

**SERVIÇO NACIONAL DE APRENDIZAGEM COMERCIAL**  
**SENAC - RIO GRANDE DO SUL**

**PROGRAMAÇÃO**  
**VBA PARA EXCEL**



A força do Sistema Fecomércio ao seu lado.

Porto Alegre, 2010.

## **INTRODUÇÃO**

Esta apostila foi desenvolvida pelo SENAC/RS, para apoio ao aluno durante o desenvolvimento do curso de Programação VBA para Excel.

A disciplina introduz o estudo da Linguagem de Programação, destacando a construção de Macros e Funções do Usuário. Abrange os conceitos fundamentais para criação de macros e funções interagindo com o Excel e outros aplicativos Microsoft.

Este material é composto de bases teóricas que abrangem o conteúdo programático do curso com exemplos da aplicação.

O conteúdo programático deste curso foi projetado com o propósito de, ao final do curso, tornar o aluno apto a desenvolver macros e funções aplicadas para o Excel, com objetivo de otimizar suas tarefas na realização dos seus trabalhos.

## Sumário

INTRODUÇÃO.....	2
VBA PARA EXCEL.....	6
MACROS.....	6
ROTINAS PÚBLICAS E PRIVADAS .....	6
PASSAGEM DE PARÂMETROS .....	7
PASSAGEM DE PARÂMETROS POR VALOR – BYVAL.....	7
PASSAGEM DE PARÂMETROS POR REFERENCIA – BYREF.....	8
CRIAR MACROS.....	9
PROGRAMAÇÃO ORIENTADA A OBJETOS .....	10
OBJETO.....	10
PROPRIEDADES .....	11
MÉTODOS .....	11
CLASSES E INSTÂNCIAS .....	12
EVENTOS.....	13
PROCEDIMENTOS FUNCTION X PROCEDIMENTOS SUB .....	15
DEFINIÇÃO DE FUNCTION .....	15
DEFINIÇÃO DE ROTINAS/SUBROTINAS (MACROS) .....	18
INPUTBOX E MSGBOX.....	19
FUNÇÃO INPUTBOX .....	19
FUNÇÃO MSGBOX.....	20
VARIÁVEL IMPLÍCITA ME .....	22
OBJETO RANGE.....	23
PROPRIEDADE NAME .....	23
PROPRIEDADE VALUE .....	23
OBJETO RANGE X MÉTODO SELECT .....	24
PROPRIEDADE CELLS .....	25
PROPRIEDADE ACTIVECELL .....	26
OBJETO FONT .....	27
OBJETO INTERIOR.....	27
PROPRIEDADE HORIZONTALALIGNMENT E VERTICALALIGNMENT .....	28
PROPRIEDADES COLOR E COLORINDEX .....	29
COLEÇÃO BORDERS .....	30
PROPRIEDADE LINESTYLE E WEIGHT .....	31
PROPRIEDADE COUNT .....	31
PROPRIEDADE COLUMN E ROW.....	32
PROPRIEDADE COLUMNS E ROWS.....	32
PROPRIEDADE RESIZE.....	32
PROPRIEDADE ADDRESS.....	32

PROPRIEDADE OFFSET.....	33
FUNÇÃO ISEMPY.....	33
MÉTODO ACTIVATE .....	33
MÉTODO COPY.....	34
MÉTODO DELETE.....	34
MÉTODO MERGE.....	34
MÉTODO CLEAR.....	34
MÉTODO SORT.....	34
OBJETO WORKSHEET.....	36
PROPRIEDADE CODENAME .....	37
PROPRIEDADE NAME .....	37
PROPRIEDADE VISIBLE .....	37
PROPRIEDADE ACTIVESHEET .....	38
MÉTODO ADD.....	38
MÉTODO ACTIVATE .....	39
MÉTODO COPY.....	39
MÉTODO DELETE.....	39
MÉTODO MOVE .....	40
MÉTODO SELECT.....	40
EVENTO DE UM WORKSHEET.....	40
EVENTO ACTIVATE .....	41
EVENTO DEACTIVATE.....	41
EVENTO BEFOREDOUBLECLICK.....	41
EVENTO BEFORERIGHTCLICK .....	42
EVENTO CALCULATE.....	42
EVENTO CHANGE .....	43
EVENTO SELECTIONCHANGE.....	43
OBJETO WORKBOOK.....	45
PROPRIEDADE FULLNAME.....	45
PROPRIEDADE PATH.....	45
MÉTODO ADD.....	46
MÉTODO ACTIVATE .....	46
MÉTODO CLOSE.....	46
MÉTODO SAVE .....	47
EVENTOS DE UM WORKBOOK .....	47
EVENTO ACTIVATE .....	47
EVENTO OPEN .....	47
EVENTO NEWSHEET.....	48
EVENTO SHEETCHANGE .....	48
OBJETO APPLICATION.....	49
PROPRIEDADE THISWORKBOOK.....	49

PROPRIEDADE DISPLAYALERTS .....	49
PROPRIEDADE SCREENUPDATING .....	50
MÉTODO QUIT .....	50
MÉTODO ONKEY .....	51
VARIÁVEIS DO TIPO OBJETO .....	52
SUPLEMENTOS.....	54
CONVERTENDO UMA PASTA DE TRABALHO EM UM SUPLEMENTO.....	54
INSTALANDO O SUPLEMENTO .....	56
FECHANDO SUPLEMENTOS.....	58
USERFORMS.....	59
CONTROLES DE INTERFACE.....	60
MÉTODOS DE UM USERFORM .....	63
EVENTOS EM UM USERFORM.....	63
CRIANDO UM USERFORM .....	64
PROGRAMANDO O USERFORM .....	64
PROGRAMANDO CONTROLES .....	65
REFERÊNCIAS BIBLIOGRAFICAS .....	67

## VBA PARA EXCEL

### MACROS

Macro é um pequeno programa também denominado de Rotina que contém uma lista de instruções a realizar no Excel, que pode ser classificada como: Sub-rotina, Função ou Procedimento de evento. Quando se trata de facilitar tarefas repetitivas, longas ou um conjunto de tarefas, as rotinas resolvem o problema. Pode ser composta por uma lista armazenada de dois ou mais comandos de aplicações que, quando acionada por um programa, reproduz os comandos que foram programados.

As instruções que formam o corpo da rotina são escritas num código próprio para que o computador as possa entender, essa linguagem é designada por VBA – Visual Basic for Applications. O VBA é uma poderosa ferramenta, para automatizar alguns procedimentos que, geralmente facilitam nosso trabalho em diversas situações.

### ROTINAS PÚBLICAS E PRIVADAS

As rotinas públicas (Public) podem ser chamadas dentro de qualquer rotina e de qualquer parte do projeto, como módulos, formulários, objetos. Também ficam disponíveis em outro projeto aberto, desde que você adicione a referência a esse projeto.

As rotinas privadas (Private), só podem ser executadas no módulo onde foram escritas. Elas só podem ser chamadas dentro das rotinas que pertençam ao mesmo módulo ou objeto. Quando chamadas a partir de outro módulo, ocorrerá o erro: “Sub ou Function não definida”.

Sub-rotinas e funções são públicas por padrão. Caso não queira que sejam públicas, é necessário que declare usando a palavra-chave Private Sub ou Private Function.

Procedimentos de evento são privados por padrão, a palavra-chave Private é automaticamente inserida antes da declaração do procedimento (Private Sub evento).

#### Sintaxe Sub-rotinas:

```
Public Sub <nome_da_macro> ( )  
<corpo_da_macro>  
End Sub
```

```
Private Sub <nome_da_macro> ( )  
<corpo_da_macro>  
End Sub
```

#### Sintaxe Funções:

```
Public Function <Nome Função>(argumentos)  
    <Nome da Função> = <Valor / Expressão>  
End Function
```

```
Private Function <Nome Função>(argumentos)  
    <Nome da Função> = <Valor / Expressão>  
End Function
```

**OBS.:** Sub-rotinas são públicas por padrão, portanto não é obrigatório escrever a palavra-chave *Public*, podemos apenas escrever *Sub <Nome da Macro>*.

## PASSAGEM DE PARÂMETROS

Ao declarar uma função ou procedimento, é possível declarar um ou mais argumentos. Ao utilizar uma função ou procedimento, você deve informar os valores para os argumentos, na mesma ordem em que foram definidos durante a declaração da função ou procedimento. Este processo é conhecido como passagem de parâmetros para a função ou procedimento, ou seja, ao chamar a função ou procedimento, passamos valores que serão utilizados pelo código da função ou procedimento.

Existem duas maneiras diferentes de fazer a passagem dos parâmetros, que são passagem por valor - **ByVal** e passagem por referência – **ByRef**.

### PASSAGEM DE PARÂMETROS POR VALOR – BYVAL

Quando declarar os argumentos de uma função/procedimento, não é necessário usar a palavra **ByVal** já que os argumentos assumem automaticamente a opção **ByVal**, pois este é o método padrão. As duas declarações a seguir são equivalentes:

Sub Dobro(Num As Integer)

Ou

Sub Dobro(ByVal Num As Integer)

Na segunda declaração, explicitamente, usa-se a opção **ByVal**, para indicar que o parâmetro **Num** será passado por valor.

O que significa a passagem de um parâmetro por valor?

Na passagem de parâmetro por valor, ao iniciar a execução, a função faz uma cópia dos valores passados para serem utilizados nas operações. Supondo que o valor passado para a função fosse uma variável criada por você. Uma cópia deste valor seria efetuada durante a execução da função, não alterando em nada o valor original de sua variável. Significa que o procedimento receberá apenas o valor do parâmetro e não uma referência ao endereço de memória onde está armazenado o valor do parâmetro.

Para entender melhor este conceito, o exemplo abaixo mostra uma função principal chamada **Sub Passagem\_valor**, na qual uma variável **X** é declarada, do tipo **Integer**, e inicializada, esta variável passa, por valor, o parâmetro para o procedimento **Sub Dobro**.

**Sub** Passagem\_valor()

Dim x As Integer

x = 3

MsgBox " Valor dado a variavel X é : " & x

*'Aqui estou chamando a procedure que vai dobrar o valor de x (sub Dobro)*

Dobro (x)

MsgBox "Valor de X Após a Execução: " & x

**End** Sub

**Sub** Dobro(ByVal Num As Integer)

MsgBox " Valor passado como parâmetro:" & Num

Num = Num \* 2

MsgBox "Dobro do valor:" & Num

**End** Sub

Na prática, ao passar um parâmetro por valor, você passa apenas uma cópia do valor da variável e é nesta cópia que a função/procedimento chamado trabalha, sem afetar o valor original da variável passada como parâmetro.

#### **PASSAGEM DE PARÂMETROS POR REFERENCIA – BYREF**

Para poder alterar o valor original de uma variável, a função/procedimento, tem que receber o parâmetro por referência – ByRef, ou seja, a função/procedimento tem que receber uma referência ao endereço de memória da variável passada como parâmetro e não uma simples cópia do valor da variável . Ao receber um parâmetro por referência (ByRef), as alterações que a função/procedimento fizer, serão feitas diretamente na variável original, pois agora, a função/procedimento tem acesso ao endereço da variável na memória e não mais apenas uma cópia do seu valor.

Para que um procedimento possa receber um parâmetro por referência, você deve utilizar a palavra ByRef, conforme o exemplo a seguir:

Public Sub DobraValor(ByRef Num As Integer)

Para entender melhor este conceito, o exemplo abaixo mostra uma função principal chamada **Sub Passagem\_Ref**, na qual uma variável **Y** é declarada, do tipo Integer e inicializada, em seguida é passada, por referência, como parâmetro para o procedimento **Dobro\_Ref**.



### **Sub** Passagem\_Ref()

Dim y As Integer

y = 3

MsgBox "Valor original atribuido a Y é : " & y

*"Aqui estou chamando a procedure que vai dobrar o valor de y (sub Dobro\_ref)*

Call Dobro\_ref(y)

MsgBox "Após a execução da procedure dobro Y é : " & y

End Sub

### **Sub** Dobro\_ref(ByRef Num As Integer)

MsgBox "Valor passado como parâmetro:" & Num

Num = Num \* 2

MsgBox "Dobro do valor :" & Num

**End** Sub

OBS.: Na passagem por valor é passado apenas uma cópia do valor da variável. Os comandos executados dentro do procedimento/função chamado não irão alterar o valor original da variável. Já na chamada por referência, o endereço da variável, na memória, é passado para o procedimento/função que foi chamado. Com isso, as alterações feitas pelo procedimento, na variável, irão alterar o valor original da variável, pois estão atuando diretamente sobre esta variável, ou melhor, no endereço de memória da variável.

## **CRIAR MACROS**

Existem duas possibilidades de criação de macros:

- Através do Gravador de Macros
- Utilizando o editor e programando em Visual Basic for Applications

Abordaremos no processo do treinamento a criação de macros programadas utilizando o Editor Visual Basic para Aplicação – VBA.

---

---

---

---

---

---

---

## PROGRAMAÇÃO ORIENTADA A OBJETOS

A POO é uma metodologia de programação muito versátil que se molda a todas as áreas de aplicação da Ciência da Computação e Análise de Sistemas. Pode ser usada por todas as classes de programadores.

Na linguagem de programação VBA, toda manipulação de elementos de uma planilha ou características do Excel segue um modelo de objetos, propriedades, métodos, eventos.

### OBJETO

- É a instância de uma classe.
- Cada objeto de uma classe tem identidade e é distinguível.
- Vários objetos de uma mesma classe ocupam diferentes posições de memória, embora compartilhem os métodos (que estão apenas uma vez na memória).
- Cada objeto conhece a sua classe.
- É uma variável do tipo de uma classe

Em VBA, um objeto é uma unidade de dados que representa determinado elemento da sua planilha ou do Excel; em outras palavras, é um objeto que permite uma referência, por exemplo, às células de sua planilha, a um gráfico ou à janela do Excel. Também pode ser objetos de outros aplicativos em seu aplicativo Visual Basic. Você pode criar seus próprios objetos e definir propriedades e métodos adicionais para eles.

### Formas de manipular um objeto:

- Alterar o seu conjunto de propriedades
- Ativar métodos específicos do objeto para executar determinadas tarefas
- Associar procedimentos aos eventos que podem ocorrer sobre o objeto

### Exemplo 1:

```
Range("A1").Value = "VBA - Excel"
```

```
Range("A1").Font.Name = "Arial"
```

```
Range("A1").Font.Size = 11
```

O valor **Range("A1")** é um **objeto**: manipulações feitas sobre ele implicam ações realizadas na célula **A1**.

