

Macros VBA para Excel Completo

Apostila VBA

Macros VBA para Excel Completo

Link para o Curso Online: <http://bit.ly/vba-udemy>

Sumário

Introdução	4
Seção 2 – Ambientação e Primeiros Passos	5
Introdução – Formas de Gravar uma Macro	5
Ambiente VBA	12
Estrutura VBA e Variáveis	15
Primeira Ferramenta – Hello World e MsgBox	18
Seção 3 – Fundamentos e Valores	20
Cells.Value e ActiveCell	20
Operações Básicas e Executando com Botão	23
Debugando	27
Segunda Ferramenta – Planilha de Cálculos Automáticos	32
Seção 4 – Fundamentos Range e Propriedades	34
Objeto Range	34
Propriedades do Range e Aprendendo com o Gravar Macros	37
Propriedade Offset	40
Seção 5 – Estruturas do VBA	43
Estrutura If – Else	43
Estrutura For e For Each	46
Terceira Ferramenta	49
Estrutura With	54
Estrutura Select Case	56
Estrutura While e Adicionando Condições	60
Seção 6 – Funções e Subs	63
Sub x Function e Organização do Código	63
Argumentos ByRef e ByVal (Evitando Erros)	66
Integração Função VBA e Fórmula Excel	68
Quarta Ferramenta - Consolidação	71
Seção 7 – Tratamento de Erros	76
GoTo e Labels	76
On Error	78
Quinta Ferramenta – Compilação Despadronizada	82
Seção 8 – Formulários e Boxes Completos	85
MsgBox Completo	85
InputBox	90
UserForm – Primeiro Formulário	93

TextBox	99
ComboBox	101
CheckBox e OptionButton	107
ListBox e Eventos do UserForm.....	111
ToggleButton e Frame.....	115
MultiPágina e TabStrip	119
Sexta Ferramenta – Registro de Compras.....	121
Seção 9 – Funções do Excel e do VBA	127
Funções do Excel.....	127
WorkSheetFunction.....	129
Funções VBA	131
Seção 10 – Acelerando o Código	133
Application.ScreenUpdating	133
Application.Calculation	136
Seção 11 - Eventos de Planilha e de Pasta de Trabalho.....	139
Eventos WorkBook.....	139
Eventos WorkSheet.....	142
Sétima Ferramenta – Atualização Automática com Eventos, Impressão e Mudança de Arquivo	146
Encerramento do Curso.....	150

Introdução

O Excel é uma ferramenta bem útil que possibilita a inserção de textos, números, fórmulas, cálculos, gráficos, entre outras operações que são bem úteis em diferentes cenários. Para facilitar processos de empresas, ou até mesmo de rotinas básicas. Desde o mais básico para anotações de dados até o mais complexo de uma planilha de análise de dados de uma empresa que consiste em várias etapas, como compra, venda, estoque, lucro, gastos, e outros dados, o que varia em cada caso.

Para tornar a utilização do Excel ainda melhor podemos utilizar o **Visual Basic for Application** (VBA) que é a programação por trás do Excel, ou seja, todas as fórmulas e ações feitas no programa podem ser descritas em código. Portanto o VBA é uma linguagem de programação que permite ao usuário reproduzir não somente o que é possível fazer com fórmulas e seleções dentro do ambiente Excel como permite ao usuário automatizar processos, otimizar problemas, criar novas fórmulas e macros, criar formulários, ou seja, há uma infinidade de atividades que podem ser feitas utilizando a programação.

Neste curso além de aprender o que é de fato o VBA, vamos aprender como é a estrutura do código, como que são as sintaxes, como navegar no ambiente VBA, como construir algumas ferramentas, formulários, entre outras atividades.

Portanto o VBA é um conhecimento aprofundado de Excel que permite fazer operações e atividades mais complexas com a utilização da programação, que é a escrita do código dizendo passo a passo o que será feito no código. É desta forma que as fórmulas do Excel funcionam.

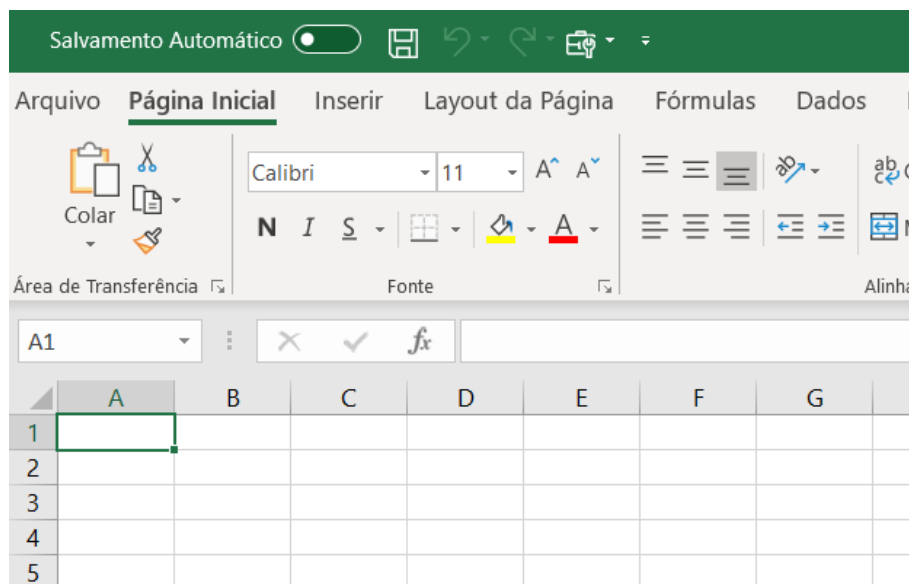
Para acompanhar o curso é necessário ter um conhecimento básico de Excel para que possam entender como a programação funciona e algumas funções, até porque o VBA possui muitas funções que são parecidas com as já existentes dentro do Excel.

Seção 2 – Ambientação e Primeiros Passos

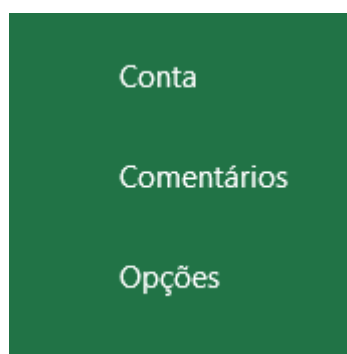
Introdução – Formas de Gravar uma Macro

O Excel permite a **criação de macros sem que haja necessidade de programação**, isso é, o usuário consegue gravar algumas ações e em seguida reproduzir essas ações sem que precise de fato escrever um código em VBA. Para que seja possível gravar uma macro é necessário inicialmente habilitar a guia Desenvolvedor.

Para habilitar a guia Desenvolvedor temos que ir na guia **Arquivo**.

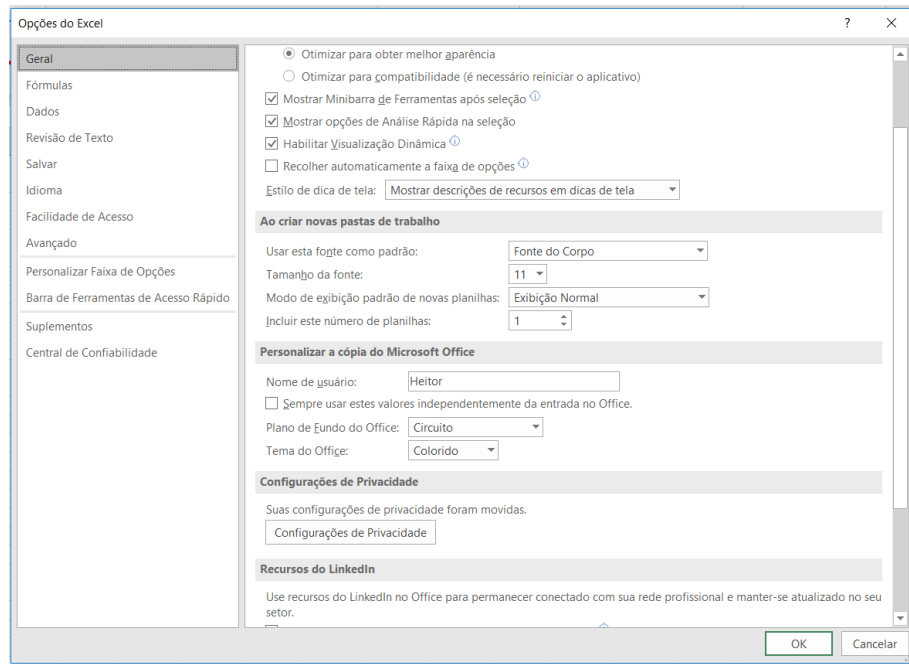


E em seguida em **Opções** no canto inferior esquerdo.

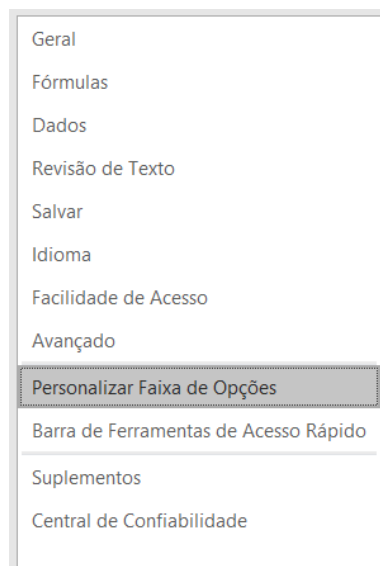


Ao clicar em opções será aberta uma nova janela com as opções do Excel.

Macros VBA para Excel Completo

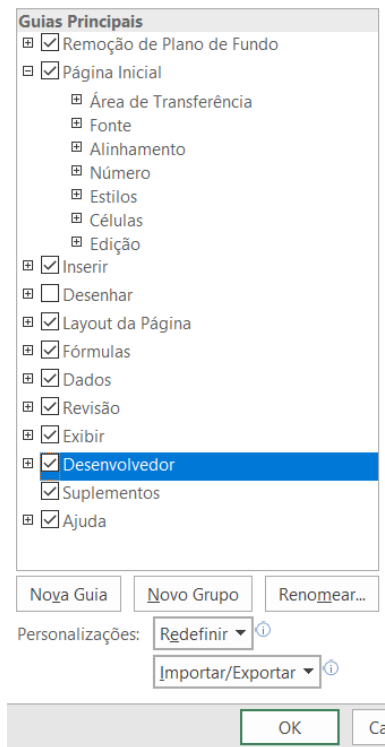


Selecionamos **Personalizar Faixa de Opções**.



E por fim marcamos a caixa da guia **Desenvolvedor** e pressionamos **OK**.

Macros VBA para Excel Completo

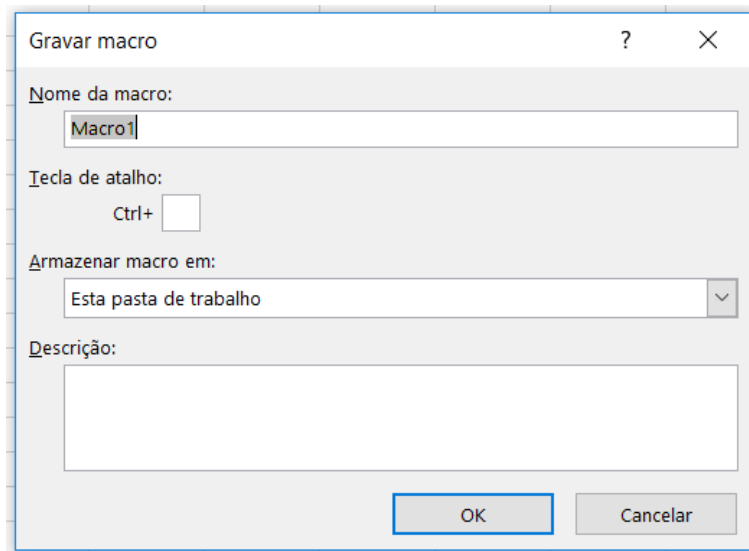


Feito isso a nova guia estará habilitada para uso e não será necessário refazer esse procedimento sempre que iniciar o Excel, essa guia ficará sempre habilitada até que o usuário desabilite.



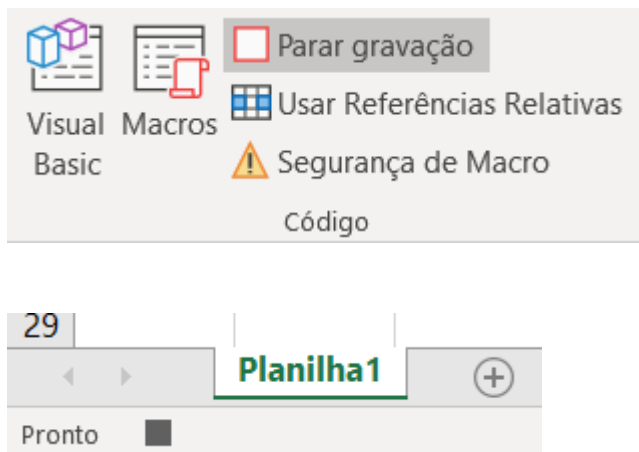
Nessa guia já podemos observar a opção **Gravar Macro**, é nesta opção que podemos gravar uma ou mais ações sem a necessidade de escrever um código para descrever essas ações. Nesta mesma aba temos a opção de **Macros** que é onde podemos utilizar as macros gravadas e o abrir o ambiente do **Visual Basic**.

Agora podemos iniciar o procedimento de gravar uma macro selecionando a opção **Gravar Macro**. Feito isso será aberta uma nova janela para colocar o nome da macro desejada, uma tecla de atalho se o usuário quiser e a descrição da macro (que também é opcional).



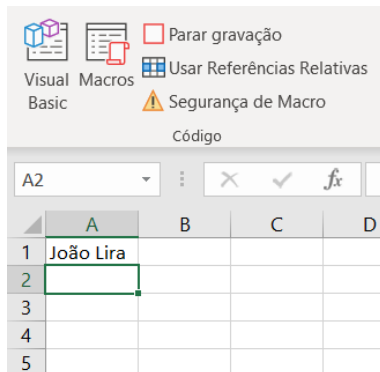
Assim que clicar em OK a macro já estará sendo gravada, então qualquer ação que o usuário fizer será gravada. Ações como selecionar uma célula, inserir uma fórmula, deletar um conteúdo da célula, formatar um conjunto de células, ou seja, todas as ações dentro do programa.

Repare também que a opção **Gravar Macro** é substituída pela opção **Parar Gravação**, que é um indicador de que a macro está sendo gravada. Outro indicador fica abaixo do nome da planilha ao lado do nome “Pronto”, aparece um quadrado (que é o símbolo de parar).

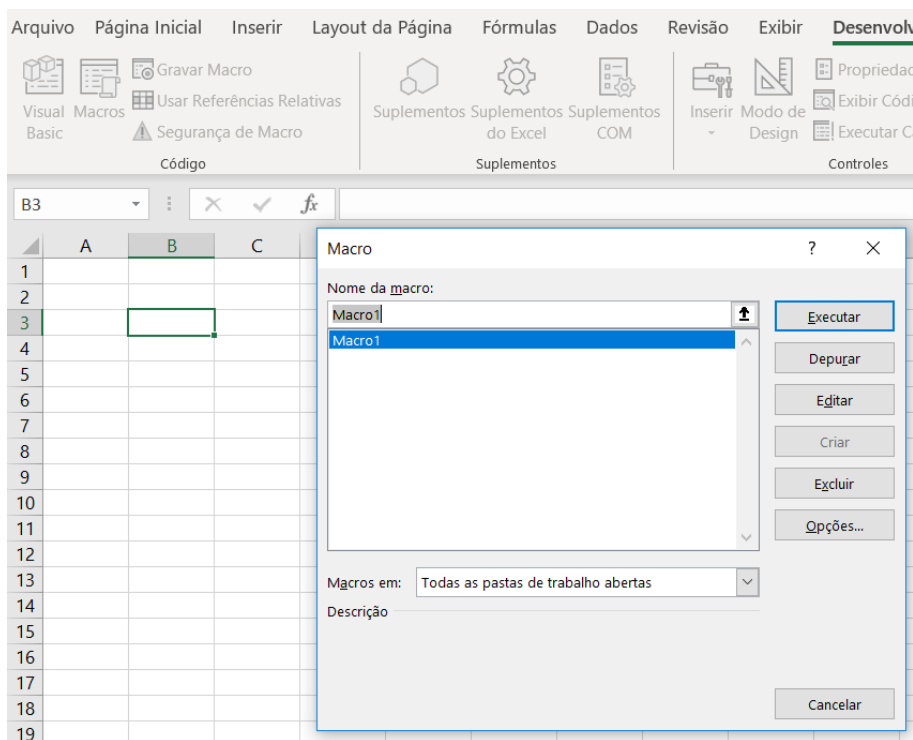


É importante lembrar que todas as ações gravadas serão transformadas em código, como foi dito anteriormente, as ações do Excel são representadas em códigos, então o usuário poderá fazer uma gravação e depois verificar qual o código gerou aquela determinada ação. Esse procedimento será utilizado no curso para que o usuário consiga saber como é a sintaxe (escrita) de certas ações dentro do VBA.

A primeira macro a ser gravada será a seguinte: Vamos **selecionar a célula A1, escrever um nome** e em seguida **pressionar a tecla Enter**. Feito isso podemos parar de gravar a macro.



Para verificar se a macro está funcionando corretamente basta deletar o nome escrito na célula, selecionar outra célula qualquer e selecionar a opção **Macros**. Feito isso será aberta uma janela contendo as macros gravadas, basta selecionar a macro que acabou de ser criado e pressionar a opção **Executar**.

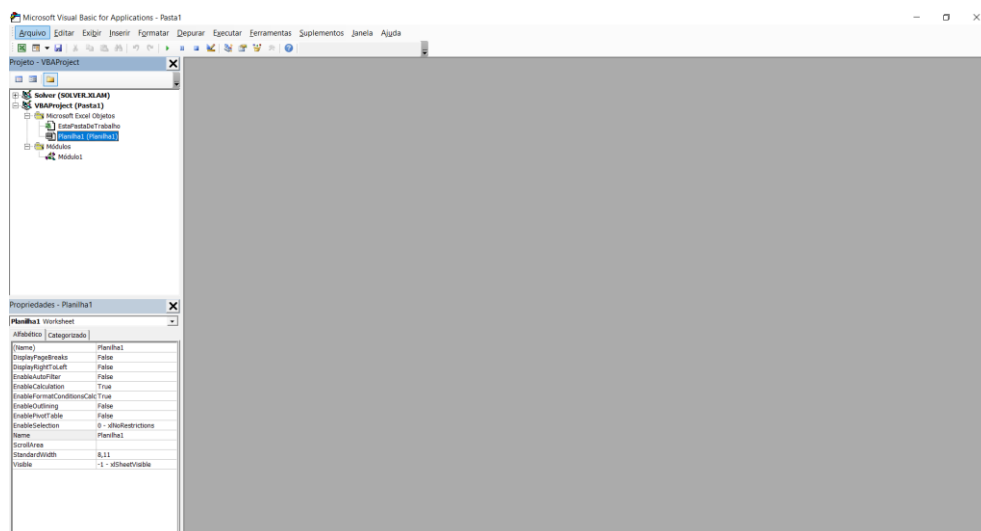


Ao executar é possível observar que o nome escrito aparece na célula A1 e a seleção se encontra na célula A2, pois foi pressionado **Enter** logo após digitar o nome.

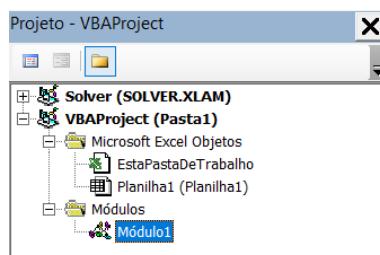
	A	B
1	João Lira	
2		
3		
4		

Essa é uma forma muito útil de se gravar uma macro quando não se tem conhecimento em VBA ou quando o usuário precisa descobrir como é aquela função dentro do VBA que é o que veremos agora. Vamos ver como o Excel escreveu o que foi gravado em código.

Para isso temos que acessar o ambiente VBA. Existem duas formas de acessar esse ambiente, a primeira é utilizando a opção já mostrada logo ao lado da opção macros que é o **Visual Basic**. A outra opção é utilizando o atalho do teclado **ALT + F11** (ou **ALT + Fn + F11**).



Essa é a tela inicial do ambiente VBA, como já gravamos uma macro podemos ir no menu à esquerda e com um clique duplo selecionar o **Módulo1**.



Feito isso será aberta uma “caixa de texto” que é onde se encontra o código da macro que foi gravada.

```
(Geral)
Sub Macro1()
'
' Macro1 Macro
'
'
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "João Lira"
    Range("A2").Select
End Sub
|
```

Portanto ao gravar uma macro é possível não só analisar o código como alterar ou corrigi-lo se for necessário. Apenas resumindo o que diz o código temos a **seleção da célula A1**, em seguida **atribuímos um nome** a essa célula e por fim **selecionamos a célula A2**.

Com o código é possível observar o que está sendo feito a cada execução, pois ao executar o código não é possível perceber todas as execuções que são feitas, pois são feitas bem rapidamente.

Portanto temos duas maneiras de criar uma macro, uma delas é gravando como foi feito e a outra é escrevendo manualmente o código, que veremos ao longo do curso e será bastante utilizado.

Ambiente VBA

Nesta parte vamos aprender um pouco sobre o ambiente do VBA, algumas informações para auxiliar esse primeiro contato com essa programação.

Na lateral esquerda temos um menu chamado **Projeto – VBAProject**. Esse menu mostra uma árvore de arquivos e pastas que estão disponíveis no momento.

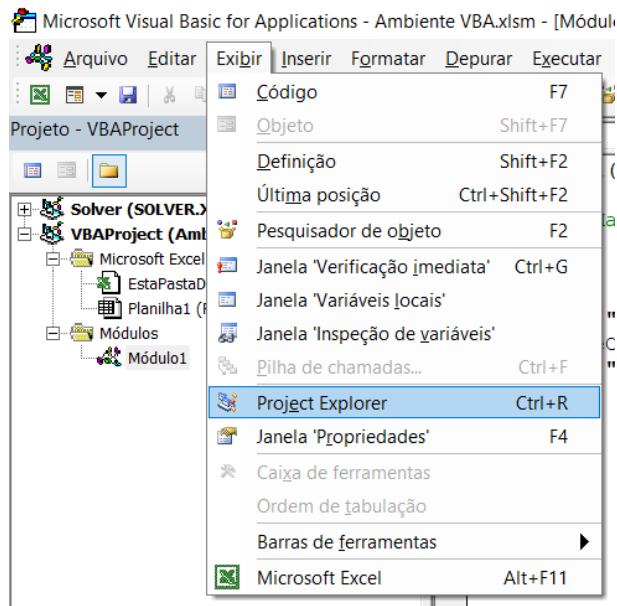


É possível observar que temos uma árvore para o **Ambiente VBA** que é o nome do nosso arquivo de Excel e dentro dele temos duas pastas. A primeira é a pasta de **objetos** que contém a pasta de trabalho e as planilhas existentes na mesma enquanto na segunda pasta temos os **módulos** que são as caixas de texto onde são escritas as macros.

Mais à frente do curso vamos ver as outras formas de onde podemos escrever as macros, mas inicialmente é o que temos neste menu lateral.

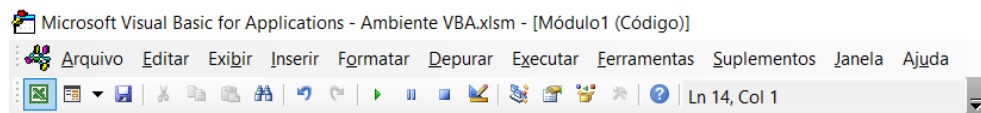
Caso não possua esse menu habilitado basta ir até a guia **Exibir** e selecionar a opção **Project Explorer** (ou utilizar o atalho **CTRL + R**).

Macros VBA para Excel Completo



Feito isso será possível habilitar esse menu que será útil para verificar o que temos dentro de cada arquivo, para sabermos onde estamos escrevendo cada código e até mesmo para navegar entre os códigos escritos.

Na parte superior do ambiente temos as guias básicas que todo programa oferece com diversas opções. O que será utilizado vai depender da aplicação e necessidade, portanto vamos apenas mostrar que existem essas guias e mais para frente vamos utilizar algumas informações dentro dessas guias.



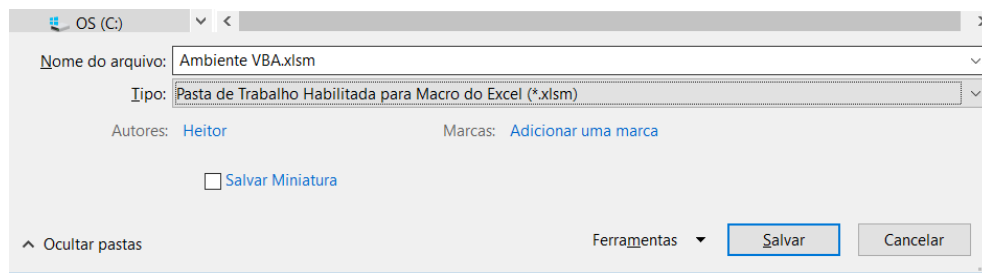
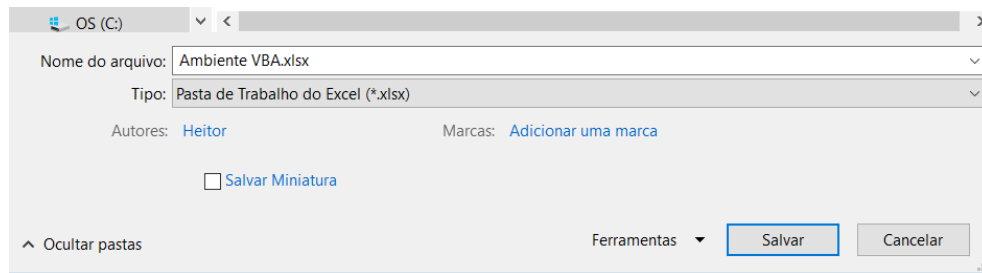
Logo abaixo das guias temos algumas opções úteis e importantes que serão utilizadas ao longo do curso, como **salvar**, **localizar**, **desfazer**, **executar a macro**, **pausar a execução**, **parar a execução**, entre outras opções.

Esse é o ambiente VBA, durante o curso será muito utilizado para a criação dos códigos, macros e formulários. Vamos sempre alternar entre o ambiente VBA e o ambiente do Excel para verificar a execução das macros e conferir se está tudo correto com o que estamos construindo.

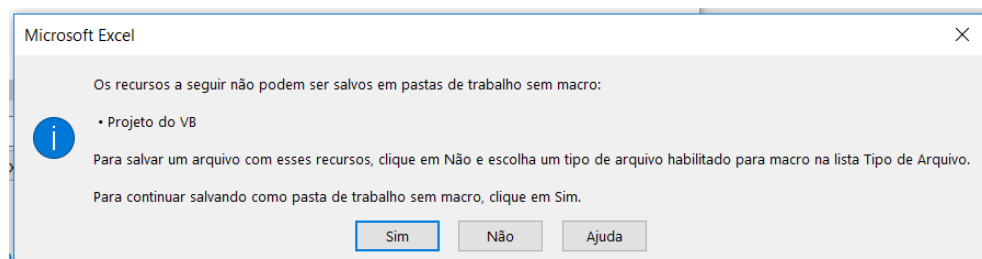
Resumindo, o ambiente VBA vai servir para escrevermos os códigos enquanto os resultados e ações serão presenciados dentro do ambiente Excel.

Um ponto **importante** é que agora que temos uma macro dentro do Excel a forma de salvar o arquivo será diferente, ao invés de escolhermos a opção **Pasta de Trabalho do Excel** vamos trocá-la pela opção **Pasta de Trabalho Habilitado para Macro do Excel**.

Macros VBA para Excel Completo



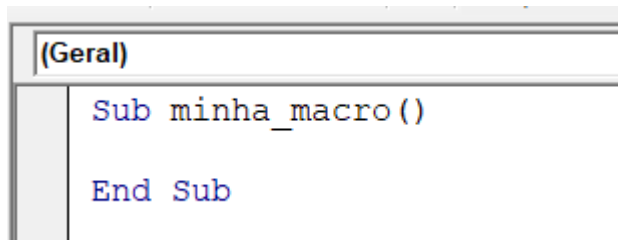
Essa mudança ocorre, pois, ao tentar salvar um arquivo que possui uma macro o Excel retorna um erro informando que o arquivo em questão possui uma macro. Caso o usuário opte por salvar mesmo assim **as macros** que estiverem dentro deste arquivo **serão excluídas** e consequentemente não irão funcionar na próxima vez que o arquivo for aberto.



Estrutura VBA e Variáveis

Nesta parte vamos aprender como é a estrutura do VBA, ou seja, o que precisamos colocar para que a nossa macro funcione e como funciona a utilização de variáveis dentro do código.

Os primeiros marcadores que vamos colocar é o **Sub** + o nome da macro e ao pressionar **ENTER** o código automaticamente já coloca o segundo marcador que é o **End Sub** e também coloca os parênteses.



Esses dois marcadores indicam o **início** e o **término** de uma macro. É importante ressaltar que o nome da macro não pode ser separado por espaço, por isso é utilizado o **underline** para separar o nome, o mesmo vale para as variáveis.

Outro marcador importante é utilizando uma **aspa simples**. Após utilizar e escrever algo após esse marcador o texto irá ficar com a **coloração verde** que indica que aquela linha é referente a um **comentário**. Os comentários são utilizados para facilitar o entendimento do código e não fazem parte do código propriamente dito, ou seja, essas linhas de comentário não são consideradas partes do código, são apenas informativos para auxiliar quem está escrevendo o código ou para quem está analisando o código.

```
Sub minha_macro()  
'Essa é minha primeira macro  
  
End Sub
```

Agora vamos aprender sobre as **variáveis** que são utilizadas nas programações em VBA, assim como qualquer programação.

As **variáveis são nomes que recebem valores**, sejam eles inteiros, decimais, sejam textos, entre outros valores. São nomes que facilitam a utilização de seus valores dentro do código, pois sempre que for feita a referência desse nome dentro do código ele vai retornar o valor que foi atribuído a ele.

```
Sub minha_macro()
'Essa é minha primeira macro
x = 10

End Sub
```

Neste exemplo temos a **variável x** e estamos atribuindo o **valor 10** a ela, portanto toda vez que colocarmos x dentro do código ele vai automaticamente considerar que o x tem valor igual a 10.

Outro ponto muito útil da utilização de variáveis é que se colocamos várias vezes essa variável no código e agora queremos mudar seu valor para outro, basta modificar onde fizemos a atribuição de valor que todos os valores serão alterados. Desta forma não é necessário alterar os valores um a um dentro de todo o código.

É de boa prática fazermos a declaração das variáveis com os tipos de dados que elas vão receber, ou seja, como temos que o número 10 é inteiro, podemos dizer ao código que essa variável irá receber um valor inteiro.

```
Sub minha_macro()
'Essa é minha primeira macro
Dim x As Integer

x = 10
y = "variavel"
End Sub
```

Utilizando o seguinte código: **Dim x As Integer**. Desta forma estamos dizendo que vamos **dimensionar** a **variável x** como **inteiro**.

Podemos fazer o mesmo procedimento para a **variável y**, que colocamos como um texto (como é texto deve estar entre **aspas duplas**). As variáveis quando são atribuídas com textos são chamadas de **strings**.

```
Sub minha_macro()
'Essa é minha primeira macro
Dim x As Integer
Dim y As String

x = 10
y = "variavel"
End Sub
```


Obs: O editor de texto do VBA corrige várias escritas para deixá-las de uma forma correta e quando não corrigir vai indicar que há algum erro naquela escrita para que o usuário possa verificar e corrigir.

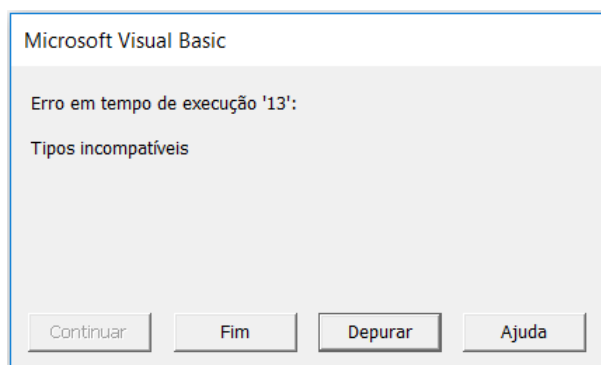
Foi então definido que a **variável y** é do tipo **string**, ou seja, ela receberá um texto ao invés de um número. Foi utilizado o mesmo código da **variável x**, no entanto trocamos a variável que está sendo dimensionada e o tipo dela.

É possível **obrigar o código a solicitar essa declaração** do tipo de variável para que quando o usuário tente colocar uma variável que não foi declarada ele gere um erro avisando sobre o ocorrido. Para isso basta escrever **Option Explicit** no início do código antes de qualquer escrita.

Option Explicit

```
Sub minha_macro()  
'Essa é minha primeira macro  
Dim x As Integer  
Dim y As String  
  
x = 10  
y = "variavel"  
End Sub
```

Um motivo importante para declarar o tipo da variável é para evitar erros futuros atribuindo tipos de valores errados para as variáveis. Então se colocarmos que **x = “variável”** o VBA irá retornar um erro.



Esse erro é para indicar que estamos atribuindo um tipo que não está compatível com o que foi declarado para aquela variável, portanto estamos atribuindo uma **string** para uma variável que foi declarada como **inteiro**.

Primeira Ferramenta – Hello World e MsgBox

Nesta parte vamos construir a primeira ferramenta em VBA e vamos dar uma introdução de como utilizar a ferramenta **MsgBox** (*Message Box*, do inglês que significa caixa de mensagem).

O **Hello World** nada mais é do que uma mensagem que significa “Olá Mundo”, que é utilizada em diversas linguagens de programação para dar início a escrita da linguagem e mostrar isso ao usuário. Portanto é exatamente o que vamos fazer, vamos utilizar a ferramenta MsgBox para mostrar essa mensagem ao usuário.

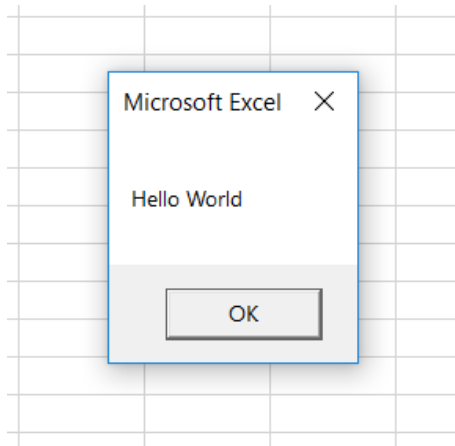
Option Explicit

```
Sub minha_macro()  
  'Essa é minha primeira macro  
  Dim x As Integer  
  Dim y As String  
  
  x = 10  
  y = "variavel"  
  
  MsgBox ("Hello World")  
  
End Sub
```

Vamos utilizar o código que estávamos utilizando na seção anterior e acrescentar a linha **MsgBox (“Hello World”)**.

É possível observar que ao começar a escrever essa função ela tem vários argumentos para preencher, no entanto só o primeiro é obrigatório e vamos utilizar somente esse por enquanto, as outras opções veremos mais adiante no curso.

Ao executar o código podemos observar que ocorre uma mudança para o ambiente do Excel e então aparece a nossa caixa de mensagem com o texto que colocamos entre parênteses e aspas duplas, que é exatamente **Hello World**.



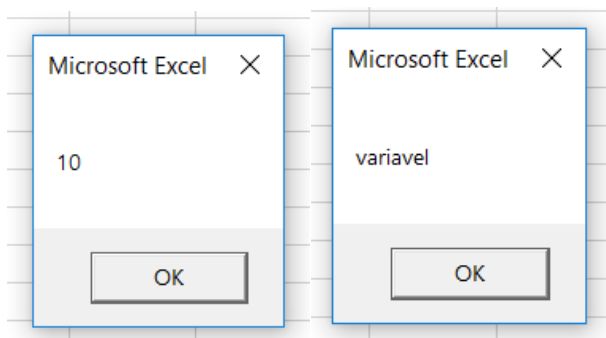
Sabendo desta primeira funcionalidade do MsgBox, podemos utilizar essa ferramenta para verificar se os valores das variáveis **x** e **y** estão sendo atribuídos corretamente.

```
MsgBox ("Hello World")
```

```
MsgBox (x)
```

```
MsgBox (y)
```

Basta acrescentar mais duas funções de MsgBox contendo as variáveis utilizadas neste código. Veja que neste caso não vamos utilizar as aspas duplas, pois queremos o **valor das variáveis e não o texto x e y**.



