Pesquisado para: "Exemplo de Mensagem"

Resultados 1-25 de acima 100 Página: **[1]**

Exemplo da função IRR

Exemplo da propriedade Number

Exemplo da função PPmt

Exemplo da instrução On Error

Exemplo da função IPmt

Exemplo da função MsgBox

Exemplo da função MIRR

Exemplo do objeto Collection

Como fazer: visão geral da interface do usuário da Faixa de Opções
Ajuda > Referência de Biblioteca de Objetos para o Microsoft Office System 2007 > Conceitos

Usando funções de planilha do Microsoft Excel no Visual Basic
Ajuda > Conceitos > Eventos, funções de planilha e formas

Trabalhando com formas (objetos de desenho)
Ajuda > Conceitos > Eventos, funções de planilha e formas

Exemplo de aprimoramentos de cor e borda

Exemplo das propriedades de seleção de texto

Como fazer: adicionando e gerenciando barras de menus e itens de menus
Ajuda > Referência de Biblioteca de Objetos para o Microsoft Office System 2007 > Como...

Valores de erro de célula
Ajuda > Conceitos > Células e intervalos

Como fazer: fazendo alterações em balões durante o tempo de execução
Ajuda > Referência de Biblioteca de Objetos para o Microsoft Office System 2007 > Conceitos

Exemplo da função NPV

Exemplo de instruções Deftype

Exemplo da propriedade HelpContext

Exemplo da propriedade HelpFile

Exemplo da função DatePart

Exemplo da função DateDiff

Exemplo do método Add

Exemplo do objeto Err

Exemplo da função DateAdd

Resultados 1-25 de acima 100 Página: **[1]**

Exemplo da função IRR

Neste exemplo, a função **IRR** retorna a taxa interna de retorno para uma série de 5 fluxos de caixa contidos na matriz `Valores()`. O primeiro elemento da matriz é um fluxo de caixa negativo que representa os custos iniciais do negócio. Os 4 fluxos de caixa restantes representam fluxos de caixas positivos para os quatro anos subseqüentes. `Estimativa` é a taxa interna de retorno estimada.

```
Dim Estimativa, Formato, TaxaDeRetorno, Mensagem, MensagemFinal
Static Valores(5) As Double    ' Define a matriz.
Estimativa = .1    ' Inicia a estimativa em 10 por cento.
Formato = "#0.00"    ' Define o formato percentual.
Valores(0) = -70.000    ' Custo inicial do negócio.
' Fluxo de caixa positivo refletindo a receita por quatro anos
' sucessivos.
Valores(1) = 22.000 : Valores(2) = 25.000
Valores(3) = 28.000 : Valores(4) = 31.000
TaxaDeRetorno = IRR(Valores(), Estimativa) * 100    ' Calcula a taxa
interna.
Mensagem = "A taxa interna de retorno para estes cinco fluxos de caixa
é"
MensagemFinal = Mensagem & Format(TaxaDeRetorno, Formato) & " por
cento."
MsgBox Mensagem    ' Exibe a taxa interna de retorno.
```

Exemplo da função IPmt

Este exemplo utiliza a função **IPmt** para calcular quanto de um pagamento são juros quando todos os pagamentos têm valor igual. São fornecidos a taxa percentual dos juros por período (`TPA / 12`), o período de pagamento para o qual se deseja a parte dos juros (`Periodo`), o número total de pagamentos (`NumTotalPagamentos`), o valor presente ou principal do empréstimo (`PVal`), o valor futuro do empréstimo (`ValFut`) e um número que indica se o pagamento vence no início ou no final do período de pagamento (`TipoPagamento`).

```
Dim ValFut, Formato, ValPres, TPA, NumTotalPagamentos, TipoPagamento,
Periodo, IntPmt, JuroTotal, Mensagem, Mensagem1
Const FINALPERIODO = 0, COMECOPERIODO = 1    ' Quando os pagamentos
são feitos.
ValFut = 0    ' Normalmente 0 para um empréstimo.
Formato = "###,###,##0.00"    ' Define o formato monetário.
ValPres = InputBox("Quanto você quer pegar emprestado?")
TPA = InputBox("Qual é a taxa de porcentagem anual do seu
empréstimo?")
If TPA > 1 Then TPA = TPA / 100    ' Certifica-se de que se trata do
                                        ' formulário apropriado.
NumTotalPagamentos = InputBox("Quantos pagamentos mensais?")
TipoPagamento = MsgBox("Você faz os pagamentos no fim do mês?",
vbYesNo)
If TipoPagamento = vbNo Then TipoPagamento = COMECOPERIODO Else
TipoPagamento = FINALPERIODO
For Periodo = 1 To NumTotalPagamentos    ' Totaliza os juros.
    IntPmt = IPmt(TPA / 12, Periodo, NumTotalPagamentos, -ValPres,
ValFut, TipoPagamento)
    JuroTotal = JuroTotal + IntPmt
Next Periodo
Mensagem1 = "Você vai pagar um total de " & Format(JuroTotal, Formato)
Mensagem = Mensagem1 & " em juros por este empréstimo."
MsgBox Mensagem    ' Exibe os resultados.
```

Exemplo da função MsgBox

Este exemplo utiliza a função **MsgBox** para exibir uma mensagem de erro crítico em uma caixa de diálogo com os botões **Sim** e **Não**. O botão **Não** é especificado como a resposta padrão. O valor retornado pela função **MsgBox** depende do botão escolhido pelo usuário. Este exemplo supõe que `DEMO.HLP` é um arquivo de **Ajuda** que contém um tópico com um número de contexto da **Ajuda** igual a `1000`.

```
Dim Mensagem, Estilo, Titulo, Help, Ctxt, Resposta, MinhaString
Mensagem = "Deseja continuar?"    ' Define a mensagem.
Estilo = vbYesNo + vbCritical + vbDefaultButton2    ' Define os
botões.
Titulo = "Demonstração de MsgBox"    ' Define o título.
Help = "DEMO.HLP"    ' Define o arquivo de Ajuda.
Ctxt = 1.000    ' Define o contexto do
        ' tópico.
        ' Exibe a mensagem.
Resposta = MsgBox(Mensagem, Estilo, Titulo, Help, Ctxt)
If Resposta = vbYes Then    ' O usuário escolheu Sim.
    MinhaString = "Sim"    ' Executa alguma ação.
Else    ' O usuário escolheu Não.
    MinhaString = "Não"    ' Executa alguma ação.
End If
```

Exemplo da função MIRR

Este exemplo utiliza a função **MIRR** para retornar a taxa interna de retorno modificada de uma série de fluxos de caixa contidos na matriz Valores(). JurosTPA representa os juros financeiros e TPAInv representa a taxa de juros recebida no re-investimento.

```
Dim JurosTPA, TPAInv, Formato, TaxaDeRetorno, Mensagem, MensagemFinal
Static Valores(5) As Double    ' Define a matriz.
JurosTPA = .1    ' Taxa de empréstimo.
TPAInv = .12    ' Taxa de re-investimento.
Formato = "#0.00"    ' Define o formato monetário.
Valores(0) = -70.000    ' Custo inicial do negócio.
' Fluxo de caixa positivo refletindo a receita por quatro anos
' sucessivos.
Valores(1) = 22.000 : Valores(2) = 25.000
Valores(3) = 28.000 : Valores(4) = 31.000
TaxaDeRetorno = MIRR(Valores(), JurosTPA, TPAInv)    ' Calcula a taxa
interna.
Mensagem = "A taxa interna de retorno modificada para estes cinco
fluxos de caixa é"
MensagemFinal = Mensagem & Format(Abs(TaxaDeRetorno) * 100, Formato) &
"%."
MsgBox MensagemFinal    ' Exibe a taxa de retorno
        ' interna.
```

