

Introdução à programação em VBA

ÍNDICE

Enquadramento	2
Noções básicas sobre programação	2
Criar Macros em Microsoft Excel	4
Accionar a barra de ferramentas de Visual Basic	4
Gravar uma macro	5
Correr uma Macro	6
Editar a Macro	6
Definição de variáveis	7
Tipos definidos (“User defined type”)	9
Declaração de constantes	9
Declaração Explícita/Implícita	10
Procedimentos e âmbito das variáveis	10
Module	10
Subroutine	11
Function	11
Passagem de argumentos	12
Âmbito das variáveis	12
Vectores e matrizes (“Arrays”)	13
Estruturas de decisão	14
If...Then...Else...End If	14
Select Case...Case...End Select	15
Estruturas “Loop”	15
For...Next	15
For Each...Next	16
Do...Loop	16
Do While...Loop	16
While...Wend	16
Do Until...Loop	16
Utilidades	17
Comentários no código	17
Continuação de linha de código	17
Message Box	17
Manuseamento de “Strings”	18
Função Len	18
Função Mid	18
Função Split	19
Juntar Strings	19
Conversão entre tipos de variáveis	20
Função Str	20
Função Int	20

Enquadramento

A utilização do computador como uma ferramenta de trabalho e de cálculo torna-se cada vez mais importante do dia-a-dia uma vez que permite a execução de tarefas repetidas, que de outra forma demorariam imenso tempo a efectuar e com potencial introdução de erros humanos. Desta forma, pretende-se introduzir a programação em Visual Basic para Aplicações (VBA) como uma ferramenta simples de cálculo, levando a uma utilização das normais folhas de cálculo (neste caso o Microsoft Excel) com mais eficiência e versatilidade, bem como proporcionar uma iniciação ao cálculo numérico, normalmente utilizado para resolução de problemas relacionados com os mecanismos de transporte e dispersão de poluentes no ambiente.

A opção pela linguagem VBA tem a mais valia de ser uma linguagem em que os interpretadores de código-fonte se encontram incorporados com as versões standard do Microsoft Excel e que funciona no sistema operativo mais popular e utilizado no mundo, o Microsoft Windows. Assim, assume-se que o acesso à ferramenta não é crítico, como acontece com outras linguagens de programação que necessitam de compiladores específicos. Além disso, a sintaxe do VBA é relativamente fácil, o que possibilita a sua rápida aprendizagem, ficando desta forma o aluno habilitado a utilizá-lo não só no âmbito da cadeira, mas também na utilização do computador como ferramenta de trabalho académico e profissional.

Noções básicas sobre programação

Mito académico nº1: A programação aparece sempre como um bicho-de-sete-cabeças, como algo de muito complicado. É preciso desmistificar este complexo.

As linguagens de programação são, como o nome indica, linguagens. São uma forma de comunicarmos com o computador. Existem várias linguagens de programação (C, C++, C#, Fortran, Visual Basic, Java, etc). Cada uma apresenta uma forma característica de comunicar com o computador de forma a ele perceber as instruções dadas pelo programador, tendo sido desenvolvidas várias linguagens que, de uma forma ou de outra, respondem a problemas específicos de forma diferente, mas que no fundo, aplicam os mesmos fundamentos básicos. Por exemplo, o FORTRAN (acrónimo para **F**ormula **T**ranslation) é a linguagem de programação mais utilizada em cálculo científico, pois foi optimizado e orientado para este tipo de funções, possibilitando, por

exemplo, o acesso a um elevado número de funções e bibliotecas matemáticas, que outras linguagens não têm. No entanto, isto não impossibilita que a mesma tarefa não possa ser executada por dois programas escritos em linguagens diferentes. A ideia da programação, é no fundo escrever num ficheiro de texto, uma série de instruções seguindo uma certa sintaxe (regra), que seja reconhecida por compilador ou interpretador. Não entrando em detalhes sobre as diferenças entre um compilador e um interpretador, o importante é saber que estes programas lêem o ficheiro de texto, normalmente denominado de código-fonte, e que convertem essa informação em linguagem-máquina (linguagem directamente entendida pelo computador) para que o computador execute as instruções definidas. Por exemplo, para poder estar a escrever este texto utilizei o programa Microsoft Word. O Microsoft Word foi escrito numa determinada linguagem de programação (possivelmente envolvendo um elevado número de ficheiros de código) com uma série de instruções, em que por exemplo uma delas seria contar o número de palavras presentes no texto. Esta instrução seria, em português, qualquer coisa como dizer ao computador: “Percorre todas as linhas deste ficheiro, e conta quantas palavras existem em cada linha. Faz o somatório do número de palavras por linha. Escreve o resultado na barra localizada no canto inferior esquerdo do ecrã “. Em Visual Basic, poderia traduzir-se estas instruções escrevendo código-fonte do seguinte modo:

