Aula 1 – O modelo de objetos do Excel
APPLICATION – É o próprio Excel. Temos diversas propriedades e métodos importantes nesse objeto. Destacamos dois exemplos:
Application.DisplayAlerts – Se for true, o Excel avisará que você vai perder os dados ao excluir uma planilha da pasta, ou se quiser sair sem salvar as últimas operações, por exemplo. Se for false, não haverá aviso nenhum.
Application.ActiveWorkBook – Se vários arquivos estiverem em uso, essa propriedade diz qual é o que está ativo no momento. Se for alterado, um outro arquivo estará aberto. Fora do VBA, esta propriedade é alterada no menu "janela" quando trocamos de planilha.

Este objeto contém uma coleção chamada **WorkBooks** que representa os diversos arquivos Excel abertos.

Esta coleção é o conjunto de todos os objetos do tipo **WorkBook** abertos no momento.
Um arquivo Excel pode ser criado através do método Add da coleção Workbooks.
Assim, emitir o comando WorkBooks.Add do VBA equivale a fazer Arquivo – Novo no menu do Excel.

Abaixo do Workbook temos a coleção **Sheets**, que são objetos do tipo **WorkSheet**
Você já deve ter imaginado como criar uma planilha nova: Sheets.Add – Insere uma planilha no Workbook (arquivo) ativo.
Para criar uma planilha nova em um Arquivo que esteja aberto mas não seja o ativo, use:
Workbooks(2).Sheets.Add ou Workbooks("xpto.xls").sheets.add

IMPORTANTE: Apesar do VBA permitir que se omita o WorkBooks(x). antes do Sheets, é uma boa norma que sempre se use explicitamente de que arquivo a planilha data pertence. Se o usuário abrir um outro arquivo Excel durante a execução da macro, isso evita que a macro pegue ou altere dados deste arquivo que acabou de ser aberto.

Abaixo do Objeto WorkSheet temos a coleção Cells, que representa todas as células de uma planilha.

Esta coleção funciona como uma matriz com linhas e colunas.
Assim , a célula D3 (linha 3 da coluna 4) pode ser referenciada como cells(3,4)

Um outro objeto que usamos muito é o objeto Range, que serve para mapear uma célula ou um retângulo de células. Para informar o endereço das células envolvidas, usa-se a mesma sintaxe que usamos nas fórmulas do Excel. (Excel: =soma("a1:b5") VBA : range("a1:b5")

Um objeto range pode ser indexado por linha e coluna.
Exemplo: range("b2")(3,4) é o mesmo que cells(4,5). A coleção Cells é um objeto do tipo Range.

Essas são as informações básicas sobre o modelo de objetos do Excel.
Vamos aprender muitas outras coisas a partir de exemplos de sub-rotinas e funções.

Programa 1

Vamos fazer sub-rotinas para listar todas as pastas abertas, com suas planilhas e células.

Para inicializar o ambiente de programação (projeto VBA) vamos solicitar a gravação de uma macro e mandar parar em seguida. O ambiente estará criado com uma macro chamada macro1 (ou outro nome que você tenha informado) sem código nenhum.

Vamos alterar o nome dela para ListaArquivos
O conteúdo será:

```
1. Sub ListaArquivos()
2.         For Each x In Workbooks
3.                 Debug.Print "Arquivo: " & x.Name
4.         Next
5. End Sub
```

Explicação:

LINHA 1 Sub é uma palavra reservada do VB que marca o início de uma sub-rotina.
ListaArquivos é o nome da sub-rotina – Nós podemos dar o nome que quisermos.

LINHA 2 For Each é um comando VB que significa "para cada" e x é uma variável (que está sendo criada nesse momento) que representará cada elemento na coleção dada a seguir.
Workbooks é o nome da coleção inspecionada.
Então o comando 2 significa: "para cada arquivo aberto, que representaremos aqui por x"

LINHA 3 O Comando Debug.Print lista o que quisermos numa janela chamada de "Verificação Imediata". Se esta janela não estiver aberta no ambiente do VB, tecla-se CTRL+G
Esse comando recebe uma única variável, constante ou expressão. No caso, ele está recebendo uma expressão, que é o resultado da concatenação (operador &) de uma constante: "Arquivo: " com uma variável, no caso a propriedade Name (nome) do objeto x, ou seja, o nome de cada pasta Excel aberta.

LINHA 4 Next é um comando VB que serve para delimitar o FOR. O que está entre o FOR e o NEXT será executado uma vez para cada planilha aberta.

LINHA 5 End Sub é um comando do VB que é usado para delimitar o código da sub-rotina, e também para retornar a quem a chamou . No caso desta sub-rotina ele será chamada diretamente pelo usuário, e não por outra rotina qualquer. Então, quando a sub-rotina executar esse comando, cessará imediatamente a atividade do programa VBA.