É possível observar que os valores atribuídos estão corretos e são mostrados ao usuário, um depois o outro. **É importante ressaltar que ao colocar essa função MsgBox dentro do código, ele só irá continuar para a próxima linha depois que o usuário pressionar OK**, ou seja, para que o MsgBox do valor de **x** seja mostrado é necessário pressionar **OK** no MsgBox do texto “**Hello World**” e o mesmo corre para que o valor de **y** seja mostrado.

Seção 3 – Fundamentos e Valores

Cells.Value e ActiveCell

Nesta parte vamos aprender uma função do VBA para alterar o valor de uma célula dentro do Excel, ou seja, através da programação vamos ser capazes de modificar o valor de uma célula dentro do ambiente Excel.

```
Sub alterarValorCelula()  
  
Dim x As Integer  
x = 15  
  
Cells(1, 1).Value = x  
  
End Sub
```

Neste caso temos o código **Cells(1, 1).Value = x**, isso quer dizer que vamos atribuir o valor de **x** ao valor da célula de linha 1 e coluna. Resumindo quer dizer que vamos pegar o valor da variável **x** que é igual a **15** e vamos modificar a propriedade valor da célula A1, pois é a célula composta pela linha 1 e coluna 1.

Então a função **Cells** pega uma célula de acordo com uma linha e uma coluna. Se colocarmos Cells(3,2) isso quer dizer que estamos pegando a célula que está na linha 3 e coluna 2, ou seja, essa será a célula B3.

A extensão **.Value** quer dizer que estamos na **propriedade valor** daquele **objeto** (que é uma estrutura do Excel, pode ser planilha, célula, aba...). Então neste caso estamos atribuindo o valor de uma variável ao valor daquela célula em questão.

Ao executar o código podemos observar que a célula A1 do Excel está preenchida com o valor da variável **x**.

	A	B
1	15	
2		
3		
4		
5		

Vamos agora utilizar outra função do VBA que é o **ActiveCell** para que possamos fazer novamente uma modificação em uma célula, no entanto essa modificação será em uma **célula ativa** (como o próprio nome já diz).

```
Sub alterarValorCelula()  
  
Dim x As Integer  
x = 15  
  
Cells(1, 1).Value = x  
  
ActiveCell.Value = "segunda-feira"  
  
End Sub
```

É possível observar que estamos utilizando a mesma propriedade de valor da célula, portanto estamos atribuindo o texto **“segunda-feira”** na célula que está ativa.

A célula ativa é a célula que está com o quadrado verde ao redor, que ocorre quando clicamos em uma célula. Para verificar como nosso código está funcionando vamos apagar o conteúdo da célula A1, vamos selecionar a célula B3 e vamos rodar o código novamente.

	A	B	C
1	15		
2			
3		segunda-feira	
4			
5			
6			

A célula A1 novamente foi preenchida com o valor da variável x porque fixamos que a célula a ser preenchida será a de linha 1 e coluna 1. No entanto a célula que está ativa no momento é a célula B3 e, portanto, a ela foi atribuído o texto escolhido.

Se selecionarmos a célula C5 por exemplo e executar nosso código novamente podemos ver que o texto será atribuído a essa célula, pois ela está ativa.

	A	B	C	D
1	15			
2				
3		segunda-feira		
4				
5			segunda-feira	
6				
7				

Como os valores não foram apagados a célula A1 continua com o valor, pois a célula está fixa para receber o valor da variável x, enquanto a célula C5 recebeu o valor do texto, pois agora é a célula que está ativa.

Então a principal diferença entre **Cells** e **ActiveCell** é que no primeiro é possível selecionar a célula informando a linha e coluna dela, enquanto a segunda função permite ao usuário utilizar a célula que está ativa naquele momento.

Essa foi apenas uma introdução a essas duas funções, vamos utilizá-las ao longo do curso para que os alunos consigam observar a diferença entre as duas e quando podemos utilizar cada uma dessas opções.

Operações Básicas e Executando com Botão

Nesta parte vamos ver algumas operações básicas dentro do VBA e como podemos executar as macros criadas com a utilização de um “botão”, ou seja, vamos criar um atalho para executar nossa macro sem a necessidade de ter que acessar algumas abas ou até mesmo ir até o ambiente VBA para executar a macro desejada.

Para isso é necessário escrever o código a ser utilizado que contém as operações básicas atribuídas a diferentes células do Excel.

```
Sub alterarValorCelula()

Dim x As Integer, y As Integer
Dim z As Double

x = 14
y = 5

'Colocando na linha 1 os valores das variáveis x e y
Cells(1, 1).Value = "x"      'Escrevendo a letra x
Cells(1, 2).Value = x        'Escrevendo o valor da variável x
Cells(2, 1).Value = "y"      'Escrevendo a letra y
Cells(2, 2).Value = y        'Escrevendo o valor da variável y

'Operações Básicas com x e y
Cells(3, 1).Value = "Soma"
Cells(3, 2).Value = x + y
Cells(4, 1).Value = "Subtração"
Cells(4, 2).Value = x - y
Cells(5, 1).Value = "Multiplicação"
Cells(5, 2).Value = x * y
Cells(6, 1).Value = "Divisão"
Cells(6, 2).Value = x / y

End Sub
```

Neste código é possível observar a declaração das variáveis x e y como inteiras assim como as atribuições de textos e das operações básicas em células diferentes.

Na segunda parte do código é possível ver quais são os símbolos responsáveis por cada uma das operações + (soma), - (subtração), * (multiplicação) e / (divisão). Para utilizar qualquer uma das operações basta seguir o modelo que foi feito no código, como se fosse uma conta na calculadora.

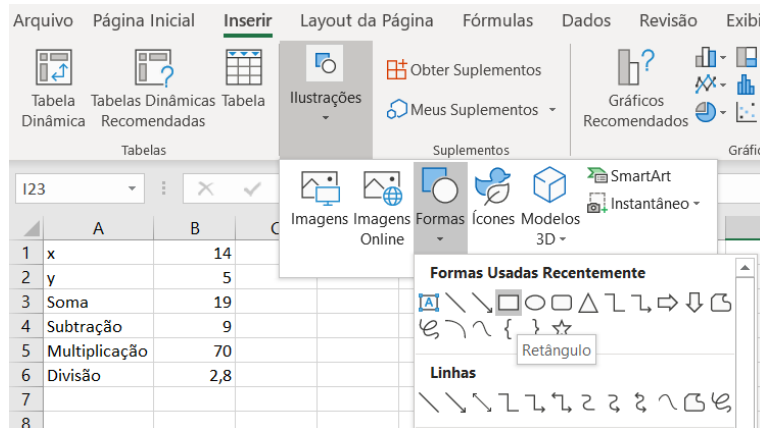
Ao executar o código acima teremos o seguinte resultado no ambiente Excel.

	A	B
1	x	14
2	y	5
3	Soma	19
4	Subtração	9
5	Multiplicação	70
6	Divisão	2,8
7		

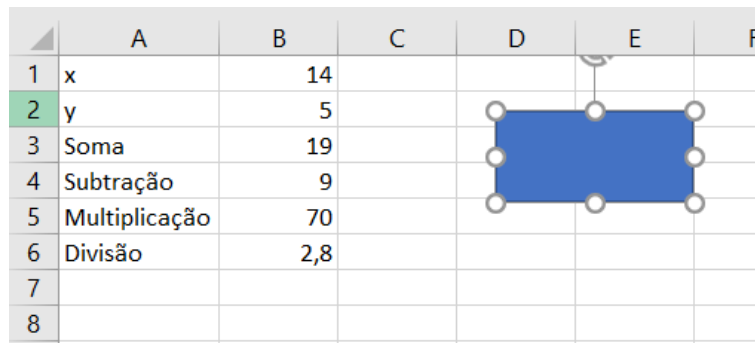
Todas as operações foram efetuadas e apenas os resultados foram atribuídos as células indicadas dentro do código, pois no código foi atribuído o resultado das operações aos valores das células.

Para a criação de um botão é necessário inserir uma forma ou imagem dentro do Excel. Neste caso vamos inserir uma forma.

Para isso basta ir até a guia **Inserir**, em seguida em **Ilustrações, Formas** e por fim selecionar a forma desejada.

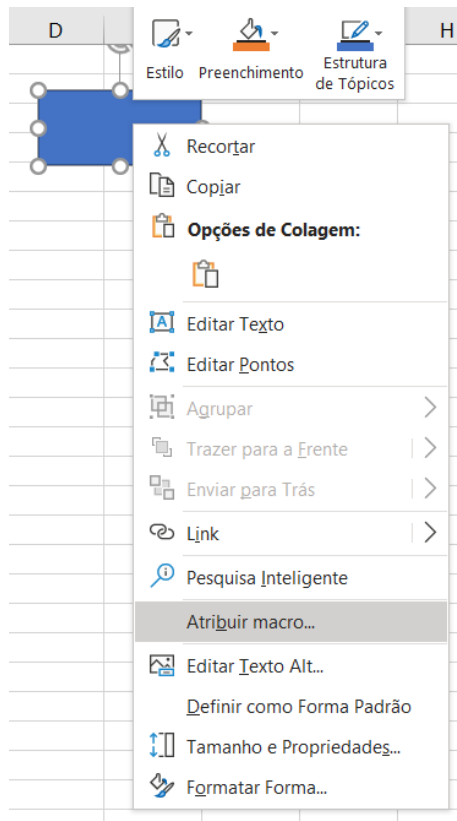


Ao selecionar essa opção basta clicar em algum ponto da planilha e arrastar para deixar a forma do tamanho desejado.

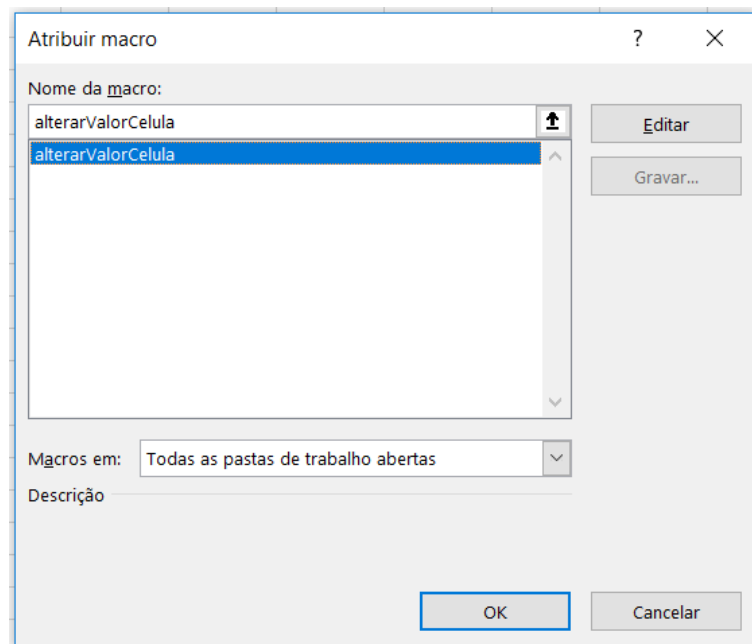


Feito isso será possível observar a forma criada do tamanho desejado. Para atribuir a macro a esse botão, basta clicar com o botão direito em cima da forma criada e selecionar a opção **Atribuir Macro**.

Macros VBA para Excel Completo



Ao selecionar essa opção será aberta a janela de macros, da mesma forma que é aberta para executar uma macro. Nesta janela basta escolher a macro para atribuir e pressionar **OK**.



Feito isso podemos deletar todos os dados que haviam sido escritos pela macro e executar a macro através do botão.

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Ao pressionar o retângulo que foi criado como “botão” é possível notar que a macro é executada e os dados do código aparecem dentro das células como foi visto anteriormente.

	A	B	C	D	E
1	x	14			
2	y	5			
3	Soma	19			
4	Subtração	9			
5	Multiplicação	70			
6	Divisão	2,8			
7					

Desta forma não há necessidade de ir até a parte de macros ou mesmo no ambiente VBA para executar a macro que acabou de ser atribuída. Basta clicar no botão que foi criado e a macro irá funcionar.

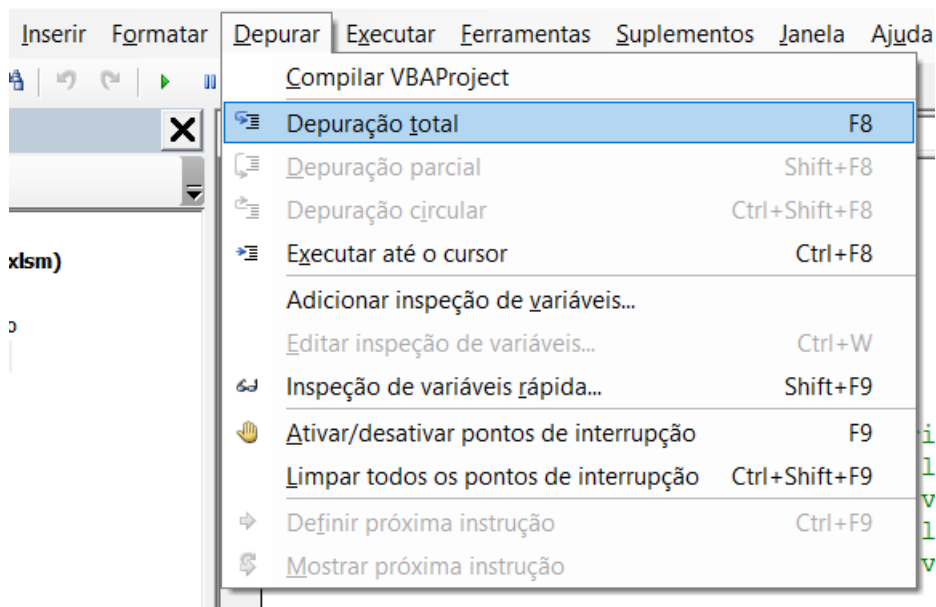
Debugando

Essa é uma nomenclatura utilizada para indicar o tratamento de erros dentro de um código, os erros são chamados de **bugs**, por isso da palavra debugar.

Nesta parte teremos algumas informações do que fazer e como fazer para debugar um código. Sempre que tiver algo de errado dentro do código o VBA irá sinalizar ao usuário, portanto é sempre bom ficar atendo a essas sinalizações.

O VBA tem seu código executado linha por linha, ou seja, ele só passa para a linha/execução seguinte quando terminar a que está sendo feita. Nos códigos que escrevemos até agora temos a impressão de que tudo está sendo escrito nas células ao mesmo tempo, no entanto não é assim que acontece.

Para verificarmos linha por linha como o código funciona podemos ir até a guia **Depurar** (dentro do ambiente VBA) e em seguida selecionar a opção **Depuração total** (atalho F8).

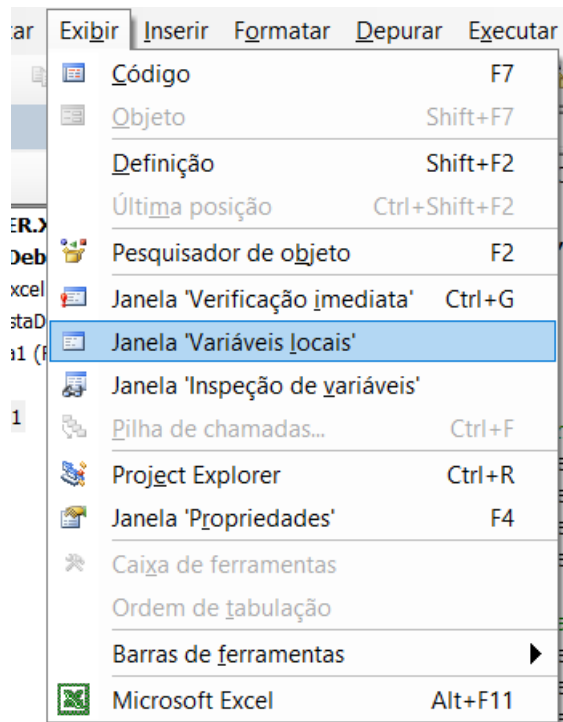


Ao utilizar essa opção o código será executado linha por linha à medida que o usuário pressiona a tecla **F8**.

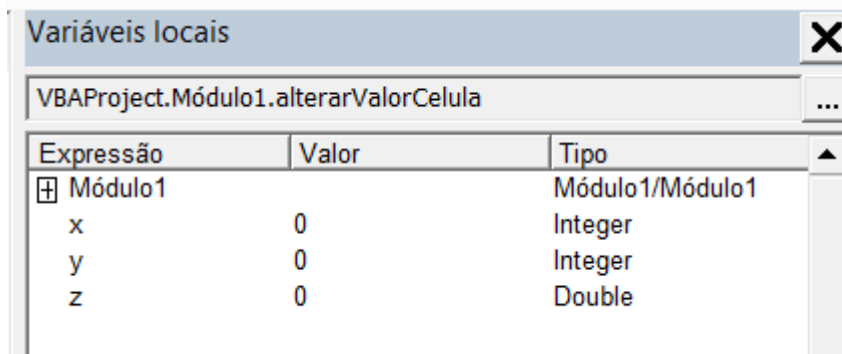
Para saber qual linha será executada o programa marca toda ela de amarelo, que é uma forma de sinalizar a linha.

```
Sub alterarValorCelula()  
  
Dim x As Integer, y As Integer  
Dim z As Double  
  
x = 14  
y = 5  
  
'Colocando na linha 1 os valores das variáveis x e y  
Cells(1, 1).Value = "x" 'Escrevendo a letra x  
Cells(1, 2).Value = x 'Escrevendo o valor da variável x  
Cells(2, 1).Value = "y" 'Escrevendo a letra y  
Cells(2, 2).Value = y 'Escrevendo o valor da variável y  
  
'Operações Básicas com x e y  
Cells(3, 1).Value = "Soma"  
Cells(3, 2).Value = x + y  
Cells(4, 1).Value = "Subtração"  
Cells(4, 2).Value = x - y  
Cells(5, 1).Value = "Multiplicação"  
Cells(5, 2).Value = x * y  
Cells(6, 1).Value = "Divisão"  
Cells(6, 2).Value = x / y  
  
End Sub
```

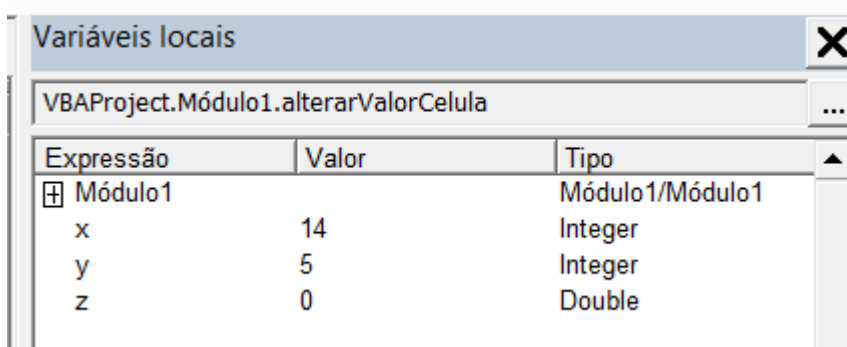
Para acompanhar o que acontece com as variáveis dentro do código quando estamos analisando linha por linha é interessante habilitar a janela de **Variáveis locais** que se encontra na guia **Exibir**.



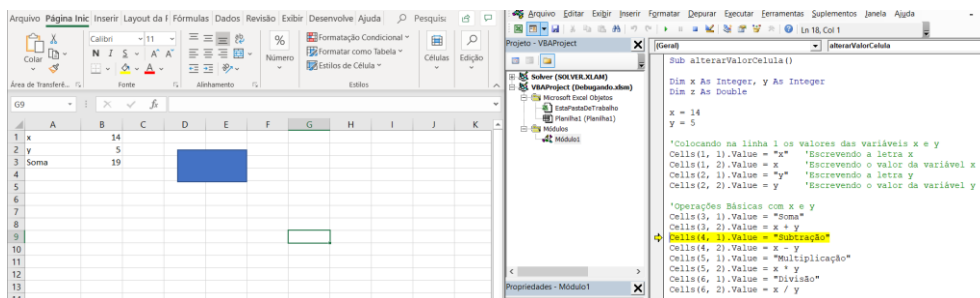
Habilitada essa janela é possível observar as mudanças das variáveis durante a análise do código utilizando a tecla F8.



Inicialmente todas as variáveis iniciam com o valor 0. Só depois das atribuições que elas recebem seus respectivos valores. Que ocorre dentro do código quando fazemos a declaração da variável e atribuímos seus valores.



Com essa ferramenta o usuário poderá executar linha por linha e verificar as mudanças ao longo do código, e também dentro do ambiente Excel.

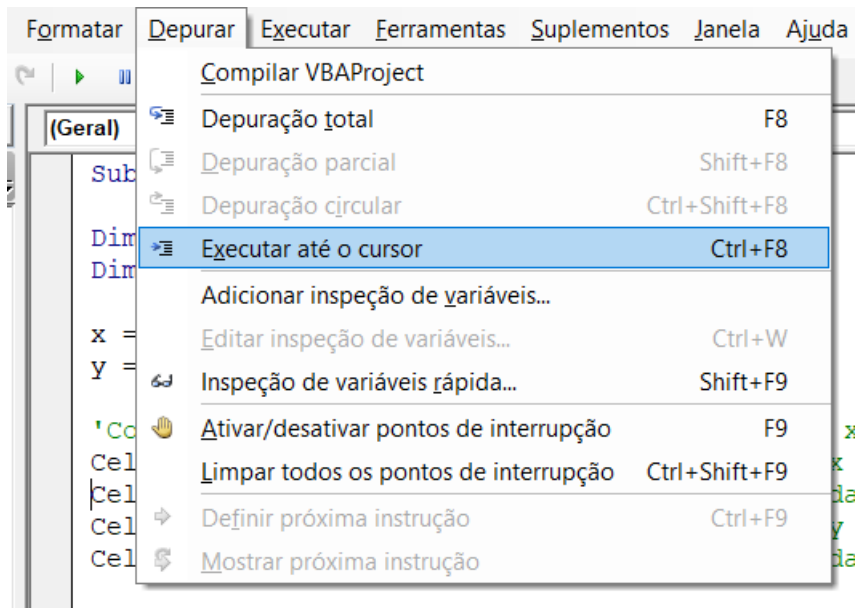


Neste caso a próxima linha a ser executada é a linha em que será atribuído o texto “**subtração**” dentro da célula A4, no entanto é possível observar que as linhas anteriores já foram escritas. Dividindo a tela é mais fácil observar que a medida em que pressiona a tecla **F8** os dados vão sendo preenchidos na tabela.

Existe uma outra forma de fazer essa execução sem que seja linha por linha, é uma execução até um ponto informado. Esse ponto pode ser informado clicando na linha desejada, neste caso para que o cursor do texto fique na linha.

```
'Colocando na linha 1 os
Cells(1, 1).Value = "x"
Cells(1, 2).Value = x
Cells(2, 1).Value = "y"
Cells(2, 2).Value = y
```

É possível observar que temos o cursor no início da linha do código que colocar o valor de x na célula.



Feito isso o código será executado até o ponto informado. Isso é útil para analisar códigos maiores onde o usuário quer fazer uma análise detalhada na região selecionada, para que não precise executar linha por linha até chegar neste ponto.

```
'Colocando na linha 1 os v
Cells(1, 1).Value = "x"
Cells(1, 2).Value = x
Cells(2, 1).Value = "y"
Cells(2, 2).Value = y
```

Uma outra forma de executar o código até pontos específicos é selecionando as linhas desejadas onde está o símbolo da seta amarela na figura acima. Desta forma é possível selecionar vários pontos de análise e quando executar o código ele automaticamente vai parar nestes pontos.

```
(Geral)
Sub alterarValorCelula()

    Dim x As Integer, y As Integer
    Dim z As Double

    x = 14
    y = 5

    'Colocando na linha 1 os valores das variáveis x e y
    Cells(1, 1).Value = "x"      'Escrevendo a letra x
    Cells(1, 2).Value = x        'Escrevendo o valor da variável x
    Cells(2, 1).Value = "y"      'Escrevendo a letra y
    Cells(2, 2).Value = y        'Escrevendo o valor da variável y

    'Operações Básicas com x e y
    Cells(3, 1).Value = "Soma"
    Cells(3, 2).Value = x + y
    Cells(4, 1).Value = "Subtração"
    Cells(4, 2).Value = x - y
    Cells(5, 1).Value = "Multiplicação"
    Cells(5, 2).Value = x * y
    Cells(6, 1).Value = "Divisão"
    Cells(6, 2).Value = x / y

End Sub
```

Desta forma fica mais fácil fazer análises no código sabendo que as outras partes estão funcionando corretamente.

Outra informação importante é que sempre que ocorre um erro na execução de um código o VBA geralmente aponta a linha em que está acontecendo esse erro. Neste caso fica mais fácil para o usuário identificar o erro. Um dos erros mais comuns, principalmente no início são erros de digitação, ou seja, escrever uma função errada, ou uma variável ou algo do gênero.

Segunda Ferramenta – Planilha de Cálculos Automáticos

Nesta parte vamos criar uma ferramenta para fazer cálculos utilizando algumas operações, no entanto esse cálculo será automático utilizando o VBA. Para isso vamos utilizar a seguinte tabela.

	A	B	C	D	E	F	G	H	I	J	K	L
1		Soma		Diferença		Produto		Divisao		Potência		Raiz
2	Valor 1		Valor 1		Valor 1		Valor 1		Valor 1		Valor 1	
3	Valor 2		Valor 2		Valor 2		Valor 2		Valor Potência		Valor Raiz	
4	Valor 3		Valor 3		Valor 3							
5	Valor 4		Valor 4		Valor 4							
6	Valor 5		Valor 5		Valor 5							
7	Valor 6		Valor 6		Valor 6							
8												
9												
10												

Vamos criar nossa ferramenta para executar as operações desta tabela, portanto o código criado irá executar **6 operações** e no final vamos atribuir o status de concluído para verificar o fim do cálculo.

O código a ser escrito é o seguinte:

```
Option Explicit
Sub operacoes()

Dim resultado_soma As Double, resultado_diferenca As Double, resultado_produto As Double
Dim resultado_divisao As Double, resultado_potencia As Double, resultado_raiz As Double

resultado_soma = Cells(2, 2).Value + Cells(3, 2).Value + Cells(4, 2).Value + Cells(5, 2).Value + Cells(6, 2).Value + Cells(7, 2).Value
Cells(8, 2).Value = resultado_soma

resultado_diferenca = Cells(3, 4).Value - Cells(4, 4).Value - Cells(5, 4).Value - Cells(6, 4).Value - Cells(7, 4).Value
Cells(8, 4).Value = resultado_diferenca

resultado_produto = Cells(2, 6).Value * Cells(3, 6).Value * Cells(4, 6).Value * Cells(5, 6).Value * Cells(6, 6).Value * Cells(7, 6).Value
Cells(8, 6).Value = resultado_produto

resultado_divisao = Cells(2, 8).Value / Cells(3, 8).Value
Cells(8, 8).Value = resultado_divisao

resultado_potencia = Cells(2, 10).Value ^ Cells(3, 10).Value
Cells(8, 10).Value = resultado_potencia

resultado_raiz = Cells(2, 12).Value ^ (1 / Cells(3, 12).Value)
Cells(8, 12).Value = resultado_raiz

Cells(10, 3).Value = "Concluído"
End Sub
```

O código é bem simples, praticamente todas as nomenclaturas, funções e símbolos foram utilizados. O único que não havia sido mostrado é o **^** que significa a **potência em linguagem de código**, ou seja, temos que **2 ^ 2 = 4**, pois 2 elevado a segunda potência é igual a 4. Para a operação inversa, que é a raiz, basta invertermos o segundo termo, portanto teremos a seguinte expressão **2 ^ 1/2 = 1**. É uma propriedade matemática.

Abaixo é possível observar a tabela já com os dados a serem calculados e os respectivos locais onde ficarão os resultados.

Macros VBA para Excel Completo

	A	B	C	D	E	F	G	H	I	J	K	L
1		Soma		Diferença		Produto		Divisão		Potência		Raiz
2	Valor 1	2	Valor 1	10	Valor 1	2	Valor 1	25	Valor 1	12	Valor 1	49
3	Valor 2	3	Valor 2	5	Valor 2	2	Valor 2	3	Valor Potência	4	Valor Raiz	2
4	Valor 3	4	Valor 3	6	Valor 3	2						
5	Valor 4	5	Valor 4	7	Valor 4	2						
6	Valor 5	6	Valor 5	8	Valor 5	2						
7	Valor 6	7	Valor 6	9	Valor 6	2						
8	Soma		Diferença		Produto		Divisão		Potência		Raiz	
9												
10			Status:									
11												

Agora basta executar o código para observar o resultado utilizando os respectivos valores para cada uma das operações.

	A	B	C	D	E	F	G	H	I	J	K	L
1		Soma		Diferença		Produto		Divisão		Potência		Raiz
2	Valor 1	2	Valor 1	10	Valor 1	2	Valor 1	25	Valor 1	12	Valor 1	49
3	Valor 2	3	Valor 2	5	Valor 2	2	Valor 2	3	Valor Potência	4	Valor Raiz	2
4	Valor 3	4	Valor 3	6	Valor 3	2						
5	Valor 4	5	Valor 4	7	Valor 4	2						
6	Valor 5	6	Valor 5	8	Valor 5	2						
7	Valor 6	7	Valor 6	9	Valor 6	2						
8	Soma	27	Diferença	-25	Produto	64	Divisão	8,333333	Potência	20736	Raiz	7
9												
10			Status:	Concluído								
11												

Os resultados foram corretamente colocados abaixo de cada uma das colunas (poderá ser feito um teste utilizando as fórmulas do ambiente Excel para verificar). E é possível ver o status de concluído.

Para que não seja necessário ficar sempre executando a macro pelo menu ou pelo ambiente VBA podemos criar um botão para essa execução.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		Soma		Diferença		Produto		Divisão		Potência		Raiz				
2	Valor 1	2	Valor 1	10	Valor 1	2	Valor 1	25	Valor 1	12	Valor 1	49				
3	Valor 2	3	Valor 2	5	Valor 2	2	Valor 2	3	Valor Potência	4	Valor Raiz	2				
4	Valor 3	4	Valor 3	6	Valor 3	2										
5	Valor 4	5	Valor 4	7	Valor 4	2										
6	Valor 5	6	Valor 5	8	Valor 5	2										
7	Valor 6	7	Valor 6	9	Valor 6	2										
8	Soma	27	Diferença	-25	Produto	64	Divisão	8,333333	Potência	20736	Raiz	7				
9																
10			Status:	Concluído												
11																

Calcular

Portanto sempre que for feita alguma alteração nos números basta clicar no botão que a macro será executada e os cálculos serão refeitos.

Esse foi um código simples, no entanto bem útil caso seja necessário efetuar cálculos de forma automática sem que seja necessário utilizar fórmulas dentro do ambiente Excel, apenas utilizando o código em VBA.

Seção 4 – Fundamentos Range e Propriedades

Objeto Range

Nesta parte vamos aprender o que é o objeto **Range** e como ele facilita a seleção de células ou conjunto de células, ou seja, é uma outra forma de fazer uma seleção que em alguns casos pode ser mais útil do que a opção **Cells**.

Option Explicit

Sub testeRange()

Range("B5").Value = 3

End Sub

Com o **range** é possível escrever a referência da célula da mesma forma que é utilizada no ambiente Excel, ou seja, não há a necessidade de escrever a linha e coluna da célula que está querendo.

Além da opção de colocar apenas uma célula, com o range é possível colocar um conjunto de células.

Range("A5", "A8").Value = 7

Range("C1:D4").Value = 10

Temos duas formas para isso, **separando por vírgulas**, ou simplesmente colocando da mesma forma que é utilizada no ambiente Excel, **utilizando dois pontos entre as células**. As duas formas têm o mesmo resultado de pegar um intervalo de células.

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

Desta forma é mais fácil para algumas aplicações o uso do **range** ao invés de utilizarmos a opção **Cells**.

Além de podermos fazer essa seleção de uma ou mais células é possível declaramos uma **variável do tipo range**, ou seja, ela vai fazer referência a todo um conjunto de células. Para que isso funcione precisamos declarar a variável como range e ao definir o range é necessário colocar a palavra **set** antes da variável.

Option Explicit

```
Sub testeRange()

Dim range1 As Range

Set range1 = Range("B2:E7")

range1.Value = "teste"

End Sub
```

Primeiro é declarada a variável como **range**, em seguida será “setado” esse range, que são as células que o compõem, e por fim, é dado um valor a esse conjunto de células.

	A	B	C	D	E
1					
2		teste	teste	teste	teste
3		teste	teste	teste	teste
4		teste	teste	teste	teste
5		teste	teste	teste	teste
6		teste	teste	teste	teste
7		teste	teste	teste	teste
8					
9					

Desta forma é possível atribuir um valor a esse intervalo apenas atribuindo este valor ao intervalo ou variável que corresponda a ele.

Outra informação importante em relação ao range é que podemos utilizar a opção **cells** dentro dele, ou seja, como é um conjunto de células podemos referenciar essas células através do **cells**.

```

Dim range1 As Range

Set range1 = Range("B2:E7")

range1.Value = "teste"
range1.Cells(3, 2).Value = "Célula"

```

É possível observar que estamos utilizando a opção `cells` dentro da variável **range1**, portanto ao escolhermos a **linha 3** e **coluna 2** essa célula escolhida não será da planilha toda, mas apenas do conjunto de células do `range1`. Desta forma podemos trabalhar com o `cells` dentro de um `range` para referenciar as células lá dentro.

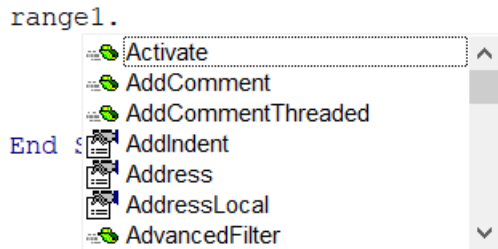
	A	B	C	D	E
1					
2		teste	teste	teste	teste
3		teste	teste	teste	teste
4		teste	Célula	teste	teste
5		teste	teste	teste	teste
6		teste	teste	teste	teste
7		teste	teste	teste	teste
8					

É possível observar que fosse a linha 3 e coluna 2 da planilha com um todo, a **célula B3** que deveria ter sido alterado e não foi esse o caso, pois a referência para esse `cells` é o `range`.

Portanto ao escrever desta forma **range1.Cells**, o `cells` passa a ser uma propriedade desse intervalo de células.

Propriedades do Range e Aprendendo com o Gravar Macros

Para acessar as propriedades de um objeto qualquer dentro do VBA, basta escrever o nome do objeto e colocar o ponto logo em seguida. Feito isso será disponibilizada uma lista com as diversas propriedades e métodos daquele objeto.



Portanto é desta forma que as propriedades de um objeto serão acessadas. O **Value** é uma propriedade que já foi utilizada nos exemplos e se refere ao valor daquele objeto. Existem diversas propriedades e a utilização delas vai depender de cada aplicação do código.

Foi mostrado no início da apostila a forma utilizada para gravar macros, essa forma será muito utilizada, pois ajuda não só a lembrar como a descobrir como algumas ações são executadas dentro do VBA, ou seja, se o usuário não sabe como escrever um código que executa uma determinada ação, basta ele gravar uma macro no ambiente Excel fazendo exatamente aquela ação.

Feito isso é possível observar o código desse procedimento no ambiente VBA e então analisar o código para observar o que cada parte está fazendo.

Vamos iniciar gravando uma macro bem simples. Essa macro consiste em selecionar a **célula B3**, **modificar o valor da célula para 29** e em seguida pressionar **enter**.

```
Sub Macro1 ()
'
' Macro1 Macro
'
'
Range("B3").Select
ActiveCell.FormulaR1C1 = "29"
Range("B4").Select
End Sub
```

Portanto ao gravar essa macro simples o resultado em VBA é exatamente esse. Inicialmente temos a seleção da **célula B3** utilizando a propriedade **select**, em seguida temos o **ActiveCell** que faz referência a célula ativa naquele momento e ainda na mesma linha é utilizada uma propriedade dessa célula que

é **FormulaR1C1** que é para modificar a fórmula dentro do Excel (será explicado com mais detalhes posteriormente no curso).