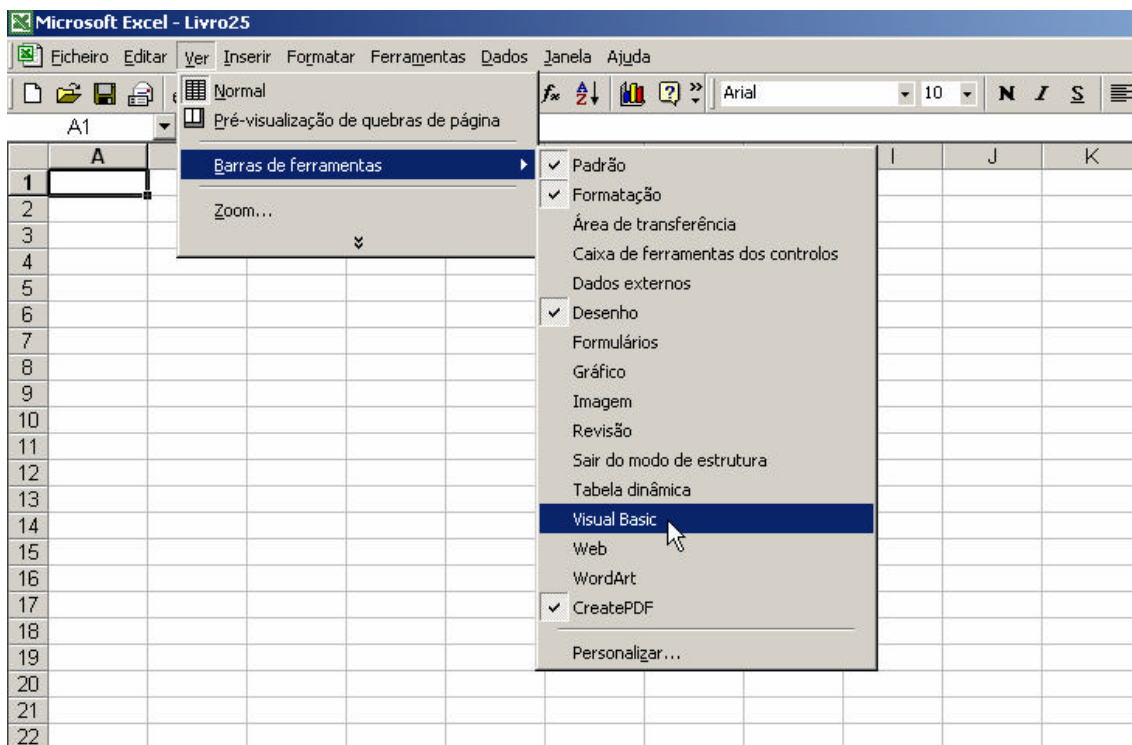
Código-fonte em Visual Basic	Tradução
Dim NumberOfWords As Integer = 0	Define “NumberOfWords” como sendo um número inteiro igual a zero
For Each Line In File.Lines	Por cada linha do conjunto de linhas que compõe o ficheiro
For Each Word In Line.Words	Por cada palavra do conjunto de palavras que compõe uma linha
NumberOfWords = NumberOfWords + 1	Adiciona mais uma palavra ao número total de palavras
Next	Passa à próxima palavra
Next	Passa à próxima linha
StatusBar.Text = Str(WordCount) + “words”	Escreve o número total de palavras do ficheiro na “StatusBar” (Barra de estado)

Como se pode ver, o raciocínio aplicado para contar o número de palavras num ficheiro é o mesmo que qualquer pessoa aplicaria, isto é, percorrer linha a linha todas as linhas do ficheiro e contar em cada linha o número de palavras, começando em zero. No entanto, torna-se bastante tedioso contar o número de palavras manualmente. Assim, facilmente se consegue programar (instruir) o computador a fazê-lo, tal como outro tipo de operações repetitivas ou de cálculo mais complexo.