Para executar essa rotina, coloca-se o cursor em qualquer linha de 1 a 5 e tecla-se F5
Ou então, no ambiente do Excel, tecla-se ALT+F8 e dá-se um clique duplo na rotina ListaArquivos.
Se ela já estiver em destaque acima da lista (deve ser o caso nesse instante) basta teclar ENTER.
Então, para executar a primeira rotina em ordem alfabética, basta teclar no Excel: Alt+F8 seguido de ENTER.

Vamos melhorar o programa, criando uma sub-rotina que liste todas as planilhas de um WorkBook.
Esta sub-rotina precisa saber qual é o WorkBook que queremos inspecionar.
No cabeçalho da sub-rotina vamos colocar entre os parêntesis um nome de variável que significará o Workbook que queremos trabalhar
O fonte da rotina será:

```
1. Sub ListaPlanilhas(W)
2.     For Each x In W.Sheets
3.         debug.Print "   Planilha: " & x.Name
4.     Next
5. End Sub
```

Esse programa, tirando o fato de que recebemos um objeto WorkBook como planilha como parâmetro (chamamos esse WorkBook de W, mas podia ser qualquer nome), é idêntico ao anterior.
Aqui temos que representar o pai de sheets, que é W (W.sheets) já que sabemos que existirão vários Workbooks e se não colocarmos o W explicitamente, será assumido o WorkBook que estiver ativado.
No exemplo anterior não precisamos colocar o pai do WorkBooks no for each, porque o pai é o objeto Application, e só existe um ativo a cada momento. Mas podíamos escrever Application.Workbooks.

Agora vamos incrementar a primeira rotina para chamar a segunda, a cada workbook obtido no for:

```
1.  Sub ListaArquivos()
2.      For Each x In Workbooks
3.          Debug.Print "Arquivo: " & x.Name
4.          ListaPlanilhas x
5.      Next
6.  End Sub
```

A novidade é a linha 4. Para chamar uma sub-rotina escreve-se o nome da mesma, seguida da lista de parâmetros separados por vírgula. Como só existe um parâmetro não se colocam vírgulas.

Finalmente, vamos fazer o programa completo, aonde o ListaPlanilhas chama o ListaCelulas, para listar as células não vazias de uma planilha.
Segue-se o fonte e depois os comentários:

```
1.  Sub ListaArquivos()
2.      For Each x In Workbooks
3.          Debug.Print "Arquivo: " & x.Name
4.          ListaPlanilhas x
5.      Next
6.  End Sub
7.  Sub ListaPlanilhas(p)
8.      For Each x In p.Sheets
9.          Debug.Print "   Planilha: " & x.Name
10.         ListaCelulas x
11.     Next
12. End Sub
13. Sub ListaCelulas(p)
14.     Set ultimacelula = p.Cells(1, 1).SpecialCells(xlLastCell)
15.     For lin = 1 To ultimacelula.Row
16.         For col = 1 To ultimacelula.Column
17.             If  p.Cells(lin, col) <> "" Then
18.                 Debug.Print "       (" & lin & "," & col & ") = " & p.Cells(lin, col)
19.             End If
20.         Next
21.     Next
22. End Sub
```

A linha 14 tem uma novidade, que é o comando SET. Ele serve para indicar ao comando de atribuição que deve-se atribuir o objeto à variável à esquerda do igual, e não o atributo padrão do objeto
Como o comando p.Cells(1, 1).SpecialCells(xlLastCell) é uma função que retorna um objeto do tipo CELL, devemos utilizar o SET para dizer que:

a) ultimacélula será um objeto
b) ela receberá a última célula preenchida da planilha, e não o seu conteúdo.

Este método SpecialCells devolve uma célula dependendo do parâmetro passado. Nesse caso, o parâmetro indica que queremos a célula mais abaixo e mais à direita da parte utilizada da planilha.
Para obter esse método, gravei uma macro aonde só digitei CTRL+END, para posicionar nessa célula. Parei a macro e verifiquei o que foi gravado. 90% do conhecimento sobre o VBA do Excel é obtido assim, gravando uma macro e analisando o que é gravado.
Desse objeto obtido na linha 14, extraímos nas linhas 15 e 16 respectivamente o número da linha (Row) e o número da coluna (Column) dele. Esta célula pode não ter sido nunca utilizada, mas alguma célula na linha obtida em 15 e alguma célula da coluna obtida em 16 foram usadas um dia (mesmo que já tenham sido apagadas).
O comando For aqui usa o seu formato tradicional (o For each é usado apenas em um ambiente orientado a objetos, e apenas para um objeto do tipo coleção).
Nesse formato dizemos que a variável de controle (lin ou col) deve variar a cada iteração, sendo que na primeira vez vai valer 1 (é o primeiro valor após o =) até o valor constante ou variável após a palavra

reservada TO. Ainda existe mais um operando (STEP) nesse comando para informar qual é a variação do valor entre cada iteração. Se não informado, a variável de controle é incrementada de 1 a cada vez.