Utilizando essa propriedade é possível observar que o valor está entre aspas duplas, neste caso o Excel vai converter esse valor para numérico posteriormente. E por fim ao pressionar **enter** temos a célula logo abaixo sendo selecionada também pela propriedade **select**.

A próxima macro a ser gravada também é simples, será apenas uma macro gravando a ação de **deletar** esse valor que foi escrito na **célula B3**.

```
Sub Macro2()  
'  
' Macro2 Macro  
'  
  
    Range("B3").Select  
    Selection.ClearContents  
End Sub
```

Novamente podemos observar a propriedade **select** para selecionar a **célula B3** e na linha abaixo é possível observar a escrita **Selection.ClearContents**, ou seja, limpar o conteúdo da seleção. Essa seleção se refere à célula que está selecionada no momento, enquanto temos a propriedade logo em seguida que é para limpar o conteúdo dela.

O **selection** é utilizado para intervalos de seleção, então por padrão o Excel grava a macro desta forma ao invés de colocar o ActiveCell, que caso fosse um intervalo de células ele iria fazer a ação apenas na primeira célula do intervalo.

Portanto com cada gravação é possível aprender como são os códigos de cada ação para que possam ser utilizados em outros códigos, ou seja, se não sabemos um código em VBA, podemos simplesmente gravar e ver como o Excel retorna essa ação em forma de código para aprendermos.

Vamos agora gravar outra macro, desta vez vamos selecionar da **célula D2 até a F6**, mudar a **cor de preenchimento para amarelo**, a **cor da fonte para vermelho** e colocar tudo em **negrito**.

```

Sub Macro3()
'
' Macro3 Macro
'
'
    Range("D2:F6").Select
    With Selection.Font
        .Color = -16776961
        .TintAndShade = 0
    End With
    With Selection.Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .Color = 65535
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
    Selection.Font.Bold = True
End Sub

```

Com esse código é possível observar e retirar algumas informações para acrescentar em um código que esteja sendo criado. No primeiro caso estamos alterando a cor da fonte, então temos duas informações importantes, que são: **Selection.Font**, que é a seleção da fonte e **Color = -16776961** que é a atribuição da cor vermelha para a célula em questão.

Essas opções podem sempre ser exploradas e modificadas pelo usuário para ver os diferentes resultados que trazem ao executar a macro.

No segundo caso temos a alteração do preenchimento da célula e novamente temos duas informações importantes, que são: **Selection.Interior**, que é a seleção do interior da célula e **Color = 65535** que é a atribuição da cor amarela a célula em questão.

Por fim temos a informação de **Selection.Font.Bold = True**, ou seja, temos novamente a seleção da fonte, mas neste caso temos a propriedade **bold** que é referente a negrito e temos que essa atribuição está sendo dada como **true** (verdadeira).

Portanto o usuário pode utilizar essas informações mais importantes para fazer essas mesmas alterações que foram feitas com a gravação da macro, ou seja, foi possível aprender como mudar a cor da fonte, a cor de preenchimento e como colocar o conteúdo da célula em negrito.

Tudo isso foi possível graças a gravação de macro, então o usuário não precisa saber todas as informações e propriedades do VBA, quando tiver dúvidas basta gravar a macro com o que deseja saber para verificar o código e tirar as informações importantes do código gerado.

É importante ressaltar que as propriedades observadas podem ser utilizadas para um range e neste caso serão atribuídas a todas as células do range selecionado, ou seja, caso seja necessário adicionar essas propriedades mais de uma vez é mais fácil utilizar o range do que fazer célula por célula.

Propriedade Offset

Essa é uma propriedade que permite um deslocamento em linha e/ou coluna de uma célula ou um range.

```
Sub testeRange()  
  
Dim range1 As Range  
  
Set range1 = Range("A1:C5")  
  
End Sub
```

Neste código estamos apenas inserindo a variável **range1** e atribuindo o range da **célula A1 até C5**.

Ao utilizarmos a propriedade offset vamos poder fazer um deslocamento desse range. Vamos atribuir valores para o range antes e depois do offset para que seja possível observar esse deslocamento.

```
Sub testeRange()  
  
Dim range1 As Range  
  
Set range1 = Range("A1:C5")  
range1.Value = 0  
  
range1.Offset(7, 0).Value = 1  
  
End Sub
```

Com esse código temos a atribuição do **valor 0** ao range que vai da **célula A1 até C5** e temos a atribuição do **valor 1** ao deslocamento desse mesmo range.

A propriedade offset tem dois argumentos, o **primeiro** é referente ao **deslocamento em linha** e o **segundo** é referente ao **deslocamento em coluna**. Esses argumentos permitem tanto valor positivo quanto negativo.

- Valor positivo para linha – o range será deslocado para baixo;
- Valor positivo para coluna – o range será deslocado para direita;
- Valor negativo para linha – o range será deslocado para cima;
- Valor negativo para coluna – o range será deslocado para esquerda;

Sabendo disso podemos observar que o offset feito é de linha e está com o número 7 positivo, portanto o range será deslocado 7 células para baixo.

	A	B	C
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6			
7			
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	1	1	1
13			

As células com **número 0** representam o range inicial, enquanto as células com o **número 1** representam esse mesmo range deslocado em **7 linhas**.

Esse offset pode ser feito para coluna, e também com a utilização dos dois em conjunto.

```
Sub testeRange()

Dim range1 As Range

Set range1 = Range("A1:C5")
range1.Value = 0

range1.Offset(7, 0).Value = 1

range1.Offset(0, 7).Value = 2

range1.Offset(7, 7).Value = 3

End Sub
```

Portanto o deslocamento pode ser feito com a combinação de deslocamentos entre linhas e colunas.

Macros VBA para Excel Completo

	A	B	C	D	E	F	G	H	I	J
1	0	0	0					2	2	2
2	0	0	0					2	2	2
3	0	0	0					2	2	2
4	0	0	0					2	2	2
5	0	0	0					2	2	2
6										
7										
8	1	1	1					3	3	3
9	1	1	1					3	3	3
10	1	1	1					3	3	3
11	1	1	1					3	3	3
12	1	1	1					3	3	3
13										

Temos então as possibilidades de deslocamento entre linha e coluna para os valores positivos. Para os valores negativos basta considerar que o range selecionado são as células que estão com o número 3.

Essa propriedade é importante para que o usuário não tenha que criar variáveis novas para pegar um valor que esteja na coluna ao lado ou na linha acima/abaixo da escolhida. Um exemplo que será utilizado durante o curso é quando vamos colocar informações dentro de uma tabela, então se temos 4 colunas para preencher vamos precisar apenas do primeiro range, pois os outros podemos apenas deslocar esse primeiro e colocar os diferentes valores que são referentes a cada coluna.

Seção 5 – Estruturas do VBA

Essa seção tem o objetivo de mostrar as estruturas do VBA que são utilizadas em diversas linguagens de programação, portanto a lógica é a mesma o que pode mudar é como essas estruturas são escritas. Essas estruturas são fundamentais para a criação da lógica de um código.

O que aprendemos até o momento são as partes mais básicas de colocar valor em uma célula e fazer deslocamentos de células ou grupo de células. Nesta parte vamos aprender a colocar a lógica no programa para que ele funcione de forma mais efetiva e inteligente.

Estrutura If – Else

Essa é uma estrutura condicional, se a condição for verdadeira o programa irá executar ação dentro da estrutura, caso contrário não vai entrar na estrutura ou vai entrar na parte do complemento dela. Funciona como a **função SE** do Excel.

```
Option Explicit  
  
Sub funcaoIF()  
  
Dim var As Double  
  
If Range("A2").Value <> "" Then  
    var = 1  
    MsgBox (var)  
End If  
  
End Sub
```

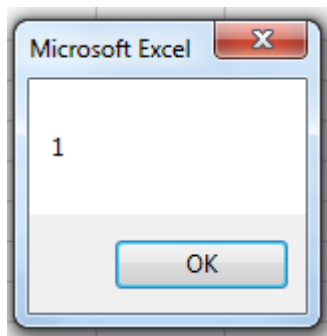
Essa é a estrutura básica da **estrutura If**. Colocamos o **If** em seguida a **condição**, que neste caso é que o valor da célula A2 seja diferente de vazio e temos o **then** que significa **então**, ou seja, caso a condição seja verdadeira o programa vai executar o que temos dentro da estrutura If.

É possível observar que essa função tem um indicador de término que é o **End If**, portanto se a condição for verdadeira tudo que está entre o **If** e **End If** será executado, caso contrário nada será executado.

Vamos então analisar a tabela que temos para entender como o código vai se comportar.

	A	B
1	Código	Produto Descrição
2	AB1	Produto AB1
3		
4		
5		
6		
7	AB2	Produto AB2
8		
9		
10	AB3	Produto AB3
11	AB4	Produto AB4
12	AB5	Produto AB5
13		

Neste caso vamos analisar se a célula A2 está vazia para que posteriormente possamos preencher esses espaços vazios com o texto que está mais acima, ou seja, as células de A3 até A6 devem ser preenchidas com AB1, pois na célula A7 já temos um valor diferente.



Ao executar o código temos que o **MsgBox** retorna o número 1, isso quer dizer que a **condição do If foi satisfeita**, ou seja, o valor da célula A2 é diferente de vazio, pois tem algo escrito nela. Feito isso foi atribuído o **valor 1** a variável **var** e em seguida esse valor foi mostrado pelo MsgBox.

Caso a condição seja falsa o que está dentro da **função If** não será executado e o código vai continuar normalmente após o **End If**.

A função If tem um **complemento** que é a função **Else**, ou seja, se a condição do If não for verdadeira ela vai para dentro da função Else, portanto é uma função que faz um complemento para executarmos um código também para o caso de ser falso. Exatamente como ocorre na **função SE** do Excel, temos um **valor para verdadeiro** e também temos um **valor para falso**.

Option Explicit

```
Sub funcaoIF()  
  
Dim var As String  
  
If Range("A2").Value <> "" Then  
    var = Range("A2").Value  
Else  
    var = ""  
End If  
  
End Sub
```

Neste caso podemos observar o complemento **Else** dentro do código. Aqui foi feita uma mudança do tipo de variável para **string** (que é o tipo utilizado para textos) e os códigos a serem executados nas funções.

Dentro do If temos que se a condição for verdadeira a variável var recebe o valor da **célula A2**, caso contrário vai para a função **Else** e recebe vazio.

É importante ressaltar que o código vai executar **apenas uma das duas opções** neste caso ou If ou Else, nunca irá executar as duas. E após executar o que está dentro de uma delas vai diretamente para o End If para dar continuidade ao código.

Portanto essas duas funções são muito úteis para fazer esse tipo de verificação e dependendo da condição será executado um código diferente, desta forma podemos abranger as condições de verdadeiro e falso da mesma forma que é feito dentro do Excel, no entanto dentro do VBA podemos escrever vários códigos a serem executados para cada um dos resultados de verdadeiro ou falso.

Estrutura For e For Each

Essa é uma estrutura de **repetição** (conhecida como **laço de repetição** ou **loop**), ou seja, ela tem a função de repetir uma ação por um número determinado de vezes para evitar com que o usuário tenha que escrever uma mesma ação inúmeras vezes e consequentemente diminuir o tamanho do código.

Então se quisermos escrever em um MsgBox os números de 1 a 10, vamos precisar apenas utilizar a **estrutura for** para que o seja repetida uma ação que está vinculada com a variável de contagem da função.

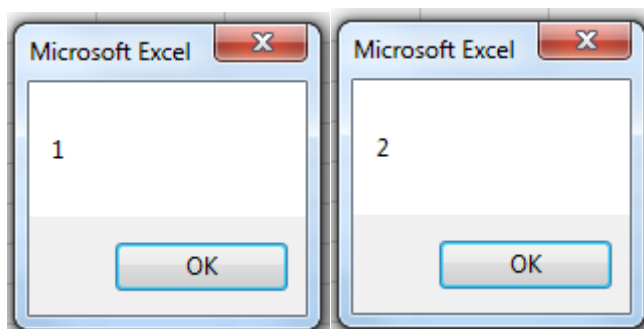
Essa função começa com **for**, é seguida de uma **variável = valor** informado pelo usuário, ou seja, nessa primeira parte temos de onde vai começar a nossa repetição, resumindo essa primeira parte seria “**de i = 1**”.

A segunda parte começa com **to** e em seguida o número final da contagem que será feita. Portanto vamos ler a seguinte expressão **for i = 1 to 10** como “**de i = 1 até 10**”, ou seja, essa variável terá seu valor acrescido de 1 sempre que chegar ao final da função.

Option Explicit

```
Sub funcaoFOR()  
  
Dim var As Integer, i As Integer  
  
For i = 1 To 10  
    MsgBox (i)  
Next  
  
End Sub
```

Ao executar esse código o programa vai retornar 10 caixas de mensagem com os números indo de 1 até 10 (uma após a outra conforme a regra do MsgBox).



Quando a variável que está dentro da função for chegar até o último valor ele vai executar o código pela última vez e então vai sair da função, ou seja, o código vai ser executado até que a variável chegue no seu último valor e então continua para o resto do código.

Ao final da função for podemos observar que temos **next** escrito, que significa próximo, ou seja, sempre que o que está dentro da função terminar de executar esse **next** vai fazer com que a variável passe para o **próximo valor**.

Sabendo como a função funciona vamos agora escrever um exemplo prático que seja mais fácil de visualizar como essa função realmente é executada.

Option Explicit

```
Sub funcaoFOR()

Dim var As Integer, i As Integer

For i = 1 To 10
    Cells(i, 7).Value = i
Next

End Sub
```

Neste outro código vamos utilizar o que já aprendemos, ou seja, vamos utilizar essa função para preencher algumas células. Neste caso vamos utilizar a função **Cells** para fazer com que o código preencher da **linha i** (que vai do número 1 até 10). Portanto a cada passagem no código esse número vai ser modificado e por consequência o número da linha vai variar assim como o valor que será atribuído a célula.

G
1
2
3
4
5
6
7
8
9
10

Ao executar o código esse é o resultado, portanto vamos da célula **G1 até a G10** e o **valor foi do 1 até o 10** crescendo de 1 em 1. Portanto para tarefas que exijam repetição essa é a função que será utilizada. Neste caso poderíamos ter escrito **cells(1,7).value = 1**, **cells(2,7).value = 2**, e assim por diante, no entanto seria uma tarefa trabalhosa e quando fosse preciso modificar alguma informação teríamos mais trabalho ainda para efetuar essa modificação.

Temos uma outra forma de utilizar essa função que é utilizando o **For Each**, que neste caso vamos utilizar o **cell** como variável. Então vamos ler da seguinte maneira “**para cada célula no range1**”, ou seja, ao invés de colocarmos o valor de início e fim, o VBA vai assumir que a primeira célula é a inicial e a repetição só finaliza na última célula.

Option Explicit

```
Sub funcaoFOREACH()

Dim var As Integer
Dim range1 As Range, cell As Range

var = 1
Set range1 = Range("H1:H15")

For Each cell In range1
    cell.Value = var
Next

End Sub
```

Então o que esse código vai fazer é pegar o **range1** que vai da célula **H1** até **H5** e vai colocar em cada célula o valor da variável **var**, que é igual a 1.

G	H
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
	1
	1
	1
	1
	1

Portanto todo o range foi preenchido com o valor 1, ou seja as duas funções **for** e **for each** tem a mesma funcionalidade, a única coisa que vai mudar é quando utilizar cada uma delas. O **for each** vamos utilizar quando tivermos um range definido e quisermos fazer alguma modificação com repetição dentro desse range. As duas funcionam da mesma forma caso a atividade seja a mesma o resultado das duas será igual.

Terceira Ferramenta

Nesta parte vamos criar uma ferramenta para preencher a tabela que foi utilizada nos exemplos de If, Else, For e For Each. A ideia é automatizar o processo para que seja possível preencher automaticamente os espaços em branco com as informações que estão mais acima, ou seja, não foram repetidas anteriormente, pois na criação da tabela pode não ter sido útil repetir todas essas informações.

Além de preencher esses espaços vazios das colunas A, B e D também vamos fazer o cálculo e preenchimento automático da coluna E que é a coluna de valor total dos itens da tabela em questão.

	A	B	C	D	E
1	Código	Produto Descrição	Quantidade Vendida	Valor Unitário	Valor Total
2	AB1	Produto AB1	849	7	
3			11		
4			646		
5			626		
6			150		
7	AB2	Produto AB2	459	3	
8			541		
9			940		
10	AB3	Produto AB3	930	7	
11	AB4	Produto AB4	19	6	
12	AB5	Produto AB5	851	2	
13			670		
14			735		
15			457		
16			891		
17			566		
18			538		
19	AB6	Produto AB6	80	5	
20			646		
21			760		
22			141		
23			203		
24			120		
25			971		
26			535		
27			337		
28			889		
29			124		
30			861		
31	AB7	Produto AB7	169	4	
32	AB8	Produto AB8	618	2	
33	AB9	Produto AB9	861	10	
34	AB10	Produto AB10	410	3	
35	AB11	Produto AB11	87	5	
36	AB12	Produto AB12	694	4	
37	AB13	Produto AB13	774	4	
38	AB14	Produto AB14	15	10	

Vamos iniciar com a análise da **coluna A**. Primeiramente vamos observar se o valor da célula analisada é diferente de vazio, em caso positivo vamos armazenar esse valor, caso contrário vamos atribuir esse valor armazenado a célula vazia.

Feito isso vamos poder fazer a repetição para essa análise. Então a célula **A1** está preenchida, então o valor armazenado é **Código**, a próxima célula a ser

analisada é **A2**, que também tem valor, portanto vai ser armazenado o valor **AB1**, já a célula **A3** está vazia, neste caso vamos atribuir o valor da variável armazenada que é **AB1** a esta célula.

Esse procedimento vai se repetir durante toda a coluna para que ela seja preenchida por completo.

```
Option Explicit

Sub ajeitarTabela()

Dim valor_atual As String
Dim range1 As Range, cell As Range

Set range1 = Range("A1:A239")

For Each cell In range1
    If cell.Value <> "" Then
        valor_atual = cell.Value
    Else
        cell.Value = valor_atual
    End If
Next

End Sub
```

Essa é a primeira parte do código, portanto é a parte que vai efetuar o preenchimento da coluna A. Então ao executar o código toda a coluna A será preenchida.

	A	
1	Código	
2	AB1	F
3	AB1	
4	AB1	
5	AB1	
6	AB1	
7	AB2	F
8	AB2	
9	AB2	
10	AB3	F
11	AB4	F
12	AB5	F
13	AB5	
14	AB5	
15	AB5	
16	AB5	
17	AB5	
18	AB5	
19	AB6	F
20	AB6	
21	AB6	

Agora precisamos repetir o mesmo procedimento para as **colunas B e D**, portanto não vamos precisar reescrever o código, basta copiar o que foi escrito para a coluna A e utilizar a **propriedade offset** para deslocar a quantidade necessária de colunas.

Este código será utilizado para a coluna B, por isso será utilizado o offset de apenas 1 coluna.

```
For Each cell In range1.Offset(0, 1)
    If cell.Value <> "" Then
        valor_atual = cell.Value
    Else
        cell.Value = valor_atual
    End If
Next
```

Este código será utilizado para a coluna D, por isso será utilizado o offset de 3 colunas.

```
For Each cell In range1.Offset(0, 3)
    If cell.Value <> "" Then
        valor_atual = cell.Value
    Else
        cell.Value = valor_atual
    End If
Next
```

Portanto ao executar o código teremos todas as colunas A, B e D preenchidas da forma que foi proposto, portanto todas as colunas agora estão completas da linha 1 até a linha 239 que é a última linha desta tabela.

Como na coluna D temos valores que representam dinheiro vamos acrescentar nesse último código uma linha que altera a formatação da célula para o **formato contábil**.

```
For Each cell In range1.Offset(0, 3)
    If cell.Value <> "" Then
        valor_atual = cell.Value
    Else
        cell.Value = valor_atual
    End If
    cell.Style = "Currency"
Next
```

Macros VBA para Excel Completo

	A	B	C	D	E
1	Código	Produto Descrição	Quantidade Vendida	Valor Unitário	Valor Total
2	AB1	Produto AB1	849	R\$ 7,00	
3	AB1	Produto AB1	11	R\$ 7,00	
4	AB1	Produto AB1	646	R\$ 7,00	
5	AB1	Produto AB1	626	R\$ 7,00	
6	AB1	Produto AB1	150	R\$ 7,00	
7	AB2	Produto AB2	459	R\$ 3,00	
8	AB2	Produto AB2	541	R\$ 3,00	
9	AB2	Produto AB2	940	R\$ 3,00	
10	AB3	Produto AB3	930	R\$ 7,00	
11	AB4	Produto AB4	19	R\$ 6,00	
12	AB5	Produto AB5	851	R\$ 2,00	
13	AB5	Produto AB5	670	R\$ 2,00	
14	AB5	Produto AB5	735	R\$ 2,00	
15	AB5	Produto AB5	457	R\$ 2,00	
16	AB5	Produto AB5	891	R\$ 2,00	
17	AB5	Produto AB5	566	R\$ 2,00	
18	AB5	Produto AB5	538	R\$ 2,00	
19	AB6	Produto AB6	80	R\$ 5,00	
20	AB6	Produto AB6	646	R\$ 5,00	
21	AB6	Produto AB6	760	R\$ 5,00	
22	AB6	Produto AB6	141	R\$ 5,00	
23	AB6	Produto AB6	203	R\$ 5,00	
24	AB6	Produto AB6	120	R\$ 5,00	
25	AB6	Produto AB6	971	R\$ 5,00	
26	AB6	Produto AB6	535	R\$ 5,00	
27	AB6	Produto AB6	337	R\$ 5,00	
28	AB6	Produto AB6	889	R\$ 5,00	
29	AB6	Produto AB6	124	R\$ 5,00	
30	AB6	Produto AB6	861	R\$ 5,00	
31	AB7	Produto AB7	169	R\$ 4,00	
32	AB8	Produto AB8	618	R\$ 2,00	
33	AB9	Produto AB9	861	R\$ 10,00	
34	AB10	Produto AB10	410	R\$ 3,00	
35	AB11	Produto AB11	87	R\$ 5,00	
36	AB12	Produto AB12	694	R\$ 4,00	
37	AB13	Produto AB13	774	R\$ 4,00	
38	AB14	Produto AB14	15	R\$ 10,00	

Para finalizar a tabela é necessário completar a coluna E, que é o valor total dos produtos, portanto será necessário acrescentar um código que faça a multiplicação da coluna C com a coluna D e novamente colocar dentro de uma repetição para que toda a coluna seja preenchida.

```

For Each cell In range1.Offset(0, 4)
    cell.Value = cell.Offset(0, -1).Value * cell.Offset(0, -2).Value
    cell.Style = "Currency"
Next

```

Vamos acrescentar esse **for** ao código para completar toda a tabela. É importante ressaltar que teremos que mudar o nosso range1 para que não dê erro quando for multiplicar, pois o range selecionado está incluindo o cabeçalho, então vamos começar da **célula A2** ao invés da **célula A1**.

```

Set range1 = Range("A2:A239")

```

Feito isso temos a tabela totalmente preenchida com todos os dados como foi proposto.

	A	B	C	D	E
1	Código	Produto Descrição	Quantidade Vendida	Valor Unitário	Valor Total
2	AB1	Produto AB1	849	R\$ 7,00	R\$ 5.943,00
3	AB1	Produto AB1	11	R\$ 7,00	R\$ 77,00
4	AB1	Produto AB1	646	R\$ 7,00	R\$ 4.522,00
5	AB1	Produto AB1	626	R\$ 7,00	R\$ 4.382,00
6	AB1	Produto AB1	150	R\$ 7,00	R\$ 1.050,00
7	AB2	Produto AB2	459	R\$ 3,00	R\$ 1.377,00
8	AB2	Produto AB2	541	R\$ 3,00	R\$ 1.623,00
9	AB2	Produto AB2	940	R\$ 3,00	R\$ 2.820,00
10	AB3	Produto AB3	930	R\$ 7,00	R\$ 6.510,00
11	AB4	Produto AB4	19	R\$ 6,00	R\$ 114,00
12	AB5	Produto AB5	851	R\$ 2,00	R\$ 1.702,00
13	AB5	Produto AB5	670	R\$ 2,00	R\$ 1.340,00
14	AB5	Produto AB5	735	R\$ 2,00	R\$ 1.470,00
15	AB5	Produto AB5	457	R\$ 2,00	R\$ 914,00
16	AB5	Produto AB5	891	R\$ 2,00	R\$ 1.782,00
17	AB5	Produto AB5	566	R\$ 2,00	R\$ 1.132,00
18	AB5	Produto AB5	538	R\$ 2,00	R\$ 1.076,00
19	AB6	Produto AB6	80	R\$ 5,00	R\$ 400,00
20	AB6	Produto AB6	646	R\$ 5,00	R\$ 3.230,00
21	AB6	Produto AB6	760	R\$ 5,00	R\$ 3.800,00
22	AB6	Produto AB6	141	R\$ 5,00	R\$ 705,00
23	AB6	Produto AB6	203	R\$ 5,00	R\$ 1.015,00
24	AB6	Produto AB6	120	R\$ 5,00	R\$ 600,00
25	AB6	Produto AB6	971	R\$ 5,00	R\$ 4.855,00
26	AB6	Produto AB6	535	R\$ 5,00	R\$ 2.675,00
27	AB6	Produto AB6	337	R\$ 5,00	R\$ 1.685,00
28	AB6	Produto AB6	889	R\$ 5,00	R\$ 4.445,00
29	AB6	Produto AB6	124	R\$ 5,00	R\$ 620,00
30	AB6	Produto AB6	861	R\$ 5,00	R\$ 4.305,00
31	AB7	Produto AB7	169	R\$ 4,00	R\$ 676,00
32	AB8	Produto AB8	618	R\$ 2,00	R\$ 1.236,00
33	AB9	Produto AB9	861	R\$ 10,00	R\$ 8.610,00
34	AB10	Produto AB10	410	R\$ 3,00	R\$ 1.230,00
35	AB11	Produto AB11	87	R\$ 5,00	R\$ 435,00
36	AB12	Produto AB12	694	R\$ 4,00	R\$ 2.776,00
37	AB13	Produto AB13	774	R\$ 4,00	R\$ 3.096,00
38	AB14	Produto AB14	15	R\$ 10,00	R\$ 150,00

Agora temos a planilha completa totalmente automatizada utilizando as funções que aprendemos até o momento.

As aplicações são diversas, o que vai mudar é como essas funções serão utilizadas para que as atividades sejam executadas de forma correta e de maneira satisfatória.

Estrutura With

Essa estrutura permite ao usuário alterar mais de uma propriedade de um objeto sem ter que ficar repetindo a escrita dele, ou seja, ela economiza tempo e deixa o código menos poluído.

Para verificar como é essa estrutura vamos gravar uma macro. A macro a ser gravada será com as seguintes ações: vamos **mudar o valor da célula A1, a cor da célula, a cor da fonte**, colocar a célula em **negrito** e também em **itálico**.

```
Sub Macro1()

    ActiveCell.FormulaR1C1 = "Teste"
    Range("A1").Select
    With Selection.Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .Color = 65535
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
    With Selection.Font
        .Color = -16776961
        .TintAndShade = 0
    End With
    Selection.Font.Bold = True
    Selection.Font.Italic = True

End Sub
```

É possível observar que o **with** no primeiro caso suprime a utilização da expressão **selection.interior** nas opções abaixo, ou seja, nesse primeiro caso seria necessário escrever essa expressão 5 vezes, no entanto, com essa estrutura podemos partir da propriedade do objeto começando com o ponto.

Sabendo disso vamos pegar essas informações que foram obtidas com o gravador de macro e vamos utilizar o with com algumas delas. Temos algumas que ainda podem ser agrupadas com essa estrutura, que é o caso das duas últimas linhas por exemplo que é a mudança da fonte para **negrito** e **itálico**.

Podemos ainda acrescentar a mudança de cor a essas duas últimas linhas, pois temos exatamente a mesma estrutura **Selection.Font**, então podemos utilizar para as três opções e evitar o uso dessa escrita várias vezes.

```

Sub Macro1()

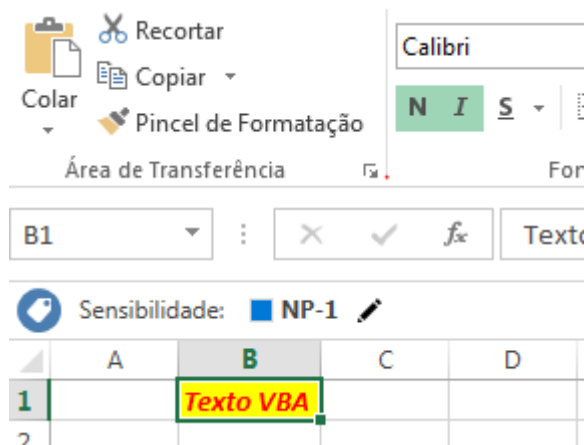
    Range("B1").Select
    With Selection
        .Interior.Color = 65535
        .Value = "Texto VBA"
        With .Font
            .Color = -16776961
            .Bold = True
            .Italic = True
        End With
    End With

End Sub

```

Neste caso estamos suprimindo a repetição da propriedade **selection** para modificar a cor da célula e o valor dela, assim como estamos suprimindo a propriedade **font** para modificar a cor da fonte, modificá-la para negrito e para itálico. Então é possível também colocar um with dentro de outro sem prejudicar a estrutura para evitar mais repetições, no último caso teríamos que escrever **selection.font.color** por exemplo para modificar a cor da fonte, e neste caso não precisamos de todas as propriedades escritas para que funcionem.

Podemos utilizar um with dentro de outro e suprimir tanto a propriedade **selection** quanto a propriedade **font**. Claro que isso será útil quando houver mais de uma propriedade a ser alterada.



É possível observar que a célula B2 está totalmente formatada como foi descrito no código que acabou de ser criado. Isso quer dizer que a estrutura with funciona normalmente para suprimir as propriedades repetidas sem que haja alguma mudança na sua execução.

Estrutura Select Case

Essa estrutura funciona como se tivéssemos várias estruturas **If**, ou seja, com ela temos a possibilidade de colocar várias opções e caso algumas delas seja verdadeira o código vai diretamente a ela. Neste caso não é necessário escrever várias vezes a estrutura **If** para fazer essa verificação.

Para fazer esse teste vamos utilizar a planilha abaixo que na célula ao lado de operações temos 4 opções: soma, diferença, multiplicação e divisão.

	A	B	C	D	E
1	Operação:				
2		Soma			
3	1º Num	Diferença	Num	4º Num	5º Num
4		Multiplicação			
5		Divisão			
6	Resultado:				
7					

Então para cada uma dessas opções vamos atribuir uma ação como se fosse um **If**, ou seja, se a **célula B2** for **igual a soma** vamos **somar os valores**, se for igual a **diferença** vamos **subtrair os valores**, se for igual a **multiplicação** vamos **multiplicar os valores** e por fim se for igual a **divisão** vamos **dividir os valores**.

Vamos ter alguns detalhes em algumas das operações para que não tenhamos problemas, pois podemos ter **multiplicações por 0**, **divisões por 0** e isso não será útil para nós e pode até gerar um erro durante a execução do código, portanto o código abaixo será mais completo abordando essas particularidades.


```

Sub estruturaSelectCase()

Dim var As String
Dim range1 As Range, cell As Range

Set range1 = Range("A4:E4")
var = Range("B1")
Dim resultado As Double

Select Case var
    Case "Soma"
        For Each cell In range1
            resultado = resultado + cell.Value
        Next
    Case "Diferença"
        For Each cell In range1
            If cell.Column = 1 Then
                resultado = cell.Value
            Else
                resultado = resultado - cell.Value
            End If
        Next
    Case "Multiplicação"
        For Each cell In range1
            If cell.Column = 1 Then
                resultado = cell.Value
            Else
                resultado = resultado * cell.Value
            End If
        Next
    Case "Divisão"
        For Each cell In range1
            If cell.Column = 1 Then
                resultado = cell.Value
            Else
                If cell.Value <> "" Then
                    resultado = resultado / cell.Value
                End If
            End If
        Next
    Case Else
        MsgBox ("Selecione uma operação para realizar")
End Select

Range("B6").Value = resultado

End Sub

```

Esse é o código com a estrutura **Select Case**, é possível observar que temos as declarações das variáveis no início do código e logo abaixo podemos verificar o início da estrutura.

Essa estrutura começa com o **nome dela mais a variável que será analisada**, ou seja, os **casos a serem analisados** são as **possibilidades que essa variável pode assumir**. Neste caso temos as 4 operações básicas mais um caso **Else** que é uma execução se nenhum desses casos for satisfeito ou

encontrado. Desta forma podemos ter uma ação corretiva no caso de não termos nenhum caso encontrado.

Vamos agora analisar os casos que temos para entender como funcionam cada um deles.

O **primeiro caso** é quando a nossa variável é **Soma**.

```
Case "Soma"
    For Each cell In range1
        resultado = resultado + cell.Value
    Next
```

Esse caso é o mais simples, só precisamos de uma estrutura de repetição para somar os valores do range selecionado que vai da **célula A4** até **E4**. O valor da célula analisada está sendo somado ao valor do resultado e atribuída a variável resultado. Isso é possível porque depois do igual estamos pegando o valor antigo da **variável resultado** e somando ao valor da célula analisada.

O **segundo caso** é quando a variável assume o valor de **Diferença**.

```
Case "Diferença"
    For Each cell In range1
        If cell.Column = 1 Then
            resultado = cell.Value
        Else
            resultado = resultado - cell.Value
        End If
    Next
```

Neste caso temos que colocar uma estrutura a mais para modificar o valor inicial da variável resultado, pois se não alterarmos todos os valores ficarão negativos, ou seja, ficaria 0 menos os outros valores e não o primeiro menos os outros valores. Por isso é necessário colocar a **estrutura if** para checar se estamos na primeira coluna, que é o primeiro dado a ser analisado.

O **terceiro caso** é quando a variável assume o valor de **Multiplicação**.

```
Case "Multiplicação"
    For Each cell In range1
        If cell.Column = 1 Then
            resultado = cell.Value
        Else
            resultado = resultado * cell.Value
        End If
    Next
```

Este caso é similar ao caso da subtração, vamos utilizar a mesma estrutura if para evitar que o resultado seja sempre 0, pois a variável resultado

começa em 0. Portanto o resultado só será 0 caso alguns dos números analisados seja igual a 0, pois qualquer número multiplicado por 0 é 0.

O **quarto caso** é quando a variável assume o valor de **Divisão**.

```
Case "Divisão"
    For Each cell In range1
        If cell.Column = 1 Then
            resultado = cell.Value
        Else
            If cell.Value <> "" Then
                resultado = resultado / cell.Value
            End If
        End If
    Next
```

Este caso tem mais uma particularidade além de analisar se estamos no primeiro valor, para que a divisão não seja sempre igual a 0 e também para evitar que seja feita uma divisão por 0, que gera um erro caso ocorra.

O **quinto** e último **caso** é o **caso Else**, que é para que alguma ação seja executada caso nenhum dos casos acima seja encontrado. Não é obrigatório, mas é sempre bom colocar para atribuir alguma ação no caso de não encontrar nenhum dos casos.

```
Case Else
    MsgBox ("Selecione uma operação para realizar")
End Select
```

Então caso não seja encontrada nenhuma opção que seja atendida pelos casos descritos no código o usuário receberá uma caixa de texto com a mensagem para selecionar uma operação.

Foi possível observar a estrutura do **select case**, podendo colocar diversos casos, lembrando que funciona igual a **estrutura if**, somente um caso será executado por vez, pois não tem como uma variável assumir dois valores ao mesmo tempo.

E assim como as outras estruturas esse também tem um indicativo de término que é o **End Select**.

Estrutura While e Adicionando Condições

Essa estrutura funciona como uma estrutura de repetição, no entanto vai se repetir até que sua condição seja falsa, ou seja, poderíamos traduzir desta forma: “**enquanto a condição seja verdadeira, faça isso**”, caso contrário ela para de executar. O **while** significa exatamente esse enquanto.

Ao contrário do **For** ou **For Each** que tem um fim definido essa estrutura não possui isso, ela só vai parar a execução quando a condição for falsa. Sabendo disso é necessário tomar cuidado para não entrar em um **loop infinito**, ou seja, entrar em uma repetição que a condição nunca seja falsa.