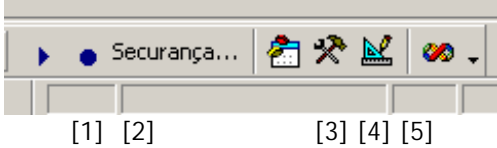
Criar Macros em Microsoft Excel

O objectivo das macros é programar funções típicas do Microsoft Excel de forma a facilitar a execução de tarefas repetitivas. A criação de uma Macro é semelhante à gravação de uma cassete, inicia-se a gravação e realiza-se a tarefa pretendida que é convertida automaticamente em código de Visual Basic (VB). A tarefa pode ser, por exemplo, abrir um determinado ficheiro de resultados, escolher algumas colunas e realizar um determinado tipo de gráfico. Com a utilização da macro podemos repetir este conjunto de tarefas o número de vezes que for necessário poupando bastante tempo no caso de estarmos a analisar uma grande quantidade de resultados.

Accionar a barra de ferramentas de Visual Basic



Selecionar no menu Ver (ou View), Barras de ferramentas (Toolbars), Visual Basic. Isto faz aparecer uma pequena janela como a seguinte. Arraste a pequena janela para a barra de estado em baixo.



[1] [2] [3] [4] [5]

[1] Escolher e correr as macros presentes

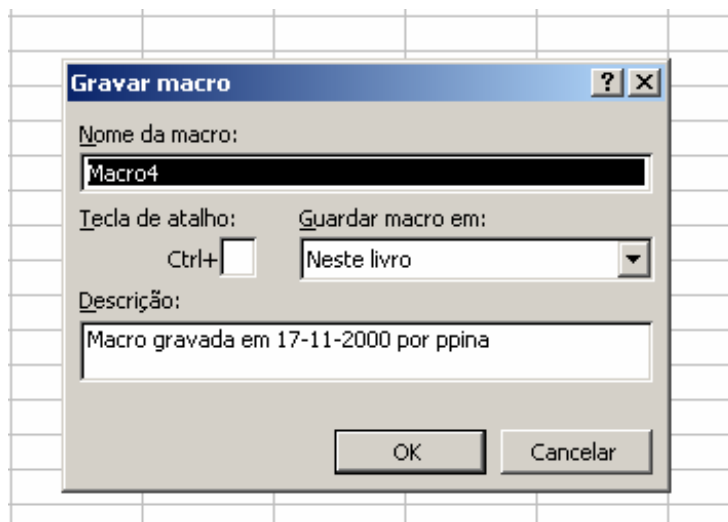
[2] Criar novas macros

[3] Abrir o editor de Microsoft Visual Basic

[4] Mostrar a caixa de ferramentas dos controlos de Microsoft Visual Basic

[5] Editar os controlos de Microsoft Visual Basic inseridos na página de Excel

Gravar uma macro

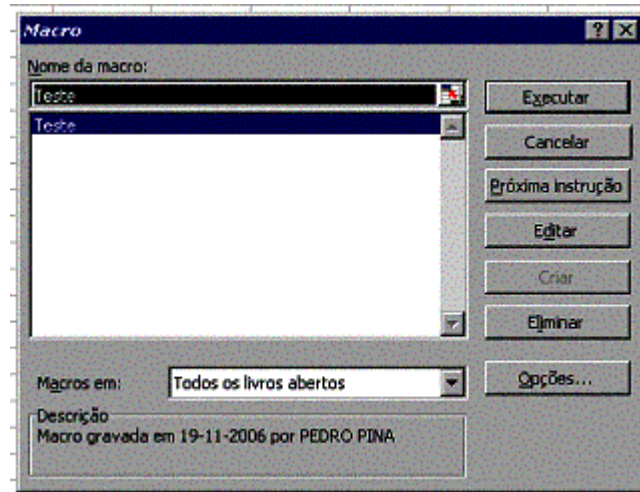


É possível nomear a macro e indicar uma tecla de atalho que permite arrancar automaticamente a macro, bem como escrever alguns comentários informativos.