Exemplo do objeto Collection

Este exemplo cria um objeto **Collection** (MinhasClasses) e, em seguida, cria uma caixa de diálogo na qual os usuários podem adicionar objetos ao conjunto. Para ver como isto funciona, escolha o comando **Módulo de classe** no menu **Inserir** e declare uma variável pública denominada NomeDaOcorrencia de nível de módulo de Class1 (digite **Public** InstanceName) para conter o nome de cada ocorrência. Deixe o nome padrão como Class1. Copie e cole o código a seguir na seção Geral de outro módulo e, em seguida, inicie-o com a instrução ClassNamer em outro procedimento. (Este exemplo funciona somente com aplicativos host que ofereçam suporte a classes.)

```
Sub NomeadorDeClasse()
    Dim MinhasClasses As New Collection    ' Cria um objeto
Collection.
    Dim Num    ' Contador para individualizar chaves.
    Dim Mensagem As String    ' Variável para conter seqüência de
caracteres de aviso.
    Dim ONome, MeuObjeto, ListaDeNomes    ' Variantes para conter
informações.
    Do
        Dim Inst As New Class1    ' Cria uma nova ocorrência da
Class1.
        Num = Num + 1    ' Incrementa Num e, em seguida, obtém um
nome.
        Mensagem = "Insira um nome para este objeto". & Chr(13) _
         & "Pressione Cancelar para ver os nomes no conjunto".
        ONome = InputBox(Mensagem, "Nomeie o conjunto de Itens")
        Inst.InstanceName = ONome    ' Coloca o nome na ocorrência de
                                              ' objeto.
        ' Se o usuário inseriu o nome, adiciona-o ao conjunto.
        If Inst.InstanceName <> "" Then
            ' Adiciona o objeto nomeado ao conjunto.
            MinhasClasses.Add item := Inst, key := CStr(Num)
        End If
        ' Limpa a referência atual em preparação para a seguinte.
        Set Inst = Nothing
    Loop Until ONome = ""
    For Each MeuObjeto In MinhasClasses    ' Cria a lista de nomes.
        ListaDeNomes = ListaDeNomes & MeuObjeto.InstanceName & Chr(13)
    Next MeuObjeto
    ' Exibe a lista de nomes em uma caixa de mensagem.
    MsgBox ListaDeNomes, , "Nomes de ocorrência no conjunto
MinhasClasses"

    For Num = 1 To MinhasClasses.Count    ' Remove o nome do conjunto.
        MinhasClasses.Remove 1    ' Como os conjuntos são reindexados
                ' automaticamente, remove o primeiro
    Next        ' membro de cada iteração.
End Sub
```

Como fazer: visão geral da interface do usuário da Faixa de Opções

Mostrar tudo

Observação

O uso de CommandBars em alguns aplicativos do Microsoft Office foi substituído pela nova interface do usuário baseada na Faixa de Opções. Para obter mais informações, pesquise a Ajuda pela palavra-chave "Faixa de Opções".

O recurso de interface do usuário da Faixa de Opções substitui o sistema atual de menus, barras de ferramentas e painéis de tarefas em camadas por um sistema mais simples de interfaces otimizado para oferecer eficiência e detectabilidade. A nova interface do usuário aprimorou menus de contexto, dicas de tela, uma mini barra de ferramentas e atalhos do teclado que melhoram a eficiência e a produtividade do usuário. A Ribbon Extensibility (ou RibbonX) introduz um modelo inovador que os desenvolvedores podem usar para aprimorar a experiência do usuário. Você usa XML (extensible markup language) e uma das várias linguagens de programação convencionais para manipular os componentes da interface do usuário da Faixa de Opções. Como o XML é texto sem formatação, é possível criar arquivos de personalização em qualquer editor de texto, ou usar o seu editor de XML favorito. Também é possível reutilizar arquivos de interface do usuário personalizada da Faixa de Opções com um mínimo de ajustes, pois cada aplicativo usa o mesmo modelo de programação. Por exemplo, você pode reutilizar os arquivos de interface do usuário personalizada criados no Microsoft® Office Word 2007, no Microsoft Office Excel® 2007, no Microsoft Office Access 2007 ou no Microsoft Office PowerPoint® 2007.

O uso de arquivos de marcação XML para personalizar a interface do usuário reduz sensivelmente a necessidade de suplementos complexos baseados no modelo de objeto **CommandBars**. No entanto, os suplementos gravados em versões anteriores do Office continuam funcionando na interface do usuário da Faixa de Opções com pouca ou nenhuma modificação. Você pode criar uma interface do usuário da Faixa de Opções personalizada no nível de aplicativo usando o Word 2007, o Excel 2007 ou o PowerPoint 2007, da seguinte forma:

- Usando suplementos de COM em código gerenciado ou não gerenciado
- Usando suplementos específicos do aplicativo, como arquivos .ppam e .xlam
- Usando modelos (arquivos .dotm) no Word 2007

Em um cenário típico, o código do suplemento de COM contém procedimentos que retornam marcação XML de um arquivo de personalização externo ou do XML contido no próprio código. Quando o aplicativo é iniciado, o suplemento carrega e executa o código que retorna a marcação XML. O Microsoft Office valida a marcação XML em relação a um esquema XSD e, em seguida, o carrega na memória e o aplica na interface do usuário. A interface da Faixa de Opções é então exibida. Os itens e controles do menu usam procedimentos de retorno de chamada para executar código no suplemento. As personalizações no nível de documento usam a mesma marcação XML e um arquivo de Formatos XML Abertos com uma destas extensões: docx, .docm, .xlsx, .xlsm, .pptx ou pptm. Nesse cenário, você cria um arquivo de personalização que contém a marcação XML e o salva em uma pasta. Em seguida, você modifica as partes no recipiente de Formatos XML Abertos para apontar para o arquivo de personalização. Quando você abrir o documento no aplicativo do Office, o arquivo de personalização será carregado na memória e será aplicado na interface do usuário da Faixa de Opções. Os comandos e controles então chamam o código contido no documento para fornecer a sua funcionalidade.

E sobre as soluções existentes?

Em versões anteriores do Microsoft Office, os desenvolvedores usaram o modelo de objeto **CommandBars** para criar o código do Microsoft Visual Basic® que modificava a interface do usuário. No Office 2007, esse código herdado continua funcionando na maioria dos casos sem modificação. Porém, as alterações feitas nas barras de ferramentas do Office 2003 agora aparecem em uma guia **Suplementos** na versão 2007 do Conjunto de Aplicativos do Microsoft Office. O tipo de personalização que aparece depende do design original do suplemento. Por exemplo, o Office cria um grupo **Comandos de Menu** que contém itens adicionados à estrutura de menus anterior (menus **Arquivo**, **Inserir**, **Ferramentas** e assim por diante). Ele também cria um grupo **Comandos da Barra de Ferramentas** que contém itens adicionados às barras de ferramentas internas anteriores (como as barras de ferramentas **Padrão**, **Formatação** e **Imagem**). Além disso, as barras de ferramentas personalizadas adicionadas por um suplemento ou documento aparecem no grupo **Barras de Ferramentas Personalizadas** da guia **Suplementos**.