Caso isso aconteça o programa ficará executando infinitamente, sem que seja possível parar e pode gerar erro no programa, consequentemente o fazendo parar de funcionar.

Neste caso vamos utilizar essa estrutura para verificar se o valor da célula atual é **menor ou igual a 100**, então enquanto esse valor for menor ou igual a 100 nossa estrutura vai **adicionar 1** ao número da linha para analisar a célula seguinte. Esse procedimento vai se repetir até que um número maior do que 100 seja encontrado. Quando for encontrado vamos colocar os dados de linha na **célula C2** e o valor encontrado na **célula D2**.

	A	B	C	D
1	Valores		Linha	Valor
2	6			
3	21			
4	98			
5	38			
6	31			
7	99			
8	98			
9	91			
10	46			
11	39			
12	6			
13	31			
14	100			
15	70			
16	49			
17	73			
18	81			
19	22			

A **coluna A** possui mais de 1000 linhas, portanto não seria muito conveniente fazer esse tipo de procura de forma manual, por isso vamos utilizar essa estrutura de repetição para que possa encontrar mais rapidamente as informações que precisamos.

```

Sub estruturaWhile()

Dim linha As Integer

linha = 2
Cells(linha, 1).Select

While ActiveCell.Value <= 100
    linha = linha + 1
    Cells(linha, 1).Select
Wend

Range("C2").Value = linha
Range("D2").Value = ActiveCell.Value

End Sub

```

O código para executar essa ação é bem simples. Primeiramente temos a declaração da **variável linha**, assim como a definição deste **valor que é igual a 2**, pois a primeira célula possui um texto e não um valor, por isso temos que começar da segunda linha.

Logo abaixo temos a seleção dessa célula utilizando o objeto **Cells**. Desta forma vamos começar a execução da estrutura **While** a partir da **célula A2**.

A estrutura do While é simples, precisamos escrever o while e logo em seguida escrever a condição que será testada, que neste caso é a verificação se o valor da célula em questão é **menor ou igual a 100**. Como toda estrutura, também precisamos dizer ao VBA quando ela será encerrada, para isso vamos escrever **Wend** depois de escrever o que será feito enquanto estiver dentro da estrutura.

Dentro da estrutura temos que o valor da linha recebe o valor anterior mais 1, e a célula selecionada passa a ser a célula seguinte. Portanto enquanto o valor da célula que está selecionada for menor ou igual a 100 essa estrutura ficará executando.

	A	B	C	D
1	Valores		Linha	Valor
2	6		1372	101
3	21			
4	98			
5	38			
6	31			

Ao executar o código foi encontrado que o primeiro valor que é maior do que 100 se encontra na **linha 1372** (que estará selecionada no momento). No final do código temos a atribuição desses valores as **células C2** e **D2** para mostrar o número da linha e qual o valor que está na célula.

Neste caso temos apenas esse único valor que é maior do que 100. Se alterarmos esse valor para um valor menor e executarmos a macro vamos entrar em um **loop infinito** porque todos os valores serão menores ou iguais a 100, então o Excel vai ficar rodando o código sem fim. **É necessário sempre tomar cuidado com isso para evitar loop infinito e consequentemente erro na execução do código em questão.**

Uma forma de evitar esse problema neste caso é colocar além da condição de valor da célula é acrescentar uma condição para o limite de célula, ou seja, neste caso vamos colocar a condição para que a linha seja menor ou igual a 2000. Desta forma temos um limite até onde o while será executado mesmo que não encontre valores maiores do que o estipulado.

```
While ActiveCell.Value <= 100 And linha <= 2000
    linha = linha + 1
    Cells(linha, 1).Select
Wend
```

Esse procedimento é feito utilizando **And** (que significa E a mesma função do ambiente Excel), ou seja, se a célula tiver o valor menor ou igual a 100 **e** o valor da linha for menor ou igual a 2000 a condição será verdadeira. Isso quer dizer que para a condição ser verdadeira e entrar dentro da estrutura **essas duas condições devem ser verdadeiras ao mesmo tempo.**

Foi possível observar que essa estrutura de repetição não tem um fim definido como a estrutura for e for each. Ela será executada até que a condição testada seja falsa e por isso é preciso tomar cuidado para que não entre em um loop infinito. Como tem a mesma função das outras duas o que irá definir qual será utilizada é realmente a aplicação do código.

Como foi mostrado no exemplo é possível colocar um limite para essa execução a fim de evitar esse loop infinito, essa pode ser uma maneira simples e fácil de tratar e evitar este erro específico.

Seção 6 – Funções e Subs

Sub x Function e Organização do Código

Nesta parte vamos ver a diferença entre **Sub** que é o que utilizamos até o momento para criar as macros e **Function** que é uma função dentro do VBA, ou seja, o usuário poderá criar funções próprias para executar diferentes ações de acordo com a sua necessidade.

A utilização de funções vai servir principalmente para não deixarmos todo o código dentro de uma única Sub, ou seja, vamos poder criar várias funções. Dentro do código principal vamos apenas fazer as “**chamadas**” dessas funções para realizar suas respectivas atividades.

A principal diferença entre elas é que a **Sub** executa tudo que está dentro dela e finaliza, enquanto com a **Function** é possível retornar um valor após a execução do código. Isso quer dizer que, por exemplo, podemos criar uma função que faça uma soma de algumas células e retorne ao código principal o resultado dessa soma.

A grande vantagem de utilizar funções dentro do código é que evita a repetição de códigos iguais durante a escrita do código. No exemplo que foi dado da soma de um conjunto de células se essa atividade for utilizada diversas vezes dentro do código basta chamar a função para que seja executada quantas vezes forem necessárias, portanto só será necessário escrevê-la apenas uma vez.

```
Sub macro()

Dim var As String

MsgBox ("Olá")
pintarAmarelo
fonteVermelha
var = FuncaoValor

MsgBox (var)

End Sub
```

```
Sub pintarAmarelo()
    MsgBox ("Preenche com Amarelo")
End Sub
```

```
Sub fonteVermelha()
    MsgBox ("Muda a cor da fonte para vermelho")
End Sub
```

```
Function FuncaoValor() As String
    MsgBox ("Execução do código dentro da função")

    FuncaoValor = "Valor de retorno da função"
End Function
```

Este é um código simples apenas para mostrar como utilizar **sub** e **function**, posteriormente será possível aprofundar um pouco mais neste assunto e com uma aplicação para melhor entendimento.

A primeira sub, chamada de macro, vai fazer a “**chamada**” de todas as outras **subs** e **functions**. Para “**chamar**” uma sub basta escrever o nome dela da mesma forma que foi escrito na criação da mesma. Para a função temos duas possibilidades: a primeira é da mesma forma da sub, apenas escrevendo o nome da função, enquanto a segunda é como está no código, colocamos uma variável igual a função. Neste último caso isso é feito quando a função retorna algum valor, desta forma podemos atribuir esse valor retornado na função para a variável dentro do código principal.

É possível observar que não será necessário executar cada macro separadamente, da forma em que o código foi escrito só será necessário executar a macro principal e ela fará a chamada das outras.

Quando o código chegar na linha da primeira sub o VBA vai “entrar” nessa sub, ou seja, vai da linha que está escrito **pintarAmarelo** para a sub **pintarAmarelo**. Vai rodar toda essa sub e ao finalizar vai voltar ao código principal e vai continuar a execução de onde parou. Então vai para a linha da sub **fonteVermelha** e repete o procedimento.

No caso da função o procedimento é o mesmo, o que vai mudar é que podemos colocar um valor para a função retornar, como neste caso a função foi declarada como **string** o valor retornado será um texto. Para indicar o valor que será retornado basta colocar no final de função o nome da função igual ao valor desejado.

Como foi dito no início esse é apenas um código para exemplificar o funcionamento de sub e function, por isso foram utilizados alguns MsgBox para que o usuário possa verificar quando a sub ou function está funcionando. É uma boa opção para testar se o código está realmente entrando na sub ou function e está executando ela normalmente.

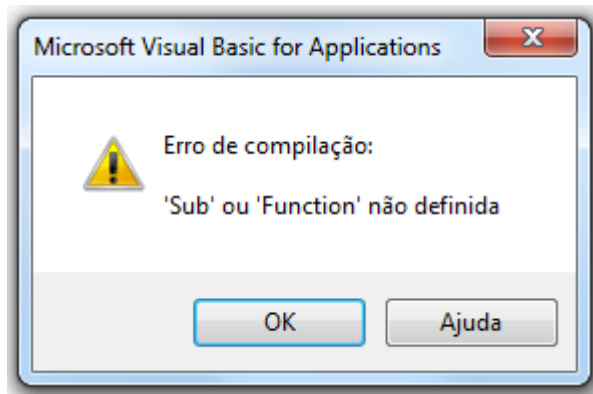
Outra informação importante sobre sub e function é que elas podem ser declaradas como **públicas** ou **privadas**. Por padrão o VBA já as deixa como públicas, ou seja, elas podem ser acessadas de outros módulos dentro do VBA, isso quer dizer que será possível utilizá-las dentro de qualquer outro módulo sem que seja necessário escrever a função.

Para colocar como privado basta colocar o nome **private** antes da função ou sub, desta forma elas ficarão sendo exclusivas daquele módulo.

```
Private Function FuncaoValor() As String
    MsgBox ("Execução do código dentro da função")

    FuncaoValor = "Valor de retorno da função"
End Function
```

Então caso o usuário tenha 2 módulos e no segundo módulo tente utilizar uma função privada do módulo 1 o programa irá retornar um erro dizendo que a sub ou function não foram definidas.



Desta forma a sub ou function fica sendo apenas do módulo em que foi declarada e não poderá ser utilizada em outros módulos. Isso quer dizer que o usuário poderá criar diferentes funções e deixá-la para ser utilizada apenas em um único módulo do programa.

Essa declaração de pública ou privada vai depender de como deverão ser utilizadas essas funções ou subs, caso precise utilizá-las em mais de um módulo, basta deixar como está que o padrão as deixa como pública. Caso o usuário queira que aquela função seja apenas de um módulo específico, pois será utilizada para uma ação específica basta colocar como privada.

Argumentos ByRef e ByVal (Evitando Erros)

Nesta parte vamos aprender como “**passar**” argumentos para as subs ou funções, isso quer dizer que podemos enviar alguns parâmetros e/ou valores para complementar a execução da função. E para isso vamos ver a diferença dos argumentos **ByRef** (por referência) e **ByVal** (por valor).

Vamos utilizar um exemplo simples para mostrar a diferença entre as duas declarações de argumentos. Primeiramente vamos utilizar o código abaixo para verificar como ele se comporta com a execução da sub teste2.

```
Sub teste1()

a = 1
b = 2

MsgBox ("A = " & a & " B = " & b)

Call teste2(a, b)

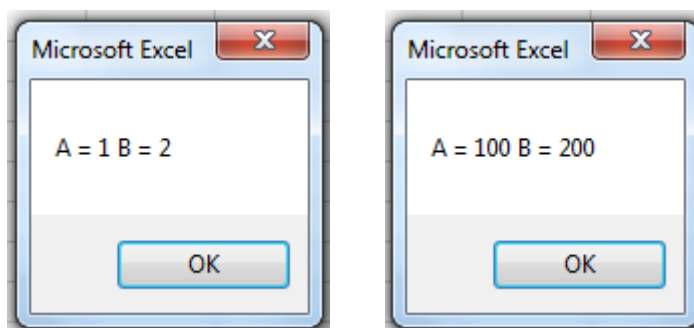
MsgBox ("A = " & a & " B = " & b)

End Sub
```

```
Sub teste2(a, b)
a = 100
b = 200
End Sub
```

No código principal temos a atribuição dos **valores 1 e 2** as variáveis **a** e **b** respectivamente. Feito isso temos dois MsgBox para que o Excel mostre o valor dessas variáveis antes e depois de entrar na **sub teste2**. Entre os dois MsgBox temos a chamada da sub que é feita utilizando **Call** e entre parênteses os argumentos que vamos passar a essa sub ou função que neste caso são as **variáveis a e b**.

Ao executar o código temos o retorno dos dois MsgBox que estão logo abaixo para mostrar os valores antes e depois da execução da segunda sub.



É possível observar que os valores foram modificados ao passarem pela sub, que faz uma nova atribuição a essas variáveis. Como padrão do Excel os

argumentos são passados por **ByRef** (que é por referência), ou seja, a chamada da função está mandando as “**posições**” onde estão as variáveis, então caso esses valores sejam alterados (como foi o caso) as variáveis vão ter seus valores alterados.

Para verificarmos a diferença vamos colocar uma variável sendo passada por referência (**ByRef**) e a outra por valor (**ByVal**) e então executar o código novamente para ver o retorno dos MsgBox.

```
Sub teste1()

a = 1
b = 2

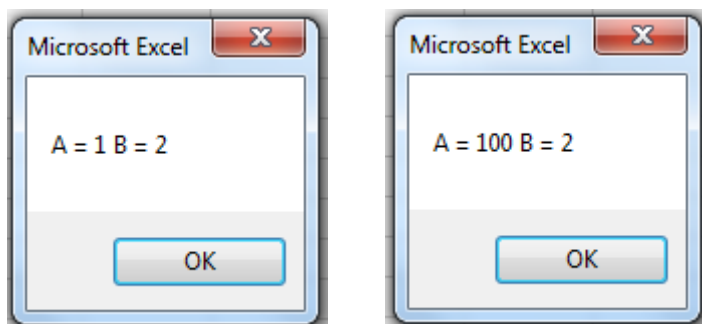
MsgBox ("A = " & a & " B = " & b)

Call teste2(a, b)

MsgBox ("A = " & a & " B = " & b)

End Sub

Sub teste2(ByRef a, ByVal b)
    a = 100
    b = 200
End Sub
```



Neste caso temos que o valor da **variável b** continua sendo **2**, pois não foi passado o local onde está o valor e sim apenas o **valor 2**, então a **sub teste2** está recebendo a **variável a** e o **valor 2**. Isso acontece porque as variáveis utilizadas dentro da **sub teste2** não são as mesmas da **sub teste1**, ou seja, as **variáveis a e b** da segunda sub tem apenas o mesmo nome, por isso ao colocar o **ByVal** na **variável b** a segunda sub atribui o valor de 200 a uma variável que não existe. Portanto não está alterando a **variável b**.

Acontece que as variáveis de cada sub são independentes, então ao passar **a** e **b** para a segunda sub não necessariamente é necessário utilizar **a** e **b** na **sub teste2** para receber esses valores e/ou posições. Então essa referência (ByRef) vamos utilizar quando queremos alterar o valor dessa variável e o valor (ByVal) vamos utilizar quando não precisarmos alterar o valor da variável.

Integração Função VBA e Fórmula Excel

Nesta parte vamos explorar um pouco mais a função (**function**) do VBA, pois além de conseguirmos utilizá-la em um código para efetuar uma ação e possivelmente retornar um valor, vamos poder utilizar essa função dentro do ambiente Excel como uma função comum, ou seja, podemos escrever dentro de uma célula = e o nome da função que ela irá executar o que foi escrito no código como uma função qualquer do Excel.

Para demonstrar essa funcionalidade vamos criar uma função que obtenha o valor mais atual de um intervalo selecionado. Para isso será utilizada a planilha abaixo.

	A	B	C	D	E	F	G	H	I
1	Produto	Valor Mais Atual	jan/15	fev/15	mar/15	abr/15	mai/15	jun/15	jul/15
2	Água		R\$ 1,50	R\$ 1,50	R\$ 1,50	R\$ 1,50	R\$ 1,50	R\$ 1,50	
3	Cerveja		R\$ 6,00	R\$ 6,00	R\$ 6,50	R\$ 6,50	R\$ 6,50	R\$ 6,50	R\$ 6,50
4	Refrigerante			R\$ 2,00					
5	Carne		R\$ 15,00	R\$ 20,00					
6	Farofa		R\$ 1,00	R\$ 1,00	R\$ 1,00	R\$ 1,00	R\$ 1,00	R\$ 1,00	R\$ 1,00

Então vamos utilizar nossa fórmula nas células da coluna B para obter o valor mais atual dos produtos da coluna A tendo como base o intervalo selecionado.

```
Function atualizarValor(intervalo As Range)

Dim cell As Range
Dim valor As Double

For Each cell In intervalo
    If cell.Value <> "" Then
        valor = cell.Value
    End If
Next

atualizarValor = valor

End Function
```

O código para essa função é bem simples. Temos a função e o argumento definido como range, pois, precisamos selecionar um intervalo para analisar qual o valor mais atual do mesmo. Em seguida são declaradas as variáveis utilizadas no código.

Logo abaixo temos a estrutura de repetição (**for each**) para verificar cada célula deste intervalo, essa verificação se dá por meio da **estrutura if**, que verifica se a célula atual é diferente de vazia, em caso positivo vamos atualizar o valor da **variável valor**, caso contrário a próxima célula será analisada até terminar o intervalo selecionado.

Por fim temos que a função retorna o valor da **variável valor**, que é o valor mais atual do intervalo selecionado.

Com o código pronto podemos ir até o ambiente Excel e escrever o nome da função que ela estará disponível para uso.

	A	B	
1	Produto	Valor Mais Atual	
2	Água	=at	R\$
3	Cerveja	ATAN	R\$
4	Refrigerante	ATAN2	
5	Carne	ATANH	R\$
6	Farofa	atualizarValor	R\$

Vamos selecionar a função e inserir o intervalo a ser analisado.

B	C
Valor Mais Atual	ja
=atualizarValor(C2:Z2)	R\$ €

Feito isso basta pressionar **enter** e teremos o resultado na célula em que foi colocada a função.

	A	B	
1	Produto	Valor Mais Atual	
2	Água	R\$ 2,00	F
3	Cerveja		F

Este é o valor mais atual do intervalo selecionado, agora para as próximas células basta arrastar a fórmula que ela irá funcionar como as funções do Excel e irá modificar as referências automaticamente.

	A	B	
1	Produto	Valor Mais Atual	
2	Água	R\$ 2,00	F
3	Cerveja	R\$ 8,50	F
4	Refrigerante	R\$ 2,00	
5	Carne	R\$ 35,00	F
6	Farofa	R\$ 2,50	F

Com essa fórmula vamos ter o valor mais atual, considerando que essa análise o VBA começa a verificação da primeira célula até a última, que neste caso a primeira representa uma data mais antiga.

Então ao criar a função no VBA podemos utilizá-la dentro do ambiente Excel. Isso permite ao usuário criar novas funções para aplicações específicas que podem ser criadas de acordo com a necessidade, isso dá ao usuário uma maior usabilidade ainda do ambiente Excel, pois se não tem uma função que atenda a necessidade ela pode ser criada no ambiente VBA.

Quarta Ferramenta - Consolidação

Nesta parte vamos criar uma **ferramenta de consolidação** para duas contas bancárias fictícias, então o objetivo é fazer um **resumo com as informações das duas contas em uma única tabela** com o intuito de facilitar a visualização de todas essas informações.

	A	B	C	D	E	F	G	H
1	Data	Dia da Semana	Tipo	Valor	Documento	Status	Saldo Anterior	Saldo Final
2	27/10/2015	Terça-feira	Entrada	R\$ 2.386,00	E27102C1	Finalizado	-R\$ 100,00	R\$ 2.286,00
3	27/10/2015	Terça-feira	Saída	R\$ 9.228,00	S27103C1	Pendente	R\$ 2.286,00	-R\$ 6.942,00
4	27/10/2015	Terça-feira	Entrada	R\$ 8.833,00	E27104C1	Finalizado	-R\$ 6.942,00	R\$ 1.891,00
5	27/10/2015	Terça-feira	Entrada	R\$ 8.387,00	E27105C1	Finalizado	R\$ 1.891,00	R\$ 10.278,00
6	27/10/2015	Terça-feira	Entrada	R\$ 99,00	E27106C1	Finalizado	R\$ 10.278,00	R\$ 10.377,00
7	27/10/2015	Terça-feira	Saída	R\$ 6.741,00	S27107C1	Finalizado	R\$ 10.377,00	R\$ 3.636,00
8	28/10/2015	Quarta-feira	Entrada	R\$ 694,00	E28108C1	Finalizado	R\$ 3.636,00	R\$ 4.330,00
9	28/10/2015	Quarta-feira	Saída	R\$ 6.167,00	S28109C1	Finalizado	R\$ 4.330,00	-R\$ 1.837,00
10	28/10/2015	Quarta-feira	Saída	R\$ 3.512,00	S281010C1	Finalizado	-R\$ 1.837,00	-R\$ 5.349,00
11	28/10/2015	Quarta-feira	Saída	R\$ 5.665,00	S281011C1	Finalizado	-R\$ 5.349,00	-R\$ 11.014,00
12	28/10/2015	Quarta-feira	Saída	R\$ 9.126,00	S281012C1	Finalizado	-R\$ 11.014,00	-R\$ 20.140,00
13	28/10/2015	Quarta-feira	Saída	R\$ 4.163,00	S281013C1	Finalizado	-R\$ 20.140,00	-R\$ 24.303,00
14	28/10/2015	Quarta-feira	Entrada	R\$ 9.982,00	E281014C1	Finalizado	-R\$ 24.303,00	-R\$ 14.321,00
15	28/10/2015	Quarta-feira	Entrada	R\$ 7.388,00	E281015C1	Finalizado	-R\$ 14.321,00	-R\$ 6.933,00
16	28/10/2015	Quarta-feira	Entrada	R\$ 9.974,00	E281016C1	Finalizado	-R\$ 6.933,00	R\$ 3.041,00
17	28/10/2015	Quarta-feira	Entrada	R\$ 7.380,00	E281017C1	Finalizado	R\$ 3.041,00	R\$ 10.421,00
18	29/10/2015	Quinta-feira	Entrada	R\$ 8.901,00	E291018C1	Finalizado	R\$ 10.421,00	R\$ 19.322,00
19	30/10/2015	Sexta-feira	Entrada	R\$ 5.569,00	E301019C1	Finalizado	R\$ 19.322,00	R\$ 24.891,00
20	30/10/2015	Sexta-feira	Entrada	R\$ 8.906,00	E301020C1	Finalizado	R\$ 24.891,00	R\$ 33.797,00
21	30/10/2015	Sexta-feira	Saída	R\$ 3.405,00	S301021C1	Finalizado	R\$ 33.797,00	R\$ 30.392,00
22	30/10/2015	Sexta-feira	Entrada	R\$ 9.326,00	E301022C1	Finalizado	R\$ 30.392,00	R\$ 39.718,00
23	30/10/2015	Sexta-feira	Entrada	R\$ 855,00	E301023C1	Finalizado	R\$ 39.718,00	R\$ 40.573,00
24	30/10/2015	Sexta-feira	Entrada	R\$ 9.863,00	E301024C1	Finalizado	R\$ 40.573,00	R\$ 50.436,00
25	02/11/2015	Segunda-feira	Saída	R\$ 2.813,00	S21125C1	Finalizado	R\$ 50.436,00	R\$ 47.623,00
26	02/11/2015	Segunda-feira	Entrada	R\$ 1.294,00	E21126C1	Finalizado	R\$ 47.623,00	R\$ 48.917,00
27	02/11/2015	Segunda-feira	Saída	R\$ 6.929,00	S21127C1	Finalizado	R\$ 48.917,00	R\$ 41.988,00
28	02/11/2015	Segunda-feira	Saída	R\$ 9.471,00	S21128C1	Finalizado	R\$ 41.988,00	R\$ 32.517,00
29	02/11/2015	Segunda-feira	Saída	R\$ 9.650,00	S21129C1	Finalizado	R\$ 32.517,00	R\$ 22.867,00
30	02/11/2015	Segunda-feira	Saída	R\$ 4.581,00	S21130C1	Finalizado	R\$ 22.867,00	R\$ 18.286,00
31	02/11/2015	Segunda-feira	Saída	R\$ 1.894,00	S21131C1	Finalizado	R\$ 18.286,00	R\$ 16.392,00
32	02/11/2015	Segunda-feira	Entrada	R\$ 9.109,00	E21132C1	Pendente	R\$ 16.392,00	R\$ 25.501,00
33	02/11/2015	Segunda-feira	Entrada	R\$ 4.845,00	E21133C1	Finalizado	R\$ 25.501,00	R\$ 30.346,00
34	03/11/2015	Terça-feira	Entrada	R\$ 3.249,00	E31134C1	Finalizado	R\$ 30.346,00	R\$ 33.595,00
35	03/11/2015	Terça-feira	Entrada	R\$ 9.888,00	E31135C1	Finalizado	R\$ 33.595,00	R\$ 43.483,00
36	03/11/2015	Terça-feira	Entrada	R\$ 8.005,00	E31136C1	Finalizado	R\$ 43.483,00	R\$ 51.488,00
37	03/11/2015	Terça-feira	Entrada	R\$ 7.307,00	E31137C1	Finalizado	R\$ 51.488,00	R\$ 58.795,00
38	03/11/2015	Terça-feira	Saída	R\$ 4.123,00	S31138C1	Finalizado	R\$ 58.795,00	R\$ 54.672,00

Temos duas abas de conta que possuem os valores de data, dia da semana, tipo da transação, o valor da transação, o número do documento que originou a transação, o status para verificar se foi finalizado ou não, o saldo anterior e por fim o saldo final da conta após a transação.

	A	B	C	D	E	F	G	H
1	Resumo Diário das Movimentações							
2	Data	Tipo	Valor Entrada	Valor Saída	Origem	Documentos	Saldo Anterior	Saldo Final
3							R\$ -	R\$ -
4								
5								

Para a aba de consolidação vamos reunir as informações das duas contas nessa tabela para que seja possível analisar esse resumo de forma mais eficiente ao invés de analisar transação por transação em cada uma das contas.

Para que isso seja feito será necessário criar um código mais completo, ou seja, será maior e mais detalhado do que foi estudado até o momento, então vamos construir em partes utilizando a parte de sub e function que já aprendemos. Então teremos um código principal e vamos ter funções e subs auxiliares para a construção e execução do código.

Resumo do Código – O código vai entrar na aba da conta 1 e vai verificar inicialmente se a movimentação foi finalizada, em caso positivo vamos obter os valores das variáveis. Em seguida vamos até a aba de consolidação e vamos verificar se existem dados da mesma conta, do mesmo tipo e da mesma data. Em caso positivo vamos acrescentar as informações obtidas, caso contrário os dados serão registrados em uma nova linha. Esse procedimento vai ser repetido para todos os dados da conta 1 e em seguida vamos repetir o procedimento para a conta 2.

```
Option Explicit
Dim valor As Double
Dim doc As String
Dim data As Date

Sub atualizarCompilado()

    Consolidar ("Conta 1")
    Consolidar ("Conta 2")

End Sub
```

Nesta primeira parte temos a declaração de variáveis globais, ou seja, são variáveis que serão reconhecidas em qualquer sub ou function do código, então qualquer valor que seja atribuído a elas valerá para qualquer parte do código. Em seguida temos o início do código principal que faz a chamada da **Sub Consolidar** para a conta 1 e conta 2.

```
Sub Consolidar(nome_aba As String)

    Dim range1 As Range, cell As Range

    Sheets(nome_aba).Activate

    Set range1 = Range("A1:A300")
    For Each cell In range1
        If AnalisarLinha(cell) Then
            RegistrarLinha
        End If
    Next

End Sub
```

A próxima parte do código consiste na Sub Consolidar que recebe o nome da aba e ativa a mesma. Em seguida temos uma estrutura de repetição para analisar todos os dados dessa aba.

Para essa análise vamos verificar se a função **AnalisarLinha** retorna **verdadeiro** (true), em caso positivo vamos executar a **Sub RegistrarLinha**.

```
Function AnalisarLinha(cell As Range) As Boolean
    If cell.Offset(0, 5).Value = "Finalizado" Then
        data = cell.Value
        doc = cell.Offset(0, 4).Value
        Call PegarValor(cell)
        AnalisarLinha = True
    Else
        AnalisarLinha = False
    End If
End Function
```

Aqui temos a função **AnalisarLinha**, que vai verificar se o status da transação é **Finalizado**, em caso positivo iremos obter as informações referentes a transação como: data, documento, valor (para valor vai verificar com a sub **PegarValor** se é positivo ou negativo). Por fim retorna o valor booleano (que pode ser apenas **True** or **False**) de acordo com as análises.

```
Sub PegarValor(cell As Range)
    If cell.Offset(0, 2).Value = "Entrada" Then
        valor = cell.Offset(0, 3).Value
    ElseIf cell.Offset(0, 2).Value = "Saída" Then
        valor = -cell.Offset(0, 3).Value
    Else
        valor = 0
        cell.Offset(0, 8).Value = "Não Compilado"
    End If
End Sub
```

Essa sub é para verificar se a transação é de entrada ou saída para que possamos atribuir o valor positivo ou negativo a variável **valor**. É também possível observar que temos uma condição no caso de o tipo ser diferente de entrada ou saída será colocado um aviso na **coluna I** de que o não foi compilado, para indicar que houve um problema.

O próximo código é maior do que os outros, pois é nele são feitas as atribuições para a aba de consolidação. E feitas algumas verificações para que não ocorram erros ou sobreposições de valores.

Macros VBA para Excel Completo

```
Sub RegistrarLinha()  
  
Dim nome_aba_conta As String, nome_aba_consolidacao As String  
Dim range_consolidado As Range, cell As Range  
  
nome_aba_conta = ActiveSheet.Name  
nome_aba_consolidacao = "Consolidação de Contas"  
  
Sheets(nome_aba_consolidacao).Activate  
  
Set range_consolidado = Range("A1", Range("A1").End(xlDown)).Offset(1, 0)  
  
For Each cell In range_consolidado  
    If cell.Value = "" Then  
        cell.Value = data  
        cell.Offset(0, 5).Value = doc  
        cell.Offset(0, 4).Value = nome_aba_conta  
        If valor < 0 Then  
            cell.Offset(0, 3).Value = valor  
            cell.Offset(0, 1).Value = "Saída"  
        ElseIf valor > 0 Then  
            cell.Offset(0, 2).Value = valor  
            cell.Offset(0, 1).Value = "Entrada"  
        End If  
        Exit For  
    End If  
  
    If cell.Value = data Then  
        If cell.Offset(0, 1).Value = "Entrada" And valor > 0 And cell.Offset(0, 4).Value = nome_aba_conta Then  
            cell.Offset(0, 2).Value = valor + cell.Offset(0, 2).Value  
            cell.Offset(0, 5).Value = cell.Offset(0, 5).Value & ";" & doc  
            Exit For  
        ElseIf cell.Offset(0, 1).Value = "Saída" And valor < 0 And cell.Offset(0, 4).Value = nome_aba_conta Then  
            cell.Offset(0, 3).Value = valor + cell.Offset(0, 3).Value  
            cell.Offset(0, 5).Value = cell.Offset(0, 5).Value & ";" & doc  
            Exit For  
        End If  
    End If  
Next  
  
Sheets(nome_aba_conta).Activate  
End Sub
```

Neste código temos uma estrutura de repetição para verificar se a linha em questão está vazia, em caso positivo serão adicionados os dados guardados nela. Caso contrário será analisado se a data, o tipo e a conta são iguais, neste caso serão acrescentadas algumas informações, como o valor e o número do documento.

Isso ocorre para que não tenhamos mais de uma linha com a mesma data, como a ideia é ter um resumo da conta, estamos pegando todas as transações de um mesmo tipo no mesmo dia e as colocando juntas, desta forma o valor é somado e o número dos documentos é acrescido.

Ao executar o código teremos nossa aba de consolidação preenchida com todos os valores. A única alteração que temos que fazer é formatar os dados das colunas de entrada e saída para moeda para melhor indicar os valores.

É possível observar que são feitas primeiras as análises das transações da conta 1 e mais abaixo teremos as transações da conta 2 também resumidas por dia, ou seja, se naquele dia tivermos mais de 1 transação de entrada ou saída elas serão representadas em apenas 1 linha, para que seja possível resumir as informações e facilitar a visualização.

Macros VBA para Excel Completo

	A	B	C	D	E	F	G	H
1	Resumo Diário das Movimentações							
2	Data	Tipo	Valor Entrada	Valor Saída	Origem	Documentos	Saldo Anterior	Saldo Final
3	27/10/2015	Entrada	R\$ 19.705,00		Conta 1	E27102C1;E27	R\$ -	R\$ 19.705,00
4	27/10/2015	Saída		-R\$ 6.741,00	Conta 1	S27107C1	R\$ 19.705,00	R\$ 12.964,00
5	28/10/2015	Entrada	R\$ 35.418,00		Conta 1	E28108C1;E28	R\$ 12.964,00	R\$ 48.382,00
6	28/10/2015	Saída		-R\$ 28.633,00	Conta 1	S28109C1;S28	R\$ 48.382,00	R\$ 19.749,00
7	29/10/2015	Entrada	R\$ 8.901,00		Conta 1	E291018C1	R\$ 19.749,00	R\$ 28.650,00
8	30/10/2015	Entrada	R\$ 34.519,00		Conta 1	E301019C1;E3	R\$ 28.650,00	R\$ 63.169,00
9	30/10/2015	Saída		-R\$ 3.405,00	Conta 1	S301021C1	R\$ 63.169,00	R\$ 59.764,00
10	02/11/2015	Saída		-R\$ 35.338,00	Conta 1	S21125C1;S21	R\$ 59.764,00	R\$ 24.426,00
11	02/11/2015	Entrada	R\$ 6.139,00		Conta 1	E21126C1;E21	R\$ 24.426,00	R\$ 30.565,00
12	03/11/2015	Entrada	R\$ 28.449,00		Conta 1	E31134C1;E31	R\$ 30.565,00	R\$ 59.014,00
13	03/11/2015	Saída		-R\$ 11.658,00	Conta 1	S31138C1;S31	R\$ 59.014,00	R\$ 47.356,00
14	27/10/2015	Entrada	R\$ 38.207,00		Conta 2	E27102C1;S27	R\$ 47.356,00	R\$ 85.563,00
15	27/10/2015	Saída		-R\$ 21.171,00	Conta 2	S28109C1;S28	R\$ 85.563,00	R\$ 64.392,00
16	28/10/2015	Saída		-R\$ 21.132,00	Conta 2	E281014C1;E2	R\$ 64.392,00	R\$ 43.260,00
17	28/10/2015	Entrada	R\$ 6.306,00		Conta 2	E281017C1	R\$ 43.260,00	R\$ 49.566,00
18	29/10/2015	Entrada	R\$ 10.657,00		Conta 2	E291018C1;E3	R\$ 49.566,00	R\$ 60.223,00
19	29/10/2015	Saída		-R\$ 8.721,00	Conta 2	S301021C1;E3	R\$ 60.223,00	R\$ 51.502,00
20	30/10/2015	Entrada	R\$ 2.311,00		Conta 2	E301024C1	R\$ 51.502,00	R\$ 53.813,00
21	31/10/2015	Saída		-R\$ 2.160,00	Conta 2	S21125C1	R\$ 53.813,00	R\$ 51.653,00
22	01/11/2015	Entrada	R\$ 6.115,00		Conta 2	E21126C1	R\$ 51.653,00	R\$ 57.768,00
23	02/11/2015	Saída		-R\$ 4.963,00	Conta 2	S21127C1	R\$ 57.768,00	R\$ 52.805,00
24	03/11/2015	Saída		-R\$ 9.989,00	Conta 2	S21128C1	R\$ 52.805,00	R\$ 42.816,00
25	04/11/2015	Saída		-R\$ 7.339,00	Conta 2	S21129C1	R\$ 42.816,00	R\$ 35.477,00
26	05/11/2015	Saída		-R\$ 13.317,00	Conta 2	S21130C1;S21	R\$ 35.477,00	R\$ 22.160,00
27	05/11/2015	Entrada	R\$ 15.660,00		Conta 2	E21133C1;E31	R\$ 22.160,00	R\$ 37.820,00

Com essa ferramenta foi possível utilizar os vários conhecimentos adquiridos no curso, foi possível utilizar as **subs** e **functions** durante o código o que facilita para que não haja necessidade de escrever códigos repetidos para ações repetidas. Foi possível utilizar as estruturas de condição **If** e **Else**, tanto quanto as estruturas de repetição **For Each**.

Como foi uma ferramenta mais complexa ela pode gerar alguns erros, portanto é necessário que o aluno consiga **debugar** o código para corrigir esses erros e analisar o código (utilizando a **Depuração Total**, tecla **F8**) para verificar pontualmente onde está ocorrendo o erro.

