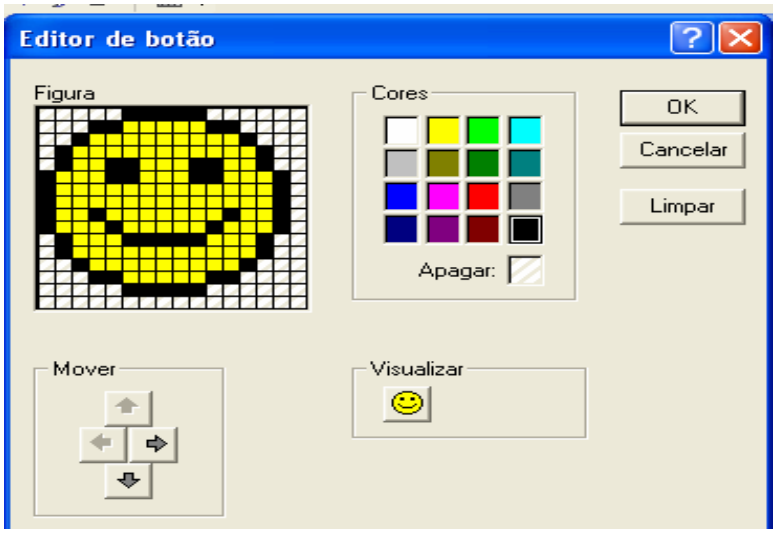
Muito freqüentemente, a imagem dos botões personalizados não representam a macro que você atribuiu ao botão. É possível editar a imagem do botão para dar a ele a aparência que você quiser.

Para editar um a imagem de botão, selecionar Exibir, Barras de ferramentas, Personalizar se a caixa de diálogo personalizar não estiver visível. Daí então clicar com o botão direito do mouse no botão. Isto fará o seguinte menu popup aparecer:



Para mudar a dica da ferramenta associada com o botão, mude o nome do botão.

Para editar o botão, selecione Editar imagem de botão. Isto fará o seguinte editor de botão aparecer. Por favor note que a imagem que aparecerá será aquela do seu botão.



Para usar este editor, você seleciona a cor que você gostar e daí então clique nas pequenas caixas na área da figura para mudar a imagem. Os botões Move permitem você posicionar a imagem no botão.

TRABALHANDO COM BARRAS DE FERRAMENTAS

Barras de ferramentas são membros da coleção CommandBars:

```
CommandBars(número)
```

ou `CommandBars("nome da barra de ferramenta ")`

A coleção commandbars inclui não somente a coleção barras de ferramentas, mas também a coleção menus.

Por exemplo, as declarações seguintes escreverão uma lista de todos objetos na coleção commandbars junto com seu número índice:

```
For i = 1 To Commandbars.Count
    Cells(i, 1) = i
    Cells(i, 2) = Commandbars(i).Name
Next
```

Se você quiser deletar uma barra de ferramenta que você criou, então você apenas se refira a ela e use o método Delete. Por exemplo:

```
CommandBars(33).Delete
```

ou
```
CommandBars("Custom 20")
.Delete
```

Não é possível deletar as barras de ferramentas que vem com o Microsoft Excel.

Se você quiser ocultar uma barra de ferramenta, então você pode configurar a propriedade Visible da barra de ferramenta para False. Para mostrar novamente, você configuraria a propriedade Visible para True. Por exemplo:

```
CommandBars("Special Task").Visible = True
```

Para adicionar uma barra de ferramenta, você usa o método Adicionar que tem um argumento Name. Por exemplo:

```
CommandBars.Add Name:="Minha Nova Barra"
     CommandBars("Minha Nova Barra").Visible = True
```

A barra de ferramenta aparecerá não tão afastada do canto esquerdo superior da tela. Isto não é tão notável. Você pode superar isto adicionando o seguinte a sua macro:

```
With CommandBars("Gerenciador de dados)
.Left = 200
     .Top = 200
End With
```

Você pode também simplificar isto acima numa declaração:

```
CommandBars.Add (Name:="Data Manager")
.Visible = True
```

Se você precisar mudar o nome numa barra de ferramentas personalizada, você pode fazer isto com a propriedade Name:

```
CommandBars("Barra de Ferramentas 33").Name = "Financial Info"
```

Para determinar o número de barras de ferramentas que existem, você pode usar a propriedade Count:

```
MsgBox "Você tem " & CommandBars.Count &
 " comandbars."
```

TRABALHANDO COM BOTÕES DA BARRA DE FERRAMENTAS

Os botões na barra de ferramenta pertencem à coleção CommandBarControls. Para trabalhar com um dado botão, você deve especificar a commandbar e o número do controle (botão):

```
CommandBars(número ou "nome").Controls(
número)
```

Para adicionar um botão a uma barra de ferramenta, você usará o método Add e especificará o botão a usar. Por exemplo:

```
CommandBars("Barra de Ferramentas 1").Controls.Add _
     Type:=msoControlButton, Id:=23
```

Note que você especificou um argumento Type e configurou-o igual ao valor. Isto é porque o Excel inclui menu e item de menus como parte de uma commandbar. Assim o argumento Type permite-lhe especificar o que está sendo adicionado. Também, em vez de se usar um argumento chamado Botão, você usa um chamado Id para identificar o botão. Como informação, se você não especificar um Type ou um Id , então um botão vazio é adicionado.

Os exemplos acima adicionaram botões que já estão associados com comandos embutidos do Excel. Neste caso ambas declarações adicionar o botão que mostra o painel abrir arquivo, embora o botão e números Id sejam diferentes. A melhor maneira par obter os números para um botão embutido é gravar uma macro que adiciona um botão a uma barra de ferramenta. A próxima seção cobre como criar suas próprias barras de ferramentas personalizadas com botões personalizados atribuidos às suas macros personalizadas.

Existem também outras coisas que você pode especificar quando você adicionar um botão. Por exemplo, você pode especificar sua posição na barra de ferramenta com o argumento Before.

A propriedade Name de um botão é também a ToolTip que aparece quando você manter o ponteiro do mouse sobre um botão. Você pode mudá-lo para qualquer texto que você queira. Por exemplo:

```
Barras de ferramentas("Barra de Ferramentas
25").ToolbarBotões(1)
.Name = _

 "Print Reports"
```

O intervalo que você colocar entre os botões é realmente considerado um botão e é identificado como botão zero! O que segue ilustra como adicionar um botão (a face smiley) e daí um espaço de botão na frente deste botão (esta declaração assume que exista já um botão na barra de ferramenta, que você não colocará um intervalo na frente do primeiro botão):

```
CommandBars("New ToolBar").Controls.Add _
        Type:=msoControlButton, _
    Id:=2950, Before:=1, BeginGroup:=True
```

a propriedade BeginGroup adiciona a distância entre os botões.

Para maiores informações sobre a commandbars, mostrar o Pesquisador de objetos, selecionar "<Todas Bibliotecas">" na caixa seleção, e role para baixo até commandbars no painel classes. Você pode então selecionar uma das listagens e selecionar a propriedade ou método e mostrar a ajuda clicando no botão question mark.

USANDO UMA MACRO PARA CRIAR UMA BARRA DE FERRAMENTA

De tempo em tempo, você deve precisar de uma barra de ferramenta especializada e botão par uma tarefa particular. Por exemplo, você deve ter de fazer um trabalho particular uma vez por mês e é mais fácil fazer se a barra de ferramenta está configurada com botões atribuidos às macros que você usará. Você poderia criar esta barra de ferramenta manualmente cada mês. Entretanto, uma macro pode criar barras de ferramentas com uma dica descritiva para cada botão.

O que segue macro ilustra o código necessário par criar uma barra de ferramenta com dois botões nela, com dicas de ferramentas descritivas.

```
Sub SpecialToolBar()
    Dim tBar, newButton
    'Deletar CommandBar se ela existir
    On Error Resume Next
    CommandBars(" Tarefa Especial ").Delete
```

```
    On Error GoTo 0
    'criar CommandBar
    CommandBars.Add Name:=" Tarefa Especial "
    'definir uma variável objeto para se referir à CommandBar
    Set tBar = CommandBars("Tarefa Especial")
    'adicionar primeiro botão
    Set newButton = tBar
.Controls.Add(Id:=2950)
    'especificar dica de ferramenta (nome), macro para rodar, e
texto da barra de status para a macro
    With newButton
         .OnAction = "AddInfo"
         .Caption = "Adicionar Info"
    End With
    'adicionar o próximo botão.  Note que o uso da propriedade
FaceId
    Set newButton = tBar.Controls.Add
    With newButton
          .Caption = "Remover Informação"
          .OnAction = "RemoveInfo"
          .FaceId = 278
          .BeginGroup = True
    End With
    'mostrar CommandBar, posição na planilha
    tBar.Visible = True
    With CommandBars("Tarefa Especial")
            .Left = 200
            .Top = 200
    End With
End Sub
```

Pontos chaves na macro acima:

· Nenhum qualificador especial é necessário para adicionar um botão; apenas usando .Controls.Add adiciona um botão vazio. Isto usa o Type default de msoControlButton. Outros tipos adicionariam item de menus ou menus.

· A propriedade FaceId é usada para configurar o botão face

· Um espaço é adicionado entre os dois botões configurando a segunda propriedade BeginGroup do botão para True.

Se você quiser que a macro acima seja rodada quando a pasta for aberta, então apenas chame-a de uma macro Auto_Open. Por exemplo:

```
Sub Auto_Open()
     SpecialToolBar
End Sub
```

Fará a macro SpecialToolBar rodar quando o arquivo for aberto. Uma macro Auto_Open é aquela que é rodada automaticamente quando uma pasta for aberta.