Os procedimentos de retorno de chamada adicionam funcionalidade à Faixa de Opções

Com a Ribbon Extensibility, você especifica retornos de chamada para atualizar propriedades e executar ações a partir da sua interface do usuário em tempo de execução. Por exemplo, considere o método de retorno de chamada **onAction** para um botão. A marcação da RibbonX tem esta aparência:

```
<button id="myButton" onAction="MyButtonOnAction" />
```

Esta marcação instrui o Office para chamar a função **MyButtonOnAction** quando o botão for clicado. A função **MyButtonOnAction** possui uma assinatura específica, dependendo da sua escolha de linguagem; aqui está um exemplo no Microsoft Visual C#®:

```
public void MyButtonOnAction (IRibbonControl control)
   {
      if (control.Id=="myButton")
      {
         System.Windows.Forms.MessageBox.Show(“Botão clicado!");
      }
   }
```

Personalizando a interface do usuário da Faixa de Opções com suplementos de COM

A personalização no nível de aplicativo resulta em uma interface do usuário da Faixa de Opções modificada que aparece no aplicativo independentemente do documento que estiver aberto. Essencialmente, você cria suplementos de COM para fazer essas modificações. Para personalizar a interface do usuário da Faixa de Opções usando suplementos de COM:

1. Crie um projeto de suplemento COM. O suplemento criado deve implementar a interface **Extensibility.IDTExtensibility2** como todos os suplementos COM e a interface adicional **IRibbonExtensibility** (encontrada no espaço para nome **Microsoft.Office.Core**).
2. Crie o suplemento e o projeto de instalação. Em seguida, instale o projeto.
3. Inicie o aplicativo do Office. Quando o suplemento for carregado, o evento **IDTExtensibility2::OnConnection** será disparado e inicializará o suplemento, como nas versões anteriores do Office.
4. Em seguida, é chamado o método **QueryInterface** que determina se a interface **IRibbonExtensibility** será implementada.
5. Se ela for implementada, será chamado o método **IRibbonExtensibility::GetCustomUI** que carrega a marcação XML (do arquivo de personalização XML ou da marcação XML incorporada no procedimento) e, em seguida, carrega as personalizações no aplicativo.
6. Finalmente, a interface do usuário personalizada está pronta para o usuário.

Personalizando a interface do usuário da Faixa de Opções com arquivos de Formatos XML Abertos do Office

No nível de documento, o processo de personalização da interface do usuário usando marcação XML envolve as etapas a seguir.

1. Crie o arquivo de personalização em qualquer editor de texto adicionando a marcação XML que inclui novos componentes na interface do usuário da Faixa de Opções, modifica componentes existentes ou oculta componentes. Salve o arquivo como **customUI.xml**.
2. Na sua área de trabalho, crie uma pasta chamada **customUI** e copie o arquivo de personalização nessa pasta.
3. Valide a marcação XML com um esquema de interface do usuário personalizado.

> **Observação**
>
> Esta etapa é opcional.

4. Crie um documento no aplicativo do Office e salve-o como um arquivo de Formatos XML Abertos com uma destas extensões: *.docx*, *.docm*, *.xlsx*, *.xlsm*, *.pptm* ou *.pptx*. Os arquivos que contêm macros

possuem um sufixo *m*. Eles podem conter procedimentos chamados pelos comandos e controles RibbonX.

5. Adicione uma extensão *.zip* ao nome do arquivo do documento e abra o arquivo.
6. Adicione o arquivo de personalização ao recipiente arrastando a pasta para o arquivo.
7. Extraia o arquivo **.rels** para a área de trabalho. Uma pasta **_rels** com o arquivo .rels será copiada para a sua área de trabalho.
8. Abra o arquivo .rels e adicione uma linha que cria uma relação entre o arquivo de documento e o arquivo de personalização e salve o arquivo.
9. Adicione novamente a pasta _rels ao recipiente, substituindo o arquivo existente.
10. Renomeie o arquivo com seu nome original removendo a extensão *.zip*. Quando você abrir o arquivo do Office, a interface do usuário da Faixa de Opções será exibida com a personalização.

Formato geral dos arquivos de marcação XML

É possível usar marcação XML para personalizar a interface do usuário da Faixa de Opções. O exemplo a seguir mostra o formato geral de um arquivo de marcação XML que personaliza essa interface no Word 2007:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon>
    <tabs>
      <tab idMso="TabHome">
        <group idMso="GroupFont" visible="false" />
      </tab>
      <tab id="CustomTab" label="My Tab">
        <group id="SampleGroup" label="Sample Group">
          <toggleButton id="ToggleButton1" size="large" label="Large Toggle
Button" getPressed="MyToggleMacro"  />
          <checkBox id="CheckBox1" label="A CheckBox" screentip="This is a
check box" onAction="MyCheckboxMacro" />
          <editBox id="EditBox1" getText="MyTextMacro" label="My EditBox"
onChange="MyEditBoxMacro"/>
          <comboBox id="Combo1" label="My ComboBox"
onChange="MyComboBoxMacro">
            <item id="Zip1" label="33455" />
            <item id="Zip2" label="81611" />
            <item id="Zip3" label="31561" />
          </comboBox>
          <advanced>
            <button id="Launcher1" screentip="My Launcher"
onAction="MyLauncherMacro" />
          </advanced>
        </group>
        <group id="MyGroup" label="My Group" >
          <button id="Button" label="My Large Button" size="large"
onAction="MyButtonMacro" />
          <button id="Button2" label="My Normal Button" size="normal"
onAction="MyOtherButtonMacro" />
        </group >
      </tab>
```

```
    </tabs>
  </ribbon>
</customUI>
```

Esse exemplo faz as seguintes alterações na interface da Faixa de Opções no Word 2007, na ordem mostrada:

1. Em primeiro lugar, o exemplo declara o namespace padrão e um namespace personalizado.
2. O exemplo então oculta o grupo **GroupFont** interno localizado na guia **Home** interna.
3. O exemplo em seguida adiciona uma nova guia **CustomTab** à direita da última guia interna.

Observação		
Use o atributo	*id= identifier*para criar um item personalizado, como uma guia. Use o atributo	*idMso= identifier*para se referir a um item interno, como a guia **TabHome**.

4.
5. O exemplo adiciona um novo grupo **SampleGroup** à guia **My Tab**.
6. O exemplo adiciona um botão **ToogleButton1** de tamanho grande a **My Group**. Um retorno de chamada **onAction** e um retorno de chamada **GetPressed** também são especificados.
7. O exemplo adiciona uma caixa de seleção CheckBox1 a **My Group** com uma dica de tela personalizada. Um retorno de chamada onAction também é especificado.
8. O exemplo adiciona uma caixa de edição **EditBox1** a **My Group**. Um retorno de chamada **onChange** também é especificado.
9. O exemplo adiciona uma caixa de combinação **Combo1** a **My Group** com três itens. Essa caixa especifica um retorno de chamada **onChange** que usa o texto de cada item.
10. O exemplo adiciona um iniciador **Launcher1** a **My Group** com o retorno de chamada **onAction** definido. Um iniciador também pode exibir uma caixa de diálogo personalizada para oferecer mais opções ao usuário.
11. O exemplo adiciona um novo grupo **My Group** à guia personalizada.
12. O exemplo adiciona um botão **Button1** de tamanho grande a **MyGroup**. Um retorno de chamada **onAction** também é especificado.
13. Finalmente, o exemplo adiciona um botão **Button1** de tamanho normal a **MyGroup**. Um retorno de chamada **onAction** também é especificado.

Trabalhando com suplementos de barra de comando herdados

Ao criar suplementos de COM, geralmente você precisa de um meio de interação dos usuários com o suplemento. Nas versões anteriores do Office, isso era feito por meio da inclusão de um item de menu ou de um botão de barra de ferramentas no aplicativo com o uso do modelo de objeto **CommandBars**. Nesta versão do Office, os aplicativos personalizados continuam funcionando na interface do usuário da Faixa de Opções sem modificação na maioria dos casos. No entanto, as alterações que você faz com o modelo de objeto **CommandBars** ou com qualquer outra tecnologia que modificava os menus ou as barras de ferramentas, como WordBasic ou XLM, aparecem em uma guia **Suplementos** separada. Isso facilita para os usuários a localização dos controles necessários para o seu trabalho e dos suplementos que usavam anteriormente.

Atualizando de maneira dinâmica a interface do usuário da Faixa de Opções

Os retornos de chamada que retornam propriedades de um controle geralmente são chamados uma vez, a não ser que você especifique que a chamada deve ser repetida. Você pode repetir a consulta do retorno de chamada implementando o retorno de chamada **onLoad** no elemento **customui**. Esse retorno de chamada é chamado uma vez, quando o arquivo de marcação da RibbonX é carregado com êxito, e, em seguida, passa o código para um objeto **IRibbonUI**. A seguir há um código de exemplo para obter o objeto **IRibbonUI**, para que você possa atualizar os controles em tempo de execução: Marcação XML:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="ribbonLoaded">
```

Em C#: Grave um retorno de chamada na classe **Connect**:

```
IRibbonUI myRibbon;
```

```
public void ribbonLoaded(IRibbonUI ribbon) {
    myRibbon = ribbon;
}
```

A nova interface do usuário no Office 2007 oferece aos usuários uma maneira flexível de trabalhar com os aplicativos do Office. O recurso Faixa de Opções usa marcação XML baseada em texto e declarativa que simplifica a criação e a personalização da Faixa de Opções. Com poucas linhas de XML, é possível criar exatamente a interface certa para o usuário. Como a marcação XML está contida em um único arquivo, é muito mais simples modificar a interface à medida que as necessidades mudam. Você também pode melhorar a produtividade do usuário colocando os comandos em locais onde eles possam ser encontrados facilmente. Finalmente, a Faixa de Opções faz com que haja consistência entre os aplicativos, o que reduz o tempo necessário para que os usuários aprendam a trabalhar com cada aplicativo.