Repare que na linha 17 utilizamos p.Cells pra indicar que estamos trabalhando com as células da planilha que recebemos como parâmetro e chamamos de p.
Se usarmos apenas Cells estaremos nos referenciando às células da planilha ativa, também conhecida como ActiveSheet.

Resumo dos conhecimentos adquiridos nesta aula até aqui.
Objetos e Coleções
Application; Workbooks, Workbook, ActiveWorkBook, Sheets, Sheet, Activesheet, Cells, Activecell
Atributos:
Name, Row, Column
Métodos: Add

INTERAÇÂO COM O MEIO EXTERNO

Nesta aula vamos aprender a exportar informações do Excel para um arquivo texto.
Também veremos como chamar um aplicativo de dentro do VBA, usando como exemplo o bloco de notas

Alteraremos as rotinas vistas até aqui para em vez de listar as informações na janela de verificação imediata escrever em um arquivo texto.

```
1.  Sub ImprimeArquivos()
2.      nomearq = ActiveWorkBook. Path & "\listagem.txt"
3.      Open nomearq For Output As #1
4.      For Each x In Workbooks
5.          Print #1, "Arquivo: " & x.Name
6.          ImprimePlanilhas x
7.      Next
8.      Close #1
9.      Shell "notepad """ & nomearq & """", vbMaximizedFocus
10. End Sub
11. Sub ImprimePlanilhas(p)
12.     For Each x In p.Sheets
13.         Print #1, "   Planilha: " & x.Name
14.         ImprimeCelulas x
15.     Next
16. End Sub
17. Sub ImprimeCelulas(p)
18.     Set ultimacelula = p.Cells(1, 1).SpecialCells(xlLastCell)
19.     For lin = 1 To ultimacelula.Row
20.         For col = 1 To ultimacelula.Column
21.             If p.Cells(lin, col) <> "" Then
22.                 Print #1, "        (" & lin & "," & col & ") = " & p.Cells(lin, col)
23.             End If
24.         Next
25.     Next
26. End Sub
```

Na linha 2 estamos vendo mais um atributo do objeto WorkBook: o endereço em disco onde ele se encontra. (.Path). Acrescentamos o nome de um arquivo, que não precisa existir, pois vamos criá-lo no comando da linha 3.
Na linha 3 vemos o comando para abrir arquivos. O formato é o dado acima e chamamos o #1 de "canal 1"
Como queremos criar o arquivo (ou sobrescrevê-lo, se já existir) usamos o comando listado, que significa: abra o arquivo "caminho\nome" para saída no canal 1
Nas linhas 5, 13 e 22 trocamos debug.print por print #1,

Ao final da listagem vamos fechar o arquivo, com o comando CLOSE (feche)
Só falta agora chamar o bloco de notas (notepad.exe) para abrir e mostrar o arquivo criado.
Usamos para isso o comando do VB Shell que recebe como parâmetro a linha de comando contendo o nome do programa e seus argumentos. O argumento (opcional) do notepad.exe é o nome do arquivo a ser aberto, mas não pode conter espaços em branco, exceto se estiver entre aspas. Como o caminho da planilha pode conter brancos (por exemplo: Meus Documentos) colocamos preventivamente o argumento entre aspas.
No VB, bem como na maioria das linguagens de programação, quando queremos ter uma constante contendo aspas duplas, colocamos duas aspas duplas para significar uma apenas.
Por isso temos 3 aspas duplas depois de notepad, na linha 9. As duas primeiras serão trocadas por apenas uma, e a terceira é a que fecha a aspa aberta logo após o comando Shell.

Muito bem.

Agora vamos executar o programa pela última vez.
Mas antes, abra algumas pastas Excel, para podermos documentá-las.

As sub-rotinas desta aula estão no endereço www.ruimedeiros.eti.br/vba/aula1.txt.
Ao abrir, recorte e cole no ambiente criado do VBA, ou num bloco de notas, salvando para uso futuro.
Se for salvar, use a extensão .bas em vez de .txt como é o habitual. Depois veremos porque isso.

Exercícios:

Usando, além do que vimos aqui, o gravador de macros, faça:

Uma sub-rotina para fechar um arquivo Excel, sem receber uma mensagem de alerta

Uma sub-rotina para excluir uma planilha de um arquivo Excel, sem mensagem de alerta.

Uma sub-rotina para criar um arquivo Excel novo, com as 3 planilhas com nomes: "ABC", CDE", EFG"

Uma sub-rotina para colocar a largura da coluna "D" em 20

Uma sub-rotina para alterar a largura da coluna da célula selecionada. O valor da largura está na célula selecionada.

Uma sub-rotina para colocar em todas as células de um retângulo selecionado o valor que está na célula na primeira linha e na primeira coluna dese retângulo.