APOSTILA 2021/1

CURSO DE VBA

PET CIVIL UFRGS

SUMÁRIO

1. Macros

a. O que é uma Macro?

Uma macro é uma sequência de comandos e funções criada pelo usuário e pode ser executada sempre que for preciso. Funciona como um atalho para tarefas em geral, economizando tempo quando estas são repetitivas e/ou demoradas. A macro gravada no Excel, armazena as informações referente a cada etapa realizada à medida que você vai executando uma série de comandos, após finalizar a gravação a macro fica disponível para ser aplicada na planilha.

b. Habilitar guia Desenvolvedor

A guia Desenvolvedor, onde encontra-se a ferramenta Macros, por padrão está desabilitada na Barra de Menus inicial do Excel. Para habilitá-la existem dois caminhos possíveis.

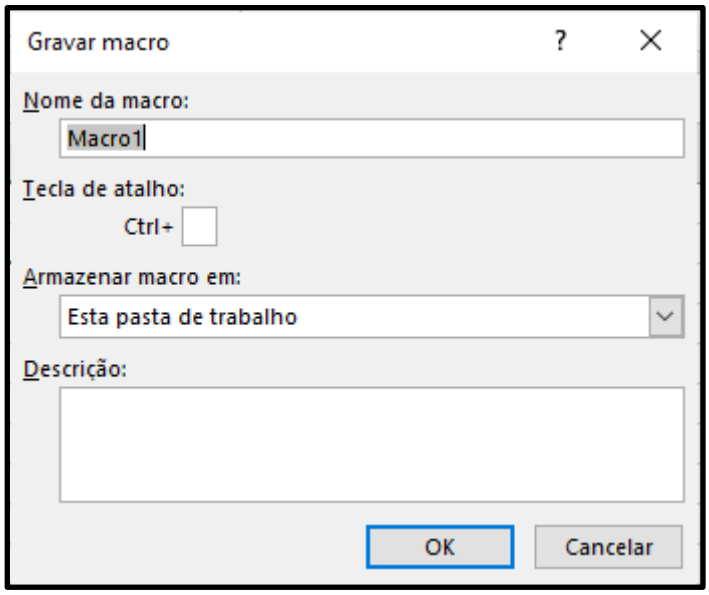
i. Seguindo os passos Arquivo > Opções > Personalizar Faixa de Opções > entre as opções em Guias Principais, marcar a guia Desenvolvedor e confirmar com Ok.

ii. Na Barra de Menus do Excel, selecione com o botão direito do mouse qualquer guia > Personalizar Faixa de Opções > entre as opções em Guias Principais, marcar a guia Desenvolvedor e confirmar com Ok.

c. Criação de Macros

Para criar Macros, deve-se ir a guia Desenvolvedor, marcar ou deixar desmarcada a opção Usar Referência Relativas e selecionar a ferramenta Gravar Macro, no grupo Código.

Em seguida será aberta a janela Gravar macro, nesta deve ser definido o nome da macro, um atalho para sua execução (opcional), onde esta macro será armazenada e uma descrição (opcional).



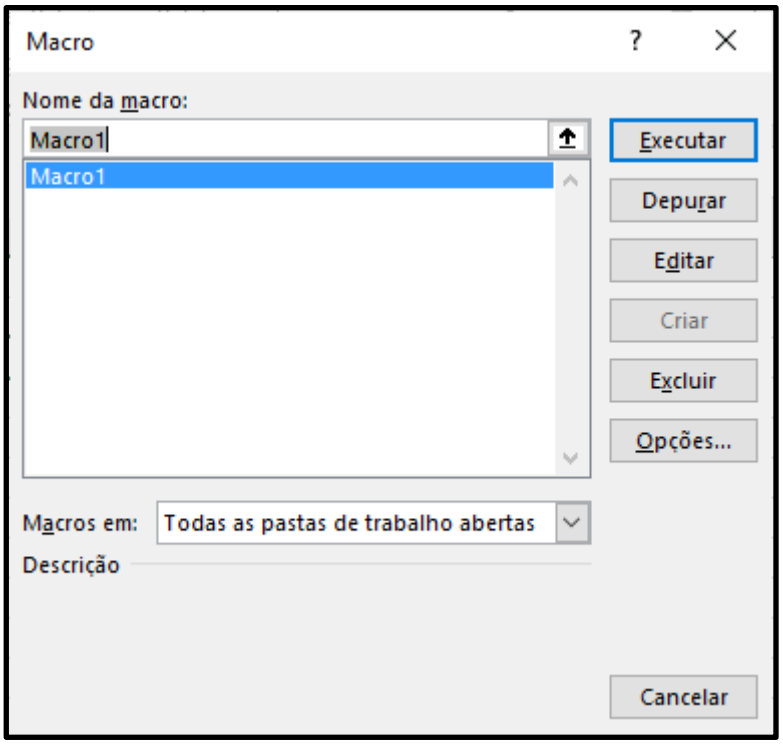
i. Nome da macro: Uma dica é escolher um nome objetivo e que exemplifique o que a macro executa. Lembre-se que existe uma restrição do Excel no nome da macro, não é possível usar a tecla espaço, podendo ser substituída pelo underline (_) ou outro caractere para separar o texto do nome;

ii. Tecla de atalho: Caso exista alguma função para a tecla de atalho escolhida pelo usuário a preexistente será apagada. Uma solução para este caso é sempre utilizar o Shift, junto com a padrão Ctrl, mais uma letra do teclado.

iii. Armazenar macro em: Se deseja salvar sua macro somente no arquivo que está gravando a macro, selecione *Esta pasta de trabalho.* A segunda opção, *Nova pasta de trabalho*, abrirá um novo arquivo Excel em branco para gravar e salvar sua macro. A primeira opção, *Pasta de trabalho pessoal de macros* (item g), salva sua macro em uma pasta específica de macros, uma vantagem é poder executar a macro gravada em todos os arquivos do seu Excel.

iv. Descrição: É destinado a descrever qual a função da macro gravada mais detalhadamente.

As macros da planilha podem ser acessadas em Desenvolvedor > Macros ou com o atalho de teclado ALT + F8. Para executá-la deve selecionar a que deseja na janela Macro e escolher a opção Executar ou utilizar o seu atalho escolhido. Depois de executá-la não é permitido desfazer a ação, seja selecionando a opção desfazer na barra superior ou com o atalho Ctrl + Z.



Nesta janela é possível excluir as macros armazenadas n'*Esta pasta de trabalho,* basta selecionar a macro que deseja e escolher a opção *Excluir*. Também pode editar sua macro através do Visual Basic, escolhendo a opção *Editar* será aberta a janela do VBE.

d. Referências Absolutas e Relativas

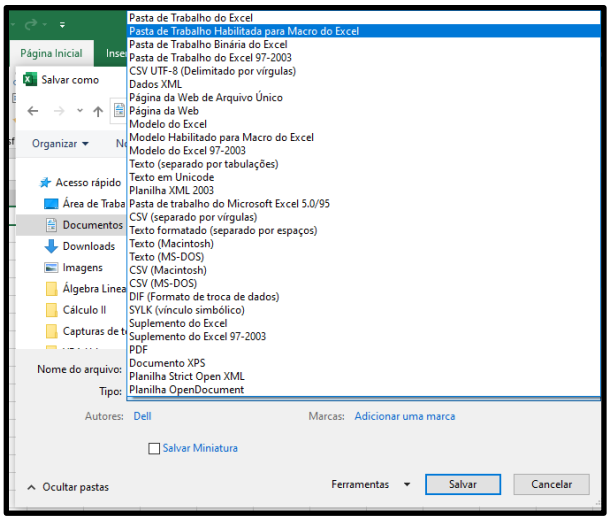
Abaixo da ferramenta Gravar Macro está a opção Usar Referências Relativas, quando selecionada a macro gravada a seguir usará Referências Relativas e quando não selecionada usará Referências Absolutas.

i. Referências Absolutas: a macro sempre será executada nas células indicadas em sua gravação;

ii. Referências Relativas: a macro executa suas ações a partir da célula selecionada no momento que executá-la (célula ativa).

e. Tipos de arquivo

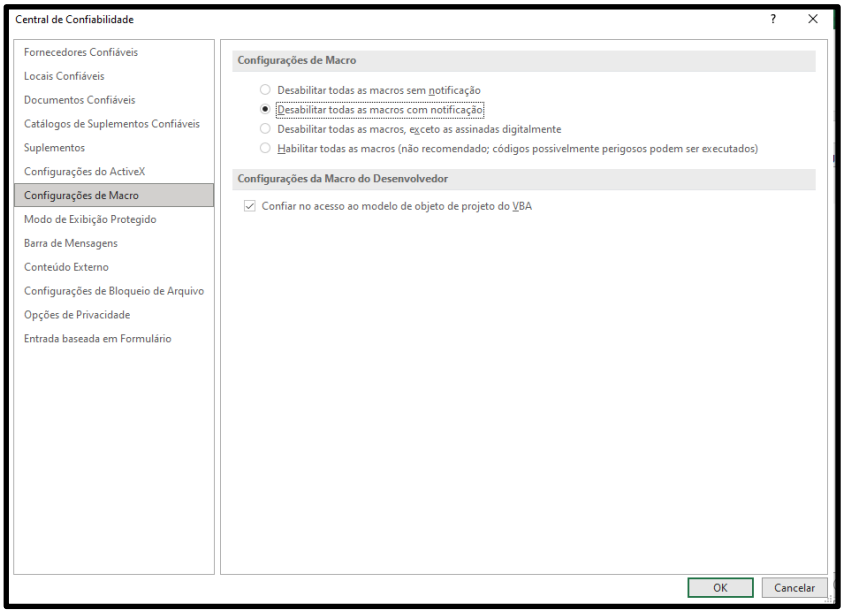
O tipo de arquivo padrão do Excel é Pasta de Trabalho do Excel (.xlsx), porém esse tipo não permite salvar as macros do arquivo. Para salvar o arquivo com as macros deve usar o tipo Pasta de Trabalho Habilitada para Macros do Excel (.xlsm). Para salvar sua macro deve seguir o caminho Arquivo > Salvar como ou o atalho de teclado **F12** e selecionar onde será salvo, o nome e o tipo do arquivo na janela.



f. Configurações de segurança

Arquivos com macros podem ser perigosos se não souber o que elas executam, podendo causar grandes problemas. Por isso não é recomendado que abra qualquer arquivo com macros se não souber sua origem.

As configurações de segurança podem ser acessadas por dois caminhos: Arquivo > Opções > Central de Confiabilidade > Configurações da Central de Confiabilidade ou Desenvolvedor > Segurança de Macro; Os dois caminhos abrem a janela Central de Confiabilidade, nesta janela, a esquerda, podem ser acessadas e alteradas as configurações de segurança do Excel. Em Configurações de Macro existem quatro opções:



i. Desabilitar todas as macros sem notificação: desabilitará todas macros do arquivo aberto sem notificação;

ii. Desabilitar todas as macros com notificação: quando um arquivo com macros for aberto perguntará se deseja desabilitar as macros, assim, caso o arquivo seja confiável, poderá habilitá-las;

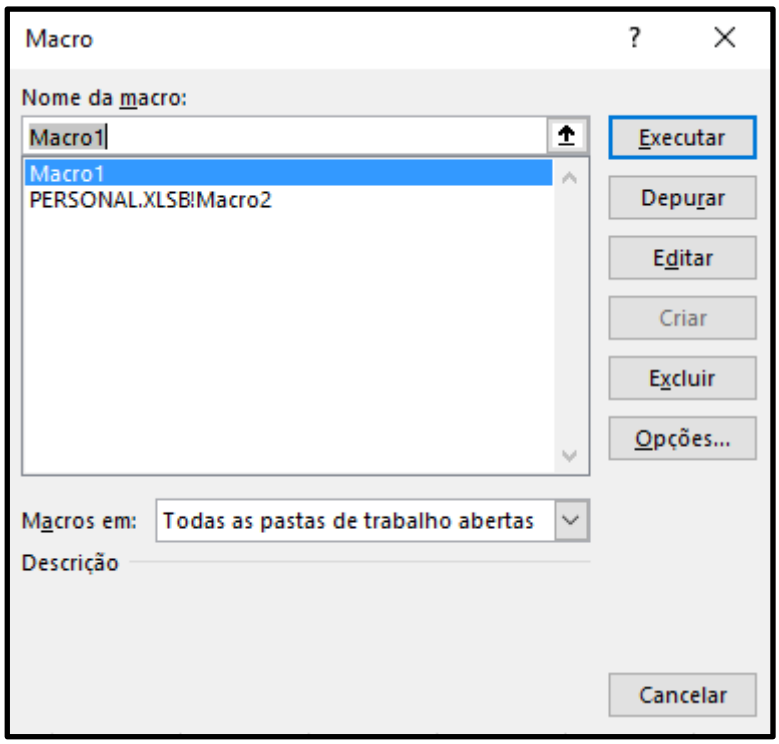
iii. Desabilitar todas as macros, exceto as assinadas digitalmente: desabilitará as macros, exceto se for assinada digitalmente por um editor confiável;

iv. Habilitar todas as macros: habilitará todas as macros do arquivo aberto, na opção o Excel já avisa que é perigoso usar esta opção.

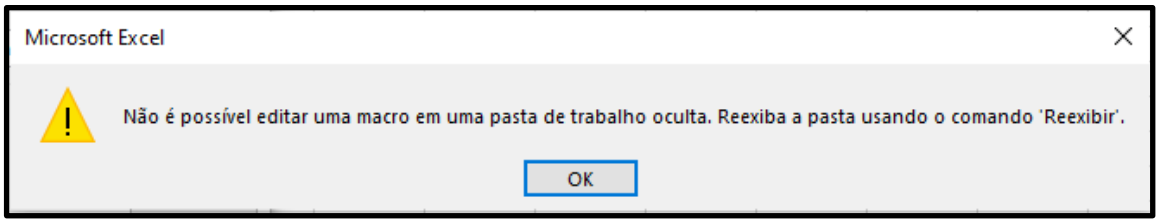
A opção mais indicada é a ii e a menos indicada é a iv. Selecione uma opção e confirme em Ok.

g. Pasta de Trabalho Pessoal de Macros

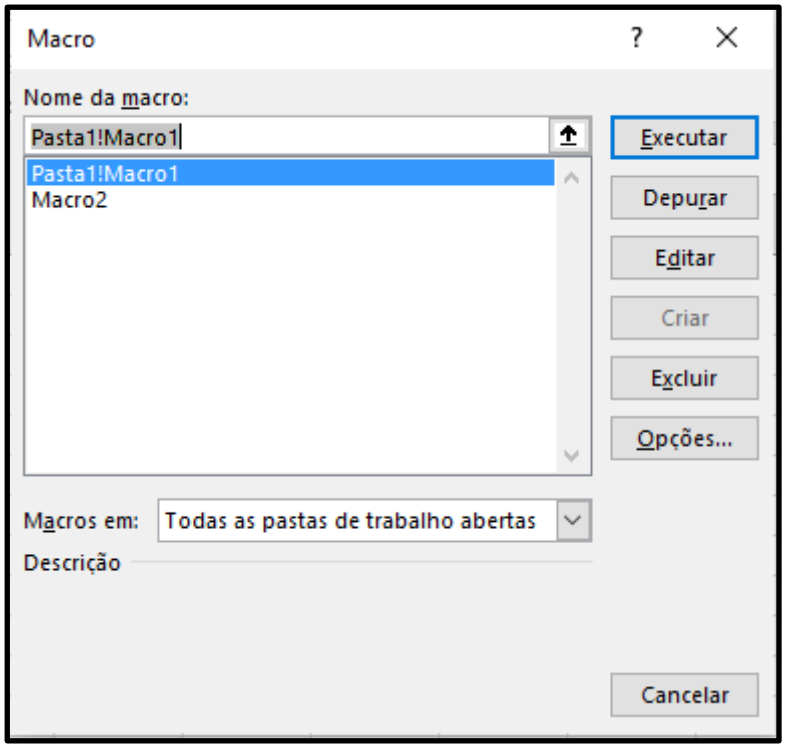
A Pasta de Trabalho Pessoal de Macros é criada quando a primeira macro é gravada na sua planilha. Como explicado no tópico 1.c, existem vantagens em armazenar sua macro nesta opção, a mais usual é poder executar esta macro em outras planilhas no seu Excel, inclusive nas que não estão habilitadas para macros. As macros armazenadas na Pasta de Trabalho Pessoal de Macros têm, antes do nome escolhido, o nome da pasta que está armazenada "PERSONAL.XLSB", como a "Macro2" na imagem a seguir.



Porém, apesar de estar na janela de macros, não é possível excluí-la na planilha aberta, e ao tentar surgirá a mensagem:



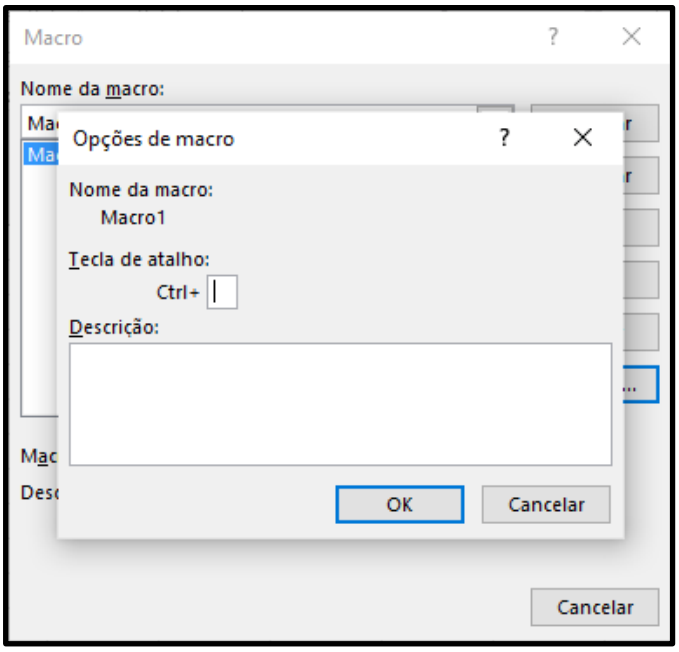
Para ‘Reexibir’ esta pasta de trabalho, que a priori sempre é oculta, deve ir na guia Exibir > Reexibir. A seguir será aberta a janela Reexibir com a pasta de trabalho oculta “PERSONAL”. Ao confirmar é aberta uma nova planilha em branco, na janela de macros, Desenvolvedor > Macros ou ALT+F8, as macros gravadas na Pasta de Trabalho Pessoal de Macros poderão ser modificadas. As macros de outras planilhas abertas, e somente as que são de planilhas abertas, também serão exibidas nesta janela com o nome do arquivo antes do seu nome e é possível excluí-la nesta planilha.



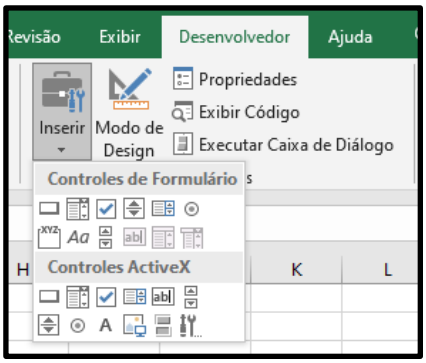
Para ocultar novamente esta planilha, deve seguir o caminho Exibir > Ocultar. Cuidado ao compartilhar planilhas com macros, caso elas estejam armazenadas na sua Pasta de Trabalho Pessoal de Macros não serão exibidas em outro Excel.

h. Formas de executar macros

Para executar uma macro foram apresentadas, até então, duas formas: na janela Macro a opção *Executar* e pelo atalho de teclado, caso não tenha definido um atalho antes de gravar a macro, pode adicioná-lo seguindo o caminho Desenvolvedor > Macros, na janela Macro selecione a macro que deseja e selecione *Opções…*, a janela Opções de macro será aberta, nela pode adicionar ou editar a tecla de atalho e a descrição.

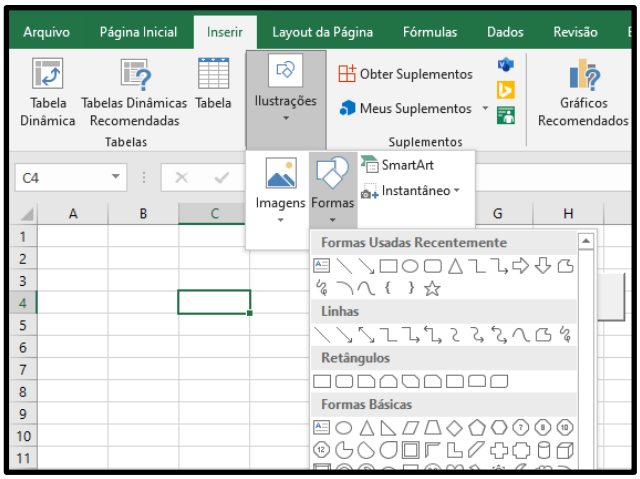


Outras opções para executar uma macro são: controle de formulário e utilizar formas do Excel. Para adicionar uma controle de formulário à sua macro deve ir na guia Desenvolvedor > Inserir > escolher um botão nas opções de Controle de Formulário, agora deve dimensionar o botão e em seguida abrirá a janela Atribuir macro, selecione a macro e confirme em Ok.



Surgirá uma nova guia Formatar, nela pode alterar a dimensão do botão. Para alterar seu nome é preciso somente dar um duplo click nele. Assim que selecionar qualquer área fora do botão, ele é ativado e quando selecionado executa a macro.

A opção de utilizar formas é transformar uma forma do Excel em um botão. Primeiro deve selecionar uma forma, vá na guia Inserir > Ilustrações > Formas, escolha uma forma. Na guia Formatar há mais opções de formatação do que no botão de Controle de formulário, além das dimensões e adicionar um texto (nome do botão), no grupo Estilo de forma e Estilo de WordArt, pode-se alterar a cor de preenchimento, contorno e efeito da forma e do texto.



Concluída a formatação é preciso atribuir uma macro a esta forma para que enfim seja um botão. Para isso, selecione a forma e abra as opções clicando com o botão direito do mouse sobre ela, selecione a opção Atribuir macro, na janela aberta escolha a macro que deseja e confirme em Ok. Assim, o botão já foi criado e quando selecionado executa a macro escolhida.

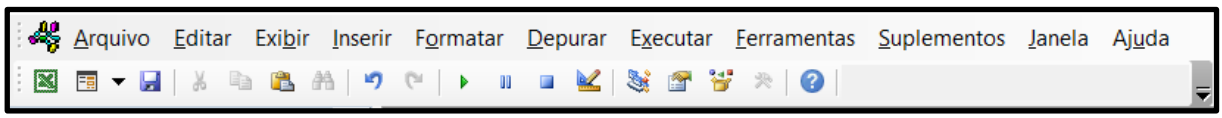
2. Introdução ao VBA

a. O que é o VBA para Excel?

VBA é a sigla para “Visual Basic for Applications” e permite que o usuário aplique recursos de programação em documentos do Microsoft Office. No Excel é utilizado para automatizar tarefas, criar funções, controlar e personalizar a planilha, tornando-a mais eficiente. Para isso, é preciso compreender a linguagem de programação do VB.

b. Conhecendo o Visual Basic Editor

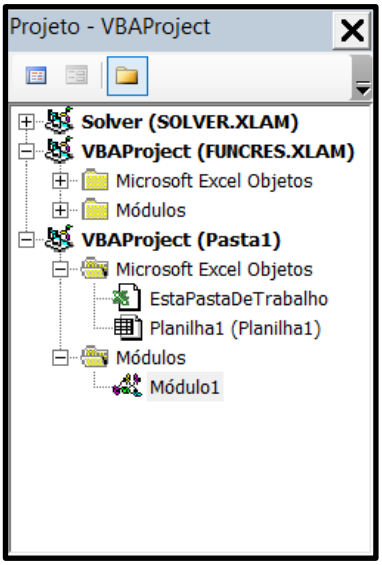
Para acessar o VBE abra o Excel e siga o caminho Desenvolvedor > VBE ou use o atalho **Alt+F11**.



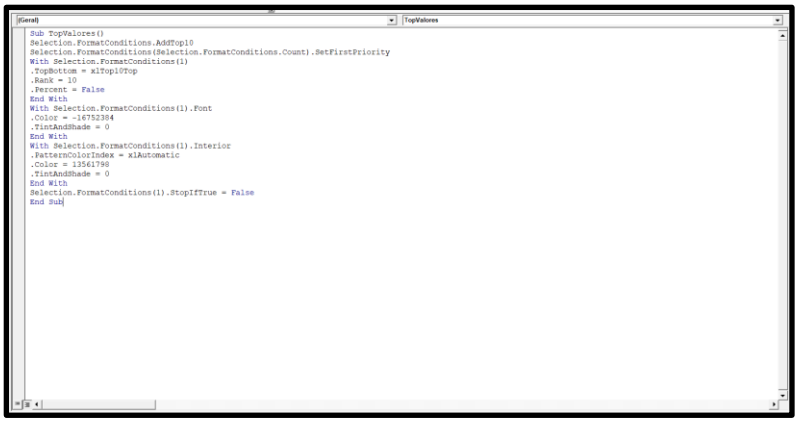
Na Barra de Menu poderão ser acessadas as diversas funcionalidades do VBA assim como a ajuda oferecida pelo software.