Clicando em OK inicia-se o processo de gravação da macro, ou seja, todas as tarefas que forem realizadas daqui para a frente vão ser convertidas automaticamente em código Visual Basic. Para terminar a gravação da macro é só clicar novamente no botão [2] (que quando está a gravar assume temporariamente a forma de um quadrado azul).

Correr uma Macro

Clicando no botão “Play” [1] surge no ecrã a lista de macros associadas ao ficheiro. Ao executar dá-se início à sequência de tarefas que foram previamente gravadas.



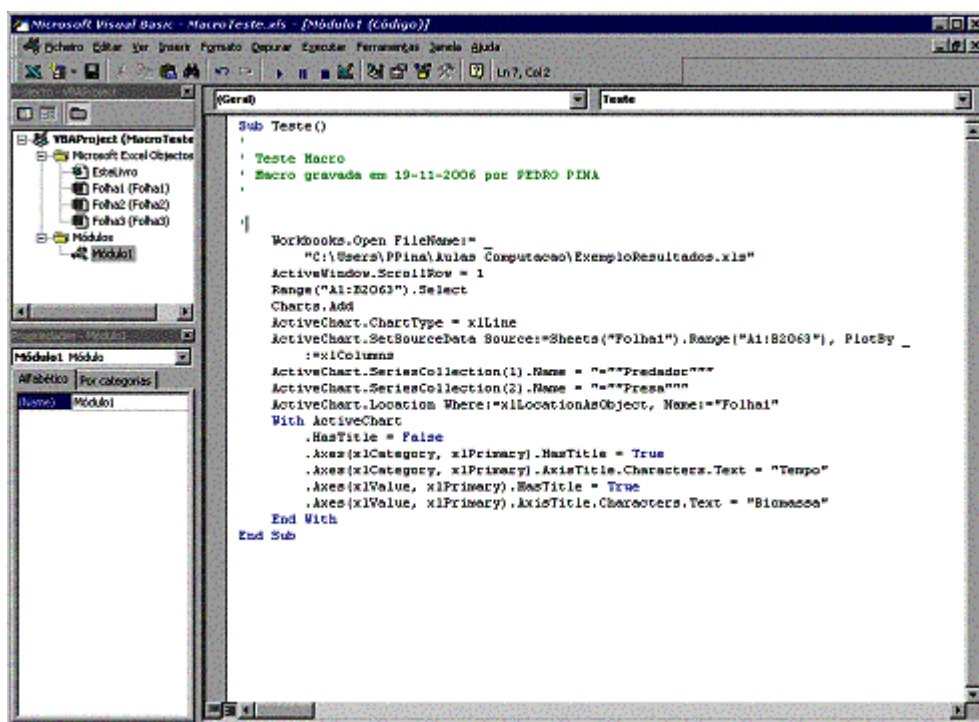
A macro executa literalmente todas as tarefas que foram gravadas. Imaginemos por exemplo que se pretende abrir um ficheiro de resultados chamado resultados.xls com 3 colunas e 100 linhas que se encontra guardado em C:\resultados e fazer um gráfico de pontos com a coluna 1 e 3. Para a macro ser bem sucedida sempre que for chamada, estas condições têm que ser mantidas, ou seja, se o ficheiro tiver outro nome ou estiver guardado noutro lugar do disco a macro dá imediatamente indicação de erro. Pode ainda acontecer o novo ficheiro de resultados ter 200 linhas, neste caso só as primeiras 100 linhas vão ser utilizadas, pois foi este o limite imposto na realização da macro; ou ainda a coluna 3 estar vazia o que irá gerar obviamente um gráfico sem pontos.

No entanto, todos estes inconvenientes podem ser resolvidos por meio de programação de operações simples na macro, em que parâmetros como o nome e o caminho para o ficheiro, ou ainda os limites para a realização de gráficos podem ser definidos pelo utilizador através de uma interface programável. Para aceder ao ambiente de programação é necessário editar a macro como é demonstrado no passo seguinte.