Lembrando que boa parte desses erros pode ser apenas pela escrita errada, colocar uma letra a mais, esquecer de outra, esse tipo de erro é muito comum e se alguma função ou variável não esteja escrita de forma correta, certamente teremos um erro, então é sempre bom verificar na linha do erro se não há erros de digitação.

Seção 7 – Tratamento de Erros

Essa é uma parte muito importante do VBA, pois se refere ao tratamento de erros. Então serão abordadas algumas formas para tratar alguns erros.

Uma informação importante ao aluno é que como o VBA envolve muito a escrita é bem provável que ocorram erros de digitação e com isso erros na execução do código, portanto o primeiro passo quando ocorrer um erro é verificar na linha do erro, se a sintaxe está correta, se os argumentos estão sendo colocados corretamente, se as variáveis estão definidas, se não há algum erro de digitação.

Se após tudo isso ainda estiver dando erro vamos entrar na parte de **debugar** o código, ou seja, passar linha a linha na parte que está gerando o erro e ir acompanhando no ambiente Excel para verificar se o código está correspondendo com a ação que era para acontecer dentro do ambiente Excel. Fazendo isso será possível analisar algumas inconformidades e com isso será possível corrigir o código para que funcione normalmente.

GoTo e Labels

Nesta parte vamos aprender a tratar alguns erros que podem ser gerados pela execução de algumas ferramentas, por variáveis não declaradas entre outras coisas.

Então vamos aprender a como tratar certos tipos de erro para evitar com que a execução do código pare quando isso acontecer.

O **GoTo** funciona para “**pular**” partes do código e ir até um ponto específico sem executar o código pulado, ou seja, ele vai de um ponto A até um ponto B sem executar o que está entre esse caminho. Então isso pode ser usado para evitar a execução de uma parte por conta de um erro.

O **Label** é um rótulo criado para que possamos dizer ao VBA para onde ele vai pular utilizando o **GoTo**, ou seja, **funciona como um endereço** para dizer ao código qual será a próxima linha a ser executada. Portanto vamos fazer a utilização desses dois em conjunto, um para fazer esse pulo e o outro para dizer onde vai parar esse pulo.

```
Sub tratandoErro()

Dim inteiro As Integer

GoTo pular

inteiro = 5

MsgBox (inteiro)

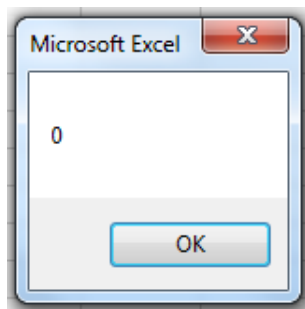
pular:

MsgBox (inteiro)

End Sub
```

Neste código simples podemos ver como funciona na prática o GoTo. É possível observar que temos a declaração da variável com inteiro e logo em seguida temos a estrutura do GoTo, que é composta por ele mais o nome do label. Logo abaixo temos a atribuição do **valor 5** a **variável inteiro** juntamente com um MsgBox para mostrar seu valor.

Em seguida temos o nome **pular:** que é o nosso label, ou seja, é para onde nosso código vai pular (que foi definido anteriormente por **GoTo pular**). E temos outro MsgBox para mostrar o valor da **variável inteiro**.



É possível observar que o valor do MsgBox que foi retornado é igual a 0. Isso quer dizer que o nosso GoTo funciona e realmente pulou a parte em que foi feita a atribuição do valor 5 a variável inteiro e a parte do MsgBox para mostrar esse valor. Então essa função nos permite pular partes do código sem executá-las, o que pode ser útil para diversas atividades, o que dependerá da aplicação.

On Error

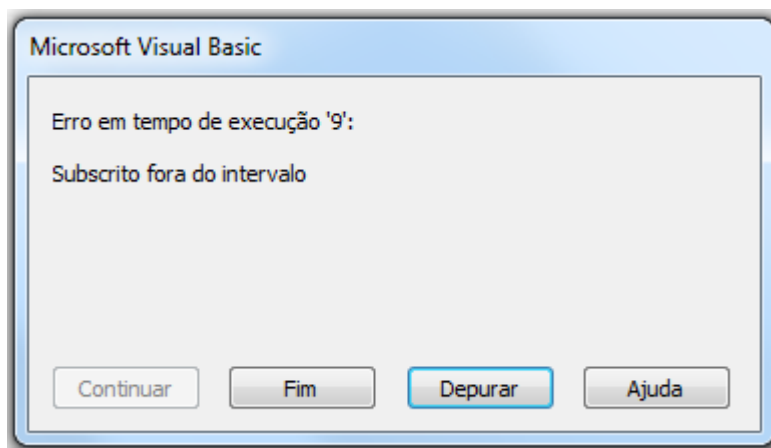
Essa função serve de complemento para a anterior, pois essa funciona para tratar os erros que são gerados pelo código, ou seja, aqueles erros que param a execução do código. Desta forma podemos colocar essa função para o caso de acontecer um erro, portanto ela vai funcionar com o **GoTo**, no entanto será de forma automática quando esse erro for gerado e não simplesmente ir direto ao label indicado.

```
Sub teste1()

    Sheets("Plan5").Activate

End Sub
```

Temos esse código simples que vai nos gerar um erro ao ativar a planilha 5, pois ela não existe. Neste arquivo temos apenas a planilha inicial, então ao tentar selecionar algo que não existe o VBA retorna um erro.



Esse é o erro gerado por tentar ativar uma planilha inexistente. Quando esse erro é gerado ele para a execução do código até que algo seja feito, então para que isso não aconteça é que vamos utilizar a função **On Error**.

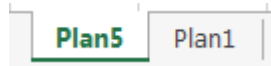
Abaixo temos o código completo para tratar o erro gerado pela seleção de uma planilha que não existe. Neste código vamos tratar esse erro criando uma planilha com o nome "Plan5".

```
Sub teste1()

    On Error GoTo tratamento_erro
    Sheets("Plan5").Activate

tratamento_erro:
    Sheets.Add.Name = "Plan5"
End Sub
```

Neste código em caso de erro o VBA vai ativar essa função e vai automaticamente utilizar a função **GoTo** para o label **tratamento_erro**. Portanto ao executar o código teremos o erro e por consequência a Plan5 será criada.



Temos uma outra forma de executar a função **On Error** é colocar **Resume Next** ao invés do **GoTo**, desta forma vamos indicar que o código deve **ignorar o erro e continuar** executando o código normalmente.

```
Sub teste1()

On Error Resume Next
    Sheets("Plan10").Activate

End Sub
```

E a última forma é uma forma que é o padrão do VBA. Essa forma é colocar **GoTo 0**, desta forma o VBA vai continuar mostrando o erro normalmente e irá parar com a execução do código.

```
Sub teste1()

On Error GoTo 0
    Sheets("Plan10").Activate

End Sub
```

Uma informação importante sobre os labels é que caso o código não tenha erro o código que está dentro do label será executado como se fosse uma linha de código normal, portanto será necessário colocar **Exit Sub** antes do label para que o código saia da Sub e não execute esse label caso não ocorram erros.

```
Sub teste1()

On Error GoTo tratamento_erro
    Sheets("Plan5").Activate

Exit Sub
tratamento_erro:
    Sheets.Add.Name = "Plan5"

End Sub
```

Desta forma podemos evitar de fazer um tratamento de erro quando não temos um erro propriamente dito. Caso aconteça algum erro o código será levado até o label informado e irá pular a parte de sair da sub e irá executar o tratamento como foi escrito.

Para o caso de querer voltar ao código após o tratamento de erro podemos inserir duas opções após esse tratamento: **Resume** ou **Resume Next**. A primeira opção vai voltar o código exatamente na linha que gerou o erro e vai executá-la novamente enquanto a segunda opção vai retornar ao código na linha seguinte ao erro.

<code>Sub teste1()</code>	<code>Sub teste1()</code>
<code>On Error GoTo tratamento_erro</code>	<code>On Error GoTo tratamento_erro</code>
<code> Sheets("Plan5").Activate</code>	<code> Sheets("Plan5").Activate</code>
<code> Exit Sub</code>	<code> Exit Sub</code>
<code>tratamento_erro:</code>	<code>tratamento_erro:</code>
<code> Sheets.Add.Name = "Plan5"</code>	<code> Sheets.Add.Name = "Plan5"</code>
<code> Resume</code>	<code> Resume Next</code>
<code>End Sub</code>	<code>End Sub</code>

Desta forma não vamos encerrar a execução do código para tratar o erro e podemos trata-lo e voltar ao código, executando ou não a linha que gerou o erro novamente. Utilizando apenas o **Resume** após o tratamento de erro temos que tomar cuidado quando o erro não for corrigido, pois podemos entrar em um **loop infinito**. Então caso o erro não seja tratado estamos dizendo ao código para executar a linha que gerou o erro, e como não foi tratado vai entrar novamente no tratamento e isso vai acontecer infinitamente.

Outra informação importante é que ao colocar **On Error** no código, todos os erros serão tratados da mesma forma, ou seja, qualquer erro que aconteça irá para o label de tratamento, portanto é importante verificar se os erros que poderão acontecer estarão sendo tratados de forma correta, pois há a possibilidade de ocorrer um erro de outra natureza e aplicarmos um tratamento do erro de planilha para esse erro aleatório.

Então podemos estar colocando informações desnecessárias e tratando erros de forma inadequada por conta de um outro erro que foi gerado. De qualquer forma é possível colocar **On Error mais de uma vez dentro do código**, desta forma é possível tratar diferentes erros de diferentes formas.


```
Sub teste3()  
  
On Error GoTo tratamento_erro  
    Sheets("Plan6").Activate  
  
On Error GoTo tratamento_erro2  
    Sheets("Plan7").Activate  
    Exit Sub  
  
tratamento_erro:  
    Sheets.Add.Name = "Plan6"  
    Resume Next  
  
tratamento_erro2:  
    Sheets.Add.Name = "Plan7"  
End Sub
```

Neste código temos dois erros de seleção de planilhas, no entanto temos dois tratamentos diferentes. Ao selecionar a planilha 6 se ela não existir vamos tratar esse erro criando a **planilha 6**, no entanto se não houver a **planilha 7** vamos criá-la e não criar outra **planilha 6** que já existe. Então são dois tratamentos distintos para dois erros distintos.

Quinta Ferramenta – Compilação Despadronizada

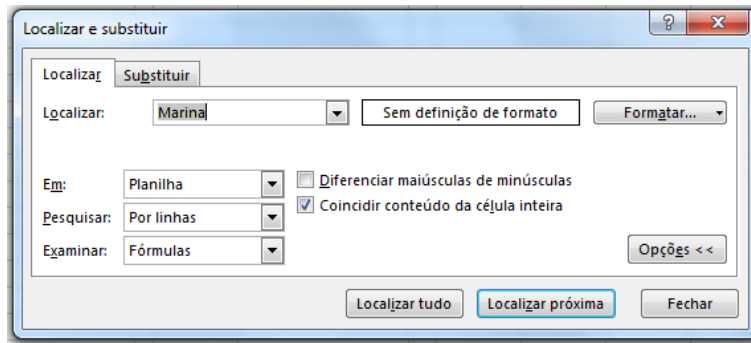
Nesta parte vamos criar uma ferramenta utilizando uma macro com a ferramenta **Localizar** do Excel para que possamos procurar os alunos em uma planilha que está desformatada (não está formatada ou organizada de forma adequada) para que seja possível organizar esses dados de maneira que fique mais fácil de visualizar.

	A	B	C	D	E	F	G	H	I	J	K
1	Carlos	5		Joao	7					Paulo	3
2	carlos@gmail.com			joao@gmail.com						paulo@gmail.com	
3											
4											
5	Joana	8		Ana	10					Gabriel	10
6	joana@gmail.com			ana@gmail.com						gabriel@gmail.com	
7											
8											
9	Marina	9		Marcos	8						
10	marina@gmail.com			marcos@gmail.com							
11											

Portanto o objetivo é gravar uma macro utilizando a ferramenta localizar para verificar a escrita dessa ferramenta e em seguida organizar os dados dentro de uma segunda planilha.

	A	B	C
1	Codigo	Aluno	Nota
2	123	Fulano	
3	124	Joana	
4	125	Marina	
5	126	Joao	
6	127	Ana	
7	128	Marcos	
8	129	Paulo	
9	130	Gabriel	
10			

Abaixo temos um breve passo a passo da gravação da macro utilizando a opção de Localizar do ambiente Excel para que possamos descobrir qual o código dessa ferramenta para que possamos utilizá-lo no código.



Ao gravar a macro vamos colocar um dos nomes que estão na primeira aba (que se encontra desorganizada) e vamos também marcar a caixa de **Coincidir conteúdo da célula inteira**, para que o Excel encontre apenas a célula que possui o nome e não o e-mail.

Abaixo temos o código obtido após a gravação da macro.

```
Sub Macro1()

    Cells.Find(What:="Marina", After:=ActiveCell, LookIn:=xlFormulas, LookAt _
        :=xlWhole, SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:= _
        False, SearchFormat:=False).Activate

End Sub
```

Tendo o código de procura vamos construir nosso código para verificar se a pessoa em questão tem nota na primeira aba, caso tenha vamos registrar a nota, caso contrário vamos passar para o próximo.

```
Sub Macro1()

    Dim range1 As Range, cell As Range
    Dim aba_compilado As String, aba_notas As String
    Dim encontrado As Integer
    aba_compilado = "Compilado"
    aba_notas = "Notas Alunos"

    Sheets(aba_compilado).Activate
    Set range1 = Range("A1", Range("A1").End(xlDown))

    For Each cell In range1
        encontrado = 1
        Sheets(aba_notas).Activate

        On Error GoTo tratamento
        Cells.Find(What:=cell.Offset(0, 1), After:=ActiveCell, LookIn:=xlFormulas, LookAt _
            :=xlWhole, SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:= _
            False, SearchFormat:=False).Activate

        If encontrado = 1 Then
            cell.Offset(0, 2).Value = Selection.Offset(0, 1).Value
        End If
    Next

    Sheets(aba_compilado).Activate

Exit Sub

tratamento:
    encontrado = 0
    Resume Next

End Sub
```

Este é um código simples que vai passar por toda a lista de alunos e vai procurar o nome de cada aluno na primeira aba, ao encontrar será registrado o valor da nota, caso não encontre o valor o código irá gerar um erro. Temos a função **On Error** para tratar este erro e então voltar para a execução do código.

Este código permite ao aluno verificar a utilização de uma nova ferramenta, que é a de procurar, e também observar a utilização do tratamento de erro dentro do código.

Seção 8 – Formulários e Boxes Completos

Nesta parte vamos aprender a parte de formulário e as caixas dentro do ambiente VBA que vai proporcionar uma melhor interação do usuário com o programa, principalmente quando precisamos que o usuário preencha esse formulário com algumas informações para que o programa funcione corretamente.

Dentro do formulário vamos poder colocar várias caixas (boxes) que serão utilizadas dentro do ambiente Excel. Será ensinado também a utilizar cada uma dessas boxes e suas finalidades para que o aluno possa criar um formulário completo com todas as ferramentas.

MsgBox Completo

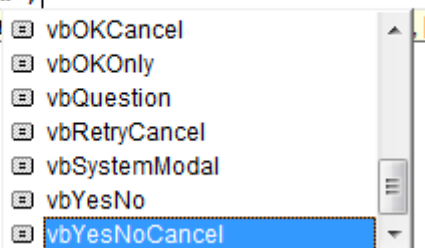
Nesta parte vamos aprender a utilização do MsgBox por completo. Até o momento só utilizamos essa ferramenta para mostrar uma mensagem. Agora vamos ter várias opções para utilizar essa ferramenta, podendo até colocar botões dentro dela, o que pode dar mais funcionalidades ao código e deixa a execução de forma mais apresentável dependendo do caso.

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

É possível observar que ao abrir o parêntese do **MsgBox** temos vários argumentos. O primeiro argumento é o que utilizamos até o momento, que é a mensagem que irá aparecer dentro do **MsgBox**.

O segundo argumento é referente aos botões disponíveis para essa caixa.

```
Sub mensagem()  
  
msgbox ("Mensagem",  
MsgBox(Prompt, [Buti  
End Sub
```



Neste exemplo vamos utilizar a última opção que vai gerar uma caixa de mensagem com as opções de **Yes** (sim), **No** (não) e **Cancel** (Cancelar). Ao colocar essa opção o VBA faz com que o usuário tenha que atribuir esse MsgBox

a uma variável, pois essas seleções de botões retornam um valor. Cada botão tem um valor padrão de retorno.

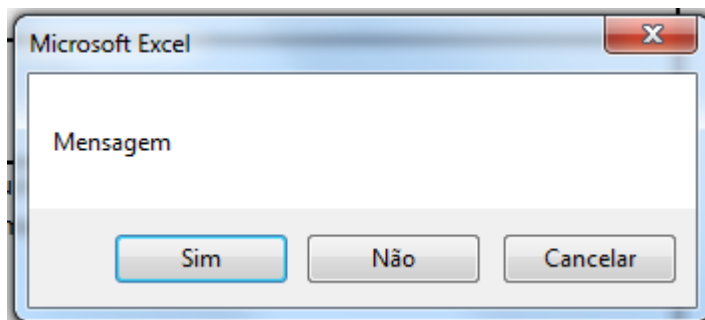
Valor Retornado

Constant	Value
OK	1
Cancel	2
Abort	3
Retry	4
Ignore	5
Yes	6
No	7

Sabendo disso teremos que atribuir uma variável ao MsgBox para que possamos posteriormente dizer ao nosso código o que será feito em cada um dos casos possíveis para aquele MsgBox.

```
Sub mensagem()  
  
Dim valor As Integer  
valor = MsgBox("Mensagem", vbYesNoCancel)  
  
End Sub
```

Ao executar o código temos o seguinte resultado.

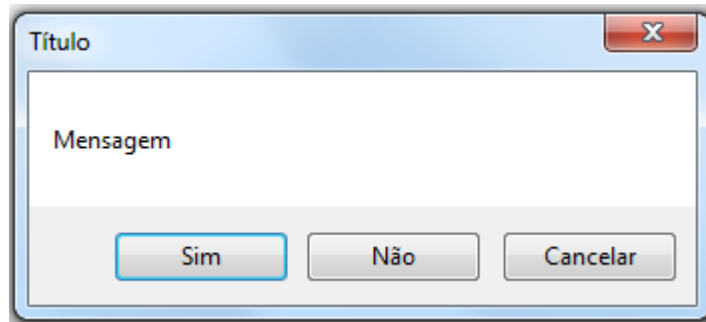


Então temos a nossa mensagem como foi utilizado até o momento. E agora temos as três opções de botões e ao pressionar cada um deles nossa **variável valor** vai assumir um valor de acordo com a tabela que foi mostrada. Esses valores são padrões, portanto não vão mudar. Então se pressionarmos o botão **sim** o valor atribuído a variável será **6**, no caso do **não** o valor atribuído será **7** e no caso do **cancelar** o valor será igual a **2**.

O próximo argumento para o MsgBox é o título, ou seja, é possível alterar o título da caixa de mensagem.

```
Sub mensagem()  
  
Dim valor As Integer  
valor = MsgBox("Mensagem", vbYesNoCancel, "Título")  
  
End Sub
```

Ao executar o código podemos observar que o título da caixa foi modificado para o nome que colocamos como título.



Para o MsgBox vamos ver apenas esses 3 argumentos, que se referem a mensagem a ser exibida, aos tipos de botões que irão aparecer para o usuário e a mudança do título da caixa de mensagem.

Temos uma outra opção no segundo argumento para colocar os tipos de botões que serão mostrados ao usuário. Ao invés de selecionar na lista podemos colocar números referentes a essas opções. Para isso podemos utilizar a tabela abaixo como base para essa seleção.

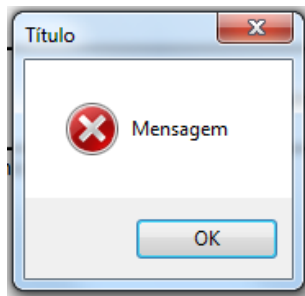
MsgBoxStyle		
Nome	Valor	Descrição
OKOnly	0	Displays OK button only.
OKCancel	1	Displays OK and Cancel buttons.
AbortRetryIgnore	2	Displays Abort, Retry, and Ignore buttons.
YesNoCancel	3	Displays Yes, No, and Cancel buttons.
YesNo	4	Displays Yes and No buttons.
RetryCancel	5	Displays Retry and Cancel buttons.
Critical	16	Displays Critical Message icon.
Question	32	Displays Warning Query icon.
Exclamation	48	Displays Warning Message icon.
Information	64	Displays Information Message icon.
DefaultButton1	0	First button is default.
DefaultButton2	256	Second button is default.
DefaultButton3	512	Third button is default.
ApplicationModal	0	Application is modal. The user must respond to the message box before continuing work in the current application.
SystemModal	4096	System is modal. All applications are suspended until the user responds to the message box.
MsgBoxSetForeground	65536	Specifies the message box window as the foreground window.
MsgBoxRight	524288	Text is right-aligned.
MsgBoxRtlReading	1048576	Specifies text should appear as right-to-left reading on Hebrew and Arabic systems.

Desta forma podemos utilizar os números ao invés de selecionar as opções. Com o uso frequente o usuário pode até gravar os números das opções que mais utiliza e facilitar o processo.

É possível observar que temos ícones também dentro dessa tabela, isso quer dizer que podemos colocar um ícone de crítico utilizando o número 16.

```
Sub mensagem()  
  
Dim valor As Integer  
valor = MsgBox("Mensagem", 16, "Título")  
  
End Sub
```

Executando esse código temos o seguinte resultado.

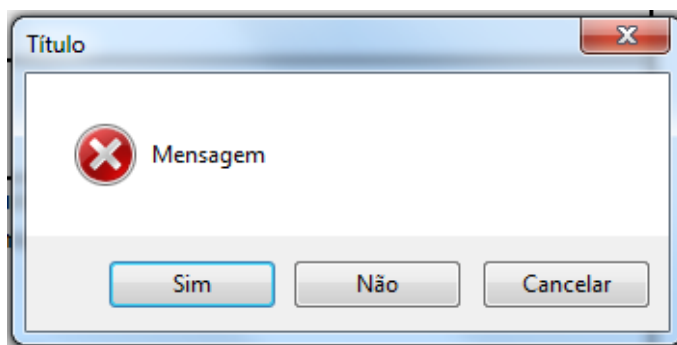


Desta forma o Excel retorna a mensagem com um ícone de crítico/erro, portanto é uma forma de dar uma prioridade a essa caixa de mensagem.

Existe uma forma também para utilizarmos mais de uma opção, como por exemplo a opção utilizada anteriormente de **sim**, **não** e **cancelar** (número 3) com essa opção de crítico. Para isso vamos colocar a seguinte expressão “**3 or 16**”, desta forma teremos as duas opções juntas na mesma caixa.

```
Sub mensagem()  
  
Dim valor As Integer  
valor = MsgBox("Mensagem", 3 Or 16, "Título")  
  
End Sub
```

Ao executar o código podemos verificar que as duas opções serão mescladas em uma única caixa de mensagem. Então podemos utilizar esse tipo de combinação para complementar e priorizar certas mensagens.



Na outra parte da lista temos uma área de botão padrão, isso quer dizer que podemos deixar um botão previamente selecionado como padrão para quando o usuário pressionar **enter** ele seja pressionado.

No exemplo acima o botão padrão é o primeiro, portanto se pressionarmos **enter** assim que a caixa for aberta esse botão será pressionado. Isso acontece em boa parte dos programas principalmente para evitar erros se for para deletar algum arquivo ou atividade similar. Então serve como uma proteção no caso de o usuário agir por impulso é já pressionar a tecla.

Para acrescentar essa opção as outras basta seguir o mesmo procedimento acrescentar “**or**” e o número da opção desejada.

Vamos agora analisar a última parte da tabela, pois ela possui algumas informações importantes.

ApplicationModal	0 Application is modal. The user must respond to the message box before continuing work in the current application.	
SystemModal	4096 System is modal. All applications are suspended until the user responds to the message box.	
MsgBoxSetForeground	65536 Specifies the message box window as the foreground window.	
MsgBoxRight	524288 Text is right-aligned.	Outros
MsgBoxRtlReading	1048576 Specifies text should appear as right-to-left reading on Hebrew and Arabic systems.	

As duas primeiras opções são modais, ou seja, são modos de funcionamento da caixa de mensagem. A primeira opção (0) diz que o usuário tem que responder a caixa antes de continuar trabalhando na aplicação atual. A segunda opção (4096) diz que todas as aplicações serão suspensas até que o usuário responda a caixa, ou seja, os outros programas também não poderão ser utilizados até que a caixa seja respondida.

A terceira opção (65536) faz com que essa caixa de mensagem vá para a frente, não importando a quantidade de abas de quantos programas estiverem abertos, ou seja, a caixa irá abrir por cima de deles para que o usuário a veja.

As duas últimas opções são referentes ao alinhamento do texto. A opção (524288) é para alinhar o texto a direita e a última (1048576) é para que as informações apareçam trocadas da direita para a esquerda, então é como se fosse um espelho, mas a escrita fica correta, apenas os ícones vão mudar de lugar inclusive o x para fechar a caixa.

InputBox

Nesta parte vamos utilizar o **InputBox (caixa de entrada)** que não foi utilizado até o momento, mas é uma ferramenta bem interessante para complementar os códigos.

Essa ferramenta nos permite inserir alguma informação dentro da caixa que será aberta e poderá ser utilizada dentro de um código para complementá-lo, ou seja, é possível pedir ao usuário essa informação para continuar.

Type

Valor	Significado
0	Uma fórmula
1	Um número
2	Texto (uma sequência)
4	Um valor lógico (True ou False)
8	Uma referência a células, como um objeto Range
16	Um valor de erro, como #N/D
64	Uma matriz de valores

Essas são as informações que poderão ser inseridas dentro dessa caixa de entrada que irá aparecer.

```
Sub InserirPopUp()
    valor = application.inputbox(
End Sub
```

`InputBox(Prompt As String, [Title], [Default], [Left], [Top], [HelpFile], [HelpContextID], [Type])`

É possível observar que temos alguns argumentos que podem ser utilizados. É importante lembrar que os argumentos entre chaves [] não são obrigatórios, ou seja, não precisam ser colocados para que a ferramenta funcione. Neste caso apenas a mensagem é obrigatória.

O primeiro argumento é a mensagem que irá aparecer, assim como acontece no MsgBox. O segundo é o título da caixa. O **default** é o que aparecerá no InputBox, ou seja, é um valor padrão inicial e o outro argumento que vamos utilizar é o **type** que é o tipo de informação mostrado na tabela acima.

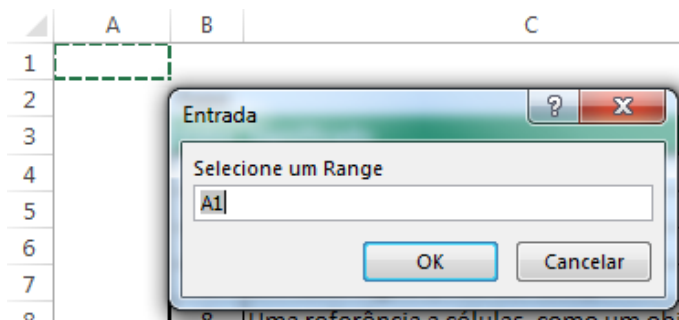
Os argumentos de **Left** e **Top** são argumentos para modificar onde a caixa irá aparecer dentro da tela, no entanto para utilizar essa opção é necessário utilizar o **InputBox** sem o **application** escrito antes. Então esses argumentos vão mudar os nomes para **XPos** e **YPos**, que são as posições em **X** e **Y** de onde a caixa vai estar posicionada.

Os outros dois argumentos são para vincular um arquivo de ajuda para aquele InputBox, mas não iremos utilizar essas opções.

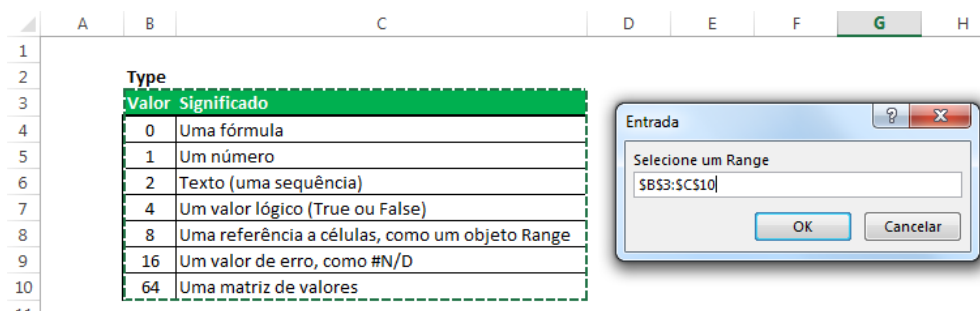
```
Sub InserirPopUp()  
    valor = Application.InputBox("Selecione um Range", , "A1", Type:=8)  
End Sub
```

Este é o nosso primeiro código, é simples mas mostra como a ferramenta funciona. É possível observar que temos duas maneiras de passar para outro argumento sem alterar ele, a primeira é apenas colocando as vírgulas sem escrever nada naquele argumento, a segunda é utilizar o nome do argumento a ser alterado e acrescentar `:=` para que seja possível ir diretamente para aquele argumento e alterá-lo.

Ao executar o código teremos o seguinte resultado.



Temos nossa caixa de InputBox com o valor inicial sendo a célula A1 que foi colocado no argumento default. Com essa caixa aberta será possível selecionar qual range o usuário quer apenas clicando e arrastando com o mouse para selecionar.



Assim que a seleção for feita o valor dentro da caixa será alterado automaticamente para o intervalo selecionado.

Ao pressionar OK o código continuará com sua execução e a variável em questão irá receber o valor inserido. É importante lembrar que ao declarar a variável que irá receber esse valor devemos fazer essa declaração de acordo com o tipo específico da variável, especialmente se tiver **Option Explicit** ativo.

```
Sub InserirPopUp()

Dim valor As Range
Set valor = Application.InputBox("Selecione um Range", , "A1", Type:=8)

End Sub
```

Então para uma declaração de range podemos fazer a declaração desta forma. Como range e ao colocar a variável devemos colocar o **set** antes, pois se trata de um range, então esse tipo de variável não pode ser colocado apenas a variável e o igual. Caso isso seja feito irá retornar um erro.

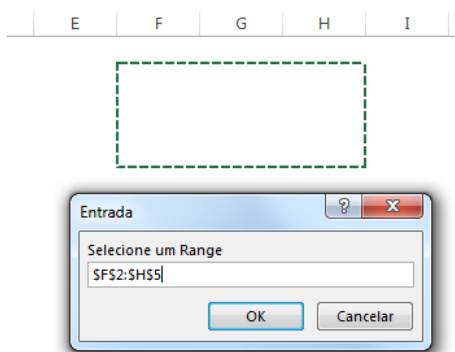
```
Sub InserirPopUp()

Dim valor As Range, cell As Range
Set valor = Application.InputBox("Selecione um Range", , "A1", Type:=8)

For Each cell In valor
    cell.Value = 1
Next

End Sub
```

Para verificar a funcionalidade vamos utilizar esse código simples para preencher todas as células selecionadas do range com o valor 1.



Ao executar o código com essa seleção teremos o seguinte resultado.

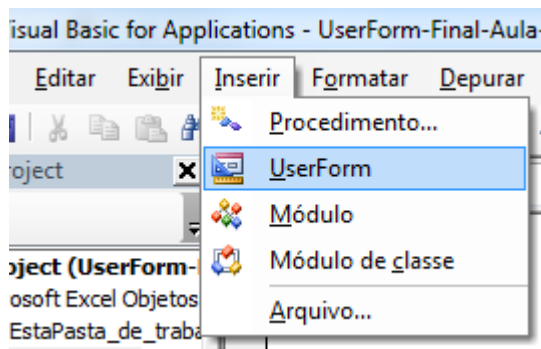
F	G	H	I
1	1	1	
1	1	1	
1	1	1	
1	1	1	

Desta forma temos uma ferramenta que nos permite interagir com o código podendo inserir dados para que o programa continue com sua execução.

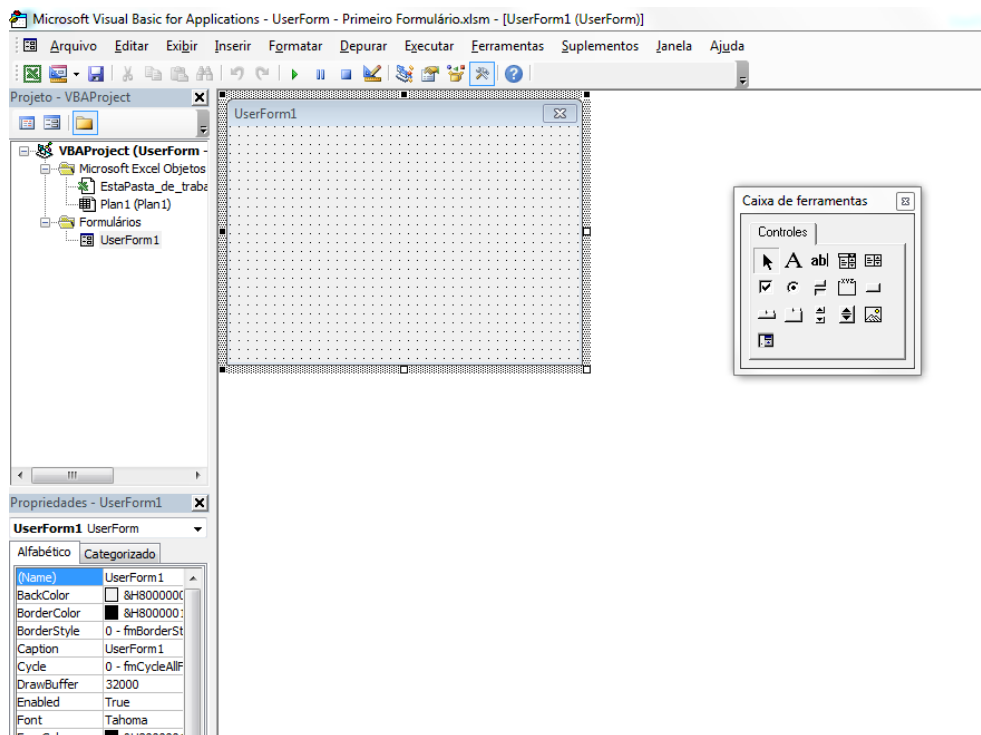
UserForm – Primeiro Formulário

Vamos aprender uma parte importante do VBA que é o **formulário**. Esse formulário permite com que o usuário crie uma janela (formulário) com caixas de textos, botões, entre outras funcionalidades, com o objetivo de deixar o código com uma aparência de programa, ou seja, o código irá ficar por trás dessas opções e a partir das informações colocadas ele irá executar alguma ação. Exatamente como acontece em um programa.

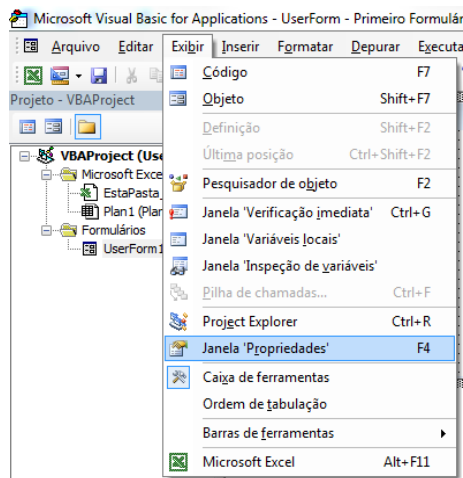
Para a criação do formulário vamos entrar no ambiente VBA e inserir um formulário. Essa opção se encontra no mesmo local onde inserimos um módulo.



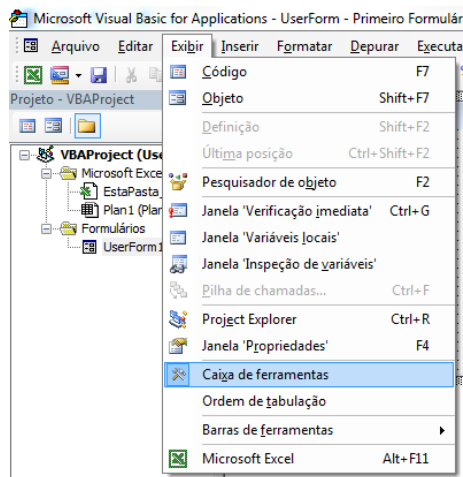
Ao inserir o formulário temos a seguinte tela onde serão colocadas as informações referentes ao formulário.



