Aula 4 – Miscelânea

Nesta última aula vamos ver alguns tópicos especiais.
Inicialmente, vamos colocar no início de nosso módulo duas declarações:
- Option Explicit
Acredito que nas três primeiras aulas todos cometeram erros de digitação em nomes de variáveis, o que causou erro em tempo de execução. Algumas vezes esse tipo de erro causa grandes problemas e é difícil de ser detectado.
Para evitá-lo, usa-se a declaração Option Explicit no início de cada módulo criado. Ela faz com que o uso de variáveis que não sejam explicitamente declaradas (por exemplo com o comando DIM) causem um erro de compilação na entrada da rotina que a usa.
Apesar desta declaração ser opcional, é uma boa prática usá-la sempre.

- Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
Esta declaração é para uma sub-rotina, mas poderia ser para uma função. Ela serve para fazer uma ponte entre o seu programa e rotinas pré-fabricadas, distribuídas com o sistema operacional, produtos de terceiros ou mesmo de sua equipe. Estas rotinas e funções estão em uma DLL, e são carregadas na memória em tempo de execução. Esta rotina Sleep, como o próprio nome indica, serve para deixar o seu programa em estado de espera (wait) por um tempo determinado, fornecido em milissegundos.
Por exemplo, dentro do programa escrevemos Sleep 1000 para deixar o programa parado por 1 segundo.

Agora vamos estudar alguns métodos e propriedades de um objeto que é utilizado extensamente, inclusive em aplicações JAVA.
Este objeto é chamado de FileSistemObject (Objeto Sistema de Arquivos) e serve para informar ou executar qualquer função que pode ser feita pelo Windows Explorer, como criar ou excluir pastas e arquivos, listar atributos de arquivos, alterá-los, renomear arquivos e pastas e muito mais.
Solicitei permissão à editora NovaTec para distribuir um guia de referência rápida em PDF que está esgotado e não será reeditado, contendo entre outras informações super úteis a descrição completa do FileSystemObject. Esse guia está em http://www.novatec.com.br/ebooks .(Windows Script Host)

Como aplicação exemplo, vamos criar uma versão muito simples do Windows Explorer, permitindo a listagem de todos os arquivos e pastas de qualquer drive de seu computador, ou mapeados da rede.

Definição da macro:
Numa planilha do Excel, coloca-se um caminho válido em uma célula, seleciona-se ela e chama-se a macro.
Ao entrar, a macro sabe qual é o diretório a analisar (está em ActiveCell) e dará a resposta da seguinte maneira:
Na Célula A1 vai ser posto o string '..' que representa o diretório pai do que foi passado para a macro.
Na Célula A2 vai ser colocado o diretório passado, e nas linha subsequentes serão listados todos os subdiretórios do informado e ao final, serão listados os arquivos residentes nesse diretório, com tamanho e data do último acesso.

Programa:

```
1.  Option Explicit
2.  Sub Explorar()
3.  Dim diretorio, fso, i As Integer, ex, d, f
4.  Set fso = CreateObject("Scripting.FileSystemObject")
5.  If ActiveCell(1, 2) > "" Then ' e um arquivo
6.     Set ex = CreateObject("Wscript.Shell")
7.     ex.Run """" & Cells(2, 1) & ActiveCell & """"
8.  Exit Sub
9.  End If
10. If ActiveCell.Text = ".." Then
11.    Set diretorio = fso.getfolder(ActiveCell(2, 1).Text & "\..")
12. Else
13.    Set diretorio = fso.getfolder(ActiveCell.Text)
14. End If
15. Cells.ClearContents
16. Cells(1, 1) = ".."
17. Cells(2, 1) = Replace(diretorio.Path & "\", "\\", "\")
18. i = 3
19. For Each d In diretorio.subfolders
20.    Cells(i, 1) = d.Path
21.    i = i + 1
22. Next
23. For Each f In diretorio.Files
24.    Cells(i, 1) = f.Name
25.    Cells(i, 2) = Format(f.datelastmodified, "dd/mm/yyyy hh:mm")
26.    Cells(i, 3) = f.Size
27.    Cells(i, 3).NumberFormat = "_(* #,##0_);_(* (#,##0);_(* ""-""??_);_(@_)"
28.    i = i + 1
29. Next
30. End Sub
```

Análise do código

1 – Informa que é obrigatório declarar todas as variáveis

2 – Nome da rotina (Explorar)

3 – Declara as variáveis que serão utilizadas. Todas são variant, exceto a "i" que é um inteiro

4 – Cria-se a variável objeto chamado fso, que representa o objeto File System

5 – Se existem dados na coluna 2 é porque é um arquivo. Nesse caso, vamos abri-lo.