## Exemplo 2

ActiveCell.Value = 10,0

O objeto **ActiveCell** se refere à célula ativa no momento da execução da macro: no exemplo acima, a célula selecionada receberá o valor **10,0**

## PROPRIEDADES

- São o estado interno de um objeto.
- São as variáveis internas de um objeto.

As propriedades ou atributos são as características dos objetos. Quando definimos uma propriedade normalmente especificamos seu nome e seu tipo. Podemos ter a idéia de que as propriedades são algo assim como as variáveis onde armazenamos os dados relacionados com os objetos.

### Observe um dos exemplos abaixo:

Range("A1").Value = "VBA - Excel!"

Range("A1").Font.Name = "Arial"

Range("A1").Font.Size = 11

Como dito anteriormente, estas instruções se aplicam ao objeto **Range("A1")**, ou seja, à célula **A1**. Elas alteram ou definem **propriedades** deste objeto. Propriedades são, portanto, características de cada objeto; por exemplo, não existe uma propriedade **Value** para o objeto **Application**, mas ela existe para uma célula (objeto **Range**).

## MÉTODOS

- São a forma de se interagir com um objeto.
- São funções ou transformações que podem ser aplicadas aos objetos.
- São a única forma de alterarmos os atributos de um objeto.
- São a interface do objeto para o mundo externo.
- São os procedimentos e funções de um objeto.

**Métodos** são ações aplicadas a um objeto. Ao contrário das propriedades, que modificam suas características, métodos realizam ações mais amplas.

### Exemplo:

ActiveCell.Delete

Application.Quit

No exemplo acima, **Delete** é um método do objeto **ActiveCell**: ele apagará a célula ativa. **Quit** é um método do objeto **Application**; ele encerra o Excel (se tiver uma planilha aberta não salva, você deve salva-lá antes da conclusão desta ação).

## CLASSES E INSTÂNCIAS

- Classe de objetos: conjunto de objetos com as mesmas propriedades, com os mesmos métodos e que respondem aos mesmos eventos
- É a combinação em um único registro de campos que conterão dados e procedimentos, funções.
- Instância: todo o objeto particular de uma classe

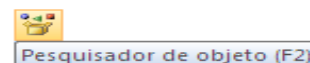
Uma classe descreve as variáveis, propriedades, procedimentos e eventos de um objeto. Objetos são instâncias de classes; Você pode criar quantos objetos precisar depois que você tenha definido uma classe.

Para entender a relação entre um objeto e sua classe, pense numa tesoura para cortar papel. A tesoura é a classe. Ela define as características de corte que vamos fazer no papel, tais como, tamanho e forma. A classe é usada para criar objetos. Os objetos são o que obtemos do papel cortado de diferentes formas e tamanhos.

## Lista de objetos, propriedades e métodos

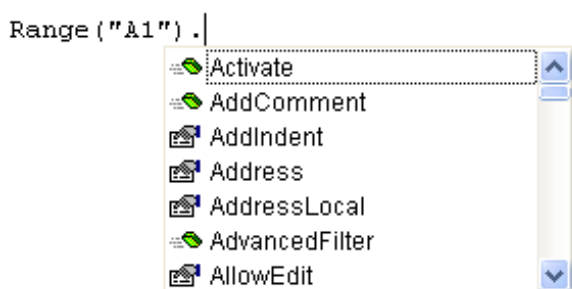
O Excel e a linguagem VBA possuem um grande número de classes (definições de objetos), métodos e propriedades. Tem por objetivo facilitar na elaboração de macros, na barra de ferramentas do Editor VB podemos encontrar uma ferramenta, *Pesquisador de Objeto*, que lista hierarquicamente todos estes elementos.

Para abrir o **Pesquisador de Objeto**, estando no editor VB, clique no ícone Pesquisador de objeto :, ou pressione F2. Será exibido uma janela conforme abaixo:







A lista à esquerda exibe todas as classes; ao selecionar um de seus itens, a região à direita exibirá todas as propriedades e métodos disponíveis para um objeto daquele tipo. Na figura acima, foi selecionada a propriedade **Font** dos objetos da classe **Range**.

As classes, métodos e propriedades subordinadas a um determinado objeto podem ser exibidos no momento da digitação de uma macro. Ao digitar o ponto após o nome do objeto em uso, esta lista será exibida automaticamente.



Observe:

Os métodos são exibidos com o ícone verde

 Clear  
 ClearComments  
 ClearContents  
 ClearFormats

Para confirmar a escolha da opção pressione a tecla **TAB**.

## EVENTOS

Eventos são ações que informam a um aplicativo que algo importante ocorreu, é algo que acontece aos objetos. Ocorre em resultado de uma ação do usuário, do sistema ou do próprio código. Por exemplo, quando o usuário clica em um formulário, o formulário pode desencadear um evento Click e chamar um procedimento que manipula o evento.

Outros exemplos: quando uma pasta de trabalho é aberta, dá-se o evento **Open** do objeto Workbook. O evento Activate ocorre quando uma pasta ou planilha é ativada. O evento BeforeSave ocorre antes da pasta de trabalho ser salva. Definindo Cancel, um dos parâmetros deste evento, como True, a pasta de trabalho será fechada sem ser salva.

Com os eventos você personaliza o comportamento dos objetos, tais como: pastas de trabalho, planilhas, gráficos, janelas, etc.

Os procedimentos correspondentes aos eventos têm todos a seguinte forma:

```
Private Sub Objeto_Evento(argumentos)
```

### Exemplos:

```
Private Sub Workbook_Open()
```

Ocorre quando o arquivo é aberto por ação do método Open.



## PROCEDIMENTOS FUNCTION X PROCEDIMENTOS SUB

Procedimentos Function são similares aos procedimentos Sub, com a diferença que as funções retornam valores e as rotinas executam ações sem necessariamente retornar valores.

### DEFINIÇÃO DE FUNCTION

Uma função definida pelo usuário assemelha-se a qualquer função de planilha pré-definida do Excel, tal como Soma e Média. Cabe ao usuário decidir o que a função deve fazer. Uma função definida pelo usuário é criada em um módulo combinando expressões matemáticas, funções pré-definidas do Excel e o código do Visual Basic. As Funções começam com a palavra-chave Function e terminam com as palavras End Function. um conjunto de valores

Uma vez que uma função produz sempre um valor ela poderá ser executada: dentro de uma célula numa planilha, à semelhança de qualquer outra função do Excel, ou dentro de qualquer outra função ou SubRotina. A sua definição tem a estrutura conforme abaixo:

**Function** <Nome da Função> ( <argumento1>, <argumento2>, ...)

...

<Nome da Função> = <Valor / Expressão>

...

**End Function**

A identificação função se dá pelo nome, pelos argumentos e tipo de dados, e tem como objetivo executar um conjunto de instruções e produzir um valor final. Isto é, sempre que se pretender executar uma função sabe-se que ela produzirá um valor.

Para definir o valor produzido por uma função basta no seu interior, atribuir ao nome da função um determinado valor ou expressão.

### Definição do tipo de dados para os argumentos e do tipo da função

Todos os elementos de entrada e saída de uma função têm um tipo de dados atribuído. Assim os argumentos deverão ser definidos com o tipo de dados respectivo e a função deverá ser definida do tipo de dados que ela for retornar o valor para o interior de uma célula do Excel.

**Function** <Nome da Função> ( <argumento1> As <Tipo>, ...) As <Tipo>

...

<Nome da Função> = <Valor / Expressão>

...

**End Function**

**Nota:** Se os tipos de dados não forem definidos será assumido por padrão como tipo Variant.

### Exemplo 1:

Criar uma função para obter o percentual do lucro. Para isso deve-se usar o preço de compra e preço de venda que se encontram na planilha. Conforme figura abaixo:

Function lucro(venda, compra)

lucro = (venda - compra) / compra

End Function

### Exemplo 2:

Crie uma função que retorne uma string com a situação de cada aluno, conforme os critérios abaixo: A frequência deve ser acima de 75%, A média deve ser superior a 7,0.

Se a média menor que 7,0, escrever a mensagem "Reprovado por média";

Se a frequência inferior a 75% , escrever a mensagem "Reprovado por frequência";

Se a média maior e igual a 7,0 e menor 9,5, escrever a mensagem "Bom";

Se a média maior e igual a 9,5, escrever a mensagem "Ótimo".

Function resultado (freq **As Single**, med **As Single**) **As String**

If freq < 0.8 Then

resultado = "reprovado por frequência"

Elseif med < 7 Then

resultado = "reprovado por média"

Elseif med <= 9 Then

resultado = "satisfatório"

Else

resultado = "ótimo"

End If

End Function

### Execução de uma Função

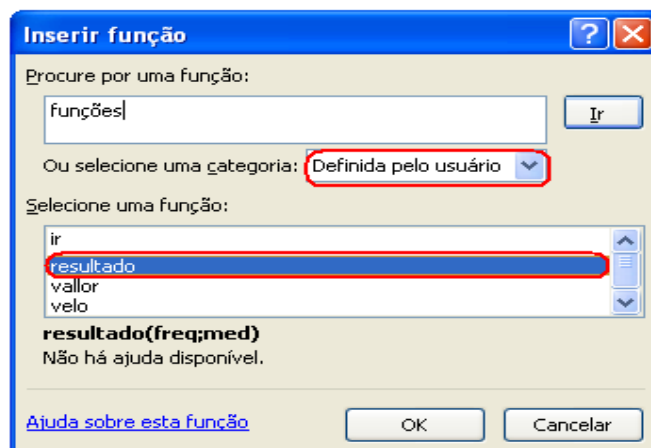
Uma vez que uma função produz um valor ela poderá ser executada:

- dentro de uma célula numa planilha, à semelhança de qualquer uma outra função do Excel.
- dentro de qualquer outra função ou SubRotina.

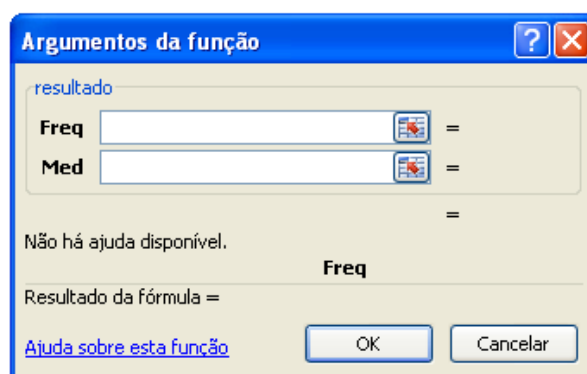
	C	D	E
	Preço de Compra	Preço de Venda	Percentual de Lucro
1			
2	R\$ 150,00	R\$ 300,00	=lucro(D2;C2)
3	R\$ 280,00	R\$ 560,00	



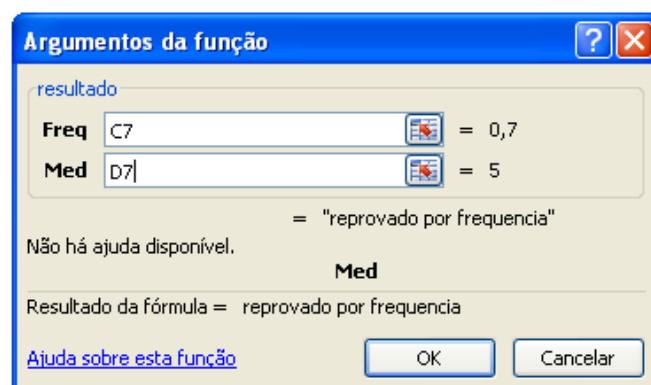
Ou pode utilizar o assistente de função:



Clique em OK para a próxima etapa do assistente de função, conforme abaixo:



Preencha as caixas de diálogo Freq e Med com a referência das respectivas células clique em OK.



### Exemplo 3

Usando uma função do Excel em uma função no módulo VBA:

**Function** Circulo(raio)

Circulo = **Application.Pi()** \* 2 \* raio

**End Function**

Para usar é necessário uma instrução para que a função seja “procurada” no aplicativo Excel. Essa instrução se chama Application, e é usada como no exemplo acima.

Quando dentro de uma rotina se faz referência ao nome de outra rotina a execução da primeira passa pela execução daquela que está a ser invocada.

#### Exemplo 4

O exemplo abaixo mostra uma função sendo chamada dentro de uma macro. A macro **Maior** faz uma chamada à função **Acha\_maior** de forma que, o valor produzido pela função seja armazenado na variável M, e assim produzir a saída final da macro **Maior**.

```
Function Acha_maior(n1 As Integer, n2 As Integer)
```

```
    If n1 > n2 Then
```

```
        acha_maior = n1
```

```
    Else
```

```
        acha_maior = n2
```

```
    End If
```

```
End Function
```

```
Sub Maior()
```

```
    Dim num1 As Integer, num2 As Integer, m As Integer
```

```
    num1 = InputBox("Informe primeiro número")
```

```
    num2 = InputBox("Informe segundo número")
```

```
    m = acha_maior(num1, num2)
```

```
    MsgBox "maior número é " & m
```

```
End Sub
```

#### **DEFINIÇÃO DE ROTINAS/SUBROTINAS (MACROS)**

As Rotinas/sub-rotinas são os procedimentos Sub, aquelas cuja definição é delimitada pelas palavras-chave Sub e EndSub. Todas as rotinas que se grava no Excel também são definidas desta forma. Rotinas/SubRotina e Macro são duas designações para a mesma realidade.

```
Sub <nome_da_macro> ( )
```

```
<corpo_da_macro>
```

```
End Sub
```

As macros são identificadas pelo nome que lhe atribuímos e não recebem variáveis no interior, têm como função desempenhar um conjunto de tarefas que são determinadas no corpo da macro.

O corpo da macro pode ser composto por um conjunto de instruções, sendo que cada instrução diferente necessita de estar numa linha diferente. Contudo, quando se trata de instruções muito longas o editor faz a sua partição por diversas linhas, recorrendo ao operador “\_”, com objetivo de facilitar a leitura.