É possível observar que além do formulário temos a **caixa de propriedades** no canto inferior esquerdo, que será bastante utilizada. Ela pode ser habilitada conforme imagem abaixo ou utilizando o atalho da **tecla F4**.

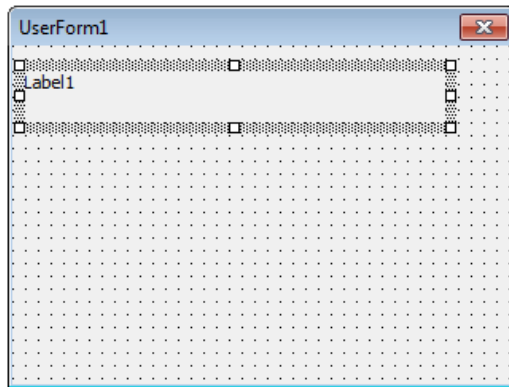


Para a caixa de ferramentas podemos habilitá-la na mesma guia **Exibir**.

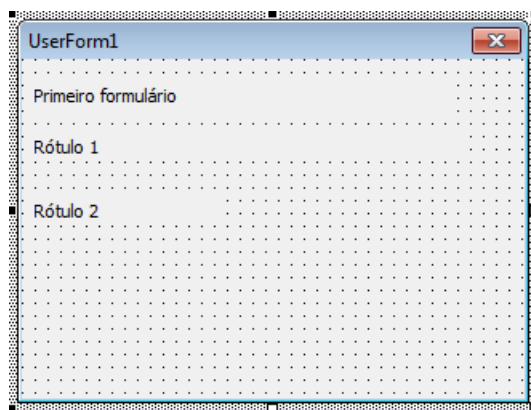


Essa caixa de ferramenta vai permitir ao usuário adicionar os controles dentro do formulário. São eles: **caixa de texto**, **botão de comando**, **rótulo**, **caixa de combinação**, entre outras ferramentas. É possível utilizar várias delas em conjunto para criar um formulário mais robusto e completo que vai se basear na necessidade do usuário.

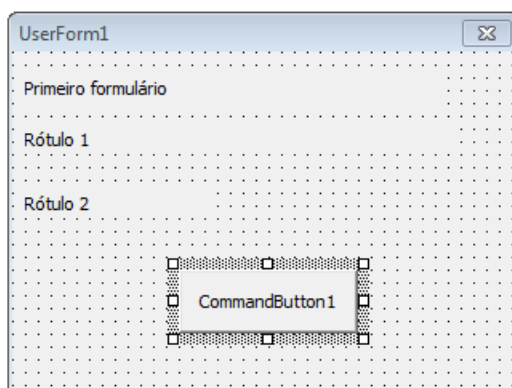
Vamos iniciar com a criação de um formulário simples. Para isso vamos utilizar a ferramenta **Rótulo**, que é a ferramenta com a letra **A**. Para usá-la basta selecioná-la, clicar e arrastar no formulário.



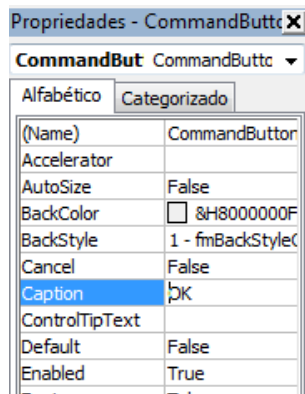
Para alterar o texto deste rótulo basta **clicar uma única vez na área selecionada** e modificar o texto. Podem ser adicionados diversos rótulos com diversos textos, tudo vai depender da necessidade para a atividade em questão.



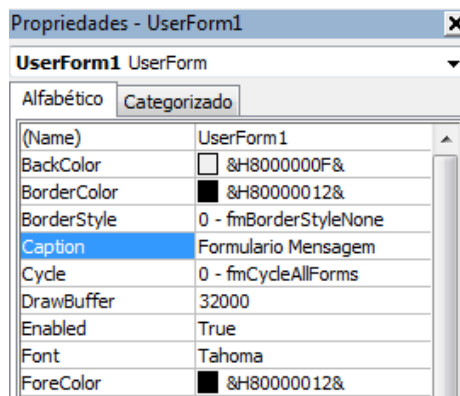
Feito isso vamos agora inserir um botão de comando.



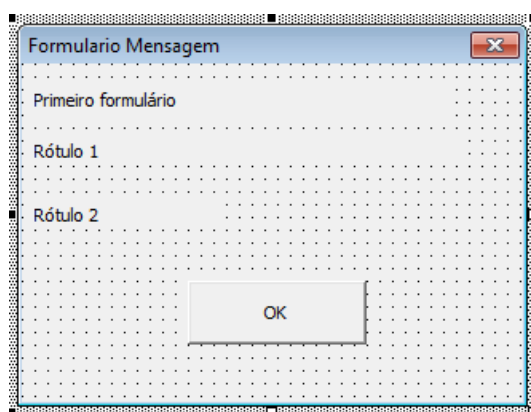
Assim como no rótulo podemos alterar o nome que aparece no botão, mas neste caso temos que mudá-lo utilizando a **caixa de propriedades**. Vamos alterar o campo **Caption** que é o nome que aparece ao usuário.



Desta forma o nome do botão será alterado para OK. É importante ressaltar que todos os objetos vão ter suas próprias propriedades, então ao selecionar o formulário por exemplo ele terá suas propriedades e será possível alterar seu nome, portanto para alterar uma propriedade basta selecionar o item desejado para efetuar a alteração.



Desta forma teremos o seguinte formulário.



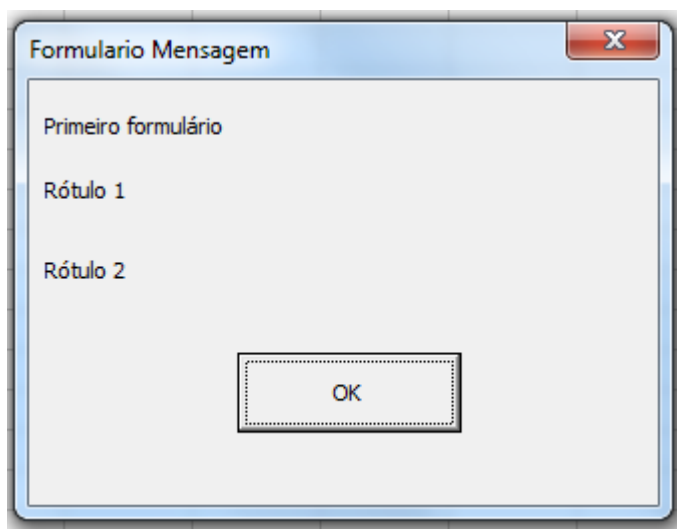
Os nomes foram modificados e já podem ser observados dentro do formulário, tanto do botão quanto do título do formulário.

Como o formulário não é uma macro não vamos conseguir executá-lo na parte de macros, no entanto é possível chamar esse formulário a partir de uma macro. Então será criada uma macro para fazer essa chamada.

```
Sub exibir_formulario()  
    UserForm1.Show  
End Sub
```

Estamos utilizando o nome **UserForm1**, pois é esse o nome que se encontra na parte (name) das propriedades do formulário, ou seja, esse é o nome em que o VBA se refere a cada um dos objetos. Então para chamar ou fazer referência a algum objeto do formulário, seja caixa de texto, botões, entre outros vamos utilizar esse nome.

Em seguida foi utilizada a propriedade **.Show** que serve para mostrar o formulário. Feito isso podemos executar a macro que irá fazer a chamada e irá mostrar o formulário.



Esse é o formulário que foi criado, no entanto é possível observar que ao clicar no **botão OK** nada acontece. Isso ocorre porque temos que escrever um código para atribuir uma ação a esse botão quando for selecionando, pois nada que é adicionado ao formulário tem um código atribuído. Todos esses códigos deverão ser atribuídos pelo usuário.

Para atribuir essa função ao botão vamos voltar ao ambiente VBA na parte de formulário e vamos dar um duplo clique no **botão OK**.



É possível observar que já temos uma macro criada e em cima dessa macro temos duas informações escritas, a primeira delas é **CommandButton1** e a segunda é **Click**. A primeira informação faz referência ao nome do botão (da mesma forma que foi feito com o formulário anteriormente), enquanto a segunda informação nos diz que aquela macro será executada ao clicar nesse botão, ou seja, toda vez que esse botão for pressionado ele irá executar essa macro.

```
Private Sub CommandButton1_Click()  
    UserForm1.Hide  
End Sub
```

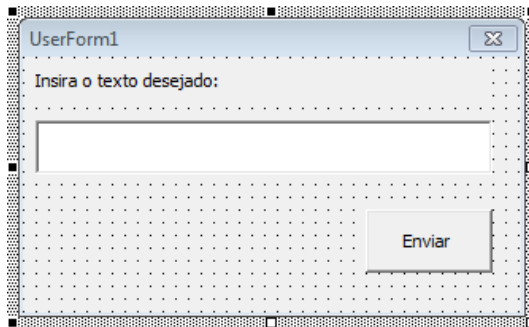
Esse código irá esconder o formulário quando o botão OK for pressionado, então ao executar a primeira macro o nosso formulário será aberto e ao pressionar OK ele será escondido.

É importante ressaltar que ao dar um duplo clique em qualquer uma das ferramentas dentro do formulário o VBA irá abrir uma nova janela para que seja possível escrever um código para essa determinada ferramenta.

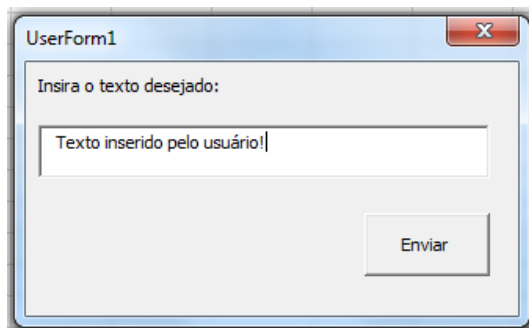
TextBox

Nesta parte vamos aprender a utilizar a **caixa de texto** dentro do formulário, ela difere do rótulo porque o rótulo é uma mensagem que ficará escrita dentro do formulário enquanto a caixa de texto é para que o usuário possa escrever dentro dela.

A inserção da ferramenta funciona igual as que vimos até agora, basta selecionar ela e em seguida clicar e arrastar para modificar o tamanho da caixa.



Esse é o novo formulário criado com a caixa de texto e um botão escrito “**enviar**”, que será utilizado para guardar o texto inserido pelo usuário. Ao executar o código teremos o seguinte resultado.

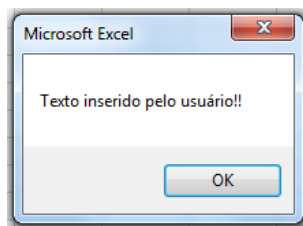


Desta forma o usuário poderá escrever alguma informação e posteriormente poderemos obter a informação dessa caixa de texto através de um código em VBA.

Vamos então atribuir um código ao botão enviar para que possamos obter o que está escrito dentro da caixa de texto, colocar essa informação em um MsgBox e salvar essa informação dentro de uma célula.

```
Private Sub CommandButton1_Click()  
    Dim texto As String  
  
    texto = TextBox1.Value  
  
    UserForm1.Hide  
  
    MsgBox (texto)  
  
    Range("A1").Value = texto  
End Sub
```

Vamos executar o código, preencher a caixa e pressionar o botão “enviar”.



Temos um MsgBox mostrando a mensagem inserida pelo usuário na caixa de texto e abaixo temos essa informação gravada na **célula A1**.

	A	B	C
1	Texto inserido pelo usuário!!		
2			
3			

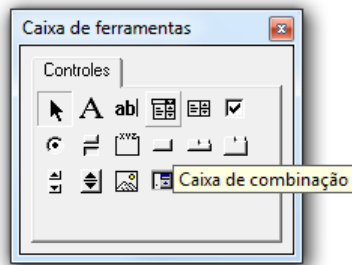
Portanto com o **UserForm** podemos também obter as informações que são inseridas dentro dele para utilizar em outras aplicações. Neste caso foi possível observar que podemos tratar essas informações como variáveis, então podemos pedir ao usuário para escrever ou cadastrar um produto que pode ser registrado em uma planilha posteriormente.

As possibilidades são diversas, e como sempre tudo vai depender da aplicação do usuário, portanto temos diversas possibilidades e formas de criar ou modificar um formulário para que atenda a necessidade atual do usuário.

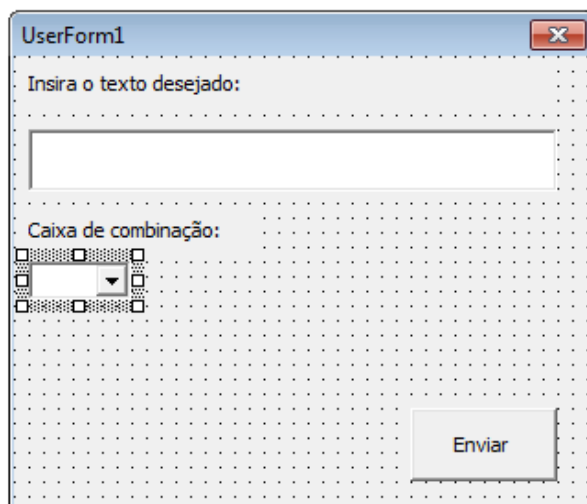
ComboBox

Nesta parte vamos ver a **caixa de combinação**, que funciona como a lista suspensa do Excel quando fazemos a **validação de dados**.

Para isso vamos seleccionar a caixa de combinação e coloca-la dentro do formulário.



É o quarto ícone da caixa de ferramentas.



Assim como na lista do Excel, dentro do VBA podemos seleccionar os dados que irão aparecer na caixa de combinação de duas formas: a primeira delas que é a mais simples é colocando a fonte onde serão obtidos esses dados na caixa da opção **RowSource**. A outra opção é acrescentando esses itens através do código em VBA.

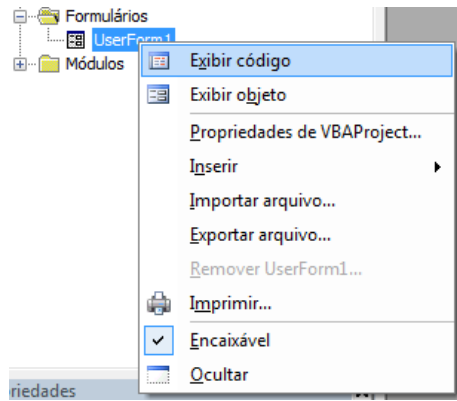
MouseIcon	(Nenhum)
MousePointer	0 - fmMousePointerDefault
RowSource	
SelectionMargin	True
ShowDropButtonWhen	2 - fmShowDropButtonWhen
SpecialEffect	2 - fmSpecialEffectSunken
Style	0 - fmStyleDropDownComb

Então vamos colocar os números de 1 a 5 no intervalo de células da **A3** até a **A7**. Desta forma podemos colocar no RowSource a fonte dos nossos dados para que eles apareçam dentro da nossa lista quando o formulário estiver aberto. Para isso podemos colocar tanto **A3:A7** na opção quanto **Plan1!A3:A7**. Essa última opção é mais recomendada, pois a planilha pode ter mais de uma aba, e com isso poderá pegar valores errados, desta forma irá pegar o valor da aba indicada sem que ocorram erros.

MouseIcon	(Nenhum)
MousePointer	0 - fmMousePointerDefault
RowSource	Plan1!A3:A7
SelectionMargin	True
ShowDropButtonWhen	2 - fmShowDropButtonWhen
SpecialEffect	2 - fmSpecialEffectSunken
Style	0 - fmStyleDropDownComb

Ao executar o código é possível observar que a caixa de combinação já está funcionando desta forma.

Como vamos utilizar essa atribuição da lista pelo código vamos deletar essas informações do **RowSource** e vamos entrar no código do formulário. Para isso basta clicar com o botão direito no **UserForm1** e selecionar a opção para **Exibir Código**.



Feito isso será aberta a janela com o código previamente escrito com a ação do **botão enviar**.

```
Private Sub CommandButton1_Click()
    Dim texto As String

    texto = TextBox1.Value

    UserForm1.Hide

    MsgBox (texto)

    Range("A1").Value = texto
End Sub
```

Da mesma forma que foi criado um **evento** ao clicar no botão enviar, será criado outro para que possamos obter os valores a serem carregados na caixa de combinação. Esse evento será vinculado ao formulário, ou seja, quando o formulário for ativado este código será executado. Para que isso aconteça vamos colocar nas opções logo acima da caixa de texto do código **UserForm** e **Activate**.



Feito isso temos um código iniciado para escrever as ações que serão executadas quando o formulário for ativado. Se abrirmos o formulário mais de uma vez quando o programa estiver aberto será possível carregar esses dados sempre. Essa ação vai nos gerar um problema, mas vamos resolver logo adiante.

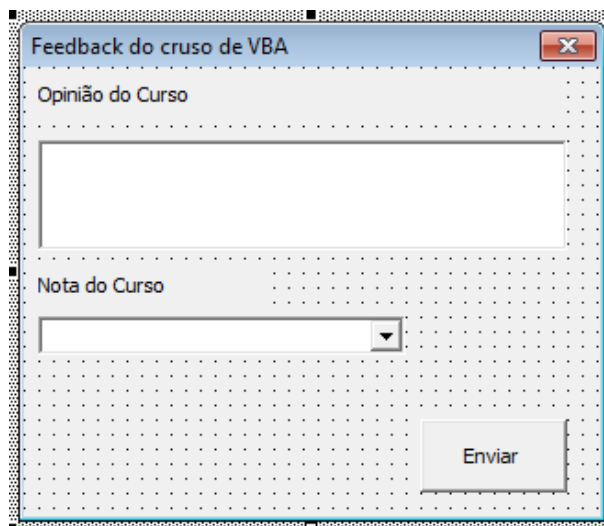
```
Private Sub UserForm_Activate()

End Sub
```

Vamos colocar o seguinte código para que seja possível adicionar os itens dentro da caixa de combinação. Vamos utilizar os conhecimentos que já aprendemos como **With** e as propriedades dos objetos.

```
Private Sub UserForm_Activate()  
  
    With UserForm1.ComboBox1  
        .AddItem ("1 - Péssimo")  
        .AddItem ("2 - Ruim")  
        .AddItem ("3 - Regular")  
        .AddItem ("4 - Bom")  
        .AddItem ("5 - Excelente")  
    End With  
  
End Sub
```

Podemos também alterar os rótulos do formulário para começar a dar forma a ele e deixá-lo com as informações necessárias para que o usuário preencha as caixas de forma correta.

The image shows a VBA UserForm window titled "Feedback do curso de VBA". The window has a standard Windows-style title bar with a close button (X) in the top right corner. The form itself is set against a dotted grid background. It is divided into two main sections. The first section, labeled "Opinião do Curso", features a large, empty text box for user input. The second section, labeled "Nota do Curso", contains a dropdown menu with a small arrow icon on the right side. At the bottom right of the form is a button labeled "Enviar".

Feito isso podemos criar um formulário para receber a opinião do curso e as notas do curso dadas pelos alunos. Agora vamos executar o código principal para verificar se ao abrir o formulário teremos a lista que foi descrita dentro do código de ativação dele como todos os itens que escritos no código.

The image shows a VBA UserForm titled "Feedback do curso de VBA". It contains two input fields: a text box labeled "Opinião do Curso" and a dropdown menu labeled "Nota do Curso". The dropdown menu is open, showing a list of ratings: "1 - Péssimo", "2 - Ruim", "3 - Regular", "4 - Bom", and "5 - Excelente". The "5 - Excelente" option is currently selected and highlighted in blue. To the right of the dropdown is a button labeled "Enviar".

É possível observar que temos todos os itens do código que foram adicionados. Agora vamos registrar também o valor selecionado pelo usuário na caixa de combinação na **célula A2**.

```
Private Sub CommandButton1_Click()
    Dim texto As String, nota As String

    texto = TextBox1.Value
    nota = ComboBox1.Value

    UserForm1.Hide

    MsgBox ("Feedback enviado com sucesso!!")

    Range("A1").Value = texto
    Range("A2").Value = nota
End Sub
```

Desta forma podemos obter tanto o feedback do aluno em relação ao curso quanto a nota dada por ele e registrar esses dados. Ao clicar em enviar os dados serão registrados nas **células A1** e **A2** e o usuário receberá uma mensagem de que o feedback foi enviado com sucesso.

Feedback do curso de VBA

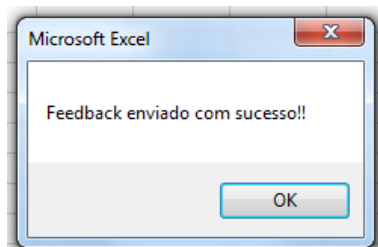
Opinião do Curso

O curso é muito bom e estou aprendendo bastante!

Nota do Curso

5 - Excelente

Enviar



	A	B	C	D	E
1	O curso é muito bom e estou aprendendo bastante!				
2	5 - Excelente				

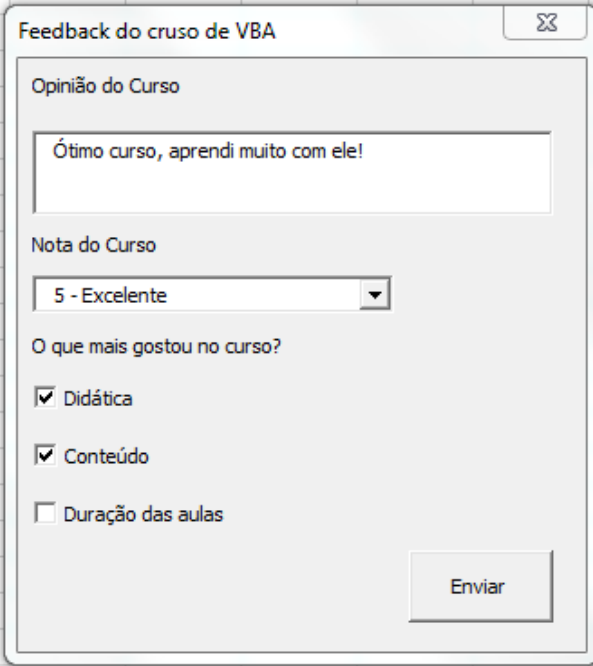
É possível observar que tanto o texto quando a nota do curso foi registrada dentro da tabela onde foi indicado. Isso possibilita a criação de uma tabela mais completa para obter várias informações através do formulário e até organizar as diversas informações solicitadas ao usuário.

Ao longo do curso vamos melhorando a utilização de cada uma das ferramentas utilizadas. Neste caso em específico podemos observar que ao abrir o formulário as informações que foram adicionadas à caixa de combinação elas começam a se repetir, isso ocorre porque estamos sempre adicionando mais dados ao abrir o formulário. Esse problema será corrigido mais à frente no curso para que possamos evitar esse tipo de erro.

CheckBox e OptionButton

Nesta parte vamos aprender a utilizar a **caixa de seleção (CheckBox)** e o **botão de opção (OptionButton)** além do formato a diferença entre eles é que a caixa de opção nos permite selecionar mais de uma opção enquanto o botão de opção só nos permite ter uma seleção, ou seja, se tentarmos selecionar outra opção a anterior será desmarcada, ficando apenas uma ativa.

Essas duas opções serão inseridas no formulário igualmente as outras, portanto basta clicar e arrastar para colocar no local desejado. A escrita das duas ferramentas poderá ser modificada como nos itens anteriores.



Feedback do curso de VBA

Opinião do Curso

Ótimo curso, aprendi muito com ele!

Nota do Curso

5 - Excelente

O que mais gostou no curso?

☒ Didática

☒ Conteúdo

☐ Duração das aulas

Enviar

É possível observar que com o **CheckBox** é possível fazer mais de uma seleção e é possível desmarcar uma das opções. Vamos acrescentar mais algumas linhas de código para atribuir os valores dessas caixas dentro da nossa planilha.

```
Private Sub CommandButton1_Click()
    Dim texto As String, nota As String

    texto = TextBox1.Value
    nota = ComboBox1.Value

    UserForm1.Hide

    MsgBox ("Feedback enviado com sucesso!!")

    Range("A1").Value = texto
    Range("A2").Value = nota
    Range("A3").Value = UserForm1.CheckBox1.Value
    Range("A4").Value = UserForm1.CheckBox2.Value
    Range("A5").Value = UserForm1.CheckBox3.Value
End Sub
```

Ao clicar no botão enviar vamos ter o seguinte resultado dentro da planilha.

	A	
1	Ótimo curso, aprendi muito com ele!	
2	5 - Excelente	
3	VERDADEIRO	
4	VERDADEIRO	
5	FALSO	
6		

É possível observar que resultado retornado pelo **CheckBox** é apenas **VERDADEIRO** ou **FALSO**, isso quer dizer que não teremos o texto sendo retornado automaticamente, o valor atribuído a essa caixa é do **tipo booleano**.

Uma possibilidade de corrigir isso para o nome selecionado é colocando uma estrutura **If** para verificar se a caixa está com o valor verdadeiro (**True**), em caso positivo podemos colocar ao invés de **.Value** o **.Caption** que é exatamente o nome que colocamos em cada uma das opções. Como estamos apenas apresentando as ferramentas vamos deixar essas modificações quando formos construir um formulário completo na construção da **Sexta Ferramenta**.

Vamos agora colocar o **OptionButton** dentro do nosso formulário e analisar quais são os valores retornados por ele.

A primeira diferença do **OptionButton** para o **CheckBox** é que só podemos selecionar **uma única opção**. Ao selecionar outra a anterior será desmarcada. Outro ponto importante é que ao selecionar uma opção não será possível desmarcá-la, ou seja, ao selecionar uma das opções esta ficará selecionada.

O código para atribuir os valores dessas opções será o mesmo utilizado anteriormente, vamos apenas alterar o nome de cada botão.

```
Private Sub CommandButton1_Click()
    Dim texto As String, nota As String

    texto = TextBox1.Value
    nota = ComboBox1.Value

    UserForm1.Hide

    MsgBox ("Feedback enviado com sucesso!!")

    Range("A1").Value = texto
    Range("A2").Value = nota
    Range("A3").Value = UserForm1.CheckBox1.Value
    Range("A4").Value = UserForm1.CheckBox2.Value
    Range("A5").Value = UserForm1.CheckBox3.Value
    Range("A6").Value = UserForm1.OptionButton1.Value
    Range("A7").Value = UserForm1.OptionButton2.Value
    Range("A8").Value = UserForm1.OptionButton3.Value
End Sub
```

Desta forma teremos os seguintes resultados dentro da planilha.

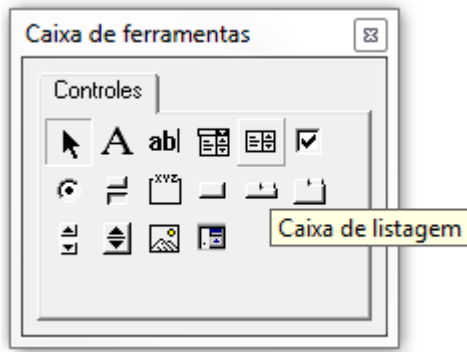
	A
1	Ótimo curso, aprendi muito com ele!
2	5 - Excelente
3	VERDADEIRO
4	VERDADEIRO
5	FALSO
6	VERDADEIRO
7	FALSO
8	FALSO
9	

Assim como o **CheckBox** o **OptionBox** também retorna valores do **tipo booleano**, ou seja, assumem o valor **VERDADEIRO** ou **FALSO**.

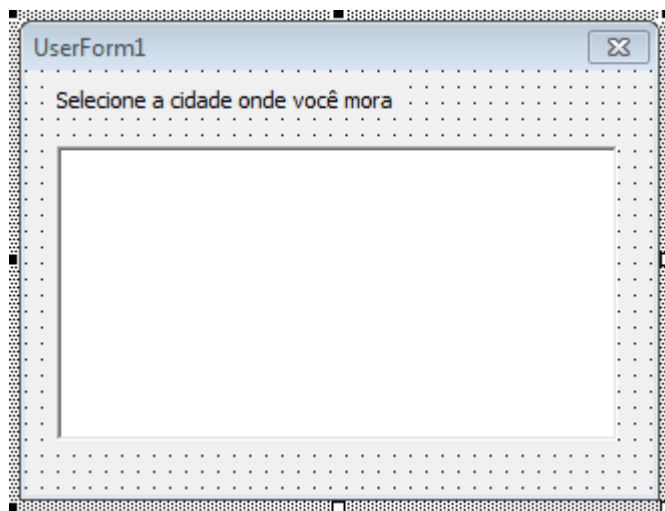
Portanto já conseguimos utilizar mais duas ferramentas para complementar a utilização do formulário e consequentemente conseguimos obter os valores dessas seleções e pudemos registrar esses valores dentro de uma célula específica da planilha.

ListBox e Eventos do UserForm

Nesta parte vamos criar um novo formulário para utilizar o **ListBox** (**Caixa de listagem**) e em seguida vamos aprender um pouco mais sobre os eventos do **UserForm**, que já foram utilizados nas ferramentas anteriores, como por exemplo **ao abrir o formulário** atualizar os valores do **ComboBox**, **ao clicar em um botão** registrar os dados nas células selecionadas.

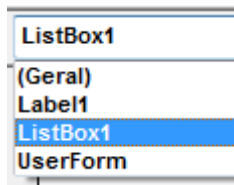


Essa caixa tem o funcionamento parecido com o **ComboBox**, no entanto ao invés de clicar na seta e aparecer uma lista para selecionar as opções, teremos as opções dentro da caixa para selecionar uma delas.

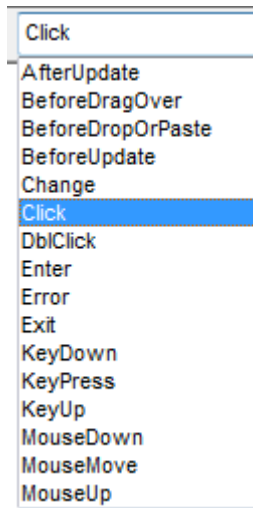


Nesta ferramenta temos exatamente a mesma propriedade que temos no **ComboBox**, o **RowSource** que é onde vamos colocar a fonte dos dados a serem mostrados no **ListBox**.

Agora vamos dar um clique duplo no **ListBox** para que possamos escrever o código. Nesta parte já conhecida vamos verificar as opções de **Eventos** do formulário.



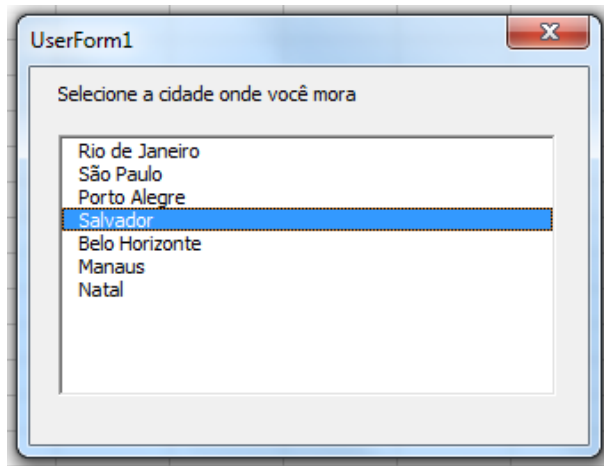
Nesta primeira parte temos os objetos inseridos no formulário, neste caso temos apenas três: o **rótulo**, o **ListBox** e o **formulário**.



Nesta segunda parte temos as ações (que são nossos eventos) que serão utilizadas no objeto selecionado para executar o código em questão. Para esse código vamos selecionar **UserForm** e o evento **Activate**, ou seja, esse código será executado sempre que o formulário for ativado. O código será para preencher o ListBox da mesma forma que foi feito com o ComboBox.

```
Private Sub UserForm_Activate()  
    ListBox1.AddItem ("Rio de Janeiro")  
    ListBox1.AddItem ("São Paulo")  
    ListBox1.AddItem ("Porto Alegre")  
    ListBox1.AddItem ("Salvador")  
    ListBox1.AddItem ("Belo Horizonte")  
    ListBox1.AddItem ("Manaus")  
    ListBox1.AddItem ("Natal")  
End Sub
```

Este é o código para adicionar os seguintes itens dentro da caixa de listagem para que apareçam dentro do nosso formulário. Ao abrir o formulário vamos ter o seguinte resultado.

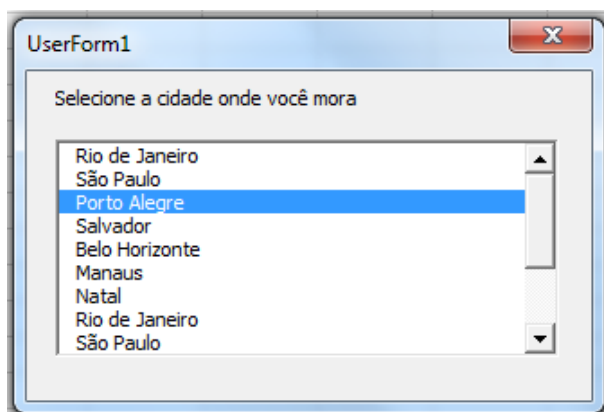


É possível observar que todos os itens estão dentro da caixa de listagem e só podemos selecionar **apenas um** deles. Neste caso ao invés de colocarmos um botão para enviar a cidade selecionada vamos colocar um **evento de duplo clique** no ListBox para registrar essa cidade na **célula A1**.

```
Private Sub ListBox1_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
    Range("A1").Value = ListBox1.Value
    UserForm1.Hide
End Sub
```

Este código irá registrar na célula A1 a opção da cidade escolhida através do clique duplo e em seguida irá esconder o formulário.

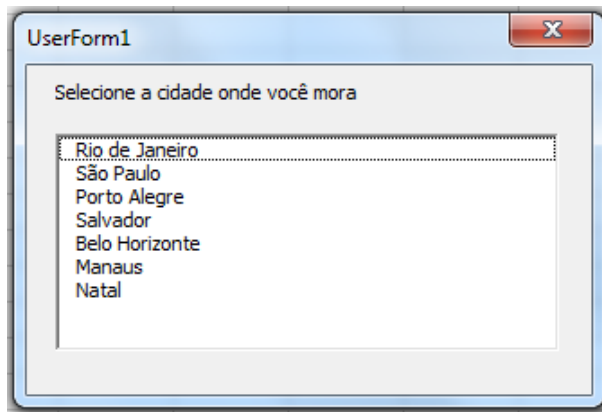
É possível verificar que ao abrir o formulário mais de uma vez os nossos itens serão adicionados novamente, ou seja, vamos começar a ter esses itens repetidos toda vez que o formulário for aberto. Isso acontece porque estamos utilizamos a opção de adicionar os itens no evento **Activate**.



Para corrigir este problema vamos adicionar uma linha de código dentro do nosso evento de duplo clique para que o conteúdo do ListBox seja limpo, ou seja, vamos apagar o conteúdo dele após o duplo clique. Então quando abrirmos

o formulário novamente, como nosso código já coloca os itens, não teremos itens repetidos.

```
Private Sub ListBox1_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
    Range("A1").Value = ListBox1.Value
    UserForm1.Hide
    ListBox1.Clear
End Sub
```



Então agora podemos abrir o formulário diversas vezes e fechar que os itens não serão mais repetidos ao ativá-lo.

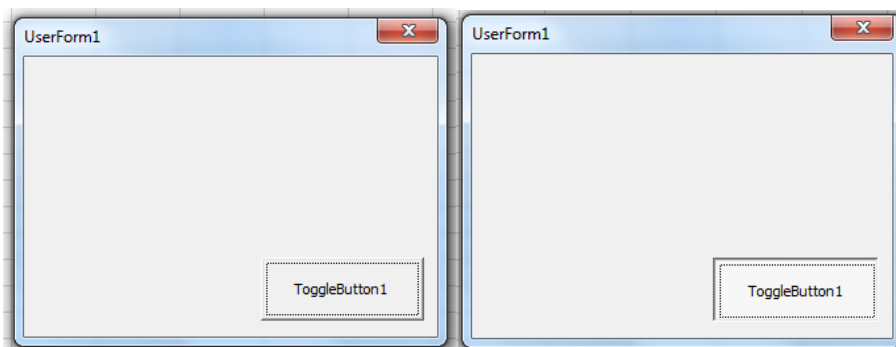
Agora temos mais uma ferramenta para complementar nosso formulário e consequentemente deixá-lo cada vez mais completo de acordo com a necessidade do usuário.