USANDO SUAS PRÓPRIAS IMAGENS DE BOTÃO

A macro acima lhe permite usar quaisquer imagens de botão do Microsoft Excel para a face do seu botão de macro. Entretanto, você pode querer usar um botão face que você criou em vez disso. Para fazer isto, você primeiro precisa fazer o seguinte:

- Vá à Caixa de Diálogo Personalizar - Selecione Exibir, Barras de ferramentas, Personalizar
- Crie um botão contendo a face de botão que você quiser.
- Com o botão selecionado, clicar com o botão direito do mouse no botão e selecionar a Face Copy Botão
- Sair do diálogo personalizar e ir à planilha onde você pode armazenar o botão face. A folha que eu uso eu chamo de "Botão Faces", e é parte da minha pasta PERSONAL.XLS.
- Nesta folha, apenas faça uma Editar, Colar normal. O botão face será colado na planilha como um objeto figura. Ele será muito pequeno, o que é OK.
- Quando você selecionar o botão face, você verá o nome que o Microsoft Excel deu a este objeto figura na caixa Nome. Por exemplo, ele pode ser chamado "Figura 1". Clique na caixa nome (adjacente à caixa de edição de fórmula) e renomeie a figura digitando um novo nome e pressionando enter. O nome pode conter espaços. Para ajudá-lo no futuro, entre com este nome numa célula adjacente ao botão face.

Neste ponto, você pode usar agora o botão face que você armazenou na planilha acima e ter uma macro para colar no seu próprio botão. O que segue macro ilustra criando a nova barra de ferramenta com um botão face chamado "Graph Face" localizado na folha "Botão Faces" na PERSONAL.XLS:

```
Sub CreateGraphButton()
    Dim tBar, newButton
    'remover CommandBar se ela existir e criar uma CommandBar
vazia
    'com o mesmo nome
    On Error Resume Next
    CommandBars("Barra de Gráfico").Delete
    CommandBars.Add Name:=" Barra de Gráfico "
    'configurar uma variável para se referir à CommandBar
    Set tBar = CommandBars("Barra de Gr'fico")
    'adicionar um botão vazio à CommandBar
    Set newButton = tBar.Controls.Add
    'especificar uma tooltip e macro para rodar
    With newButton
         .Caption = "Gráfico dos dados sobre as vendas"
         .OnAction = "PERSONAL.XLS!graphdata"
    End With
    'selecionar a face a ser usada, copiá-la, e daí colar no
```

```
botão
    Workbooks("Personal.Xls").Worksheets("
Botão Faces") _
                    .DrawingObjects("Graph Face").Copy
    newButton.PasteFace
    'torne a CommandBar visível
    With tBar
           .Visible = True
           .Left = 200
           .Top = 200
    End With
End Sub
```

CRIANDO BOTÕES COM TEXTO NA BARRA DE FERRAMENTAS

Os exemplos até agora ilustraram como criar botões de barras de ferramentas que tenham uma face do tipo figura. Entretanto, você pode também criar um botão que tenha texto em vez de uma figura. O texto que aparece é idêntico à tooltip do botão. O que segue ilustra a criação de um simples botão commandbar com um text botão.

```
Sub TextButtonExample()
    Dim newBar, newbutton
    On Error Resume Next
    CommandBars("barra exemplo ").Delete
    On Error GoTo 0
    Set newBar = CommandBars.Add(Name:="
Barra Exemplo ")
    Set newbutton = newBar.Controls _
            .Add(Type:=msoControlButton, Id:=2949)
    With newbutton
        .Style = msoButtonCaption
        .Caption = "Isto é um Texto de Botão"
        .OnAction = "Alô"
    End With
    newBar.Visible = True
End Sub


Sub Alô()
   MsgBox "Alô Mundo"
End Sub
```

A chave para criar um botão na barra de ferramentas com uma face de texto é configurar a propriedade Style para msoButtonCaption. Você pode também configurar a propriedade Style para msoButtonIconAndCaption que mostra ambos um ícone e um texto descrição.

CRIANDO BARRAS DE FERRAMENTAS POP-UP

Uma outra característica que você pode criar são commandbars *pop-up*. Tais commandbars são mostradas por um comando numa macro. O benefício das commandbars *pop-up* é que elas têm o foco. Isto é, o usuário deve escolher nas seleções *pop-up* e não pode fazer outras ações. Neste sentido, elas são como diálogos, mas são muito mais fáceis de serem construidas e podem ser tão complexas quanto for necessário. O que segue ilustra como criar e mostrar uma commandbar *pop-up* que se parece com o seguinte:


```
Sub POPUP_EXAMPLE()
    Dim popUpBar As CommandBar
    Dim newButton As CommandBarControl
    On Error Resume Next
    CommandBars("Barra Pop UP").Delete
    On Error GoTo 0
    Set popUpBar = CommandBars _
        .Add(Name:=" Barra Pop UP", Position:=msoBarPopup)
    Set newButton = popUpBar.Controls _
          .Add(Type:=msoControlButton, Id:=1)
    With newButton
        .FaceId = 67
        .OnAction = "Macro1_To_Run"
        .Caption = "Esvaziar a Lixeira"
    End With
    Set newButton = popUpBar.Controls _
           .Add(Type:=msoControlButton, Id:=1)
    With newButton
        .FaceId = 69
        .OnAction = "Macro2_To_Run"
        .Caption = "Assistir a TV"
    End With
    popUpBar.ShowPopup
End Sub

Sub Macro1_to_Run()
     MsgBox "Esvaziar a lixeira!"
End Sub

Sub Macro2_to_Run()
     MsgBox "vez da TV!"
End Sub
```

O método ShowPopup faz o popup ser mostrado. As macros OnAction associadas com os controles na barra determinam qual ação tomar quando um botão é clicado. O *popup* automaticamente desaparecerá quando um comando é selecionado. Também, o popup pode ser tão elaborado quanto você queira - você pode adicionar menus, sub menus, e muitas outras caractereísticas de a um commandbar *popup*.

BOTÃO FACE ID'S

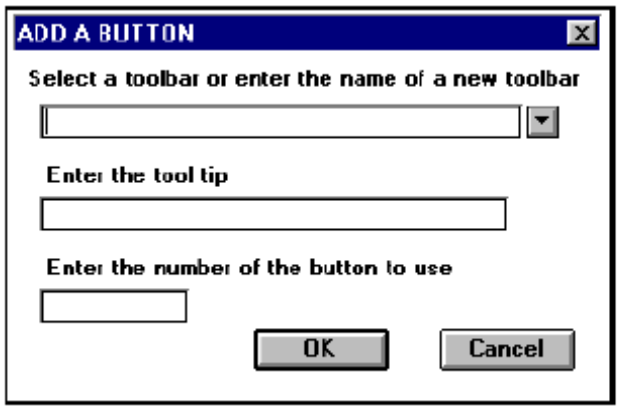
A macro que segue cria uma barra de ferramenta gigante com os primeiros 250 FaceID's que se pode encontrar noExcel. Se você quiser mais, então você pode rodar esta macro e mostrar cerca de 1000 faces diferentes! Se você fizer, você poderá querer modificar um pouco a macro par criar barras de ferramentas múltiplas. Como informação, gravando uma macro mudar a face de um botão não grava a face usada! Assim esta macro pode ser de uso para você!

```
Sub DisplayFaces()
    Dim tBar As Object, newButton As Object
    Dim Y As Integer
    Dim startNum As Integer
    startNum = Val(InputBox("Entrar com o número para o
primeiro" & _
        " botão: número (1, 251, 501, etc.)"))
    If startNum = 0 Then Exit Sub
    'liga o modo de erro daquela macro e continua se a barra de
ferramenta não existir se deletada
    'deixa ligado o modo erro e reiniciar é também necessário
para evitar colisões com algum botão
    'números são pulados pelo Microsoft Excel
    On Error Resume Next
    CommandBars("Todos os Botões!!!").Delete
    'cria a barra de ferramenta
    CommandBars.Add Name:="Todos os Botões!!!"
    Set tBar = CommandBars("Todos os Botões
!!!")
    'Microsoft Excel tem cerca de 1000 faces.
    For Y = startNum To startNum + 249
       Application.StatusBar = "criando face " & Y
            Set newButton = tBar.Controls.Add
            With newButton
               .Caption = "FaceId = " & Y
               .FaceId = Y
            End With
    Next Y
     Application.StatusBar = False
    'mostra a barra de ferramenta e a torna wider
    tBar.Visible = True
    tBar.Width = 504
    'limpa a mensagem da barra de status
    Application.StatusBar = False
End Sub
```

UMA MACRO PARA ADICIONAR UM BOTÃO

Embora você possa escrever uma macro que cria uma barra de ferramenta personalizada e botão a cada vez que você precisar de um, a macro seguinte adicionará um botão a qualquer barra de ferramenta e também adicionará uma *tooltip* ao botão. Se

você entrar com o nome de uma nova barra de ferramenta esta macro criará uma nova barra de ferramenta. Ela usa o userform seguinte:



O nome na lista *drop down* é DropDown1. As duas caixas de texto são TextBox1 e TextBox2. Os botões command foram configurados, e o seguinte código coloca o userform no módulo código (facilmente atingida dando um duplo clique sobre um botão:

```
Private Sub CommandButton1_Click()
    'Botão OK
    Me.Hide
    bResponse = True
End Sub

Private Sub CommandButton2_Click()
   'Botão Cancelar
    Me.Hide
    bResponse = False
End Sub
```

A primeira caixa é uma combinação das caixas lista *edit-dropdown*. Quando você criar o userform, assegure-se de configurar a alça ordem para ser pela ordem das caixas.

Para usar, você precisa preencher todas as três caixas. Use a barra de ferramenta Todos os Botões!!! Para obter o número do botão que você quer usar. Qualquer botão pode ser usado. Após um novo botão ter sido adicionado, você pode atribuir uma macro a ele apenas clicando sobre ele e selecionando uma macro do diálogo *pop-up* que aparece.