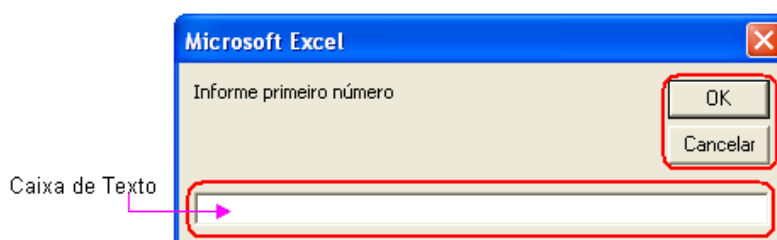
## INPUTBOX E MSGBOX

Permitem a interação entre o usuário e uma macro ou programa, funciona como uma interface de comunicação. Mostraremos dois elementos básicos para estabelecer esta ligação: InputBox e MsgBox.

### FUNÇÃO INPUTBOX

InputBox é uma função que permite ao usuário introduzir dados no programa é portanto um mecanismo de entrada.

Exibe uma janela com uma caixa de texto para a inserção de dados. Espera que o utilizador introduza os dados e/ou acione um dos botões OK ou Cancela. Conforme figura abaixo:



Como é uma função produz um valor final. Este consiste nos dados inseridos pelo usuário no formato texto que normalmente são armazenados temporariamente em variáveis de uma macro ou programa.

### Sintaxe:

InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])

O único argumento da função que não está entre colchetes é a mensagem do prompt, pois é o único argumento obrigatório; os outros são opcionais. Vamos conferir cada um:

**Prompt** – É a mensagem a ser exibida ao usuário quando executada a inputbox.

**Title** – É o título da caixa de mensagem, situado na barra de identificação (barra azul).

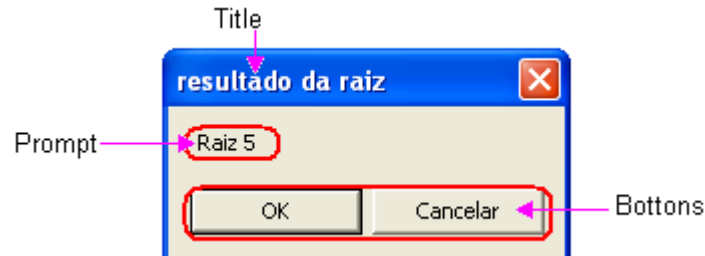
**Default** – É um valor que já aparece preenchido na inputbox como valor padrão.

**XPos, YPos** – É a posição em que a caixa de mensagem situa-se na tela. O padrão é o centro.

**HelpFile, Context** – Utilizados para designar um arquivo e um contexto de ajuda para a caixa de mensagem.

## FUNÇÃO MSGBOX

O MsgBox é um mecanismo de saída e permite ao usuário visualizar os dados produzidos pelo programa. Também pode receber entradas através de cliques em seus botões.



### Sintaxe

**MsgBox**(prompt[, buttons] [, title] [, helpfile, context])

MsgBox também possui diversos parâmetros, mas somente o primeiro é obrigatório, sendo que todos os outros quando ignorados assumem valores atribuídos por padrão.

**Prompt** – É a mensagem a ser exibida na caixa de mensagem. Se construir uma mensagem muito longa poderá utilizar o caractere Enter Chr(13); **Exemplo:**

```
MsgBox "A Soma de 3 com 5 é : " & Chr(13) & " 8 "
```

**Buttons** – Aqui temos várias opções de configuração de uma MsgBox. Se for omitido assumirá o valor 0 por padrão. A configuração “buttons” pode ser dividida em botões, comunicação visual e botões padrão; Veremos detalhes abaixo:

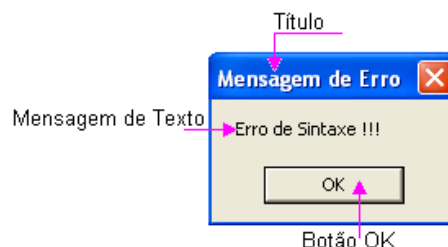
**Title** – É o título da caixa de mensagem. Se este for omitido, aparecerá o padrão que é o nome da aplicação.

**HelpFile** - Nome do arquivo que será utilizado para ajuda. Se for indicado este parâmetro o seguinte é obrigatório.

**Context** - Número do índice do tópico de ajuda.

### Exemplo:

```
MsgBox "Erro de Sintaxe !!!", , "Mensagem de Erro"
```



Na janela de saída será exibida a mensagem “**Erro de Sintaxe**”, o botão exibido será o de OK (por padrão) e o título da janela será “**Mensagem de Erro**”.

Opções de configuração Buttons, temos as seguintes constantes:

**VbOKOnly** - Exibe somente o botão de OK.

**VbOKCancel** - Exibe os botões OK e Cancelar.

**VbAbortRetryIgnore** - Exibe os botões Anular, Repetir e Ignorar..

**VbYesNoCancel** - Exibe os botões Sim, Não e Cancelar .

**VbYesNo** - Exibe os botões Sim e Não.

**VbRetryCancel** - Exibe os botões Repetir e Cancelar.

**VbCritical** - Exibe o ícone de Mensagem Crítica.

**VbQuestion** - Exibe o ícone de mensagem de Alerta.

**VbExclamation** - Exibe o ícone de mensagem de InformaçãoWarning Message.

**VbInformation** - Exibe o ícone de Information Message.

**VbDefaultButton1** - O primeiro botão é o selecionado por default.

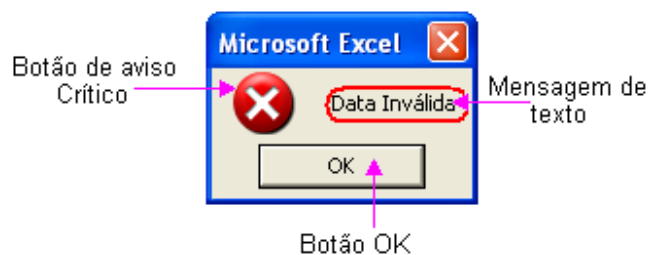
**VbDefaultButton2** - O segundo botão é o selecionado por default.

**VbDefaultButton3** - terceiro botão é o selecionado por default.

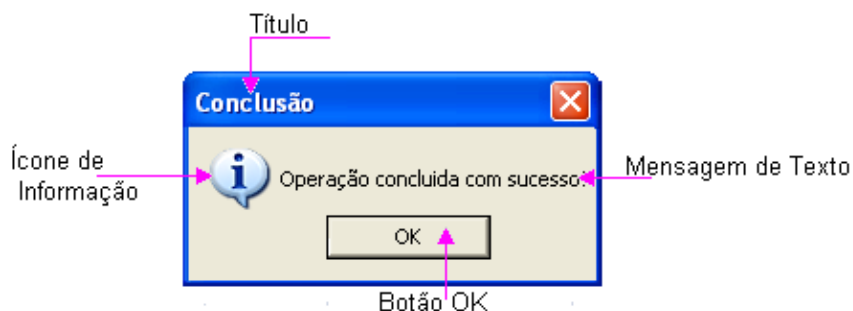
**VbDefaultButton4** - O quarto botão é o selecionado por default.

### Exemplos:

MsgBox "Data Inválida", vbCritical



MsgBox "Operação concluída com sucesso!", vbInformation, "Conclusão"





## OBJETO RANGE

O Objeto range é uma propriedade do objeto Worksheet. Isso significa que objeto Range requer que uma planilha esteja ativa ou que ele referencie uma planilha.

Esse objeto representa uma célula, uma coluna, uma linha, um conjunto de células, até todas as células de uma planilha; por isso é tão genérico e funcional. O objeto Range talvez seja o objeto mais utilizado no Excel VBA, afinal, você está manipulando dados em uma planilha.

Um objeto Range pode ser representado de diversas formas. Pode ser:

**Range("A1"):** - Uma célula ,A1;

**Range("B2:C3"):** - Um intervalo de células, B2:C3.

**Range("A1, B2:C3"):** - Uma célula, A1, mais um intervalo B2:C3

**Range("Lucro"):** - Um intervalo com o nome "Lucro"

**Range.Value:** - Valores das células do range

**Range("B2:C3").Cells(1, 2):** - Referência à célula na 1ª linha e na 2ª coluna dentro do intervalo B2:C3 (célula C2)

**Range.Offset(RowOffset, ColumnOffset):** - Desloca o range RowOffset linhas para cima/baixo e ColumnOffset colunas para a direita/esquerda

**Range.Address(RowAbsolute, ColumnAbsolute):** - Endereço do range.

**Range.Count:** - Número de células do range

### Exemplo:

```
Sub Ex1()  
Range("A1:B4") = "Olá"  
End Sub
```

### PROPRIEDADE NAME

Esta propriedade permite nomear o intervalo de um Range especificado.

### PROPRIEDADE VALUE

Retorna ou define o valor de uma determinada célula Range.

### Exemplo1:

```
Sub Ex1_Range()  
Range("A4:D10").Value = 1 'colocar no intervalo do range A1 a D10 o valor 1, na planilha ativa.  
End Sub
```

## Exemplo2:

### Sub Ex2\_range()

```
Range("A10:F20").Name = "Conjunto"
```

```
Range("Conjunto").Value = 2
```

*'Onde, na primeira instrução se atribui ao range "A1:F20" o nome "Conjunto", e na última instrução se utiliza essa designação para referenciar o respectivo conjunto de células e atribuir-lhe o valor 1, na planilha ativa.*

End Sub

## Exemplo3:

Com base na tabela abaixo vamos criar uma macro para calcular o salário de uma pessoa. Para isso a macro recebe as informações da planilha e devolve o salário na planilha.

	A	B	C	D
1	Nome	Horas Trabalhadas	Valor Hora	Salário
2	Amelia	155	25,5	

### Sub salario()

```
Dim hora_trab As Single, val_hora As Single
```

```
Dim sal As Single
```

```
hora_trab = Range("B2")
```

```
val_hora = Range("C2")
```

```
sal = hora_trab * val_hora
```

```
Range("D2") = sal
```

End Sub

## OBJETO RANGE X MÉTODO SELECT

O Método Select seleciona o Range especificado.

**Range("A1").Select** - Seleciona uma única célula que você vai escrever

**Range("A1,A5,B4").Select** - Selecionar um conjunto de células alternadas.

## Exemplo:

### Sub Selecionar()

```
Range("A1:B6").Select 'seleciona um intervalo de células
```

End Sub



## PROPRIEDADE CELLS

A propriedade Cells refere-se todas as células do objeto especificado no intervalo, pode ser uma planilha ou o intervalo de células. Podemos utilizá-la com o objeto Range representando todas as células de dentro do range. Onde **Cells(linha,Coluna)**.

### Range(Cell1,Cell2)

Por exemplo, Range("A1:B2") é equivalente a Range(Cells(1,1),Cells(2,2)).

**Cells.Select** – Seleciona todas as células de uma planilha:

**Cells.ClearContents** - Limpa todas as células de valores ou fórmulas que pode estar em uma planilha:

### Exemplo1:

Crie uma macro baseada nos exemplos acima, execute o mesmo e observe que o intervalo foi preenchido com o número 2, logo após acrescente a linha conforme abaixo e execute a macro observando que está limpo.

```
Sub Ex2_range_cells()
```

```
    Range("A10:F20").Name = "Conjunto"
```

```
    Range("Conjunto").Value = 2
```

```
    Cells.ClearContents 'limpa o intervalo anteriormente preenchido
```

```
End Sub
```

### Exemplo2:

Com base na planilha abaixo criar uma macro que calcule um acréscimo de 30% para os valores maior ou igual a 500 e escreva o valor reajustado na coluna ao lado;

	A	B
1	Valor	Valor Reajustado
2	550	
3	780	
4	250	
5	480	

---



---



---



---



---



---

**Sub** valor\_reajuste()

Dim preco, acrec As Single

Dim L As Integer

L = 2

While Cells(L, 1) <> ""

    If Cells(L, 1) <> "" Then

        preco = Cells(L, 1)

        If preco <= 500 Then

            acrec = preco \* 0.3 + preco

        Else

            acrec = preco

        End If

    End If

    Cells(L, 2).Select

    Cells(L, 2) = acrec

    L = L + 1

Wend

**End Sub**

As três afirmações abaixo são equivalentes:

**ActiveSheet.Range.Cells (1,1)**

**Range.Cells(1,1)**

**Cells(1,1)**

A propriedade Cells tem uma propriedade com nome de **Item** que torna a propriedade Cells muito útil. A propriedade **Item** permite referenciar uma célula específica em relação ao objeto Range.

**Sintaxe:**

**Cells.Item(2,"C")**

**Cells.Item(2,3)**

#### **PROPRIEDADE ACTIVECELL**

O ActiveCell propriedade retorna um objeto Range que representa a célula que está ativa na planilha especificada ou na planilha ativa. Você pode aplicar qualquer das propriedades ou métodos de um objeto Range para a célula ativa.

Com base na planilha abaixo a macro que recebe 2 valores da planilha e uma opção, retorna o valor célula ativa.

	A	B	C	D
1	Valor 1	Valor 2	Opção	Resultado
2	12	25	1	

**Sub** Range\_activecell()

Dim v1 As Single, v2 As Single, result As Single

result = 0

v1 = Range("A2")

v2 = Range("B2")

cod = Range("C2")

Range("D2").Select

Select Case cod

Case 1

result = v1 + v2

ActiveCell = result

Case 2

result = v1 - v2

ActiveCell = result

Case Else

ActiveCell = "invalido"

End Select

**End Sub**

### **OBJETO FONT**

O objeto Font possui toda a formatação de fonte que podemos utilizar na interface do Excel. Vejamos as principais propriedades:

**Name** – Define o tipo da fonte.

**Bold** – True define o estilo da fonte é negrito.

**Itálic** - True define se o estilo da fonte é itálico.

**Size** – Define o tamanho da fonte.

**ColorIndex** – define cor da fonte com valor inteiro de 1 a 56.

### **OBJETO INTERIOR**

O objeto **Interior** define o plano de fundo para as células selecionadas. Suas principais propriedades são:

**Color**- Define a cor de preenchimento de uma célula em RGB

**ColorIndex** - Define cor de preenchimento em valor inteiro de 1 a 56 cores.

**Pattern** – Define um padrão para ser aplicado no fundo de uma célula. Existem dezoito padrões representados em valor inteiro de 1 a 18.