Usando funções de planilha do Microsoft Excel no Visual Basic

Você pode usar a maioria das funções de planilha do Microsoft Excel em suas instruções de Visual Basic. Para obter uma lista das funções de planilha que você pode usar, consulte Lista de funções de planilha disponíveis para o Visual Basic.

Observação

Algumas funções de planilha não são úteis no Visual Basic. Por exemplo, a função **Concatenate** não é necessária, pois no Visual Basic você pode usar o operador **&** para agrupar vários valores de texto.

Chamar uma função de planilha a partir do Visual Basic

No Visual Basic, as funções de planilha do Microsoft Excel estão disponíveis através do objeto **WorksheetFunction**.

O procedimento **Sub** a seguir usa a função de planilha **Min** para determinar o menor valor em um intervalo de células. Primeiro, a variável `meuIntervalo` é declarada como um objeto **Range**, e, em seguida, é definida com o intervalo A1:C10 de Sheet1. A uma outra variável, `resposta`, é atribuído o resultado de se aplicar a função **Min** a `meuIntervalo`. Finalmente, o valor de `resposta` é exibido em uma caixa de mensagem.

```
Sub UseFunction()
    Dim meuIntervalo As Range
    Set meuIntervalo = Worksheets("Plan1").Range("A1:C10")
    resposta = Application.WorksheetFunction.Min(meuIntervalo)
    MsgBox resposta
End Sub
```

Se você usar uma função de planilha que requer uma referência de intervalo como argumento, você precisará especificar um objeto **Range**. Por exemplo, você pode usar a função de planilha **Match** para pesquisar um intervalo de células. Em uma célula de planilha, você digitaria uma fórmula tal como =CORRESP(9;A1:A10;0). Entretanto, em um procedimento do Visual Basic, você especificaria um objeto **Range** para obter o mesmo resultado.

```
Sub FindFirst()
    minhaVar = Application.WorksheetFunction _
        .Match(9, Worksheets(1).Range("A1:A10"), 0)
    MsgBox minhaVar
End Sub
```

Observação

As funções do Visual Basic não usam o qualificador **WorksheetFunction**. Uma função pode ter o mesmo nome que uma função do Microsoft Excel e ainda assim funcionar de maneira diferente. Por exemplo, `Application.WorksheetFunction.Log` e `Log` retornam valores diferentes.

Inserir uma função de planilha em uma célula

Para inserir uma função de planilha em uma célula, especifique a função como valor da propriedade **Formula** do objeto **Range** correspondente. No exemplo seguinte, a função de planilha ALEATÓRIO (que gera um número randômico) é atribuída à propriedade **Formula** do intervalo A1:B3 de Sheet1 na pasta de trabalho ativa.

```
Sub InserirFormula()
    Worksheets("Plan1").Range("A1:B3").Formula = "=RAND()"
End Sub
```

Exemplo

Este exemplo usa a função de planilha **Pmt** para calcular a prestação da hipoteca de um imóvel. Observe que este exemplo usa o método **InputBox** em vez da função **InputBox** para que o método possa efetuar uma verificação de tipo. As instruções **Static** fazem o Visual Basic manter os valores das três variáveis, que serão exibidos como valores padrão na próxima vez em que você executar o programa.

```
Static AmortizacaoDoEmprestimo
Static JuroDoEmprestimo
Static PrazoDoEmprestimo
AmortizaçãoDoEmprestimo = Application.InputBox _
    (Prompt:="Quantia emprestada (100.000 por exemplo)", _
        Default:=AmortizacaoDoEmprestimo, Type:=1)
JuroDoEmprestimo = Application.InputBox _
    (Prompt:="Taxa de juros anual (8,75 por exemplo)", _
        Default:=JuroDoEmprestimo, Type:=1)
PrazoDoEmprestimo = Application.InputBox _
    (Prompt:="Prazo em anos (30 por exemplo)", _
        Default:=PrazoDoEmprestimo, Type:=1)
pagamento = Application.WorksheetFunction _
    .Pmt(JuroDoEmprestimo / 1200, PrazoDoEmprestimo * 12,
AmortizacaoDoEmprestimo)
MsgBox "Os pagamentos mensais são " & Format(pagamento, "Currency")
```

Trabalhando com formas (objetos de desenho)

As formas ou objetos de desenho são representadas por três objetos diferentes: a coleção **Shapes**, a coleção **ShapeRange** e o objeto **Shape**. Em geral, você usa a coleção **Shapes** para criar formas e para fazer uma iteração através de todas as formas em uma determinada planilha; você usa o objeto **Shape** para formatar ou modificar uma única forma; e usa a coleção **ShapeRange** quando deseja modificar várias formas da mesma maneira que você trabalha com várias formas na interface do usuário.

Definir propriedades para uma forma

Muitas propriedades de formatação de formas não são definidas por propriedades que se aplicam diretamente ao objeto **Shape** ou **ShapeRange**. Ao invés disso, atributos de forma relacionados são agrupados sob objetos secundários, como o objeto **FillFormat**, o qual contém todas as propriedades relacionadas ao preenchimento da forma ou o objeto **LinkFormat**, o qual contém todas as propriedades que são exclusivas de objetos OLE vinculados. Para definir propriedades para uma forma, você precisa primeiro retornar o objeto que representa o conjunto de atributos de forma relacionados e, em seguida, definir propriedades desse objeto retornado. Por exemplo, você usa a propriedade **Fill** para retornar o objeto **FillFormat** e, em seguida, define a propriedade **ForeColor** do objeto **FillFormat** para definir a cor de primeiro plano do preenchimento da forma especificada, conforme mostrado no exemplo seguinte.

```
Worksheets(1).Shapes(1).Fill.ForeColor.RGB = RGB(255, 0, 0)
```

Aplicar uma propriedade ou método a várias formas ao mesmo tempo

Na interface do usuário, você pode realizar algumas operações com várias formas selecionadas; por exemplo, você pode selecionar várias formas e definir todos os seus preenchimentos individuais de uma vez. Você pode realizar outras operações com uma única forma selecionada; por exemplo, você pode editar o texto em uma única forma se uma única forma estiver selecionada.

No Visual Basic, existem duas maneiras de se aplicar propriedades e métodos a um conjunto de formas. Essas duas maneiras permitem que você efetue qualquer operação que você possa efetuar em uma única forma em um intervalo de formas, quer você possa ou não efetuar a mesma operação na interface do usuário.

- Se a operação funciona em várias formas selecionadas na interface do usuário, você pode efetuar a mesma operação no Visual Basic construindo uma coleção **ShapeRange** contendo as formas com as quais você deseja trabalhar e aplicando as propriedades e métodos apropriados diretamente à coleção **ShapeRange**.
- Se a operação não funciona em várias formas selecionadas na interface do usuário, você ainda pode efetuar a operação no Visual Basic fazendo um loop pela coleção **Shapes** ou por uma coleção **ShapeRange** que contenha as formas com as quais você deseja trabalhar, e aplicando as propriedades e métodos apropriados aos objetos **Shape** individuais na coleção.

