



VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Código de exemplo usando funções de conversão de tipo:

```
Sub converter()
    Range("a1") = CInt(9.5555)
    Range("a2") = CSng(9999999999999#)
    Range("a3") = CDbl(9999999999#)
    Range("a4") = CBool(9.5555)
    Range("a5") = CDate(9.5555)
End Sub
```

Resultado abaixo após a conversão de tipo:

	A
1	10
2	1E+13
3	9999999999
4	VERDADEIRO
5	9/1/1900 13:19
6	12

Nota: Iremos explorar melhor esse recurso tratando entrada de dados.

CAPÍTULO 3 – USANDO ESTRUTURAS DE CONTROLE

As estruturas de controles são essenciais em qualquer linguagem de programação, são usadas para se alterar o fluxo de um programa (sequência de execução) direcionando a um resultado específico.

Existem dois tipos de estruturas de controle: **estruturas condicionais** e **estruturas de repetição** (laços de repetição), ambas existem e apresentam o momento certo de uso.

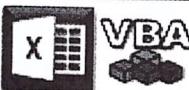
3.1. Estruturas Condicionais

São estruturas responsáveis em interromper o fluxo contínuo e sequencial de um programa, que com base em uma “decisão lógica”, desvia o processamento do código de forma apropriada.

Operadores Condicionais

São sinais que na linguagem indicam combinações lógicas de acordo a obter um resultado lógico que pode se destacar como **Verdadeiro (True)** ou **Falso (False)**. A esse conceito de combinações chamamos de **Lógica Booleana**.





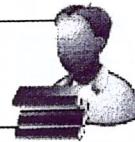
Operador	Função	Exemplos
=	Igual	5 = 3, resultado: False (falso) 3 = 3, Resultado: True (verdadeiro)
>	Maior que	5>3, resultado: True (verdadeiro) 3>5, resultado: False (falso)
<	Menor que	5<3, resultado: False (falso) 3<5, resultado: True (verdadeiro)
<=	Menor ou igual	5<=3, resultado: False (falso) 3<=5, resultado: True (verdadeiro) 3<=3, resultado: True (verdadeiro)
>=	Maior ou igual	5>=3, resultado: True (verdadeiro) 3>=5, resultado: False (falso) 3>=3, resultado: True (verdadeiro)
<>	Diferente	3<>5, resultado: True (verdadeiro) 3<>3, resultado: False (falso)

Operadores Lógicos

Os operadores lógicos têm a finalidade de complementar os operadores de comparação permitindo associar mais de uma comparação ou escolher uma entre várias comparações:

Operador	Função	Exemplos
AND	E Lógico	(5 = 5) And (3<4), resultado: True (verdadeiro) (5<>5) And (3<4), resultado: False (falso)
Obs (AND): Só uma das comparações for falsa o resultado será considerado falso.		
OR	Ou Lógico	(5 = 5) Or (3<4), resultado: True (verdadeiro) (5<>5) Or (3<4), resultado: True (Verdadeiro) (5=5) Or (3>4), resultado: True (Verdadeiro) (5<>5) Or (3>4), resultado: False (falso)
Obs (OR): Só é considerado falso se todas as comparações forem falsas.		
XOR	Ou Lógico Exclusivo	(5 = 5) XOr (3<4), resultado: False (verdadeiro) (5<>5) XOr (3<4), resultado: True (Verdadeiro) (5=5) XOr (3>4), resultado: True (Verdadeiro) (5<>5) XOr (3>4), resultado: False (falso)
Obs (XOR): É considerado verdadeiro se somente uma das condições for verdadeira.		
NOT	Inversão Lógica	Not(5>3), resultado: False (False) Not(5<3), resultado: True (Verdadeiro)





Obs (NOT): Inverte a lógica do resultado da comparação.

If...Then... End If

Essa estrutura testa uma condição, executando um determinado grupo de instruções se a condição ocorrer:

```
If <condição> Then  
<Executa grupo de instruções se a instrução for verdadeira>  
End If
```

Exemplo de código usando a instrução If...then...End If:

```
Sub Verifica()  
    If Range("B2") >= 7 Then  
        Range("C2") = "Aprovado"  
    End If  
End Sub
```

Aplicando esse código na planilha abaixo obtemos o resultado Aprovado se o valor da média for maior ou igual a 7:

	A	B	C
1	Aluno	Média	Resultado
2	Ana	7	Aprovado
3			
4	Verifica Resultado		

If...Then...Else...End If

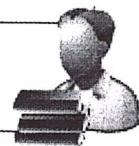
A Estrutura if...then...end if é utilizada em casos de análise simples onde somente uma condição basta, em casos mais complexos precisamos diversificar o resultado, nessa situação utilizamos o else.

Exemplo de uma estrutura If...Then...Else...End If:

```
If <condição> Then  
<Executa grupo de instruções se a instrução for verdadeira>  
Else  
<Executa grupo de instruções se a instrução for falsa>  
End If
```

Alterando o código anterior temos:

```
Sub Verifica()  
    If Range("B2") >= 7 Then  
        Range("C2") = "Aprovado"  
    Else  
        Range("C2") = "Reprovado"  
    End If  
End Sub
```



O resultado dessa modificação será dois tipos de resultados distintos que variam conforme o valor:

	A	B	C		A	B	C
1	Aluno	Média	Resultado	1	Aluno	Média	Resultado
2	Ana	7	Aprovado	2	Ana	3	Reprovado
3				3			
4		Verifica Resultado		4		Verifica Resultado	

Estruturas Encadeadas

Se houver um grande número de análises uma estrutura encadeada será perfeita, pois diversifica a análise de uma condição em várias condições agregadas. Abaixo, exemplo de código de uma estrutura encadeada:

```
Sub Verifica()
    If Range("B2") >= 7 Then
        Range("C2") = "Aprovado"
    Else
        If Range("B2") < 7 And Range("B2") > 5 Then
            Range("C2") = "Prova Final"
        Else
            If Range("B2") > 4 Then
                Range("C2") = "Recuperação"
            Else
                Range("C2") = "Reprovado"
            End If
        End If
    End If
End Sub
```

Observe que estruturas encadeadas apresentam uma natureza complexa, sua analisa se dá em diversas **eliminações condicionais** restando ao final da análise uma última **exceção condicional** quando as demais condições falharem.

Usando ElseIf

Se o uso de instruções encadeadas se tornarem muito complexo temos uma maneira mais simplificada e escrever estruturas encadeadas.

Esse recurso será usar a instrução **ElseIf** que é uma forma elegante de agregar diversas instruções **If...Else**.

Exemplo de código utilizando a instrução ElseIf:

```
Sub Verifica()
    If Range("B2") >= 7 Then
        Range("C2") = "Aprovado"
    ElseIf Range("B2") < 7 And Range("B2") > 5 Then
        Range("C2") = "Prova Final"
    ElseIf Range("B2") > 4 Then
        Range("C2") = "Recuperação"
    Else
        Range("C2") = "Reprovado"
    End If
End Sub
```

Repare que o grande problema das estruturas encadeadas é o fechamento dos If's caso ocorra uma falta de atenção do programador poderão ocorrer erros de sintaxe.





Usando If Composto

Essa é outra estrutura interessante que simplifica muito as estruturas If...Else, pois permite escrever essa estrutura lógica em uma única linha.

Exemplo de código utilizando IF composto:

Sub Verifica()

```
If Range("B2") >= 7 Then Range("C2") = "Aprovado" Else Range("C2") = "Reprovado"  
End Sub
```

Essa estrutura apesar de compacta não é indicada em estruturas encadeadas muito complexas pois será difícil analisar um código muito agrupado e longo.

Select Case...Case...End Select

Instruções Select Case, assim como as instruções If...Else, utilizadas para analizar condições e exibir resultados. Essas instruções adotam um conceito de "estudo de caso" onde é feito a verificação da ocorrência de valores exibido resultados conforme o valor agregado.