**Patterncolor e PatternColorindex** – Definem a cor padrão a ser aplicada ao fundo da célula.

### ***PROPRIEDADE HORIZONTALALIGNMENT E VERTICALALIGNMENT***

Controla o alinhamento de dados das células contidas no range selecionado, tanto horizontalmente quanto verticalmente. As constantes podem ser:

#### **Alinhamento Horizontal:**

**xlLeft** – Alinhamento a esquerda.

**xlCenter**- Alinhamento centralizado.

**xlRight**- Alinhamento a direita

**xlDistributed ou xlJustify** - Alinhamento justificado.

#### **Alinhamento Vertical:**

**xlTop** - Alinhamento superior

**xlCenter**- Alinhamento centro (meio).

**xlBottom**- Alinhamento inferior

**xlDistributed ou xlJustify**.- Alinhamento justificado

#### **Exemplo1:**

As macros abaixo mostram recursos para alterar o tipo da fonte para Tahoma, tamanho 12, cor azul, estilo Negrito e alinhamento centralizado.

#### **Sub cor\_fonte()**

```
Range("A1:B4").Value = 10
```

```
Range("A1:B4").Font.color = vbBlue
```

```
Range("A1:B4").Font.Name = "Tahoma"
```

```
Range("A1:B4").Font.Size = 12
```

```
Range("A1:B4").Font.Bold = True
```

```
Range("A1:B4").Interior.ColorIndex = 3
```

```
Range("A1:D5").HorizontalAlignment = xlCenter
```

#### **End Sub**

#### **Sub CoRFonte()**

```
Range("A1:C1").Select
```

```
With Selection.Font
```

```
    Name = "Arial"
```

ColorIndex = 10

FontStyle = "Bold"

Size = 14

End With

**End Sub**

Exemplo do uso da propriedade Pattern.

**Sub** Cor\_Fundo()

Range("A5:B7").Value = 10

Range("A5:B7").Font.color = RGB(0, 0, 255)

Range("A5:B7").Interior.Pattern = 18

**End Sub**

Exemplo do uso da propriedade PatternColorIndex.

**Sub** Limpa\_Cor\_fundo()

Range("A8:B10").Value = 5

Range("A8:B10").Font.color = RGB(0, 0, 255)

Range("A5:B7").Interior.PatternColorIndex = 20

**End Sub**

### **PROPRIEDADES COLOR E COLORINDEX**

As propriedades **Color** e **ColorIndex** definem a cor de uma borda, fonte ou preenchimento de uma célula. Sendo que **color** em RGB, ex: Color=RGB(255,0,0), enquanto **ColorIndex** especifica em valor inteiro, ex: ColorIndex=3, os dois são equivalentes.

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.

#### **Exemplo:**

**Sub** Fundo\_RGB()

Range("A1").Interior.color = RGB(0, 0, 0) ' Preto

Range("A2").Interior.color = RGB(255, 0, 0) ' Vermelho

Range("A3").Interior.color = RGB(0, 0, 255) ' azul

Range("A4").Interior.color = RGB(128, 128, 128) ' cinza

**End Sub**

#### **Exemplo:**

A macro abaixo gera uma tabela de cores com os respectivos números para ser utilizado com a propriedade ColorIndex.



## PROPRIEDADE LINESTYLE E WEIGHT

Propriedade LineStyle determina o estilo da linha a ser usada em bordas, Weight determina a espessura da borda.

Adiciona uma borda a um intervalo definindo um estilo. Constantes para **LineStyle**.

xlLineStyleNone – Sem estilo

xlContinuous - padrão - 

xlDash - 


xlDashDot - 

xlDashDotDot - 

xlDot - 

xlDouble - 

Constantes para **Weight** a qual determina e espessura da borda.

xlThin - padrão - 

xlMedium - 

xlThick - 

xlHairline - 

O exemplo a seguir adiciona uma borda dupla para as células C1 até D5 na planilha ativa.

**Sub** Bordas()

Range("C1:D5").Borders.LineStyle = xlDouble

**End Sub**

O exemplo a seguir define a cor da borda superior e inferior das células A2: G2 na cor vermelho.

**Sub** Bordas\_Top\_bottom()

Range("A2:G2").Borders(xlEdgeTop).Color = RGB(255, 0, 0)

Range("A2:G2").Borders(xlEdgeBottom).Color = RGB(255, 0, 0)

**End Sub**

O exemplo a seguir define a cor da borda inferior das células A10: D10 na cor Azul no estilo traçado.

**Sub** Linha()

With Range("A10:D10").Borders(xlEdgeBottom)

ColorIndex = 5

LineStyle = xlDash

Weight = xlMedium

End With

**End Sub**

## PROPRIEDADE COUNT

Esta propriedade conta número de células, colunas ou linhas de um Range.

### **PROPRIEDADE COLUMN E ROW**

As propriedades Column e Row indicam coluna e linha, que formam a base de um range, ou seja, relativas à célula do canto superior esquerdo.

### **PROPRIEDADE COLUMNS E ROWS**

Essas propriedades agem como coleções, que armazenam as informações de colunas e linhas de uma range.

#### **Exemplos:**

##### **Sub Columns\_Rows()**

Columns("A:C").Value = 2 *'preenche as colunas de um Range("A:C")*

Rows("4:7").Value = 3 *'preenche as linhas de um Range("4:7")*

**End Sub**

##### **Sub Conta\_Colunas\_linhas()**

Dim lin As Integer, col As Integer

lin = Range("A1:D10").Rows.Count *'retorna o número de linhas no intervalo.*

col = Range("A1:D10").Columns.Count *' retorna o número de colunas no intervalo.*

Range("E1") = lin

Range("E2") = col

**End Sub**

### **PROPRIEDADE RESIZE**

Essa propriedade permite alterar o tamanho de um intervalo com base na localização da célula ativa. Você pode determinar um novo intervalo conforme suas necessidades.

#### **Exemplo:**

##### **Sub Teste\_Resize()**

Range("A1").Resize(3, 2).Interior.ColorIndex = 7

Range("C6").Resize(, 4).Interior.Pattern = 8

Range("C9").Resize(Rowsize:=11, columnsize:=4).Interior.Pattern = 3

**End Sub**

### **PROPRIEDADE ADDRESS**

Retorna o endereço de células do objeto Range em quetão.



### **PROPRIEDADE OFFSET**

Provoca um deslocamento de um determinado número de linhas e colunas, tendo com base o range ao qual será aplicado o deslocamento. Offset(linha, coluna), onde linha e coluna são os deslocamentos de linha e coluna.

#### **Exemplo:**

A macro abaixo seleciona e preenche com um valor a célula que se encontra 3 linhas abaixo e 1 coluna à direita da célula que encontra-se ativa na planilha ativa.

#### **Sub Testa\_offset()**

```
Selection.Offset(3, 1).Range("A1").Select
```

```
ActiveCell.Value = 500
```

**End Sub**

### **FUNÇÃO ISEMPY**

Retorna um valor booleano se uma única célula estiver vazia ou não. A Célula deve realmente estar vazia, um espaço que não possa ser visto é considerado como preenchido, ou seja, não vazia. Retorna TRUE se célula vazia e FALSE se tiverem conteúdo ou espaço.

Exemplo:

#### **Sub Test\_Iempty()**

```
Dim i As Byte
```

```
For i = 1 To 7
```

```
    If IsEmpty(Cells(i, 1)) Then
```

```
        Cells(i, 1).Resize(1, 10).Interior.ColorIndex = 5
```

```
        Cells(i, 1).Resize(1, 10).Value = " Excel"
```

```
    End If
```

```
Next i
```

**End Sub**

### **MÉTODO ACTIVATE**

Método Activate torna uma célula ativa.

O exemplo abaixo torna "C5" a célula ativa. Após a mesma recebe o formato negrito.

#### **Sub Active\_Celula()**

```
Range("C5").Activate
```

```
ActiveCell.Font.Bold = True
```

**End Sub**

OBS.: Você pode usar o método **Select** para selecionar um intervalo de células, e para tornar uma única célula ativa, use o método **Activate**.

### **MÉTODO COPY**

Copia o conteúdo de um range para outro range de destino ou para o clipboard.

Exemplo:

```
Sub Copiar_celula()  
    Range("a1:b3").Select  
    Selection.Copy  
    Range("H10:I13").Select  
    ActiveCell.PasteSpecial
```

**End Sub**

### **MÉTODO DELETE**

Exclui as células do Range especificado.

Exemplo:

```
Sub Delete()  
    Range("A2:F4").Rows().Delete  
    Worksheets("Plan1").Rows(3).Delete
```

**End Sub**

### **MÉTODO MERGE**

Este método permite mesclar as células de um determinado Range.

### **MÉTODO CLEAR**

Limpa todo e qualquer conteúdo das células dentro de um range. O conteúdo envolve valores, formatação e fórmulas.

Exemplo:

```
Sub Limpar_Mesclar()  
    Range("A1:E1").Merge across:=True  
    Range("A2:E4").Clear
```

**End Sub**

### **MÉTODO SORT**

Este método é utilizado para classificar os dados dentro de um Range de células. Os parâmetros mais relevantes desse método são:



## OBJETO WORKSHEET

O objeto Worksheet representa uma planilha, especificada da coleção Worksheets. O objeto Worksheet é também um membro da coleção Sheets. A coleção Sheets representa todas as planilhas da pasta de trabalho especificada.

Quando queremos referenciar uma Worksheets de uma pasta de trabalho precisamos indicar qual planilha, para isso temos que especificar com o número de índice qual a posição da planilha. Por exemplo, Worksheets (1) é a primeira planilha da esquerda na pasta de trabalho, Worksheets(2),... e assim sucessivamente. Todas as planilhas são incluídas na contagem do índice, mesmo se eles estão escondidos.

Use Worksheets (índice), ou o nome da planilha Worksheets("Plan1"), ou ainda se a planilha foi renomeada Worksheets("Balanço"), para retornar um objeto de planilha única.

**Sheets** é um conjunto de planilhas,. Sheets (1) é a primeira planilha na pasta de trabalho, ou Sheets("plan1"), ou se a planilha foi renomeada indicar o novo nome Sheets("FolhaPgto").

**OBS:** Um objeto é um tipo especial de variável que contém os dados e códigos. A coleção é um grupo de objetos da mesma classe.

### Exemplos:

A macro abaixo escreve na Plan1 no intervalo A1:C5 o texto "OLÁ".

**Sub** Escreve\_Plan1()

Worksheets("Plan1").Range("A1:C5") = "OLÁ"

**End Sub**

Preenche com valor de 1 á 100 o intervalo de 10 linhas por 10 colunas.

**Sub** Escreve\_valor()

num = 1

For Row = 1 To 10

For col = 1 To 10

**Sheets**("plan2").Cells(Row, col).Value = num

    num = num + 1

Next col

Next Row

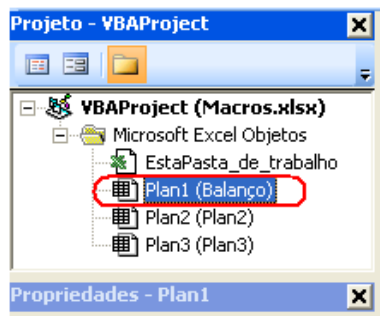
**End Sub**

Veremos a seguir algumas das várias propriedades, métodos e eventos que são aplicados aos objetos Worksheets.

## PROPRIEDADE CODENAME

Retorna o nome de código de uma planilha, que substitui a declaração Worksheets(index).

Os programadores VBA mais experientes que utilizam esta forma de referenciar planilha. Cada planilha em uma pasta de trabalho possui um nome de código único que não muda mesmo quando a planilha é movida, renomeada ou outras planilhas são adicionadas. O CodeName da planilha só pode ser visto entrando no Editor do Visual Basic no Project Explorer.



Conforme mostra a figura acima, o CodeName para a planilha com nome de guia “Balanço” é Plan1. Um CodeName de uma planilha é sempre o nome que está fora dos parênteses quando se olha no Project Explorer. Podemos fazer referência a esta planilha na pasta de trabalho usando: Plan1.Select ou Sheets ("Balanço"). Select ou Sheets (1). Select.

**OBS:** Você não pode usar um codinome PlanX quando você faz referência a uma Planilha que está em uma outra pasta de trabalho, na qual o código não reside.

## PROPRIEDADE NAME

A propriedade Name retorna o nome de uma planilha. Também pode ser usada para trocar o nome de uma planilha. O nome de uma planilha pode ser escrito com letras, números e a maioria dos símbolos, limite de 31 caracteres, porém não pode ser escrito com caracteres reservados como [ , ] , / , \* , : , ? e \.

### Exemplo:

A macro abaixo muda o nome de uma planilha especificada.

```
Sub Renomear_plan1()  
    Sheets("Plan1").Name = "Equilibrio"  
End Sub
```

## PROPRIEDADE VISIBLE

Controla a visibilidade de uma planilha, Podemos usar valores conforme a seguir:

**PlanilhaHidden** – vai ocultar a planilha indicada.

**planilhaVisible** - vai deixar visível a planilha anteriormente oculta.

### **Exemplos:**

O exemplo abaixo oculta Plan2

**Sub** Oculta\_Plan2()

Sheets("plan2").Visible = False

**End Sub**

Exemplo abaixo exhibe a Plan2 oculta pela macro anterior.

**Sub** Exibe\_Plan2()

Sheets("Plan2").Visible = True

Sheets("Plan2").Select

**End Sub**

### **PROPRIEDADE ACTIVESHEET**

Retorna um objeto que representa a planilha ativa na pasta de trabalho ativa ou na janela especificada. Se uma pasta de trabalho aparece em mais de uma janela, a propriedade ActiveSheet pode ser diferente em diferentes janelas.

### **Exemplos:**

O exemplo abaixo escreve na célula B2 da planilha ativa e mostra o nome da planilha.

**Sub** Test\_Active\_sheet()

ActiveSheet.Range("B2").Value = 3.14

MsgBox "A planilha ativa é: " & ActiveSheet.Name

**End Sub**

Esse exemplo renomeia a planilha ativa.

**Sub** ActiveSheet2()

ActiveSheet.Name = "Linda"

**End Sub**

### **MÉTODO ADD**

O método Add permite adicionar novas planilhas à pasta de trabalho ativa.