Muitas propriedades e métodos que se aplicam ao objeto **Shape** e à coleção **ShapeRange** falham quando aplicados a determinados tipos de forma. Por exemplo, a propriedade **TextFrame** falha quando aplicada a uma forma que não pode conter texto. Se você não tem certeza de que uma determinada propriedade ou método pode ser aplicado a cada uma das formas de uma coleção **ShapeRange**, não aplique a propriedade ou método à coleção **ShapeRange**. Se você desejar aplicar uma dessas propriedades ou métodos a uma coleção de formas, você precisará fazer um loop pela coleção e testar cada forma individual para certificar-se de que ela é do tipo apropriado antes de aplicar a propriedade ou método a ela.

Criar uma coleção ShapeRange contendo todas as formas de uma planilha

Você pode criar um objeto **ShapeRange** contendo todos os objetos **Shape** de uma planilha selecionando as formas e, em seguida, usando a propriedade **ShapeRange** para retornar um objeto **ShapeRange** contendo as formas selecionadas.

```
Worksheets(1).Shapes.Select
Set sr = Selection.ShapeRange
```

No Microsoft Excel, o argumento ***Index*** não é opcional para a propriedade **Range** da coleção **Shapes**, portanto você não pode usar essa propriedade sem um argumento para criar um objeto **ShapeRange** contendo todas as formas de uma coleção **Shapes**.

Aplicar uma propriedade ou método a uma coleção ShapeRange

Se você pode efetuar uma operação em várias formas selecionadas na interface do usuário ao mesmo tempo, você pode fazer o equivalente por programa construindo uma coleção **ShapeRange** e, em seguida, aplicando as propriedades ou métodos apropriados a ela. O exemplo seguinte constrói um intervalo de formas contendo as formas chamadas "Big Star" e "Little Star" em `myDocument` e aplica um preenchimento gradual a elas.

```
Set meuDocumento = Worksheets(1)
Set meuIntervalo = meuDocumento.Shapes.Range(Array("Estrela Grande", _
    "Estrela Pequena"))
myRange.Fill.PresetGradient _
    msoGradientHorizontal, 1, msoGradientBrass
```

Veja a seguir diretrizes genéricas de como as propriedades e métodos se comportam quando são aplicadas a uma coleção **ShapeRange**.

- A aplicação de um método à coleção é equivalente à aplicação do método a cada objeto **Shape** individual dessa coleção.
- A definição do valor de uma propriedade da coleção é equivalente à definição do valor da propriedade de cada forma individual desse intervalo.
- Uma propriedade da coleção que retorne uma constante retorna o valor da propriedade para uma forma individual da coleção se todas as formas da coleção têm o mesmo valor nessa propriedade. Se nem todas as formas da coleção têm o mesmo valor para a propriedade, ela retorna a constante "mista".
- Uma propriedade da coleção que retorne um tipo de dados simples (como **Long**, **Single** ou **String**) retorna o valor da propriedade de uma forma individual se todas as formas da coleção têm o mesmo valor nessa propriedade.
- O valor de algumas propriedades só pode ser retornado ou definido se houver exatamente uma forma na coleção. Se a coleção contiver mais de uma forma na coleção, ocorrerá um erro em tempo de execução. Isso geralmente é o caso do retorno ou definição de propriedades quando a ação equivalente na interface do usuário só é possível com uma única forma (ações tais como edição de texto em uma forma ou edição dos pontos de uma forma livre).

As diretrizes anteriores também se aplicam quando você está definindo propriedades de formas que estão agrupadas sob objetos secundários da coleção **ShapeRange**, tais como o objeto **FillFormat**. Se o objeto secundário representar operações que possam ser efetuadas em vários objetos selecionados na interface do usuário, você poderá retornar o objeto de uma coleção **ShapeRange** e definir suas propriedades. Você pode, por exemplo, usar a propriedade **Fill** para retornar o objeto **FillFormat** que representa os preenchimentos de todas as formas da coleção **ShapeRange**. A definição das propriedades desse objeto **FillFormat** definirão as mesmas propriedades para todas as formas individuais da coleção **ShapeRange**.

Loop através de uma coleção Shapes ou ShapeRange

Mesmo que você não possa efetuar uma operação em várias formas na interface do usuário ao mesmo tempo selecionando-as e usando em seguida um comando, você pode efetuar a ação equivalente por programa fazendo um loop através da coleção **Shapes** ou **ShapeRange** que contenha as formas com as quais você deseja trabalhar, aplicando as propriedades e métodos apropriados aos objetos **Shape** individuais da coleção. O exemplo seguinte faz um loop através de todas as formas de `myDocument` e altera a cor de primeiro plano de cada forma que seja uma AutoForma.

```
Set meuDocumento = Worksheets(1)
For Each sh In meuDocumento.Shapes
    If sh.Type = msoAutoShape Then
        sh.Fill.ForeColor.RGB = RGB(255, 0, 0)
    End If
Next
```

O exemplo seguinte constrói uma coleção **ShapeRange** contendo todas as formas selecionadas no momento na janela ativa e define a cor de primeiro plano para cada forma selecionada.

```
For Each sh in ActiveWindow.Selection.ShapeRange
    sh.Fill.ForeColor.RGB = RGB(255, 0, 0)
Next
```

Alinhar, distribuir e agrupar formas em um intervalo de forma

Use os métodos **Align** e **Distribute** para posicionar um conjunto de formas umas em relação às outras ou em relação ao documento que as contém. Use o método **Group** ou o método **Regroup** para formar uma única forma agrupada a partir de um conjunto de formas.

Exemplo das propriedades BackColor, BackStyle, BorderColor, BorderStyle, ForeColor e SpecialEffect

O exemplo a seguir demonstra as propriedades **BorderStyle** e **SpecialEffect**, exibindo cada borda disponível através dessas propriedades. O exemplo também demonstra como controlar as definições de cores utilizando as propriedades **BackColor**, **BackStyle**, **BorderColor** e **ForeColor**.

Para utilizar este exemplo, copie esse código de exemplo na parte de Declarações de um formulário. Certifique-se de que o formulário contenha:

- Seis controles **TextBox** denominados TextBox1, TextBox2, TextBox3, TextBox4, TextBox5 e TextBox6.
- Dois controles **ToggleButton** (botão de ativação ou alternância) denominados ToggleButton1 e ToggleButton2.

```
Private Sub UserForm_Initialize()
'Inicializa cada Caixa de Texto com um estilo de borda ou efeito
especial,
'e cores de primeiro plano e plano de fundo

'Caixa de Texto1 inicialmente usa um estilo de borda
TextBox1.Text = "Borda de Linha Única"
TextBox1.BorderStyle = fmBorderStyleSingle
TextBox1.BorderColor = RGB(255, 128, 128)
'Cor - Salmão
TextBox1.ForeColor = RGB(255, 255, 0)
'Cor - Amarela
TextBox1.BackColor = RGB(0, 128, 64)
'Cor - Verde #2

'Caixas de Textos 2 até 6 inicialmente usa efeitos especiais
TextBox2.Text = "Plano"
TextBox2.SpecialEffect = fmSpecialEffectFlat
TextBox2.ForeColor = RGB(128, 0, 0)
'Cor - Marrom
TextBox2.BackColor = RGB(0, 0, 255)
'Cor - Azul

'Garanta que o estilo de background para a Caixa de Texto2 seja
'inicialmente opaco.
TextBox2.BackStyle = fmBackStyleOpaque

TextBox3.Text = "Entalhado"
TextBox3.SpecialEffect = fmSpecialEffectEtched
TextBox3.ForeColor = RGB(128, 0, 255)
'Cor - Purple
TextBox3.BackColor = RGB(0, 255, 255)
'Cor - Ciano

'Defina a BorderColor para o último uso (quando
borderstyle=fmBorderStyleSingle)
```

```
TextBox3.BorderColor = RGB(0, 0, 0)
'Cor - Preto

TextBox4.Text = "protuberante"
TextBox4.SpecialEffect = fmSpecialEffectBump
TextBox4.ForeColor = RGB(255, 0, 255)
'Cor - Magenta
TextBox4.BackColor = RGB(0, 0, 100)
'Cor - Azul marinho

TextBox5.Text = "Em relevo"
TextBox5.SpecialEffect = fmSpecialEffectRaised
TextBox5.ForeColor = RGB(255, 0, 0)
'Cor - Vermelho
TextBox5.BackColor = RGB(128, 128, 128)
'Cor - Cinza

TextBox6.Text = "Afundado"
TextBox6.SpecialEffect = fmSpecialEffectSunken
TextBox6.ForeColor = RGB(0, 64, 0)
'Cor - Oliva
TextBox6.BackColor = RGB(0, 255, 0)
'Cor - Verde #1

ToggleButton1.Caption = "Troca estilos"
ToggleButton2.Caption = "Transparente/Opaco" _
    & " background"

End Sub

Private Sub ToggleButton1_Click()

‘Troca as bordas entre as Caixa de Texto 1 e  TextBox1 e Caixa de
‘Texto 3
If ToggleButton1.Value = True Then
    'Muda o Estilo de Borda da Caixa de Texto 1 para Entalhado
    TextBox1.Text = "Entalhado"
    TextBox1.SpecialEffect = fmSpecialEffectEtched

    'Muda a Caixa de Texto 3 de Entalhado para Linha Única
    TextBox3.Text = "Borda de Linha Única"
    TextBox3.BorderStyle = fmBorderStyleSingle
Else
    'Muda a Caixa de Texto 1 de volta para Linha Única
    TextBox1.Text = "BorderStyle-Single"
    TextBox1.BorderStyle = fmBorderStyleSingle

    'Muda a Caixa de Texto 3 de volta para Entalhado
    TextBox3.Text = "Entalhado"
    TextBox3.SpecialEffect = fmSpecialEffectEtched
End If
End Sub

Private Sub ToggleButton2_Click()
'Configura o background para Opaco ou Transparente
If ToggleButton2.Value = True Then
    'Muda a TextBox2 para um background transparente
    TextBox2.BackStyle = fmBackStyleTransparent
Else
```

```
    'Muda TextBox2 de volta para background opaco
    TextBox2.BackStyle = fmBackStyleOpaque
End If
End Sub
```

Exemplo das propriedades Enabled, EnterFieldBehavior, SelLength, SelStart e SelText

O exemplo a seguir rastreia as propriedades relacionadas à seleção (**SelLength**, **SelStart** e **SelText**) que se alteram à medida que o usuário move o ponto de inserção e amplia a seleção utilizando o teclado. Esse exemplo também utiliza as propriedades **Enabled** e **EnterFieldBehavior**.

Para utilizar este exemplo, copie esse código de exemplo na parte de Declarações de um formulário. Certifique-se de que o formulário contenha:

- Um **TextBox** grande denominado TextBox1.
- Três controles **TextBox** em uma coluna denominados TextBox2, TextBox3 e TextBox4.

```
Private Sub TextBox1_KeyUp(ByVal KeyCode As _
    MSForms.ReturnInteger, ByVal Shift As Integer)
    TextBox2.Text = TextBox1.SelStart
    TextBox3.Text = TextBox1.SelLength
    TextBox4.Text = TextBox1.SelText
End Sub

Private Sub UserForm_Initialize()
    TextBox1.MultiLine = True
    TextBox1.EnterFieldBehavior = _
        fmEnterFieldBehaviorRecallSelection

    TextBox1.Text = "Digite o seu texto aqui. Use " _
        & "CTRL+ENTER para começar uma nova linha."
End Sub
```

Como fazer: adicionando e gerenciando barras de menus e itens de menus

Observação

O uso de CommandBars em alguns aplicativos do Microsoft Office foi substituído pela nova interface do usuário baseada na Faixa de Opções. Para obter mais informações, pesquise a Ajuda pela palavra-chave "Faixa de Opções".

Alguns aplicativos recipientes não fornecem um meio para você criar novas barras de menus, portanto será preciso criar barras de menus usando o Visual Basic. Depois de ter criado uma barra de menus no Visual Basic, você pode personalizá-la usando a interface do aplicativo recipiente ou usando o Visual Basic.

Adicionando barras de menus em tempo de execução

Ao adicionar uma barra de menus a um aplicativo em tempo de execução, você usa o método **Add** para a coleção **CommandBars** e especifica **True** para o argumento **MenuBar**. O exemplo a seguir adiciona uma barra de menus que não pode ser movida. O exemplo também encaixa essa barra de menus ao longo do lado direito da janela do aplicativo.

```
Set menubar = CommandBars.Add _
    (Name:="mBar", Position:=msoBarRight, MenuBar:=True)
With menubar
    .Protection = msoBarNoMove
    .Visible = True
End With
```

Fazendo modificações em tempo de execução em barras de menus

Você pode fazer alterações na barra de menus e nos controles da barra de menus em tempo de execução. As alterações feitas na barra de menus podem afetar sua aparência ou posição; as alterações feitas nos controles dependem do tipo de controle. As propriedades e os métodos listados na tabela a seguir são os mais usados para modificar barras de menus em tempo de execução.

Propriedade ou método	Descrição
Add	Adiciona uma barra de menus usando o método **Add** da coleção **CommandBars** e especificando **True** para o argumento **MenuBar**.
Enabled	Se esta propriedade for definida como **True**, o usuário poderá tornar visível a barra de menus especificada usando o código do Visual Basic. Se esta propriedade for definida como **False**, o usuário não poderá tornar a barra de menus visível, mas ela aparecerá na lista de barras de comandos disponíveis.
Protection	Permite proteger a barra de menus contra ações específicas do usuário. Pode ser uma, ou uma combinação das, seguintes constantes **MsoBarProtection**: **msoBarNoChangeDock**, **msoBarNoChangeVisible**, **msoBarNoCustomize**, **msoBarNoHorizontalDock**, **msoBarNoMove**, **msoBarNoProtection**, **msoBarNoResize** e **msoBarNoVerticalDock**.
Position	Especifica a posição da nova barra de menus relativa à janela do aplicativo. Pode ser uma das seguintes constantes **MsoBarPosition**: **msoBarLeft**, **msoBarTop**, **msoBarRight**, **msoBarBottom**, **msoBarFloating**, **msoBarPopup** (usada para criar menus de atalho) ou **msoBarMenuBar** (somente Macintosh).
Visible	Especifica se o controle será exibido ou se ficará oculto. Se o controle ficar oculto, o nome da barra de menus ainda aparecerá na lista de barras de comando disponíveis.

O exemplo a seguir oculta a barra de menus ativa e a substitui por uma barra de menus temporária, encaixada ao longo do lado direito da janela do aplicativo e protegida contra o usuário.

```
Set oldMbar = CommandBars.ActiveMenuBar
Set newMbar = CommandBars.Add _
(Name:="newMenubar", Position:=msoBarRight, _
MenuBar:=True, temporary:=True)
With newMbar
    .Visible = True
    .Protection = msoBarNoMove
End With
```

Mesclando barras de menus em tempo de execução

Se você tiver barras de menus personalizadas em um aplicativo destinado a ser um suplemento, você pode especificar como os controles serão representados no aplicativo recipiente. A propriedade **OLEMenuGroup** do objeto **CommandBarPopup** pode ser usada para especificar como a mesclagem da barra de menus ocorrerá.

Fazendo modificações em tempo de execução a itens de menu

O intervalo de modificações que podem ser feitas em um item de menu depende do tipo de controle. Em geral, os botões estão ativados ou ocultos. As caixas de edição, caixas de listagem suspensa e caixas de combinação são mais versáteis porque você pode adicionar ou excluir itens da lista, além de poder determinar a ação realizada examinando o valor selecionado. É possível alterar a ação de qualquer controle para uma função interna ou personalizada.