Abaixo, estrutura Select Case usando Else:

```
Select Case <expressão>  
    Case Valor_1  
        <Grupo de instruções_1>  
    Case Valor_2  
        <Grupo de instruções_2>  
End Select
```

Exemplo de código usando instruções Select Case:

```
Sub Verifica()  
    Dim valor As Single  
  
    valor = Range("B2")  
  
    Select Case valor  
        Case 10  
            Range("C2") = "Aprovado"  
        Case 9  
            Range("C2") = "Aprovado"  
        Case 8  
            Range("C2") = "Aprovado"  
        Case 7  
            Range("C2") = "Aprovado"  
        Case 6  
            Range("C2") = "Recuperação"  
        Case 5  
            Range("C2") = "Recuperação"  
        Case 4  
            Range("C2") = "Recuperação"  
        Case 3  
            Range("C2") = "Reprovado"  
        Case 2  
            Range("C2") = "Reprovado"  
        Case 1  
            Range("C2") = "Reprovado"  
        Case 0  
            Range("C2") = "Reprovado"  
    End Select  
End Sub
```





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Repare que cada caso gera um resultado segundo um “valor comparado”.

Podemos escrever essa estrutura de outra forma:

```
Sub Verifica()
    Dim valor As Single
    valor = Range("B2")
    Select Case valor
        Case Is >= 7
            Range("C2") = "Aprovado"
        Case Is >= 4
            Range("C2") = "Recuperação"
        Case Is >= 0
            Range("C2") = "Reprovado"
    End Select
End Sub
```

Usando comparador condicional “Is” é possível simplificar as comparações no Select Case.

Outra maneira de comparador condicional “To” que agrupa valores a serem comparados no Select Case:

```
Sub Verifica()
    Dim valor As Single
    valor = Range("B2")
    Select Case valor
        Case 7 To 10
            Range("C2") = "Aprovado"
        Case 4 To 6
            Range("C2") = "Recuperação"
        Case 0 To 3
            Range("C2") = "Reprovado"
    End Select
End Sub
```

Select Case...Case...Case Else...End Select

Assim como as instruções “If” a instrução “Select Case” possui um desvio condicional “Else”, a modificação no código não é extensa em relação ao código anterior:

```
Select Case <expressão>
    Case Valor_1
        <Grupo de instruções_1>
    Case Valor_2
        <Grupo de instruções_2>
    Case Else
        <Grupo de instruções_Else>
End Select
```





Exemplo de código Select Case usando Else: (Caso seja fornecido um valor menor negativo será exibida a última mensagem para o usuário).

```
Sub Verifica()
    Dim valor As Single

    valor = Range("B2")

    Select Case valor
        Case Is >= 7
            Range("C2") = "Aprovado"
        Case Is >= 0
            Range("C2") = "Reprovado"
        Case Else
            Range("C2") = "Digite um valor válido"
    End Select

End Sub
```

3.2. Estruturas de Repetição

As estruturas de repetição apresentam como principal característica a execução de instruções em ciclos, ou seja, elas permitem executar instruções mais de uma vez.

Uma ideia para melhor visualizar um loop, imagine um cálculo de soma de dois valores que será feito em uma planilha para depois exibir um resultado, exemplo do programa abaixo:

```
Sub soma()
    Range("C1") = Range("A1") + Range("B1")
End Sub
```

Olhando assim o programa é extremamente simples; agora você resolveu calcular mais valores, então foi feita a modificação abaixo:

```
Sub soma()
    Range("C1") = Range("A1") + Range("B1")
    Range("C2") = Range("A2") + Range("B2")
    Range("C3") = Range("A3") + Range("B3")
    Range("C4") = Range("A4") + Range("B4")
    Range("C5") = Range("A5") + Range("B5")
    Range("C6") = Range("A6") + Range("B6")
End Sub
```

Nesse ritmo, seu programa ficará muito grande, quanto mais linhas incluirmos no código maior ele será. Aprendendo estruturas de repetição esse tipo de construção de código poderá ser evitado.

For...Next

Esse tipo de estrutura tem como característica repetir um conjunto de instruções um número determinado de vezes. A sintaxe dessa estrutura é mostrada abaixo:

```
For <variável> = <valor inicial> To <valor final> Step <incremento>
    <Conjunto de instruções a serem executadas>
Next <variável>
```





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Exemplo abaixo de código usando um loop For...Next:

```
Sub loopFor()
    Dim cont As Integer

    For cont = 1 To 6 Step 1
        Range("C" & cont) = Range("A" & cont) + Range("B" & cont)
    Next cont
End Sub
```

Esse programa faz exatamente que o código anterior descreveu; a variável "cont" armazena o valor de contagem do loop onde será contado de 1 até 6 (To), essa contagem será sequencial de um em um (Step).

O resultado do código é mostrado abaixo:

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

Outra maneira de escrever esse código:

```
Sub loopFor()
    Dim cont As Integer

    For cont = 1 To 6
        Range("C" & cont) = Range("A" & cont) + Range("B" & cont)
    Next
End Sub
```

Step é opcional se o incremento for 1 (um) e o nome da variável após Next é opcional.

Caso seja alterado o valor do incremento (Step) para 2:

```
Sub loopFor()
    Dim cont As Integer

    For cont = 1 To 6 Step 2
        Range("C" & cont) = Range("A" & cont) + Range("B" & cont)
    Next
End Sub
```

Será exibido o resultado abaixo:

	A	B	C
1	5	6	11
2	5	3	
3	4	5	9
4	3	4	
5	3	4	7
6	6	7	





Do While...Loop (Loop Condicional)

O loop **Do While**, e demais loops que estudaremos agora, utilizam como princípio o uso de condições; caso essas condições apresentem determinado estado irá impedir do loop prosseguir ou não seu funcionamento.

A sintaxe abaixo representa a estrutura de um loop Do While:

```
Do While <condição>
    <conjunto de instruções>
    <variável> = <variável> + incremento
Loop
```

Embora estranha, a estrutura do loop Do While apresenta maior flexibilidade de modificações que a estrutura de repetição usando For. Veja um exemplo de um código usando uma estrutura Do While:

```
Sub loopDoWhile()
    Dim cont As Integer
    cont = 1 'Valor inicial do contador
    Do While cont < 7 'Condição executada no loop
        'Processamento executado
        Range("C" & cont) = Range("A" & cont) + Range("B" & cont)
        cont = cont + 1 'Contador de passo 1
    Loop
End Sub
```

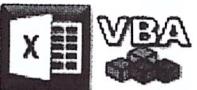
Olhando esse código faremos a seguinte interpretação:

1. A variável **cont** (que representa o contador) **inicia em 1**;
2. O valor da variável **cont** é testado dentro da **condição**;
3. Se a condição for válida (verdadeira) o loop segue adiante no ciclo;
4. Em seguida, um cálculo é realizado;
5. Então o contador **incrementa em um** (soma o valor da variável cont com um e armazena o valor)
6. O loop retorna a condição, executando o código até a **condição ser atingida**.

O resultado do loop é mostrado abaixo:

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





Observe que não houve diferença no resultado obtido com a estrutura for, altere o comportamento do loop, realizando as modificações na condição do loop:

```
Sub loopDoWhile()
    Dim cont As Integer

    cont = 1 'Valor inicial do contador

    Do While Range("A" & cont) <> "" 'Condição executada no loop

        'Processamento executado
        Range("C" & cont) = Range("A" & cont) + Range("B" & cont)

        cont = cont + 1 'Contador de passo 1
    Loop
End Sub
```

Preencha na planilha valores adicionais na coluna A e B:

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

Ao executar o código veremos o preenchimento do resultado nas demais linhas da planilha:

	A	B	C
1	5	6	11
2	5	4	9
3	4	5	9
4	3	4	7
5	3	4	7
6	6	7	13
7	8	5	13
8	9	6	15
9	8	3	11
10	5	2	7

Como isso ocorreu? Na condição anterior estipulamos que o funcionamento do loop era feito verificando o limite do contador sendo menor que 7 atuando sobre a alteração da linha no endereço de célula.

Nessa nova condição foi verificado se o conteúdo das células da coluna B não se encontra vazio (diferente de vazio); caso não se encontre o loop funcionará.

Do Until...Loop (Loop Condicional)

O loop Do Until tem um comportamento semelhante ao loop Do While; a diferença fica a cargo do tratamento da condição, o loop Do While é executado se a condição ocorrida for verdadeira (TRUE), já no loop Do Until funciona somente se a condição ocorrida for falsa (FALSE).