O que segue é o código da macro:

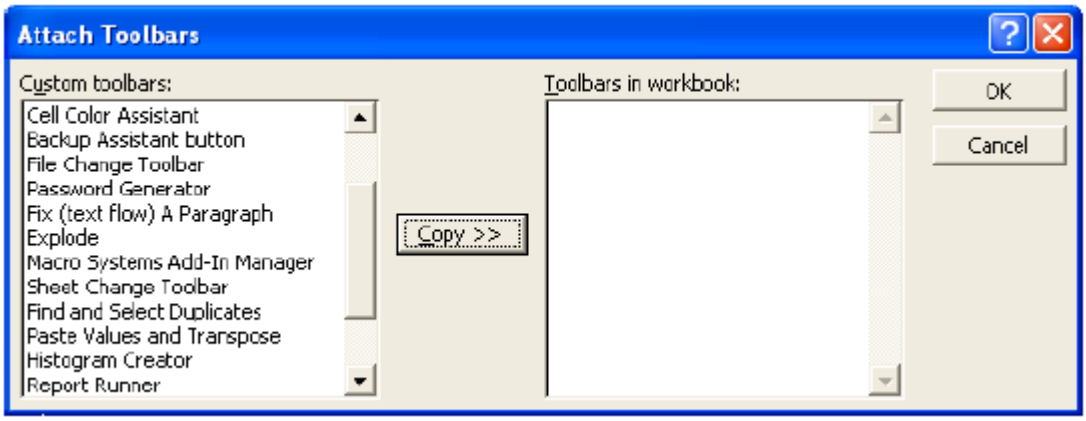
```
Global bResponse As Boolean

Sub ButtonAdder()
    Dim item As Object
    Dim I As Integer
```

```
    Dim cBar As CommandBar
    Dim bOK As Boolean
    Dim barName As String
    Dim barExists As Boolean
    Dim sToolTip As String
    Dim FaceIdNum As Integer
    Dim newButton As Object
    'multiplique a lista drop down
    For Each cBar In CommandBars
        'commandbars do tipo 0 são barras de botão
        If cBar.Type = 0 Then
            I = I + 1
            UserForm1.ComboBox1.AddItem cBar
.Name
        End If
    Next cBar
    'configure o número do botão face happy como o default
    UserForm1.TextBox1.Text = 59
    Do
        'Mostrar o userform
        UserForm1.Show
        If bResponse = False Then
           'se o botão cancel for clicado, descarregue e saia
            Unload UserForm1
            Exit Sub
        End If
        'Confirmar todas as informações que foram preenchidas
nele:
        With UserForm1
            If .ComboBox1.Text = "" Or _
                  .TextBox1.Text = "" Or _
                  .TextBox2.Text = "" Then
                  MsgBox _
                    "Todas as caixas no diálogos devem estar
preenchidas"
            Else
                  bOK = True
            End If
        End With
    'isto fará um laço todas caixas serem preenchidas
    Loop Until bOK
    'obtenha os valores do userform
    barName = UserForm1.ComboBox1.Text
    sToolTip = UserForm1.TextBox2.Text
    FaceIdNum = Val(UserForm1.TextBox1.Text)
    'descarregue o form quando o info tiver sido armazenado
    Unload UserForm1
    'Determine se a barra de ferramenta existe e se não, criá-la
    barExists = False
    For Each cBar In CommandBars
        If UCase(cBar.Name) = UCase(barName) Then _
            barExists = True
```

```
    Next
    If Not barExists Then
       CommandBars.Add barName
    End If
    'criar uma variável para se referir à barra de ferramenta
    Set cBar = CommandBars(barName)
    'adicionar  o botão à barra de ferramenta e obter seu número
    Set newButton = cBar.Controls.Add
    'atribuir o botão face ao novo botão
    On Error GoTo BadButtonNumber
    newButton.FaceId = FaceIdNum
    On Error GoTo 0
    'atribuir tooltip e macro ao botão
    newButton.Caption = sToolTip
    'adicionar um espaço entre o novo botão e os botões
'existentes
    newButton.BeginGroup = True
    'tornar a barra de ferramenta visível
    cBar.Visible = True
    MsgBox "Para atribuir uma macro ao botão, selecionar " &
        "Exibir, Personalizar e clicar com o botão direito do
mouse no botão."
    Exit Sub
    'Isto manipula um mal número de botão
BadButtonNumber:
    MsgBox "O número do botão que você especificou " & _
          "não existe.  Atividade paralizada."
End Sub
```

ATRIBUINDO UMA BARRA DE FERRAMENTA A UMA PASTA

Se você for a uma pasta e selecionar Exibir, Barras de ferramentas, Personalizar , e clicando no botão "Atribuir...". O painel que segue aparece.



Barras de ferramentas que você criou estão listadas na caixa esquerda. Se você selecionar uma barra de ferramenta, você pode copiá-la para a pasta. Isto lhe permite distribuir uma pasta com uma barra de ferramenta personalizada sem ter que re-criar

uma macro para a barra de ferramenta. Você pode querer incluir uma macro Auto_Open na pasta de modo que a barra de ferramenta seja mostrada quando o arquivo for aberto. Uma macro Auto_Open é aquela que roda automaticamente quando uma pasta é manualmente aberta.

Um dos problemas com esta abordagem é que as barras de ferramentas com o mesmo nome têm precedência sobre as barras de ferramentas atribuidas a uma pasta. Isto significa que as barras de ferramentas antigas aparecerão em vez daquelas barras de ferramentas novas se a pasta for movida. Assim a melhor abordagem é ter suas pastas criando suas barras de ferramentas e evitar o uso desta característica.

OCULTANDO E RESTAURANDO AS BARRAS DE FERRAMENTAS

Você pode algum dia querer desenvolver uma aplicação onde você oculta todas as barras de ferramentas do usuário, junto com objetos como os menus, a caixa de edição, e as barras de rolamento. Ocultar as barras de ferramentas é relativamente fácil. Entretanto, quando sua aplicação é feita, você precisar restaurar as barras de ferramentas de volta de modo que o usuário as tenha outra vez. Isto não é tão simples quanto parece, desde que você precisa saber quais barras de ferramentas serão mostradas, e a ordem que você mostrará essas barras de ferramentas afetam a sua localização. O que é exigido é que você armazene informação nas barras de ferramentas, e daí então quando mostrá-las novamente, você começa no topo esquerdo da barra de ferramenta. Deste modo, as barras de ferramentas terminarão exatamente onde elas estavam antes de sua aplicação rodar.

Ocultar e restaurar as barras de ferramentas e menus é muito fácil. O que segue oculta os menus, barras de ferramentas e a barra de edição de fórmula e daí então restaurá-las:

```
Sub MainHideShow
      HideCommandbars
      MsgBox "Pressione OK para restaurar as barras"
      ShowCommandbars
End Sub

Sub HideCommandbars()
    Dim cBar As CommandBar
    For Each cBar In CommandBars
         cBar.Enabled = False
    Next
    Application.DisplayFormulaBar = False
End Sub

Sub ShowCommandbars()
    Dim cBar As CommandBar
    For Each cBar In CommandBars
         cBar.Enabled = True
    Next
    Application.DisplayFormulaBar = True
End Sub
```

SUMÁRIO

Os botões que estão nas suas barras de ferramentas são aqueles da maioria das ferramentas mais importantes que você tem no Excel. Eles salvarão você, toda vez que você os usar. Por esta razão, você deverá personalizar suas barras de ferramentas e colocar nelas as ferramentas que você mais gosta de usar, e remover aquelas que você não gosta de usar.

EXEMPLOS DE BARRAS DE FERRAMENTAS

Usando Barras de Ferramentas Anexadas

No Excel você pode criar sua própria barra de ferramentas selecionando Exibir, Barra de ferramentas, e usar a opção Nova barra de ferramentas disponível no diálogo que aparece. E, você pode atribuir botões à barra de ferramentas selecionando a opção Personalizar. Se você escolher uma macro categoria de botões, você pode atribuir sua próprias macros aos botões na barra de ferramentas.

Uma tal barra de ferramentas personalizada é tipicamente útil somente para uma pasta particular. Você pode anexar a barra de ferramentas a uma pasta fazendo o seguinte:

Escolher Exibir, Barra de ferramentas, Personalizar
Clique no botão Anexar e selecione a barra de ferramentas personalizada e copie a barra de ferramentas personalizada para a pasta

Se você modificar uma barra de ferramentas anexada, você precisa repetir os passos acima para anexar a nova versão. Caso contrário, as modificações serão perdidas.

Há vários problemas com barras de ferramentas anexadas:

A barra de ferramentas não desaparece quando você fechar a pasta
A barra de ferramentas não aparece quando você abrir a barra de ferramentas
Se você distribuir a pasta aos outros, mudar a barra de ferramentas, e a re-distribuir, os usuários podem não ver a nova barra de ferramentas.

Para resolver estes problemas, use macros como a seguinte, assumindo que sua barra de ferramentas seja chamada "MinhaBarradeFerramentas", e que ela tenha dois botões nela:

```
Sub Auto_Open()
    SetUpButtons
End Sub

Sub SetUpButtons()
     With CommandBars("MinhaBarradeFerramentas ")
          .Visible = True
          .Controls(1).OnAction = "PrimeiraMacro"
          .Controls(2).OnAction = "SegundaMacro"
     End With
End Sub


Sub Auto_Close()
     On Error Resume Next
     CommandBars("MinhaBarradeFerramentas ").Delete
```

```
End Sub
```

Recompondo As Macros Numa Barra de Ferramentas Personalizada

O Excel permite-lhe criar uma barra de ferramentas personalizada e anexá-la a uma pasta. Entretanto, se você der esta pasta a alguém, os botões da barra de ferramentas se referirão de novo ao *path* da pasta original. O resultado é uma caixa asquerosa de erro dizendo que uma macro não pôde ser encontrada. A propriedade botões' OnAction apontará ainda para o local original da pasta, quando as macros forem feitas. O código seguinte ilustra um modo elegante para resolver isto, e ainda funciona com qualquer barra de ferramentas personalizada.

```
Sub Fix_Toolbar
      'o nome da macro acima seria incluido na sua macro
Auto_Open
      'de modo que ela execute cada vez que a pasta for aberta.
      Dim tToolbar As String
      Dim i As Integer
      'coloque o nome da sua barra de ferramentas no lugar do
Nome da barra de ferramentas
      tToolbar = "Nome da barra de ferramentas"
      'rode por cada um dos botões na barra de ferramentas
      For i = 1 To CommandBars(tToolbar).Controls.Count
      'obtenha a OnAction atual, que inclui o path
            nName = CommandBars(tToolbar).Controls(i).OnAction
            're-atribuir o nome da macro excluindo o path
            CommandBars(tToolbar).Controls(i).OnAction = _
                  Right(nName, Len(nName) - InStr(nName, "!"))
      Next i
End Sub
```