Ao acessar haverão 4 janelas principais do VBE:

i. Projeto: nela aparecem todas objetos que permitem inserção de códigos;

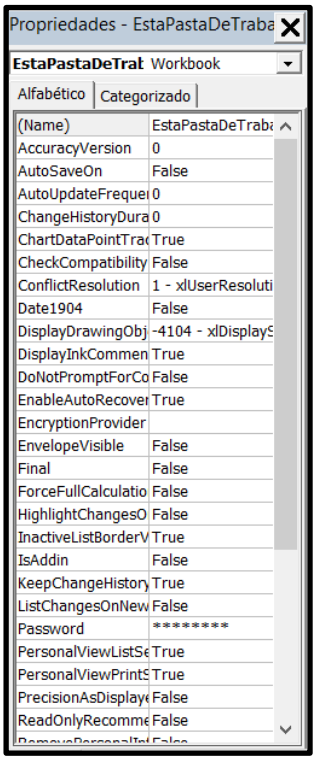


ii. Códigos: aqui são inseridos os códigos referentes ao objeto selecionado na janela Projeto. É aqui onde se faz a maioria da

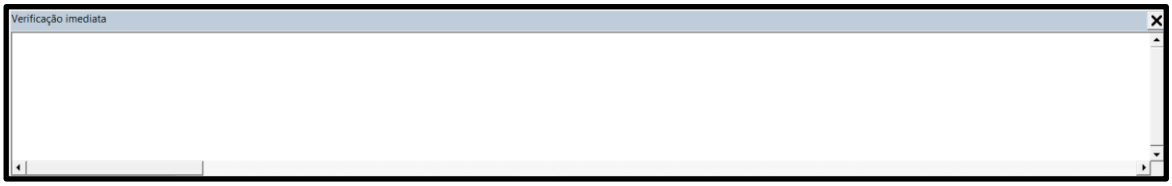


programação do VBA;

iii. Propriedades: Aqui você poderá identificar especificidades do objeto selecionado na janela Projeto.



iv. Verificação Imediata: aqui podem ser testadas as rotinas de códigos e macros criadas de maneira rápida.



c. Cores nos códigos do VBE

Ao escrever o código na janela de Códigos, o software automaticamente colorirá algumas partes deste fazendo com que o usuário tenha mais facilidade para organizar-se:

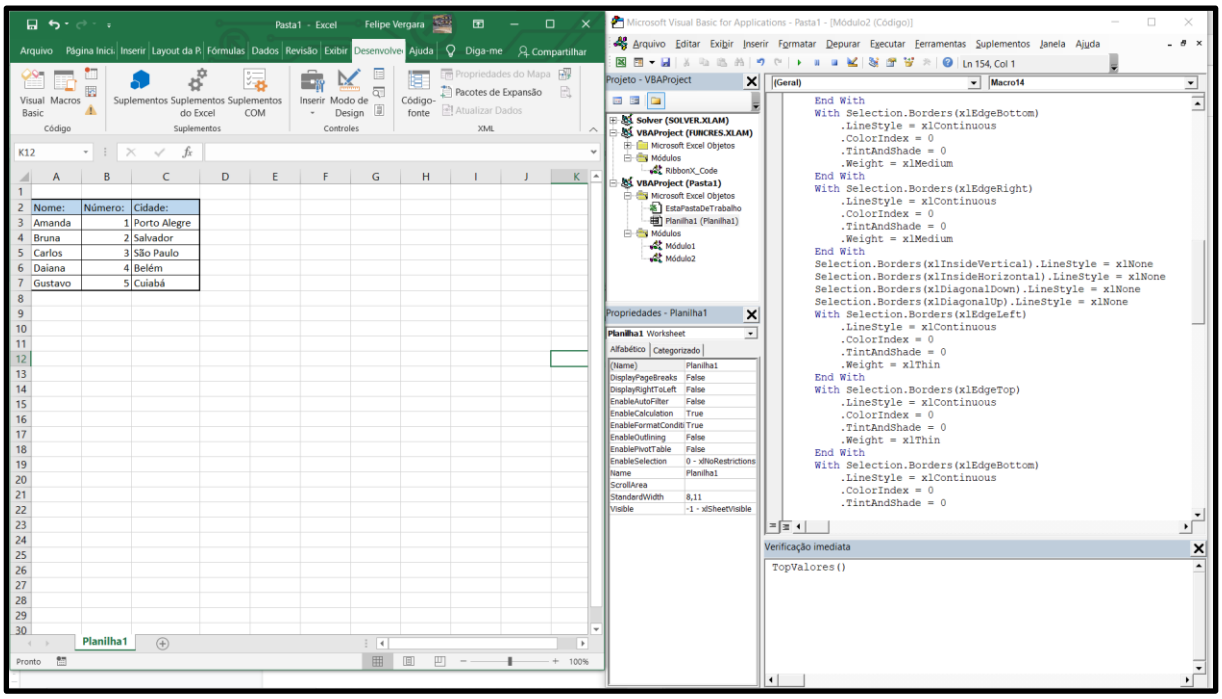
i. Azul: Palavras chave da linguagem de programação (sub, function, end sub e etc.);
ii. Vermelho: Código falho;
iii. Preto: Nomes de variáveis, procedimentos, valores e etc.
iv. Verde: Comentários;
v. Seleção amarela: Linha de código selecionada na depuração passo a passo.

d. Gravação de Macros e Janela de Códigos

Ao gravar uma macro o usuário pode visualizar todo o código referente a esta no VBA e agilizar o trabalho de não ter que escrever o código inteiro. Para isto deve-se:

i. Abra o VBE;

ii. Redimensione as janelas para que caibam simultaneamente no seu



monitor, ficando como na imagem a seguir;

iii. Selecione na janela “Projeto” a planilha onde será gravada a macro;

iv. Grave a macro e tenha acesso aos códigos gerados por esta macro.

e. Depuração passo a passo:

A depuração passo a passo, **F8**, é uma ferramenta que tem por finalidade auxiliar o usuário a verificar linha por linha o seu código. A linha do código ressaltada em amarelo no VBE será executada na planilha do Excel selecionada após acionar novamente o atalho **F8**. Para visualizar pode ser utilizada divisão da tela em duas janelas, VBE e Planilha, como na seção anterior. Desta forma podem ser identificados erros e em qual linha eles ocorrem.

```
Sub mensagem()
MsgBox ("Você está aprendendo VBA")
End Sub
```

f. Edição de Macros no VBE:

Caso haja uma macro gravada, os parâmetros desta podem ser modificados a partir do VBE. É útil em situações em que o usuário deseja realizar uma ou mais alterações a Macro sem ter que regravar toda ela. Para isto deve-se acessar a o código da Macro em questão (2d), identificar onde deve ser feita a mudança, utilização da depuração passo a passo pode facilitar e por último alterar o código manualmente de acordo com os novos parâmetros estabelecidos pelo usuário.

g. Objetos, Propriedades, Métodos e Eventos:

Os conceitos a seguir ajudarão a ter uma noção mais intuitiva de como se interrelacionam as funcionalidades do VBE, e auxiliarão o usuário a melhor utilizar a ferramenta, cometendo menos erros de lógica de programação.

i. **Objetos**: Um objeto representa um elemento de um aplicativo, planilha, célula, gráfico, formulário ou relatório. No código do VBE, você deve identificar um objeto antes de aplicar um dos métodos do objeto ou alterar o valor de uma das suas propriedades. Neles podem ser realizadas alterações e executados procedimentos. Os principais são: Application, Workbooks, Sheets e Range.

ii. **Propriedades**: Uma propriedade é um atributo que define uma das características do objeto, como tamanho, cor ou localização na tela, ou um aspecto do comportamento dele, como se o objeto está habilitado ou visível. Para alterar as características de um objeto, deve-se alterar os valores de suas propriedades. Para definir o valor de uma propriedade, siga a referência a um objeto com um período, o nome da propriedade, um sinal de igual (=) e o novo valor da propriedade.

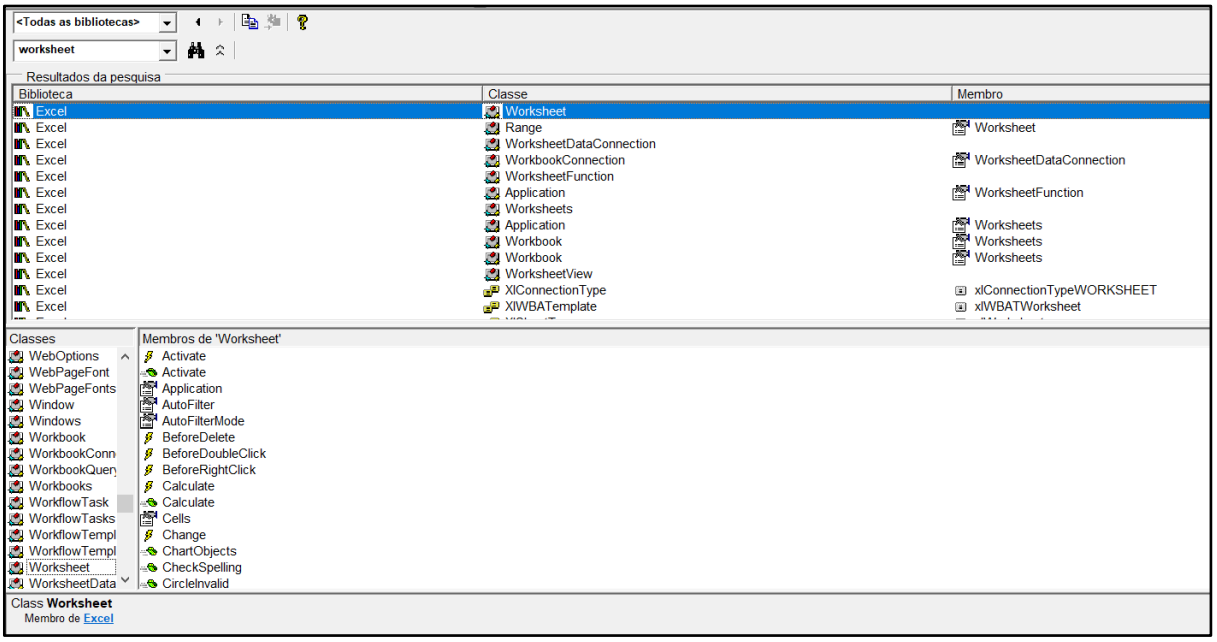
iii. **Método**: É uma ação que um objeto pode executar. Por exemplo, executar uma função matemática (método) em uma célula desejada (objeto).

iv. **Evento**: É uma ação reconhecida por um objeto, como clicar com o mouse, selecionar uma célula e salvar ou encerrar a pasta de trabalho. Os eventos podem ocorrer como resultado de uma ação do usuário ou de um código do programa, ou podem ser disparados pelo sistema.

h. Ajuda e Pesquisa de Objetos:

A ferramenta “Ajuda” pode ser acessada de duas maneiras: indo na própria guia “Ajuda” no Barra de Menus do VBE ou pelo atalho **F1**. Ao acessar pela Barra de Menus é aberto o site da Microsoft para dúvidas gerais do VBA. Caso haja uma palavra da janela de Códigos selecionada ao apertar **F1** é aberta uma página específica de dúvidas com conceitos, exemplos e imagens relacionadas ao objeto, propriedade, método e evento selecionado.

Utilizando o atalho **F2** pode ser acessada a Pesquisa de Objetos. Aqui podem ser procuradas propriedades, objetos e métodos referentes ao termo pesquisado.



3. Procedimentos e Funções

a. Módulos

Os módulos são objetos localizados na janela de Projeto que permitem a escrita de códigos, na janela de Códigos, ao serem selecionados. Os módulos podem alterar funções de células, planilhas, pasta de trabalho, do aplicativo Excel e até do Sistema operacional do computador. Por isto você deve ser cauteloso com os códigos que você executa no seu VBA, especialmente aqueles baixados da internet.

Para inserir um novo módulo deve-se dar clique direito na janela Projeto > Inserir > Módulo. Para alterar o nome do módulo deve-se ir à coluna ao lado de "(NAME)" na janela de Propriedades. Após a criação de um novo módulo, utiliza-se a janela de Códigos para estabelecer quais procedimentos executará o módulo a partir dos Objetos, Propriedades, Métodos e Eventos programados pelo usuário.

Módulos podem ser importados e exportados (salvos) no VBA. Em ambos casos utiliza-se o formato .bas da seguinte maneira:

i. Importar: clique direito na janela Projeto > Importar arquivo > Seleção do local do arquivo .bas computador;
ii. Exportar: clique direito na janela Projeto > Exportar arquivo > Seleção do local para salvar o arquivo .bas computador.

Para remover o módulo selecione-o > clique direito > Remover “nome do módulo”. Neste momento o excel oferece 5 opções, fechar, cancelar, exportar o módulo + remover, não exportar o módulo + remover e Ajuda.

b. Procedimento SUB

Declara o nome, argumentos e o código que formam o corpo de um procedimento Sub. Deve ser formatado seguindo estes parâmetros de sintaxe:

```
Sub nomesub ()

Corpo do código

End Sub
```

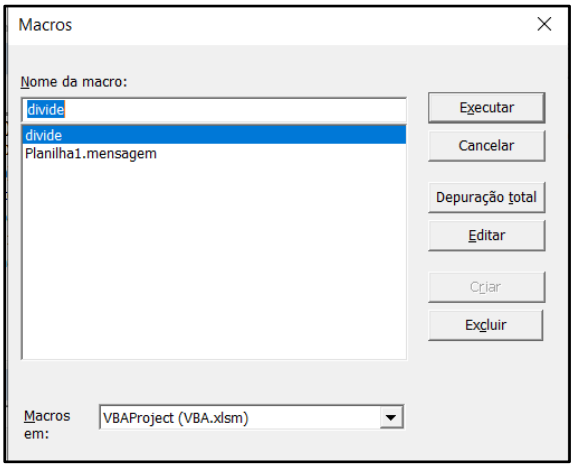
Após escrever seu código e nomeando o Sub é possível executá-lo, caso não tenha erros de compilação. Para isto existem as seguintes opções, clicando no botão *play* verde na barra de menus, usando o atalho **F5,** colocando o nome da Sub na janela de Verificação Imediata ou mesmo utilizando a Depuração Passo a Passo **F8.** Caso o cursor esteja inserido dentro do código, este será executado automaticamente. Caso contrário o software abrirá uma tabela de macros para que o usuário selecione aquela a ser executada.

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

End Sub
```

Neste caso, o cursor está situado dentro do código, circulado em vermelho. Ao aplicar qualquer um dos métodos mencionados acima, este será executado imediatamente.

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

End Sub
```

Neste segundo caso, o cursor está fora do código, portanto, será aberta a tabela a seguir:



Basta selecionar a Macro e clicar em executar ou pressionar enter para executar.

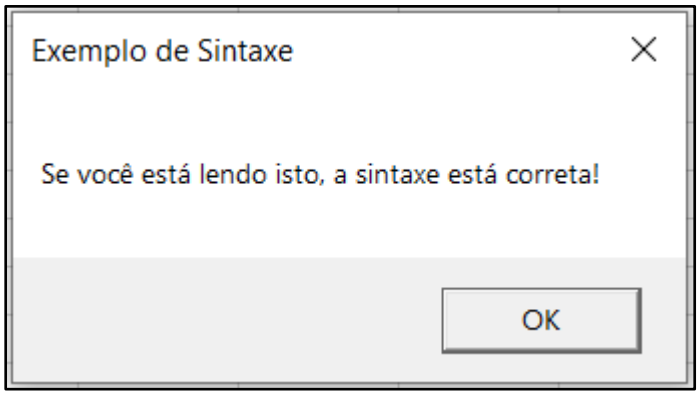
c. Noções Básicas sobre Sintaxe

A sintaxe é a formatação correta na qual devem ser escritos os comandos. Ao seguir a sintaxe correta no código, o VBA pode "ler" e executar exatamente o que o usuário deseja. Caso contrário surgirão problemas, quer sejam de compilação (o código não roda) ou de lógica (o código roda mas não executa exatamente o desejado pelo usuário). Para saber da correta sintaxe basta escrever o comando, ainda sem especificar os parâmetros e o VBA exibirá opções de sintaxe. Depende do usuário a seleção e a escrita do comando, agora sim especificando os parâmetros daquilo que se deseja executar. Maiores detalhes e exemplos podem ser encontrados pressionando **F1** e procurando o comando desejado no site da Microsoft.

```
Sub mensagemo()

MsgBox
MsgBox(Prompt, [Buttons As VbMsgBoxStyle = vbOKOnly], [Title], [HelpFile], [Context]) As VbMsgBoxResult
End Sub
```

Neste exemplo o usuário vai executar o comando MsgBox (uma caixa de mensagem a ser exibida no excel). Ao escrever o nome do comando aparecem as opções corretas e variadas de sintaxe do comando. Após decidir como ele desejava executar o comando, acabou escrevendo o seguinte:

```
Sub mensagemo()

MsgBox "Se você está lendo isto a sintaxe está correta!", vbOKOnly, "Exemplo de Sintaxe"
End Sub
```



E ao executar (**F5**):

d. Procedimento Function

Podem ser criadas novas funções utilizando o VBA, completamente adaptáveis pelo usuário. Após a criação da função, esta pode ser utilizada igualmente às funções predefinidas no excel. A sintaxe é a seguinte:

function NOMEFUNÇÃO (parâmetro 1, parâmetro 2, … parâmetro N)

'Procedimentos da função'

End function

Exemplo: O usuário, engenheiro da área de transportes, deve calcular a velocidade média dos 67 caminhões da empresa em que trabalha para verificar se os itinerários estão sendo cumpridos corretamente. Para facilitar seu cálculo ele criou a seguinte função:

```
Function VELOCIDADEMEDIA(distância, tempo)
VELOCIDADEMEDIA = distância / tempo
End Function
```

Para executar basta ele acessar o Excel e aplicar igual a qualquer outra função:

Distância (km)	Tempo (h)	Velocidade média (km/h)
136	2	68
133	4	33,25
133	2	66,5
200	2	100
300	2,5	=VELOCIDADEMEDIA(B8;C8)

4 Elementos essenciais da programação VBA

a. Uso de comentários

Os comentários no VBA servem para auxiliar a compreender o código. O comentário NÃO é lido pelo depurador durante a execução do código.

Como o comentário não faz parte do código propriamente dito, ele deve ficar na cor verde e ser iniciado com um apóstrofo (‘).

```
Sub mostrar_abas()

'vai ocultar a aba comentarios

Sheets("Comentarios").Visible = True

End Sub
```

b. Declaração de variáveis

Para declarar variáveis, antes de iniciar o código, devemos colocar a opção *Option Explicit.*

A declaração de variáveis é simples, basta utilizar o dim, colocar o nome da variável e o tipo de dado que é desejado.

Ex → dim x as double; dim y as integer, dim z as string…

O tipo de variável pode ser diverso. Os mais comumente utilizados são:

- *String*: Contém texto - até 65.000 caracteres. Pode limitar usando *;
- *Double*: Contém números com ou sem casa decimal;
- *Integer*: Contém apenas números inteiros
- *Date*: Contém datas no formato = #08/21/2017# e horário #10:45:30#
- *Variant*: Aceita qualquer dado, é usado em variáveis não declaradas.

```
Option Explicit

Sub escopo1()

Dim x As Double
Dim y As Double

x = 10
y = 20
resultado = "Sua resposta é " & x + y
MsgBox resultado

End Sub
```

Toda variável tem seu escopo de uso, ou seja, o lugar onde ela está sendo declarada e utilizada. Ela pode ser declarada junto ao Option Explicit para ser utilizada em todo módulo, independente do número de códigos que existem no módulo; ou pode ser declarada em um módulo e ser usada em outro.

Para a primeira opção temos:

```
Option Explicit
Private x As Double
Private y As Double
Private resultado As String
```

Para a segunda:

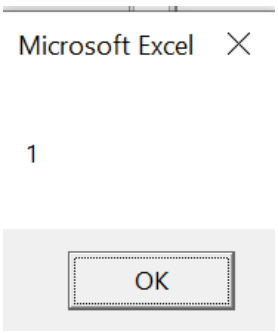
```
Option Explicit
Public x As Double
Public y As Double
Public resultado As String
```

c. Função STATIC e armazenamento de variáveis

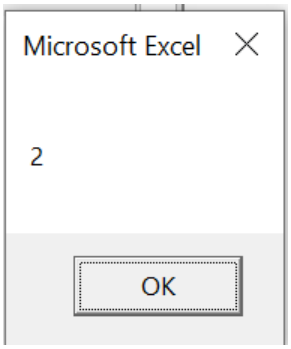
No VBA as variáveis são descartadas ao final da execução de um código. Para que elas sejam armazenadas existe a opção STATIC; essa opção armazena o valor final que a variável assume e continua a execução do código, tornando a soma acumulativa.

```
Option Explicit

Sub expiracao()

Static x As Double
x = x + 1
MsgBox x

End Sub
```

Primeira execução:



Segunda execução:



Ao utilizar a função Static é preciso estar atento para possíveis erros na depuração e no código, se existir algum, a sequência é quebrada e zerada, retornando ao valor inicial.

d. Uso de constantes

Podemos facilitar o trabalho no VBA por meio do uso de constantes. Elas são valores fixos que, uma vez declarados, podem ser usados em todo o módulo (private) ou em toda janela do VBE (public). Sua declaração pode ser feita dentro do Option Explicit, assim como a declaração de variável, ou fora dele.

```
Sub constantes()

Dim x As Long
Dim resultado As Double
Const potencia2 As Integer = 2

x = 10
resultado = x ^ potencia2
MsgBox resultado

End Sub
```

O VBA tem várias constantes previamente definidas, um dos exemplos são as constantes associadas às cores. Podemos procurar essas constantes colocando um ponto de interrogação e a cor que queremos (ex: ?green).

e. String

A variável do tipo String é uma armazenadora de texto do VBA. Sua declaração é simples e armazena muitos caracteres, por padrão. Porém, quando é mais conveniente, podemos delimitá-la:

EX: Dim x As String * 10

Com essa declaração ela irá mostrar até ao décimo caracter da frase, texto, palavra para a qual x está declarado.

f. Função DATA e HORÁRIO

Essas duas funções têm declaração bem simples:

DATA = #mes/dia/ano#

HORÁRIO = #hora:min:segundo#

Para utilizá-las basta completar a data e o horário com os valores desejados. Lembrando que, tanto a data quanto o horário seguem padrões americanos, ou seja, a data

inverte mês e dia no CÓDIGO e o horário, para 17h por exemplo, o código automaticamente se altera para 5 PM.

g. Operadores aritméticos

São operadores utilizados em cálculos.

- Adição +
- Subtração -
- Multiplicação *
- Divisão /
- Exponenciação ^
- Concatenação &
- Divisão inteira \
- Resto da divisão MDD

Para os operadores aritméticos é importante prestar atenção conforme o uso de parênteses, pois o VBA realiza primeiro o que está contido dentro deles.

h. Operadores de comparação

São operadores que comparam valores entre si. Para os operadores de comparação também devem ter atenção quanto ao uso de parênteses.

- Igual =
- Diferente <>
- MAior >
- Menor <
- Maior igual/ menor igual: >=; <=;

i. Operadores lógicos

Podem ser utilizados para complementar os operadores de comparação. Os mais utilizados são os seguintes:

- Negação - NOT
- E - AND (aceito se satisfazer as duas condições)
- Ou - OR (aceito se satisfazer só uma das condições)

j. Vetores/Matriz unidimensional

Vetores, no código VBA, são armazenadores de valores de forma linear. É importante lembrar que para vetores, o código VBA assume a posição zero sempre, ou seja, todo vetor declarado tem a posição de número zero. Por exemplo, se você quer declarar um vetor de 3 posições você o fará das seguintes formas:

- dentro do código

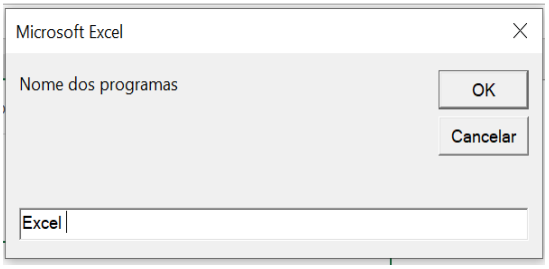
```
(Geral)
Sub vetor()

Dim vetor(1 To 3) As String
vetor(1) = "Excel"
vetor(2) = "Office"
vetor(3) = "Power Point"

End Sub
```

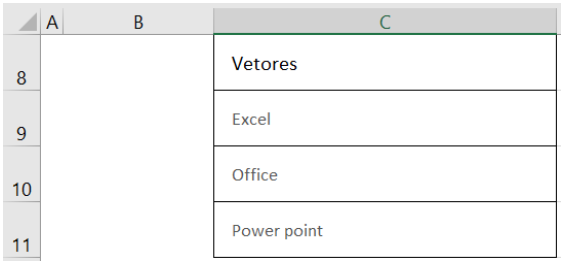
- Antes do código iniciar

```
(Geral)
Option Base 1

'utilizando o option base 1 o VBA entende que seu vetor
'iniciará a partir da posição número 1.

Sub vetor()

Dim vetor(3) As String
vetor(1) = "Excel"
vetor(2) = "Office"
vetor(3) = "Power Point"

End Sub
```

Os vetores podem servir para completar um intervalo dentro da planilha através do próprio usuário por meio da função Range:

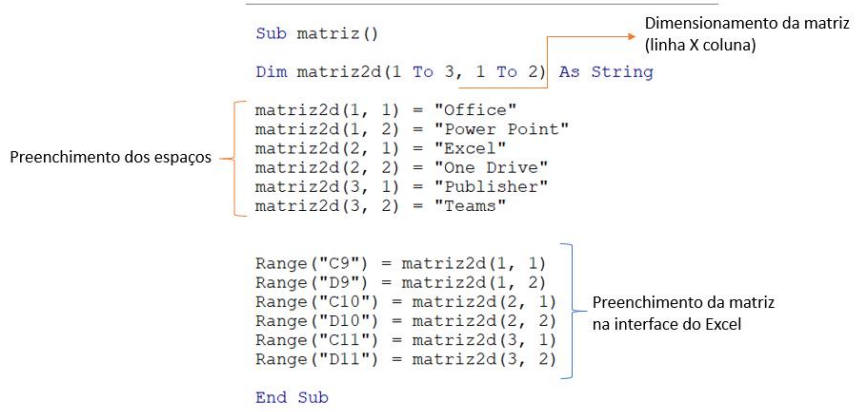
```
Option Base 1
Sub vetor()

Dim vetor(3) As String

For a = 1 To 3
    vetor(a) = InputBox("Nome dos programas")
Next

Range("C9") = vetor(1)
Range("C10") = vetor(2)
Range("C11") = vetor(3)

End Sub
```





k. Matrizes

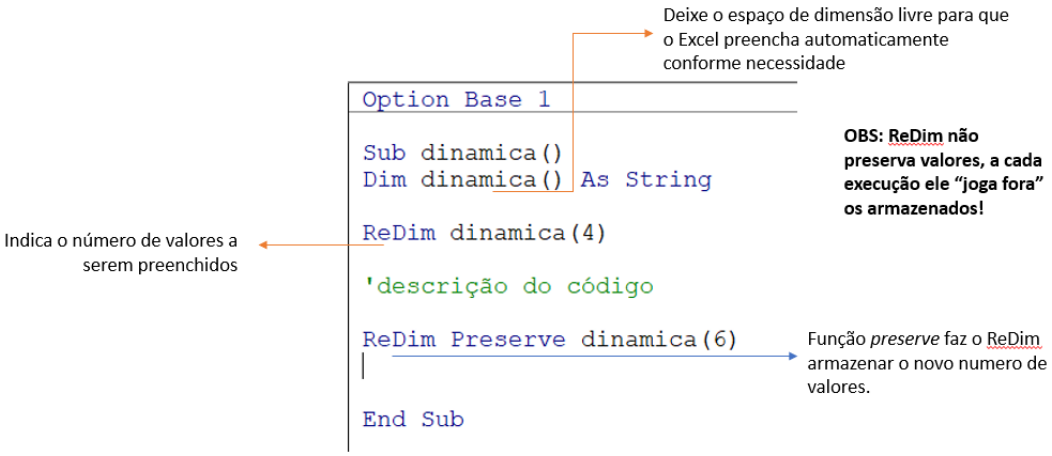
Matrizes são vetores bidimensionais ou tridimensionais, que existem no VBA.



As matrizes podem ser preenchidas pelo usuário através do InputBox também, da mesma forma dos vetores. Podemos declarar uma matriz cúbica, com três dimensões declarando 3 espaços onde, para duas dimensões, declarávamos dois.

I. Matriz dinâmica

É uma matriz que tem seus espaços preenchidos de acordo com a necessidade. Ou seja, o programador não delimita um intervalo de colunas e linhas.



5 Manipulação de Objetos no excel

Objetos

Um objeto representa um elemento de um aplicativo, como uma planilha, uma célula, um gráfico, um formulário ou um relatório. No código do Visual Basic, você deve identificar um

objeto antes de aplicar um dos métodos do objeto ou alterar o valor de uma das suas propriedades. Os objetos mais utilizados são:

- **Application**: Excel em si;
- **Workbooks**: Pastas de trabalho;
- **Sheets**: Abas de uma planilha;
- **Range**: Intervalo de células.

Métodos e propriedades

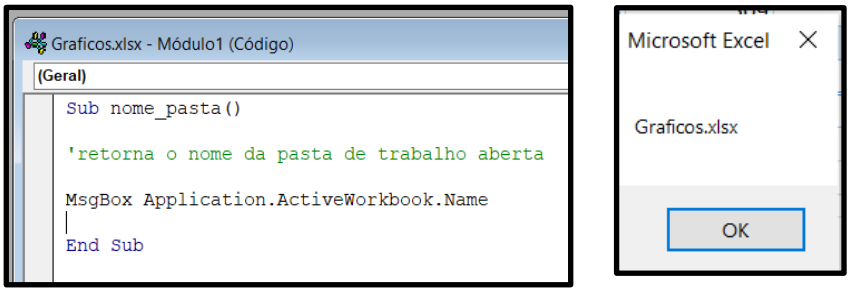
Após retornar um objeto, que representa o elemento apropriado que irá ser trabalhado, o Visual Basic manipula aquele objeto usando suas propriedades e métodos.