Comparação de um código Do While e Do Until

Loop Do While: (Condição Verdadeira)

```
Sub loopDoWhile()
    Dim cont As Integer
    cont = 1
    Do While Range("A" & cont) <> ""
        Range("C" & cont) = Range("A" & cont) + Range("B" & cont)
        cont = cont + 1
    Loop
End Sub
```

Loop Do Until: (Condição Falsa)

```
Sub loopDoUntil()
    Dim cont As Integer
    cont = 1
    Do Until Range("A" & cont) = ""
        Range("C" & cont) = Range("A" & cont) + Range("B" & cont)
        cont = cont + 1
    Loop
End Sub
```

Nota: O emprego do loop **Do While** e **Do Until** dependerá de adequação a lógica desenvolvida no programa que puramente das características funcionais de cada loop.

While...Wend (Loop Condicional)

O loop While...Wend é um tipo de loop Do While adaptado para ser compatível com outras linguagens, meramente é um loop Do While escrito ligeiramente diferente.

Loop Do While:

```
Sub loopDoWhile()
    Dim cont As Integer
    cont = 1
    Do While Range("A" & cont) <> ""
        Range("C" & cont) = Range("A" & cont) + Range("B" & cont)
        cont = cont + 1
    Loop
End Sub
```

Loop While...Wend:

```
Sub loopWhile()
    Dim cont As Integer
    cont = 1
    While Range("A" & cont) <> ""
        Range("C" & cont) = Range("A" & cont) + Range("B" & cont)
        cont = cont + 1
    Wend
End Sub
```

Reparou como a escrita dos loops é parecida? Em programação feita é um código podemos optar pela utilização de ambas as estruturas.

Do...Loop While / Do ...Loop Until (Loop Condicional)

Esses loops Do...Loop While e Do ... Loop Until são semelhantes aos loops Do While e Do Until, a diferença fica a cargo da posição da condição no loop, que vem ao final.





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno



Veja a representação dos loops Do...Loop While e Do...Loop Until abaixo

Do Loop While:

```
Sub Do loopWhile()
    Dim cont As Integer
    cont = 1
    Do
        Range("C" & cont) = Range("A" & cont) + Range("B" & cont)
        cont = cont + 1
    Loop While Range("A" & cont) <> ""
End Sub
```

Loop While...Wend:

```
Sub DoloopUntil()
    Dim cont As Integer
    cont = 1
    Do
        Range("C" & cont) = Range("A" & cont) + Range("B" & cont)
        cont = cont + 1
    Loop Until Range("A" & cont) = ""
End Sub
```

Na prática essas duas estruturas de repetição apresentam funcionamento semelhante ao loop com condição inicial, a principal característica funcional fica a caráter da condição, nos loops com **condição inicial** se a condição **não for satisfeita** o loop tem o **ciclo interrompido**.

Caso a condição vem ao final, mesmo que ela não seja satisfeita um círculo de instruções **será executado pelo menos uma única vez**.

For...Each

Essa estrutura de repetição se diferencia das demais estruturas anteriores, pois ela apresenta como característica atuar sobre **conjunto de objetos (coleção)**, quando nos referimos a conjunto significa, por exemplo, conjunto de células, conjunto de planilhas, conjunto de campos e etc.

Abaixo é apresentada a sintaxe do loop For Each:

```
For Each <Variável> In <coleção de objetos>
    <conjunto de instruções a serem executadas>
Next <nome do objeto>
```

Abaixo um exemplo de um código usando o loop For Each:

```
Sub loopDoWhile()
    Dim celula As Range
    cont = 1
    For Each celula In Range("C1:C10")
        celula.Formula = "=A" & cont & "+B" & cont
        cont = cont + 1
    Next
End Sub
```

Nesse código o **conjunto** usado no loop For Each fica representada pela instrução `Range("C1:C10")` que representa o grupo de células onde o loop atualizará.

Combinando Estruturas condicionais e Repetição

Em alguns códigos que utilizam estruturas de repetição a casos que precisa se combinar estruturas condicionais para realizar códigos mais sofisticados.



**VBA**

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Para testar na prática esse conceito construa a planilha abaixo:

	A	B	C
1	Aluno	Média	Resultado
2	Carolina	7	
3	Marcos	6	
4	Pedro	4	
5	Ana	8	
6	Paulo	5,5	
7	Sérgio	9	

Construa o código a seguir:

```
Sub Estruturas()
    Dim media As Double, i As Byte
    i = 2
    'Estrutura de repetição percorrendo a planilha
    Do While Cells(i, 2) <> ""
        'Estrutura condicional verificando a média do aluno
        If Cells(i, 2) >= 7 Then
            Cells(i, 3) = "Aprovado"
        Else
            Cells(i, 3) = "Reprovado"
        End If
        i = i + 1
    Loop
End Sub
```

O resultado do código é mostrado abaixo:

	A	B	C
1	Aluno	Média	Resultado
2	Carolina	7	Aprovado
3	Marcos	6	Reprovado
4	Pedro	4	Reprovado
5	Ana	8	Aprovado
6	Paulo	5,5	Reprovado
7	Sérgio	9	Aprovado

Interrompendo fluxo de processamento: Exit Do, Exit For e Exit Sub

A alguns casos que interromper o fluxo de um processamento de um código poderá ser útil, existem dois tipos de fluxo que poderemos interromper: o fluxo de rotina e o fluxo de repetição.



**VBA**

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

No código abaixo forçamos uma interrupção em um loop Do While usando Exit Do:

```
Sub Estruturas()
    Dim media As Double, i As Byte

    i = 2

    Do While Cells(i, 2) <> ""
        If Cells(i, 2) >= 7 Then
            Cells(i, 3) = "Aprovado"
        Else
            Exit Do
        End If
        i = i + 1
    Loop

    Cells(i, 3) = "Parei por aqui!"
End Sub
```

O Resultado do loop é mostrado abaixo: Após a interrupção do loop qualquer instrução após o loop será executada.

	A	B	C
1	Aluno	Média	Resultado
2	Carolina	7	Aprovado
3	Marcos	6	Parei por aqui!
4	Pedro	4	
5	Ana	8	
6	Paulo	5,5	
7	Sérgio	9	

Para estruturas de repetição For...Next se faz interrupção por Exit For. Veja o código de exemplo abaixo:

```
Sub Estruturas()
    Dim media As Double, i As Byte

    For i = 2 To 7
        If Cells(i, 2) >= 7 Then
            Cells(i, 3) = "Aprovado"
        Else
            Exit For
        End If
    Next

    Cells(i, 3) = "Parei por aqui!"
End Sub
```

O resultado será semelhante ao código anterior.





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Caso você use **Exit Sub** dentro de um fluxo ele será **interrompido totalmente**. Veja o código abaixo:

```
Sub Estruturas()
    Dim media As Double, i As Byte

    For i = 2 To 7

        If Cells(i, 2) >= 7 Then
            Cells(i, 3) = "Aprovado"
        Else
            Exit Sub
        End If
    Next

    Cells(i, 3) = "Parei por aqui!"
End Sub
```

Observe que o código não conseguiu agora exibir a mensagem fora do fluxo:

	A	B	C
1	Aluno	Média	Resultado
2	Carolina	7	Aprovado
3	Marcos	6	
4	Pedro	4	
5	Ana	8	
6	Paulo	5,5	
7	Sérgio	9	

3.3. Verificando Tipo

Existem funções especiais na linguagem chamadas **verificador de tipo**, essas funções retornam um valor **booleano** verdadeiro (True) ou falso (False) ao se verificar o estado de um valor é válido ou não.

Abaixo uma tabela contendo os principais verificadores de tipos usados na linguagem:

Verificador de Tipo	Função
IsNumeric	Se for número retorna TRUE
IsDate	Se for data retorna TRUE
IsNull	Se valor for nulo retorna TRUE
IsEmpty	Se valor estiver vazio ou "" retorna TRUE
IsError	Se valor contiver erro retorna TRUE





Exemplo de código utilizando verificadores de tipo:

```
Sub Estruturas()
    Dim i As Byte

    For i = 2 To 7

        If IsEmpty(Cells(i, 2)) Then
            Cells(i, 3) = "Célula Vazia"
        ElseIf IsDate(Cells(i, 2)) Then
            Cells(i, 3) = "Isso é uma data"
        ElseIf Not IsNumeric(Cells(i, 2)) Then
            Cells(i, 3) = "Não é um número"
        ElseIf Cells(i, 2) >= 7 Then
            Cells(i, 3) = "Aprovado"
        Else
            Cells(i, 3) = "Reprovado"
        End If
    Next
End Sub
```

Resultado do código após a execução:

	A	B	C
1	Aluno	Média	Resultado
2	Carolina	7	Aprovado
3	Marcos	Nota 6	Não é um número
4	Pedro	4	Reprovado
5	Ana		Célula Vazia
6	Paulo	5,5	Reprovado
7	Sérgio	12/5/2010	Isso é uma data

3.4. Comparando Strings

Strings é um tipo de informação bastante manipulada dentro da linguagem, geralmente quando queremos realizar algum tipo de comparação utilizamos uma estrutura condicional.