Se você fizer as correções a uma barra de ferramentas anexada, você terá de anexá-la novamente, ou as correções não serão salvas

Usando Uma Macro Para Criar Uma Barra De Ferramentas

A macro seguinte ilustra os códigos necessários para se criar uma barra de ferramentas com vários botões nela, com dicas de ferramentas descritivas. A face na barra de ferramentas está configurada pela propriedade FaceID. A macro após isto lhe mostra como encontrar as Ids para mais de 2000 faces de botão.

```
Sub CreateAToolbar()
      Const tBarName As String = "Nome da Barra de Ferramentas"
      'Deletar CommandBar se ela existir
      On Error Resume Next
      CommandBars(tBarName).Delete
      On Error GoTo 0
      'criar CommandBar
      CommandBars.Add Name:=tBarName
```

```
      'definir uma variável objeto para se referir a CommandBar
      With CommandBars(tBarName)
            'adicionar o botão use 1 para especificar uma face
personalizada vazia
With .Controls.Add(ID:=1)
                  .OnAction = "AddInfo"
                  'this adds the smiley FaceID
                  .FaceID = 59
                  .Caption = " Adicionar as Informações do
Relatório"
            End With
            'adicionar o próximo botão com uma barra separadora
            With .Controls.Add(ID:=1)
                  .OnAction = "RemoveInfo"
                  'isto adiciona a borracha FaceID
                  .FaceID = 47
                  .Caption = "Remover as Informações do Relatório"
                  'isto adiciona a barra separadora
                  .BeginGroup = True
            End With
            'mostrar a barra de ferramentas
            .Visible = True
      End With
 End Sub
```

Usando FaceIDs para especificar um ButtonFace da Barra de Ferramentas

A propriedade FaceID de um controle especifica o botão face ou imagem. Por exemplo:

```
CommandBars("Minha barra de ferramentas).Controls(1).FaceID = 80
```

Coloca um botão face com a letra "A" no controle.

A macro seguinte cria uma pasta que lista todos os botões FaceIDs disponíveis:

```
Sub DisplayControlFaces()
      Const tBarName As String = "Temp barra de ferramentas"
      'adiciona uma nova Pasta para as faces
      Workbooks.Add
      'Deleta a CommandBar se ela existir
      On Error Resume Next
      CommandBars(tBarName).Delete
      'cria a CommandBar
      CommandBars.Add Name:=tBarName
      'define uma variável objeto para se referir a CommandBar
      With CommandBars(tBarName)
            'use uma armadilha de erro para manejar as FaceIDs
perdidas no código abaixo
On Error GoTo eTrap
```

```
            'especificar uma ID de um para uma face de botão
personalizada vazia
            With .Controls.Add(ID:=1)
            'muda a imagem do botão através de todos aqueles que
estiverem disponíveis
                  For i = 1 To 5500
                        .FaceID = i
                        'copia a face e cola na planilha
                        .CopyFace
                        ActiveSheet.Paste
                        'grava a face ID
                        With ActiveCell.Offset(1, 0)

.Value = i
                              .HorizontalAlignment = xlLeft
                        End With
                        'aumenta o contador e seleciona a próxima
célula destino
                        J = J + 1
                        If J <= 5 Then
                              ActiveCell.Offset(0, 1).Select
                        Else
                              ActiveCell.Offset(3, -5).Select
                              J = 0
                        End If donext:
                  Next
            End With
      End With
      'remova a commandbar quando ela não for necessária
      CommandBars(tBarName).Delete
      Exit Sub
eTrap:
      'nem todas as FaceIDs existem.  Isto maneja qualquer uma
perdida
      Resume donext
End Sub
```

Colocando Faces de Botões Personalizados Numa Barra de Ferramentas

A macro seguinte cria uma barra de ferramentas personalizada e daí adiciona botões a ela usando faces botão personalizadas e que tenham sido salvas numa folha chamada "Faces de Botão". Para criar faces personalizadas:

Vá para a Caixa de Diálogo Personalizar - Selecione Exibir, Barra de ferramentas, Personalizar

Selecione o botão que você quer editar. Daí clique com o botão direito do mouse nele para mostrar o menu de botão *popup*. Selecione a opção Editar e modifique a face de botão.

Com o botão selecionado clique com o botão direito do mouse no botão e selecione Copiar Face de Botão

Saia da caixa de diálogo personalizar e vá par uma planilha onde você possa armazenar a face do botão. O nome da folha usado neste exemplo é "Faces de Botão" e faz parte da pasta contendo o código da macro.

Nesta folha, apenas faça um normal Editar, Colar. A face de botão será colada na planilha como uma figura objeto. Ela será muito pequena, que é OK. Com o botão selecionado, selecione **Formatar**, Selecione **Estilo...** e remova o frame ao redor do objeto se existir algum.

Quando você selecionar a face do botão, você verá o nome que o Microsoft Excel deu a esta figura de objeto na caixa Nome. Por exemplo, ela pode ser chamada "Figura 1". Clique na caixa nome (adjacente à caixa de edição de fórmula) e re-nomear a figura digitando no novo nome e pressionando entrar. O nome pode ter espaços. Para ajudá-lo no futuro, entre com este nome numa célula adjacente ao botão face.

Até este ponto, você agora pode usar uma face de botão que você armazenou na planilha acima e ter uma macro colada a ela no seu próprio botão. A macro seguinte ilustra a criação de uma nova barra de ferramentas com dois botões personalizados e um botão regular. As faces para os botões personalizados são chamadas "adicionar botão" e "mudar botão", e estão localizadas numa folha chamada "faces de botão" na pasta contendo este código.

```
Sub ToolBarWithCustomFace()
      Const tBarName As String = "Nome da Barra de Ferramentas"
      'Deletar a CommandBar se ela existir
      On Error Resume Next
      CommandBars(tBarName).Delete
      On Error GoTo 0
      'cria a CommandBar
      CommandBars.Add Name:=tBarName
      'defina um objeto variável para se referir à CommandBar
      With CommandBars(tBarName)
            'adicione botão use 1 para especificar uma face
personalizada vazia
With .Controls.Add(ID:=1)
                  .OnAction = "Adicionar Info"
                  .Caption = "Adiciona Informação de Relatório"
                  'isto adiciona uma face personalizada ao botão
                  ThisWorkbook.Sheets("Faces de Botão")
                  .Shapes("Figura").Copy
                  .PasteFace
            End With
            'mostra a barra de ferramentas
            .Visible = True
      End With
 End Sub
```

Ocultando E Restaurando As Barras De Ferramentas E Menus

O código seguinte ocultará todas as barras de ferramentas e menus:

```
Dim c
For Each c In CommandBars
     c.Enabled = False
Next
```

O código acima não somente ocultará todas as barras de ferramentas e menus, ele desabilitará todos os menus, com uma exceção. Isto é a barra de ferramentas que aparece se clicar com o botão direito do mouse num botão da barra de ferramentas – se o usuário não fez *upgrade* ao Excel SR-2.

Para mostrar uma barra de ferramentas específica ou menu, vocÊ primeiro tem de configurar sua propriedade *Enabled* de volta para *True*. Se não estiver visível, você também terá de configurar a propriedade *Visible* para True. Por exemplo:

```
With CommandBars("Meu Menu Personalizado")
     .Enabled = True
     .Visible = True
End With
```

Para mostrar novamente os menus e as barras de ferramentas quando o usuário as tiver, use o código seguinte:

```
Dim c
For Each c In CommandBars
     c.Enabled = True
Next
```

Você pode também desabilitar menus específicos referindo-se a eles pelo nome:

```
CommandBars("Worksheet Menu Bar").Enabled = False
```

Como Evitar Seus Botões da Barra de Ferramentas Personalizada De Aparecerem Desbotados

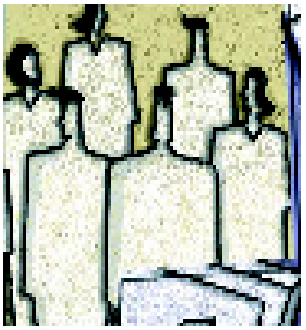
Se você construir uma barra especial de ferramentas de botões usando o editor de botão da barra de ferramentas, os botões podem aparecer desbotados na máquina de outros usuários. Este fenômeno ocorre quando os botões da barra de ferramentas forem construídos com o conjunto de cores do Windows para "milhões de cores" e o usuário com a barra de ferramentas desbotada, tiver o seu conjunto de cores do Windows numa configuração mais baixa. A solução é usar uma configuração de cores mais baixa para mostrar.

Adicionando Dicas das Ferramentas Aos Botões

Quando você criar uma barra de ferramentas, a dica do botão ferramenta é o *caption* que está atribuído ao botão. Por exemplo:

```
With CommandBars("Minha Barra de Ferramentas")
      .Controls(1).Caption ="Carregar novos dados"
End with
```

Lição 12: Os Suplementos



Se você quiser distribuir macros e aplicações que você escreveu, v provavelmente vai querer considerar em distribuí-las como um arqu suplemento em vez de um arquivo pasta. A principal vantagem de um arqu suplemento é que ele é mais fácil para usar funções. Uma outra vantagem, suplemento pode ser armazenado em qualquer diretório e ajustado para ser carrega quando o Microsoft Excel iniciar. Isto elimina a necessidade de colocar o arquivo diretório de início do Microsoft Excel (tipicamente o *Excel\Xlinicio*). E, os suplemen não solicitam se serão ou não gravados quando se fecha o Microsoft Excel. Tambe arquivos suplementos podem ter neles as macros AUTO_OPEN e AUTO_CLO Tais macros podem ser usadas para se criarem e removerem itens de menu e barras de ferramentas.

Se você quiser trocar macros com muitos outros usuários e que estão numa LAN ou sistema de redes, você provavelmente vai querer usar um suplemento em vez de uma pasta. Um suplemento pode ser aberto por muitos usuários sem qualquer problema. Entretanto, uma pasta pode somente ser aberta por um usuário de cada vez ao mesmo tempo sem aparecer uma mensagem de advertência.