As propriedades são características ou atributos de um objeto e os métodos são ações que os objetos podem executar.

a. Objeto Application

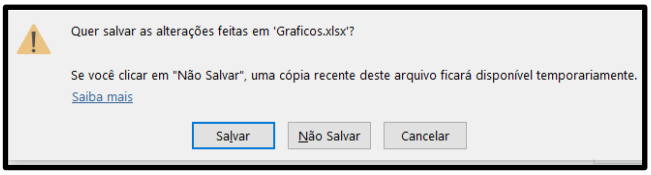
O objeto Application representa todo o aplicativo Microsoft Excel e seus derivados. Através dele, podemos configurar a visualização, execuções e outras funcionalidades do Excel. O Objeto Application possui várias propriedades e métodos:

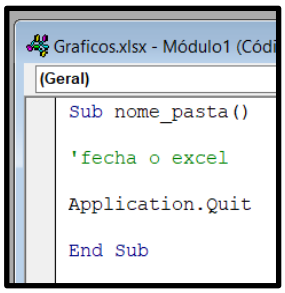
- **Método Activeworkbook:** Retorna a pasta de trabalho ativa;



- **Método Activesheet**: Retorna a planilha ativa na pasta de trabalho, ela é utilizada de forma análoga ao exemplo acima;

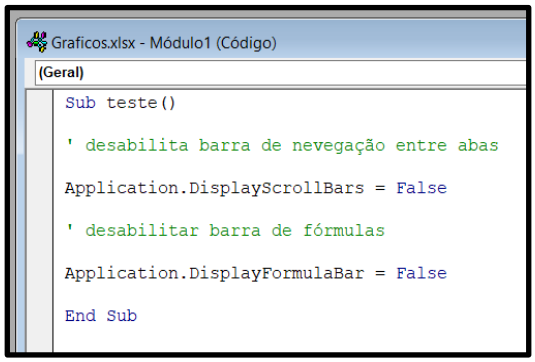
- **Método Quit:** Fecha as pastas de trabalho ativas;

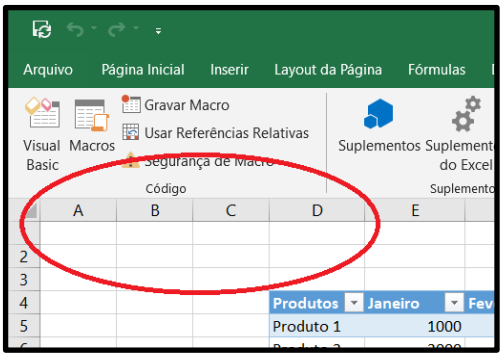




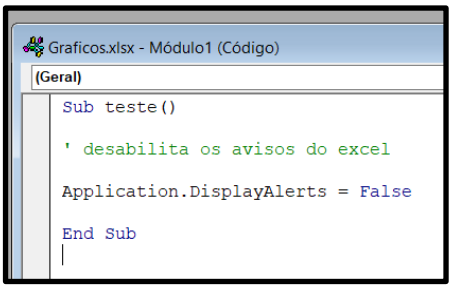
- **Propriedade DisplayScrolBars:** Habilita e desabilita a barra de navegação entre as abas do excel. É necessário atribuir o valor verdadeiro (true) ou falso (false).

- **Propriedade DisplayFormulaBar**: Habilita e desabilita a barra de fórmulas do excel e também precisa da atribuição de valor (true or false).





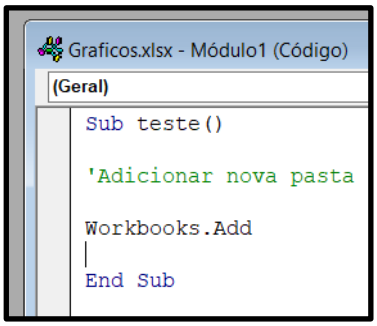
- **Propriedade Displayalerts**: Habilita e desabilita os avisos do excel e precisa da atribuição de valor (true or false). Se a planilha for finalizada, por exemplo, não será dado o alerta solicitando o salvamento.



b. Objeto Workbook

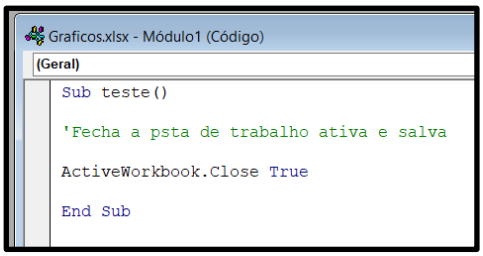
O objeto Workbook representa uma pasta de trabalho do Microsoft Excel. O objeto pasta de trabalho é um membro da coleção Workbooks. A coleção de pastas de trabalho contém todos os objetos Workbook atualmente abertos no Microsoft Excel.

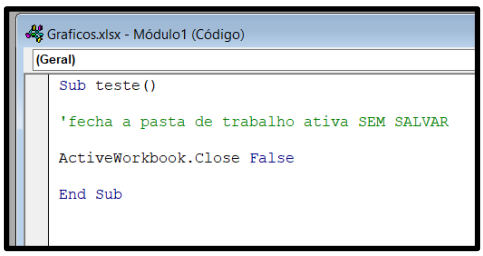
- **Método Add**: O método Add permite criar uma nova pasta de trabalho vazia e adicioná-la à coleção.



-

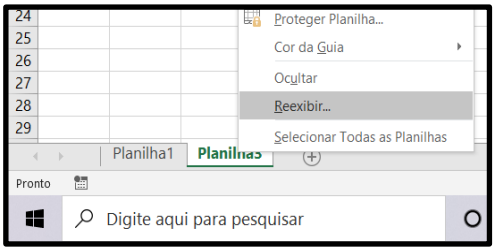
- **Método Close:** Fecha a pasta de trabalho ativa. O método close tem alguns parâmetros, ele verifica se a propriedade Saved do objeto Workbook é true ou false. Se for true fechará a pasta e salvará, caso contrário, fechará a pasta sem salvar.



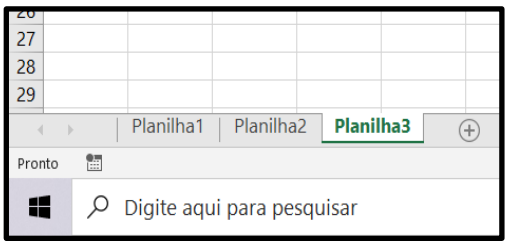


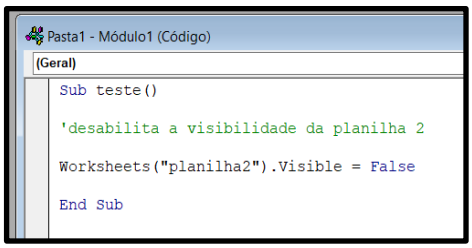
c. Objeto Worksheets

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

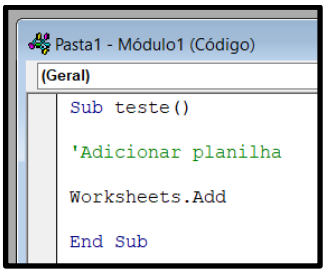


- **Propriedade Visible:** habilita ou desabilita a visibilidade da planilha indicada da pasta de trabalho ativa.

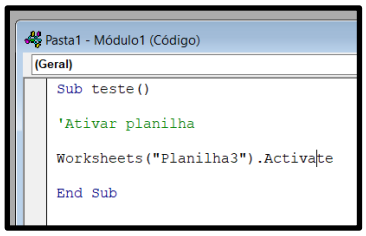




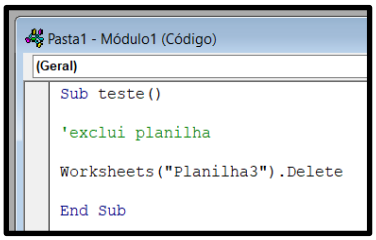
- **Método Add:** Permite adicionar uma nova planilha na pasta de trabalho ativa.



- **Método Activate:** Torna uma planilha indicada da coleção WorkSheets ativa.

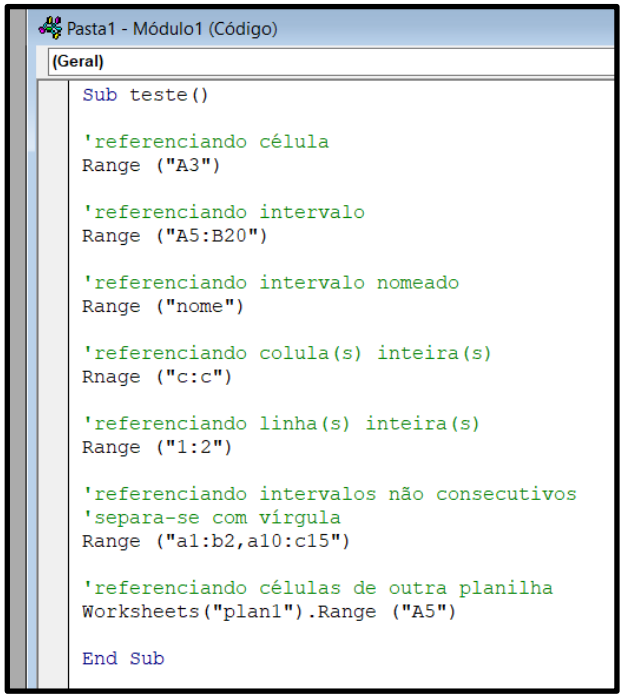


- **Método Delete**: Exclui a planilha indicada da coleção WorkSheets.

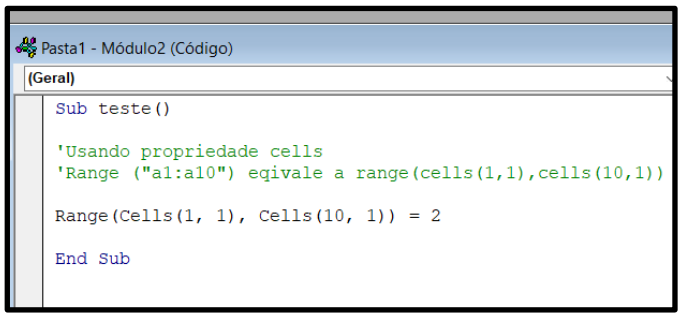


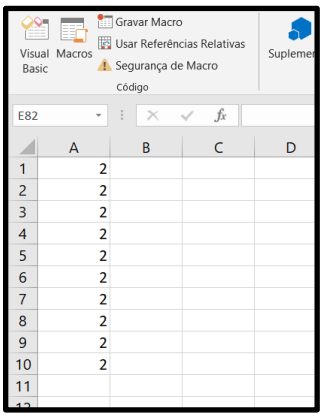
d. Objeto Range

Esse objeto representa uma célula, uma coluna, uma linha, um conjunto de células, até todas as células de uma planilha. O objeto Range talvez seja o objeto mais utilizado no Excel VBA, e para ser usado deve-se referenciar a célula ou conjunto de células que será manipulada.

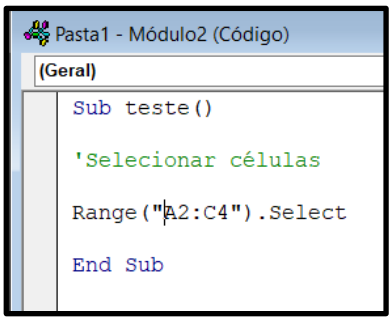


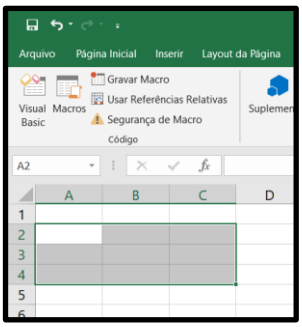
- **Propriedade Cells:** A propriedade Cells refere-se a todas as células do objeto especificado no intervalo. Podemos utilizá-la com o objeto Range representando todas as células de dentro do range.



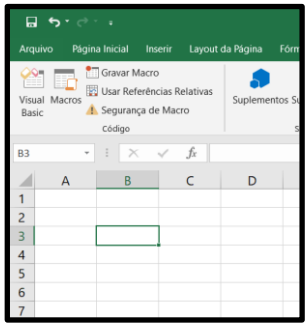


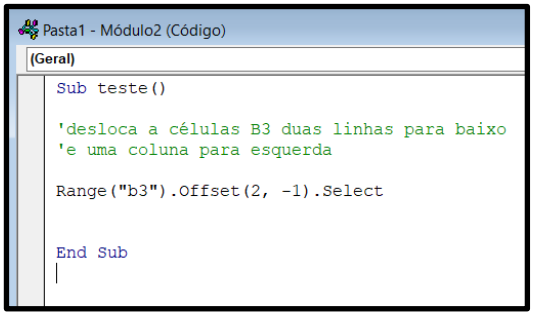
- **método select:** Esse método seleciona todas as células indicadas pelo intervalo.

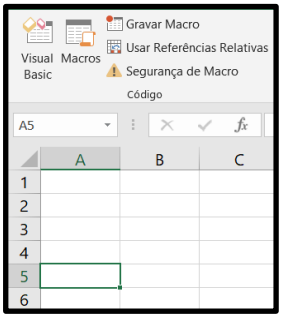




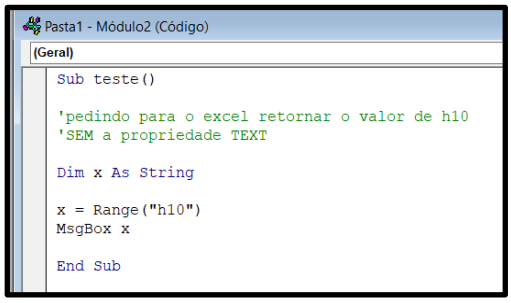
- **Propriedade offset:** Essa propriedade desloca a célula indicada de acordo com os valores fornecidos. Primeiro deve-se fornecer o números de linhas a ser deslocado, números positivos deslocam a célula para baixo e números negativos deslocam para cima. Em seguida, deve-se fornecer o número de colunas a ser descolado, números positivos deslocam a célula para direita e números negativos deslocam para esquerda.

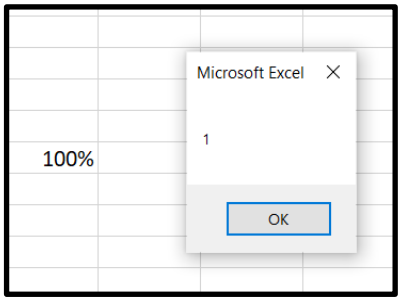


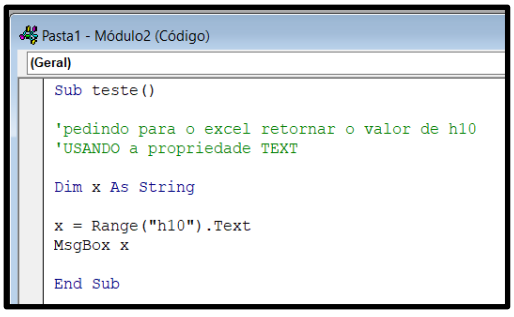


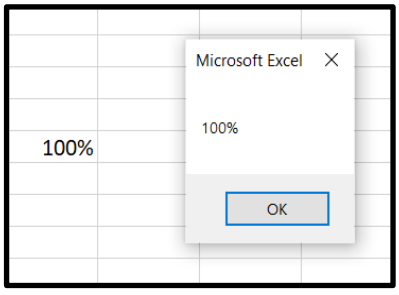


- **Propriedade text**: Essa propriedade é utilizada para que o excel retorne a formatação na célula, não apenas o valor numérico.

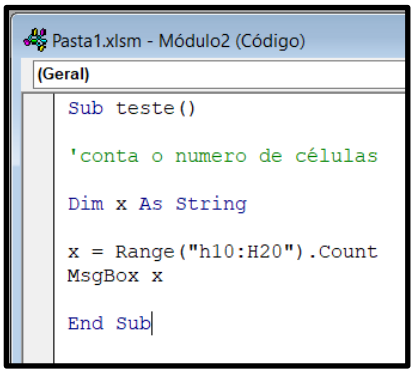


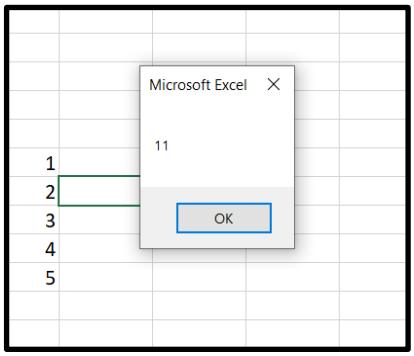




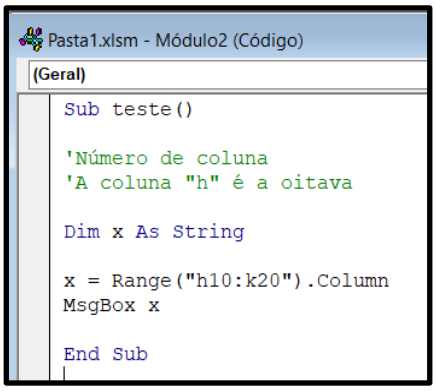


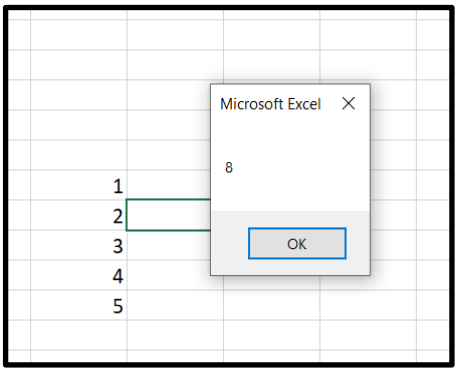
- **Propriedade count:** Essa propriedade conta o número de células do intervalo indicado.



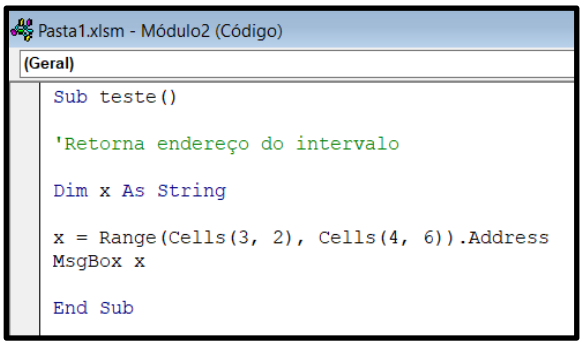


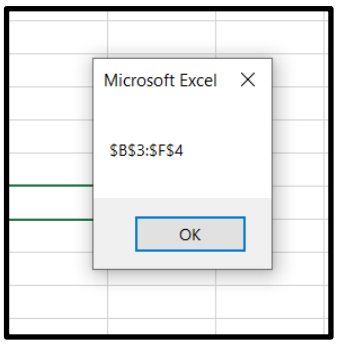
- **Propriedade Column**: Retorna o número da primeira coluna do intervalo de células indicado no código.



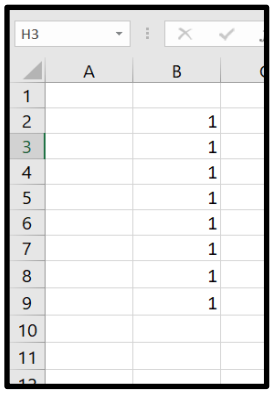


-
- **Propriedade Row**: Essa propriedade retorna a primeira linha do intervalo de células indicado no código. Ela é utilizada de forma análoga à propriedade anterior.
-
- **Propriedade Address:** Retorna o endereço do intervalo de células indicado no código.

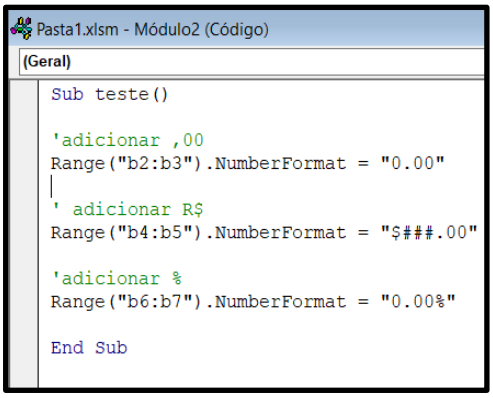




- **Propriedade NumberFormat:** Essa propriedade altera a formatação das células indicadas. Deve-se informar ao código o formato desejado sempre entre aspas.



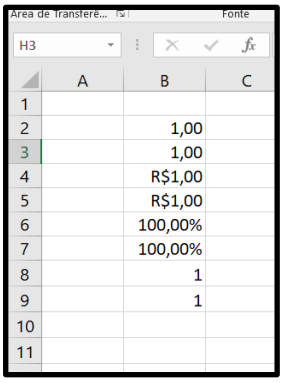
	A	B
1		
2		1
3		1
4		1
5		1
6		1
7		1
8		1
9		1
10		
11		



Pasta1.xlsm - Módulo2 (Código)

(Geral)

```
Sub teste()

'adicionar ,00
Range("b2:b3").NumberFormat = "0.00"

' adicionar R$
Range("b4:b5").NumberFormat = "$###.00"

'adicionar %
Range("b6:b7").NumberFormat = "0.00%"

End Sub
```



	A	B	C
1			
2		1,00	
3		1,00	
4		R$1,00	
5		R$1,00	
6		100,00%	
7		100,00%	
8		1	
9		1	
10			
11			

- **Propriedade formulas:** Essa propriedade é utilizada para fazer cálculos diretamente no Visual Basic. Ela funciona de forma semelhante às fórmulas na interface do VBA, mas deve-se tomar cuidado para usar sempre o nome em inglês. As fórmulas mais utilizadas são:

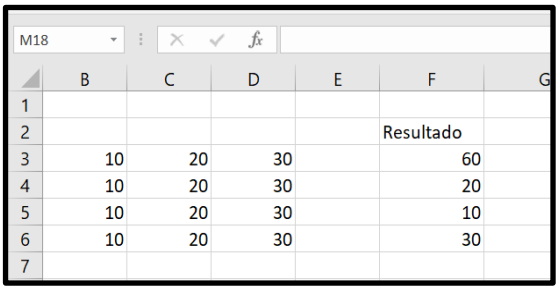
Sum - Soma os valores de um intervalo

Average - Faz a média dos valores de um intervalo

Min - Retorna o menor valor do intervalo

Max - Retorna o maior valor do intervalo

```
Sub teste()

'Somar valorer
Range("f3").Formula = "=sum(b3:d3)"

' fazer média
Range("f4").Formula = "=average(b4:d4)"

'valor minimo
Range("f5").Formula = "=min(b5:d5)"

'valor máxima
Range("f6").Formula = "=max(b6:d6)"

End Sub
```

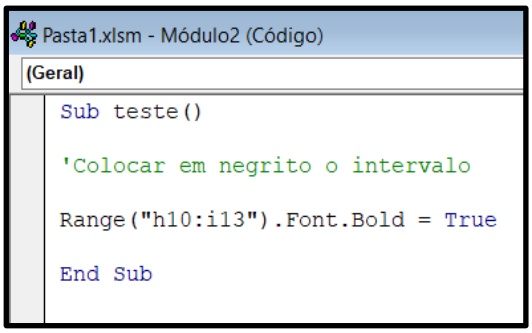


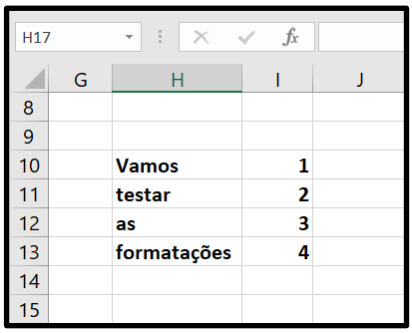
	B	C	D	E	F	G
1						
2					Resultado	
3	10	20	30		60	
4	10	20	30		20	
5	10	20	30		10	
6	10	20	30		30	
7						

e. Objeto Font

Esse Objeto refere-se a formatação de fonte do Excel. Ele é utilizado em conjunto com o objeto Range para modificar essas formatações na interface do Excel. Por exemplo, modificar a cor, tamanho, nome da fonte, etc.

- **Propriedade Bold**: Formata o intervalo de células indicado como negrito. É necessário atribuir o valor true (para ativar e negrito), ou false (para desativar o negrito).

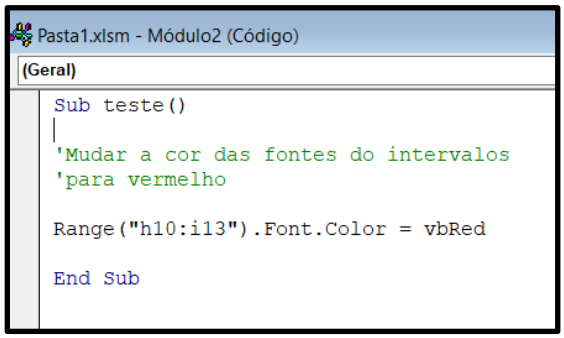


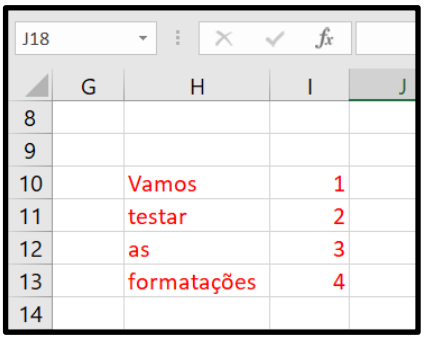


- **Propriedade Itálic:** Formata o intervalo de células indicado como itálico. É utilizada de forma análoga ao exemplo anterior.

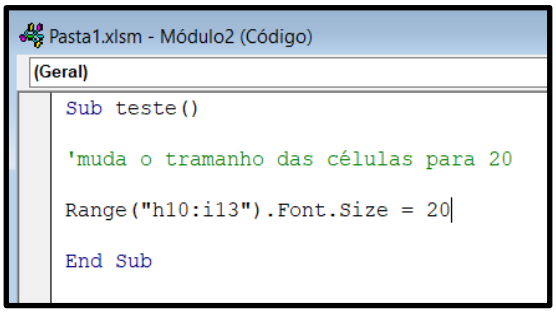
- **Propriedade Underline:** Essa propriedade torna sublinhado os valores no interior das células do intervalo indicado. É necessário atribuir o valor true (para ativar o underline), ou false (para desativar o underline). Também é utilizado de forma semelhante à propriedade Bold.

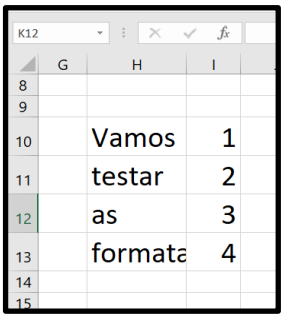
- **Propriedade Color**: Essa propriedade é utilizada para modificar a formatação da cor da fonte. É necessário atribuir o valor de algumas das constantes de cor do excel.



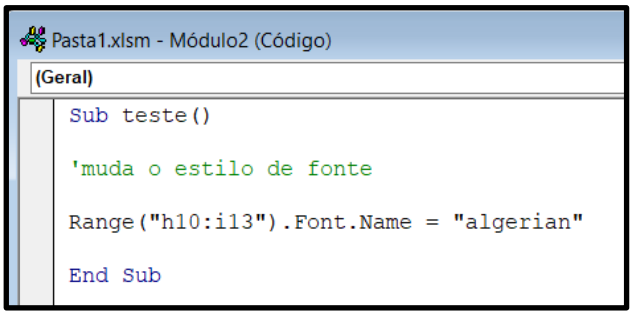


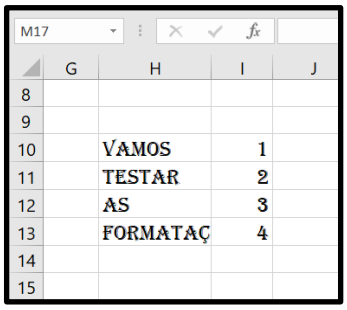
- **Propriedade Size**: Altera o tamanho da fonte. Para utilizar essa propriedade é necessário atribuir o valor referente ao tamanho da fonte desejado.