ToggleButton e Frame

Nesta parte vamos ver o **ToggleButton (botão de ativação)** e o **Frame (quadro)** que são duas ferramentas interessantes para acrescentar no formulário. O botão de ativação tem a característica de ficar pressionado ou não, enquanto o quadro é uma linha de contorno que limita a área selecionada do resto do formulário, e com isso podemos efetuar ações com os objetos no interior de forma conjunta. Também podemos fazer uma separação do **OptionButton** dentro desses quadros, ou seja, cada **OptionButton** dentro de um quadro é independente dos que estão fora, então desta forma podemos ter mais de uma parte que utiliza esses objetos sem atrapalhar a funcionalidade dos outros.



O botão de ativação é chamado de **Toggle**, porque significa alternar em inglês, ou seja, ele vai alternar entre pressionado ou não pressionado sempre que o usuário clicar nele.

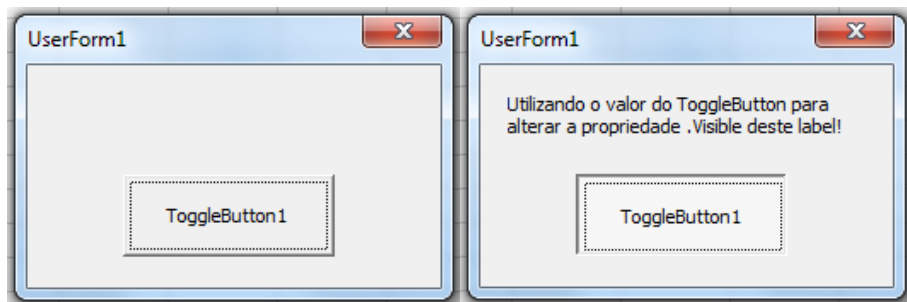


Desta forma temos duas opções para esse botão, que são as opções booleanas de **VERDADEIRO** ou **FALSO**, então podemos utilizar essas duas opções para executar diferentes ações para cada uma.

```
Private Sub ToggleButton1_Click()
    Range("A1").Value = ToggleButton1.Value

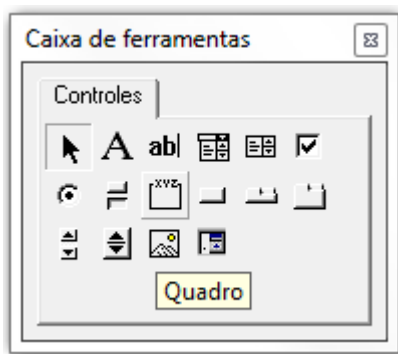
    Label1.Visible = ToggleButton1.Value
End Sub
```

Este código simples é para alterarmos a propriedade **.Visible** do Label de acordo com o valor do ToggleButton, ou seja, quando o botão assumir um valor **FALSO** ou **VERDADEIRO** a propriedade visible do label receberá este mesmo valor, então quando o botão estiver pressionado o Label aparece e quando não estiver pressionado ele desaparece (não fica visível).



Desta forma podemos mostrar ou esconder informações de acordo com o valor deste botão.

Vamos agora utilizar o **Frame**. Essa ferramenta é um **separador** do formulário para o que está dentro dele, ou seja, tudo que está dentro dele é restrito e não tem interação com o que está fora, como é o caso do **OptionButton**, pois é possível colocar vários dentro e fora do frame e será possível selecionar uma opção tanto dentro do **Frame** quanto fora dele.



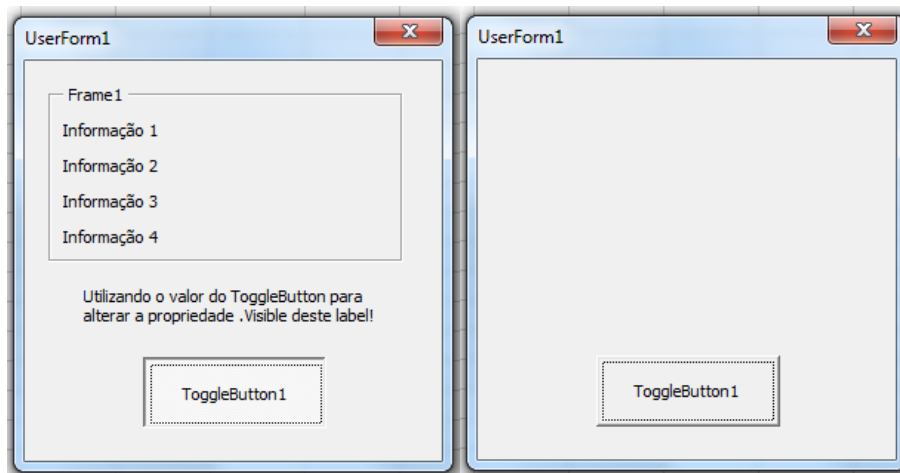
Vamos colocar um Frame e alguns Labels dentro dele, em seguida vamos associar o ToggleButton a propriedade Visible do Frame para ver o que acontece.

```
Private Sub ToggleButton1_Click()
    Range("A1").Value = ToggleButton1.Value

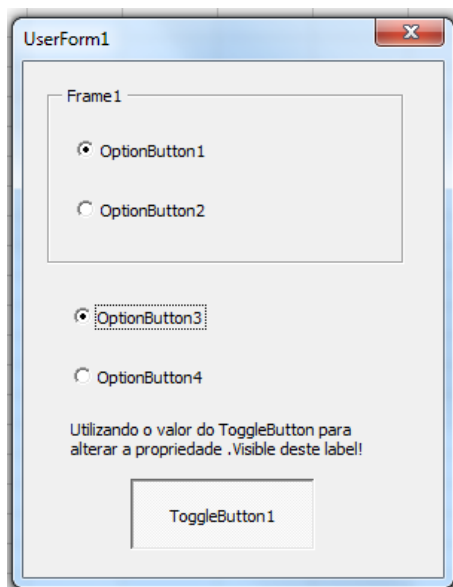
    Label1.Visible = ToggleButton1.Value
    Frame1.Visible = ToggleButton1.Value

End Sub
```

Ao escrever o código acima é possível abrir o formulário e verificar o que acontece ao pressionar o botão.

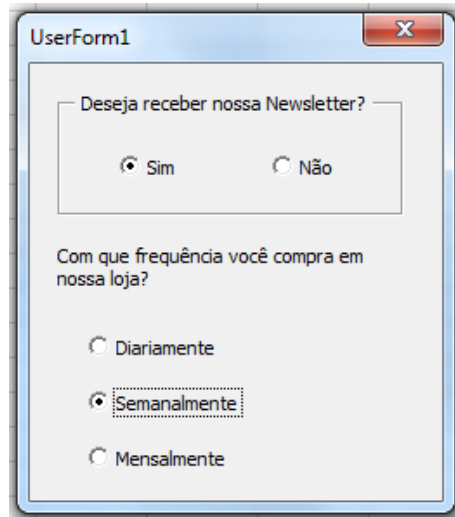


É possível observar que ao aplicar o valor **FALSO** para a propriedade Visible do **Frame** tudo que está dentro dele também recebe este valor, ou seja, não ficam mais visíveis juntamente com ele, portanto não há necessidade de escrever uma linha de código para alterar a visibilidade de cada um dos itens, basta apenas alterar a propriedade do **Frame** que ela será atribuída a todos os itens contidos nele.



Como foi dito anteriormente o Frame separa o que está dentro dele do restante do formulário, então neste caso é possível selecionar duas opções no OptionButton, porque temos eles dentro e fora do Frame, mas a propriedade desse botão continua, desta forma **não é possível selecionar o 1 e 2 ao mesmo tempo** ou o **3 e 4** ao mesmo tempo.

Portanto é possível colocarmos mais de uma informação utilizando esse botão dentro do nosso formulário sem uma interferir na outra.

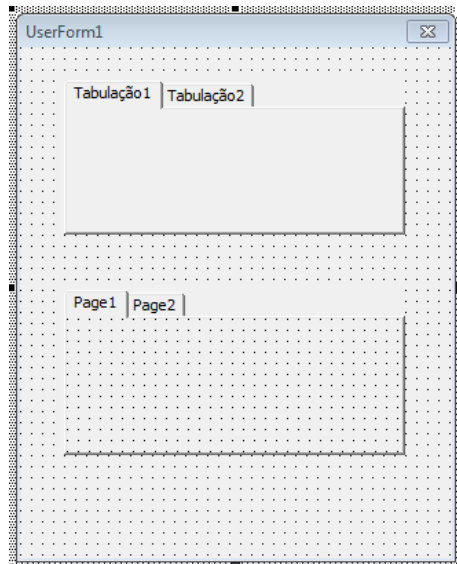


The image shows a VBA UserForm titled 'UserForm1'. It contains two distinct sections, each enclosed in a frame. The first section is titled 'Deseja receber nossa Newsletter?' and contains two radio buttons: 'Sim' (selected) and 'Não'. The second section is titled 'Com que frequência você compra em nossa loja?' and contains three radio buttons: 'Diariamente', 'Semanalmente' (selected), and 'Mensalmente'. The 'Semanalmente' button is highlighted with a dashed border.

Essas duas novas ferramentas permitem aumentar ainda mais as opções a serem inseridas no formulário, portanto ele fica cada vez mais completo, podendo detalhar mais cada pergunta ou informação solicitada ao usuário.

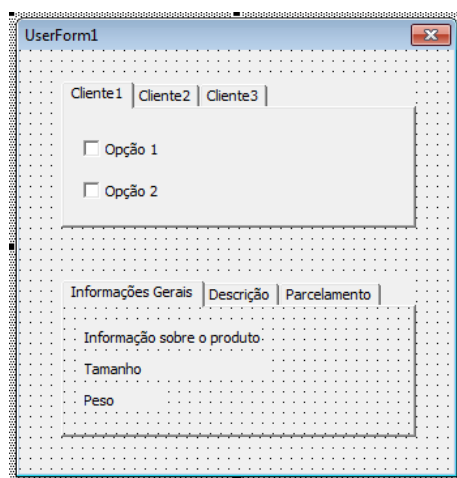
MultiPágina e TabStrip

Nesta parte vamos trabalhar com essas duas ferramentas que possibilitam colocar no formulário uma espécie de **frame com abas**, ou seja, podemos utilizar diferentes abas dentro dessas ferramentas com o objetivo de colocar mais informações dentro de um só lugar economizando espaço.

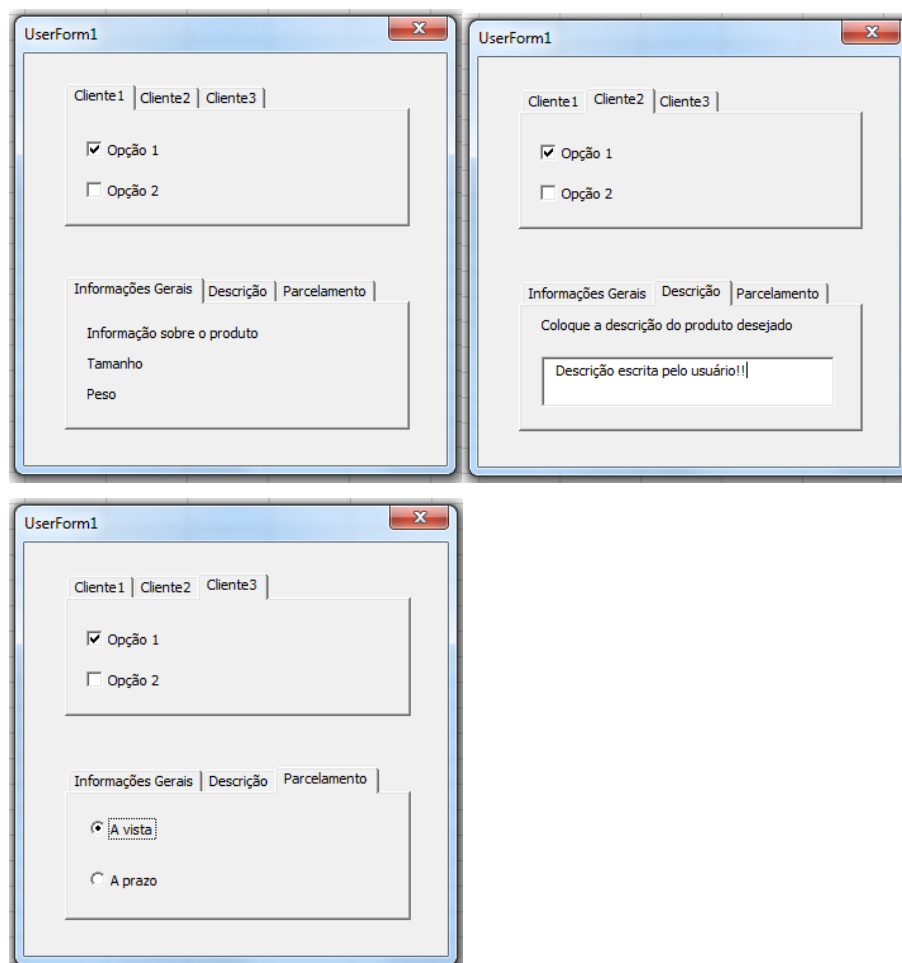


Nos dois casos para acrescentar abas basta clicar com o botão direito e inserir uma nova. Para renomear o procedimento é o mesmo, basta clicar com o botão direito e selecionar a opção de renomear.

A principal diferença entre essas duas ferramentas é que a **Tabulação** mantém o que for colocado dentro dela para todas as abas, enquanto a **Multi-Página** tem uma nova página para cada aba.



Agora vamos abrir o formulário dentro do Excel para verificarmos seleção de cada uma das abas e as diferenças entre elas.



É possível observar que as abas dos clientes não modificam ao trocar de uma para outra, e a seleção delas continua a mesma para todas as abas, neste caso foi selecionada a **Opção 1** e ela ficou fixa para todas as abas.

Na parte inferior que é referente a **Multi-Página** é possível observar que temos diferentes informações nas diferentes abas, isso pode ser uma forma de colocar várias informações em um mesmo espaço sem ter que aumentar o tamanho do formulário (para deixá-lo mais compacto).

Portanto a utilização dessas duas ferramentas vai depender da aplicação do usuário, pois talvez a primeira seja mais adequada para alguma atividade, como a segunda pode ser mais adequada para outra atividade. Lembrando que é possível saber qual das abas está selecionada em ambos os casos, isso para a Tabulação é importante para saber qual cliente está sendo selecionado, visto que as opções marcadas serão sempre as mesmas para qualquer cliente.

Sexta Ferramenta – Registro de Compras

Nesta parte vamos construir um formulário completo de uma solicitação de compra, ou seja, teremos diversas informações a serem preenchidas no formulário e vamos coloca-las dentro de uma tabela todas essas informações.

	A	B	C	D	E	F	G	H	I	J	K
1	Setor Solicitante	Fornecedor	Nota Emitida?	IR	PIS	COFINS	ISS	Tipo	Prazo Pagamento	Valor	Descrição
2											
3											
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											

Vamos criar um formulário para preencher esses dados na nossa planilha de solicitação de compra.

Este será o formulário utilizado para preencher todos os dados da nossa planilha, abaixo temos as outras duas abas da nossa **Multi-Página**. Em seguida teremos os códigos utilizados para cada uma das nossas ferramentas com algumas explicações do que cada parte está fazendo para que seja possível entender cada passo, as novas ferramentas e alguns novos códigos para o registro das informações obtidas.

The screenshot shows a VBA UserForm with three tabs at the top: 'Pagamento', 'Descrição', and 'Valor'. The 'Descrição' tab is selected. Inside the form, there is a text box with the instruction 'Escreva todas as especificações do produto não abordadas nos outros itens do formulário'.

The screenshot shows the same VBA UserForm, but now the 'Valor' tab is selected. It displays a label 'Valor Total' next to a text box for input.

Vamos a primeira parte do código.

```
Private Sub CommandButton1_Click()
    'cadastrar informações
    Call Cadastrar

    'limpar formulário
    UserForm1.Hide
    For Each objeto In UserForm1.Controls
        On Error Resume Next
        objeto.Value = ""
    Next
End Sub
```

Começamos com um código simples que funciona ao **clique de um botão** (que é o botão Registrar), nesta parte será chamada a **Sub Cadastrar**, o formulário será escondido e será atribuído o valor vazio a propriedade value dos objetos, para resetá-los ao valor inicial.

```
Sub Cadastrar()
    Dim range1 As Range
    If RefEdit1.Value <> "" Then
        Set range1 = Range(RefEdit1.Value)
    ElseIf Range("A2").Value = "" Then
        Set range1 = Range("A2")
    Else
        Set range1 = Range("A1").End(xlDown).Offset(1, 0)
    End If

    range1.Value = UserForm1.ComboBox1.Value
    range1.Offset(0, 1).Value = UserForm1.ListBox1.Value
    range1.Offset(0, 2).Value = UserForm1.ToggleButton1.Value
    range1.Offset(0, 3).Value = UserForm1.CheckBox1.Value
    range1.Offset(0, 4).Value = UserForm1.CheckBox2.Value
    range1.Offset(0, 5).Value = UserForm1.CheckBox3.Value
    range1.Offset(0, 6).Value = UserForm1.CheckBox4.Value

    'cadastro do tipo (produto ou serviço)
    If UserForm1.OptionButton1.Value = True Then
        range1.Offset(0, 7).Value = UserForm1.OptionButton1.Caption
    Else
        range1.Offset(0, 7).Value = UserForm1.OptionButton2.Caption
    End If

    'cadastro prazo de pagamento
    If UserForm1.OptionButton3.Value = True Then
        range1.Offset(0, 8).Value = UserForm1.OptionButton3.Caption
    ElseIf UserForm1.OptionButton4.Value = True Then
        range1.Offset(0, 8).Value = UserForm1.OptionButton4.Caption
    Else
        range1.Offset(0, 8).Value = UserForm1.OptionButton5.Caption
    End If

    range1.Offset(0, 9).Value = CDb1(UserForm1.TextBox2.Value)
    range1.Offset(0, 9).Style = "Currency"
    range1.Offset(0, 10).Value = UserForm1.TextBox1.Value
End Sub
```

Aqui temos a **Sub Cadastrar**, na primeira parte estamos verificando se a segunda caixa do nosso formulário está vazia ou não, se não estiver vamos começar os registros na linha selecionada, caso contrário vamos começar os registros na próxima linha vazia.

Esse **RefEdit** é uma ferramenta para selecionar um range, ou seja, funciona como as opções de dentro do Excel quando precisamos selecionar um range para que uma certa ferramenta funcione. Então estamos dando a possibilidade de o usuário selecionar onde serão cadastradas as informações, caso não informe nenhum, como foi mostrado, ele irá cadastrar na próxima linha vazia.

Logo abaixo podem ser observadas as atribuições dos valores das ferramentas em cada uma das células correspondentes.

Na parte de **cadastro do tipo** é possível observar que temos a **estrutura If** para verificar qual das opções está selecionada para que seja possível registrar na planilha a opção correta. Para o **prazo de pagamento** é feito o mesmo procedimento, para verificar qual das opções está selecionada e estamos registrando o **Caption** (nome que aparece no OptionButton) dentro da célula selecionada.

Por fim temos algumas alterações interessantes nas últimas linhas de código. A primeira delas é para alterar o tipo do dado que será atribuído a célula, neste caso temos a função **Cdbl** que é para **converter o valor** entre parênteses para o tipo **Double (decimal)**, isso é necessário, pois os dados ficam como se fossem textos e apontam um erro dentro do ambiente Excel.

A segunda alteração é a modificação do formato da célula para **Currency (moeda)**, desta forma teremos o nosso valor já com o formato correto e não apenas um número.

A última alteração é apenas a atribuição do texto que foi escrito na parte de descrição na célula indicada.

```
Private Sub ToggleButton1_Click()
    Frame2.Visible = ToggleButton1.Value
End Sub
```

Este código é apenas para alterarmos a visibilidade do quadro de impostos de acordo com valor do botão **Nota Emitida**.

```
Private Sub UserForm_Initialize()

    With ComboBox1
        .AddItem ("Financeiro")
        .AddItem ("Marketing")
        .AddItem ("Operações")
        .AddItem ("Administrativo")
    End With

    Frame2.Visible = False

End Sub
```

Para finalizar temos a adição dos itens do **ComboBox** que faz referência ao setor solicitante e tira a visibilidade do quadro de impostos.

É possível observar que no código não foram inseridos os fornecedores no **ListBox**, isso ocorre porque a fonte deles foi escrita na caixa de **RowSource**.

MultiSelect	0 - fmMultiSelectSingle
RowSource	Ref!A1:A10
SpecialEffect	2 - fmSpecialEffectSunken
TabText...	

Como temos a lista desses fornecedores em uma das nossas abas do ambiente Excel, basta escrever a fonte deles para que sejam mostrados dentro da caixa de listagem.

Escrito todo o código podemos testar para verificar a funcionalidade. O usuário poderá testar aos poucos quando estiver construindo um formulário do zero para ter certeza de que cada parte está funcionando corretamente com as ferramentas já ensinadas no curso.

Formulário de Registro de Compra

SetorSolicitante: Marketing

Selecione o Range para Cadastro (se vazio será cadastrado na próxima linha da base):

Fornecedor: Fornecedor 4, Fornecedor 5, Fornecedor 6, Fornecedor 7 (selecionado), Fornecedor 8

Tipo de Solicitação: Produto (selecionado), Serviço

Impostos: IR (selecionado), PIS (selecionado), COFINS, ISS

Nota Emitida: []

Pagamento: Antecipado, Na entrega (selecionado), 30 Dias

Registrar

Ao selecionar o botão Registrar podemos observar todas essas informações registradas na nossa planilha. Como não colocamos um range nossos dados serão registrados na próxima linha vazia da tabela.

	A	B	C	D	E	F	G	H	I	J	K
1	SectorSolicitante	Fornecedor	Nota Emitida?	IR	PIS	COFINS	ISS	Tipo	Prazo Pagamento	Valor	Descrição
2	Marketing	Fornecedor 7	VERDADEIRO	VERDADEIRO	VERDADEIRO	FALSO	FALSO	Produto	Na entrega	R\$ 245,97	Produto deverá ser dividido em 3 embalagens para a entrega.
3											
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											

Todas informações foram cadastradas em suas devidas colunas conforme o código escrito. Portanto temos um formulário mais elaborado referente a uma solicitação de compra.

O usuário poderá cadastrar diversos produtos e verificar que todos serão cadastrados corretamente inclusive se escolher uma célula da coluna A para que as informações sejam cadastradas na linha escolhida.

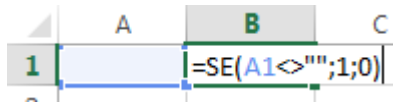
Então agora somos capazes de criar diversos tipos de formulários com as ferramentas aprendidas para atender as necessidades do usuário. Além do formulário aprendemos também como manipular essas informações e consequentemente registrá-las ou utilizá-las de alguma forma dentro do código.

Seção 9 – Funções do Excel e do VBA

Funções do Excel

Nesta parte vamos verificar como podemos utilizar as funções do Excel dentro do VBA, ou seja, vamos aprender a escrevê-las dentro do VBA para que apareçam corretamente dentro do ambiente Excel. Até o momento só fizemos registros dentro de células, alteramos valores, verificamos células e intervalos. Agora vamos aprender a utilizar essas funções dentro do ambiente VBA.

Assim como outras funções que não sabíamos nós gravamos um macro para verificar seu código aqui vamos fazer exatamente o mesmo. Vamos gravar uma macro simples da função SE na célula A2 e em seguida vamos verificar o código em VBA e aprender como ele funciona.



Ao gravar essa macro teremos o seguinte código.

```
Sub Macro1 ()
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "=IF(RC[-1]<>\"\"\",1,0) "
    Range("B2").Select
End Sub
```

A primeira e última linha do código já sabemos como funcionam, no entanto, a linha central é a parte nova. Primeiramente temos a utilização da propriedade **FormulaR1C1**, isso é porque ele utiliza essa referência para inserir a fórmula dentro da célula. Essa é uma referência **Row (R)** e **Column (C)** que a referência de **linha** e **coluna**.

Após o símbolo de igual podemos verificar alguns pontos importantes, o primeiro deles é que toda a fórmula foi colocada dentro de aspas duplas, o segundo ponto é que as fórmulas inseridas dentro do ambiente VBA deverão ser todas em inglês, ou seja, por mais que seja utilizada a **função SE** no ambiente **Excel**, dentro do ambiente **VBA** será utilizada a **função IF**.

Logo após o símbolo de igual dentro da fórmula temos **RC[-1]**, isso é exatamente a referência de linha e coluna, ou seja, para a linha (**R**) não temos nada, isso quer dizer que estamos analisando a mesma linha da célula, enquanto para a coluna (**C**) temos **-1**, isso quer dizer que estamos analisando uma coluna anterior (ou uma coluna à esquerda) a célula da fórmula. Como a célula da fórmula é a **B1**, a célula de mesma linha e coluna anterior é a **célula A1**, portanto estamos analisando **SE** a **célula A1** possui valor diferente de vazio.

Outro ponto importante é que ao colocarmos a condição de ser diferente de vazio temos que o VBA coloca 4 aspas duplas ao invés de 2, ou seja, para

essa indicação precisamos colocar textos entre aspas duplas, então temos que colocar as aspas duplas entre aspas duplas, totalizando 4 delas.

O restante da fórmula segue o mesmo padrão da **função SE**, o primeiro argumento é a comparação, o segundo é o resultado no caso **VERDADEIRO** e o terceiro é o resultado no caso **FALSO**.

Mais um ponto importante é que como o VBA é em inglês a separação dos argumentos é feita por **vírgulas** e não ponto e vírgula como utilizamos dentro do ambiente Excel que está em português.

Com isso podemos agora trabalhar com as funções do Excel dentro do VBA, pois agora sabemos como elas funcionam, principalmente a relação RC que são as referências de linha e coluna. Caso o usuário não saiba como escrever uma fórmula ou como representar um intervalo basta gravar um macro fazendo exatamente o que deseja para verificar o código. Vale ressaltar que dentro do ambiente VBA as funções sempre deverão ser em inglês, portanto temos abaixo um link para ajudar na tradução das fórmulas português inglês.

<http://fazaconta.com/excel-ingles-portugues.htm>

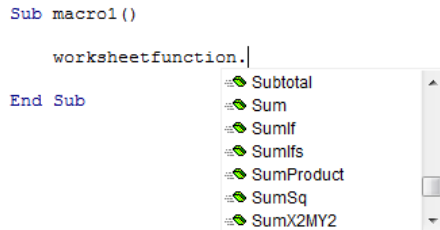
Abaixo temos outro link para algumas funções do VBA que talvez não tenhamos visto ou que sejam necessárias em algum ponto. Na construção da Sexta Ferramenta foi utilizada uma função de conversão de valor que será possível encontrar também no link abaixo.

<https://msdn.microsoft.com/pt-br/library/32s6akha.aspx>

Portanto mesmo não sabendo todas as fórmulas ou como elas funcionam é possível verificar a escrita delas em inglês, verificar no site acima algumas funções que podem ser úteis para a aplicação do usuário e ainda temos o gravador de macro que ajuda bastante no aprendizado de VBA.

WorksheetFunction

Nesta parte vamos aprender a utilizar as funções do Excel utilizando um método do VBA, para isso vamos escrever **WorksheetFunction** e ao acrescentar o ponto podemos observar as diversas funções do Excel dentro do VBA.



Desta forma podemos selecionar alguma função do Excel que esteja nessa lista (lembrando que todas irão aparecer em inglês) e dar prosseguimento a função selecionada. Neste caso vamos utilizar a função **SOMA (SUM em inglês)**. Esta função funciona da mesma forma que no ambiente Excel, portanto basta colocar o intervalo de soma.

Como essa fórmula vai retornar um valor podemos atribuí-la a uma variável ou a uma célula, que será o caso.

```
Sub macro1()

    Range("B1").Value = WorksheetFunction.Sum(Range("A1:A10"))

End Sub
```

Ao executar o código teremos a soma do intervalo da **célula A1** até **A10**, e o resultado será registrado na **célula B1**.

B1		:	X	✓	<i>fx</i>	48
Sensibilidade: NP-1						
	A	B	C	D		
1	2	48				
2	5					
3	8					
4	4					
5	6					
6	9					
7	1					
8	3					
9	3					
10	7					

É possível observar que desta forma o VBA coloca na célula o resultado da soma e não a fórmula da soma, diferentemente do que ocorre quando atribuímos diretamente a função dentro da célula tipo “=SUM(A1:A10)” ou a propriedade **R1C1**, que é o padrão do Excel.

Essas duas últimas formas vão atribuir a função dentro da célula, isso quer dizer que ao modificar um desses 10 valores analisados essa célula terá seu valor modificado automaticamente, da mesma forma que acontece quando escrevemos diretamente no Excel.

A atribuição utilizando o **WorksheetFunction** ela vai atribuir somente o resultado, ou seja, se alterarmos algum valor no intervalo o resultado só será alterado quando o código for executado novamente.

Portanto podemos utilizar as diversas funções do Excel dentro do VBA de uma forma diferente e sem mostrar ao usuário que fórmula foi utilizada naquela célula ou quais os dados que estão sendo utilizados para obter aquele resultado.

Funções VBA

Existem algumas funções que são específicas do VBA, que podem ser encontradas no link abaixo (que já foi passado na parte de Funções do Excel).

<https://msdn.microsoft.com/pt-br/library/32s6akha.aspx>

Nesta seção

[Funções de Conversão](#)

[Funções Matemáticas](#)

[Funções da Cadeia de Caracteres](#)

[Funções de Conversão do Tipo](#)

[Função CType](#)

Neste site é possível observar os tipos de funções, e em cada seção é possível observar como escrever e utilizar cada uma delas. Na **Sexta Ferramenta** foi utilizada uma **Função de Conversão do Tipo** que é a função que está em destaque. Essa função serve para converter um tipo qualquer para **Double** (decimal).

Sintaxe

```
CBool(expression)
CByte(expression)
CChar(expression)
CDate(expression)
CDBl(expression)
CDec(expression)
CInt(expression)
CLng(expression)
CObj(expression)
CShort(expression)
CSByte(expression)
CShort(expression)
CSng(expression)
CStr(expression)
CUInt(expression)
CULng(expression)
CShort(expression)
```

É possível observar que a função marcada que utilizamos para converter um valor para o tipo **Double**. Além dessa função temos diversas outras somente para a conversão de tipo.

Abaixo temos alguns exemplos das funções, como utilizá-las e os valores que aquele tipo pode assumir.

Tipo de dados do valor de retorno

O nome da função determina o tipo de dados do valor retornado por ele, conforme mostrado na tabela a seguir.

Nome da função	Tipo de dados de retorno	Intervalo para <code>expression</code> argumento
<code>CBool</code>	Tipo de Dados Boolean	Qualquer <code>Char</code> ou <code>String</code> ou expressão numérica.
<code>CByte</code>	Tipo de Dados Byte	<code>Byte.MinValue</code> (0) por meio <code>Byte.MaxValue</code> (255) (não assinado); são arredondados para partes fracionárias. ¹ Começando com 15.8 do Visual Basic, Visual Basic otimiza o desempenho de ponto flutuante para conversão de bytes com o <code>CByte</code> função; consulte a comentários seção para obter mais informações. Consulte a CInt exemplo seção para obter um exemplo.
<code>CChar</code>	Tipo de Dados de Caractere	Qualquer <code>Char</code> ou <code>String</code> expressão; apenas primeiro caractere de um <code>String</code> é convertido; valor pode ser de 0 a 65535 (não assinado).

Desta forma fica mais fácil para consultar e utilizar essas funções dentro de um código sempre que necessário.

Como não vamos analisar cada uma das funções fica a cargo do usuário pesquisar mais sobre essas e outras funções para aprender e procurar sempre que precisar de uma explicação ou uma nova aplicação. É sempre possível encontrar códigos prontos ou blogs para auxiliar nos códigos, então é interessante pesquisar essas informações para complementar os estudos.

Seção 10 – Acelerando o Código

Nesta parte vamos aprender sobre duas ferramentas para acelerar a execução dos códigos em VBA. Os códigos que vimos até o momento não foram grandes, e portanto, não pudemos ver a diferença em tempo de execução.

Vamos utilizar nesta seção uma planilha um pouco mais pesada em que vamos poder observar a diferença em tempo de execução do código em questão. Essas ferramentas são úteis principalmente quando temos uma grande quantidade de dados para processar, o que é comum em grandes empresas.

Para isso vamos utilizar duas propriedades do objeto **Application** (que é o Excel em si), ou seja, são propriedades do Excel.

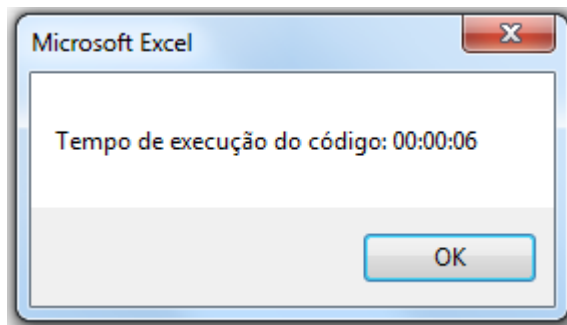
Application.ScreenUpdating

Para que seja possível verificar essa aceleração do código vamos utilizar o código de **Consolidação de Contas** que já fizemos, no entanto, esse terá uma quantidade bem maior de informações, ou seja, teremos muito mais linhas a serem analisadas.