Podemos utilizar o operador “=” (igual) para comparar duas strings, a também o operador Like que permite realizar comparações complexas em cadeias de string. Veja exemplos de códigos demonstrando exemplos de comparação:

Comparação simples entre duas Strings:

O programa abaixo resultara em uma afirmação positiva, pois não existe diferença entre as strings.

```
Sub comparandoStrings()
    Dim string1, string2 As String

    string1 = "Bom dia": string2 = "Bom dia"

    If string1 = string2 Then
        resultado = "As Strings são idênticas"
    Else
        resultado = "As Strings são diferentes"
    End If

End Sub
```





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Já o programa abaixo resultará em uma afirmação negativa, embora o texto seja o mesmo, a diferença entre letras maiúsculas e minúsculas resultará em uma afirmação negativa.

```
Sub comparandoStrings()
    Dim string1, string2 As String

    string1 = "Bom dia": string2 = "BOM DIA"

    If string1 = string2 Then
        resultado = "As Strings são idênticas"
    Else
        resultado = "As Strings são diferentes"
    End If

End Sub
```

Comparação complexa entre duas Strings:

Agora imagine uma situação onde queremos localizar se existe uma palavra em uma String muito grande, o comparador igual não irá nos atender, para isso utilizamos o operador like.

```
Sub comparandoStrings()
    Dim string1, string2 As String

    string1 = "Bom dia Brasil": string2 = "Brasil"

    If string1 Like "*" + string2 Then
        resultado = "A palavra existe na string"
    Else
        resultado = "A palavra não existe na string"
    End If

End Sub
```

O resultado será que “A Palavra existe na string”, isso ocorre porque o comparador “Like” pode associar a string comparada a um “Escopo”, esse escopo apresenta a função de ignorar o resto da string para permite uma busca específica.

Exemplos de escopos utilizados com Like:

Escopos utilizados com Like	Definição	Exemplos
*	Substitui o restante de uma String.	“Casa*”, “*Casa”, “*Casa*”
?	Substitui um caracter de uma String.	“C?s?”, “?asa”, “Ca?a”
Pode-se ainda combinar os dois escopos:		“Casa?*”, “*a?a”, “????dia*”

Outro recurso interessante é ignorar a diferença entre maiúsculas e minúsculas através da instrução Option Compare Text.





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Essa instrução deve ser inserida de forma isolada fora do código, veja um exemplo abaixo:

```
Option Compare Text
Sub comparandoStrings()
    Dim string1, string2 As String

    string1 = "BRASIL": string2 = "brasil"

    If string1 = string2 Then
        resultado = "As strings são idênticas"
    Else
        resultado = "As strings são diferentes"
    End If

End Sub
```

Caso as strings estejam com texto idêntico, mas haja variação entre caixa alta e baixa (maiúscula e minúscula) terá um resultado afirmativo pois as strings tem texto semelhante.

CAPÍTULO 4 – REGISTROS E MATRIZES

Em programação estudamos o uso de variáveis, aprendemos que variáveis são essenciais para codificação no VBA, pois a principal característica de uma variável é o armazenamento de dados na memória para futura utilização.

O problema de utilizar variáveis simples será que alguns casos o uso se torna constante e em larga escala, nesse caso uma variável simples não funciona, aprenderemos nesse capítulo sobre o uso de matrizes e registros para suprir essa necessidade exigida em alguns códigos.

Matrizes

Sobre Matrizes:

Matrizes são variáveis que apresentam como principal qualidade o **armazenamento múltiplo de dados** onde a mesma variável é associada a um **índice**, esse índice tem a função de acessar determinada informação na matriz. As matrizes podem ser chamadas também pelo nome de **Arrays** ou **Vetores**.

Abaixo é exibido um comparativo de trecho de código usando variáveis e matrizes:

Usando Variáveis:

```
Sub UsoVariavel()
    'Declaração de variáveis
    Dim nome1, nome2, nome3, nome4 As String

    'Atribuição de valores a variáveis
    nome1 = "Ana"
    nome2 = "Pedro"
    nome3 = "Paulo"
    nome4 = "Carlos"

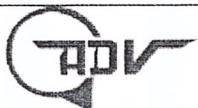
    'Retorno de valor da variável
    Range("A2") = nome1
End Sub
```

Usando Matrizes:

```
Sub UsoMatriz()
    'Declaração de uma matriz
    Dim nome(4) As String

    'Atribuição de valores a matriz
    nome(0) = "Ana"
    nome(1) = "Pedro"
    nome(2) = "Paulo"
    nome(3) = "Carlos"

    'Retorno de valor da matriz
    Range("A2") = nome(0)
End Sub
```



**VBA**

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Repare que os dois códigos são semelhantes, ao usar variáveis declaramos cada variável utilizada pelo código e no uso de matrizes declaramos uma única matriz.

A matriz apresenta um valor de índice agregado que foi utilizado tanto para realizar atribuição como na leitura de valores de uma matriz.

Nota: Na prática, a principal diferença no uso de variáveis e matrizes será o controle maior no acesso aos dados na matriz, pois o índice facilita a referência a esses valores em relação às variáveis.

Alterando o Índice da Matriz:

Em toda matriz quando criada é obrigatório a iniciar o índice em zero, isso não é uma regra pois o valor inicial do índice pode ser facilmente alterado através da instrução **Option Base**. O código abaixo altera o índice inicial da matriz:

```
Option Base 1
Sub UsoMatriz()
    'Declaração de uma matriz
    Dim nome(4) As String

    'Atribuição de valores a matriz
    nome(1) = "Ana"
    nome(2) = "Pedro"
    nome(3) = "Paulo"
    nome(4) = "Carlos"

    'Retorno de valor da matriz
    Range("A2") = nome(1)
End Sub
```

Observe que o índice da matriz foi alterado para 1(um), ou seja, em **termos de codificação** o valor zero é **pouco prático** pouco prático e o **valor 1** é mais amigável na programação.

Nota: Índices incorretos podem gerar mensagens de erro no código processado.

Redimensionando o Índice da Matriz (Matrizes Dinâmicas):

Um dos problemas em se manusear matrizes está na rigidez do tamanho estipulado na matriz, essa limitação pode ser facilmente contornada redimensionando o índice através da instrução "Redim".



ADV – Curso de Informática

Av. Treze de Maio, 23 – 8º andar – Centro – Rio de Janeiro - RJ

<http://www.cursoadv.com.br/>

Telefone: (21) 2210-1180

Página 51



VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Veja o código de exemplo de código abaixo onde é feito o redimensionamento de uma matriz:

```

Sub UsoMatriz()
    ' Declaração de uma matriz sem índice
    Dim nome() As String

    'Redimensionando a matriz para 4
    ReDim nome(4)

    nome(0) = "Ana"
    nome(1) = "Pedro"
    nome(2) = "Paulo"
    nome(3) = "Carlos"

    For i = 0 To 3
        Cells(i + 2, 1) = nome(i)
    Next

    ReDim nome(5)

    'Redimensionando a matriz para 5
    nome(0) = "Ana"
    nome(1) = "Pedro"
    nome(2) = "Paulo"
    nome(3) = "Carlos"
    nome(4) = "Adriana" 'Novo elemento da matriz

    For i = 0 To 4
        Cells(i + 2, 2) = nome(i)
    Next

End Sub

```

Repare nesse código que inicialmente a matriz é declarada sem índice, depois a matriz é redimensionada para “4”, em seguida foi redimensionada para “5”. O resultado abaixo demonstra o comportamento da matriz:

	A	B
1	Antes	Depois
2	Ana	Ana
3	Pedro	Pedro
4	Paulo	Paulo
5	Carlos	Carlos
6		Adriana

Preservando os valores de uma Matriz (Matrizes Dinâmicas):

No redimensionamento que executamos no código anterior usamos valores similares para cada redimensionamento. Ao redimensionar uma matriz os valores são automaticamente excluídos, por exemplo:

Matriz com 3 elementos:		
Elemento(0):	Elemento(1):	Elemento(2):
Ana	Carlos	Bianca

Matriz de 4 elementos:			
Elemento(0):	Elemento(1):	Elemento(2):	Elemento(3):
Flávio	?	?	?