6 - Para abrir o arquivo, temos que "instanciar" o objeto Wscript.Shell . A variável que o representa é : ex

7 – Executamos o método Run (rodar=executar= abrir) desse objeto passando o caminho completo do arquivo: seu diretório, que está em cells(2,1) uma \ e o nome do arquivo. Como pode existir algum espaço em branco nesse nome ou no caminho, deve-se colocar entre aspas duplas esse parâmetro.

8 - Terminamos a execução da rotina. O arquivo continua aberto (pelo notepad, word, etc)

9 – Fim do teste da linha 5

10 – Testa se o cursor está solicitando o diretório pai. Se for, representamos o diretório pai usando a variável objeto diretório (observe que trata-se de um set e não de uma atribuição normal)

11 e 13 – O método getfolder do File System recebe um nome de diretório e devolve um objeto Folder

15 – Os dados da planilha são apagados

16 – Coloca-se na linha 1 o indicativo de diretório pai

17 – Coloca-se o nome completo do diretório escolhido na linha 2. Se o diretório escolhido termina com \ a concatenação vai gerar uma contra barra dupla: \\ Para corrigir este eventual erro, troca-se \\ por \ (se não existir este string, nada será trocado nem será dada uma mensagem de erro)

18 – A variável i indica qual é a próxima linha da planilha livre para receber dados

19 – para cada subdiretório do diretório escolhido, usamos a variável objeto d para representá-lo

20 – Colocamos o endereço do diretório d na próxima linha livre (começa com 3)

21 – Incrementa-se o número da linha livre

22 – Fim do for

23 a 29 – Ao final da apresentação de todos os subdiretórios, passamos a mostrar todos os arquivos, só que agora devemos informar na coluna 3 o tamanho e na coluna 2 a data e hora da ultima atualização. A função Format foi apresentada na aula 2. Na linha 27 usamos a propriedade NumberFormat para dizer que o formato dos dados da célula que contém o tamanho do arquivo (coluna 3). O formato é de números

inteiros com .separador de milhares. Como eu sei este formato complicado? Não sei, gravei uma macro selecionando esse formato, parei de gravar, editei, recortei o código e colei aqui. Como já disse, a maior parte do código é construída assim, com o gravador de macros.
30 – Fim da Rotina

Execute a macro em sua planilha

Vamos agora mudar radicalmente o rumo do nosso aprendizado, já que hoje é uma miscelânea. Aprenderemos a trabalhar com elementos gráficos fazendo uma rotina bastante útil.
Você já deve ter instalado softwares em sua máquina, que demoram um pouco, e ficam mostrando uma barra que vai aumentando até chegar no máximo. Quando chega, a tarefa acaba. Dá-se o nome de barra de progresso à esse controle. Pois bem, vamos fazer uma barra de progresso diferente: Ela começa cheia e vai diminuindo até acabar, e acabará junto com a sua tarefa.
Para poder utilizá-la, você tem que saber a priori o numero de passos que a sua tarefa terá de dar, e a cada passo você informa à rotina que faremos quantos passos foram dados até agora e qual é o número máximo.
Suponha que teremos 100 passos (por exemplo, processar dados de 100 funcionários.)
Inicialmente chamamos com (0,100) – Nenhum passo dado de um total de 100).
A rotina nesse momento vai criar uma barra (retângulo) no topo da planilha. A cada passo chamamos com (n,100) e a rotina faz a barra ficar com tamanho = Tamanho inicial * (100 – n) / 100, de tal modo que No primeiro passo dado ela ficará com 99% do tamanho, no passo numero 50 ficará com 50% e no último passo, ficará com 0% do tamanho. A rotina então elimina esta figura da planilha.
Para poder fazer, precisamos gravar uma macro enquanto criamos um retângulo.
Infelizmente, a macro não grava com parâmetros nomeados. Por isso, deve-se usar o help do VBA para saber o que é cada valor.
Aqui eu vou explicar o significado de cada parâmetro, para tornar a aula mais produtiva, mas aconselho a pesquisar depois no help para aprender a aprender sozinho. No futuro este aprendizado será o mais importante, e você será capaz de aprender o VBA do Word e do Power Point sozinho.

Fonte da rotina:

```
1.  Sub progresso(atual, max)
2.  Static barra
3.  If atual = 0 Then
4.     Set barra = ActiveSheet.Shapes.AddShape(msoShapeRectangle, 0, 0, 300, 11.25)
5.     barra.Fill.ForeColor.SchemeColor = 11
6.     barra.Fill.Visible = msoTrue
7.     barra.Fill.Solid
8.     exit sub
9.  End If
10. If atual >= max Then
11.    barra.Delete
12.    Exit Sub
13. End If
14. barra.Width = 300 * (max - atual) / max
15. End Sub
```