-





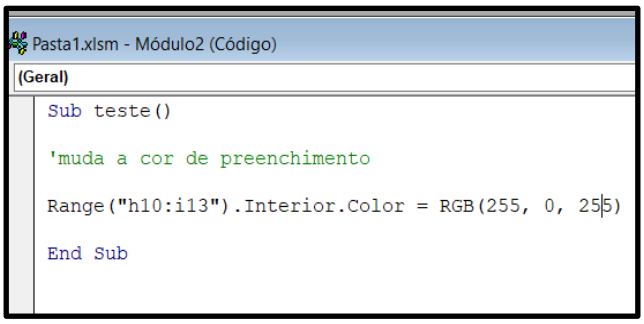
Propriedade name: Modifica os estilo da fonte. É necessário indicar o nome na fonte que será utilizada entre aspas.

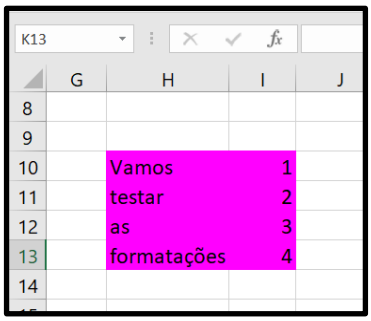
f. Objeto Interior

O objeto interior define a formatação do preenchimento das células. Ele é utilizado junto com o objeto range e altera a cor de fundo da célula.

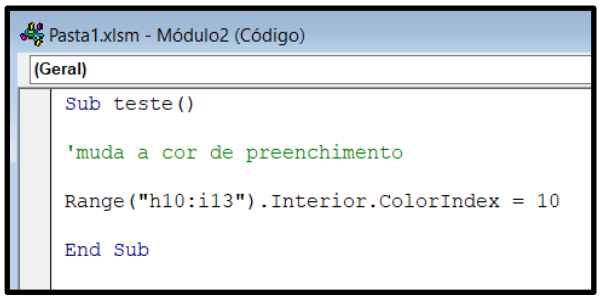
- **Propriedade Color:** Essa propriedade define a cor do preenchimento da célula, para utilizá-la é necessário atribuir uma constante de cor do Excel (vbred, vbgreen, vbblack, etc).

- Outra alternativa da propriedade Color é formatar a cor usando o RGB, um sistema de cores aditivas em que o Vermelho, o Verde e o Azul são combinados de modo a reproduzir diversas outras cores. Deve-se atribuir o valor RGB e , entre parênteses, definir a tonalidade das cores, vermelho, azul e verde respectivamente. A intensidade das cores varia de 0 a 255, sendo 0 a ausência da cor e 255 a maior intensidade da cor.





- **Propriedade ColorIndex**: Essa propriedade define a cor de preenchimento das células em valor inteiro de 1 a 56 cores. Cada valor inteiro corresponde a uma cloração



g. Objeto Borders

O objeto Borders é responsável por determinar as características das bordas de um

determinado Range. Abaixo são apresentadas as constantes que representam as bordas que podem ser alteradas no código VBA.

xlDiagonalDown – Diagonal de cima para baixo.

xlDiagonalUp – Diagonal de baixo para cima.

xlEdgeTop – Borda superior.

xlEdgeBottom – Borda inferior.

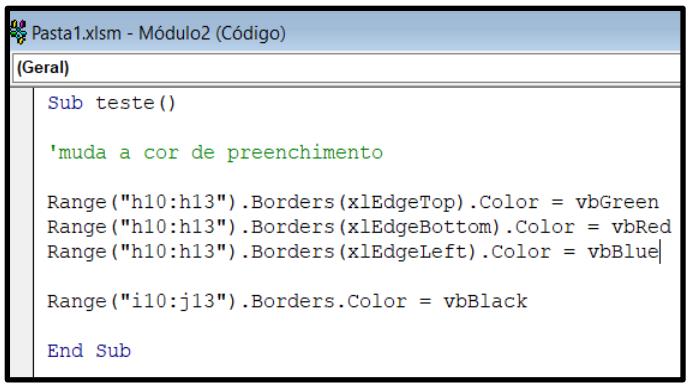
xlEdgeLeft – Borda esquerda.

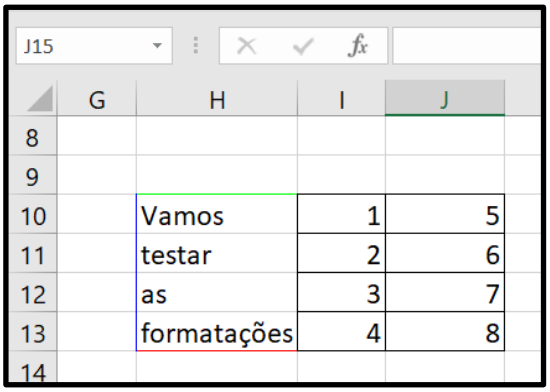
xlEdgeRight – Borda direita.

xlInsideHorizontal – Linha interna horizontal.

xlInsideVertical – Linha interna vertical.

Para utilizar essas constantes é necessário, depois do objeto borders, inserir a constante entre parênteses e, em seguida, a propriedade a ser alterada. Se não for indicada a borda que será alterada, o código aplica a propriedade a todas as bordas daquele intervalo de células.





6 Funções do VBA

As funções são propriedades que leem entradas (como números e textos), manipulam esses valores e retornam uma saída. As saídas podem assumir 3 tipos: números; texto ou booleanos (possuem valor de VERDADEIRO ou FALSO).

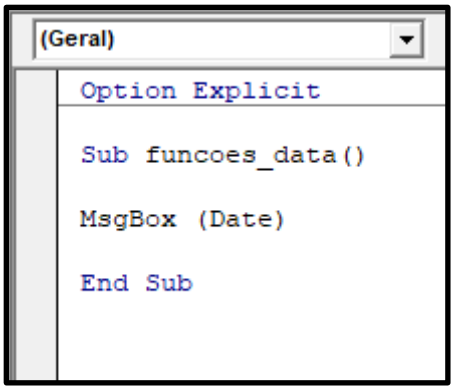
Existem 3 tipos de funções que podem ser usadas no VBA:

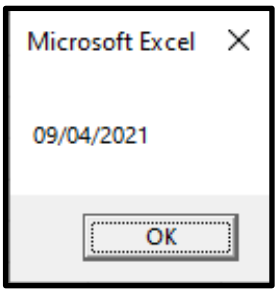
1) As funções do Excel, adaptadas a linguagem do VBE;
2) As funções próprias do VBA;
3) As funções criadas pelo usuário (UDF - User defined functions).

a. Funções de data e tempo

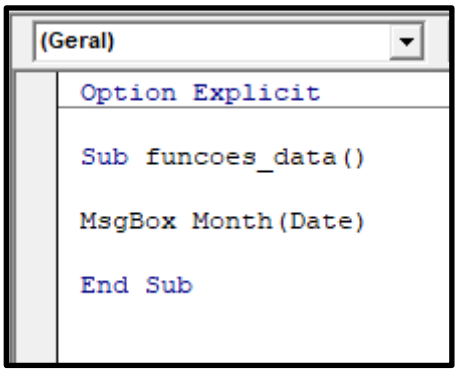
Essas funções são interessantes para aquelas planilhas que possuem prazos. Entre elas se destacam:

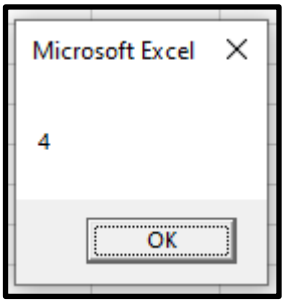
- **Date** : sem argumentos de entrada, retorna a data atual;
- **Time** : sem argumentos de entrada, retorna a hora;
- **Now** : sem argumentos de entrada, retorna a data + hora.



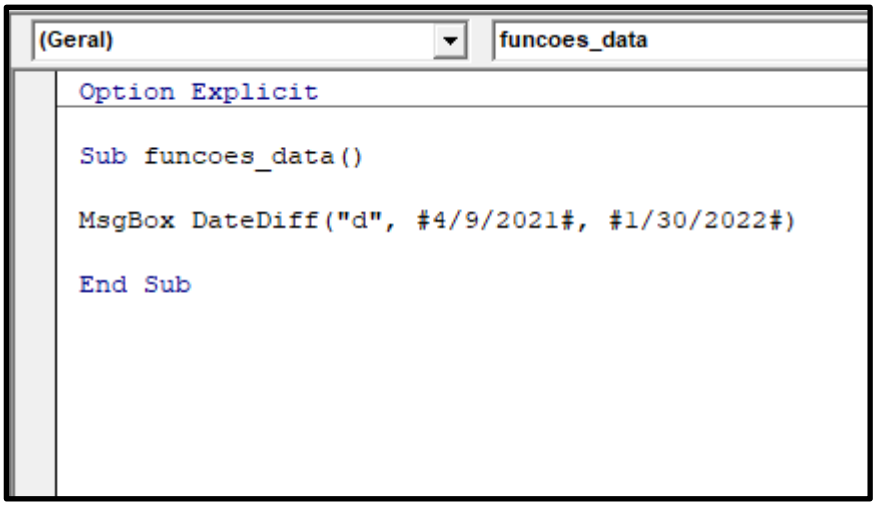


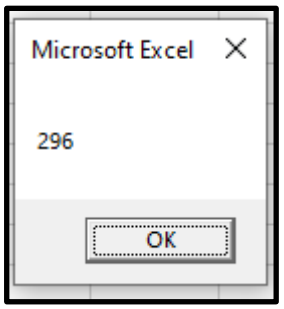
- **Month()** : com argumentos de entrada, retorna o número do mês;
- **Year()** : com argumentos de entrada, retorna o ano do valor de entrada;
- **Day()** : com argumentos de entrada, retorna o dia.





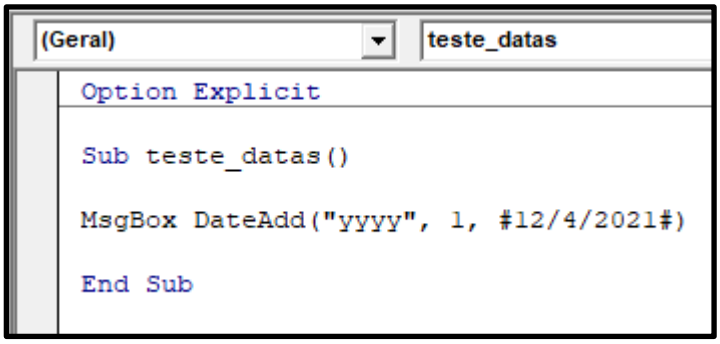
- **MonthName()** : Possui argumentos de entrada. Retorna o nome do mês do valor de entrada por extenso.
- **WeekdayName()** : Possui argumentos de entrada. Retorna o nome do dia da semana da data do valor de entrada.
- **DateDiff ()** : Essa função calcula a diferença entre duas datas. Deve ser fornecido 3 argumentos de entrada: o tipo de data que quer ser calculada (dias, mês ou ano); a data de início, e a data final, por exemplo:

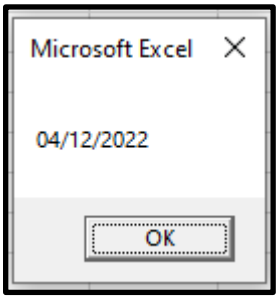




Neste caso, calculamos quantos dias (1° argumento “d”) separam as datas 09/04/2021 e 30/01/2022. O VBA nos retorna na caixa de mensagem 296 dias. Note que, como o VBE utiliza o idioma inglês, as datas são no formato mês/dia/ano, porém, como o Excel está configurado em PT-BR, as datas irão aparecer no Excel como dia/mês/ano.

- **DateAdd()** : É outra função que faz operações com as datas. Ela tem 3 argumentos de entrada: o tipo de data que será adicionada, quantidade, a data que sofrerá o aumento. Por exemplo:

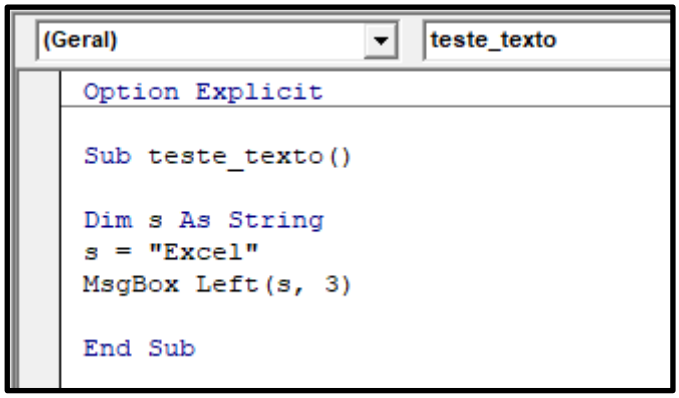


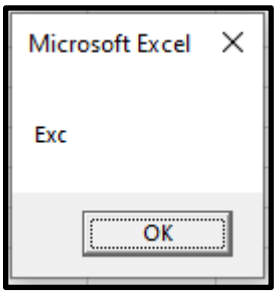


b. Funções de manipulação de texto

O tipo de variável mais utilizada nas funções de texto é a **string**. Ela serve para armazenar textos em variáveis. Os textos devem estar sempre acompanhados de aspas duplas (“ “).

- **Len()** : Essa função possui 1 argumento de entrada, e retorna o número de caracteres da expressão. Note que é possível inserir uma string diretamente na função, ou uma variável(tipo string) ou ainda um range. Se não houver nada escrito, o valor de retorno será 0.
- **Left/Right()** : Ela possui 2 argumentos de entrada, o primeiro é a string a ser analisada e o segundo, o número de caracteres a serem lidos. Ela retorna os caracteres, da direita para esquerda(right) ou esquerda para direita(left) de um número pré determinado.

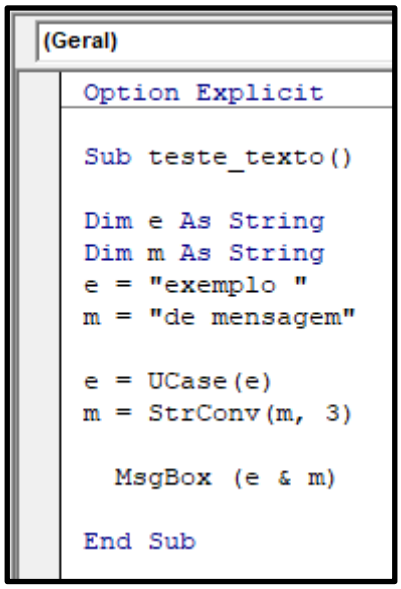




- **Ltrim/Rtrim()** : Essa função possui 1 argumento de entrada: a string a ser analisada. Ela irá suprimir todos os espaços em branco da direita ou esquerda da expressão.
- **Lcase/Ucase()** :Também possui apenas 1 argumento de entrada. Ela irá transformar as letras da string em maiúsculas(Ucase, o “U” é de upper case = ‘caixa alta’) ou em

minúsculas (Lcase, o "L" é de lower case = 'caixa baixa'). O retorno será a palavra escrita da forma desejada.

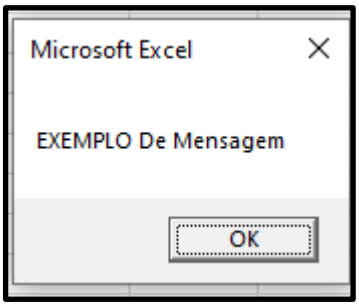
- **StrConv()** : Essa função engloba a Lcase/Ucase e adiciona uma nova forma de formatação de caixa de texto. Ela possui 2 argumentos de entrada, 1)string a ser analisada; 2)qual a conversão desejada. As conversões são classificadas da seguinte maneira:



1 → Ucase;

2 → Lcase;

3 → ProperCase (essa conversão transforma a primeira letra das palavras em maiúsculas e as demais em minúsculas).



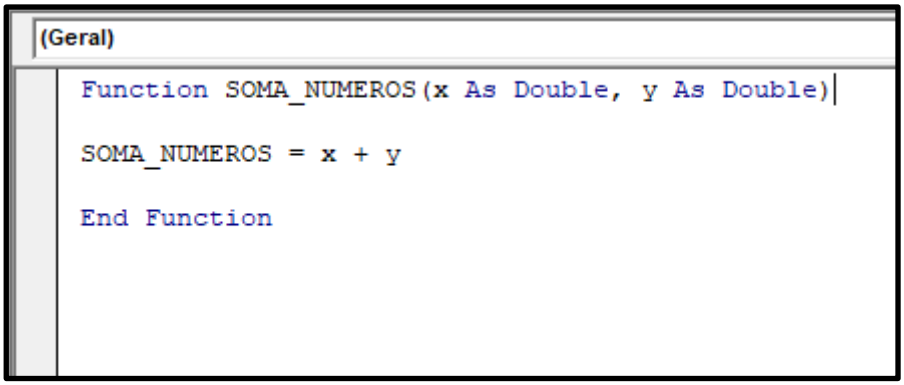
- **Replace()** : Possui 3 argumentos de entrada: expressão a se analisar, a palavra a ser encontrada e a palavra que irá substituir. Essa função substitui no texto a palavra que o usuário escolher por outra determinada.

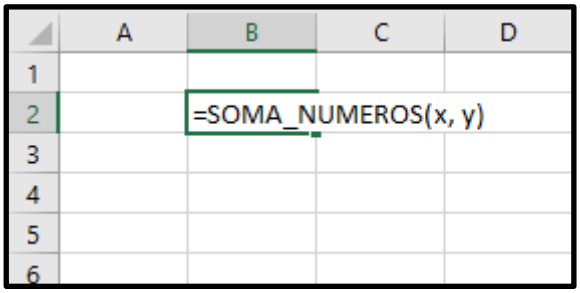
c. Funções do Excel no VBA

O excel possibilita que sejam utilizadas funções da planilha do excel (como soma ou médiase) dentro do VBE. Deve ser feita uma tradução da função em portguês para o inglês, já que o idioma do VBE é o inglês. Para podermos utilizar essas funções, devemos acessá-las com a expressão: **Application.WorksheetFuncion.função_em_ingles(argumentos da função)**. Neste link você pode conferir algumas das principais funções do worksheet em inglês.

d. Funções Criadas pelo Usuário

Também chamadas de User Defined Functions, isso permite que o usuário crie uma operação que pode ser usada tanto no VBE quanto na planilha do Excel. Isso deve ser feito dentro de um módulo. Usamos o procedimento Function Nome_da_Função(argumentos) ... End Function. No campo dos argumentos, você deve inserir as variáveis de que serão usadas na função e declarar de que tipo elas são. Por exemplo, se desejarmos fazer uma função que soma dois números reais, faríamos o seguinte:





Os números que forem colocados em x e y serão somados e o excel retornará o valor de x+y.

7 Fluxo de Execução

Nesta seção, iremos aprender a como controlar o fluxo de execução do código do VBA utilizando uma série de argumentos que geram repetições e mudanças de linha durante a leitura do código.

a. If ... Then... Else

A condicional 'if' se baseia, principalmente, nos valores de VERDADEIRO e FALSO. A lógica por trás desse argumento é: se algo for verdadeiro, então faça tal procedimento, senão faça outro procedimento. As palavras destacadas em vermelho são respectivamente o IF, THEN e ELSE do nosso código. Além disso, caso o escopo do IF tenha mais de uma linha, devemos adicionar o End If que irá indicar ao VBE o final daquela lógica.

No exemplo abaixo, queremos saber se x é múltiplo de y. Para isso, se feita a divisão x/y, o quociente deve ser um número inteiro. Portanto, dados x e y inteiros, se x/y é inteiro, então "x é múltiplo de y", senão "x não é múltiplo de y". Traduzindo isso para o VBE:

```
(Geral)
Option Explicit

Sub condicional()

'vamos começar declarando as
'variaveis x, y e resultado:
Dim x As Integer
Dim y As Integer
Dim resultado As Double

'agora, atribuimos os valores
'de x e y, e fazemos a operação:
x = 4
y = 2

resultado = x / y

    If resultado = Int(resultado) Then
    MsgBox "x é multiplo de y!"
    Else
    MsgBox "x NÃO é multiplo de y."

     End If

End Sub
```

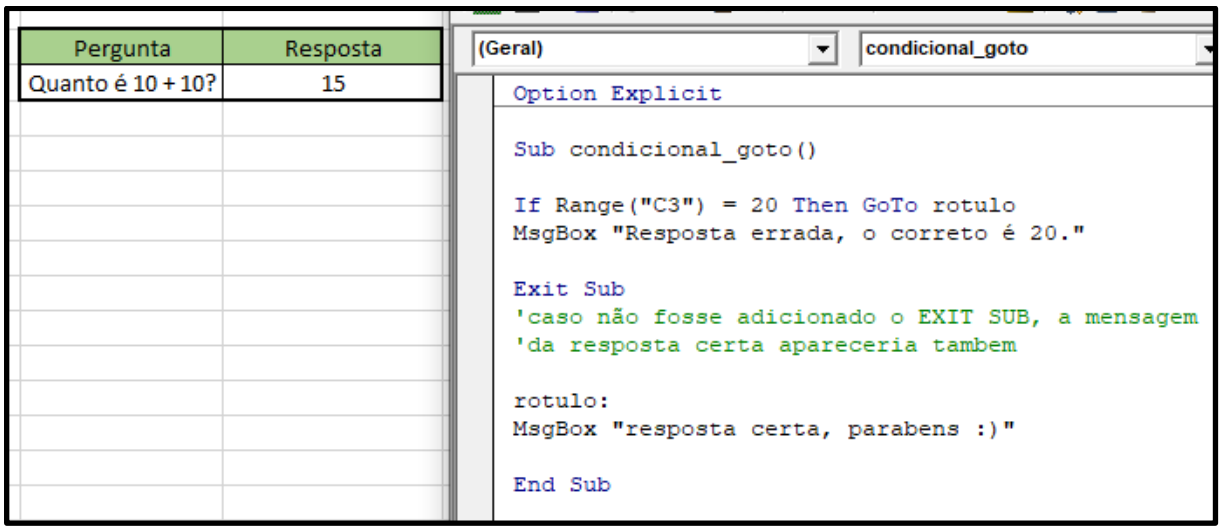
Nesse caso, escolhemos valores múltiplos de x e y, portanto, a caixa de mensagem que aparecerá será a primeira: "x é múltiplo de y!". Experimente fazer a depuração do código para verificar que caso a condição do IF não fosse satisfeita, a linha da primeira MsgBox seria "pulada" e o depurador cairia direto na segunda caixa.

OBS: A função Int(resultado) retorna a parte inteira de determinado valor. Por exemplo: Int(12,625) retornará 12. Nesse código ela foi usada para comparar o valor "cheio" do resultado com a parte inteira dele. Se esses retornos forem iguais, então a condição é satisfeita.

b. If… Then… GoTo

Essa função é bem similar à anterior. A diferença é que ao invés de realizar uma ação, ela "pula" para determinado ponto do código. É justamente o que GoTo significa em português : 'Vá para'. Por isso, devemos indicar ao VBE onde queremos que o código vá. Isso é feito por meio de **rótulos**. Os rótulos são como âncoras no código, elas marcam pontos estratégicos da sua lógica. Eles devem ser escritos como uma palavra seguida de dois pontos (:).

Porém, em alguns casos, mesmo usando o GoTo, as ações subsequentes (que não deveriam) são executadas. Para evitar esse problema e encerrar o código em determinado ponto, utilizamos o Exit Sub. Esse argumento será como uma saída em uma grande rodovia, é uma rota alternativa para chegar ao destino final.



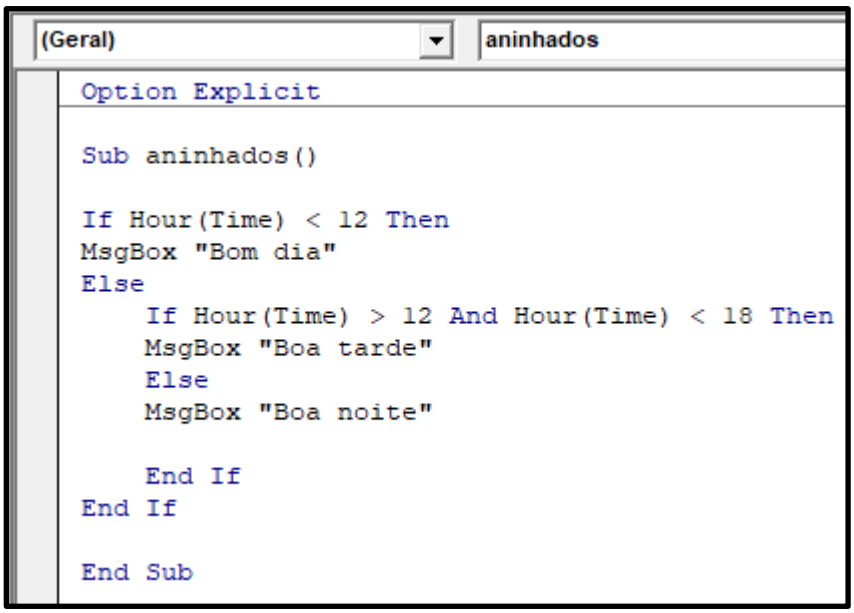
Múltiplos critérios:

Note que, em alguns casos, devemos usar mais de um operador lógico para que seja possível escrever aquela condição. Por exemplo, números reais maiores que 3 e menores que 5. Nesse caso, o número em questão é 4, pois satisfaz as duas condições. Agora, se o exemplo fossem números inteiros menores que 3 ou maiores que 5, estamos selecionando todo conjunto dos inteiros a exclusão do 4. Isso acontece pois não há obrigatoriedade de se cumprir as duas condições, se apenas uma for satisfeita, então a afirmação é verdadeira.

No VBA temos a mesma lógica, porém as palavras mudam: para 'e', substituir por AND e para 'ou', substituir para OR. Portanto se queremos o número 4 baseado na primeira condição do parágrafo anterior, escrevemos: x > 3 AND x<5.

Ifs aninhados:

Se assemelha a aninhar "se" no excel. Basicamente, quando um IF for aberto, no lugar de uma ação no "valor caso falso", deve ser escrito um novo IF com sua própria condição. Isso pode ser feito indefinidamente, desde que você se lembre de "fechar" todos os IFs com o respectivo número de End IF.



c. ElseIf

O ElseIf é usado principalmente para reduzir as linhas de código. Ele é usado no lugar de um End If para aninhar os Ifs. A vantagem de utilizar o Else If é que é como se fosse feito o aninhamento de 2 Ifs, mas na verdade utilizamos apenas 1 End If, ou seja, conseguimos economizar linhas de código a fim de acelerar nossos procedimentos.