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Se esses valores não forem repassados a matriz é “apagada”, para resolver existe a instrução “Preserve” que mantém os valores anteriores armazenados caso a matriz seja redefinida para um tamanho maior.

Exemplo de código usando a instrução Preserve:

```
Sub UsoMatriz()
    'Declaração de uma matriz sem índice
    Dim nome() As String

    'Redimensionando a matriz para 4
    ReDim nome(4)

    nome(0) = "Ana"
    nome(1) = "Pedro"
    nome(2) = "Paulo"
    nome(3) = "Carlos"

    For i = 0 To 3
        Cells(i + 2, 1) = nome(i)
    Next

    ReDim Preserve nome(5)

    'Redimensionando a matriz para 5
    nome(4) = "Adriana" 'Novo elemento da matriz

    For i = 0 To 4
        Cells(i + 2, 2) = nome(i)
    Next

End Sub
```

Note que nesse código não foi necessário repassar os valores anteriores a matriz, pois forma “preservados” sendo adicionado apenas o valor novo a matriz.

Registros

Registros é um recurso da linguagem que apresenta um uso bastante interessante, geralmente programadores vão criando variáveis dentro do código à medida que vai senso necessário, os registros adotam um conceito de **variáveis em categoria**.





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Abaixo é exibido um código com e sem uso de registros:

Código sem registro:

```
Sub Registros()
    'Declaração de variáveis
    Dim Produto As String
    Dim Categoria As String
    Dim Quantidade As Integer
    Dim preco As Single

    Produto = "Refrigerante"
    Categoria = "Bebidas"
    Quantidade = 10
    preco = 1.4

    Range("A2") = Produto
    Range("B2") = Categoria
    Range("C2") = Quantidade
    Range("D2") = preco
End Sub
```

Código usando registro:

```
'Criação de um registro
Type venda
    Produto As String
    Categoria As String
    Quantidade As Integer
    preco As Single
End Type

Dim Supermercado As venda
Sub Registros()

    'Atribuindo valores ao registro
    Supermercado.Produto = "Refrigerante"
    Supermercado.Categoria = "Bebidas"
    Supermercado.Quantidade = 10
    Supermercado.preco = 1.4

    Range("A2") = Supermercado.Produto
    Range("B2") = Supermercado.Categoria
    Range("C2") = Supermercado.Quantidade
    Range("D2") = Supermercado.preco
End Sub
```

Embora possa parecer que o código usando registro pareça muito mais trabalhoso, os registros tornam o uso de variável mais elegante, pois elas assumem um aspecto de **grupo de categorias**.

Observe a criação abaixo do registro:

```
Type venda
    Produto As String
    Categoria As String
    Quantidade As Integer
    preco As Single
End Type
```

O registro recebeu o **identificador “venda”**, esse nome representam um **conjunto de variáveis** associadas ao **mesmo registro**. A variável “supermercado” foi associada ao registro permitindo acessar as variáveis internas do registro:

Dim Supermercado As venda

Transforme a variável supermercado em uma matriz conforme a alteração abaixo:

Dim Supermercado(2) As venda





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Essa modificação permitirá um uso do registro mais robusto, veja o exemplo abaixo:

```
'Criação de um registro
Type venda
    Produto As String
    Categoria As String
    Quantidade As Integer
    preco As Single
End Type

Dim Supermercado(2) As venda
Sub Registros()

    'Atribuindo valores ao registro
    Supermercado(0).Produto = "Refrigerante"
    Supermercado(0).Categoria = "Bebidas"
    Supermercado(0).Quantidade = 10
    Supermercado(0).preco = 1.4

    Supermercado(1).Produto = "Contra Filé"
    Supermercado(1).Categoria = "Carnes"
    Supermercado(1).Quantidade = 5
    Supermercado(1).preco = 7

    Dim linha, cont As Integer
    cont = 0
    For linha = 2 To 3
        Range("A" & linha) = Supermercado(cont).Produto
        Range("B" & linha) = Supermercado(cont).Categoria
        Range("C" & linha) = Supermercado(cont).Quantidade
        Range("D" & linha) = Supermercado(cont).preco
        cont = cont + 1
    Next
End Sub
```

Observe que ao tornar a variável associada ao registro uma matriz permitirá aumentar a utilização de registros pelo código. Veja o resultado abaixo:

	A	B	C	D
1	Produto	Categoria	Quantidade	Preço
2	Refrigerante	Bebidas	10	R\$ 1,40
3	Contra Filé	Carnes	5	R\$ 7,00

CAPÍTULO 5 – MANIPULANDO CÉLULAS

Nesse capítulo aprenderemos a manipular o principal meio de contato do usuário com o Excel, as planilhas, nelas temos células que armazenam o conteúdo de dados que podem ser os mais variados.

Podemos basicamente realizar todas as operações feitas diretamente na planilha pelo VBA tais como: Seleção, Inserção, Formatação, enfim, tudo que você precisa para criar planilhas rápidas e dinâmicas.

Seleção Estática de Células

A seleção de células estáticas é algo muito simples, realizamos esse tipo de seleção diversas vezes no Excel, onde precisamos aplicar alguma alteração em células específicas.





Seleção Simples:

Basicamente essa seleção é feita referenciando a célula específica acompanhada da propriedade "Select".

	A	B	C	D	E	F	G
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21

Código correspondente:

```
Sub selecaoSimples()
    Range("D2").Select
End Sub
```

Seleção de um conjunto de células:

Seleciona-se uma range em um conjunto agrupado de células do início ao fim.

	A	B	C	D	E	F	G
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21

Código correspondente:

```
Sub selecaoAgrupada()
    Range("A1:G1").Select
End Sub
```

Seleção alternada de células:

É semelhante à seleção anterior, porém é intercalado um conjunto variado de células.

	A	B	C	D	E	F	G
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21

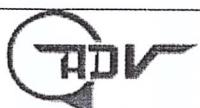
Código correspondente:

```
Sub selecaoAlternada()
    Range("G3,E2,C2,A3").Select
End Sub
```

Seleção Dinâmica de Células

Essa seleção possui uma aplicação mais sofisticada em relação às seleções anteriores, pois nem sempre temos como prever o limite de uma seleção em planilhas, geralmente o conteúdo de uma planilha é muito dinâmico.

Por exemplo, imagine que você queira formatar uma planilha que sofre entrada de novas linhas constantemente, nesse caso você poderá pensar em alternativas como loops, porém nem sempre isso é interessante sob o risco de constituirmos códigos muito complexos, veja as alternativas a seguir de seleções dinâmicas.





Seleção de uma área de células:

Essa seleção é muito útil se possuirmos um conjunto uniforme de células.

	A	B	C	D	E	F	G
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21

Código correspondente:

```
Sub selecaoArea()
    Range("A1").CurrentRegion.Select
End Sub
```

Observe que usamos a instrução **CurrentRegion**, ela verifica células próximas a célula mencionada para realizar uma seleção de uma área inteira de células.

Seleção o final de uma área de células:

Caso queira selecionar o final de uma área de células informamos a **direção de deslocamento** a seleção através da **instrução “End”** acompanhada da direção de deslocamento da área (**xlUp**, **xlDown**, **xlToLeft**, **xlToRight**).