Analise da rotina:
2 - Criamos uma variável para guardar o objeto barra (retângulo) que deve permanecer entre chamadas da rotina, ou seja, o objeto barra não será destruído quando a rotina acabar. Independente de o objeto ser destruído ou não, o retângulo permanecerá desenhado na planilha. Só que não conseguiremos trabalhar com ele se o objeto for destruído, nem mesmo apagá-lo.
3 – Quando chamamos com passo atual = 0, devemos desenhar um retângulo, através do método AddShape da coleção Shapes da planilha ativa. Se você é um bom observador, vai perguntar porque AddShapes? Normalmente as coleções tem um método padrão chamado Add (Sheets.Add, QueryTables.Add, etc). Acontece que a coleção Shapes guarda um grande número de objetos de tipos diferentes, como Fotos (Picture), Curvas, Caixas de Texto, etc. O "principal" objeto é o objeto Shape que pode representar uma gama enorme de desenhos, que se encontrar na Barra de Desenhos do Office (pelo menos até o 2003, já que no 2007 as barras de ferramentas mudaram radicalmente) . O formato desejado é informado como primeiro parâmetro do AddShape. No caso estamos criando um retângulo, e a constante msoShapeRetangle (mso de microsoft office) que vale 1, indica que vamos criar um retângulo

Ao parâmetros seguinte dão os valores dos atributos (nessa ordem): Left, Top, Width e Height (achei isso no help do AddShape)
Ao criar o retângulo eu estava com o gravador de macros ligado e pedi para colorir a barra de verde. As instruções 5 a 7 foram gravadas automaticamente, e eu simplesmente mantive com pequenas alterações. Se você gravar a macro para fazer isso, vai ver que ela não cria uma variável para guardar o objeto retângulo, mas simplesmente manda selecioná-lo e depois usa o objeto Selection para atribuir cor, etc. Como eu atribui o retângulo a uma variável, eu a uso no lugar do objeto Selection.
Em seguida vamos ver o que fazer quando o atual é igual ou maior que o máximo: Simplesmente deletar o objeto, o que só é possível porque criamos um objeto (barra) para guardar o retângulo, e evitamos a sua perda criando-o como estático. É claro que eu poderia definir o objeto barra como global no módulo, mas sempre que possível deve-se evitar esta prática. Como apenas esse módulo vai utilizar o objeto, criamos internamente a ela. Se outra Sub usar uma variável chamada barra, certamente não vai representar esse retângulo.

Em qualquer dos casos acima vistos, após a criação ou a destruição do retângulo a Sub termina sua execução.
Se não for esse o caso, devemos apenas redimensionar o retângulo, alterando o seu atributo Width (comprimento), conforme a fórmula já analisada, através da instrução da linha 14

Para testar a barra de progresso, vamos utilizar dois recursos ainda não vistos:
Precisamos "dar um tempo" entre uma chamada e outra, e para isso vamos chamar a rotina Sleep da API do Windows, conforme declaração no topo do módulo.
Outro ponto importante é que o Excel não altera nenhuma característica visível enquanto está executando uma função de uso exclusivo de CPU. Assim, se simplesmente a ficamos alterando o tamanho do retângulo, não conseguimos ver essa alteração até que a rotina termine, e aí o retângulo já foi destruído. Ou seja, o desenho animado não se move.
Quando emitimos o comando Sleep paramos de acionar a CPU, e outras tarefas sendo executadas pelo Windows podem funcionar, mas a interface não se alterará no Excel porque ele estará dormindo!
Por isso, executamos um comando que o programador VBA deve sempre usar dentro de loops controlados (For , do-loop ou While - wend). É a instrução DoEvents, que permite que todos os eventos pendentes sejam executados (um clique do mouse, uma tecla digitada, etc). Assim, nesse momento, a interface é atualizada, permitindo que se veja a barra de progresso diminuindo até acabar.
A sub para testar a barra de progresso tem o seguinte código:

```
Sub testabarra()
   Dim i As Integer
   For i = 0 To 500
      progresso i, 500
      DoEvents   ' mostra agora o efeito
      Sleep 10  ' duração total: 5 segundos
   Next
End Sub
```

Não há o que comentar aqui.

Agora que temos uma barra de progresso funcionando corretamente, vamos aprender a trabalhar com outras formas.
Vamos analisar uma macro que cria uma estrela de 32 pontas, roda ela e diminui o seu tamanho, de tal forma que após 500 iterações ela fica cm tamanho 0 e desaparece.
A cada iteração vamos chamar a barra de progresso, que acabará juntamente com a estrela após 500 iterações.
Nesse exemplo vamos aprender quais são as principais propriedades de qualquer forma: Sua posição, que é dada pelas propriedades LEFT e TOP, seu tamanho que é dado pelas propriedades Width e Height.
Alem disso, vamos conhecer o método IncrementRotation para girar um desenho em um determinado número de graus. Se positivo, gira no sentido horário. Se negativo gira no sentido anti-horário.
Se o gravador de macro do seu Excel 2007 não registrar as operações quando você trabalhar com formas, grave em uma versão anterior do Office e traga a macro já gravada para o seu projeto.