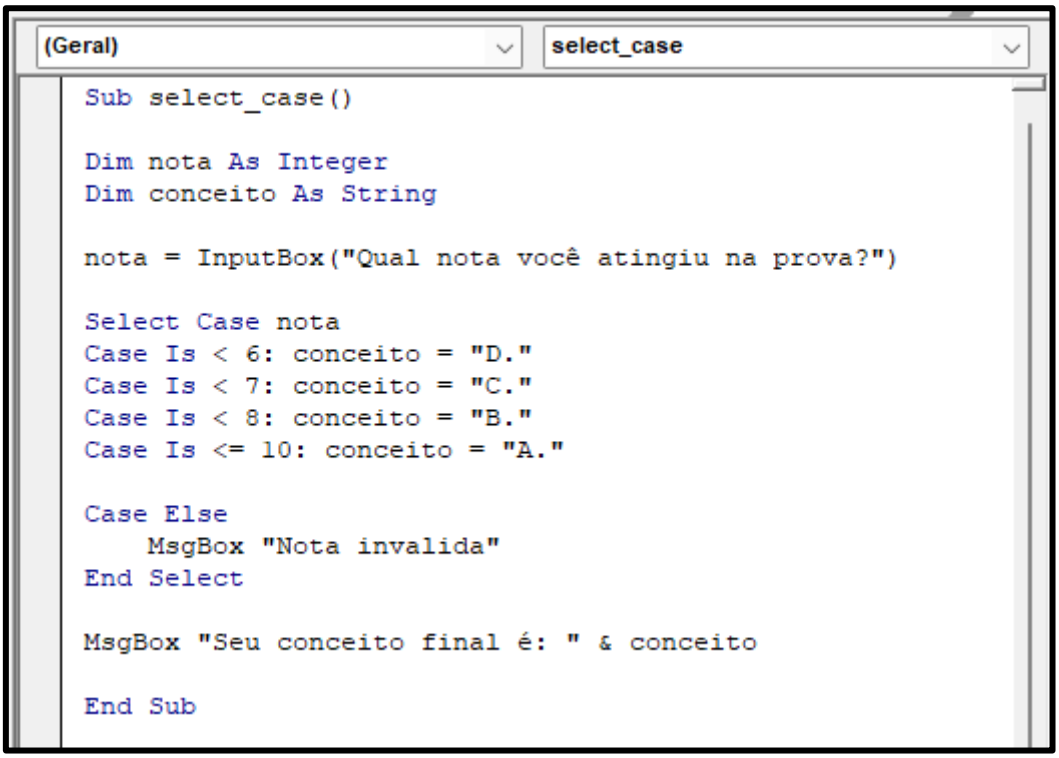
(Geral)

```
Option Explicit

Sub else_if()

If Hour(Time) < 12 Then
MsgBox "Bom dia"
ElseIf Hour(Time) >= 12 And Hour(Time) < 18 Then
'note que, diferentemente do Else, o ElseIf exige uma condição
MsgBox "Boa tarde"
Else
MsgBox "Boa noite"

End If

End Sub
```

d. Select Case

O Select Case é outra ótima alternativa para substituir os comandos if, pois possibilita linhas de códigos mais claras e resumidas.

A lógica da função baseia-se na leitura e teste das condições impostas, podendo analisar, de uma só vez, inúmeras situações.



e. Loop For… Next

Para situações onde o usuário necessita criar procedimentos que devem se repetir conforme condições impostas, o loop for é uma excelente alternativa.

Dentre os diferentes tipos de loop, encontra-se o loop for next, que realiza a mesma atividade durante o número de vezes que for programado para fazer. Esse processo é realizado atribuindo valores a variáveis, e, consequentemente, encerrado assim que o valor da variável estiver fora do intervalo estipulado.

Exemplo: preencher células de determinada coluna de 1 a 10.

```
Option Explicit

Sub teste_fornext()

Dim x As Integer

For x = 1 To 10
    ActiveCell.Offset(x, 0) = x
Next

End Sub
```

f. Loop For Each... In... Next

Nesse comando, o código analisa os ítens de uma coleção e realiza ações conforme programado. Planilhas abertas, abas e células de um determinado intervalo são exemplos de coleções.

Exemplo: alguma linha específica (2) de todas as abas.

```
(Geral)
Option Explicit

Sub remover_colunal()

Dim As Worksheet

For Each x In ActiveWorkbook.Worksheets
    x.Rows(2).Delete
Next

End Sub
```

g. Loop do While

Neste comando, determinado procedimento será executado enquanto uma dada condição for verdadeira e encerra-se assim que for falso.

Exemplo: duplicar os valores das células na coluna anterior.

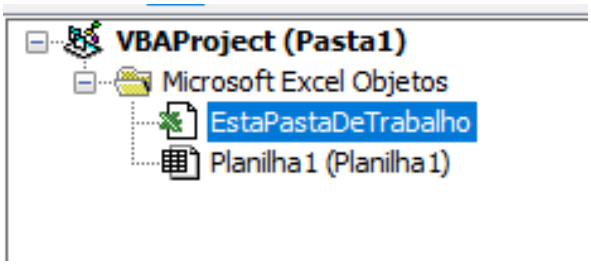
```
(Geral)
Option Explicit

Sub teste_doWhile()

Dim x As Double

x = ActiveCell.Offset(1, -1).Value

Do While x <> Empty

    ActiveCell.Offset(1, 0).Value = x * 2
    ActiveCell.Offset(1, 0).Activate
    x = ActiveCell.Offset(1, -1).Value

Loop

End Sub
```

8. Eventos

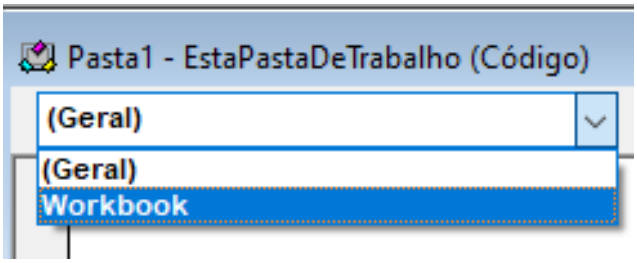
Eventos no excel são ações que o programa controla e monitora, como janelas de mensagens ao fechar planilhas. Através do VBA é possível criar novas ações ou alterar as já existentes de acordo com a necessidade do usuário.

a. Criação de eventos

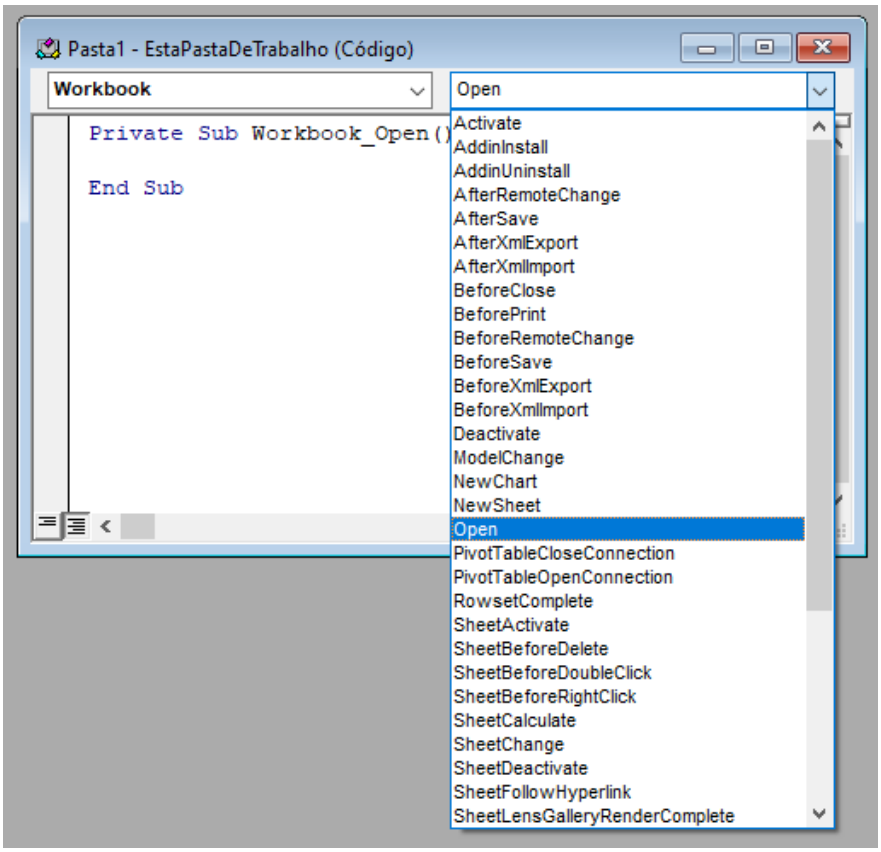
Para a criação de um evento, primeiro deve-se abrir “EstaPastaDeTrabalho”, dentro de Microsoft Excel Objetos.



Após abrir a janela, na primeira aba superior altera-se de “(Geral)” para “Workbook”.



Então, logo na aba a direita, abrirá diversas possibilidades de criação de eventos, os quais alguns serão abordados nos seguintes tópicos.

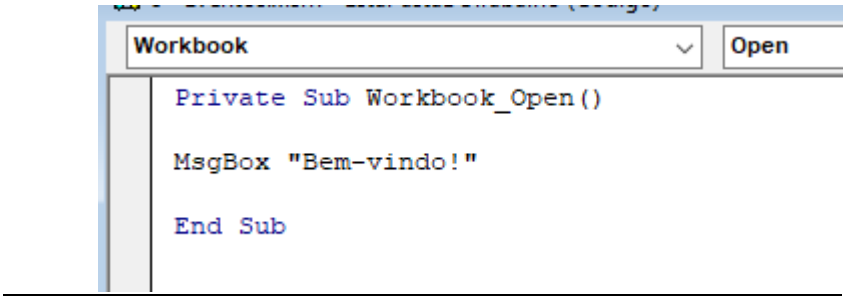


b. Open

Estes eventos são executados assim que a planilha em questão for aberta. Através dos eventos open é possível realizar ações como solicitação de senha, exibição de janelas de mensagem, etc.

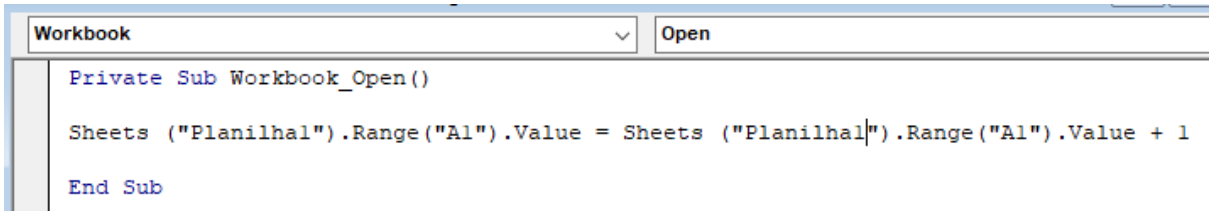
Janela de mensagem

Exemplo: Criar uma janela de mensagem que será exibida assim que a planilha for aberta.



Contadores

Exemplo: Será contada toda a vez que a planilha for aberta na célula A1.



c. BeforeClose

Assim como os eventos open, que são executados assim que a planilha é aberta, também há os eventos que podem ser executados instantes antes da planilha ser fechada. Dentre as diversas possibilidades, é possível exibir ou deletar janelas de mensagem, modificar valores de células, etc.

Janela de mensagem

Exemplo: Assim que o usuário der o comando para fechar a planilha aparecerá um aviso para ele confirmar se deseja ou não sair. Caso confirme, a planilha será fechada, caso contrário continuará aberta.

Workbook | BeforeClose

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)

Dim resposta As Integer

resposta = MsgBox("Tem certeza que deseja sair?", vbYesNo, "Aviso")

    If resposta = vbNo Then
    Cancel = True
    End If

End Sub
```

d. Call

O uso do comando Call é útil quando se têm diversas linhas de códigos de eventos, possibilitando o uso e uma leitura mais simples. Este comando funciona chamando eventos em um determinado código.

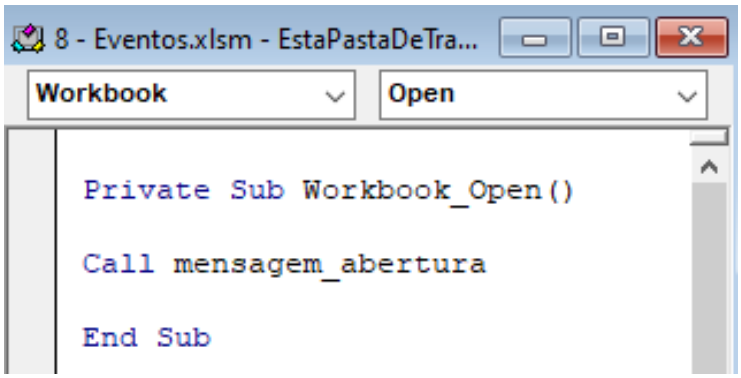
Primeiramente, o código deve ser armazenado dentro de um Módulo.

8 - Eventos.xlsm - Módulo1 (Código)

(Geral) | mensagem_abertura

```
Sub mensagem_abertura()

Dim msg As String

msg = "Bem-vindo!"

MsgBox msg

End Sub
```

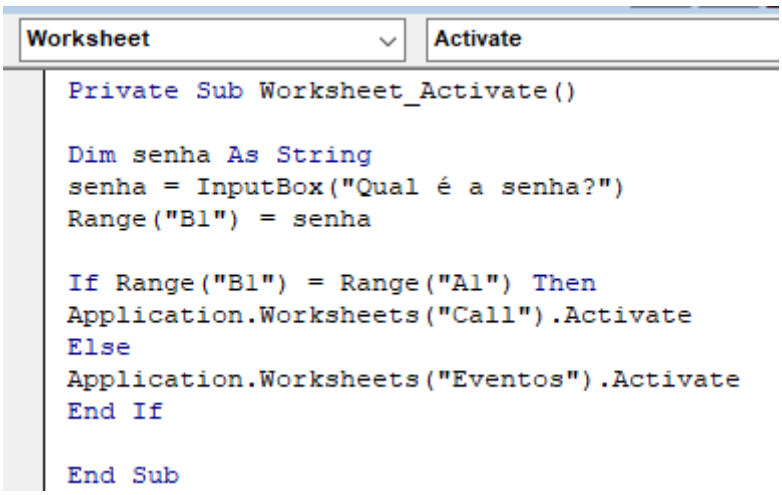
Após isso, volta-se na Workbook e utiliza-se o comando Call para inserir o evento



e. Criação de senha

Uma das melhores formas de proteger o conteúdo da planilha é gerando uma senha, a qual é solicitada assim que o usuário a abrir. Possibilitando o acesso controlado apenas de quem possuir a senha.

Exemplo: colocar senha em uma aba.

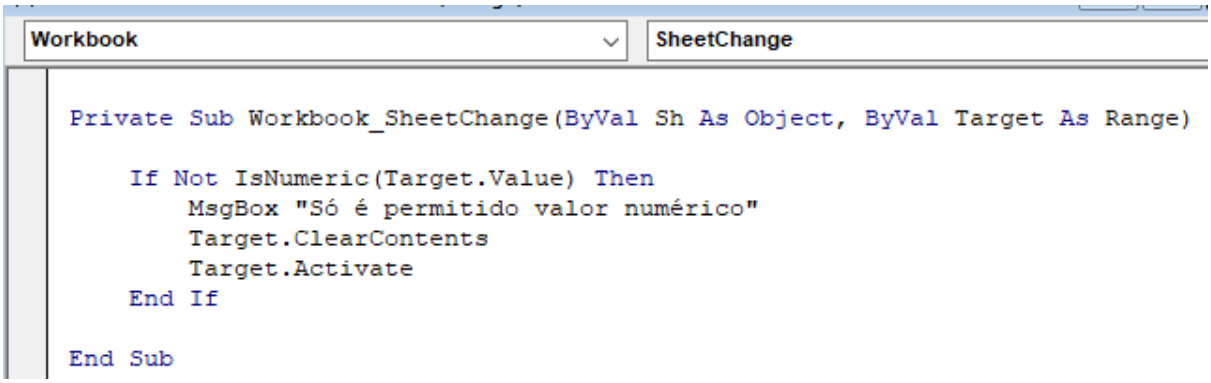


Caso os caracteres digitados sejam iguais aos armazenados na célula A1, a aba será liberada para uso.

f. Limitando o uso de uma planilha

Também é possível limitar as ferramentas de uma planilha quando é desejado que o usuário exerça funções específicas.

Exemplo: determinada aba só aceita valores numéricos.



9. Depuração e manipulação de erros

Quando o VBA não entende o que queremos dizer em certa parte do programa ele envia uma mensagem de erro.

a. Tipos de erros no VBA

- ❏ Erros de compilação:
  Acontecem quando ocorre erro na escrita do código.

  Procedimentos sem nome ou com nome inválido (digitado errado); declarações escritas de maneira errada; comandos sem expressão correspondente (esquecer partes necessárias da função).

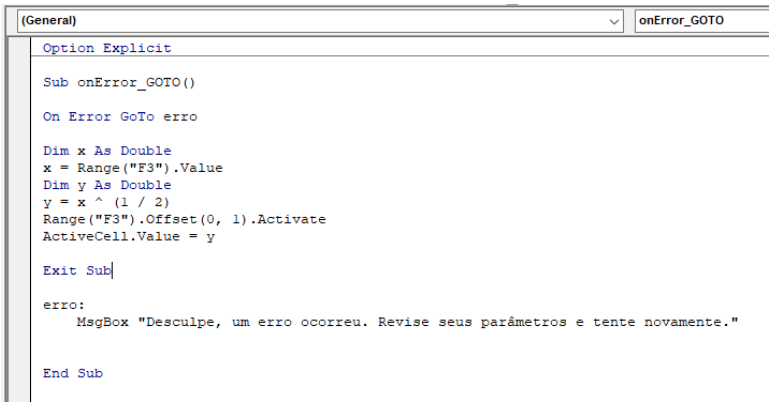
- ❏ Erros de tempo de execução:
  Acontecem quando o programa tenta fazer uma operação que é impossível de ser executada por faltar algum parâmetro essencial.

- ❏ Erros de lógica:
  O programa entende o código de maneira diferente da desejada, dando uma resposta inesperada. Ocorre quando você não incorpora ao código algo que precisava para a execução ocorrer como desejada.

Quando ocorre um erro no VBA, aparece um aviso em formato de caixa de mensagem. No entanto, o texto nela contido não explica exatamente qual foi o erro. Assim, de forma a explicitar o que deve ser reparado, pode-se mudar as mensagens originais para algumas mais compreensíveis.

Pode-se fazê-lo utilizando:

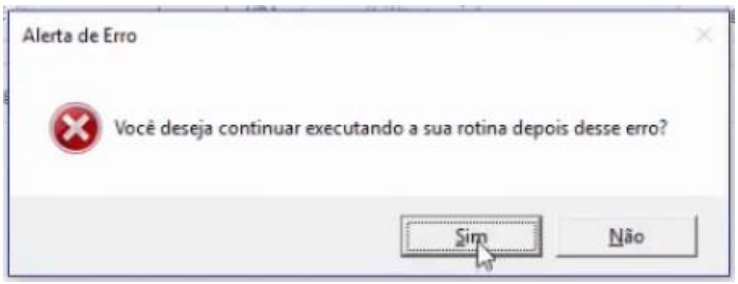
b. On Error...Goto...Erro - (em caso de erro… vá para… nome escolhido): Essa é uma função que percebe um erro e envia o programa para uma determinada parte do código, previamente determinada (que é o “nome escolhido”) onde podemos colocar uma MsgBox para enviar um novo texto em caso de erro. Entretanto, não ocorrerá erro sempre, então precisa-se inserir um “Exit Sub” para que o programa consiga identificar quando não há erros, como visto na imagem a seguir:



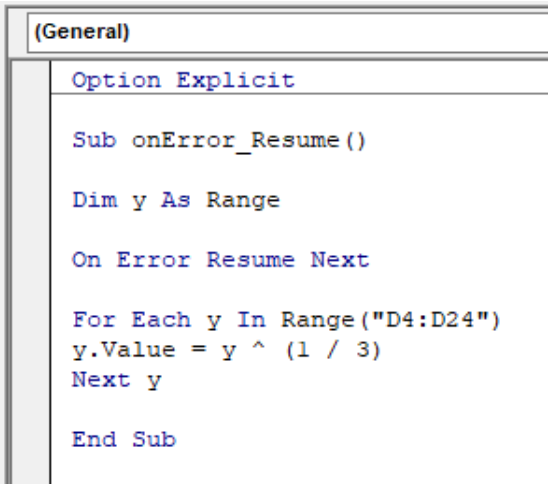
c. Porém, ao invés de apenas informar o usuário sobre o erro (MsgBox), podemos questioná-lo se deseja continuar a rotina, ignorando o erro, ou não. Para isso, usa-se "On Error...Goto...Erro", inserindo uma MsgBox para perguntar se o usuário deseja continuar com as opções de resposta "Yes" ou "No". Então, usa-se a função "If" combinada com a "Resume" para continuar o programa caso a resposta seja "yes".

```
Option Explicit

Sub onError_escolha()

On Error GoTo erro

Dim x As Range

    For Each x In Range("G5:G11")
    x.Value = x ^ (1 / 2)
restart:

Next x

Exit Sub

erro:
    Dim z As Integer
    z = MsgBox("Você deseja continuar executando a sua rotina depois desse erro?", vbYesNo + vbCritical, "Alerta de Erro")
    If z = vbYes Then Resume restart

End Sub
```

vbCritical: símbolo, visível na imagem abaixo, que pode ser adicionado a MsgBox para aparecer quando der o erro.



d. de erro que deveriam ser enviadas e possibilita que sua rotina seja executada até o final. Caso essa função não fosse implementada, o programa pararia no primeiro erro, mesmo que houvesse mais a ser feito.

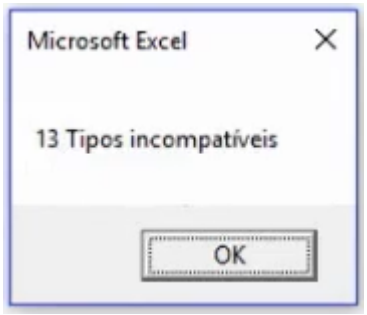


e. Err. - Essa é uma função para identificar o valor que o VBA associa a

cada tipo de erro e a sua descrição, a fim de enviar uma mensagem específica e inteligente quando ocorrer um erro. Err.Number é utilizado para mostrar o valor associado ao tipo de erro e Err.Description para a descrição do tipo de erro.

Por exemplo:

```
erro:
    MsgBox Err.Number & " " & Err.Description
```



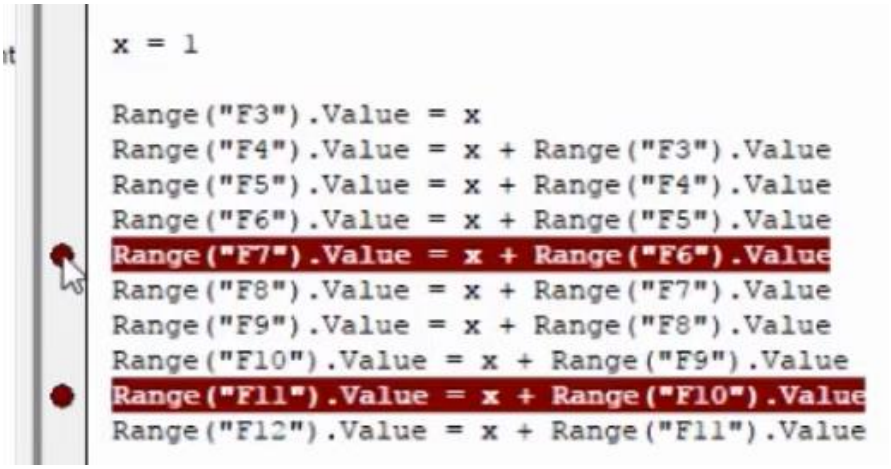
Além disso, para a descrição específica, utiliza-se o comando de Condicionais juntamente com o Err. e com a MsgBox, assim:

```
erro:
    Dim msg As String
    If Err.Number = 13 Then
    msg = "Você não pode extrair a raiz quadrada de um texto"
    End If
    If Err.Number = 5 Then
    msg = "Você não pode extrair a raiz quadrada de um valor negativo"
    End If
    MsgBox msg
```

Sendo 13 e 5 valores definidos pelo VBA.

f. Pontos de Interrupção - sinal que informa o depurador para suspender a execução de um código em um determinado ponto.

Para inserir esse ponto basta clicar, para qualquer linha do seu código, na barra cinza da lateral esquerda. E para retirar a marcação, clique novamente no mesmo local (que agora estará com uma “bolinha” bordô de marcação).



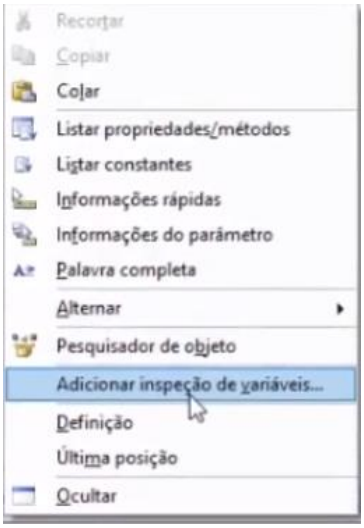
A marcação mostrada também pode ser feita, com o cursor do mouse na linha desejada, clicando na aba “depurar” e selecionando a opção “ativar/desativar pontos de interrupção”. Ou até mesmo clicando com o botão direito do mouse na linha desejada, selecionando “Alternar” e, após, “Pontos de interrupção”.

Obs.: A última linha do código a ser rodada será a imediatamente anterior àquela que está com o ponto de interrupção.

Esse ponto pode ser útil quando desejamos rodar apenas uma parte do código, para testar sua funcionalidade, interrompendo o teste em um certo momento. Caso seja decidido continuar a rodar o código de onde ele foi interrompido, pode-se clicar em “F5” no teclado ou apertar o botão de “play” no VBE.

g. Inspeção de variáveis: ferramenta que possibilita a verificação de uma condição no decorrer de uma rotina.

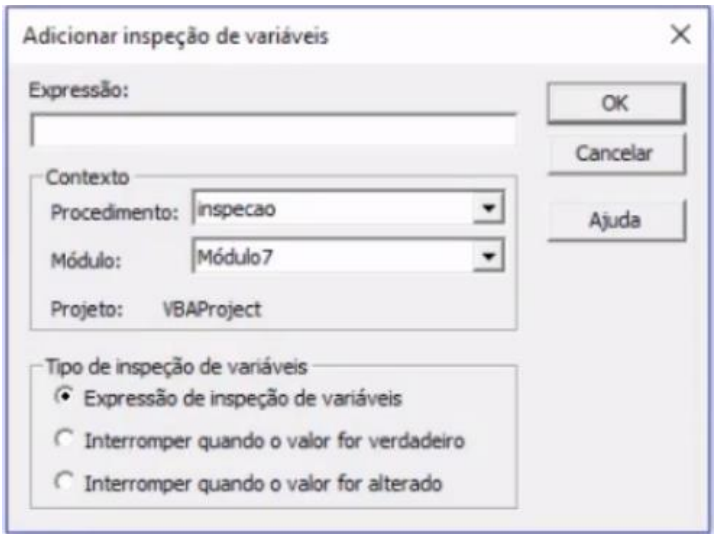
Para inserir a inspeção de variáveis você deve, ao final do código que deseja inspecionar, clicar com o botão direito do mouse e selecionar a opção “adicionar inspeção de variáveis” conforme imagem:



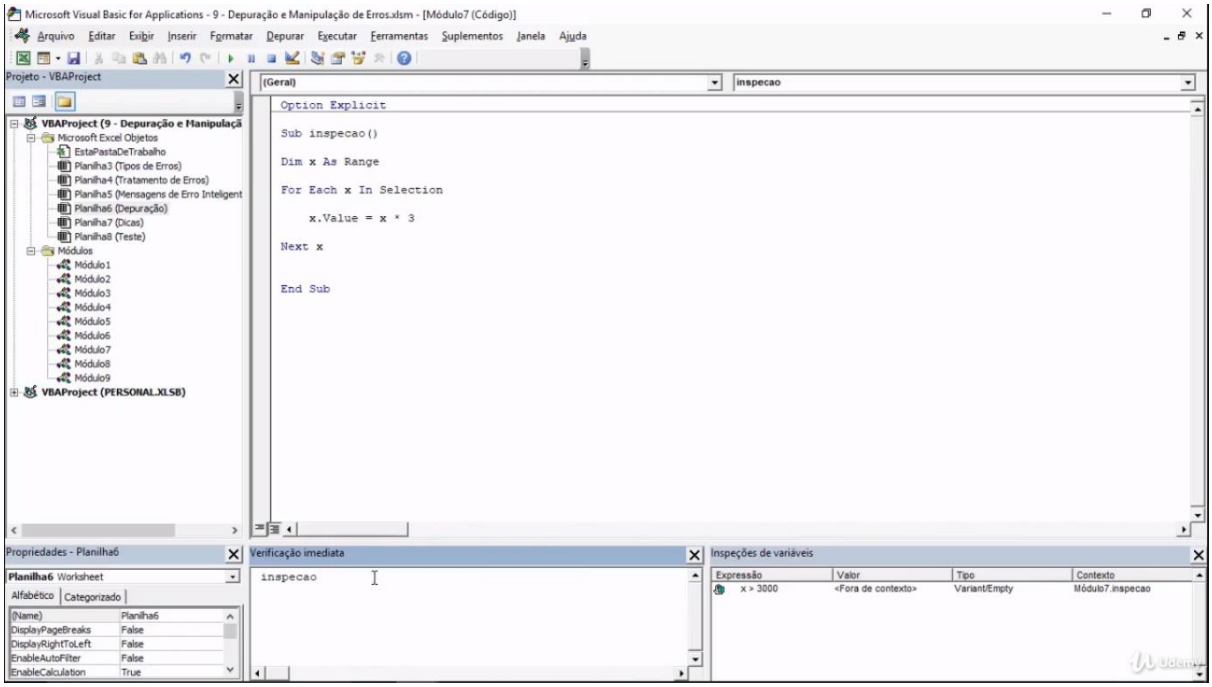
Nele você pode inserir a expressão limitante, o procedimento, o módulo onde ele ocorrerá e qual o tipo de inspeção de variáveis utilizar. No qual a inspeção se subdivide em:

- ❏ Expressão de inspeção de variáveis
- ❏ Interromper quando o valor for verdadeiro
- ❏ Interromper quando o valor for alterado

Conforme imagem:



Ao clicar em “ok” o layout do VBA mudará adicionando a inspeção de variáveis na parte inferior direita, conforme imagem.



Ao rodar o código, quando atingido os valores limitantes, inseridos anteriormente, o código para. Caso queira continuar rodando o código basta apertar o botão “play” novamente.

h. Dicas para acelerar a execução no VBA.

Desabilitação do ScreenUpdating:

O ScreenUpdating preenche células uma a uma, mostrando na tela o percurso do preenchimento para o usuário, o que pode demorar um pouco. Desativando essa função antes da sequência de tarefas disposta no Módulo e reativando logo depois, faz com que os dados apareçam todos juntos, em uma única vez, depois que a rotina tenha sido realizada, agilizando o processo.