Existem algumas desvantagens com suplementos. A maior é o sofrimento de descarregar, editar a pasta fonte, recriar o suplemento, descarregar a pasta, e recarregar o suplemento. Por favor, note que você pode evitar isto se você mudar as propriedades da pasta suplemento para True ao invés de criar um suplemento da sua pasta.

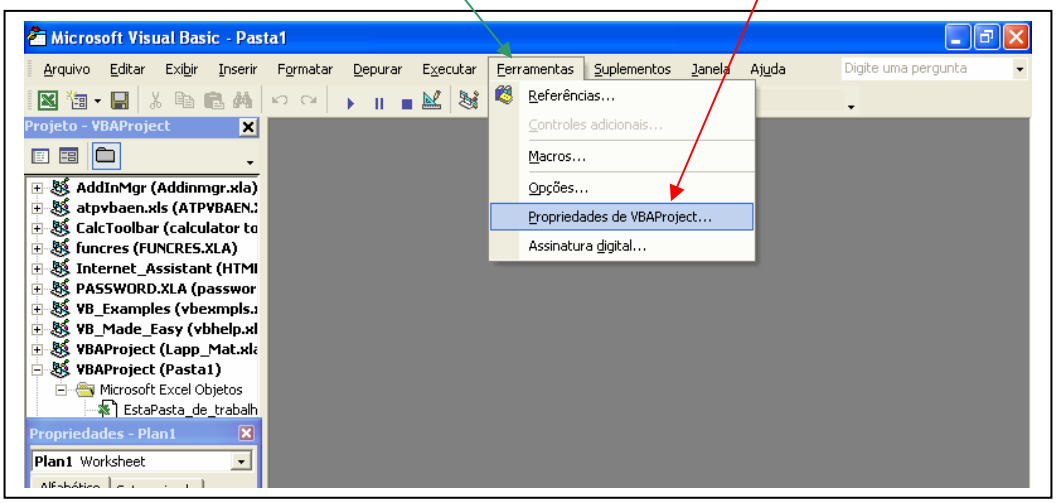
Um outro é que a maioria dos usuários não está familiarizada com eles e também como carregá-los. Um outro ainda é que as macros no suplemento não aparecem em qualquer lista de macros, o que torna difícil atribuir manualmente uma macro suplemento a um botão ou rodar. Assim seu suplemento deverá adicionar itens de menu ou criar barra de ferramentas automaticamente via um Auto_Open ou uma macro Workbook_Open. Também, você não pode mostrar qualquer uma das folhas num suplemento. Assim, você não pode mostrar gráficos ou editar manualmente uma planilha num suplemento. Se você precisar modificar freqüentemente suas macros ou adicionar algumas outras novas à sua coleção, você não deverá usar um suplemento, pois você não pode modificar as macros de um suplemento. Você necessitaria carregar o arquivo contendo o código toda vez que você precisar modificar as macros e daí re-criar o arquivo suplemento para ele ser alterado.

Por favor, note que criar um suplemento não evita um usuário de ver seu código macro. Torna-se apenas mais difícil de obtê-lo.

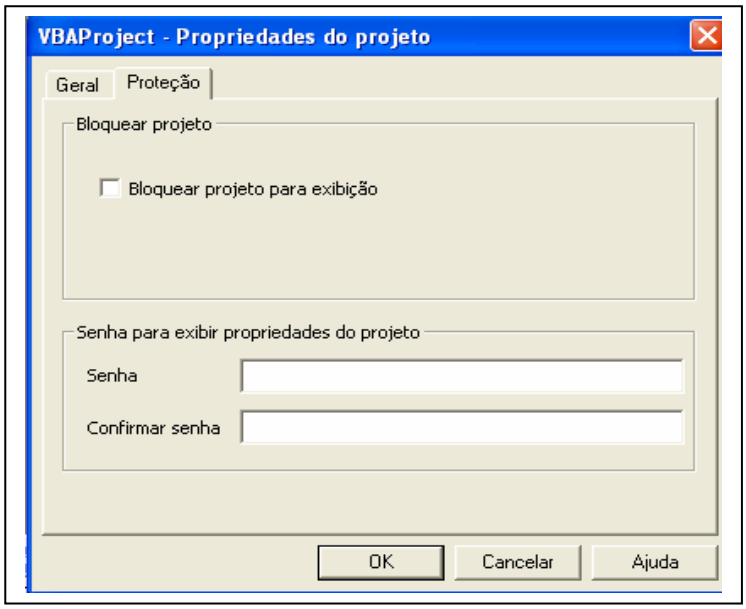
COMO CRIAR UM ARQUIVO SUPLEMENTO

Existem dois modos de se criar suplementos que eu chamei de Suplemento Aproximação Um e Dois. Entretanto, para evitar que os outros vejam o seu código suplemento, você deverá fazer o seguinte:

- No editor de VB, selecione **Ferramentas**, **Propriedades de VBAProject...**



- Selecionar a alça **Proteção** da janela **Propriedades do projeto**.

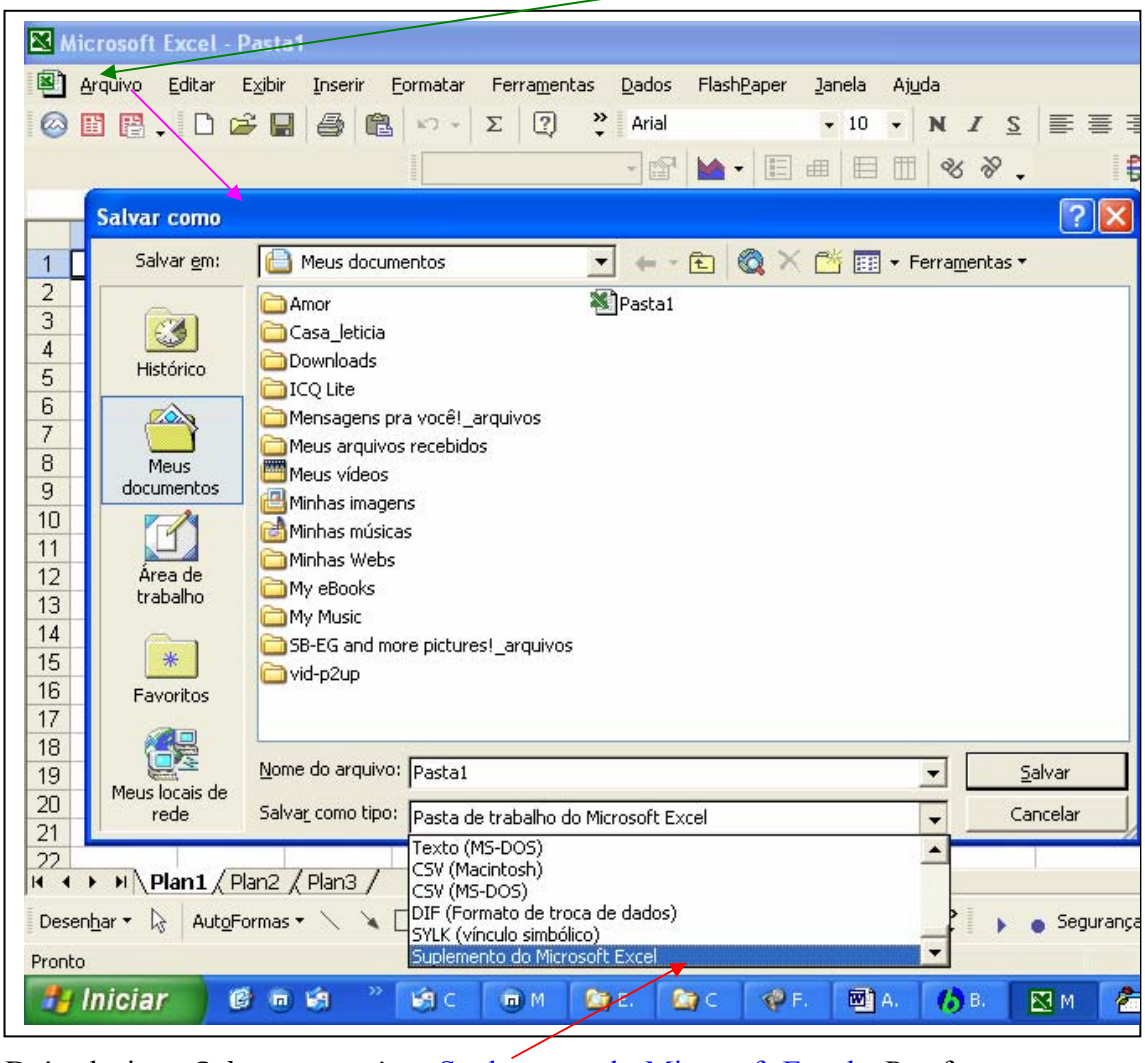


- Clicar na caixa **Bloquear projeto par exibição**.
- Especificar uma **Senha** (*password*) e após confirmá-la.