1. Selecionando a última célula abaixo na área de células:

	A	B	C	D	E	F	G
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21

A instrução Range é associada à instrução **End(xlDown).Select**. Exemplo do código abaixo:

```
Sub selecionaUltimaCelulaAbaixo()
    Range("A1").End(xlDown).Select
End Sub
```

2. Selecionando a última célula à direita na área de células:

	A	B	C	D	E	F	G
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21

A instrução Range é associada à instrução **End(xlToRight).Select**. Exemplo do código abaixo:

```
Sub selecionaUltimaCelulaDireita()
    Range("A3").End(xlToRight).Select
End Sub
```

3. Selecionando a última célula acima na área de células:

	A	B	C	D	E	F	G
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21



**VBA**

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

A instrução Range é associada à instrução End(xlUp).Select. Exemplo do código abaixo:

```
Sub selecionaUltimaCelulaAcima()
    Range("G3").End(xlUp).Select
End Sub
```

4. Selezionando a última célula à esquerda na área de células:

	A	B	C	D	E	F	G
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21

A instrução Range é associada à instrução End(xlToLeft).Select. Exemplo do código abaixo:

```
Sub selecionaUltimaCelulaEsquerda()
    Range("G1").End(xlToLeft).Select
End Sub
```

Seleção conjunto de células em uma área de células:

Assim como a seleção de área, o VBA deduz a seleção de um conjunto de células a partir de uma célula inicial, sendo que essa seleção será direcionada.

Além da instrução End e dos deslocamentos de área utilizamos em conjunto a instrução "Selection" que permite gerar uma sequência de células a partir de uma célula inicial

1. Selezionando uma sequência de células na Horizontal:

	A	B	C	D	E	F	G
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21

Podemos selecionar uma sequência de células tanto em direção a esquerda (xlToLeft) como para a direita (xlToRight).

Veja o código abaixo:

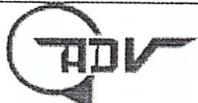
```
Sub selecionarSequenciaHorizontal()
    Range("A1").Select
    Range(Selection, Selection.End(xlToRight)).Select
End Sub
```

Esse código também pode ser construído assim (eliminando a instrução selection):

```
Sub selecionarSequenciaHorizontal()
    Range("A1", Range("A1").End(xlToRight)).Select
End Sub
```

2. Selezionando uma sequência de células na Vertical:

	A	B	C	D	E	F	G
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Podemos selecionar uma sequência de células tanto em direção acima (xlUp) como abaixo (xlDown).

Veja o código abaixo:

```
Sub selecionarSequenciaVertical()
    Range("A1").Select
    Range(Selection, Selection.End(xlDown)).Select
End Sub
```

Esse código também pode ser construído assim (eliminando a instrução selection):

```
Sub selecionarSequenciaVertical()
    Range("A1", Range("A1").End(xlDown)).Select
End Sub
```

Manipulando Valores da Célula

Ao manipular as células, além da seleção, é possível manipular os valores da célula através de algumas propriedades que iremos abordar agora.

1. Obtendo o valor da célula: Usando a propriedade “Value” (Padrão) retornamos ou atribuímos qualquer valor à célula, caso você omita essa propriedade será subentendida seu uso pelo VBA.

Exemplo de Código:

```
Sub obterValores()
    Range("A1").Value = "Bom dia"
    resultado = Range("A1")
End Sub
```

Resultado Obtido: Retorna o texto “Bom dia”.

2. Obtendo o Endereço da célula: Caso nosso código necessite obter a posição física do endereço de uma célula a propriedade “AddressLocal” será utilizada.

Exemplo de Código:

```
Sub obterEnderecoCelula()
    resultado = Range("A1").AddressLocal
End Sub
```

Resultado Obtido: “\$A\$1” (O endereço é obtido como uma referência absoluta).

3. Retornando a referência a posição da coluna da célula: É possível também obter a referência numérica da posição coluna da célula selecionada através da propriedade “Column”.

Exemplo de Código:

```
Sub obterPosicaoColuna()
    resultado = Range("A1").Column
End Sub
```

Resultado Obtido: Retorna o valor 1 (coluna “A”).

4. Retornando a referência a posição da linha da célula: O Mesmo pode ser feito em relação à posição da linha através propriedade “Row” retornando a referência numérica.



ADV – Curso de Informática

Telefone: (21) 2210-1180

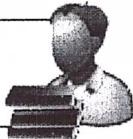
Av. Treze de Maio, 23 – 8º andar – Centro – Rio de Janeiro - RJ

<http://www.cursoadv.com.br/>

Página 59

**VBA**

MICROSOFT EXCEL 2013 COM VBA

Apostila do AlunoExemplo de Código:

```
Sub obterPosicaoLinha()
    resultado = Range("A1").Row
End Sub
```

Resultado Obtido: Retorna o valor 1 (Linha 1).

5. Inserindo fórmulas do Excel: Quando programamos código VBA geralmente desenvolvemos fórmulas no código, através da propriedade “FormulaLocal” podemos inserir fórmulas do Excel da mesma forma que um usuário faz na planilha.

Exemplo de código abaixo:

```
Sub inserindoFormula()
    Range("F2").FormulaLocal = "=MÉDIA(B2:E2)"
End Sub
```

Resultado Obtido: Na célula F2 será inserida uma fórmula de média, você poderá conferir que a fórmula foi inserida na planilha através da barra de fórmulas.

6. Localizando valores de um conjunto de células: Existe uma possibilidade interessante com células, a de localizar valores específicos de um conjunto de células através da instrução “Find”.

Construa a planilha abaixo:

	A	B	C	D	E
1	Produto	Valor		Produto procurado:	Feijão
2	Arroz	R\$ 1,45		Valor resultante:	
3	Feijão	R\$ 1,60			
4	Batata	R\$ 2,25			
5	Farinha	R\$ 2,67			
6	Vinagre	R\$ 3,45			
7	Óleo	R\$ 5,40			

Construa o código abaixo:

```
Sub localizarValor()
    'Selecione conjunto de células
    Range("A2", Range("A2").End(xlDown)).Select

    'Obtem o produto referente a busca
    Dim busca As String
    busca = Range("E1").Value

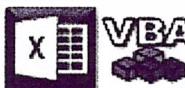
    'Realiza a busca do valor do produto
    Dim linha As Integer
    linha = Selection.Find(busca).Row

    'Retorna o valor do produto
    Range("E3") = Cells(linha, 2)
End Sub
```

Resultado obtido: Ao usarmos a instrução “Find” foi localizada a palavra “Feijão” no conjunto de células selecionadas retornando a posição da linha correspondente e obtendo em seguida o valor correspondente do produto na coluna B.

D	E
Produto procurado:	Feijão
Valor resultante:	R\$ 1,60





CAPÍTULO 6 – FORMATANDO CÉLULAS

Neste capítulo iremos abordar o conceito de formatação visual da planilha podendo realizar diversas modificações na planilha através do VBA.

Formatando Valores

Os valores de uma célula poderão ser rapidamente formatados pelo VBA, existem diversos tipos de formatos que podemos aplicar aos valores como, por exemplo: formato moeda, formato decimal, formato de data. Esse tipo de alteração visual é também chamado de “máscara numérica”. A função que utilizaremos para formatar valores é a função “Format”.

Exemplo de código que aplica formatação de valores:

```
Sub formataValor()
    Dim valor As Variant
    valor = Range("A2")

    Range("B2") = Format(valor, "#.###")
    Range("C2") = Format(valor, "#,##.##")
    Range("D2") = Format(valor, "R$ #,##.##")
End Sub
```

Resultado obtido:

	A	B	C	D
1	Valor	Formato decimal	Formato Separador	Formato Monetário
2	4585,65	458565,00%	4.585,65	R\$ 4.585,65

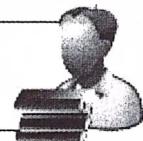
Abaixo temos uma tabela contendo alguns tipos de formatação comuns e algumas específicas:

Tipo de Formatação	Formato	Resultado
Casas decimais	"0.00"	1,45
Porcentagem	"0.00%"	2,26%
Separação da ordem de milhares	"#,##"	1.256
Dinheiro	"R\$ #,##.##"	R\$ 2.545.654,23
Data Abreviada	"dd/mm/yyyy"	12/05/2010
Data Completa	"dddd, dd 'de' mmmm 'de' yyyy"	Sábado, 18 de setembro de 2010
Hora	"Hh:mm:ss"	14:20:30
Telefone	"(00) 0000-0000"	(21) 2554-5621
Converter em caixa alta	">"	ANA PEREIRA
Converter em caixa baixa	"<"	ana pereira

Formatando Células

Formatar células pelo código VBA é tão simples quanto formatar da maneira usual no Excel, existe algumas propriedades que aplicadas a células modificam a aparência de seu conteúdo.