Editar a Macro

Clicando no botão [3] o Excel arranca o Editor do Microsoft VB. Este “programa” é o ambiente de trabalho que permite visualizar e alterar o código que está por trás da

macro. Este código foi gerado automaticamente quando começamos a gravar a macro e traduz as tarefas realizadas, para a linguagem de programação VB.



A janela de topo no lado esquerdo indica os vários elementos do projecto em que estamos a trabalhar, neste caso o MacroTeste.xls. Como é possível observar este projecto inclui 3 “worksheets” do Excel e um módulo podendo ainda incluir outros elementos de que falaremos mais tarde. O módulo em questão encontra-se aberto na janela central e não é mais do que uma sub-rotina onde foram programadas as tarefas realizadas pela macro.

A janela no canto inferior esquerdo indica as propriedades de cada objecto, módulos, “worksheets”, controlos, etc e permite alterar as características, ou definições de cada um destes elementos.

Definição de variáveis

O computador precisa de saber quando executa uma operação se está a trabalhar com números ou com letras, ou com outro tipo de variáveis. As variáveis são usadas para temporariamente armazenar valores (numéricos ou não-numéricos), tendo cada variável associado um *Nome* e um *Tipo*. Assim, a definição de variáveis, em VBA, é feita da seguinte forma:

Dim Nome **As** Tipo

Por exemplo, um programa que precise de ter acesso aos dados dos alunos de uma escola, precisaria de variáveis como o nome do aluno, data de nascimento, nº do bilhete de identidade, turma a que pertence, etc. O nome do aluno pode ser definido por uma variável composta por caracteres denominada “Nome”. Em Visual Basic estas variáveis denominam-se do tipo **String**. O número do aluno (“Numero”) pode ser definido por um número inteiro (**Integer**). A data de nascimento (“DataDeNascimento”) pode ser definida por uma data (**Date**) e por aí em diante.

Dim Nome **As** String

Dim Numero **As** Integer

Dim DataDeNascimento **As** Date

Nome = “Luís Fernandes”

Numero = 91

DataDeNascimento = “19-08-1977”

O VBA suporta os seguintes tipos de variáveis (apresentados os mais comuns):

Tipo de variável		Domínio de aplicação	
		Mínimo	Máximo
Byte		0	255
Boolean		True ou False	
Integer		-32768	32767
Long		-2147483648	2147483647
Single	Valores negativos	-3.402823e38	-1.401298e-45
	Valores positivos	1.401298e-45	3.402823e38
Double	Valores negativos	-1.79769313486232e308	-4.94065645841247e-324
	Valores positivos	4.94065645841247E-324	1.79769313486232e308
Date		1 de Janeiro de 100	31 de Dezembro de 9999
Object		Qualquer tipo de variável	
String		Conjunto de caracteres (limite de aprox. 2 biliões)	
Variant		Variável com valor numérico ou não-numérico (caracteres)	
“User defined type”		Tipo de variável definido pelo utilizador, que pode consistir de uma variável formada pelo conjunto de outras variáveis primárias (ver exemplo)	

Tipos definidos (“User defined type”)

Os tipos definidos são variáveis definidas pelo utilizador, que podem consistir de uma variável formada pelo conjunto de outras variáveis primárias. Um exemplo de um tipo definido, pode ser a variável do tipo “Estudante”.

Type Estudante

Nome **As String**

DataDeNascimento **As Date**

Numero **As Integer**

MediaCurso **As Single**

PagouAsPropinas **As Boolean**

End Type

Dim UmEstudante **As** Estudante

Dim OutroEstudante **As** Estudante

UmEstudante.Nome = “Luís Fernandes”

UmEstudante.PagouAsPropinas = **False**

OutroEstudante.MediaCurso = 15.3

OutroEstudante.Numero = 103

Declaração de constantes

É possível declarar constantes nos programas. Por exemplo, imagine-se que se está a fazer um programa em que se recorre várias vezes ao valor de Pi e que se necessita de alguma precisão nos cálculos a efectuar. Para evitar ter de escrever o valor de Pi no programa, pode-se declarar a constante “Pi” como tendo o valor 3.1415...