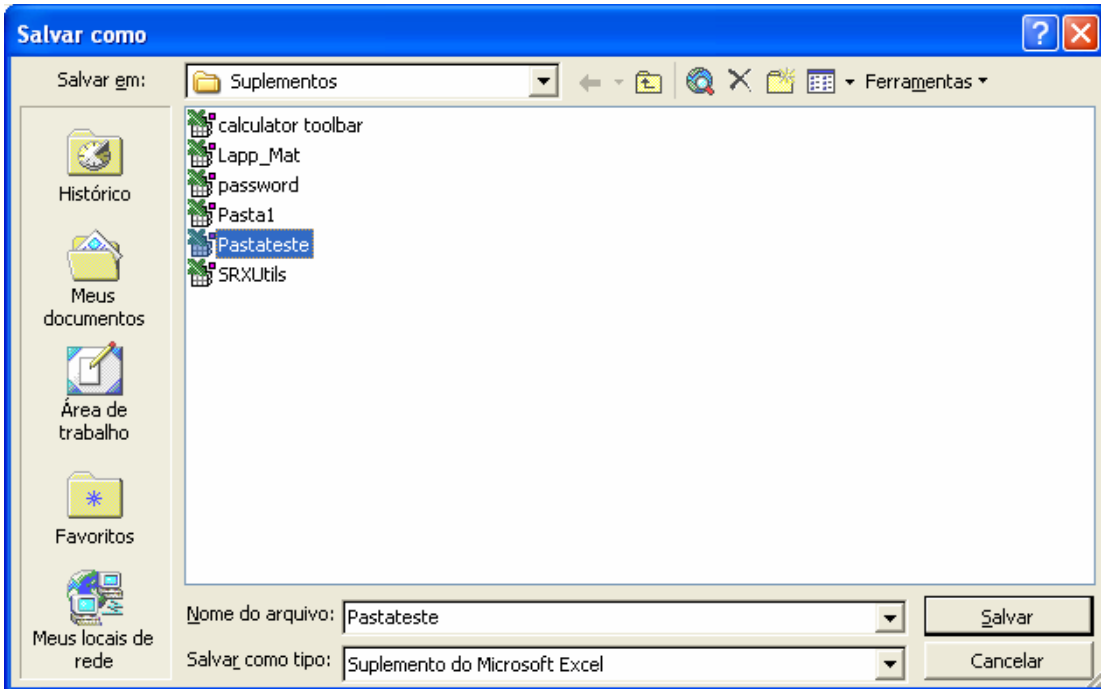
Por favor note que uma pessoa pode comprar um cracker de senhas e abrir seu código.

Suplemento - Aproximação Um

Vá para uma planilha na pasta e lance os comandos **Arquivo**, **Salvar como...**.



Daí selecione *Salvar como tipo:* Suplemento do Microsoft Excel. Por favor, note que você precisa compilar seu arquivo primeiro no Editor VBE, quando estiver salvando o arquivo como um arquivo XLA não o verifique se têm erros. Você acabará com dois arquivos, um do tipo XLS e um do tipo XLA.

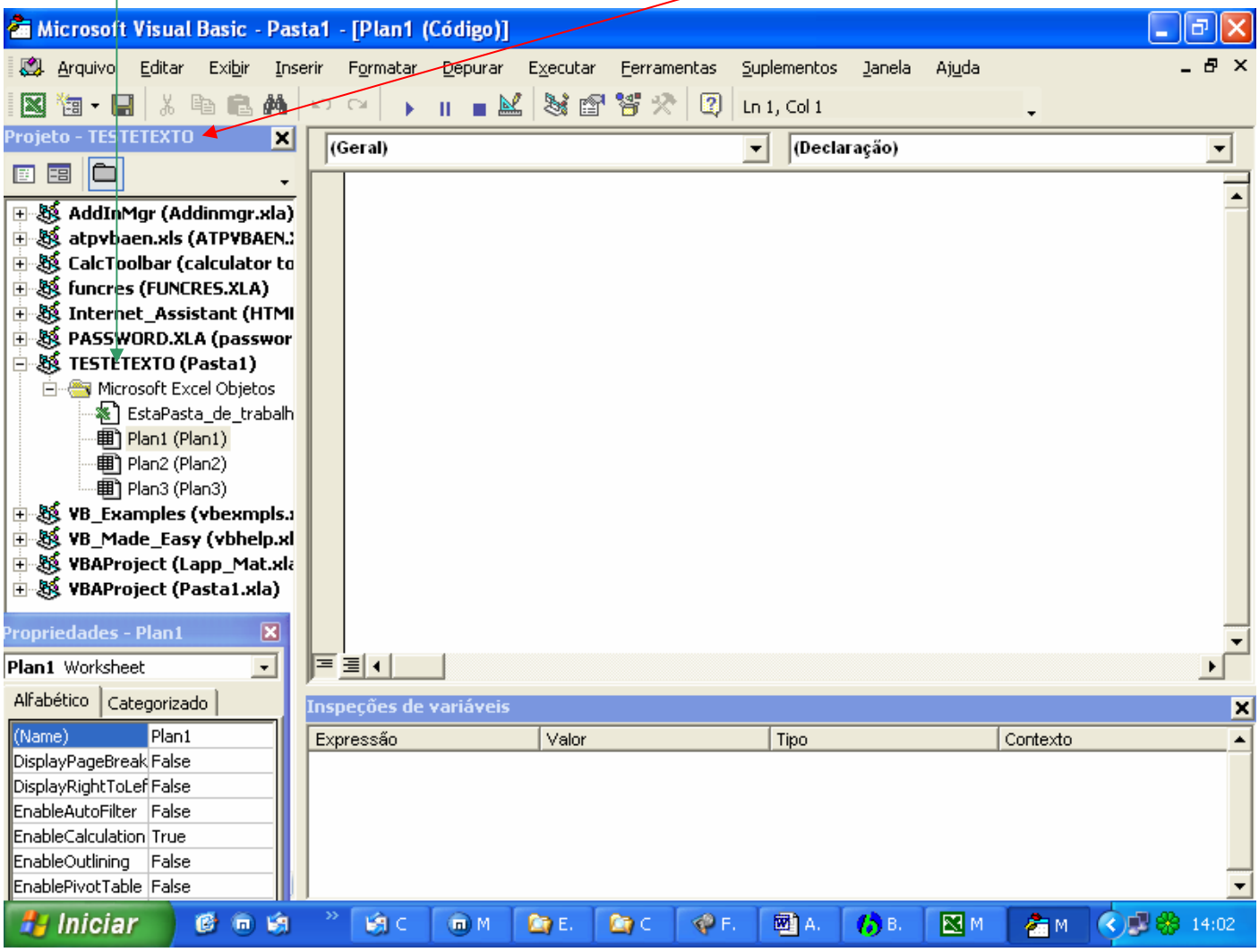


O próximo passo é especificar o diretório para o arquivo suplemento. Você pode salvar um suplemento em qualquer diretório. Você provavelmente vai querer iniciar um diretório especial para suplementos ou salvar seus suplementos no diretório *C:\Documents and Settings\nome do usuário principal do Windows\Dados de aplicativos\Microsoft\Suplementos*. Este é o diretório *default* que a Microsoft Excel dispõe para você selecionar suplementos.

Uma vez tendo especificado o nome do arquivo e o diretório, selecionar Salvar. O Microsoft Excel criará então o arquivo suplemento que eu denominei *Pastateste*. Ele será ligeiramente menor em tamanho que o arquivo fonte. Tudo que se faz para criar um suplemento é tornar um arquivo em suplemento. Ele não carrega o arquivo no Microsoft Excel.

Suplemento - Aproximação Dois

Na janela *Project Explorer* do Editor VB, aparece o título Projeto – TESTETEXTO. Nela selecione o objetoTESTETEXT (Pasta1):



e mostre também a *janela de propriedades*.

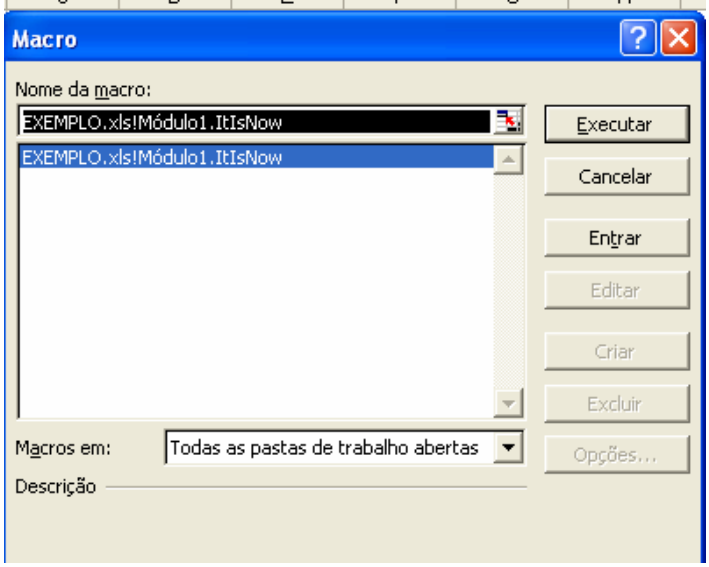
Na *janela de propriedades*, ajuste o valor da propriedade "IsAddin" para True. Isto fará qualquer planilha ficar oculta. Salve o arquivo. Por favor note que o tipo de arquivo não mudará.

A vantagem desta segunda aproximação é que você não precisa manejar dois arquivos, um do tipo XLS e um do tipo XLA. Para converter o arquivo de volta para um arquivo tipo XLS, simplesmente mude a propriedade "IsAddin" para False. A desvantagem é que o tipo de arquivo não é XLA, o qual é o que o gerenciador de suplemento reconhece.

Para ilustrar a criação de um arquivo suplemento, crie uma pasta com um modulo que contenha a seguinte macro e função. Este suplemento também será usado em vários outros exercícios neste texto. O código para a macro e função é:

```
Function LbsToKg(lbs)
    LbsToKg = lbs / 2.2046
End Function

'**************************************************

Sub ItIsNow()
    MsgBox "O dia e o tempo é " & Now
End Sub

'**************************************************
```

Salve o arquivo como EXEMPLO.XLS em Meus Documentos (se você salvar de outro lugar, anote o *path*). Agora, vá para uma nova pasta e selecione Ferramentas, Macro, e rode a macro *ItIsNow*. Você a verá listada na lista das macros disponíveis.



Finalmente, vá a uma célula numa pasta diferente e teste a função. Você precisará especificar a pasta para usar a função. Assumindo que você queira ver quantos kilogramas são100 lbs, a fórmula seria:

```
=EXEMPLO.xls!LbsToKg(100)
```

Como um lembrete, para evitar ter que colocar na pasta uma referência para um arquivo você pode configurar uma referência ao arquivo função. Isto é feito indo para uma folha de modulo na pasta e selecionando Ferramentas, Referências e especificando o arquivo. Não faça isto neste exercício. Isto foi apenas um lembrete!

O próximo passo neste exercício é criar o arquivo suplemento.