	A	B	C	D	E	F	G	H
1	Data	Dia da Semana	Tipo	Valor	Documento	Status	Saldo Anterior	Saldo Final
2	27/10/2015	Terça-feira	Entrada	R\$ 2.386,00	E27102C1	Finalizado	-R\$ 100,00	R\$ 2.286,00
3	27/10/2015	Terça-feira	Saída	R\$ 9.228,00	S27103C1	Pendente	R\$ 2.286,00	-R\$ 6.942,00
4	27/10/2015	Terça-feira	Entrada	R\$ 8.833,00	E27104C1	Finalizado	-R\$ 6.942,00	R\$ 1.891,00
5	27/10/2015	Terça-feira	Entrada	R\$ 8.387,00	E27105C1	Finalizado	R\$ 1.891,00	R\$ 10.278,00
6	27/10/2015	Terça-feira	Entrada	R\$ 99,00	E27106C1	Finalizado	R\$ 10.278,00	R\$ 10.377,00
7	27/10/2015	Terça-feira	Saída	R\$ 6.741,00	S27107C1	Finalizado	R\$ 10.377,00	R\$ 3.636,00
8	28/10/2015	Quarta-feira	Entrada	R\$ 694,00	E28108C1	Finalizado	R\$ 3.636,00	R\$ 4.330,00
9	28/10/2015	Quarta-feira	Saída	R\$ 6.167,00	S28109C1	Finalizado	R\$ 4.330,00	-R\$ 1.837,00
10	28/10/2015	Quarta-feira	Saída	R\$ 3.512,00	S281010C1	Finalizado	-R\$ 1.837,00	-R\$ 5.349,00
11	28/10/2015	Quarta-feira	Saída	R\$ 5.665,00	S281011C1	Finalizado	-R\$ 5.349,00	-R\$ 11.014,00
12	28/10/2015	Quarta-feira	Saída	R\$ 9.126,00	S281012C1	Finalizado	-R\$ 11.014,00	-R\$ 20.140,00
13	28/10/2015	Quarta-feira	Saída	R\$ 4.163,00	S281013C1	Finalizado	-R\$ 20.140,00	-R\$ 24.303,00
14	28/10/2015	Quarta-feira	Entrada	R\$ 9.982,00	E281014C1	Finalizado	-R\$ 24.303,00	-R\$ 14.321,00
15	28/10/2015	Quarta-feira	Entrada	R\$ 7.388,00	E281015C1	Finalizado	-R\$ 14.321,00	-R\$ 6.933,00
16	28/10/2015	Quarta-feira	Entrada	R\$ 9.974,00	E281016C1	Finalizado	-R\$ 6.933,00	R\$ 3.041,00
17	28/10/2015	Quarta-feira	Entrada	R\$ 7.380,00	E281017C1	Finalizado	R\$ 3.041,00	R\$ 10.421,00
18	29/10/2015	Quinta-feira	Entrada	R\$ 8.901,00	E291018C1	Finalizado	R\$ 10.421,00	R\$ 19.322,00
19	30/10/2015	Sexta-feira	Entrada	R\$ 5.569,00	E301019C1	Finalizado	R\$ 19.322,00	R\$ 24.891,00
20	30/10/2015	Sexta-feira	Entrada	R\$ 8.906,00	E301020C1	Finalizado	R\$ 24.891,00	R\$ 33.797,00
21	30/10/2015	Sexta-feira	Saída	R\$ 3.405,00	S301021C1	Finalizado	R\$ 33.797,00	R\$ 30.392,00
22	30/10/2015	Sexta-feira	Entrada	R\$ 9.326,00	E301022C1	Finalizado	R\$ 30.392,00	R\$ 39.718,00
23	30/10/2015	Sexta-feira	Entrada	R\$ 855,00	E301023C1	Finalizado	R\$ 39.718,00	R\$ 40.573,00
24	30/10/2015	Sexta-feira	Entrada	R\$ 9.863,00	E301024C1	Finalizado	R\$ 40.573,00	R\$ 50.436,00
25	02/11/2015	Segunda-feira	Saída	R\$ 2.813,00	S21125C1	Finalizado	R\$ 50.436,00	R\$ 47.623,00
26	02/11/2015	Segunda-feira	Entrada	R\$ 1.294,00	E21126C1	Finalizado	R\$ 47.623,00	R\$ 48.917,00
27	02/11/2015	Segunda-feira	Saída	R\$ 6.929,00	S21127C1	Finalizado	R\$ 48.917,00	R\$ 41.988,00
28	02/11/2015	Segunda-feira	Saída	R\$ 9.471,00	S21128C1	Finalizado	R\$ 41.988,00	R\$ 32.517,00
29	02/11/2015	Segunda-feira	Saída	R\$ 9.650,00	S21129C1	Finalizado	R\$ 32.517,00	R\$ 22.867,00
30	02/11/2015	Segunda-feira	Saída	R\$ 4.581,00	S21130C1	Finalizado	R\$ 22.867,00	R\$ 18.286,00
31	02/11/2015	Segunda-feira	Saída	R\$ 1.894,00	S21131C1	Finalizado	R\$ 18.286,00	R\$ 16.392,00
32	02/11/2015	Segunda-feira	Entrada	R\$ 9.109,00	E21132C1	Pendente	R\$ 16.392,00	R\$ 25.501,00
33	02/11/2015	Segunda-feira	Entrada	R\$ 4.845,00	E21133C1	Finalizado	R\$ 25.501,00	R\$ 30.346,00
34	02/11/2015	Segunda-feira	Entrada	R\$ 3.249,00	E31134C1	Finalizado	R\$ 30.346,00	R\$ 33.595,00
35	02/11/2015	Segunda-feira	Entrada	R\$ 9.888,00	E31135C1	Finalizado	R\$ 33.595,00	R\$ 43.483,00
36	02/11/2015	Segunda-feira	Entrada	R\$ 8.005,00	E31136C1	Finalizado	R\$ 43.483,00	R\$ 51.488,00
37	02/11/2015	Segunda-feira	Entrada	R\$ 7.307,00	E31137C1	Finalizado	R\$ 51.488,00	R\$ 58.795,00
38	02/11/2015	Segunda-feira	Saída	R\$ 4.123,00	S31138C1	Finalizado	R\$ 58.795,00	R\$ 54.672,00

Na aba da conta 1 temos agora **833 linhas** preenchidas, portanto temos uma quantidade de dados muito maior do que antes.

Vamos executar o código de consolidação de contas e verificar o tempo aproximado de execução do código.



O tempo de execução desse código testado foi de 6 segundos. Para verificar esse tempo fizemos um código simples adicionando a variável tempo. No início do código temos **tempo = now()** e no final do código temos um **MsgBox** para informar o tempo final menos o inicial.

```
Sub AtualizarCompilado()
Dim tempo As Double

tempo = Now()

consolidar ("Conta 1")
consolidar ("Conta 2")

Sheets("Consolidação de Contas").Activate

MsgBox ("Tempo de execução do código: " & Now() - tempo)

End Sub
```

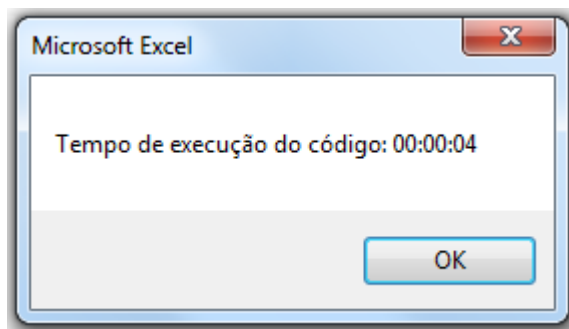
Desta forma podemos ter uma noção de diminuição de tempo com a execução dos métodos.

Vamos agora acrescentar o código do **ScreenUpdating** que é responsável por atualizar a tela do Excel, ou seja, a cada operação que é feita o Excel vai atualizando e mostrando isso ao usuário, no entanto isso é um dos pontos que deixa o código mais lento, pois ele atualiza a todo momento e as vezes não é útil nem vantajoso ficar visualizando esse processamento.

Então vamos acrescentar esse método no início do código atribuindo o valor **FALSO** para que ele não mostre essas atualizações para o usuário e no final do código vamos voltar o valor para **VERDADEIRO** para que as informações possam ser atualizadas uma única vez e volte ao padrão.

```
Sub AtualizarCompilado()  
Dim tempo As Double  
  
tempo = Now()  
Application.ScreenUpdating = False  
  
consolidar ("Conta 1")  
consolidar ("Conta 2")  
  
Sheets("Consolidação de Contas").Activate  
  
Application.ScreenUpdating = True  
MsgBox ("Tempo de execução do código: " & Now() - tempo)  
  
End Sub
```

Vamos executar o código com essas duas linhas para verificar a redução do tempo de execução do código.



Tivemos uma redução de 2 segundos, o que neste caso corresponde a 33,33% do tempo. Então já tivemos uma redução considerável de tempo. Pode ser que esse tempo de execução varie de computador para computador também, isso é importante ser lembrado, pois podem ter tempos diferentes inclusive em execuções diferentes.

Neste caso na primeira execução tinha sido obtido o valor de 3 segundos, ou seja, a redução tinha sido de 50%, mas isso varia de caso a caso. O importante é observar que temos uma redução do tempo de execução do código.

Application.Calculation

Essa é outra propriedade do objeto **Application** para acelerar a execução dos códigos. Portanto vamos utilizar a propriedade **Calculation** para reduzir o tempo de execução do código de **Consolidação de Contas**.

```
Sub AtualizarCompilado()
Dim tempo As Double

tempo = Now()
Application.Calculation = xlCalculationManual

consolidar ("Conta 1")
consolidar ("Conta 2")

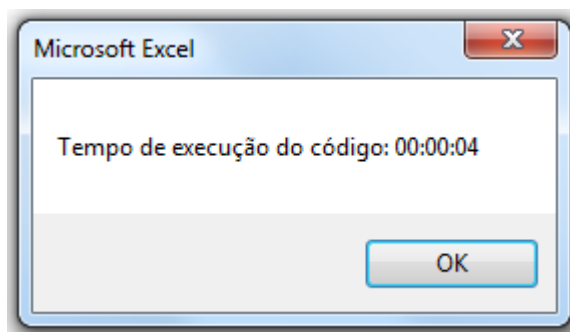
Sheets("Consolidação de Contas").Activate

Application.Calculation = xlCalculationAutomatic
MsgBox ("Tempo de execução do código: " & Now() - tempo)

End Sub
```

Esse código é um pouco diferente do **ScreenUpdating**, porque esse nos permite alterar o método de cálculo das fórmulas dentro do ambiente Excel, ou seja, quando temos uma fórmula dentro do Excel e alteramos alguma célula que faça parte desse cálculo a célula da fórmula se atualiza automaticamente, porque esse é o padrão do Excel, no entanto vamos alterar essa propriedade para **Manual**. Desta forma o Excel não vai ficar atualizando os valores até indicarmos para que ele o faça, então neste caso só vamos dizer para ele atualizar os valores no final do código, da mesma forma que fizemos com o **ScreenUpdating**.

Neste caso estamos colocando os cálculos manuais no início do código, o que faz com que as fórmulas não sejam atualizadas até o final de código, que é onde colocamos os cálculos de volta para automático, então no final do código todas as fórmulas serão atualizadas uma única vez.



Neste caso também tivemos uma **redução de 2 segundos** na execução do código, ou seja, também foi possível acelerar o código por meio dessa propriedade.

É possível também utilizar essas duas propriedades ao mesmo tempo para obter um resultado ainda melhor.

```
Sub AtualizarCompilado()
Dim tempo As Double

tempo = Now()
Application.Calculation = xlCalculationManual
Application.ScreenUpdating = False

consolidar ("Conta 1")
consolidar ("Conta 2")

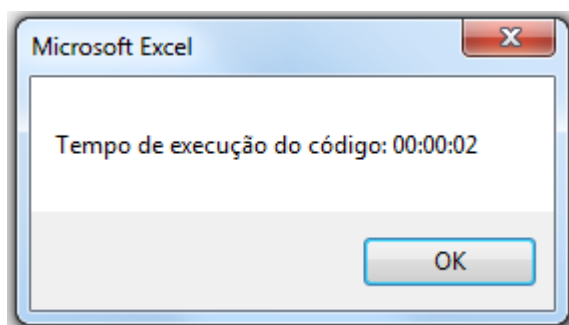
Sheets("Consolidação de Contas").Activate

Application.Calculation = xlCalculationAutomatic
Application.ScreenUpdating = True

MsgBox ("Tempo de execução do código: " & Now() - tempo)

End Sub
```

Neste caso temos tanto a propriedade **Calculation** quanto **ScreenUpdating** para verificar a redução do tempo de execução do código da consolidação de contas que foi **executado em 6 segundos** inicialmente.



Tivemos agora a **redução de 4 segundo**, que neste caso corresponde a **66,66%** do tempo total, ou seja, reduzimos a execução deste código em específico em dois terços. Como foi dito anteriormente isso vai depender de código para código. Neste exemplo tivemos esses resultados, que ao considerar as porcentagens foram muito bons.

Portanto podemos utilizar essas duas propriedades em conjunto para acelerar bastante a execução de códigos, que acaba sendo muito útil e necessário quando estamos trabalhando com uma quantidade muito grande de informações. Nesta planilha utilizamos por volta de **800 informações**

(aproximadamente 800 linhas com 8 colunas) apenas para a **conta 1**, o que é uma quantidade pequena comparada aos dados de uma grande empresa que pode ter uma planilha parecida com milhares de linhas e diversas contas a serem consolidadas por exemplo.

O usuário deve tomar cuidado com a utilização da propriedade **Calculation**, pois dependendo da forma que for utilizada o usuário poderá obter valores incorretos caso esteja pegando o valor de alguma célula que contenha uma fórmula. Se a fórmula não for atualizada isso quer dizer que vamos pegar um valor antigo e que não está correto, desta forma teremos alguns erros durante a execução do código. Por isso é preciso verificar cuidadosamente o código para se certificar que não ocorra este problema, ou o usuário poderá utilizar essa propriedade mais de uma vez dentro do código para evitar esse tipo de erro.

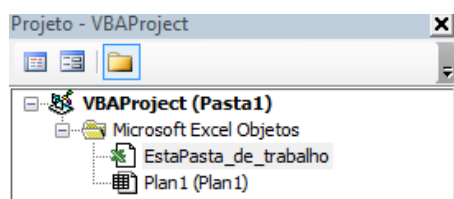
Poderá modificar o cálculo para automático antes de fazer a obtenção do valor desejado, desta forma irá obter o valor correto e irá evitar erros futuros. Em seguida poderá voltar para manual para continuar agilizando o código. Portanto é possível fazer essa alteração antes e depois de alguma fórmula que precise do seu valor atualizado para prosseguir com a execução do código sem erros.

Seção 11 - Eventos de Planilha e de Pasta de Trabalho

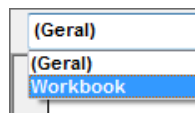
Eventos Workbook

Nesta parte vamos abordar os eventos da **pasta de trabalho (Workbook)**, que são eventos relacionados a pasta de trabalho, que podem estar associados a salvar a planilha, alterar alguma informação, mudar de aba, entre outras coisas.

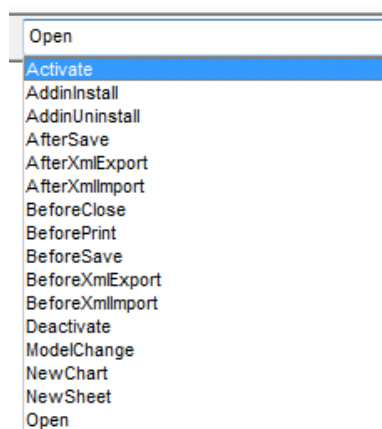
Para que possamos criar esses eventos vamos selecionar a pasta de trabalho dentro do ambiente VBA com um duplo clique.



Ao selecionar a pasta de trabalho temos a conhecida caixa de texto para começarmos a escrever, no entanto temos a opção de escolhermos os objetos como fizemos anteriormente, mas neste caso temos apenas o **Workbook**.



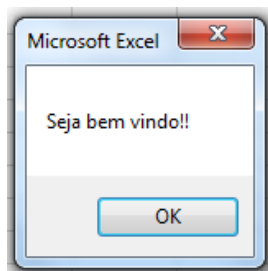
Na parte de eventos temos diversos eventos como já vimos, no entanto vão se aplicar ao objeto escolhido que neste caso é o **Workbook**.



A primeira opção que ele nos dá ao selecionar o **Workbook** é a opção **Open**, ou seja, toda vez que esse arquivo de Excel for aberto ele vai executar o código.

```
Private Sub Workbook_Open()  
    MsgBox ("Seja bem vindo!!")  
End Sub
```

Temos esse código simples para demonstrar que ao abrirmos esse arquivo ele irá iniciar com essa mensagem.



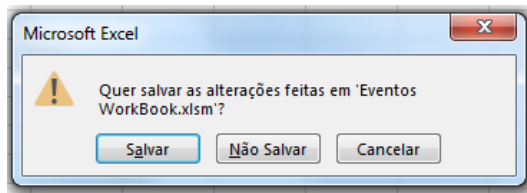
Assim que o arquivo é aberto temos a nossa **MsgBox** como era esperado, lembrando que a mensagem vai aparecer após **habilitarmos as macros** no arquivo, pois como padrão o Excel as bloqueia (por motivos de segurança). Ao iniciar o programa com uma macro será mostrada uma aba em amarelo solicitando a habilitação das macros para que elas possam funcionar.

Vamos utilizar agora o evento **antes de fechar (BeforeClose)**, ou seja, assim que clicarmos para fechar a planilha esse código será executado.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)  
    MsgBox ("Obrigado!")  
End Sub
```



Caso alguma alteração seja feita na planilha, o código (neste caso a mensagem) será executado antes de perguntar se o usuário deseja salvar as alterações, que é uma mensagem padrão caso o usuário não tenha salvado as alterações até o momento.



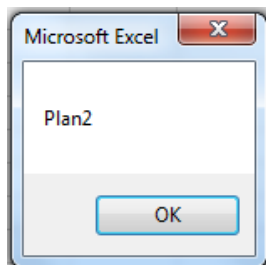
Outro exemplo que vamos utilizar é o evento **após salvar (AfterSave)**.

```
Private Sub Workbook_AfterSave(ByVal Success As Boolean)
    ActiveWorkbook.Close
End Sub
```

Neste caso vamos escrever um código para que o VBA feche a pasta de trabalho ativa após salvá-la. Então ao salvar a pasta será fechada e antes de fechar teremos a mensagem de Obrigado.

O próximo evento é o **SheetActivate**, que permite a execução de um código sempre que uma aba seja ativada.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    MsgBox (Sh.Name)
End Sub
```



Então toda vez que o usuário mudar de aba esse código será executado e irá mostrar o nome da aba selecionada. Isso vai acontecer sempre, independente do usuário já ter ativado aquela aba ou não, portanto sempre que uma aba for ativada esse código será executado.

Como pode ser visto no início, temos alguns eventos que começam com **Sheet**, isso significa que são eventos para as abas. Desta forma podemos colocar esse código dentro da pasta de trabalho, que será executado para qualquer aba e também podemos colocar dentro de cada aba. Neste último caso o código só irá funcionar para aquela aba em específico, que tem o código.

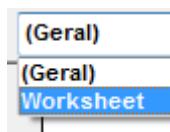
Portanto temos diversas opções para utilizar os eventos da pasta de trabalho, o que depende da aplicação de cada usuário, e vale lembrar que podemos utilizar mais de um evento de uma vez como foi mostrado.

Eventos WorkSheet

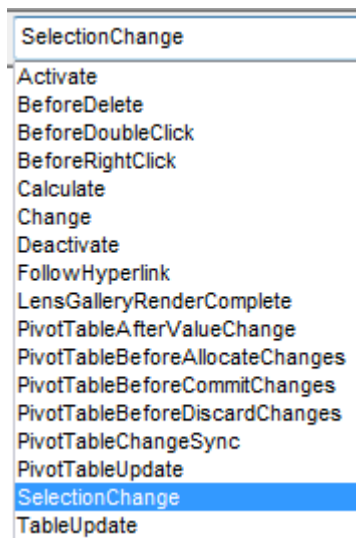
Nesta parte vamos ver os eventos que são vinculados às **Sheets** (**planilhas, abas**) do Excel. Será possível atribuir eventos as abas do Excel para que um determinado código só seja executado dentro daquela planilha em específico.

Isso quer dizer que se tivermos várias abas podemos inicialmente colocar um **MsgBox** explicativo referente aquela aba por exemplo.

Então para que possamos escrever um código para cada uma das abas basta selecionar a aba em questão com um **duplo clique** (dentro do ambiente VBA), selecionar o objeto **Worksheet** e em seguida selecionar o evento que será utilizado.



Assim como os outros objetos o **Worksheet** também possui diferentes eventos que podemos utilizar.



Vamos começar com um exemplo simples utilizando o evento **Activate**. Portanto vamos colocar para que esse código escreva na **célula A1** da **Plan1** a data com o horário do momento atual que é a fórmula **Now** no VBA e **AGORA** dentro do Excel.

```
Private Sub Worksheet_Activate()  
    Range("A1").Value = Now()  
End Sub
```

Então toda vez que ativarmos essa aba teremos esse horário sendo atualizado na célula A1 para o horário atual.

	A
1	22/07/2019 08:34

Ao invés de colocar a data e a hora na célula A1 vamos colocar apenas o horário dentro dessa célula, para isso vamos precisar alterar o formato da célula A1 para hora. Para isso vamos gravar uma macro com essa alteração e pegar o código dessa formatação para acrescentarmos ao nosso código.

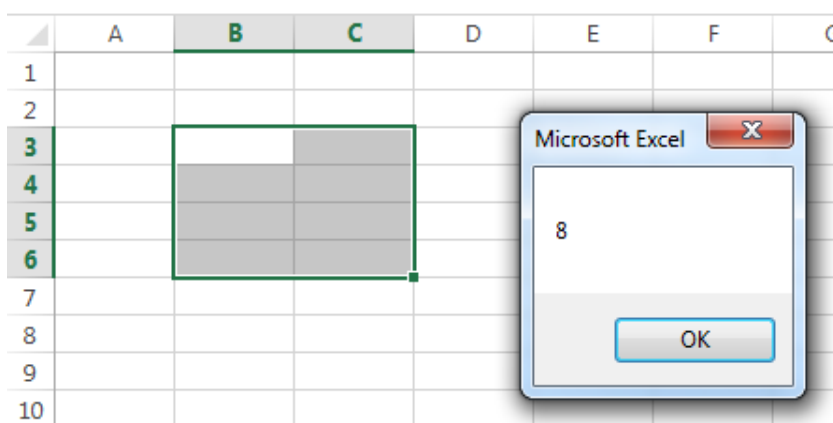
	A
1	08:38:19

Desta forma a célula sempre ficará no formato de hora mostrando apenas a hora, pois sempre que ativamos a aba o código será executado e estará formatando esse dado da célula A1.

Para a **Plan2** vamos utilizar o evento **SelectionChange**, ou seja, o código será executado sempre que o usuário trocar a seleção, isso quer dizer que sempre que for feita uma seleção de célula diferente ele irá executar o código.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    MsgBox (Target.Count)
End Sub
```

Aqui temos a variável **Target** inicializada com o valor da seleção que é do **tipo range**, então vamos apenas criar um **MsgBox** para contar as células que temos dentro dessa variável **Target**.

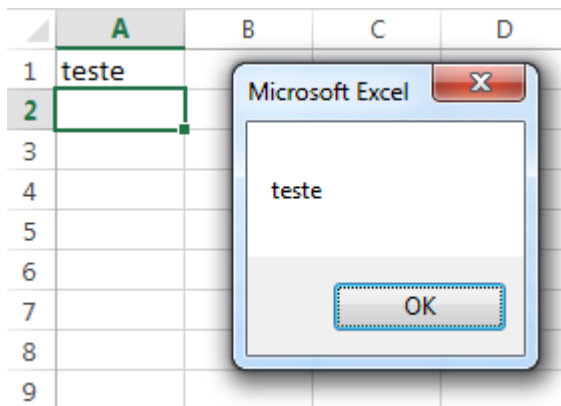


Então a medida que o usuário faz uma seleção diferente o código será executado e teremos um **MsgBox** com a quantidade das células selecionadas.

Outro evento que vamos exemplificar é o **Change** que funciona quando ocorre alguma alteração no **Target** selecionado, ou seja, sempre que alterarmos o valor de uma célula que esse código vai ser executado. Vamos utilizar esse evento na **Plan3**.

```
Private Sub Worksheet_Change(ByVal Target As Range)
    MsgBox (Target.Value)
End Sub
```

Esse código vai exibir um MsgBox com o valor da célula alterada, ou seja, assim que mudarmos o valor de alguma célula teremos essa caixa de mensagem com o valor que colocamos na célula.



Neste caso foi escrito a palavra **teste** na **célula A1** e pressionado a **tecla enter**, portanto o valor da célula foi alterado e se foi alterado o nosso código será executado.

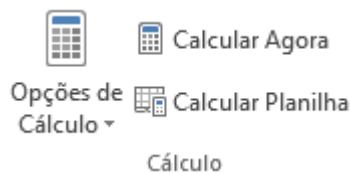
Na **Plan4** vamos utilizar o evento **Calculate**, que terá o código executado sempre que as informações da planilha forem calculadas. Vamos iniciar com o mesmo código da **Plan1**, para colocar a hora na célula, e vamos acrescentar uma fórmula dentro do ambiente Excel para quando a fórmula for calculada o nosso código será executado e teremos a hora sendo inserida na **célula A1**.

```
Private Sub Worksheet_Calculate()
    Range("A1").Value = Now()
    Range("A1").NumberFormat = "[$-F400]h:mm:ss AM/PM"
End Sub
```

Vamos colocar o **valor 3** na **célula C1** e na **célula D1** vamos colocar a **fórmula C1 + 5**. Assim que esse cálculo for realizado teremos a hora sendo inserida na **célula A1**.

	A	B	C	D
1	09:02:28		3	8
2				

Então toda vez que um cálculo for efetuado o código será executado. É importante lembrar que temos uma opção no Excel assim como utilizamos no VBA para o **cálculo manual**, ou seja, desta maneira o cálculo só será executado quando o usuário disser para o Excel efetuar o cálculo. Tendo o cálculo manual o código só será executado ao selecionar a opção **Calcular Agora**.



Essas opções ficam na guia **Fórmulas** do Excel, então o usuário poderá alterar entre o cálculo automático e manual. É importante ressaltar que essa opção de cálculo afeta a execução do código caso seja utilizado este evento.

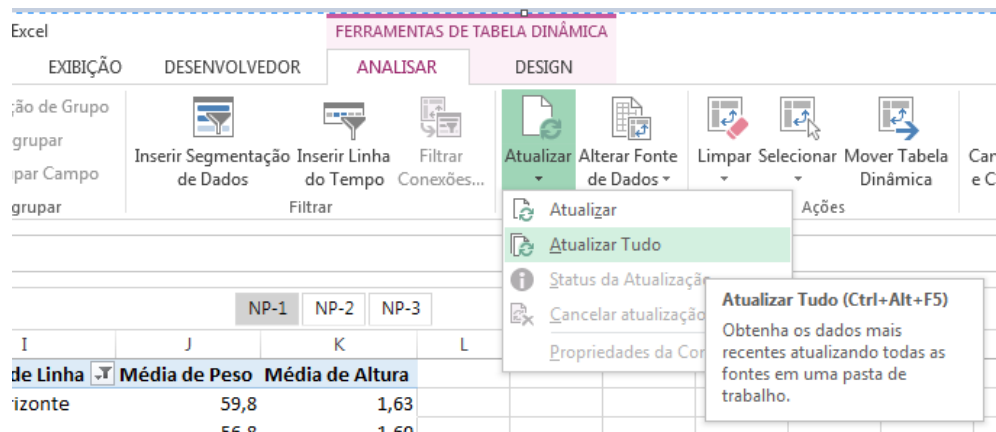
Foi possível aprender um pouco dos principais eventos de **Worksheet** no entanto o aluno poderá explorar os outros eventos e possibilidades assim como poderá criar diferentes códigos para testar a execução de cada um deles.

Sétima Ferramenta – Atualização Automática com Eventos, Impressão e Mudança de Arquivo

Nesta parte vamos construir a **Sétima Ferramenta** que será para **atualizar tabelas dinâmicas** de forma automática, pois sabemos que elas não se atualizam sozinhas quando fazemos alterações nas informações que geraram a tabela dinâmica. Portanto vamos fazer essa atualização de forma automática e sempre que o programa for salvo vamos atualizar um **PDF** com as informações da aba **Impressão** para que tenhamos sempre os dados mais atualizados em um arquivo em PDF.

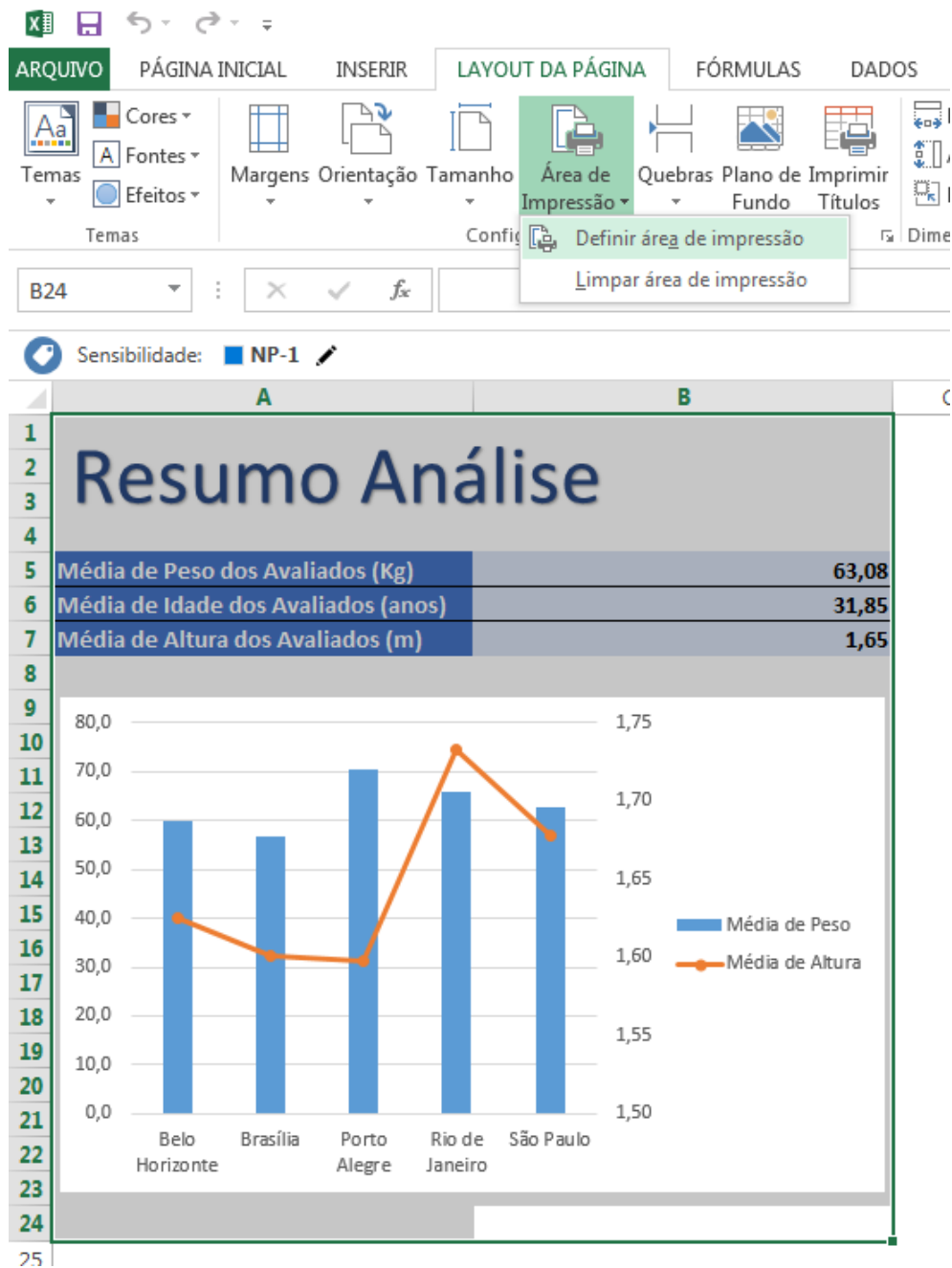
Por fim ao fechar o programa vamos registrar em outra planilha a data e a hora da última alteração que foi feita na planilha principal.

Como já aprendemos anteriormente sempre que precisamos escrever um código que não sabemos basta gravar uma macro com o gravador, então vamos prosseguir com essa gravação para gravar a atualização da tabela dinâmica.



Para abrir essa guia **Analisar** e **Design** é necessário clicar em algum dado da tabela dinâmica, feito isso essas abas serão habilitadas e podemos selecionar a opção de **Atualizar Tudo**.

Ainda na mesma gravação vamos mudar para a aba de Impressão, selecionar todos os dados a serem salvos (que neste caso vão da **célula A1** até a **célula B24**), em seguida vamos até a guia **Layout da Página** e vamos definir esse intervalo como **área de impressão**.



Feito isso vamos **salvar** essa área definida **como PDF** para finalizar a gravação da macro.

Nome do arquivo:

Tipo:

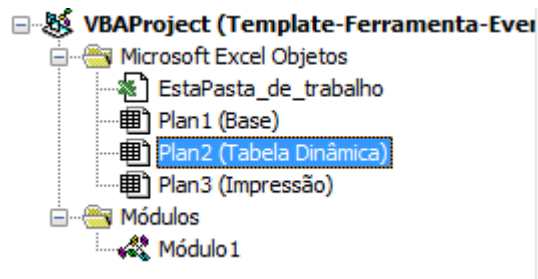
Feito isso podemos parar a gravação da macro e verificar quais os códigos foram gerados a partir dessa gravação.

Macros VBA para Excel Completo

```
Sub Macro1()  
    ActiveWorkbook.RefreshAll  
  
    Sheets("Impressão").Select  
    Range("A1:B24").Select  
    Range("B24").Activate  
    ActiveSheet.PageSetup.PrintArea = "$A$1:$B$24"  
    ActiveSheet.ExportAsFixedFormat Type:=xlTypePDF, Filename:= _  
        "E:\VBA\Relatório.pdf", Quality:=xlQualityStandard, IncludeDocProperties:= _  
        True, IgnorePrintAreas:=False, OpenAfterPublish:=False  
End Sub
```

Na primeira linha temos o código para a atualização da tabela dinâmica e nas linhas abaixo temos a parte da seleção dos dados, marcação como área de impressão e a parte de salvar como PDF.

O primeiro código que vamos escrever é dentro da aba Tabela Dinâmica, para que essa tabela seja atualizada sempre que essa aba seja ativada.



```
Private Sub Worksheet_Activate()  
    ActiveWorkbook.RefreshAll  
End Sub
```

Lembrando que utilizamos a opção de **atualizar tudo** na nossa gravação de macro, isso é importante, pois se houver mais de uma tabela dinâmica ao invés de atualizar uma a uma é possível atualizar todas ao mesmo tempo.

Agora vamos continuar com a outra parte do código que é para salvar a área de impressão em formato PDF. Basta selecionar a pasta de trabalho, selecionar o objeto **WorkBook** e o evento **AfterSave** (após salvar).

```
Private Sub Workbook_AfterSave(ByVal Success As Boolean)  
    Sheets("Impressão").PageSetup.PrintArea = "$A$1:$B$24"  
  
    Sheets("Impressão").ExportAsFixedFormat Type:=xlTypePDF, Filename:= _  
        ActiveWorkbook.Path & "\Relatório.pdf", Quality:=xlQualityStandard, IncludeDocProperties:= _  
        True, IgnorePrintAreas:=False, OpenAfterPublish:=False  
End Sub
```

Este é o código que será utilizado para a impressão. Foram feitas algumas alterações para que o código fique mais fácil e menor. A primeira alteração é tirar a parte inicial e deixar apenas o VBA indo até a aba Impressão ao invés de mudar de aba e selecioná-la.

Outra alteração é o local onde será salvo o relatório em PDF, como vamos salvar exatamente no mesmo local onde está a planilha do Excel podemos utilizar o código **ActiveWorkbook.Path** para obter o caminho deste arquivo e apenas concatenar com o nome do arquivo que vamos salvar e sua extensão.

É um código bem simples apenas para selecionar a área desejada, defini-la como área de impressão e salvá-la como PDF. Isso irá ocorrer após salvar a planilha.

Para finalizar será salvo em um outro arquivo chamado **Registro** a data e hora da última alteração que foi feita na planilha principal para que seja possível ter um arquivo de verificação da alteração desses dados.

Esse último código será escrito ainda dentro da pasta de trabalho juntamente com o código que salva a área de impressão em PDF.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    On Error Resume Next
    Workbooks.Open ("E:\VBA\Registro")

    Workbooks("Registro").Activate
    Range("A2").Value = Date
    Range("B2").Value = Now()
End Sub
```

Feito isso vamos abrir e/ou ativar nossa planilha **Registro** e salvar na **célula A2 a data atual** e na **célula B2 o horário atual**.

	A	B
	Data da Última	Hora da Última
1	Alteração	Alteração
2	22/07/2019	11:34:51
3		

Assim foi finalizada a construção da última ferramenta do curso. Lembrando que os alunos podem sempre aprimorar as ferramentas com o que foi ensinado no curso, ou seja, sempre será possível acrescentar mais códigos para melhorar o funcionamento do programa e se adequar as necessidades do usuário para cada uma de suas atividades.

Encerramento do Curso

Esse foi o curso de Macros VBA para Excel completo. O conteúdo do curso é um pouco complicado no início por se tratar de uma linguagem de código, no entanto a medida em que o aluno vai praticando se torna mais fácil executar certas atividades e acaba pegando prática com a programação em VBA.

Com o Excel já temos inúmeras formas de trabalhar, resolver problemas, criar gráficos, fazer análises, entre outras atividades. Agora com o VBA é possível fazer tudo isso e muito mais, pois temos como automatizar processos, deixar programas mais dinâmicos, facilitar análises e outras inúmeras atividades que vão depender apenas da necessidade e conhecimento do aluno.

O aprendizado de VBA não se limita apenas ao que foi visto no curso, assim como no Excel temos diversas ferramentas que não foram abordadas de forma aprofundada, mas foi possível ensinar uma boa base para entender o funcionamento da programação em VBA, da sintaxe utilizada, das ferramentas tanto do Excel quanto do VBA entre outros conceitos.

Portanto com o conhecimento adquirido é possível criar inúmeros programas diferentes para as mais diversas finalidades. Como foi visto diversas vezes caso o aluno não tenha conhecimento de alguma ferramenta ou código basta gravar a ação com o gravador de macro para verificar como o código foi escrito e como deverá utilizá-lo. É possível fazer buscas tanto na internet quanto no site da Microsoft para entender como certas ferramentas ou códigos funcionam.

Tendo toda essa base do curso qualquer informação a mais que seja necessária para a criação de um novo código será facilmente entendida e aplicada para esse novo problema.