Abaixo é exibida uma tabela que contém algumas propriedades comuns de formatação de células:

Propriedade	Descrição	Valores aplicados	Exemplo
Font.Size	Altera o tamanho da fonte	Valores numéricos inteiros	Font.size = 12
Font.Name	Altera a aparência da fonte	"Arial", "Tahoma", "Verdana", etc.	Font.Name = "Arial"
Font.Bold	Aplica estilo de fonte negrito	True (Aplica) False (Retira)	Font.Bold = True
Font.Underline	Aplica estilo de fonte sublinhado	True (Aplica) False (Retira)	Font.Underline = True
Font.Italic	Aplica estilo de fonte itálico	True (Aplica) False (Retira)	Font.Italic = True
Font.Color	Altera a cor da fonte	(Será explicado adiante)	Font.Color = vbBlue
Interior.Color	Altera a cor de preenchimento da célula	(Será explicado adiante)	Interior.Color = vbRed

A propriedade "Color" apresenta valores constantes para cada tipo de cor:

Constante	Cor
vbBlack	Preto
vbRed	Vermelho
vbGreen	Verde
vbYellow	Amarelo

Constante	Cor
vbBlue	Azul
vbMagenta	Magenta
vbCyan	Ciano
vbWhite	Branco

Exemplo de código para alterar a aparência de uma planilha:

```
Sub formatarCelulas()
    Range("A1:C1").Interior.Color = vbYellow
    Range("A1:C1").Font.Color = vbRed
    Range("A1:C1").Font.Bold = True
    Range("A2:A6").Font.Italic = True
    Range("B2:B6").Font.Name = "Arial"
    Range("C2:C6").Font.Underline = True
End Sub
```

Resultado da formatação:

Antes			Depois		
	A	B	A	B	C
1	País	Vendas	Classificação	1	País
2	Uruguai	455662	1º Lugar	2	Uruguai
3	Brasil	200000	2º Lugar	3	Brasil
4	Bolívia	45566	3º Lugar	4	Bolivia
5	Argentina	30000	4º Lugar	5	Argentina
6	Paraguai	5000	5º Lugar	6	Paraguai

Outras alternativas para alterar a cor de células:

O VBA possui algumas alternativas de propriedades mais eficientes para alterar cores: usando a função "RGB" e a propriedade "ColorIndex".

- Função "RGB":** amplia o número de cores obtidas pela propriedade Color através da combinação de cores do padrão RGB (Vermelho, Verde e Azul).

A sintaxe da função RGB é: **RGB (<cor vermelha>,<cor verde>,<cor Azul>)**, onde, cada cor é representada por um intervalo de 0 a 255.

Exemplo de uso da função:

```
Range("A4").Font.Color = RGB(100,100,100) 'Cor cinza
```





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno



2. **Propriedade "ColorIndex":** Essa propriedade substitui a propriedade Color, ela representa um conjunto 56 cores padronizadas, sendo que cada cor possui um valor numérico referencial.

A sintaxe da propriedade ColorIndex é: `ColorIndex = <valor da cor>`.

Exemplo de uso da função:

```
Range("A4").Font.ColorIndex = 45
```

Dica: Embora não tenham sido demonstrados os valores referentes aos códigos de cores o próprio Help do VBA possui uma tabela referencial aos códigos de cores.

Aplicando Bordas em Células da Planilha

A aplicação de bordas em uma planilha tem a função de destacar o contorno da célula, usarmos para isso a propriedade **"Borders"** onde definimos o valor correspondente a borda a ser modificada.

Abaixo é exibida uma tabela onde destacamos os tipos de bordas que podemos alterar em uma célula:

Nome	Valor	Descrição
<code>xlDiagonalDown</code>	5	Borda vindo do canto superior esquerdo para o inferior direito de cada célula do intervalo.
<code>xlDiagonalUp</code>	6	Borda vindo do canto inferior esquerdo para o superior direito de cada célula do intervalo.
<code>xlEdgeLeft</code>	7	Borda na extremidade esquerda do intervalo.
<code>xlEdgeTop</code>	8	Borda na parte superior do intervalo.
<code>xlEdgeBottom</code>	9	Borda na parte inferior do intervalo.
<code>xlEdgeRight</code>	10	Borda na extremidade direita do intervalo.
<code>xlInsideVertical</code>	11	Bordas verticais de todas as células do intervalo exceto a parte externa do intervalo.
<code>xlInsideHorizontal</code>	12	Bordas horizontais de todas as células do intervalo exceto a parte externa do intervalo.

A essas bordas também atribuímos propriedades visuais:

Propriedade LineStyle: Altera o estilo da borda.

- `xlNone` (Sem Bordas);
- `xlContinuous` (Linha Continua);
- `xlDash` (Linha tracejada);
- `xlDashDotDot` (Traço seguido por dois pontos);
- `xlDashDot` (Linha pontilhada);
- `xlDouble` (Linha Dupla);
- `xlLantDashDot` (Traços inclinados).

Propriedade Weight: Alterar a espessura da borda.

- `xlThin` (Borda fina);
- `xlMedium` (Borda semi-fina);
- `xlThick` (Borda Grossa).

Propriedade Color ou ColorIndex: Alterar a cor da borda.





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

Exemplo de código abaixo:

```
Sub aplicarBorda()
    Dim bordasInternas, bordasExternas As Integer

    'Seleção de células a serem formatadas
    Range("A1:C6").Select

    'Formata as bordas externas
    For bordaExterna = 7 To 10
        Selection.Borders(bordaExterna).Weight = xlThick
        Selection.Borders(bordaExterna).LineStyle = xlContinuous
        Selection.Borders(bordaExterna).Color = vbBlue
    Next

    'Formata as bordas internas
    For bordaInterna = 11 To 12
        Selection.Borders(bordaInterna).Weight = xlThin
        Selection.Borders(bordaInterna).LineStyle = xlDot
        Selection.Borders(bordaInterna).Color = vbRed
    Next
End Sub
```

Resultado após a execução do código:

País	Vendas	Classificação
Uruguai	455662	1º Lugar
Brasil	200000	2º Lugar
Bolívia	45566	3º Lugar
Argentina	30000	4º Lugar
Paraguai	5000	5º Lugar

Excluindo Conteúdo das Células

Abaixo comandos destinados a realizar exclusão de conteúdo de células:

- **ClearContents:** Exclui somente o conteúdo da célula.

Exemplo de código:

```
Sub excluindoConteudo()
    Range("A1:C6").Select
    Selection.ClearContents
End Sub
```

- **ClearFormats:** Exclui a formatação da planilha.

Exemplo de código:

```
Sub excluindoFormatacao()
    Range("A1:C6").Select
    Selection.ClearFormats
End Sub
```

- **Clear:** Exclui todo conteúdo da planilha.





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do AlunoExemplo de código:

```
Sub exclusaoAtual()
    Range("A1:C6").Select
    Selection.Clear
End Sub
```

Excluindo e Inserindo Células na Planilha

Abaixo comandos destinados a realizar exclusão e inserção de células, linhas e colunas na planilha:

Inserindo Células na Planilha:

- **EntireColumn.Insert:** Inseri uma nova coluna na planilha.

Código de Exemplo:

```
Sub novaColuna()
    Range("A1").EntireColumn.Insert
End Sub
```

- **EntireRow.Insert:** Inseri uma nova linha na planilha.

Código de Exemplo:

```
Sub novaLinha()
    Range("A1").EntireRow.Insert
End Sub
```

- **Insert.xlToRight:** Inseri uma nova célula deslocando células próximas para a direita.

Código de exemplo:

```
Sub novaCelulaDireita()
    Range("D2:D8").Insert xlToRight
End Sub
```

- **Insert.xlToDown:** Inseri uma nova célula deslocando células próximas para a baixo.

Código de exemplo:

```
Sub novaCelulaAbaixo()
    Range("A3:D3").Insert xlDown
End Sub
```

Excluindo Células na Planilha:

- **EntireColumn.Delete:** Exclui coluna da planilha.

Código de Exemplo:

```
Sub excluiColuna()
    Range("A1").EntireColumn.Delete
End Sub
```



**VBA**

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

- **EntireRow.Delete:** Exclui linha da planilha.

Código de Exemplo:

```
Sub excluiLinha()
    Range("A1").EntireRow.Delete
End Sub
```

- **Delete.xlToLeft:** Apaga uma célula deslocando células próximas para a esquerda.