COMPILANDO SUAS MACROS

Compilar seu código é muito fácil - apenas selecione Depurar, Compilar VBAProject. O nome do seu projeto de pasta será mostrado imediatamente após a palavra Compilar VBAProject no menu. Uma técnica muito útil é nomear cada um dos projetos, e não deixá-los para serem nomeados pelo "VBAProject". O Visual Basic quando necessário ajustará o código a ser compilado para outro além daquela pasta cujos códigos são mostrados na janela ativa. Se todas as suas pastas tem o nome do projeto *default*, você não pode ter certeza que você está compilando a pasta certa.

MODIFICANDO SUAS MACROS PARA USAR NO SUPLEMENTO

Se suas macros fizerem referências às suas pastas fontes, então elas devem ser modificadas antes do suplemento ser criado ou então resultarão erros destas referências. Por exemplo, se houver uma macro no EXEMPLO.XLS que tenha a referência Pasta("Exemplo.XLS"), e você criou um suplemento com um tipo de XLA, quando esta declaração é rodada o suplemento virá com uma mensagem de erro. Para evitar tais erros, você precisará modificar as macros para ou especificar o arquivo suplemento ou usar a palavra chave *ThisWorkbook* ao invés disso. *ThisWorkbook* é um comando que se refere ao arquivo que contém a macro em execução. Se o *Exemplo*.XLA teve este problema, então as referências terão de serem mudadas ou para *Workbooks*("Exemplo.XLA") ou *ThisWorkbook*. O uso do *ThisWorkbook* é, obviamente, o melhor caminho.

Também, se você tem botões ou itens de menu numa pasta que se refira ao arquivo XLS e uma de suas macros, você precisará mudar o link naquela pasta para se referir ao arquivo suplemento. Você pode fazer isto manualmente selecionando Editar, Vinculos e especificando o arquivo XLA para o arquivo XLS. Por favor note que você precisará movimentar-se nestes links para trás e para frente quando você editar no arquivo XLS. Assim, você provavelmente vai querer gravar macros que façam estas mudanças para você.

CARREGANDO UM ARQUIVO SUPLEMENTO

Existem vários modos de se carregar um arquivo suplemento. O modo mais rápido é selecionar Arquivo, Abrir e especificar o arquivo suplemento. Entretanto, ele tem várias pequenas desvantagens:

- Você terá que abrir manualmente o arquivo cada vez
- Você não pode fechar manualmente o arquivo sem fechar o Microsoft Excel

A principal vantagem desta aproximação é que ela não exige que você tome ação para evitar o suplemento de ser carregado na próxima vez que você carregar o Microsoft Excel. Se você usar somente um arquivo suplemento não freqüentemente, então você provavelmente vai preferir esta aproximação.

Você também pode ter uma macro abrindo e fechando um arquivo suplemento. Você usará os mesmos comandos que você usa com pastas regulares. Por exemplo:

```
Workbooks.Open "\ESPECIAL.XLA"
Workbooks("ESPECIAL.XLA").Close
```

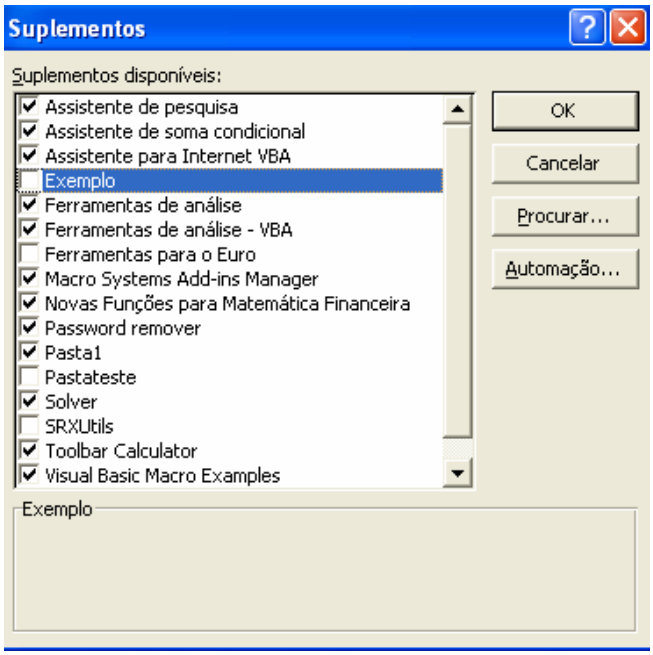
Por favor, note que você precisa especificar o *path* para o arquivo suplemento, caso contrário o Excel enxergará somente o diretório atual e um erro ocorrerá se o arquivo não estiver lá.

Você pode também usar argumentos com o método Open. Por exemplo, se o suplemento tem vínculos a outros arquivos que você não quer alterar quando o suplemento for carregado, você pode adicionar o argumento updatelinks:=0 para evitar atualização. E, rodar qualquer macro Auto_Open no suplemento, você usará uma declaração como:

```
Workbooks("SPECIAL.XLA").RunAutoMacros
xlAutoOpen
```

Se o suplemento tem uma macro Workbook_Open, ele roda automaticamente quando aberto via código.

Um outro modo para abrir um arquivo suplemento é selecionar Ferramentas, Suplementos do menu Microsoft Excel. Isto mostra a seguinte caixa de diálogo:



Você verá vários suplementos listados. Sua lista pode ser diferente. Uma curta descrição de cada suplemento é mostrada na lista de rolamento. Você não encontrará o EXEMPLO.XLA se ainda estiver na pasta exemplo. É bom sair dela primeiro e daí

executar os passos acima de outra pasta. Ao clicar num suplemento serão feitas duas coisas:

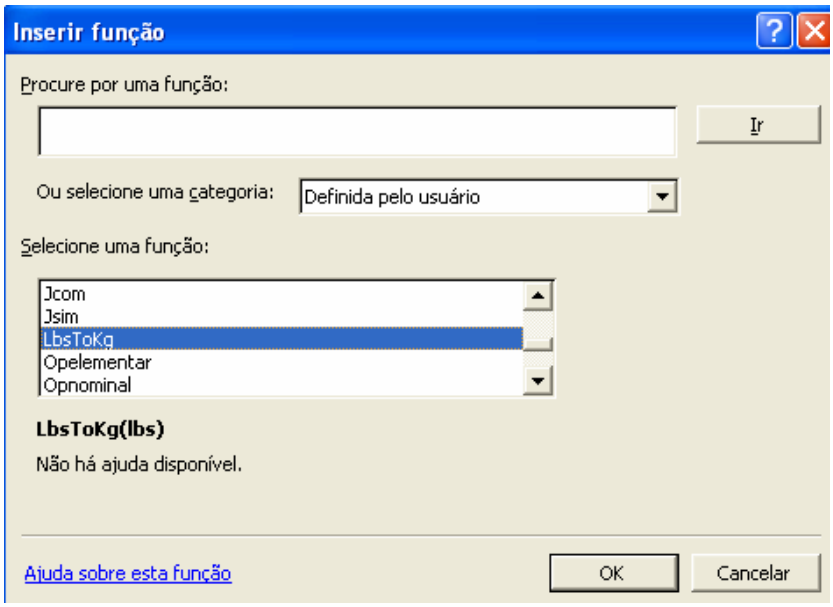
- Ele muda o suplemento de desligado para ligado ou de ligado para desligado
- Ele mostra uma longa descrição do suplemento, o qual vem da informação do arquivo resumo.

Se você clicou num suplemento e ele não for um selecionado previamente, ele será carregado quando você selecionar OK, e cada vez que você carregar o Microsoft Excel daí pra frente. Clicando num suplemento numa segunda vez reverte o efeito da primeira seleção.

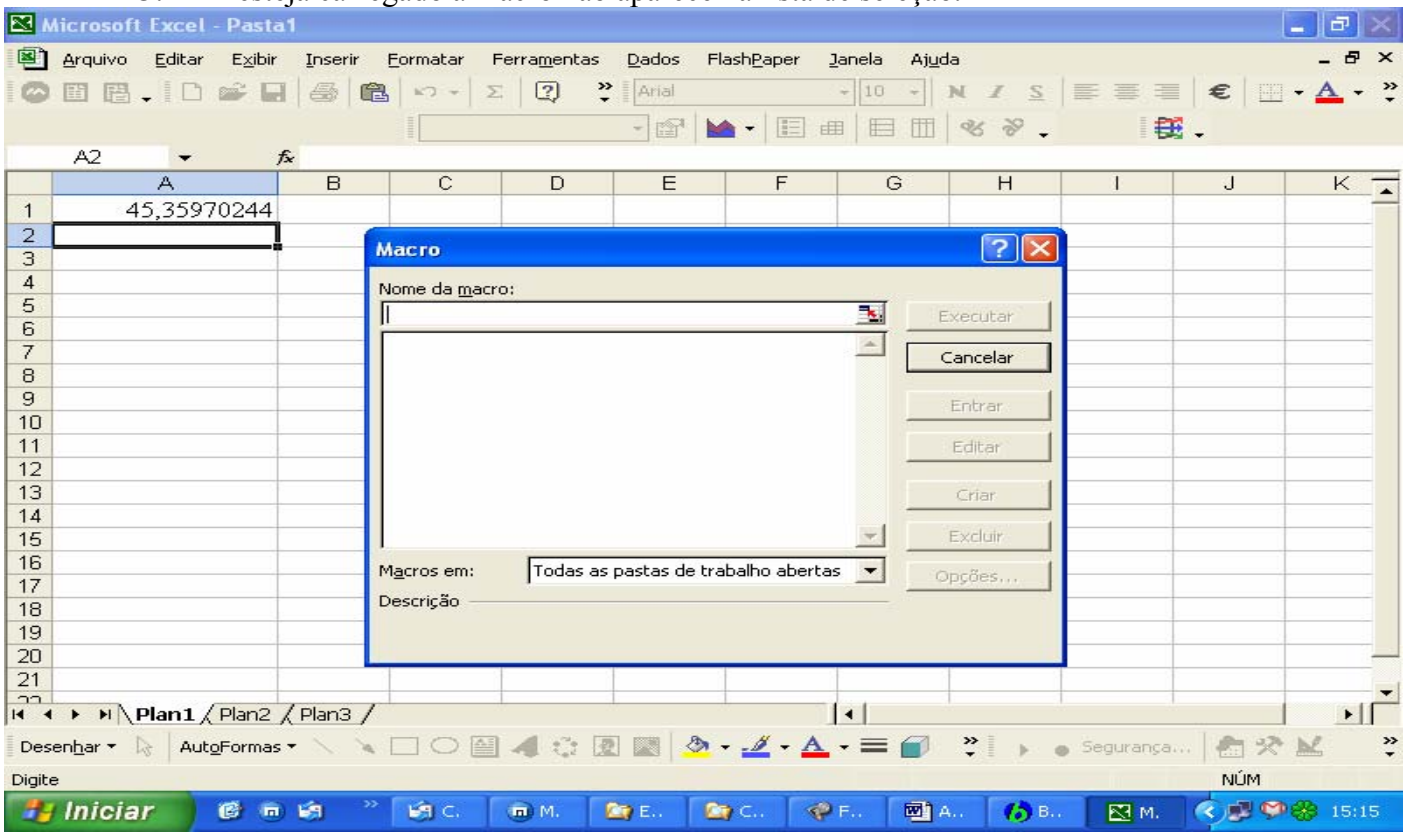
Para adicionar um suplemento à lista, você precisa selecionar Procurar... e daí encontrar e clicar no arquivo suplemento. Quando você selecionar o arquivo suplemento, ele será adicionado à lista e a caixa ativa será checada.