Vamos ao fonte desta aplicação:

```
Sub RodaEstrela()
   Dim estrela, i As Integer
   i = msoShape32pointStar
   Set estrela = ActiveSheet.Shapes.AddShape(i, 32.25, 45.75, 100#, 100#)
   For i = 0 To 300
     estrela.IncrementRotation 5
     estrela.Width = 100 - i / 3
     estrela.Height = 100 - i / 3
     progresso i, 300
     Sleep 10
     DoEvents
   Next
   estrela.Delete
End Sub
```

Agora vamos a uma aplicação extremamente útil quando queremos importar dados digitados em outra máquina, que exportou seus dados em um arquivo texto para ser enviado a um escritório central, via email, por exemplo. No exemplo, vamos supor que o arquivo Excel é o mesmo em todos os locais, mas as planilhas tem nomes diferentes, para evitar conflitos de nomes iguais durante a importação.
Os digitadores externos voa clicar em um botão, ou uma figura, para gerar um arquivo texto com todos os seus dados, usando a técnica de delimitar os campos do registro de exportação com um delimitador qualquer.
Basicamente o formato que eu sugiro para esse arquivo é:
- Registro mestre de planilha: * <dlm> nome (onde dlm é um delimitador qualquer, que não pode existir no nome)
- Registro de célula não vazia desta planilha: linha <dlm> coluna <dlm> dado

Ora, a única restrição é que o <dlm> não ocorra dentro de nenhuma célula das planilhas.
O único caractere que eu conheço que não dá para ser colocado em uma célula é o TAB, porque ao se digitar um tab a célula à direita é posicionada. No vb, temos uma constante para indicar este símbolo: vbTab
Por isso, eu aconselho que sempre se use o vbTab como delimitador em arquivos de troca de dados.
Vou então listar agora as duas sub-rotinas que devem ser codificadas: Uma para criar o arquivo texto a partir de uma planilha (exporta) e outra para criar uma planilha a partir de um arquivo texto (importa).

```
Sub Exporta()
  Dim caminho, nome, p, c, w
  nome = Range("nome")
  caminho = ActiveWorkbook.Path
  Set w = Workbooks.Open(caminho & "\" & nome & ".xls")
  Open caminho & "\" & nome & ".txt" For Output As #1
  For Each p In w.Sheets
     Print #1, "*" & vbTab & p.Name
     For Each c In Range(p.Cells(1, 1), p.Cells(1, 1).SpecialCells(xlLastCell))
        If c > "" Then
          Print #1, c.Row & vbTab & c.Column & vbTab & c
        End If
     Next
  Next
  Close #1
  w.Close
End Sub
```

```
Sub Importa()
   Dim nome, caminho, w As Workbook, i As Integer
   Dim linha, pedacos, p, ADeletar
   caminho = ActiveWorkbook.Path
   nome = Range("nome")
   Set w = Workbooks.Add
   w.SaveAs caminho & "\" & nome & ".xls"
   i = w.Sheets.Count
   Application.DisplayAlerts = False
   For i = i To 2 Step -1
     w.Sheets(i).Delete
   Next
   Set ADeletar = ActiveSheet
   Open caminho & "\" & nome & ".txt" For Input As #1
   Do Until EOF(1)
      Line Input #1, linha
      pedacos = Split(linha, vbTab)
      If pedacos(0) = "*" Then
         Set p = w.Sheets.Add(w.Sheets(1))
         w.Sheets(1).Name = pedacos(1)
      Else
         p.Cells(CInt(pedacos(0)), CInt(pedacos(1))) = pedacos(2)
      End If
    Loop
    ADeletar.Delete
    Close #1
    w.Save
End Sub
```

Você já deve estar qualificado para analisar este código e entender cada um de seus comandos.

Com isso, encerramos o nosso curso esperando que todos façam uso deste ferramental fantástico que o Excel coloca à nossa disposição para fazer virtualmente qualquer coisa.

Entretanto, para tirar proveito desta linguagem ou de qualquer outra, você deve ganhar experiência em programação. Depois de algum tempo você vai conseguir soluções simples e eficazes para solucionar problemas que hoje parecem impossíveis de serem resolvidos.

O Instituto Columbia poderá oferecer a vocês um curso avançado, onde os alunos (preferentemente de uma mesma empresa) vão propor problemas para serem solucionados usando o VBA do Excel. Isso, além de acelerar o seu aprendizado, vai ser útil desde a primeira aula para dar vantagem competitiva e/ou para reduzir custos de sua empresa. Assim o curso se pagará através dos benefícios que ele proporcionará de imediato.

Até a próxima jornada, e bem-vindos à comunidade de programadores.