### **Exemplo 1:**

A macro abaixo adiciona a planilha “Bacana” a esquerda da planilha selecionada.

**Sub** Add\_planilha()

Worksheets.Add().Name = "Bacana"

**End Sub**

### **Exemplo 2:**

O exemplo insere 4 novas planilhas após a última planilha.

**Sub** Adiciona\_ultima()

Worksheets.Add After:=Worksheets(Worksheets.Count), Count:=4

**End Sub**

**Exemplo 3:**

Insere uma planilha com nome após a última.

**Sub** Add\_ultima()

Worksheets.Add(After:=Worksheets(Worksheets.Count)).Name = "Contas"

**End Sub**

**MÉTODO ACTIVATE**

Torna uma planilha da coleção WorkSheets ativa, movendo o foco para a planilha especificada sem alterar a seleção do usuário.

**Exemplo:**

O exemplo abaixo torna ativa a “Plan2” na qual “D7” torna-se ativa e recebe o formato negrito.

**Sub** Active\_Plan2()

Worksheets("plan2").Activate

Worksheets("plan2").Range("D7").Activate

ActiveCell.Font.Bold = True

**End Sub**

**MÉTODO COPY**

Permite criar uma cópia fiel de uma determinada planilha, em uma posição especificada na coleção Worksheets.

**Exemplo:**

A macro abaixo faz um cópia fiel da planilha ativa, inserindo a cópia após a última planilha da coleção.

**Sub** Copiar\_Planilha ()

ActiveSheet.Copy After:=Worksheets(Worksheets.Count)

**End Sub**

**MÉTODO DELETE**

Exclui um planilha da coleção WorkSheets.

**Exemplos:**

O exemplo abaixo exclui a planilha especificada.

**Sub** Excluir\_plan1()

Worksheets("plan1").Delete

**End Sub**

O exemplo abaixo exclui a última planilha da Coleção Worksheets.

**Sub** Excluir\_ultima()

Worksheets(Worksheets.Count).Delete

**End Sub**

### **MÉTODO MOVE**

Move uma planilha da coleção para local especificado.

O exemplo abaixo move a planilha ativa para a última posição

**Sub** Mover\_Ativa()

ActiveSheet.Move After:=Worksheets(Worksheets.Count)

**End Sub**

### **MÉTODO SELECT**

O método Select seleciona uma planilha ou varias planilhas, movendo a seleção do usuário para a nova planilha especificada.

#### **Exemplo1:**

**Sub** Seleciona\_ultima()

Worksheets(Worksheets.Count).Select

**End Sub**

#### **Exemplo2:**

**Sub** Seleciona\_Todas()

Dim plan As Integer

For plan = 1 To Worksheets.Count

Worksheets(plan).Select

Next

**End Sub**

#### **Exemplo3:**

**Sub** Seleciona\_plan3()

Sheets("Plan3").Select

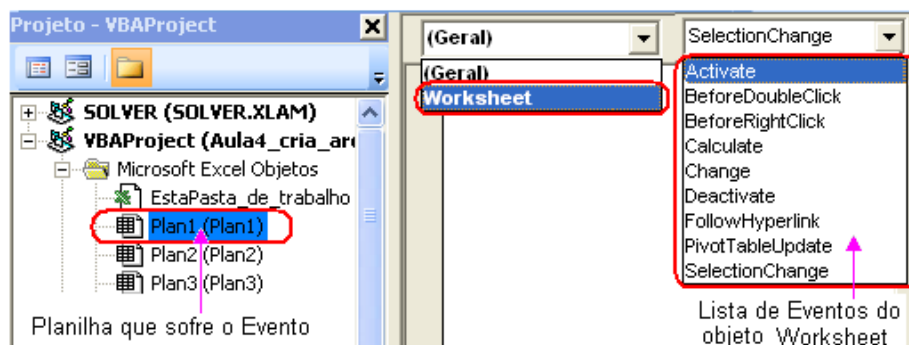
**End Sub**

### **EVENTO DE UM WORKSHEET**

Os eventos relacionados a um objeto Worksheet afetam apenas a planilha onde o evento ocorre. Para acessar os eventos de um objeto Worksheet dentro do ambiente VBA, você dá um duplo clique na planilha desejada. Vai aparecer a janela conforme abaixo. Clicando na caixa de



combinação onde aparece Geral você encontrará o objeto Worksheet, na caixa de combinação da direita você escolhe o evento desejado.



### EVENTO ACTIVATE

Este método é equivalente a clicar na guia na parte inferior da Planilha. Ocorre quando a planilha em que está o evento se torna a planilha ativa.

#### Exemplo:

```
Private Sub Worksheet_Activate()
```

```
    MsgBox "Você deu um clique na Plan1!!!!", vbExclamation
```

```
End Sub
```

OBS.: O evento Activate ocorre quando um arquivo (WorkBook), pasta (WorkSheet) ou planilha (Sheets) é ativada.

### EVENTO DEACTIVATE

Deactivate ocorre quando outra planilha se torna a planilha ativa.

#### Exemplo:

```
Private Sub Worksheet_Deactivate()
```

```
    MsgBox "Você clicou em outra Planilha !!!!", vbExclamation
```

```
End Sub
```

### EVENTO BEFOREDOUBLECLICK

O evento BeforeDoubleClick permite controlar o que acontece quando o usuário dá um duplo clique na planilha. Esse evento possui argumentos:

**Cancel** – que proporciona a possibilidade de cancelá-lo.

**Target** – do tipo Range, através desse argumento, podemos identificar qual célula recebeu o duplo clique e dar os tratamentos desejados ao evento.

### Exemplo:

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
```

```
    MsgBox "Você deu um duplo clique na célula"
```

```
    Target.Interior.ColorIndex = 5
```

```
    Cancel = True
```

```
End Sub
```

**OBS.:** A utilização de **Cancel = True** no exemplo acima evita que a ação padrão ocorra na célula que sofreu o duplo clique, que é colocar a célula em modo de edição.

### EVENTO BEFORERIGHTCLICK

Evento desencadeado quando o usuário clica com o botão direito do mouse em um intervalo. Target é o objeto clicado com o botão direito do mouse; Cancel configurado como True impede que a ação-padrão aconteça.

### Exemplo:

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
```

```
    MsgBox "Você deu um clique com botão direito na célula"
```

```
    Target.Interior.Pattern = 13
```

```
    Cancel = True
```

```
End Sub
```

### EVENTO CALCULATE

Ocorre toda vez que um cálculo é executado ou um valor de uma expressão matemática é alterado em uma célula da planilha que contém a macro.

### Exemplo:

No exemplo abaixo, a cada vez que a célula C2 sofre uma alteração exibe uma mensagem, essa alteração pode ser no momento que está editando a expressão ou se um dos valores das células envolvidas na expressão sofrer uma alteração de valor.

```
Private Sub Worksheet_Calculate()
```

```
    Dim S As Single
```

```
    S = Range("C2").Value
```

```
    If S < 0 Then
```

```
        MsgBox "Negativo!!!! Você Gastou mais que ganhou!", vbCritical
```

```
    Else
```

```
        MsgBox "Positivo!!!! Parabéns!!!!", vbExclamation
```

```
    End If
```

```
End Sub
```

	A	B	C
1	Receita	Despesas	Lucro
2	2000	800	=A2-B2

## EVENTO CHANGE

Ocorre quando uma das células da planilha teve o seu conteúdo alterado pelo usuário ou por agente externo. Por exemplo, quando o texto é inserido, editado ou excluído. Target é a célula que foi alterada.

### Exemplo1:

```
Private Sub Worksheet_Change(ByVal Target As Range)
MsgBox "Célula alterada" & Target.Address, vbCritical
End Sub
```

### Exemplo 2:

Quando uma célula for preenchida com um valor, a cor de fundo da célula ficará em vermelho.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    For Each cel In Target
        cel.Interior.ColorIndex = 3
    Next
End Sub
```

### Exemplo 3:

A macro abaixo força a entrada do texto em caracteres maiúsculos.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    If Target.Column = 1 Then 'target.column - numero coluna onde ocorrerá o efeito
        If Not (Target.Text = UCase(Target.Text)) Then 'target.text: conteúdo texto a ser digitado
            Target = UCase(Target.Text)
        End If
    End If
End Sub
```

## EVENTO SELECTIONCHANGE

Ocorre quando um novo intervalo de células é selecionado dentro de uma planilha. Target é o intervalo recentemente selecionado e podemos dar o tratamento desejado. Este é um evento padrão de uma Worksheet.

---

---

---

---



## OBJETO WORKBOOK

Workbook representa uma pasta de trabalho do Microsoft Excel. O objeto pasta de trabalho é um membro da coleção Workbooks. A coleção de pastas de trabalho contém todos os objetos Workbook atualmente aberto no Microsoft Excel.

Como um objeto Workbook representa uma pasta de trabalho, podemos acessá-lo como um índice da coleção Workbooks da seguinte maneira: workbooks(1) ou pelo nome do arquivo workbooks("nome arquivo"). O número de índice indica a ordem em que os arquivos foram abertos ou criados. Workbooks(1) é o primeiro livro criado e Workbooks(Workbooks.Count) é o último criado.

**OBS.:** Pastas de trabalho são uma coleção de todos os objetos Workbook. Planilhas são uma coleção de objetos Worksheet. O objeto Workbook representa uma pasta de trabalho, o objeto Worksheet representa uma planilha, o objeto sheet representa uma folha de planilha e o objeto Range representa um intervalo de células.

### **PROPRIEDADE FULLNAME**

Obtém o nome da pasta de trabalho, incluindo o seu caminho no disco.

#### **Exemplo1:**

Essa função retorna o caminho do disco e o nome da pasta de trabalho ativa.

**Function** RetornaCaminho() As String

Let RetornaCaminho = ThisWorkbook.FullName

**End Function**

#### **Exemplo2:**

A função retorna apenas o nome da pasta de trabalho.

**Function** RetornaNomeArq() As String

Let RetornaNomeArq = ThisWorkbook.Name

**End Function**

#### **Exemplo3:**

Esse exemplo de macro mostra caminho do arquivo ativo.

**Sub** teste()

MsgBox ActiveWorkbook.FullName

**End Sub**

### **PROPRIEDADE PATH**

Retorna o caminho onde esta a pasta de trabalho solicitada.

## **MÉTODO ADD**

O método Add permite criar uma nova pasta de trabalho vazia e adicioná-la à coleção.

### **Exemplo 1:**

O exemplo seguinte adiciona uma nova pasta de trabalho vazia ao Microsoft Excel.

#### **Sub CriaPasta()**

```
Workbooks.Add
```

**End Sub**

### **Exemplo 2:**

A macro abaixo cria um novo arquivo fecha salvando com nome.

#### **Sub CriaNovo\_arq()**

```
Workbooks.Add
```

```
ActiveWorkbook.SaveAs Filename:="C:\Macros_vba\Cadastro.xls", FileFormat:=xlExcel7
```

**End Sub**

## **MÉTODO ACTIVATE**

Esse método torna uma pasta de trabalho ativa.

## **MÉTODO CLOSE**

Fecha a pasta de trabalho desejada. Close tem alguns parâmetros, se caso omitido o funcionamento da ação será padrão Windows. O método verifica se a propriedade Saved do objeto Workbook é True. Se for, fechará a pasta, caso contrário perguntar se deseja salvar.

A macro abaixo fecha a pasta de trabalho ativa sem salvar.

### **Exemplo 1:**

#### **Sub Feccha\_SemSalvar()**

```
ActiveWorkbook.Close False
```

**End Sub**

### **Exemplo 2:**

Esse exemplo a macro fecha a pasta de trabalho ativa e o salva.

#### **Sub Fecha\_salva()**

```
ActiveWorkbook.Close True
```

**End Sub**

### **Exemplo 3:**

A macro do exemplo abaixo fecha a pasta de trabalho ativa dando a opção de escolha se o usuário deseja salvar.

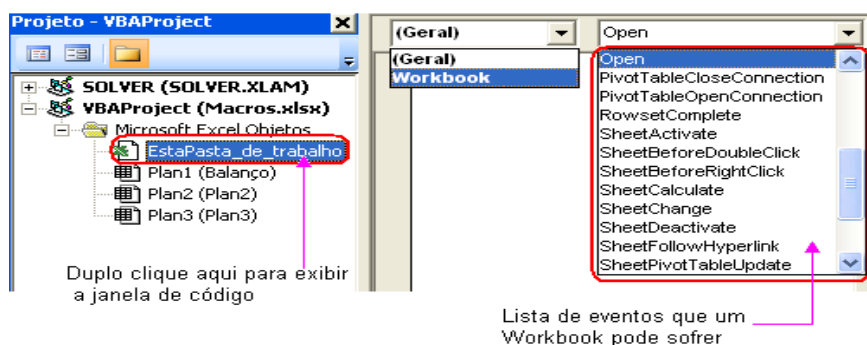
```
Sub Salva_sim_nao()  
    ActiveWorkbook.Close  
End Sub
```

### MÉTODO SAVE

Salva mudanças na pasta de trabalho.

### EVENTOS DE UM WORKBOOK

A maioria dos eventos de um Workbook afetam globalmente uma aplicação. Para acessar os eventos de um Workbook dentro do ambiente VBA, você deve dar duplo clique em **EstaPasta\_de\_Trabalho**, que está no Project Explorer. Vai aparecer a janela conforme abaixo. Clicando na caixa de combinação onde aparece Geral você encontrará o objeto Workbook, na caixa de combinação da direita você escolhe o evento desejado.



Vamos mostrar alguns dos principais eventos de um objeto Workbook e o que eles podem fazer.

### EVENTO ACTIVATE

Ocorre quando a pasta de trabalho que contém esse evento torna-se ativa.

### EVENTO OPEN

Open é o evento de pasta de trabalho padrão. Esse procedimento é ativado quando uma pasta de trabalho é aberta, não necessita de nenhuma interface com o usuário. Ele tem uma variedade de usos, como verificar o nome de usuário e personalizar os privilégios do usuário na pasta de trabalho.