Const Pi **As Double** = 3.1415926535897932384626433832795

Perímetro = 2 * Pi * Raio

Área = Pi * Raio * Raio

Declaração Explícita/Implícita

Um passo importante na definição de variáveis é a opção “Option Explicit”. Este comando deve ser sempre utilizado, principalmente quando os programas apresentam uma complexidade grande, uma vez que obrigam a que não existam variáveis com a mesma denominação e que todas as variáveis utilizadas sejam declaradas, isto é, seja dito ao computador de que tipos são.

Option Explicit

Esta declaração explícita de variáveis previne igualmente erros ortográficos de programação, como no exemplo abaixo.

```
Numero = 36  
Raiz = Sqr(Numro)
```

Foi iniciada uma variável “Numero” com o valor de 36 e pretende-se saber a sua raiz quadrada (utilizando a função “Sqr”, intrínseca do VBA, e que devolve a raiz quadrada de um número). Mas por erro do programador, omitiu-se uma letra na chamada da função “Sqr”, utilizando-se uma variável “Numro”. Neste caso, o valor de “Raiz” vai ser igual a 0, uma vez que a variável “Numro” não foi inicializada, tendo sido assumido que tinha o valor 0 (raiz quadrada de zero é igual a zero). O programa correu perfeitamente sem devolver nenhuma mensagem de erro, uma vez que não foi feita nenhuma operação ilegal, pois a opção de definição explícita de variáveis não foi activada. No caso de ter sido activada, o programa não correria, avisando prontamente que a variável “Numro” não tinha sido declarada, evitando-se assim erros deste tipo.

Procedimentos e âmbito das variáveis

Um programa pode ser construído de várias formas. Se o programa for pequeno, geralmente, é apenas constituído apenas por um “corpo” principal. Felizmente, se o programa começar a tornar-se complexo, pode-se subdividir em várias partes, em que cada uma pode fazer uma determinada operação. O VBA permite subdividir o programa em módulos (**Module...End Module**), sub-rotinas (**Sub...End Sub**) ou funções (**Function... End Function**).

Module

Um módulo pode conter várias sub-rotinas e funções, e consiste essencialmente em porções de código organizadas por efectuarem operações comuns. De forma a

resguardar a informação dentro de um módulo, pode-se fazer uso da propriedade **Private**, que permite que as variáveis assim definidas, apenas possam ser alteradas dentro do próprio módulo e nunca a partir, quer do programa principal, quer de outro módulo. Quando se pretende dar acesso a uma determinada variável que se encontra dentro de um módulo, esta terá de ser definida como **Public**. O mesmo acontece com a declaração de sub-rotinas e funções que podem ser declaradas como públicas ou privadas.

Subroutine

Uma sub-rotina é uma porção de código identificada por um nome, e que pode ser chamada a qualquer altura do código. Este tipo de estrutura pode receber variáveis por argumento, isto é, podem-lhe ser dadas variáveis para efectuar uma certa operação. Por exemplo, se quisermos efectuar uma soma de dois números inteiros ‘a’ e ‘b’ que é dada pela variável ‘c’, podemos construir uma sub-rotina “Soma” da seguinte forma.

```
Dim a, b, c As Integer  
Call Soma(a, b, c)  
  
Sub Soma(a As Integer, b As Integer, c As Integer)  
c = a + b  
End Sub
```

Function

As funções são como sub-rotinas, mas com a diferença que devolvem um valor como resultado da operação, logo terão de ter um tipo associado.

```
Dim a, b, c As Integer  
c = Soma(a,b)  
  
Function Soma(a As Integer, b As Integer) As Boolean  
Soma = a + b  
End Function
```

Passagem de argumentos

Os argumentos podem ser passados, essencialmente de duas maneiras: por referência (**ByRef**) ou por valor (**ByVal**). Os argumentos passados por referência podem ser alterados dentro da sub-rotina ou função, enquanto os passados por valor, não são alterados dentro da sub-rotina ou função.

```
Sub Teste
Dim a As Integer
a = 10
Call PorValor(a)
MsgBox "Teste após por valor: " & a
Call PorReferencia(a)
MsgBox "Teste após por referência: " & a
End Sub

Sub PorReferencia(ByRef a As Integer)
a = a + 1
MsgBox "Por Referência: " & a
End Sub

Sub PorValor(ByVal a As Integer)
a = a + 1
MsgBox "Por Valor: " & a
End Sub
```