```
Option Explicit

Sub screen_update()

Dim x As Long, y As Long, z As Long

Application.ScreenUpdating = False

z = 0
    For x = 1 To 100
        For y = 1 To 28
            z = z + 1
            Cells(x, y).Select
            Cells(x, y).Value = z
        Next y
    Next x

Application.ScreenUpdating = True

End Sub
```

Mudança na Calculation:

A função Calculation pode ser manual (xlCalculationManual e o cálculo é feito quando solicitado pelo usuário) ou automática (xlCalculationAutomatic e o Excel controla o recálculo). Quando faz-se ela de uma só vez, a conta é acelerada.

(Geral)

```
Option Explicit

Sub screen_update()

Dim x As Long, y As Long, z As Long

Application.ScreenUpdating = False
Application.Calculation = xlCalculationManual

z = 0
    For x = 1 To 100
        For y = 1 To 28
            z = z + 1
            Cells(x, y).Select
            Cells(x, y).Value = z
        Next y
    Next x

Application.ScreenUpdating = True
Application.Calculation = xlCalculationAutomatic

End Sub
```

Desabilitando os alertas:

Os alertas pedem permissões, confirmações e dão avisos, sendo eles “conversas” do Excel com o usuário. Ao desativar essas trocas, tem-se mais agilidade na programação. Por exemplo, para apagar uma aba sem o questionamento do sistema:

```
Sub alertas()

Application.DisplayAlerts = False

Worksheets("Teste").Delete

Application.DisplayAlerts = True
```

E no fim, se desejado, coloca-se o DisplayAlerts novamente como verdadeiro, para que ele mostre os alertas nas próximas ações.

i. Dicas para termos grandes ou repetidos no VBA.

Declaração SET: Uma linha de código com muitos caracteres é considerada um termo grande, e às vezes precisa-se utilizá-la várias vezes. Nesse momento, pode-se simplificar o código com a declaração “SET”. Para isso, inicia-se declarando uma variável como intervalo (range), então, com o uso do “SET”, a variável anterior é definida como o “termo grande” a ser usado diversas vezes. Veja no exemplo:

SET variável = termo grande que você quer evitar

```
Option Explicit

Sub tamanho()

Workbooks("9 - Depuração e Manipuação de Erros.xlsm").Worksheets("Dicas").Range("C3:C9").Interior.Color = vbBlue

Dim intervalo As Range

Set intervalo = Workbooks("9 - Depuração e Manipuação de Erros.xlsm").Worksheets("Dicas").Range("C3:C9")

End Sub
```

Com isso, pode-se utilizar apenas a variável declarada para referir-se a uma grande quantidade de caracteres, como vê-se abaixo:

```
Option Explicit

Sub tamanho()

Workbooks("9 - Depuração e Manipuação de Erros.xlsm").Worksheets("Dicas").Range("C3:C9").Interior.Color = vbBlue

Dim intervalo As Range

Set intervalo = Workbooks("9 - Depuração e Manipuação de Erros.xlsm").Worksheets("Dicas").Range("C3:C9")

intervalo.Interior.Color = vbBlue


End Sub
```

Utilização do WITH: Para termos repetidos não terem que ser reescritos várias vezes. Esse comando é ainda mais interessante para códigos maiores e com mais repetições, ele facilita a escrita e o entendimento do programa. Ao invés de escrever todas as funções completas, coloca-se as mudanças dentro de um WITH. Por exemplo:

```
Selection.Font.Color = -16776961
Selection.Font.TintAndShade = 0
Selection.Font.Bold = True
```

```
With Selection.Font
    Color = -16776961
    TintAndShade = 0
    Bold = True
End With
```

sem WITH com WITH

10. Interação com Usuário Utilizando Formulários

a. MSGBOX

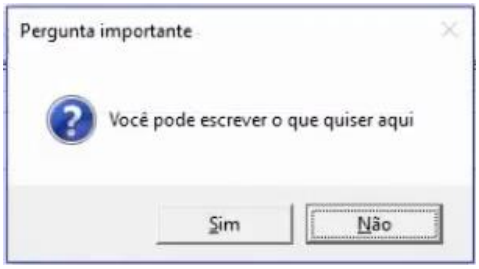
É a caixa de mensagem. Ela é composta de título, texto e botão. Ela é utilizada basicamente como:

MsgBox "mensagem", botões, "título"

Na mensagem e no título, para textos, é obrigatório o uso de aspas para que o VBE identifique o que foi escrito. Para os botões há uma lista de palavras propostas pelo programa, como Sim e Não (vbYesNo). Além disso, a fim de adicionar características a eles, coloca-se o sinal de mais '+' e, em seguida, indica-se o desejado. Para mostrar a imagem do ponto de interrogação: +vbQuestion; e para destacar um botão: +vbDefaultButton1, +vbDefaultButton2 ou +vbDefaultButton3, para, respectivamente, os botões 1, 2 ou 3.

```
Sub msgOK()

MsgBox "Você pode escrever o que quiser aqui", vbYesNo + vbQuestion + vbDefaultButton2, "Pergunta importante"

End Sub
```

No VBE.



Para fazer com que a caixa de mensagem retorne algum conteúdo declara-se uma variável x e a iguala à MsgBox, isso faz com que ao x seja atribuído o valor de sim ou de não. Então, para que o programa saiba o que deve ser feito em cada caso, usa-se o Select Case, como na imagem abaixo:

```
Option Explicit

Sub msgOK()


Dim x As Integer

x = MsgBox("Tem certeza disso?", vbYesNo + vbQuestion + vbDefaultButton2, "Pergunta importante")

Select Case x
    Case vbYes
        Range("D7").Value = "Você disse sim"
    Case vbNo
        Range("D7").Value = "Você disse não"
End Select

End Sub
```

Caso a resposta na MsgBox seja “No”, na célula D7 do Excel aparecerá escrito:

b. INPUTBOX

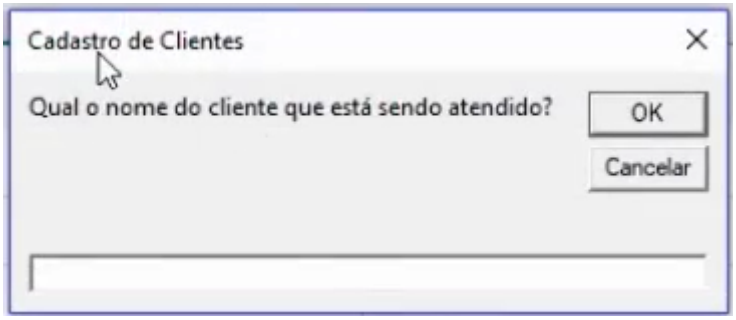
A INPUTBOX é a caixa de entrada, ela aceita respostas abertas, como valores ou mensagens textuais. Para criar uma INPUTBOX, no procedimento Sub, declara-se uma variável x como string e a faz assumir o valor da INPUTBOX:

X = INPUTBOX (“texto”, “título”)

No VBE:

```
Option Explicit

Sub entrada_nome()

Dim x As String

x = InputBox("Qual o nome do cliente que está sendo atendido?", "Cadastro de Clientes")

Range("D5").Value = x

End Sub
```

No Excel:



Ademais, pode-se adicionar uma condicional (IF) que fale com o usuário caso ele não preencha a INPUTBOX.

```
Option Explicit

Sub entrada_nome()

Dim x As String

x = InputBox("Qual o nome do cliente que está sendo atendido?", "Cadastro de Clientes")

If x = "" Then
MsgBox "Você esqueceu de preencher o nome do cliente"
Else
Range("D5").Value = x
End If

End Sub
```

A caixa de entrada é um artifício que proporciona agilidade em preenchimentos de tabelas quando utilizada adequadamente. Por exemplo, para o cadastro de nome e idade, utiliza-se duas INPUTBOX juntamente com um INSERT (inserir) nas células desejadas. Observe:

```
Sub cadastro_clientes()

Dim x As String
Dim y As String

x = InputBox("Qual o nome do cliente?", "Cadastro")
y = InputBox("Qual a idade do cliente?", "Cadastro")

Range("G4").Value = x
Range("H4").Value = y

Range("G4:H4").Insert

End Sub
```

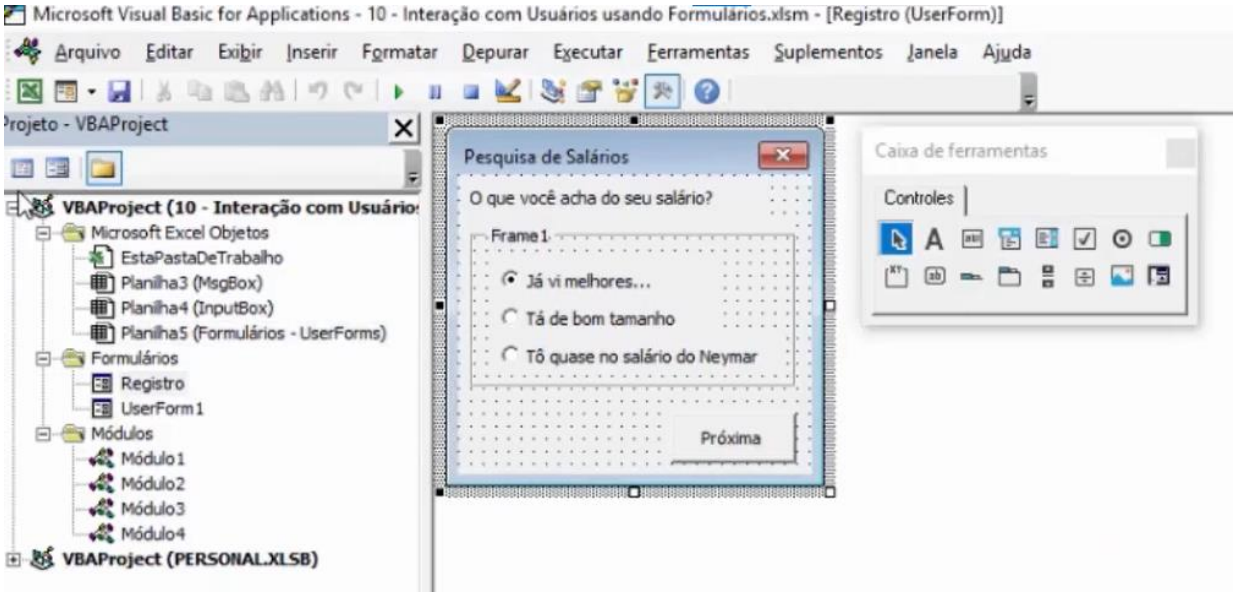
E, após alguns preenchimentos, tem-se:

Cadastro	Nome	Idade
	Daniel	36
	Paula	27
	Rafael	31

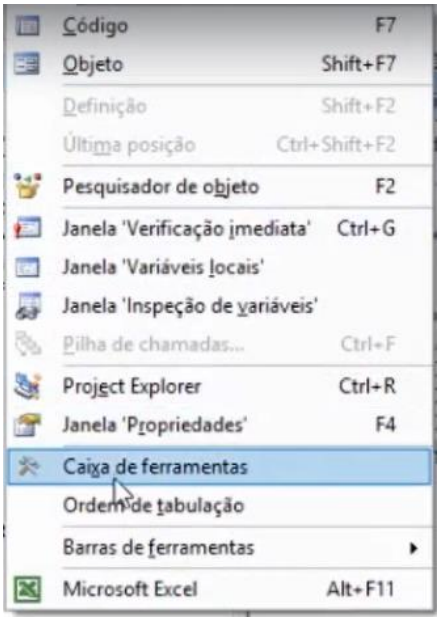
Sendo as duas células (G4 e H4) as próximas a serem preenchidas e, passando o seu conteúdo para a próxima linha, esvaziadas.

c. Formulários

Os formulários permitem mais interação programa/usuário que a MSGBOX e a INPUTBOX. Para criar um formulário, clique com o botão direito do mouse na janela de projetos, em Inserir e em UserForm. Observe que a barra dos formulários fica à esquerda na tela logo abaixo dos objetos, no entanto, ela só aparecerá se já existir um formulário criado.



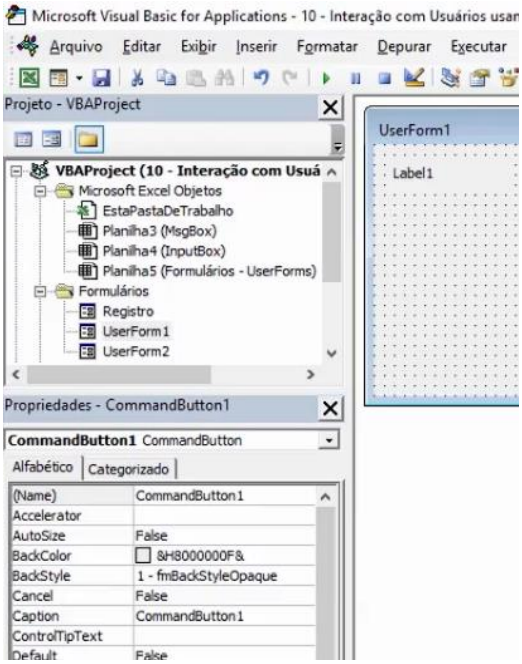
Para visualizar um formulário existente dê duplo clique sobre o seu nome e ele aparecerá juntamente com a sua caixa de ferramentas, é nela que será feita toda a edição dos formulários. Caso a última não esteja aparente, clique no menu “Exibir” e, logo, em “caixa de ferramentas” como na imagem.



Normalmente as ferramentas básicas já estão aparentes, mas, caso deseje editá-las, clique com o botão direito do mouse em seu corpo, busque “Controles Adicionais” e selecione os que quiser.

Para editar um formulário, clique e arraste o item desejado diretamente da caixa de ferramentas para o espaço do formulário (adicionar item) ou dê um clique no espaço de edição e faça as mudanças necessárias (reeditar). Vale ressaltar que cada elemento dentro do

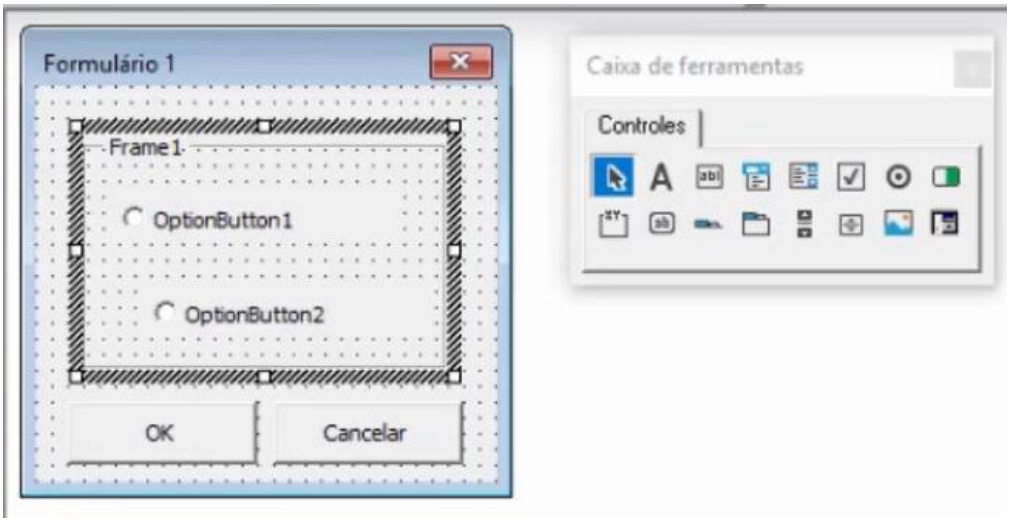
formulário possui uma lista de propriedades que ficam à mostra no canto inferior esquerdo da tela, podendo ser estendida para cima a fim de ter uma melhor e mais completa visualização. Nela, pode-se mudar as características dos itens inseridos no formulário. Observe a imagem abaixo.



Os botões são utilizados para ativar e para desativar ações do formulário. Para editar o código relacionado a cada botão, dê duplo clique nesse e, ainda dentro da aba do formulário, a rotina será exibida e poderá ser modificada.. No código, o início [Private Sub nomedobotao_Click()] e o fim [End Sub] já estão escritos, apenas será necessário que você escreva seu corpo (ações a serem realizadas ao clicar em cada botão).

d. Uso de Botões de opção em Quadro

Os botões de opção são as opções que um formulário apresenta ao usuário, sendo apenas uma selecionada e confirmada ao clicar em OK. Para fazê-las funcionar corretamente, é necessário que haja um quadro que as englobe. Assim, primeiramente, adiciona-se um quadro (Frame) ao formulário (clique na opção “quadro” e o desenhe no formulário). Depois, arraste para dentro do quadro a quantidade de Botões de opção (OptionButton) desejada, veja como isso arranja-se antes da edição:



Edição do formulário:

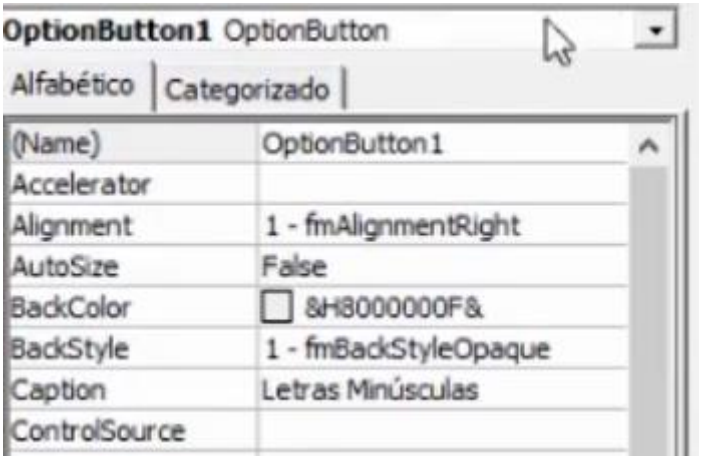
Para mudar o nome do quadro e dos botões de opção, modifique o Caption (clique no nome existente, apague-o e escreva o novo) de cada um deles na Barra de propriedades.

Para alterar a posição ou o tamanho dos itens, respectivamente, clique no desejado e mova-o arrastando o mouse ou arraste o limite lateral de cada item, expandindo-o ou comprimindo-o. Para alinhar os botões de opção, selecione-os, clicando uma vez em cada com o botão esquerdo do mouse depois clique em um deles com o botão direito do mouse, selecione Alinhar e escolha para qual posição de alinhamento (esquerda, centro, meio, …).

e. Adição do Código para um Formulário Funcionar

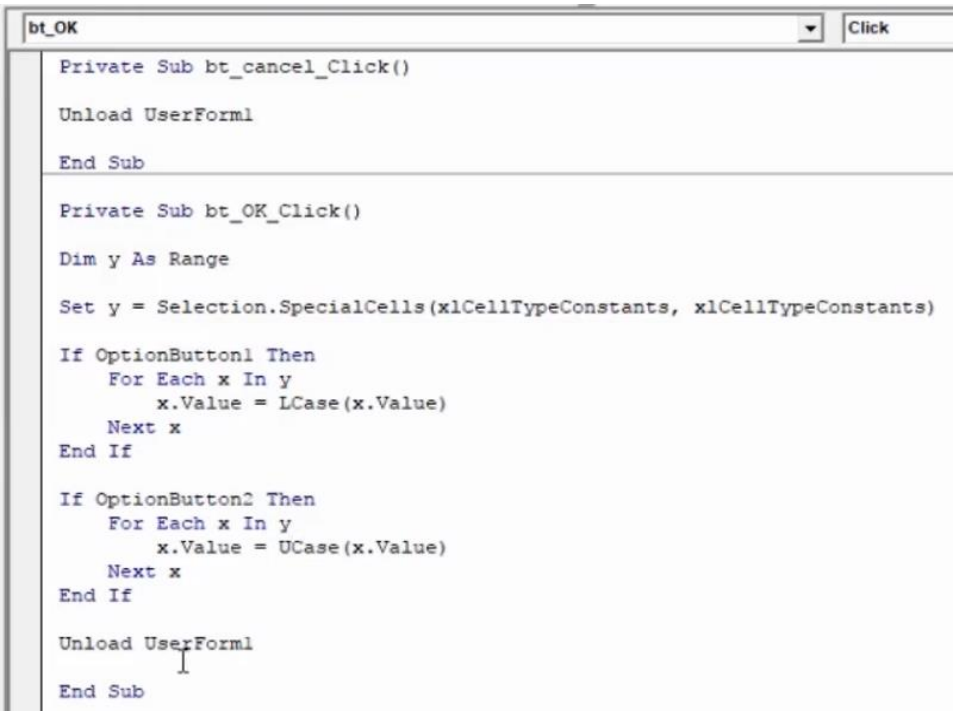
Após a estrutura do formulário estar pronta, é necessário que se programe o código implícito em cada botão. Para isso, no VBE, dentro da janela do botão desejado (por exemplo, o OK) já terá escrito Private Sub bt_OK_Click() e End Sub, que são, respetivamente, o início e o fim do programa; no seu meio, você escreve o que deseja que aconteça caso cada um dos botões de opção seja selecionado com um IF utilizando o nome de cada um deles. Por exemplo, para o botão abaixo ficaria:

```
If OptionButton1 Then

(o que fazer)

End If
```



Por fim, para fechar o formulário após o clique no botão desejado e a realização de seu corpo, usa-se o Unload UserForml. Para botões como o Cancel, utiliza-se apenas essa função.

Por exemplo, para que o formulário coloque todas as palavras das células selecionadas em minúsculo (OptionButton1) ou maiúsculo (OptionButton2), dependendo da escolha do usuário, é necessário que haja uma estrutura como:

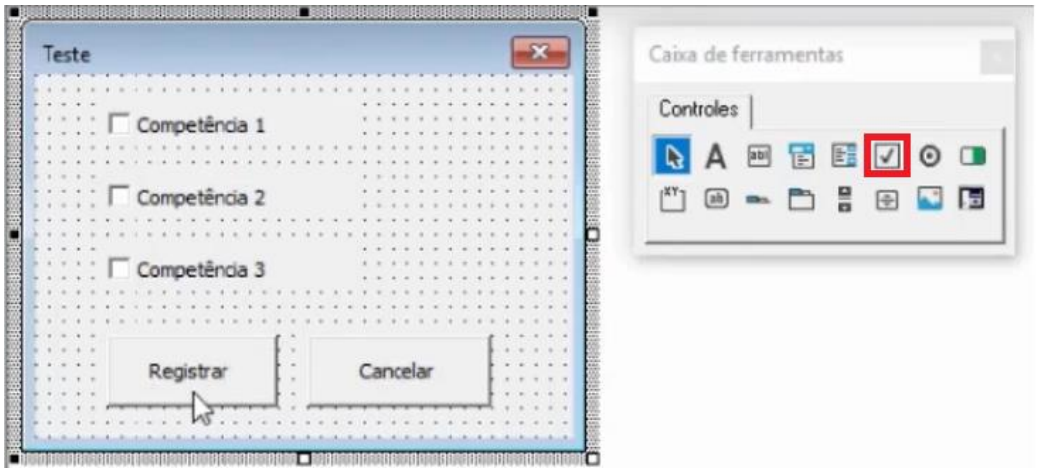


Obs.: A função Set y = Selection.SpecialCells (xlCellTypeConstants, (xlCellTypeConstants) faz com que todas as células selecionadas sejam alteradas (selection) e garante que as células só tenham texto dentro, o que é essencial ao processo desejado.

11. Outras formas de uso de formulários

a. Caixa de seleção (CheckBox)

É uma ferramenta que te permite fazer múltiplas escolhas com a possibilidade de selecionar mais de uma opção. Para utilizá-la, arraste-a da caixa de ferramentas para o editor do formulário quantas vezes quiser, como fez-se com os botões de opção. Colocando três caixas de seleção e mudando seus nomes para Competência 1, Competência 2 e Competência 3, pode-se visualizar:

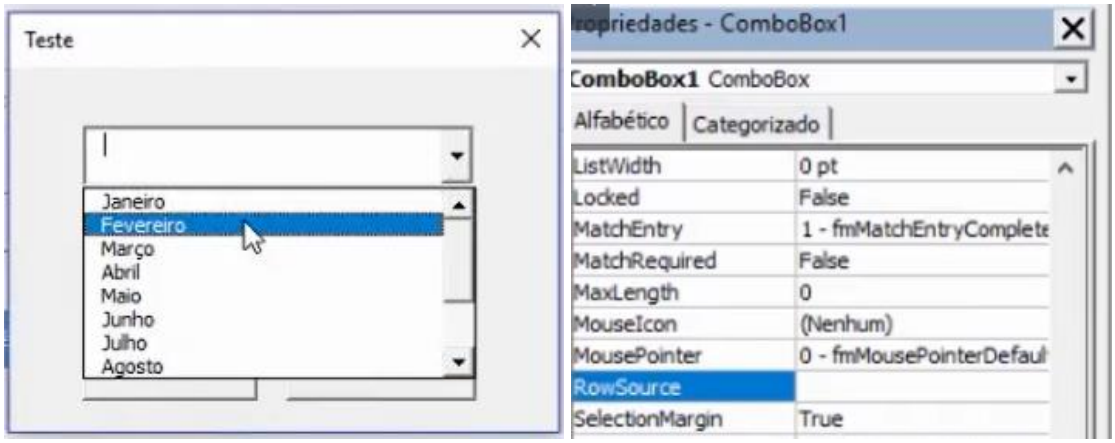


Após a estruturação do formulário, insere-se o código contido em cada botão. Para editar o que ocorre ao clicar no botão bt_ok, por exemplo, dê duplo clique nesse e a aba de inserção de código aparecerá. Nela, como na aba dos botões de opção, usa-se a condicional If … Then para agir caso a caixa de seleção esteja marcada; e fechar no final. Observe:

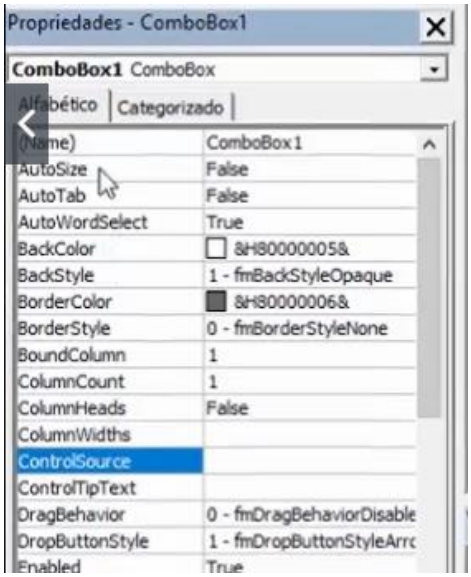
```
Private Sub bt_cancel_Click()