#### Exemplo:

A macro abaixo abre a pasta de trabalho com a planilha Janela ativada.

```
Private Sub Workbook_Open()  
    Worksheets("Janela").Activate  
    ActiveWindow.DisplayHorizontalScrollBar = False 'desativa a barra de rolagem horizontal  
    ActiveWindow.DisplayVerticalScrollBar = False ' desativa a barra de rolagem vertical  
End Sub
```

## **EVENTO NEWSHEET**

Ocorre quando uma nova planilha é adicionada à pasta de trabalho ativa. Esse evento tem como argumento uma variável **Sh**, do tipo Object, que vai fazer referência para a planilha adicionada e pode ser utilizada dentro do código como um elemento Worksheet.

No exemplo abaixo a macro será acionada quando inserir nova planilha, a nova planilha será inserida após a última já existente.

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    Sh.Move After:=Worksheets(Worksheets.Count)
End Sub
```

## **EVENTO SHEETCHANGE**

Ocorre quando qualquer intervalo em uma planilha é alterado. **Sh** é a planilha; **Target** é o intervalo alterado. Para afetar uma planilha específica, referencie Worksheet\_Change.

### **Exemplo1:**

A macro abaixo abre uma pasta de trabalho quando você digita o nome da pasta de trabalho desejada em uma célula na planilha da pasta de trabalho ativa:

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Excel.Range)
    Workbooks.Open Filename:="C:\VbaExcel\" & Target.Value
End Sub
```

### **Exemplo2:**

Esse exemplo só aceita valores menores ou igual a 500. Se você digitar um valor maior que 500 em uma célula na pasta de trabalho, será exibida uma caixa de mensagem e o valor é removido.

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Excel.Range)
    If Target.Value > 500 Then
        MsgBox "Valor Acima do permitido Erro!!!!!!"
        Target.Clear
        Target.Select
    End If
End Sub
```

Vamos ver abaixo alguns exemplos onde usaremos Propriedades, Métodos e Eventos de um Workbook descritos acima.

### **Exemplo 1:**

Esse exemplo cria um novo arquivo solicitando que o usuário digite o nome, após salva e fecha o novo arquivo.



### **Sub Cria\_Salva\_Fecha\_Arquivo()**

Dim nomearq As String

Workbooks.Add

nomearq = InputBox("Informe o nome do novo arquivo")

ActiveWorkbook.SaveAs Filename:=nomearq, FileFormat:=xlExcel7

ActiveWorkbook.Close savechanges:=False *'Fecha a pasta recém criada*

**End Sub**

### **Exemplo 2:**

A macro abaixo abre uma pasta de trabalho existente, localizando o caminho onde a mesma se encontra.

### **Sub Acha\_Caminho()**

Dim Caminho As String

Caminho = ActiveWorkbook.Path

Workbooks.Open (Caminho & "\" & "classificar.xls")

**End Sub**

## **OBJETO APPLICATION**

O objeto Application representa todo o aplicativo Microsoft Excel e seus derivados. Através dele, podemos configurar a visualização, execuções e outras funcionalidades do Excel. O Objeto Application possui várias propriedades e métodos, aqui mostraremos os principais.

### **PROPRIEDADE THISWORKBOOK**

Retorna um objeto de pasta de trabalho que representa a pasta onde a macro código atual está funcionando. ThisWorkbook sempre devolve a pasta no qual o código está sendo executado.

### **PROPRIEDADE DISPLAYALERTS**

Habilita ou desabilita a exibição de mensagens de alerta como resposta de algumas ações do Microsoft Excel enquanto uma macro está em execução. **True** habilita as mensagens, **False** desabilita.

### **Exemplo1:**

Este exemplo fecha a pasta de trabalho "Tempo.xls" sem perguntar se deseja salvar alterações.

### **Sub Fecha\_arq()**

Application.DisplayAlerts = False

Workbooks("Tempo.xls").Close

Application.DisplayAlerts = True

**End Sub**

### **Exemplo2:**

A macro abaixo exclui uma planilha sem aviso "Application.DisplayAlerts = False", antes de excluir a segunda planilha mostra um aviso "Application.DisplayAlerts = True".

#### **Sub Excluir\_plan()**

```
Application.DisplayAlerts = False  
Application.Worksheets("plan7").Delete  
MsgBox "A planilha foi excluída sem aviso...."  
Application.DisplayAlerts = True  
Application.Worksheets("plan8").Delete  
MsgBox "A planilha foi excluída com aviso...."
```

#### **End Sub**

### **PROPRIEDADE SCREENUPDATING**

Essa propriedade permite desativar a atualização quando False. True a atualização da tela é ativada. Ou seja, quando você não quer ver seu ecrã acompanhar as ações de seu procedimento. O padrão do VBA é sempre mostrar o ecrã durante a execução do procedimento.

### **Exemplo:**

No exemplo abaixo você verá uma folha em branco, sem qualquer movimento e, em seguida, uma folha onde as células de A1 a A10000 são iguais a "77".

#### **Sub Oculta\_ecra()**

```
Range("A1").Select  
Application.ScreenUpdating = False  
Do Until Selection.Row = 10000  
    Selection.Value = 77  
    Selection.Offset(1, 0).Select  
Loop  
Range("A1").Select  
Application.ScreenUpdating = True
```

#### **End Sub**

### **MÉTODO QUIT**

Esse método encerra o Microsoft Excel com a opção que pergunta ao usuário se deseja salvar os arquivos que estão sendo fechados, caso marque "não" fecha sem salvar.

### **Exemplo:**

#### **Sub Fechar()**

```
Application.Quit
```

#### **End Sub**

### Exemplo1:

### Sub Desabilita\_Tab()

End Sub

### Exemplo2:

**Sub** Habilita\_Tab()

End Sub

### Exemplo3:

Quando pressionar a tecla TAB será exibido uma caixa de mensagem.

### Sub DemoOnKey()

```
Application.OnKey "{TAB}", "Message"
```

End Sub

## Sub Message()

MsgBox "Oi"

**End Sub**

[illegible]

## VARIÁVEIS DO TIPO OBJETO

Até agora trabalhamos com variáveis que armazenam um único valor. Também é possível ter uma variável mais poderosa chamada variável objeto, essa representa uma referência a um objeto. Uma variável de objeto pode armazenar muitos valores, todas as propriedades associadas ao objeto estão associadas a variável objeto. Uma variável de extrema importância que facilita a codificação e melhora o desempenho de uma rotina.

### Declaração da Variável Objeto

Sintaxe:

Dim <Var\_Objeto> As Objeto 'Atribuição de uma variável Objeto

Set <Var\_Objeto> = <Objeto>

Onde:

**Set** – palavra chave que indica a associação do objeto a uma variável do tipo objeto

**<Var\_Objeto>** - Variável Objeto

**<Objeto>** - Tipo do objeto a ser atribuído à variável

- **Dim oob As Object** - O tipo de variável de objeto que você usar para armazenar o objeto retornado pode afetar o desempenho do seu aplicativo. Declarar uma variável de objeto com a cláusula As Object cria uma variável que pode conter uma referência a qualquer tipo de objeto.
- **Dim Cel As Range** - Quando você define uma variável de objeto para um objeto Range, você pode facilmente manipular o intervalo usando o nome da variável.
- **Dim Planilha WorkSheet** - Uma variável objeto Worksheet pode representar uma planilha ou um conjunto de planilhas, assim você pode manipular as planilhas utilizando o nome da variável.
- **Dim Arquivo Workbook** – Quando você definir uma variável de objeto para um objeto Workbook, você pode manipular as pastas de trabalho do Microsoft Excel fazendo referência a variável.
- **Dim X As Application** – Uma variável objeto application pode representar todo o aplicativo Microsoft Excel, contendo opções de configurações, métodos que retornam objetos e assim por diante.

### Exemplo da Declaração:

Dim aplicativo As Excel.Application

Dim arquivo As Excel.Workbook

Dim planilha As Excel.WorkSheet

```
Set aplicativo = CreateObject("Excel.Application")
```

```
Set arquivo = aplicativo.Workbooks.Add
```

```
Set planilha = arquivo.Worksheets(1)
```

### **Exemplo1:**

O exemplo seguinte cria a variável de objeto **cel**, após atribui a mesma o intervalo A1:E10 da plan1 na pasta de trabalho ativa. As instruções modificam propriedades do intervalo utilizando o nome da variável que é referente ao objeto de intervalo.

```
Sub intervalo_cel()
```

```
Dim cel As Range
```

```
Set cel = Worksheets("plan2").Range("A1:D5")
```

```
cel.Formula = "=RAND()"
```

```
cel.Font.Bold = True
```

```
cel.Font.ColorIndex = 3
```

```
End Sub
```

### **Exemplo2:**

Você pode combinar vários intervalos em um objeto Range usando o método Union. O exemplo seguinte cria um objeto Range chamado **unicel**, define-o com os intervalos A1:B2 e C3:D4 e, em seguida, formata com negrito os intervalos combinados.

```
Sub uniao_cel()
```

```
Dim cel1 As Range, cel2 As Range, unicel As Range
```

```
Set cel1 = Sheets("plan2").Range("A1:B2")
```

```
Set cel2 = Sheets("plan2").Range("C3:D4")
```

```
Set unicel = Union(r1, r2)
```

```
unicel.Font.Bold = True
```

```
End Sub
```

O exemplo abaixo abre uma nova pasta de trabalho do Microsoft Excel.