Âmbito das variáveis

As variáveis podem ser definidas em várias secções do programa:

- no programa principal, passando a poderem ser acedidas a partir de qualquer parte do programa;
- no módulo, podendo ser acedidas em qualquer ponto do módulo ou a partir de outras partes do programa se forem definidas como públicas;
- na sub-rotina, tendo o seu tempo de vida estabelecido pela execução da sub-rotina e apenas podem ser utilizadas dentro da sub-rotina;
- na função, semelhante à sub-rotina.

Há que ter em atenção que, variáveis definidas como globais no programa, podem ser ultrapassadas por variáveis locais (definidas em módulos, sub-rotinas ou funções) com o mesmo nome.

Vectores e matrizes (“Arrays”)

Variáveis de um determinado tipo, podem ser armazenadas em vectores ou matrizes (normalmente denominados de “arrays”). Os *arrays* podem, utilizando o mesmo nome, armazenar uma infinidade (limitação dada pela capacidade do computador) de variáveis do mesmo tipo, identificadas através de um índice. O armazenamento pode ser feito em vectores uni ou multi-dimensionais. O utilizador tem liberdade de fazer esse armazenamento consoante o problema que pretende resolver. Se se quisesse armazenar numa só variável uma turma com 25 estudantes, esta poderia organizar-se de duas formas: (1) numa lista ou, por exemplo, (2) de acordo com a sua distribuição numa sala de aula (assumindo que a sala de aulas se encontrava organizada em 5 colunas de 5 mesas cada).

No caso 1,

Dim Turma(1 To 25) As Estudante
--

No caso 2,

Dim Turma(1 To 5, 1 To 5) As Estudante
--

Nos dois casos, verifica-se que a dimensão dos *arrays* é conhecida *à priori*, utilizando assim o que é conhecido por alocação estática, uma vez que as dimensões são conhecidas e não podem ser alteradas. Mas imagine-se que não se sabe a dimensão de um determinado *array*. O VBA permite a alocação dinâmica, nesses casos. Imagine-se que a determinada altura do programa se deseja adicionar mais 2 alunos à turma. Basta para isso utilizar o comando **ReDim**.

ReDim Turma(1 To 27) As Estudante
--

Neste caso, o array Turma é novamente alocado com 27 posições, o que permitiria adicionar os dois novos alunos. No entanto, este comando, apagaria da memória toda a

informação existente na variável Turma. Isto pode ser o desejado, mas o VBA permite a realocação sem perda de informação, bastando para tal utilizar o comando **Preserve**.

```
ReDim Preserve Turma(1 To 27) As Estudante
```

Note-se que o dimensionamento de um *array* pode ser feito especificando o índice inicial, como nos casos apresentados acima, ou assumir o índice da primeira casa do array como sendo zero (valor assumido por defeito pelo VBA).

```
Dim Turma(27) As Estudante
'Estudante(0).Nome = "Zé"
'Estudante(1).Nome = "Chico"
'...
'Estudante(27).Nome = "António"
```

Estruturas de decisão

As estruturas de decisão permitem o programador decidir durante a execução do programa entre uma, duas ou mais opções. Existem dois tipos principais de estruturas de decisão.

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

```
Dim a As Integer
a = 1

If a > 0 Then
    MsgBox "O valor é maior que zero"
End If

If a > 0 Then
    MsgBox "O valor é maior do que zero"
Else
    MsgBox "O valor não é maior do que zero"
End If

If a > 0 Then
    MsgBox "O valor é maior do que zero"
ElseIf a = 0 Then
    MsgBox "O valor é igual a zero"
```

Else

MsgBox “O valor é menor do que zero”

End If

Select Case...Case...End Select

Dim a As Integer

Select Case a

Case -1

MsgBox “O valor é igual a -1”

Case 0

MsgBox “O valor é igual a zero”

Case 1

MsgBox “O valor é igual a 1”

Case Else

MsgBox “O valor é outro que não -1, 0 ou 1”

End Select

Estruturas “Loop”

As estruturas “loop” permitem repetir partes do código um determinado número de vezes que pode ser pré-definido ou até que uma determinada condição se verifique.