A tabela a seguir lista as propriedades mais comuns e métodos para alterar o estado, ação ou conteúdo de um controle.

Propriedade ou método	Propósito
Add	Adiciona um item de menu em uma barra de comando. Pode ser uma das seguintes constantes **MsoControlType**para argumento Type de um controle interno: **msoControlButton**, **msoControlEdit**, **msoControlDropdown** ou **msoControlComboBox**.
AddItem	Adiciona um item à parte da lista suspensa de uma caixa de listagem suspensa ou caixa de combinação. Você pode especificar o número de índice do novo item na lista existente, mas se o número for maior do que o número de itens na lista, o método **AddItem** irá falhar.
Style	Especifica se (e em caso afirmativo, como) a face do botão exibe seu ícone, sua legenda ou ambos. Pode ser uma das seguintes constantes **MsoButtonStyle**: **msoButtonAutomatic**, **msoButtonIcon**, **msoButtonCaption**, **msoButtonIconAndCaption**, **msoButtonIconAndCaptionBelow**, **msoButtonIconAndWrapCaption**, **msoButtonIconAndWrapCaptionBelow** ou **msoButtonWrapCaption**.
OnAction	Especifica o procedimento a ser executado sempre que o usuário altera o valor do controle especificado.
Visible	Especifica se o controle será exibido ou se ficará oculto.

O exemplo a seguir adiciona um controle pop-up temporário chamado "Custom" na parte final da barra de menus ativo e, em seguida, um controle de botão chamado "Import" na barra de comandos pop-up Custom.

```
Set myMenuBar = CommandBars.ActiveMenuBar
Set newMenu = myMenuBar.Controls.Add(Type:=msoControlPopup, _
    Temporary:=True)
newMenu.Caption = "Personalizar"
Set ctrl1 = newMenu.Controls _
    .Add(Type:=msoControlButton, Id:=1)
```

```
ctrl1.Caption = "Importar"
ctrl1.TooltipText = "Importar"
ctrl1.Style = msoButtonCaption
```


Consulte também

- Como fazer: adicionando e exibindo menus de atalho
- Como fazer: adicionando e modificando barras de ferramentas
- Como fazer: usando barras de comando

Valores de erro de célula

Você pode inserir um valor de erro de célula em uma célula ou verificar se há um valor de erro em uma célula usando a função **CVErr**. Um valor de erros de células pode ser uma das seguintes constantes **XlCVError**.

Constante	Número de erro	Valor de erro de célula
xlErrDiv0	2007	#DIV/0!
xlErrNA	2042	#N/A
xlErrName	2029	#NAME?
xlErrNull	2000	#NULL!
xlErrNum	2036	#NUM!
xlErrRef	2023	#REF!
xlErrValue	2015	#VALUE!

Exemplo

Este exemplo insere os sete valores de erro de célula nas células A1:A7 de Plan1.

```
minhaMatriz = Array(xlErrDiv0, xlErrNA, xlErrName, xlErrNull, _
    xlErrNum, xlErrRef, xlErrValue)
For i = 1 To 7
    Worksheets("Plan1").Cells(i, 1).Value = CVErr(minhaMatriz(i - 1))
Next i
```

Este exemplo exibe uma mensagem quando a célula ativa de Sheet1 contém um valor de erro de célula. Você pode usar este exemplo como modelo para uma rotina de tratamento de erro baseada no valor de erro de célula.

```
Worksheets("Plan1").Activate
If IsError(ActiveCell.Value) Then
    errval = ActiveCell.Value
    Select Case errval
        Case CVErr(xlErrDiv0)
            MsgBox "#DIV/0! error"
        Case CVErr(xlErrNA)
            MsgBox "#N/A error"
        Case CVErr(xlErrName)
            MsgBox "#NAME? error"
        Case CVErr(xlErrNull)
            MsgBox "#NULL! error"
        Case CVErr(xlErrNum)
            MsgBox "#NUM! error"
        Case CVErr(xlErrRef)
```

```
            MsgBox "#REF! error"
        Case CVErr(xlErrValue)
            MsgBox "#VALUE! error"
        Case Else
            MsgBox "Isto nunca deveria acontecer!!"
    End Select
End If
```

Como fazer: fazendo alterações em balões durante o tempo de execução

O Assistente do Microsoft Office foi reprovado na versão 2007 do sistema Microsoft Office.

Depois de criar um balão para o **Assistente do Office**, você pode personalizá-lo com a adição de bitmaps, ícones ou metarquivos do Windows ao título ou ao texto do balão. Você pode adicionar controles como caixas de seleção ou botões aos seus balões para que possa responder ao usuário quando ele clica em um item no balão. Você também pode usar atributos de texto como cores e sublinhado para enfatizar elementos de texto em um balão.

Adicionando ícones e bitmaps aos balões

Para adicionar um ícone, atribua uma constante **MsoIconType** à propriedade **Icon** do objeto **Balloon**. Para adicionar um bitmap ou metarquivo do Windows ao texto em um balão do Assistente do Office, especifique o tipo, local e fator de dimensão (se aplicável) ao definir a propriedade **Text** de um título, texto, caixa de seleção ou rótulo. O exemplo a seguir insere um arquivo de bitmap do Windows no texto de um balão.

```
myBmp = "{bmp c:\Windows\circles.gif}"
myText1 = "This is before the picture, "
myText2 = " and this is after the picture"
Set bln = Assistant.NewBalloon
With bln
    .Heading = "Instructions for Choosing a Bitmap."
    .Text = myText1 & myBmp & myText2
    .Show
End With
```

Adicionando controles a balões

Há dois tipos de controles que você pode adicionar a um balão: caixas de seleção e botões. Há cinco caixas de seleção no balão quando ele é criado; você pode tornar qualquer uma delas visível especificando o texto para o controle. Há cinco botões de rótulo no balão também (se o tipo de balão for **msoBalloonTypeButtons**), e você pode expor qualquer rótulo da mesma forma que faria com uma caixa de seleção. Se você tentar expor mais do que cinco caixas de seleção ou cinco botões de rótulo em um balão, ocorre um erro.

Você pode alterar a aparência de qualquer controle em um balão alterando a propriedade **Text** do controle. Você pode alterar a funcionalidade de uma caixa de seleção ou um botão em um balão especificando um outro procedimento que será executado sempre que a caixa de seleção estiver marcada (clicada).

O exemplo a seguir cria um balão com um título, texto e três opções de região. Quando o usuário seleciona uma ou mais caixas de seleção e clica em **OK**, o exemplo chama o procedimento apropriado ou procedimentos.

```
With Assistant.NewBalloon
    .Heading = "Dados de Vendas Regionais"
    .Text = "Selecione a sua região"
    For i = 1 To 3
        .CheckBoxes(i).Text = "Região " & i
    Next
    .Button = msoButtonSetOkCancel
    .Show
```

```
    If .CheckBoxes(1).Checked Then
        runregion1
    End If
    If .CheckBoxes(2).Checked Then
        runregion2
    End If
    If .CheckBoxes(3).Checked Then
        runregion3
    End If
End With
```

Adicionando cores e sublinhado a texto em balões

Há dois atributos que você pode adicionar ao texto em um balão: cores e sublinhado. Há suporte para 16 cores de sistema que podem ser usadas para enfatizar o texto. Você também pode alterar a aparência do texto em um balão sublinhando o texto. Os atributos de texto são incorporados no texto do balão e são envoltos por chaves no código. O exemplo a seguir cria um novo balão do **Assistente do Office** que contém texto de título sublinhado, texto em vermelho e texto em azul que também é sublinhado.

```
With Assistant.NewBalloon
    .Heading = "Underlined {ul 1}Heading{ul 0}"
    .Text = "Some {cf 249}Red{cf 0} text and some " & _
    "underlined {cf 252}{ul 1}Blue{ul 0}{cf 0} text."
    .Show
End With
```


Consulte também

- Como fazer: criando e modificando balões
- Como fazer: visão geral do Assistente do Office
- Objeto Assistant
- Objeto Balloon