Unload UserForm1

End Sub

Private Sub bt_ok_Click()

If CheckBox1.Value Then
Range("D8") = "Possui"
End If

If CheckBox2.Value Then
Range("E8") = "Possui"
End If

If CheckBox3.Value Then
Range("F8") = "Possui"
End If

Unload UserForm1

End Sub
```

b. Caixa de combinação

É uma lista onde você pode escolher um dos valores listados e, de acordo com a sua seleção, você pode ter resultados modificados dentro da sua planilha. Essa caixa de combinação precisa de uma lista de dados, os quais serão listados e, depois, selecionados pelo usuário. Dentro das propriedades da caixa de combinação, mais abaixo, tem a opção RowSource; nela coloca-se o intervalo de células do Excel que se encontra a lista de dados. (Ex. : A4:A15 de acordo com a aba que está sendo utilizada ou pode-se colocar um intervalo nomeado).



Após a colocação da lista de dados, na propriedade ControlSource, define-se qual será a célula do Excel que receberá o valor escolhido na caixa de combinação. (Ex.: E3).



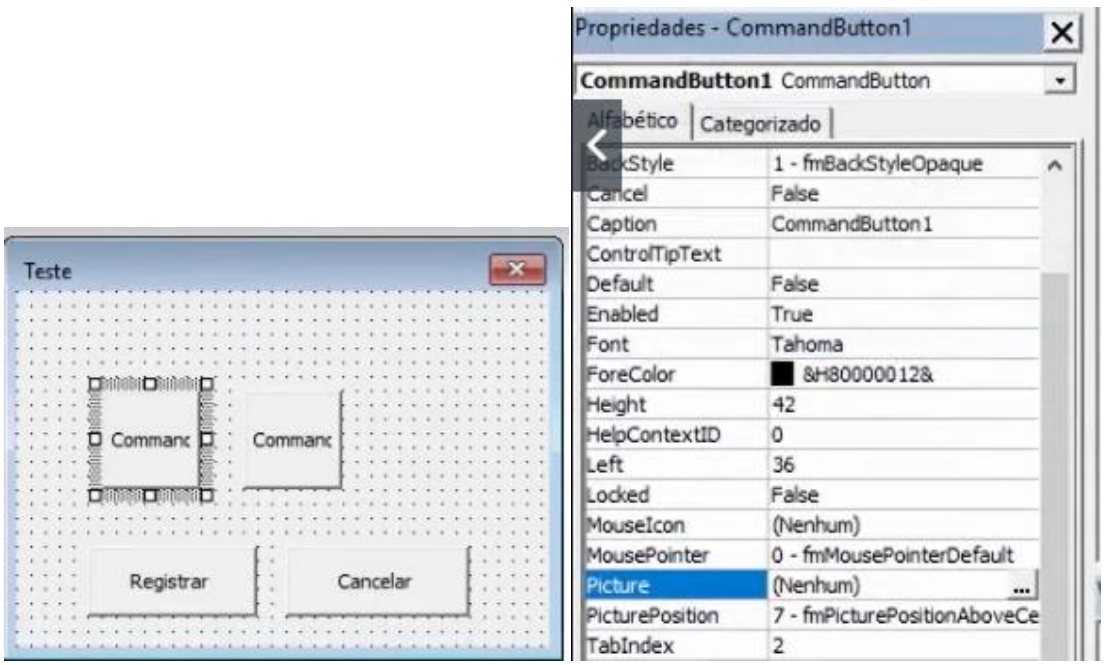
Para relacionar outras informações, após a utilização da caixa de combinação, pode-se usar funções como PROCV ou PROCH, dependendo se a tabela está na vertical ou na horizontal, respectivamente.

c. Caixa de listagem

É uma alternativa à caixa de combinação; elas têm basicamente as mesmas propriedades. Na opção RowSource coloca-se o intervalo de células do Excel que se encontra a lista de dados e na opção ControlSource define-se qual será a célula do Excel que receberá o valor escolhido na caixa de combinação.

d. Imagem no botão de comando

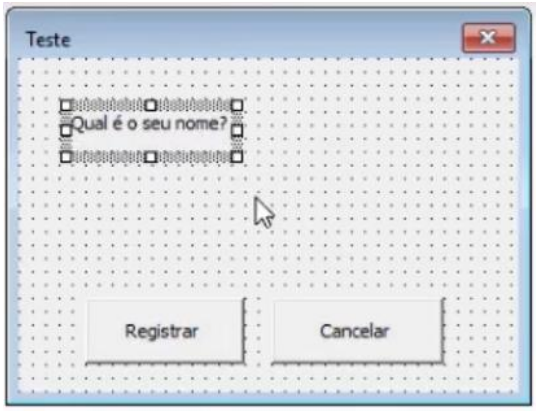
Para formulários mais visuais, pode-se adicionar imagens em botões de comando. Para isso, você precisa criar botões no formulário e alterar, em propriedades, a sua Picture (imagem) clicando nos três pontinhos e selecionando uma imagem do seu computador. Ou, no Excel, você pode copiar uma imagem e colar na propriedade Picture do botão a ser modificado.

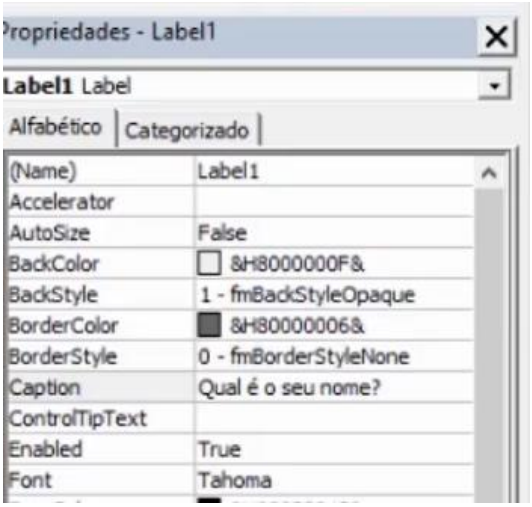


Obs.: cuidar para que botão e imagem tenham basicamente o mesmo formato para que o botão final fique razoável esteticamente e proporcionalmente.

e. Rótulos e Caixa de texto

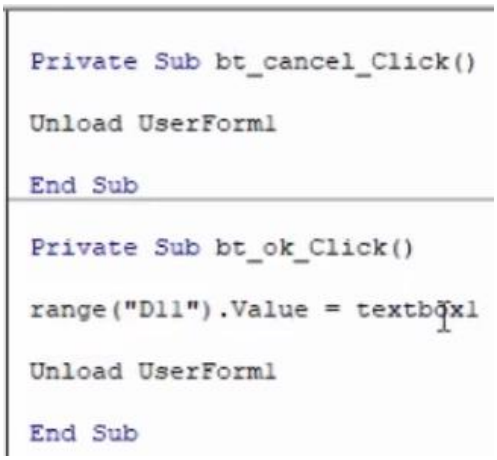
Os rótulos são elementos mais ilustrativos, são espaços para colocar mensagens fixas no formulário, como, por exemplo, perguntas. Na barra de propriedades do rótulo, pode-se mudar a cor de suas letras, de seu fundo, seu tipo de letra, entre outros.





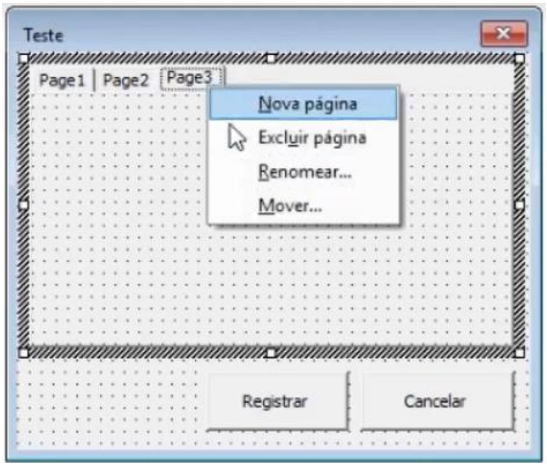
As caixas de texto servem para registrar as respostas dos usuários durante o uso do formulário. E para que a resposta seja salva em uma célula do Excel, utiliza-se o padrão:

range (célula) . Value = nome da caixa de texto



f. Ferramentas multi páginas

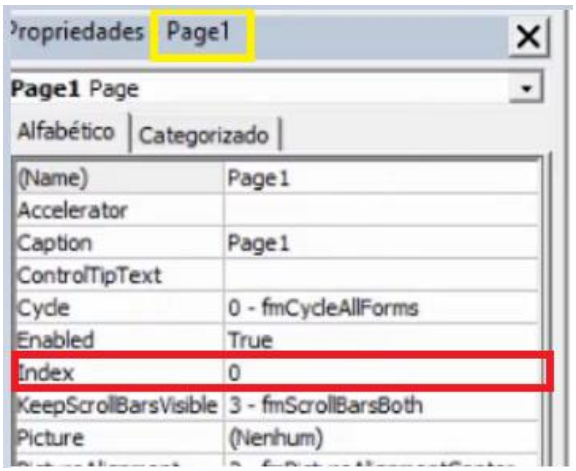
O controle multi página permite que você tenha mais de uma página em um único formulário, com conteúdos diferentes. Para inseri-lo no formulário basta arrastar o ícone da caixa de ferramentas para dentro da tela do formulário. Para inserir, excluir, renomear ou mover as páginas, clique com o botão direito do mouse na aba com o nome da página.



Os botões podem ser tanto gerais para o formulário, estando fora da ferramenta multi páginas, quanto específicos de cada página (para isso arraste o botão para dentro da página desejada). Para criar um botão que leve o usuário de uma página a outra, o código desse botão deve ser na forma:

nome da página que está . Value = Index da página que deseja ir

Cada página tem, em suas propriedades, seu Index, que é um valor numérico inteiro e positivo atrelado a ela (Página 1 -> Index 0; Página 2 -> Index 1; Página 3 -> Index 2 e segue).



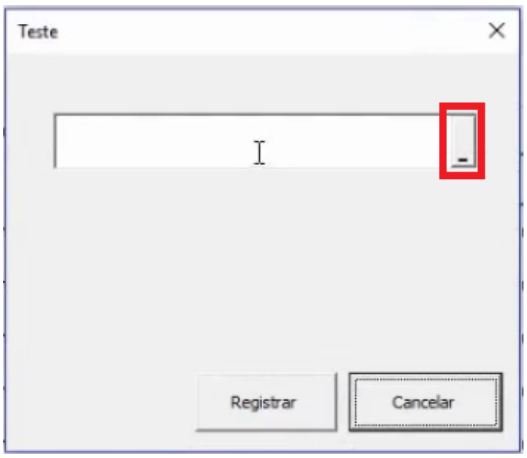
Assim, para ir da página 1 para a 2 o código dentro do botão fica:

```
MultiPage1.Value = 1
```

g. Seleção de um intervalo

Edição de referências (RefEdit): retorna o texto do intervalo que estiver selecionado. Se ela não estiver aparente, colocar em “buscar controles adicionais” clicando com o botão direito na barra de ferramentas.

Na utilização do RefEdit, primeiramente, deve-se selecionar um intervalo de edição. Para isso, na aba do Excel, clica-se, primeiramente, no botão criado por você que abrirá o formulário, depois, clica-se na barra lateral direita do editor de referências.

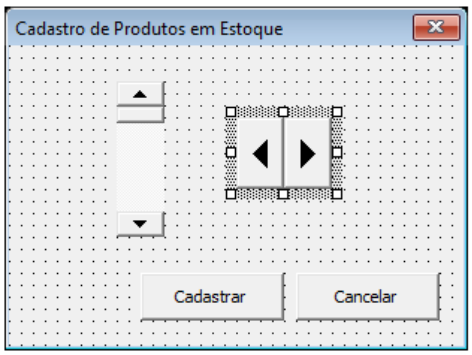


O próximo passo é selecionar o intervalo desejado pelo usuário e clicar no botão “Registrar” do formulário para que ocorra o que estiver em seu código. Tem-se como exemplo o código abaixo, preenchendo o interior das células de verde e mudando a cor dos caracteres para branco.

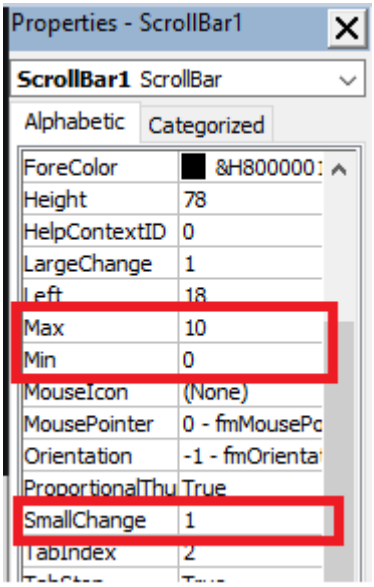
```
Private Sub bt_ok_Click()

Dim x As Range

Set x = Range(RefEdit1.Value)

With x
    .Interior.Color = vbGreen
    .Font.Color = vbWhite
End With

Unload UserForm1

End Sub
```

h. Barra de rolagem e botão de rotação

As barras de rolagem (ScrollBar) e os botões de rotação (SpinButton) são ferramentas utilizadas para modificar valores de acordo com seu uso.



Barras de rolagem: podem ser verticais ou horizontais, basta que o usuário a “estique” ou a “comprima” com o mouse, arrastando suas extremidades. Em uma barra de rolagem ou em um botão de rotação, define-se um valor máximo (Max), um valor mínimo (Min) e um valor que será a diferença entre cada aumento (SmallChange).



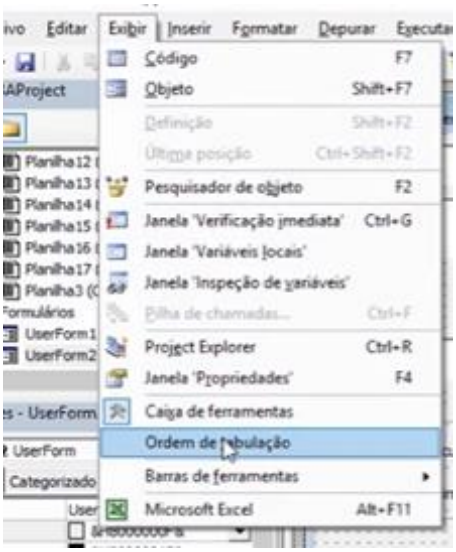
Para podermos visualizar o valor que está sendo marcado, adicionamos um rótulo dinâmico e colocamos uma linha de código dentro da barra de rolagem ou do botão de rotação para criar uma ligação entre eles. Por exemplo, para a barra de rolagem:

```
Private Sub ScrollBar1_Change()