```
Sub AdicionaArquivo()
```

```
Dim Arquivo As Workbook
```

```
Dim ArqVelho As Workbook
```

```
Set ArqVelho = ActiveWorkbook
```

```
Set Arquivo = Workbooks.Add
```

```
ArqVelho.Activate
```

```
MsgBox Arquivo.FullName & " Nova pasta de trabalho adicionada"
```

```
End Sub
```

## SUPLEMENTOS

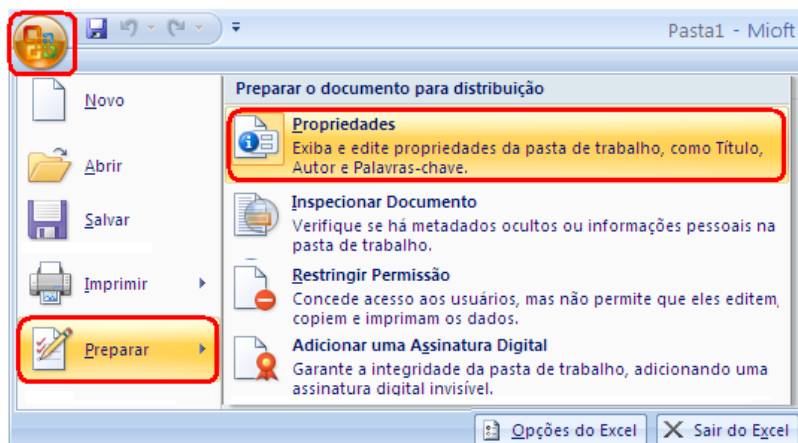
Os suplementos fornecem comandos e funcionalidades opcionais para o Microsoft Excel. Por predefinição, os suplementos não estão imediatamente disponíveis no Excel, devem primeiramente ser instalados ou ativados para poder utilizá-los.

Suplementos são projetos desenvolvidos em VBA Excel, com a extensão XLA completamente ocultos aos olhos do usuário. Uma vez que um suplemento é criado e instalado no Excel, as suas macros ou funções podem ser utilizadas em qualquer pasta de trabalho que estiver aberta. Como o suplemento nunca pode ser exibido, seu código não pode selecionar nem ativar nenhuma célula na pasta de trabalho de suplementos. É permitido salvar dados no arquivo de suplementos, mas você não pode selecionar o arquivo. Além disso, se você gravar os dados no arquivo de suplementos que você quer disponibilizar futuramente, os códigos de suplementos precisam tratar o salvamento do arquivo. Você pode adicionar **ThisWorkbook.Save** ao evento **Workbook\_BeforeClose** do suplemento.

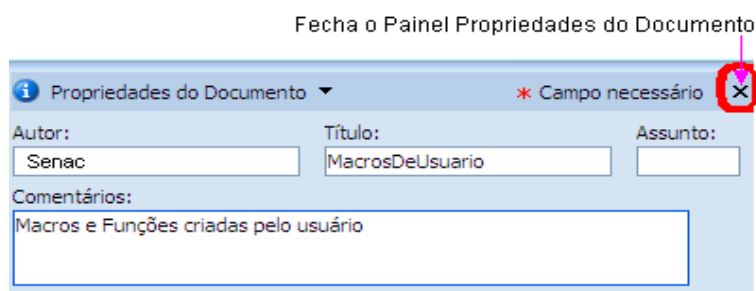
### CONVERTENDO UMA PASTA DE TRABALHO EM UM SUPLEMENTO

Siga os passos abaixo para converter uma pasta de trabalho em suplementos.

1. Clique no Ícone do Office, Preparar, Propriedades.



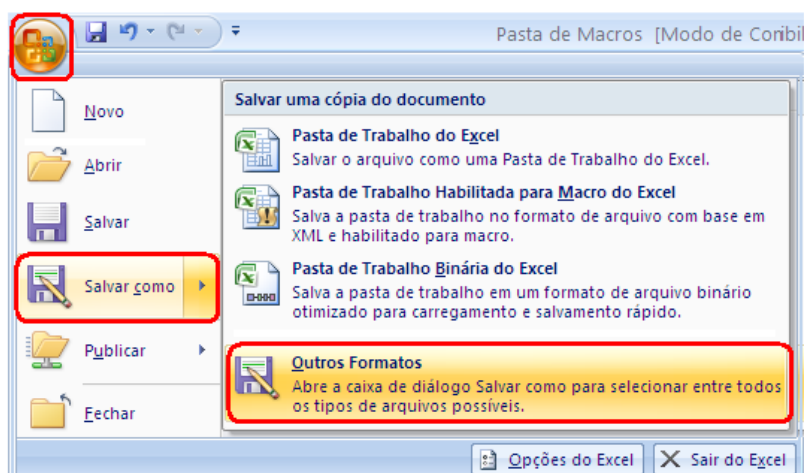
O Excel exibirá o painel Propriedades do Documento na parte superior da planilha. Conforme figura abaixo:



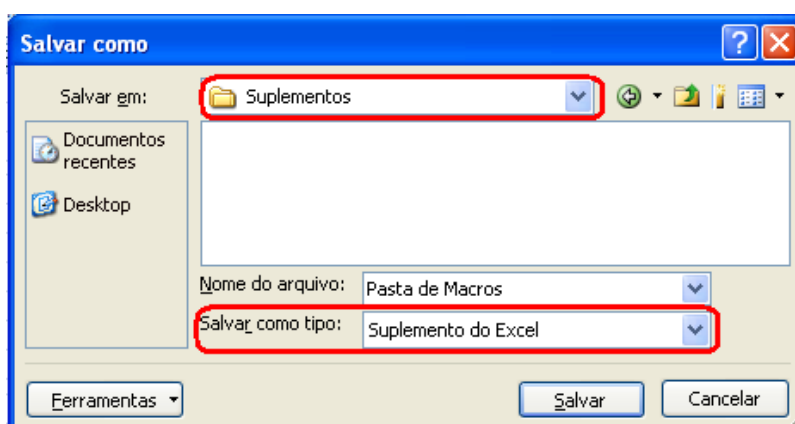
2. No campo Título insira o nome para o suplemento.
3. No campo Comentários insira uma breve descrição.
4. Clique no X no canto superior direito do painel Propriedades do Documento para fechá-lo.

Existem duas maneiras de converter o arquivo em um suplemento. O primeiro método, utilizando o comando Salvar Como, é o mais fácil. O segundo método utiliza o Editor do VBA e oferece algum controle extra.

- **Converter arquivo em suplemento utilizando o comando Salvar Como:** - clique no ícone do Office , selecione Salvar Como, Outros Formatos.



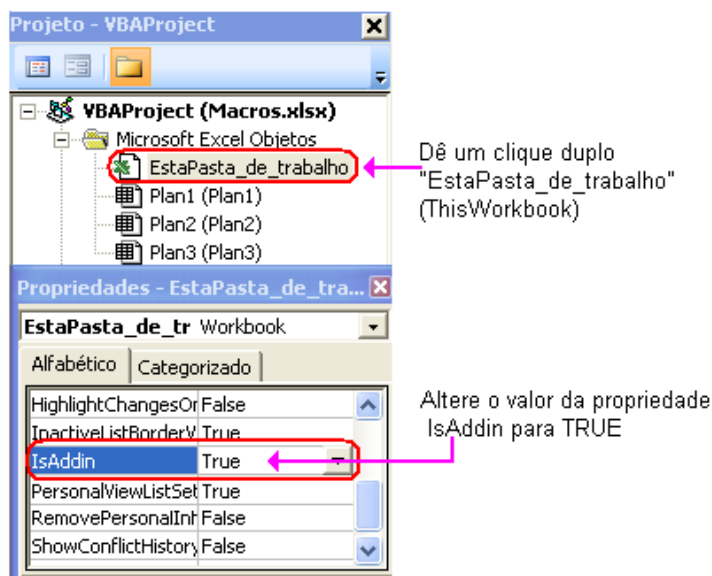
No campo **Salvar Como Tipo**, role pela lista e selecione **Suplemento do Excel (\*.xlam)**.



**Obs:** Se o suplemento puder ser utilizado nas versões anteriores através do Excel 2007, escolha suplemento Excel 97-2003(\*.xla).

Note que o arquivo muda a extensão de xlsx para xlam e o local de salvamento muda automaticamente para uma pasta de suplementos.

- **Converter arquivo em suplemento utilizando o Editor do VBA** - Abra a pasta de trabalho que você deseja converter em um suplemento. Alterne para o Editor VBA. No Project Explorer, dê um clique duplo em ThisWorkbook. Na janela, Propriedades localize a propriedade chamada IsAddin e mude seu valor para True, conforme figura abaixo:



Pressione Ctrl+G para exibir a janela de verificação Imediata. Nessa janela, salve o arquivo, utilizando uma extensão xlam, conforme a figura abaixo:

#### Verificação imediata

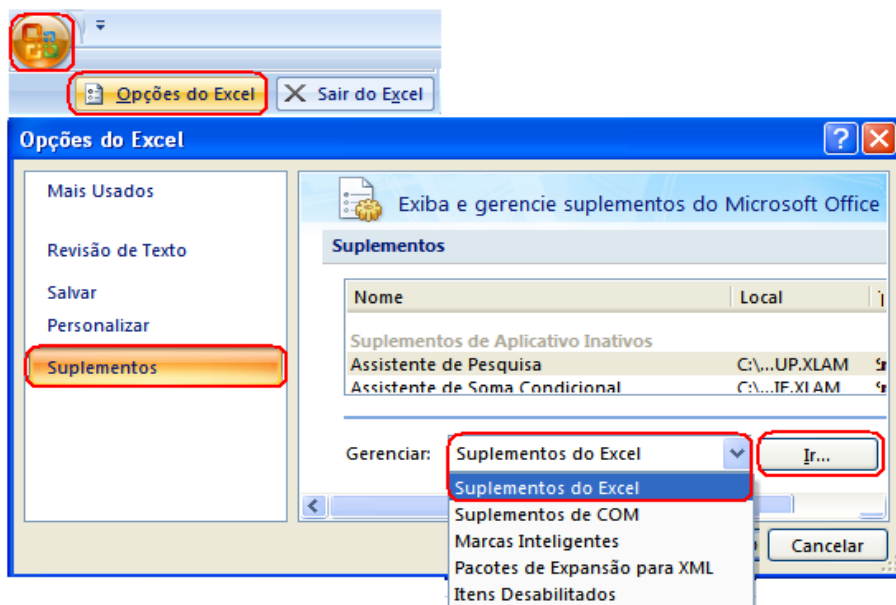
```
ThisWorkbook.SaveAs FileName:="C:\PastaMacros\Macros.xlam", fileformat:=xlAddin8
```

**Obs.:** Se o suplemento for utilizado no Excel97-2003, mude o parâmetro final xlAddin8 para xlAddin.

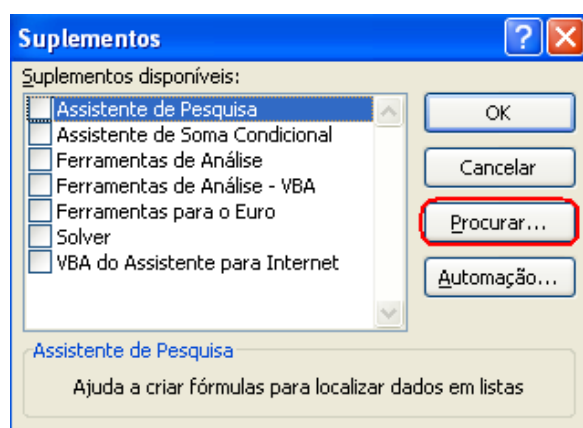
#### INSTALANDO O SUPLEMENTO

Agora que você gerou um suplemento, vamos instalar o suplemento. Para isso, vamos abrir o Excel. A partir do botão do **Office**, selecionar **Opções do Excel**. Na janela Opções do Excel clique em Suplementos. Na parte inferior da janela, escolher Suplementos do Excel a partir da lista suspensa Gerenciar. Após clicar no botão Ir, veja figura abaixo:

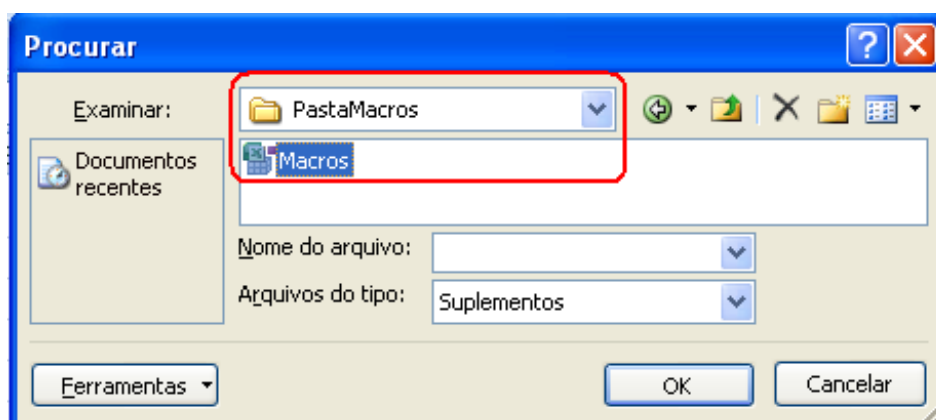




O Excel exibirá a conhecida caixa de diálogo Suplementos.



Na caixa de diálogo Suplementos, clicar no botão **Procurar**. Na caixa diálogo Procurar, selecionar o arquivo suplemento no local onde você o criou e clicar no botão OK.



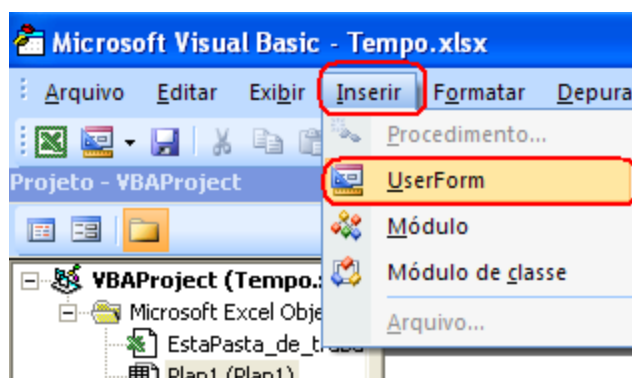
O suplemento é então instalado. O Excel copia arquivo do local onde você salvou para a localização adequada da pasta de suplementos. Na caixa de diálogo Suplementos, o título do



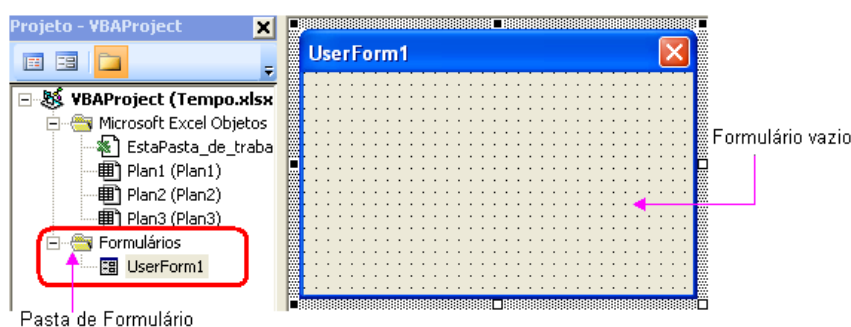
## USERFORMS

Um objeto UserForm é uma janela ou caixa de diálogo que constitui parte da interface personalizada pelo programador, tendo a finalidade de facilitar a interação entre a pasta de trabalho Excel e o usuário de forma amigável. Os formulários podem ser construídos com poucos objetos ou conforme sua necessidade. Esse ambiente personalizado estabelece a entrada de dados, sem o usuário precisar preencher e navegar pelas inúmeras linhas das planilhas. A vantagem disso é que podemos criar pequenos e inteligentes sistemas que solicitem dados aos usuários, trate esses dados e devolva os resultados, armazenando informações necessárias em planilhas Excel.

Para inserir um UserForm no ambiente VBA clique no menu **Inserir/UserForm**, veja na figura abaixo:

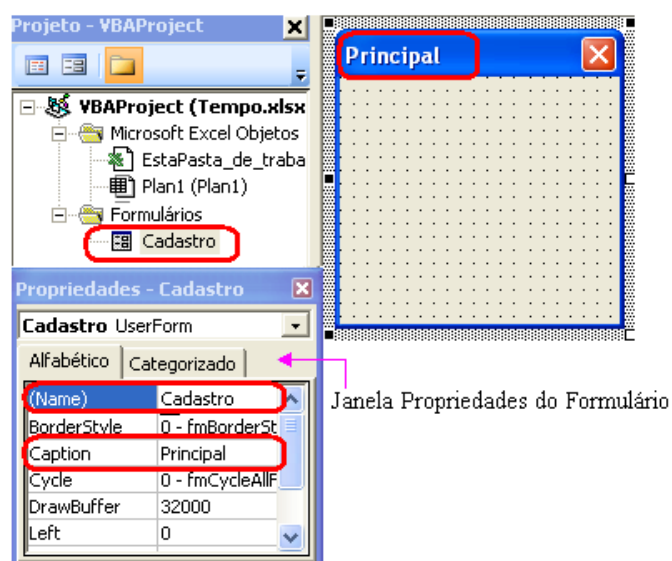


Após ter dado o clique em Inserir/UserForm a primeira visão que temos de um userform é um formulário vazio, conforme figura abaixo:

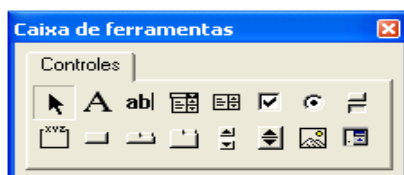


Um UserForm aparece na janela de projeto como um item da pasta de formulário. Logo abaixo, temos janela de propriedades do UserForm, que definem suas características iniciais e podem ser alteradas. Um UserForm no VBA provisoriamente tem o nome UserForm1. Se desejarmos podemos substituí-lo por outro nome mais adequado. Conforme mostra a figura abaixo onde mudamos o nome do UserForm1 da pasta de formulário, para isso, alteramos a

propriedade **Name** e digitamos “Cadastro”. Também mudamos o rótulo da barra de título que tem o mesmo nome UserForm1, Para isso, clicamos na propriedade **Caption** e digitamos “Principal”.



Quando selecionamos o formulário, a caixa de ferramentas fica ativa. Esta possui todos os controles necessários para criarmos nosso formulário. Sempre que você desejar ter uma idéia da montagem do seu formulário, você pode executá-lo, clicando em executar ou tecla F5. Fechando o formulário você volta para o ambiente de criação.



## CONTROLES DE INTERFACE

**A Rótulo** (Label1) – permite colocar um texto dentro do formulário. Para alterar o texto a ser exibido no controle, clique na propriedade Caption e digite o texto desejado. Para formatar use as propriedades Font e ForeColor.

**abl Caixa de Texto** (TextBox1) – são as melhores formas de receber dados de entrada. A caixa de texto permite digitação de qualquer tipo de dados alfanumérico, numérico, datas e até mesmo caracteres especiais.

**Método set Focus** – esse método dá foco à caixa de texto que o chamou.

**Evento Change** - ocorre sempre que uma caixa de texto tem sua propriedade Text alterada, ou seja, com qualquer digitação executada.

**Evento Enter** – ocorre quando a caixa de texto recebe o foco.

**Evento Exit** – entra em ação sempre que a caixa de texto perde o foco.



**Botão de Comando** (CommandButton1) – os botões de comandos, normalmente, são os responsáveis pelo serviço pesado de um formulário. É através deles que as funções finais são executadas e que determinam as consequências da aplicação. Os botões são sempre o ponto de saída de um formulário. As propriedades recomendáveis para alteração: Name, Caption. As demais propriedades podem ser alteradas conforme conceitos descritos no item **Propriedades Comuns**.



**Caixa de Combinação** (ComboBox1) – serve para mostrar ao usuário uma lista de opções múltipla escolha. Elas podem, ou não, permitir digitação, é uma boa forma de impedir que o usuário insira em sua planilha informações que não estejam de acordo com seus objetivos. Abaixo segue as principais propriedades específicas, Métodos e Eventos do controle caixa de combinação.

**Propriedade List** – permite acesso a lista de opções. Por exemplo, se uma lista possuir cinco elementos, ComboBox1.list(0) será o primeiro, e ComboBox1.list(5) será o último elemento da lista.

**Propriedade Style** – essa propriedade vai definir o estilo da caixa de combinação. Temos duas opções fmStyleDropDownCombo – permite a digitação, e fmStyleDropDownList – obriga a escolha de um item na lista.

**Propriedade ListIndex** – essa é uma propriedade muito importante das caixas de combinação, pois retorna ou determina qual item está selecionado. Este retorno é um índice que nos permite acessar qualquer elemento de uma caixa de combinação.

**Propriedade ListCount** – retorna a quantidade de itens dentro da lista.

**Método AddItem** - método que permite adicionar um novo item em uma caixa de combinação. Sua sintaxe:

Onde **Item** é uma string ou variável indicando o texto do item a ser inserido, e **Índice** é a posição em que o item deve ser inserido na caixa de combinação.