Código de exemplo:

```
Sub excluiCelulaEsquerda()
    Range("D2:D8").Delete xlToLeft
End Sub
```

- **Delete.xlToUp:** Apaga uma célula deslocando células próximas para a cima.

Código de exemplo:

```
Sub excluiCelulaAcima()
    Range("A3:D3").Delete xlUp
End Sub
```

Alinhando Conteúdo em Células

Abaixo comandos destinados a realizar alinhamento do conteúdo nas células:

Alinhamento Horizontal:

Esse alinhamento é utilizado para o alinhamento do conteúdo em relação à largura da célula. Abaixo é apresentado a propriedade "HorizontalAlignment" e seus valores de alinhamento:

- **HorizontalAlignment.xlRight:** Aplica alinhamento horizontal a direita.
- **HorizontalAlignment.xlLeft:** Aplica alinhamento horizontal a esquerda.
- **HorizontalAlignment.xlCenter:** Aplica alinhamento horizontal ao centro.
- **HorizontalAlignment.xlJustify:** Aplica alinhamento horizontal justificado (por igual).

Exemplo de código:

```
Sub aplicarAlinhamentoHorizontal()
    Range("A1:C6").HorizontalAlignment = xlCenter
End Sub
```

Resultado Esperado:

	A	B	C
1	País	Vendas	Classificação
2	Uruguai	455662	1º Lugar
3	Brasil	200000	2º Lugar
4	Bolívia	45566	3º Lugar
5	Argentina	30000	4º Lugar
6	Paraguai	5000	5º Lugar





Alinhamento Vertical:

Esse alinhamento é utilizado para o alinhamento do conteúdo em relação à altura da célula. Abaixo é apresentado a propriedade "VerticalAlignment" e seus valores de alinhamento:

- **VerticalAlignment.xlTop**: Aplica alinhamento vertical a cima.
- **VerticalAlignment.xlCenter**: Aplica alinhamento vertical ao centro.
- **VerticalAlignment.xlBottom**: Aplica alinhamento vertical a baixo.

Exemplo de código:

```
Sub aplicarAlinhamentoVertical()
    Range("A1:C6").VerticalAlignment = xlCenter
End Sub
```

Resultado Esperado:

	A	B	C
1	País	Vendas	Classificação
2	Uruguai	455662	1º Lugar
3	Brasil	200000	2º Lugar
4	Bolívia	45566	3º Lugar
5	Argentina	30000	4º Lugar
6	Paraguai	5000	5º Lugar

Ajustando Espaçamento de Células

Abaixo serão abordadas propriedades relacionadas ao ajuste do espaçamento vertical e horizontal da célula:

- **ColumnWidth = <valor do tamanho>**: Controla a largura da célula a partir de valores fornecidos a propriedade.
- **RowHeight = <valor do tamanho>**: Controla a altura da célula a partir de valores fornecidos a propriedade.

Exemplo de código:

```
Sub alterandoLarguraAltura()
    Range("A1:C6").ColumnWidth = 30
    Range("A1:C6").RowHeight = 30
End Sub
```

Existem também propriedades de auto-ajuste que podem ser aplicadas:

- **entireColumn.AutoFit**: Aplica um ajuste automática a largura da célula.
- **entireRow.Autofit**: Aplica um ajuste automático a altura da célula.

Exemplo de código:

```
Sub aplicandoAjusteAutomático()
    Range("A1:C6").EntireColumn.AutoFit
    Range("A1:C6").EntireRow.AutoFit
End Sub
```





Mesclando Células

A mesclagem de células tem a função de unir diversas células em uma única célula principal, a mesclagem (união de células) é realizada de maneira simples através das propriedades abaixo.

- **Merge:** Mescla um conjunto de células em uma única célula.
- **UnMerge:** Desfaz a mesclagem aplicada.

Exemplo de código:

```
Sub mesclarCelulas()
    Range("A1:C1").Merge
End Sub
```

Resultado esperado:

	A	B	C
1		Levantamento de Vendas	
2	País	Vendas	Classificação
3	Uruguai	455662	1º Lugar
4	Brasil	200000	2º Lugar
5	Bolívia	45566	3º Lugar
6	Argentina	30000	4º Lugar
7	Paraguai	5000	5º Lugar

Usando Conjuntos

Vimos até o momento diversos comandos de formatação para ser aplicado pelo VBA, aplicar um ou dois comandos é fácil, agora aplicar diversos tipos de formatação ao mesmo tempo de torna complicado.

Através do comando With, podemos simplificar a construção de comandos associados ao mesmo fim. Abaixo um exemplo sem o uso do comando With e usando o comando With.

1º Exemplo - (sem utilizar conjunto With):

```
Private Sub CommandButton1_Click()
    Range("B4:D4").Select
    Selection.Merge
    Selection.Font.Color = vbRed
    Selection.Font.Size = 14
    Selection.HorizontalAlignment = xlCenter

    Range("E4:H4").Select
    Selection.Merge
    Selection.Font.Color = vbBlue
    Selection.Font.Size = 14
    Selection.HorizontalAlignment = xlCenter

    Range("B5:H5").Select
    Selection.Font.Bold = True
    Selection.HorizontalAlignment = xlCenter

    Range("B6:H12").Select
    Selection.HorizontalAlignment = xlCenter
    Selection.Font.ColorIndex = 55
End Sub
```

2º Exemplo - (Utilizar conjunto With):

```
Private Sub CommandButton1_Click()
    Range("B4:D4").Select
    With Selection
        .Merge
        .Font.Color = vbRed
        .Font.Size = 14
        .HorizontalAlignment = xlCenter
    End With

    Range("E4:H4").Select
    With Selection
        .Merge
        .Font.Color = vbBlue
        .Font.Size = 14
        .HorizontalAlignment = xlCenter
    End With

    Range("B5:H5").Select
    With Selection
        .Font.Bold = True
        .HorizontalAlignment = xlCenter
    End With

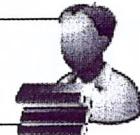
    Range("B6:H12").Select
    With Selection
        .HorizontalAlignment = xlCenter
        .Font.ColorIndex = 55
    End With
End Sub
```





VBA

MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno

CAPÍTULO 7 – SUBROTINAS E FUNÇÕES

O assunto que abordaremos nesse capítulo diz respeito ao conceito de organização de códigos que se destaca em dois tipos: **Programação Estruturada** e **Programação Orientada a Objetos**.

A maior parte do código dessa apostila utiliza **programação estruturada**, já a **programação orientada a objetos** se destaca pela ideia de aproveitamento de código onde podemos criar códigos mais eficientes e funcionais.

Começaremos a estudar a diferença entre **sub-rotinas** e **funções**, passando por **passagem de parâmetros e valores** e abordando herança de funções e sub-rotinas.

Construindo Sub-rotinas

Todo código construído no VBA utiliza sub-rotinas, uma sub-rotina apresenta um **aspecto modular** onde pode ser utilizada sozinha ou em conjunto com outras sub-rotinas.

Criando Sub-rotinas:

Sem você perceber, estamos construindo sub-rotinas o tempo inteiro, abaixo um exemplo de uma sub-rotina:

```
Sub subrotina()
    With Range("A1:C1")
        .Font.Color = vbRed
        .Merge
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlCenter
    End With

    For linha = 2 To 7
        Cells(linha, 2) = Format(Cells(linha, 2), "R$ #,###.00")
    Next
    Range("A1:C7").EntireColumn.AutoFit
End Sub
```

A sub-rotina apresentada um conjunto de comandos definidos a realizar uma determinada ação.

Chamando Sub-Rotinas:

Uma sub-rotina isolada como a apresentada já é possível obter um resultado, observe o código abaixo

```
Sub subrotina()
    'Altera a cor de fundo das células selecionadas
    Range("A1").CurrentRegion.Select
    Selection.Interior.Color = vbYellow

    'Aplica bordas a planilha
    Dim i As Byte
    For i = 7 To 12
        With Selection.Borders(i)
            .Weight = xlThin
            .LineStyle = xlContinuous
            .Color = vbRed
        End With
    Next

    'Formata as células da planilha
    With Range("A1:C1")
        .Font.Color = vbRed
        .Merge
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlCenter
    End With

    For linha = 2 To 7
        Cells(linha, 2) = Format(Cells(linha, 2), "R$ #,###.00")
    Next
    Range("A1:C7").EntireColumn.AutoFit
End Sub
```