Label1.Caption = ScrollBar1.Value


End Sub
```

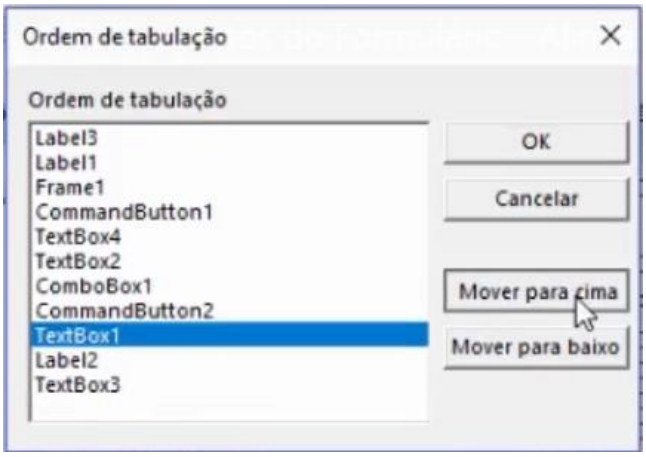
i. Alinhamento de controles

O alinhamento da posição das ferramentas dentro do formulário serve para uma elegante e organizada visualização. Ele pode ser feito manualmente, selecionando e arrastando cada item ou selecionando os itens desejados e os alinhando entre eles; para isso, clica-se no botão direito do mouse, seleciona-se Alinhar e, então, o lado de preferência.

Ordem de tabulação: ao clicar no atalho de teclado TAB o cursor vai para o próximo espaço a ser preenchido. Para organizá-la, clica-se na aba Exibir e seleciona-se Ordem de Tabulação.



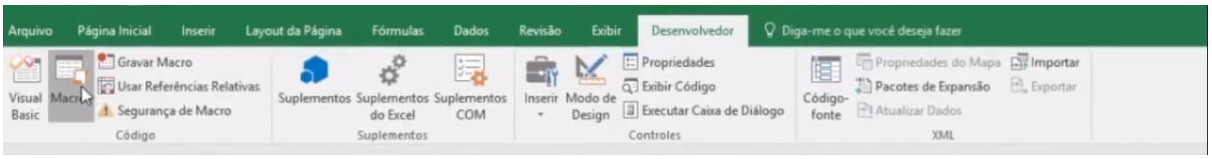
Então, você terá uma lista de itens ordenados, onde os mais altos são os primeiros. Para modificá-la, selecione o item a ser mudado e vá clicando nos botões laterais a fim de organizá-los.



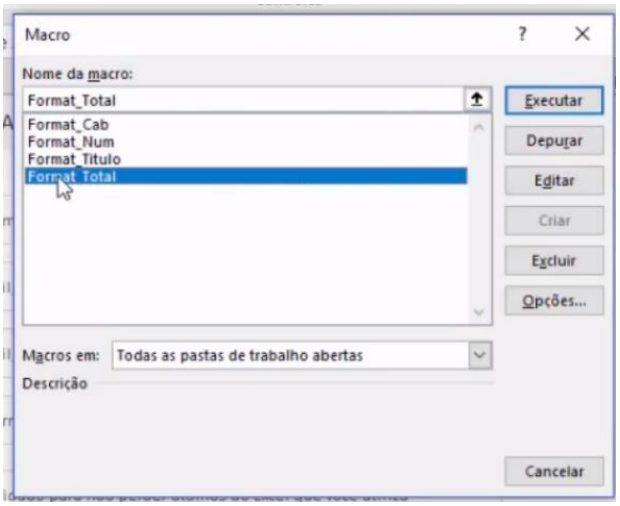
12. Rodando e armazenando código VBA

a. Macros na barra de ferramentas de acesso rápido

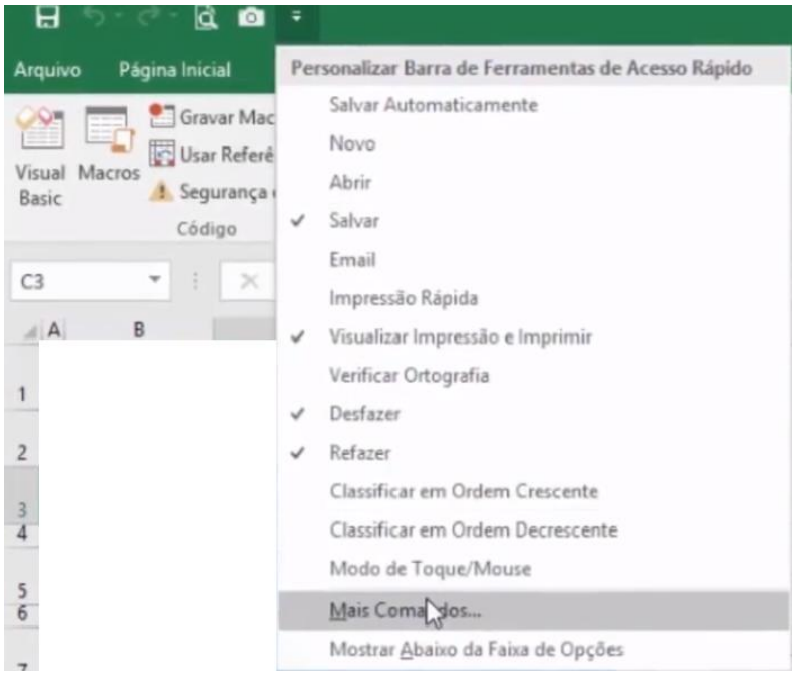
Todos os procedimentos criados até agora podem ser acessados através da guia Desenvolvedor, dentro do grupo de Código na opção Macro. Ou utilizando o atalho de teclado “ALT + F8”.



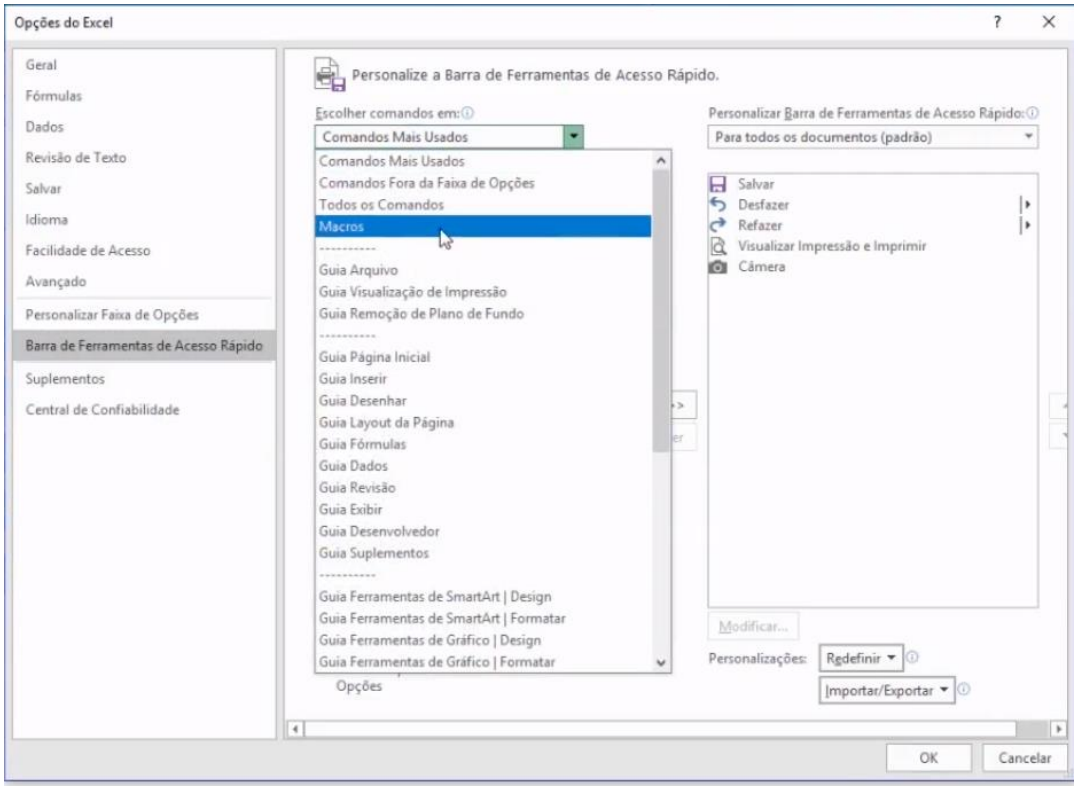
Após clicar na opção macro ou utilizar o atalho de teclado você terá a visualização de todas as macros armazenadas na planilha em questão, conforme imagem.



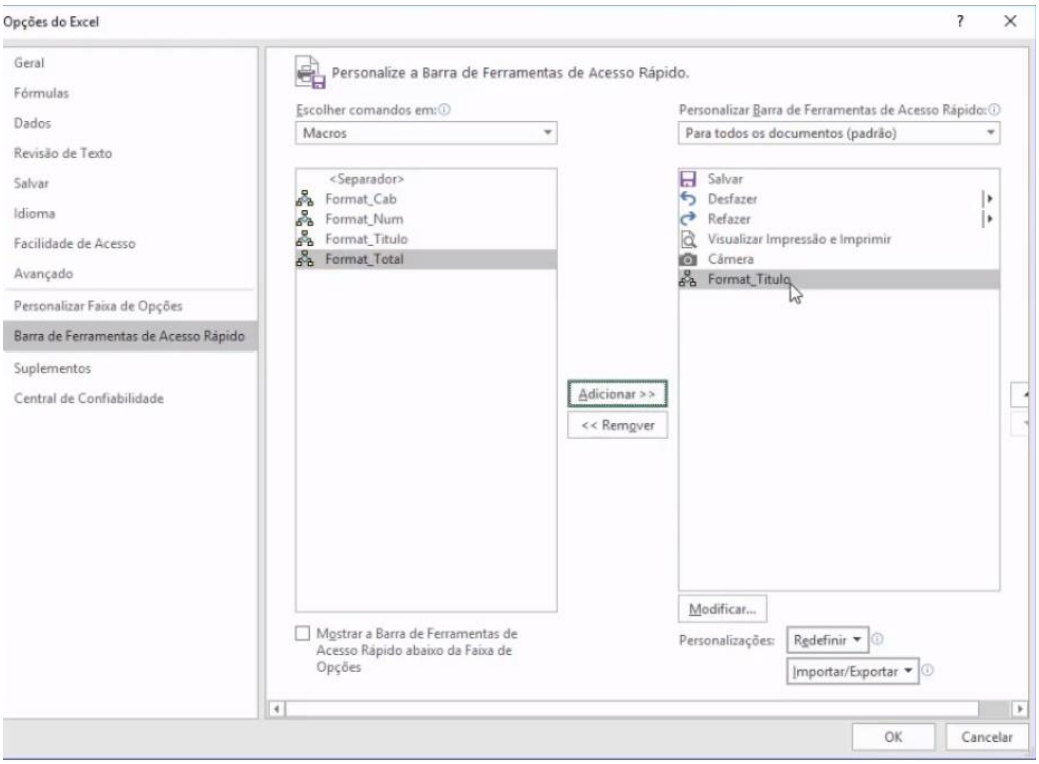
Porém existe a possibilidade de deixar as macros em locais mais intuitivos para os usuários, por exemplo, colocando-as na Barra de Acesso Rápido. Para fazer isso você deve clicar na opção "Personalizar Barra de Ferramentas de Acesso Rápido" e após clicar na opção "Mais Comandos".



Em seguida clique na lista suspensa “comandos Mais Usados” e selecione a opção “Macros”, conforme imagem.



Você pode adicionar a macro à barra de acesso rápido clicando em adicionar.

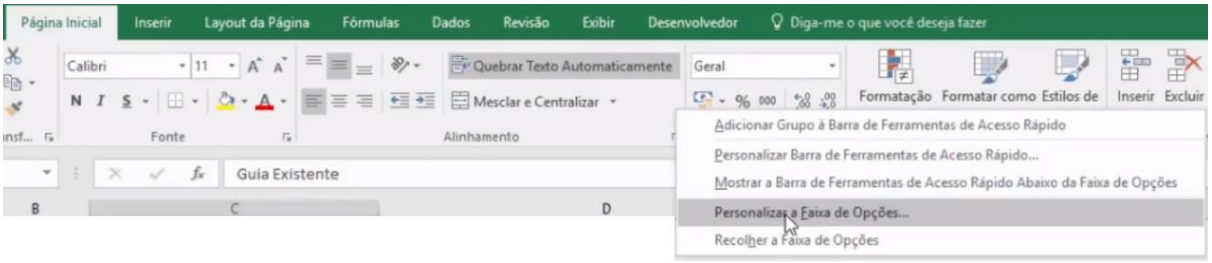


Além disso, clicando na opção “Modificar”, você pode alterar o nome do macro exposto na barra de ferramenta de acesso rápido para um formato mais amigável - essa alteração NÃO altera o nome do macro salvo somente o apresentado na barra de acesso rápido. E alterar a imagem da macro para uma a seu gosto.

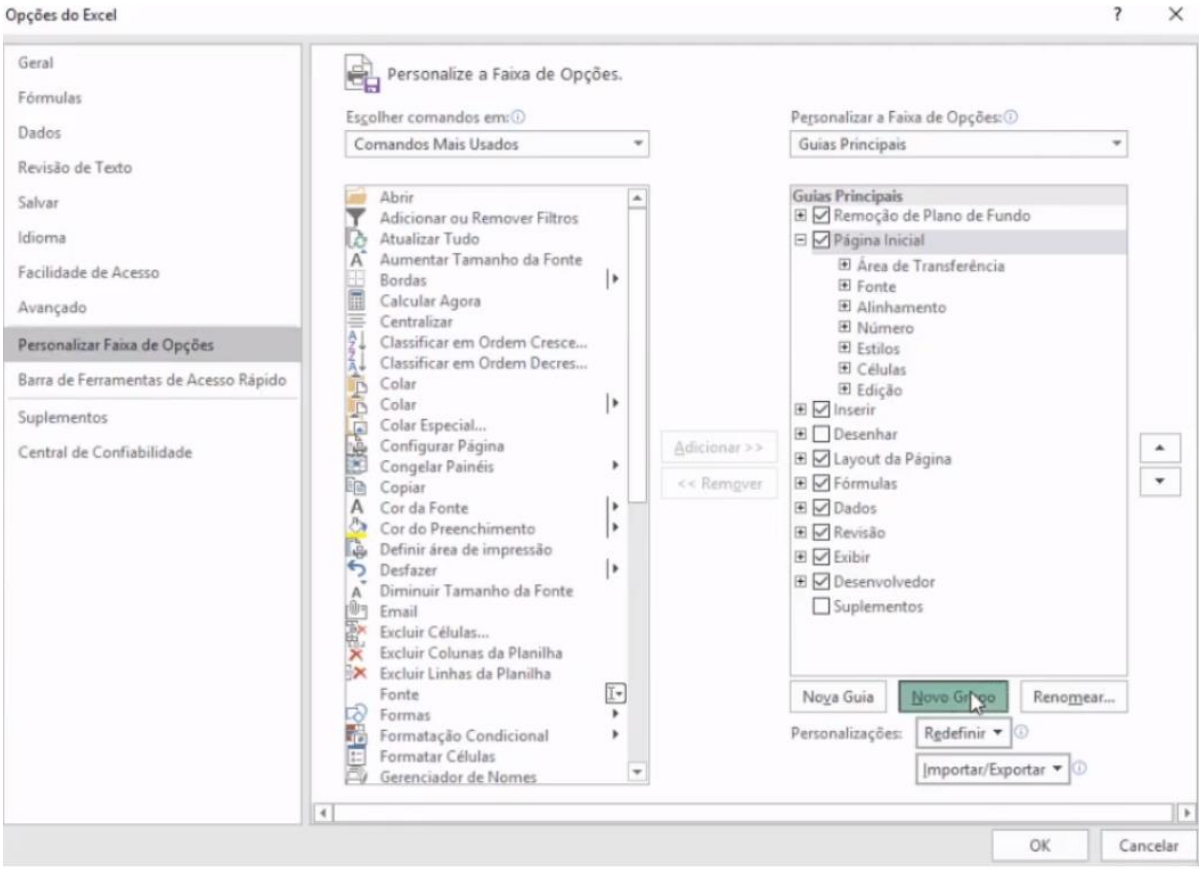
O resultado final é uma nova ferramenta na Barra de Acesso Rápido podendo ser acessado com apenas um clique, como por exemplo na imagem a seguir.

b. Vincule macros a uma guia

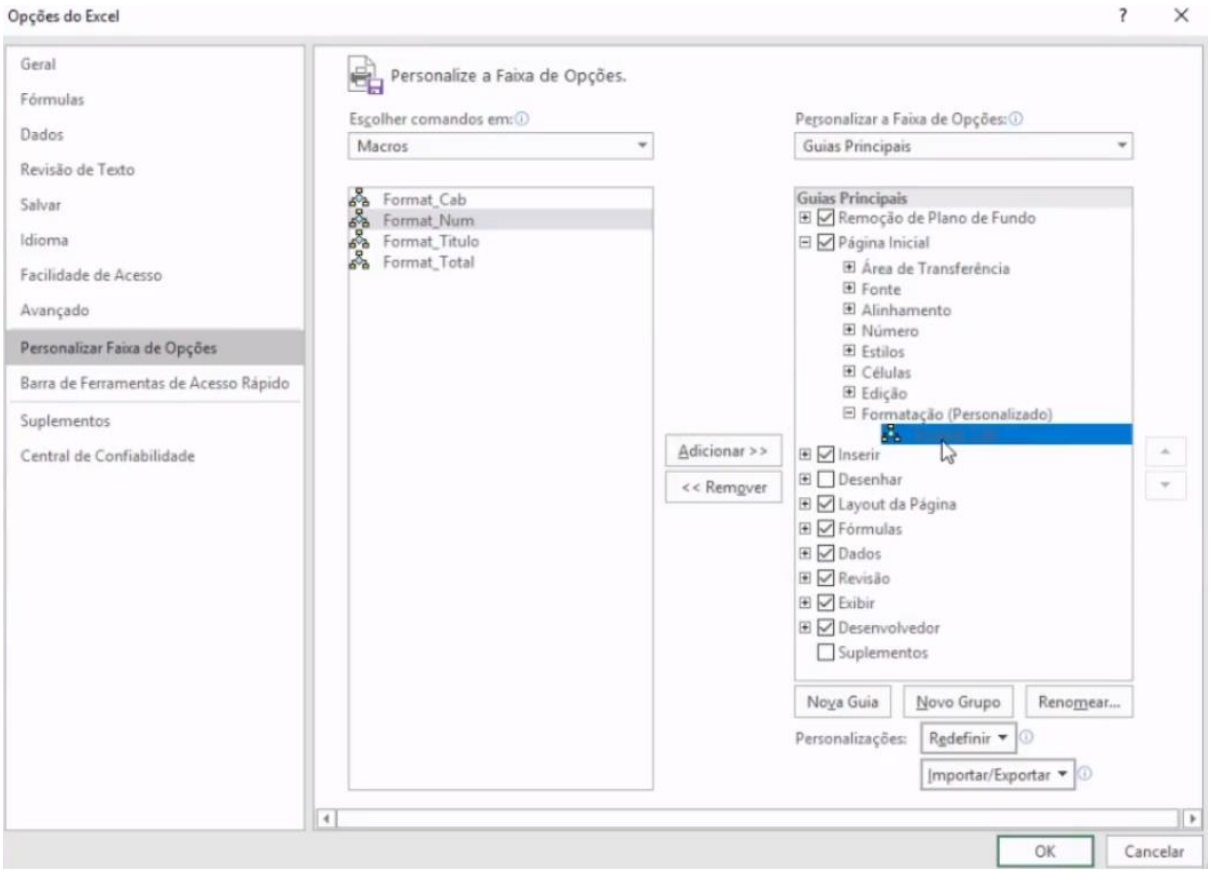
Para adicionar uma macro a uma guia basta clicar com o botão direito do mouse na guia que se deseja adicionar a macro e em seguida clicar na opção “Personalizar Faixa de Opções”, conforme imagem.



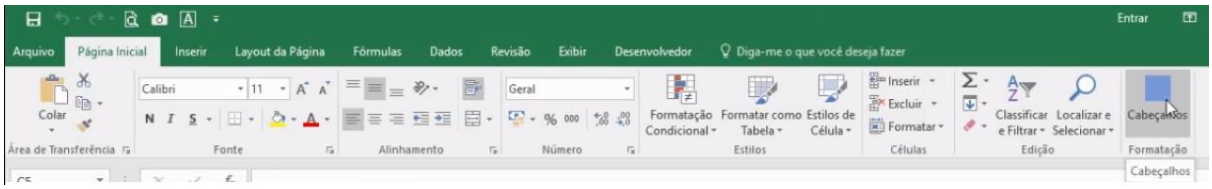
Na sequência, com a guia destino selecionada, clique em novo grupo, e após clique em renomear para editar o nome e o ícone do novo grupo, de forma semelhante ao explicado no tópico anterior.



Em seguida, escolha a opção “Macros” na lista suspensa “Escolher comando em:” escolha a macro desejada e clique em “Adicionar”.

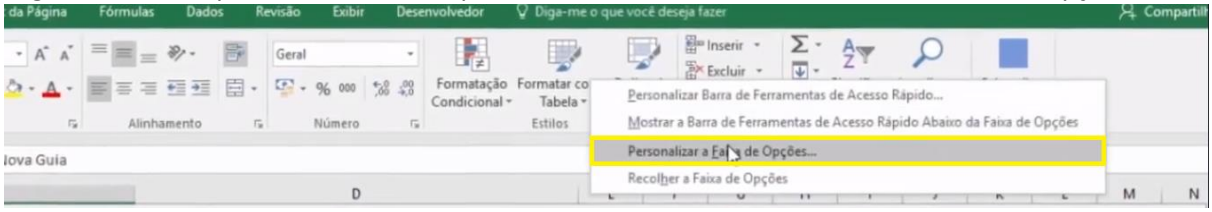


Por fim, clique em renomear para editar o nome e ícone, da mesma forma feita anteriormente e clique em “OK”. O resultado final é uma nova função na de guia de ferramentas podendo ser acessado com apenas um clique, como por exemplo na imagem a seguir.

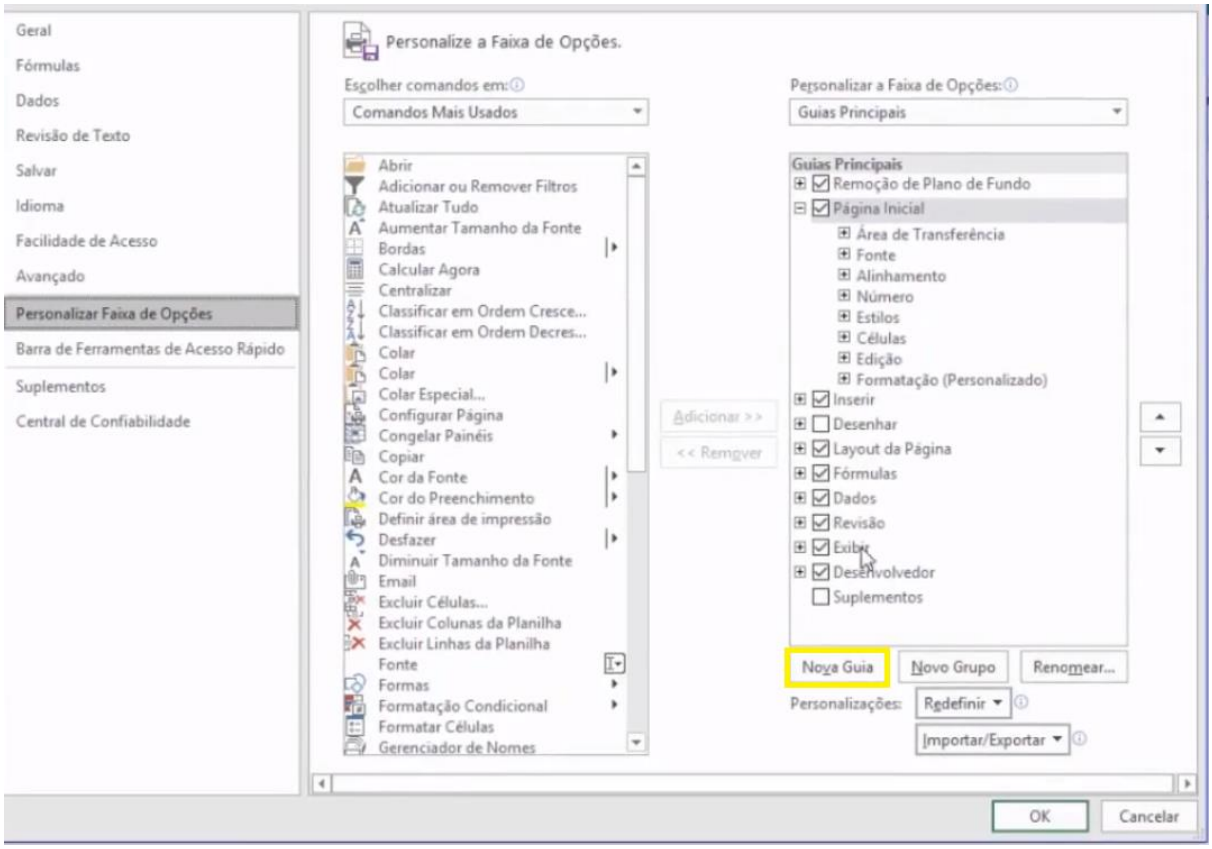


c. Crie uma nova guia

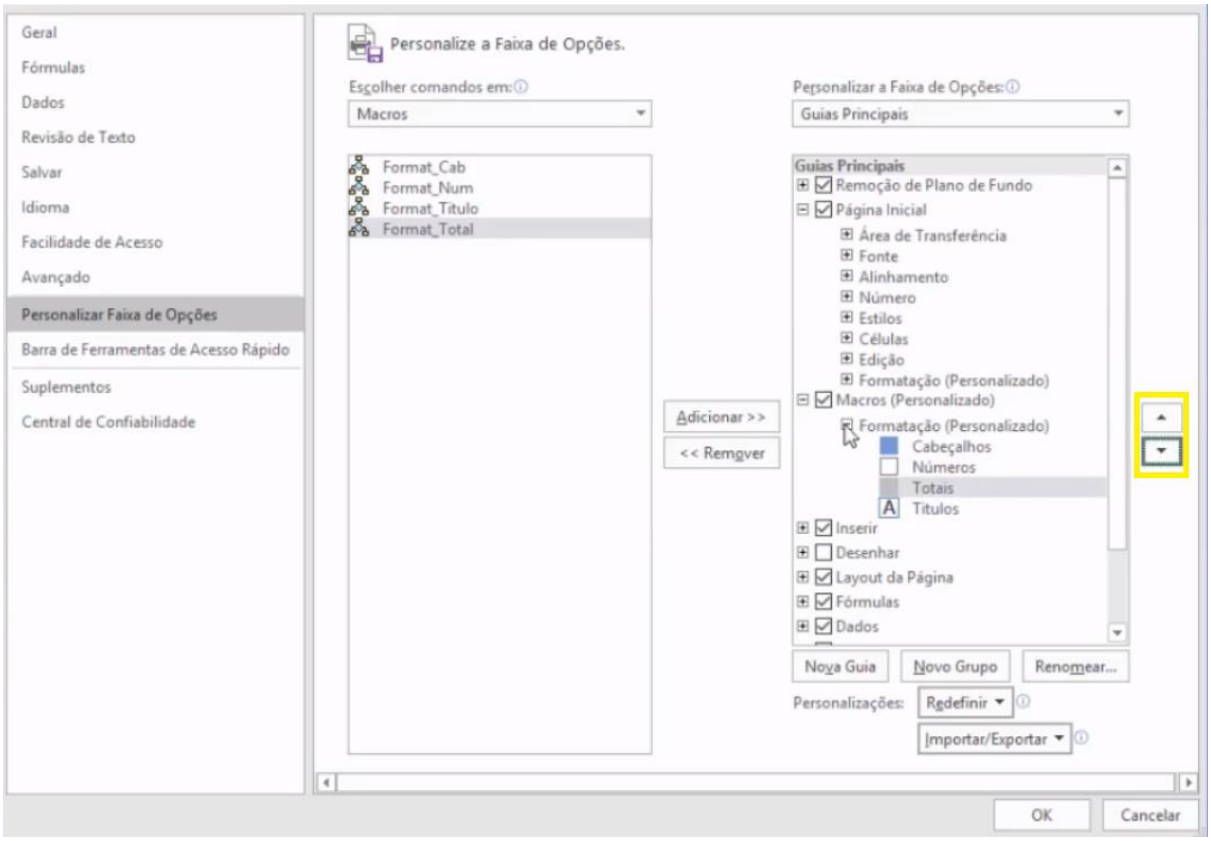
Quando se deseja adicionar mais de um atalho de macro nas guias talvez seja interessante criar uma nova guia para eles. Para isso, clique com o botão direito nas guias, em seguida clique em personalizar a faixa de opções.



na nova janela, em guias principais clique em uma nova guia e renomeie conforme desejar, após adicione as macros já criadas conforme explicado no tópico anterior.



Para finalizar defina a posição onde as macros devem se encontrar, bem como a posição na qual a nova guia deve ficar e clique em ok. Para isso utilize as setas laterais destacadas abaixo para movimentar as posições.

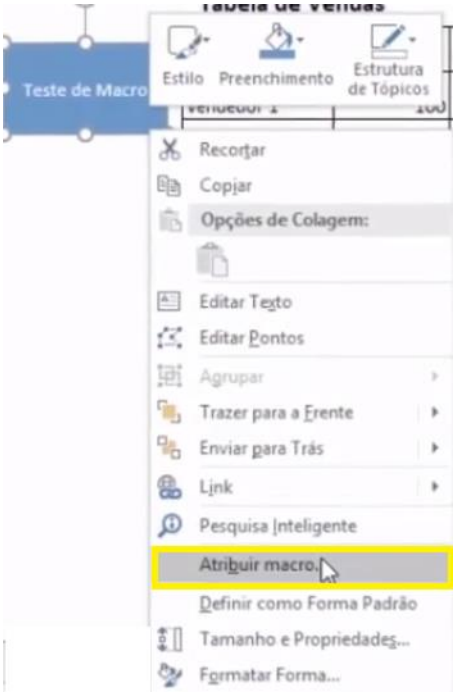


por fim o resultado deve ser semelhante a imagem abaixo:

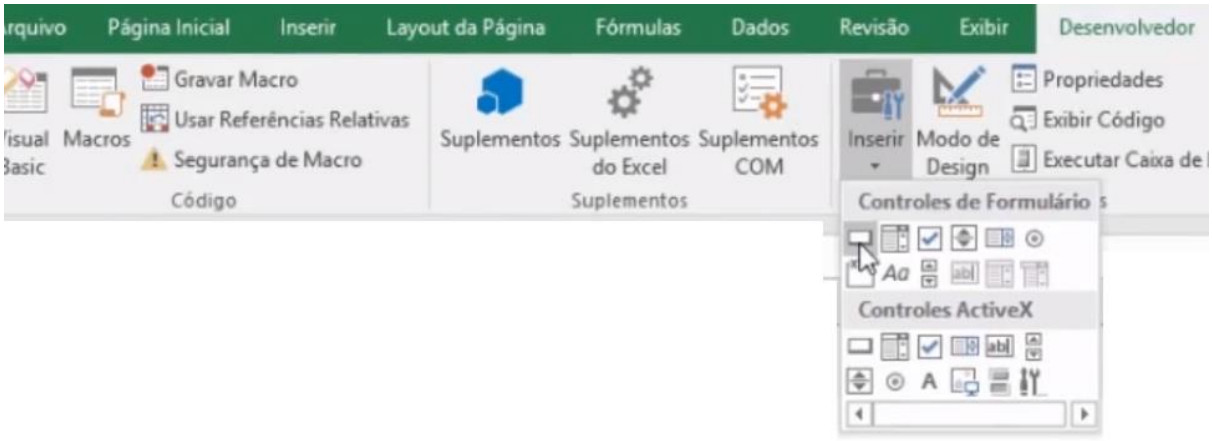
d. Adição de botões em planilhas

Para adicionar botões de macros no Excel você deve clicar na pasta na qual a macro será executada, em sequência clique na guia inserir e insira uma forma.

Após, clique com o botão direito do mouse na forma e selecione a opção atribuir macro e adicione a macro desejada. Termine clicando em ok. Para executar a macro selecione um intervalo em seguida clique no botão adicionado.

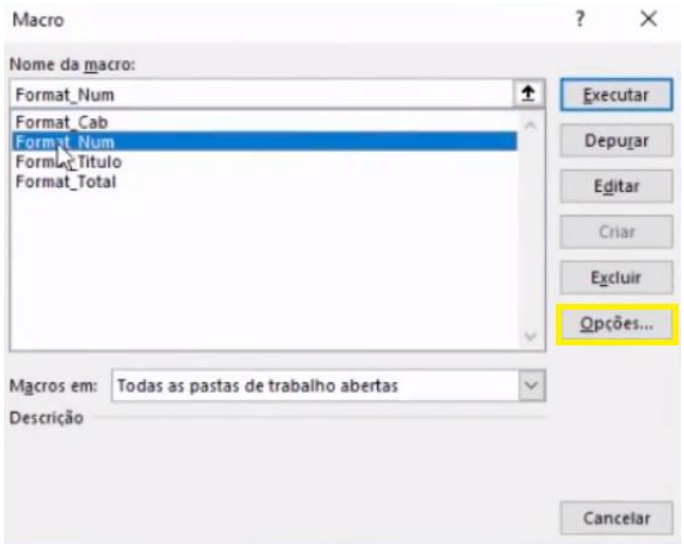


Outra forma de adicionar botões no Excel é por meio da guia desenvolvedor na opção inserir. Lá você pode adicionar controles de formulário que funcionam de forma análoga ao explicado anteriormente. Para adicionar controles de formulário siga o caminho da foto abaixo.

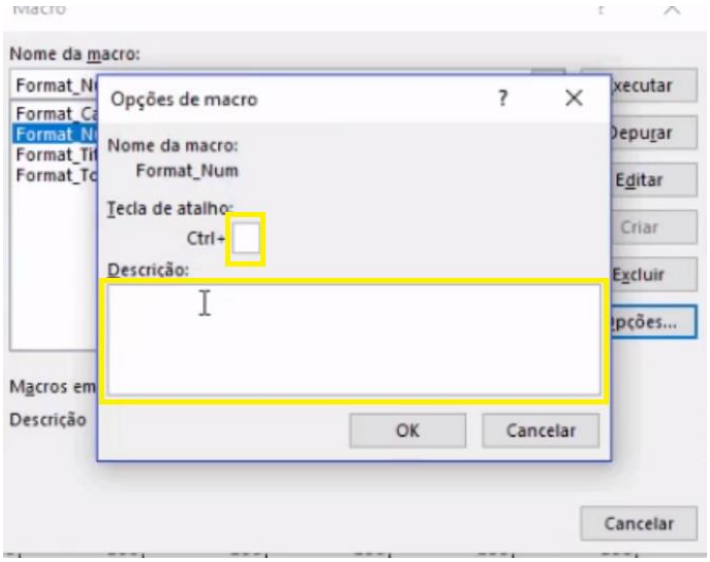


e. Adição de atalho de teclado

Na guia desenvolvedor no grupo códigos clique na opção Macros, em sequência clique na macro que você deseja adicionar um atalho de teclado e clique em opções.

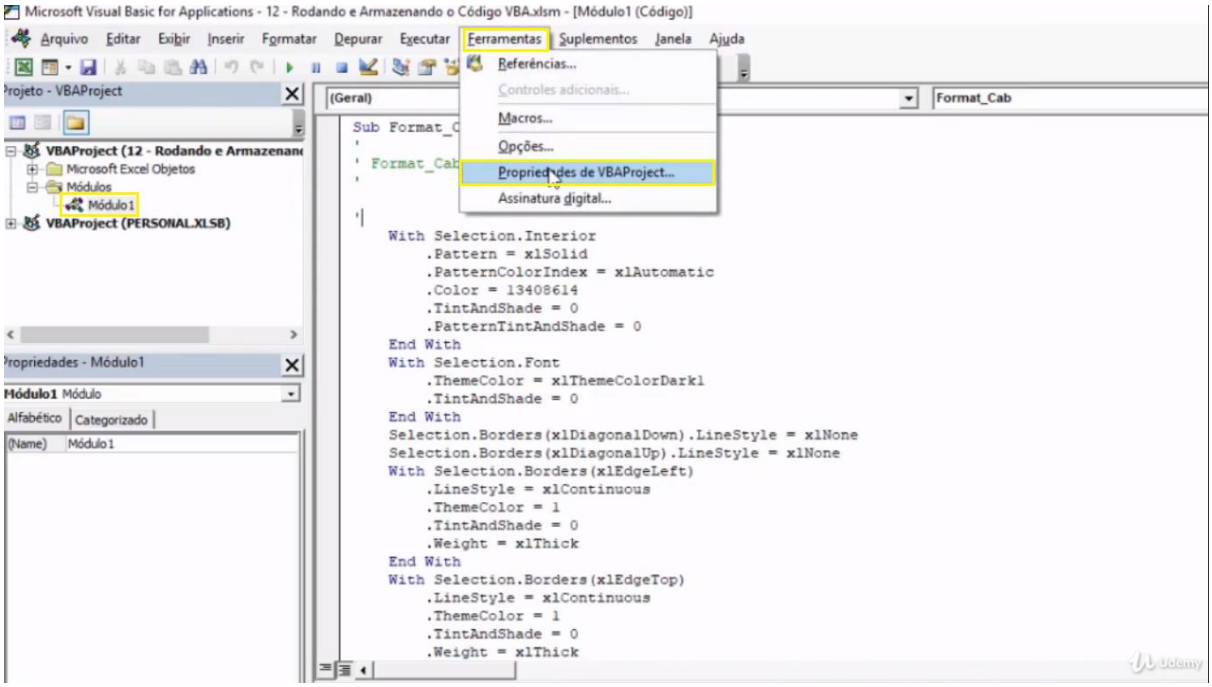


Feito isso, adicione uma descrição para a sua macro e adicione o atalho que desejar, fique atento para não utilizar um atalho já existente, para evitar que isso ocorra é sugerido que se utilize a tecla “shift” junto ao botão selecionado. Para adicionar a descrição e o local do atalho oriente-se pela imagem abaixo.



f. Salvar seu código VBA com senha.

Para bloquear a visualização de seu código você deve abrir o visual basic através da guia desenvolvedor, em sequência selecionar o módulo em que se encontra o código, após isso clicar em: ferramentas > propriedades de VBAProject.



Finalize clicando em: proteção > bloquear projeto para exibição, em sequência insira a senha e confirme-a, após clique em “ok”.