Se você criou o arquivo EXEMPLO.XLA anteriormente, então você pode adicioná-lo à lista usando a opção Procurar.... Quando você selecionar OK, o arquivo EXEMPLO.XLA será carregado. Note que somente o nome do arquivo aparece na lista de suplementos e que não há uma longa descrição associada com o suplemento. Uma seção mais tarde deste texto ilustra como fornecer estas descrições.

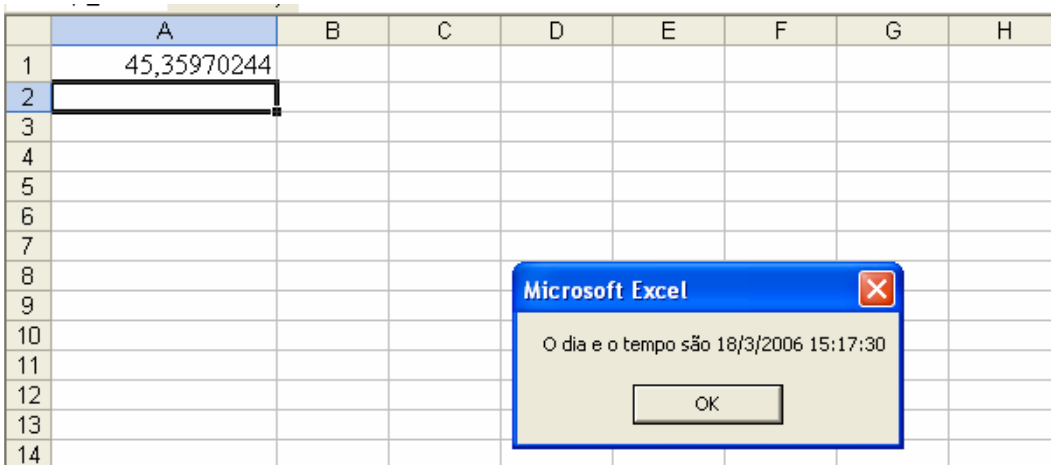
Para ilustrar a diferença entre suplementos e pastas de arquivos, feche o EXEMPLO.XLS. Neste ponto do exercício, você deve ainda ter um arquivo teste para abrir. Se não, abrir um novo arquivo. Para demonstrar a função chamada LbsToKg que está no EXEMPLO.XLA, vá para uma célula vazia e digite =LbsToKg(100) e pressione enter. Note que você não precisou especificar o nome do arquivo nem configurar uma referência. Também, se você selecionar **Inserir**, **Função**, e clicar na categoria Definida pelo usuário, você verá LbsToKg listada lá.



Agora, selecione Ferramentas, Macro e procure a macro *ItIsNow* que você criou no EXEMPLO.XLS, e que está no suplemento, EXEMPLO.XLA. Embora o EXEMPLO.XLA esteja carregado a macro não aparece na lista de seleção.



Entretanto, se você digitar seu nome na caixa de nome de macro e pressionar enter, a macro roda e diz a você a hora do dia.



Se você quiser atribuir uma macro suplemento a um botão da barra de ferramentas, você também terá de digitar no nome da macro, como macros suplemento não são mostrados na lista de seleção da macro. Porque a Microsoft mostra as funções dos suplementos e não as macros, não tem lógica. Entretanto, este é o modo como os suplementos funcionam.

DESCARREGANDO UM ARQUIVO SUPLEMENTO

Você pode descarregar um arquivo suplemento que você carregou selecionando a barra de ferramentas, Suplementos, e clicar na macro que você quiser descarregar. Isto removerá a marca na caixa de opção pelo seu nome ou descrição. Você não pode descarregar manualmente um suplemento que você carregou usando Arquivo, Abrir.

Se você descarregar um suplemento e você tiver que abrir arquivos que estão usando uma função que está armazenada no suplemento, a função não funcionará. Em vez disso, o Microsoft Excel mostrará #NAME? como o valor daquela célula contendo a função e em qualquer célula que se referir a ela. Se a função é exclusiva da pasta que usará, então você deverá armazenar a função naquela pasta em vez de um arquivo suplemento ou uma outra pasta.

ALTERANDO UM ARQUIVO SUPLEMENTO

Suponha que você queira modificar as macros ou funções que você tem num arquivo suplemento. Para fazer isto, você precisa

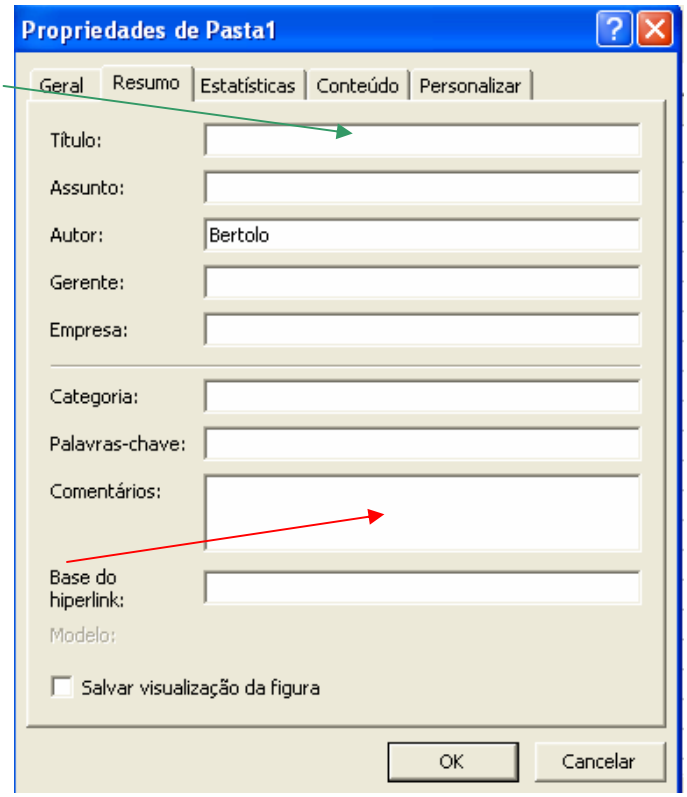
- descarregar o arquivo suplemento,
- abrir a pasta contendo as macros,
- modificar as macros,
- re-criar o arquivo suplemento.
- recarregar o arquivo suplemento

Se você tentar recriar o arquivo suplemento sem primeiro descarregá-lo, o Microsoft Excel mostrará uma mensagem.

ADICIONANDO DESCRIÇÕES PARA OS SEUS SUPLEMENTOS

Se você quiser mudar ou a descrição curta que aparece na lista de seleção suplemento ou na longa descrição que aparece na caixa descrição abaixo na lista de seleção suplemento, você precisa:

- Descarregar o suplemento se ele estiver carregado
- Abra o XLS arquivo que criou o suplemento
- Selecionar Arquivo, Propriedades do menu Microsoft Excel
- Preencha a caixa de títulos e a caixa de comentários.



- Salvar o arquivo XLS
- Re-criar o arquivo suplemento
- Re-load o arquivo suplemento
- Encerre o Microsoft Excel e retorne

A entrada caixa de títulos é a descrição que aparece na lista de seleção suplemento. A entrada comentário é a longa descrição. Afim de novas descrições aparecerem no diálogo seleção suplemento, você deve primeiro sair do Microsoft Excel e daí carregá-lo novamente.

SUPLEMENTOS E BOTÕES DA BARRA DE FERRAMENTAS

Se você criou botões da barra de ferramentas que se refiram a macros num arquivo XLS, e você criou um suplemento, os botões continuarão a se referirem ao arquivo XLS. Você precisará mudar manualmente os botões para referirem ao arquivo XLA. Por outro lado, quando você clicar num botão, o Excel abrirá o arquivo XLS, e ignorará a macro suplemento.

DELETANDO SUPLEMENTOS DA LISTA DE SUPLEMENTO

Embora o Microsoft Excel forneça um significado ao adicionar suplementos para a lista de suplementos disponíveis, ele não fornece uma maneira para deletá-los desta lista. A única maneira que eu encontrei para fazer isto é fechar o Microsoft Excel, deletar o arquivo suplemento usando Gerenciador de Arquivo, e reiniciar o Microsoft

Excel. Quando o Microsoft Excel iniciá-lo de volta remova qualquer suplementos ele pode não se encontrar na lista. Se o suplemento é selecionado para ser carregado, você será estimulado a removê-lo da lista.

RESUMO

Suplementos podem ser muito úteis se você precisar distribuir funções. Se você usar suplementos, você deverá ter o suplemento ou criar uma barra de ferramentas ou adicionar um menu ou itens de menu aos menus do Excel de modo que o usuário possa rodar as macros. Isto é necessário desde que as macros num suplemento não aparecem na caixa de diálogo de execução de macro (Barra de ferramentas, Macros, …).

THE

END