**Método RemoveItem** – Permite remover um item da lista de uma caixa de combinação. Sua sintaxe:


**Método Clear** – remove todos os itens de uma caixa de combinação.




**Caixa de Listagem** (ListBox1) – as caixas de listagem tem basicamente as mesmas funções das caixas de combinação. Sua única diferença é que elas se apresentam no formato de lista aberta. Vejamos algumas propriedades específicas.


**Propriedade MultiSelect** – permite múltipla escolha em uma caixa de mensagem. Temos três opções: **fmMultiSelectSigle** – permite a seleção de um único item da lista; **fmMultiSelectMulti** – permite mais de um item selecionada, utilizando o mouse para marcar e desmarcar os itens e **fmMultiSelectExtended** – permite mais de um item selecionado, utilizandoas teclas Ctrl e Shift em suas funções padrões de seleção.

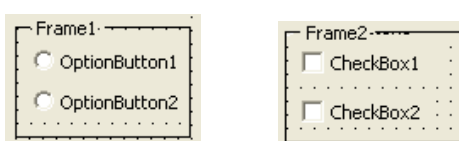
**Propriedade Selected** – através dessa propriedade, aliada ao índice, podemos determinar se um item está ou não selecionado.

 **Caixa de Seleção** (CheckBox1) – caixa de seleção é um controle booleano, ou seja, permite escolher entre Sim e Não para um determinado item.

**Propriedade Value** – esta propriedade define o estado do botão. True marca o botão e false desmarca.

 **Botão de Opção** (OptionButton1) - Os botões de opção tem características semelhantes as caixas de combinação. Porem os botões não podem trabalhar sozinhos; são necessários pelo menos dois botões ou mais para que possa haver uma escolha entre os botões. Se colocar três ou mais botões no formulário, ao escolher um os outros automaticamente são desmarcados. Para conseguir criar vários blocos de opção em um mesmo formulário, devemos colocar um cotrole Frame, para que possa servir de contêiner de controle, assim os botões que estão dentro de um frame não influenciam no estado dos botões que estejam em outros frames ou no formulário.

 **Quadro** (Frame1) – Muito utilizado como contêiner de controles, tais como, Botão de Opção e Caixa de Seleção.



## Propriedades Comuns aos Controles

**Name** – propriedade que identifica o controle nos códigos.

**Caption** – texto presente no controle.

**Left /Top** - posição relativa ao canto superior esquerdo do formulário.

**Height / Width** - altura e comprimento do controle

**BackColor** - define cor do fundo do objeto selecionada.

**ForeColor** - define cor do texto do objeto selecionado.

**BorderColor** – permite definir a cor da borda do formulário.

**BorderStyle** – determina o estilo de borda do formulário.

**BackStyle** – determina o estilo do fundo do controle (transparente/opaco)

**Font** – Exibe a caixa padrão de Fonte do Sistema onde podemos determinar a configuração dos textos exibidos no formulário ou nos controles.

**ControlTipText** – exibe um texto de ajuda quando se deixa o mouse sobre o controle.

### **Propriedades comuns em Caixa de Combinação e Caixa de Listagem**

**RowSource** - origem dos dados do controle. Pode ser um intervalo de células da planilha.

**ColumnCoun** - número de colunas a serem exibidas no controle.

**ColumnWidths** - tamanho de cada coluna.

**BoundColumn** - coluna dependente associada à propriedade Value.

**TextColumn** - coluna dependente associada à propriedade Text.

**Value** - valor da coluna dependente.

**Text** - valor presente na caixa.

**ControlSource** - célula ligada à propriedade Value da caixa

### **MÉTODOS DE UM USERFORM**

**Show** - apresenta o formulário

**Hide** - esconde o formulário mas não o remove da memória

**Unload** - remove o formulário da memória

### **EVENTOS EM UM USERFORM**

Os eventos mostrados na sequência servem também para vários outros controles, portanto mais a frente podemos apenas expandir o conceito.

**Activate** – ocorre quando o formulário for ativado.

**Click** – dá-se toda vez que o formulário for clicado.

**Doubleclick** – acontece sempre que o formulário receber um duplo clique.

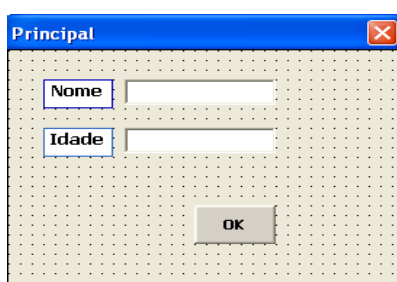
**Deactivate** – ocorre quando um formulário for desativado.

**Initialize** – acontece toda vez que um formulário for carregado para memória.

## **CRIANDO UM USERFORM**

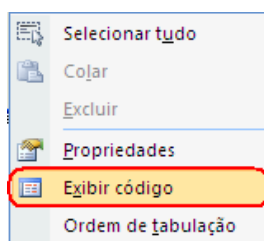
Os userform combinam as capacidades de Inputbox e MsgBox para criar uma maneira mais eficiente de interagir com o usuário.

Você pode redimensionar o formulário pegando e arrastando as alças no lado direito, parte inferior ou canto direito inferior do userform. Para adicionar qualquer um ou mais controles descritos acima, clique no controle desejado na caixa de ferramentas, mova o mouse até o formulário clique e arraste para dimensionar o controle, depois faça os ajustes necessários a seu critério, como também, alterar manualmente as propriedades na janela Propriedades, conforme as descrições das principais propriedades acima.



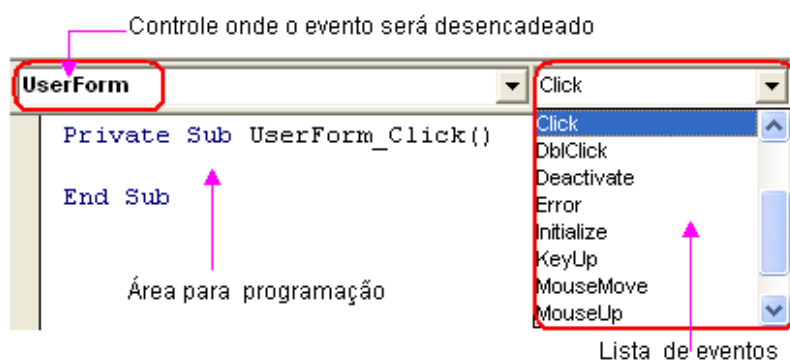
## **PROGRAMANDO O USERFORM**

Diferentemente dos outros módulos, dar um duplo clique no módulo do formulário abre o formulário no Modo de Criação. Para visualizar o código, clique com o botão direito do mouse no módulo ou no userform no Modo de Criação e selecione **Exibir Código**, ou de um duplo clique no userform.

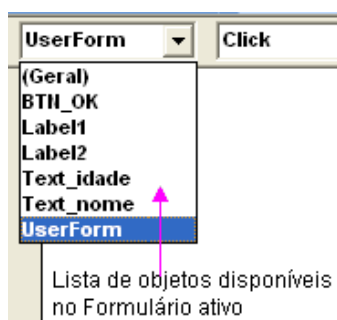


Exatamente como uma planilha, um userform tem eventos desencadeados por ações. Depois que o userform foi adicionado ao projeto, os eventos estarão disponíveis na lista suspensa que fica no canto superior direito da janela de código, selecionando-se Userform na lista suspensa Objeto, canto superior esquerdo da janela código.





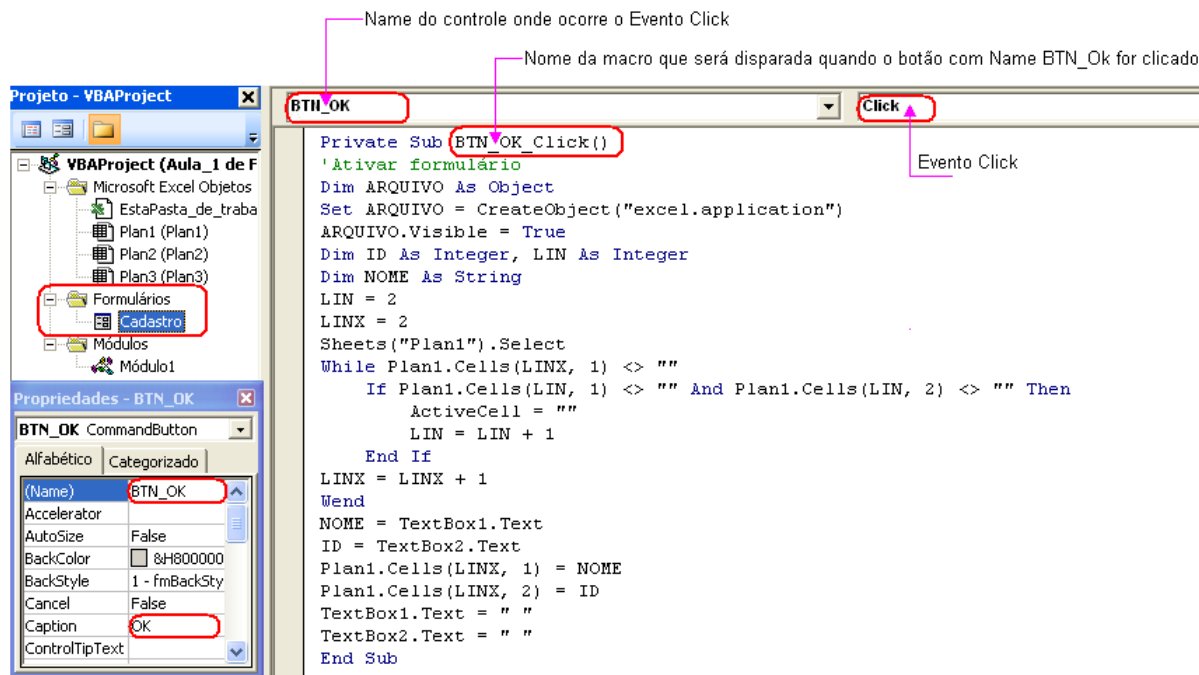
Vários eventos para o userform podem ser selecionados na lista suspensa na parte superior esquerda da janela Código



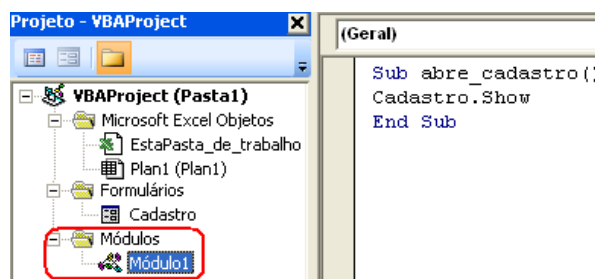
## PROGRAMANDO CONTROLES

Para programar um controle, selecione o controle no qual você deseja programar, clique com o botão direito do mouse sobre o controle, selecione **Exibir Código** ou dê um duplo clique no controle. O rodapé, o cabeçalho e a ação padrão para o controle são automaticamente inseridos na janela de código. Para ver e/ou escolher outras ações para um controle, selecione o controle na lista de Objeto e visualize as ações na lista de eventos. Podem ser escolhidas várias ações para um mesmo controle na lista de eventos.

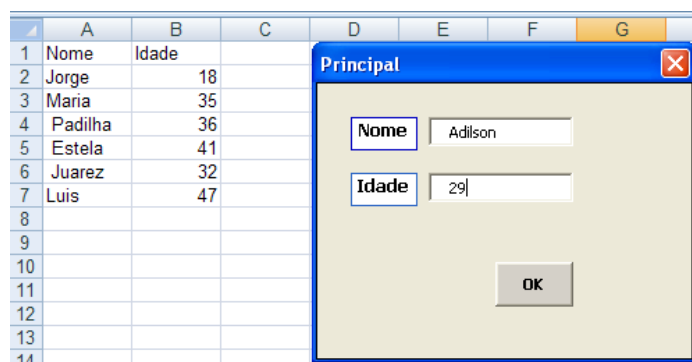
No exemplo abaixo criamos um programa no controle **Botão de Comando** com o **Name** BTN\_OK, o programa tem o objetivo de inserir nome e idade em uma planilha do excel.



Antes de executar o userform precisamos chamá-lo a partir de qualquer módulo, para isso utilizamos o método Show, sua sintaxe: NameForm.Show.



Após executar o código acima você visualizará seu formulário conforme abaixo a cada informação que inserir nas caixas de textos e clicar no botão Ok uma planilha do Excel estará recebendo os dados.



## REFERÊNCIAS BIBLIOGRÁFICAS

JELLEN, Bill – VBA e Macros para Microsoft Office Excel 2007

FERNANDES, Maicris – Desenvolvendo Aplicações Poderosas com Excel e VBA