Exemplo da função NPV

Este exemplo utiliza a função **NPV** para retornar o valor presente líquido de uma série de fluxos de caixa contidos na matriz Valores(). TaxaRet representa a taxa interna fixa de retorno.

```
Dim Formato, Estimativa, TaxaDeRetorno, ValorPresenteLiquido, Mensagem
Static Valores(5) As Double    ' Define a matriz.
Formato = "###,##0.00"    ' Define o formato monetário.
Estimativa = .1    ' Inicia a estimativa em 10 por cento.
TaxaDeRetorno = .0625    ' Define a taxa interna fixa.
Valores(0) = -70.000    ' Custo inicial do negócio.
' Fluxo de caixa positivo refletindo a receita por quatro anos
' sucessivos.
Valores(1) = 22.000 : Valores(2) = 25.000
Valores(3) = 28.000 : Valores(4) = 31.000
ValorPresenteLiquido = NPV(TaxaDeRetorno, Valores())    ' Calcula o
valor presente líquido.
Mensagem = "O valor presente líquido destes fluxos de caixa é "
Mensagem = Mensagem & Format(ValorPresenteLiquido, Formato) & "."
MsgBox Mensagem    ' Exibe o valor presente líquido.
```

Exemplo de instruções Deftype

Este exemplo mostra vários usos das instruções **Def***tipo* para definir os tipos de dados padrão de variáveis e procedimentos de função cujos nomes iniciam com caracteres especificados. O tipo de dados padrão pode ser sobrescrito somente por atribuição explícita usando-se a instrução **Dim**. As instruções **Def***tipo* somente podem ser usadas no nível de módulo (ou seja, fora dos procedimentos).

```
' Os nomes de variável que começam com as letras entre A e K têm como
' padrão Integer.
DefInt A-K
' Os nomes de variável que começam com as letras entre L e Z têm como
' padrão String.
DefStr L-Z
CalcVar = 4    ' Inicializa Inteiro.
StringVar = "Olá, pessoal"    ' Inicializa String.
AnyVar = "Olá"     ' Provoca o erro "Tipos incompatíveis".
Dim Calc As Double    ' Define explicitamente o tipo como Double.
Calc = 2.3455    ' Atribui um Double.

' As instruções Deftipo também se aplicam a procedimentos de função.
CalcNum = ATestFunction(4)    ' Chama a função definida pelo usuário.
' Definição do procedimento de função ATestFunction.
Function ATestFunction(INumber)
    ATestFunction = INumber * 2    ' O valor retornado é um inteiro.
End Function
```

Exemplo da propriedade HelpContext

Este exemplo usa a propriedade **HelpContext** do objeto **Err** para mostrar o tópico da **Ajuda** do Visual Basic relativo ao erro Estouro.

```
Dim Mensagem
Err.Clear
On Error Resume Next
Err.Raise 6 ' Gerar o erro "Estouro".
If Err.Number <> 0 Then
    Mensagem = "Pressione F1 ou HELP para ver " & Err.HelpFile & " o
tópico para" & _
    " o seguinte HelpContext: " & Err.HelpContext
    MsgBox Mensagem, , "Erro: " & Err.Description, Err.HelpFile, _
Err.HelpContext
End If
```

Exemplo da propriedade HelpFile

Este exemplo usa a propriedade **HelpFile** do objeto **Err** para iniciar o sistema de **Ajuda**. Como padrão, a propriedade **HelpFile** contém o nome do arquivo de **Ajuda** do Visual Basic.

```
Dim Mensagem
Err.Clear
On Error Resume Next    ' Suprimir os erros por finalidade
demonstrativa.
Err.Raise 6    ' Gerar o erro "Estouro".
Mensagem = "Pressione F1 ou HELP para ver " & Err.HelpFile & _
" o tópico relativo a este erro"
MsgBox Mensagem, , "Erro: " & Err.Description,Err.HelpFile,
Err.HelpContext
```

Exemplo da função DatePart

Este exemplo considera uma data e, usando a função **DatePart**, exibe o trimestre do ano em que ocorre.

```
Dim TheDate As Date    ' Declara as variáveis.
Dim Mensagem
TheDate = InputBox("Insira uma data:")
Mensagem = "Trimestre: " & DatePart("q", TheDate)
MsgBox Mensagem
```

Exemplo da função DateDiff

Este exemplo usa a função **DateDiff** para exibir o número de dias entre uma determinada data e hoje.

```
Dim AData As Date    ' Declara as variáveis.
Dim Mensagem
AData = InputBox("Insira uma data")
Mensagem = "Dias a partir de hoje: " & DateDiff("d", Now, AData)
MsgBox Mensagem
```

Exemplo do método Add

Este exemplo utiliza o método **Add** para adicionar objetos Inst (ocorrências de uma classe denominada Class1 que contém uma variável **Public** NomeDaOcorrência) a um conjunto denominado MinhasClasses. Para ver como isto funciona, insira um módulo de classe e declare uma variável pública denominada NomeDaOcorrência em nível de módulo da Class1 (digite **Public** NomeDaOcorrência) para conter os nomes de cada ocorrência. Deixe o nome padrão como Class1. Copie e cole o código a seguir no procedimento de evento Form_Load de um módulo de formulário.

```
Dim MinhasClasses As New Collection    ' Cria um objeto Collection.
Dim Num As Integer    ' Contador para individualizar chaves.
Dim Mensagem
Dim ONome    ' Contém os nomes que o usuário insere.
Do
    Dim Inst As New Class1    ' Cria uma nova ocorrência da Class1.
    Num = Num + 1    ' Incrementa Num e, em seguida, obtém um nome.
    Mensagem = "Insira um nome para este objeto". & Chr(13) _
     & "Pressione Cancelar para ver os nomes no conjunto".
    ONome = InputBox(Mensagem, "Nomeie o conjunto de Items")
    Inst.InstanceName = ONome    ' Coloca o nome na ocorrência de
                                                ' objeto.
    ' Se o usuário inseriu o nome, adiciona-o ao conjunto.
    If Inst.InstanceName <> "" Then
        ' Adiciona o objeto nomeado ao conjunto.
        MinhasClasses.Add item := Inst, key := CStr(Num)
    End If
    ' Limpa a referência atual em preparação para a seguinte.
    Set Inst = Nothing
Loop Until ONome = ""
For Each x In MinhasClasses
    MsgBox x.InstanceName, , "Nome da ocorrência"
Next
```

Exemplo do objeto Err

Este exemplo utiliza as propriedades do objeto **Err** na construção de uma caixa de diálogo de mensagem de erro. Observe que se você utilizar primeiro o método **Clear**, ao gerar um erro do Visual Basic com o método **Raise**, os valores padrão do Visual Basic se tornarão as propriedades do objeto **Err**.

```
Dim Mensagem
' Se ocorrer um erro, construa uma mensagem de erro
On Error Resume Next    ' Adia o tratamento de erro.
Err.Clear
Err.Raise 6    ' Gera um erro "Sobrecarga".
' Verifica se há erro e, em seguida, exibe a mensagem.
If Err.Number <> 0 Then
    Mensagem = "Erro # " & Str(Err.Number) & " foi gerado por " _
            & Err.Source & Chr(13) & Err.Description
    MsgBox Mensagem, , "Erro", Err.Helpfile, Err.HelpContext
End If
```

Exemplo da função DateAdd

Este exemplo considera uma data e, usando a função **DateAdd**, exibe uma data correspondente em um número especificado de meses no futuro.

```
Dim FirstDate As Date    ' Declara as variáveis.
Dim IntervalType As String
Dim Number As Integer
Dim Mensagem
IntervalType = "m"    ' "m" especifica meses como intervalo.
FirstDate = InputBox("Insira uma data")
Number = InputBox("Insira o número de meses a adicionar")
Mensagem = "Nova data: " & DateAdd(IntervalType, Number, FirstDate)
MsgBox Mensagem
```