For...Next

Dim i As Integer

For i = 1 To 10

MsgBox “Esta é a operação nº ” & i

Next

For i = 1 To 10

MsgBox “Esta é a operação nº ” & i

If i = 7 Then Exit For

Next

For Each...Next

```
Dim Estudante As Estudante
Dim Estudantes As Collection1

For Each Estudante In Estudantes
    MsgBox Estudante.Nome
Next
```

Do...Loop

```
Dim i As Integer
i = 0
Do
    MsgBox “Esta é a operação nº ” & i
    If i > 10 Then Exit Do
Loop
```

Do While...Loop

```
Dim i As Integer
i = 0
Do While i < 10
    MsgBox “Esta é a operação nº ” & i
Loop
```

While...Wend

```
Dim i As Integer
i = 0
While i < 10
    MsgBox “Esta é a operação nº ” & i
Wend
```

Do Until...Loop

```
Dim i As Integer
i = 0
Do Until i = 10
    MsgBox “Esta é a operação nº ” & i
Loop
```

¹ Mais adiante iremos ver o que é uma Collection, por enquanto pode-se assumir como uma lista de estudantes que pode ser percorrida de forma sequencial, a começar no primeiro até ao último.

Utilidades

Existem uma série de funções úteis fornecidas pelo VBA que podem ser utilizadas frequentemente ao escrever-se um programa. Recomenda-se a utilização da Ajuda do Microsoft Visual Basic, que pode ser acedida através da tecla F1 no Editor de Microsoft Visual Basic ou através do menu Ajuda (ou Help), para ficar a saber quais são as funções disponibilizadas. Ficam em seguida alguns exemplos dessas funções, bem como algumas outras noções de sintaxe do VBA.

Comentários no código

No interior do código podem ser inseridos comentários de forma a ajudar e orientar o programador e futuros utilizadores do código. O interpretador do código simplesmente ignora esses comentários quando faz a compilação do programa, pelo que a introdução destes não afecta a execução do programa. Para inserir comentários basta precedê-lo de um apóstrofo (ver exemplo abaixo). O comentário aparece no editor de código com a cor verde. O comentário pode ser inserido no princípio ou a meio de uma linha de código. No entanto, tudo o que está escrito à esquerda do apóstrofo é considerado como comentário.

```
'Programa  
Dim n As Integer 'n é o número de estudantes
```

Continuação de linha de código

Por vezes torna-se necessário, por questões de espaço e de leitura do código, continuar a linha de código na linha seguinte. O exemplo seguinte demonstra como isso pode ser feito. Note-se que o “underscore” que define a quebra de linha deve ser precedido de um espaço.

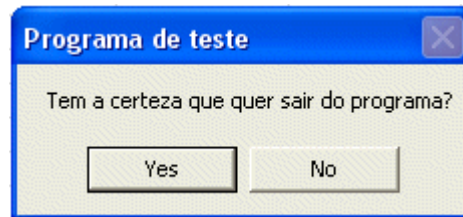
```
Dim a, b, c As Integer  
a = b + c*b / sqr(b) + _  
    c*b*0.5
```

Message Box

A “Message Box” é uma potencialidade do VBA que utiliza a caixas de diálogo do Microsoft Windows para comunicar com o utilizador do programa. Para aceder e configurar uma “Message Box” basta chamar a função “MsgBox”.

MsgBox [Mensagem], [Estilo], [Título], [Ficheiro de Ajuda], [Contexto]

MsgBox “Tem a certeza que quer sair do programa?”, vbYesNo, “Programa de teste”



Esta função recebe por argumento a mensagem que se pretende dar ao utilizador (e que é obrigatório definir) e mais uma série de argumentos úteis para facilitar o diálogo que são facultativas de definir. São eles:

- o Estilo, que corresponde ao tipo de botões e de ícone identificativo para mensagem (estão disponíveis botões como “Yes”, “No”, “Cancel”, “OK”, etc e ícones como Exclamação, Informação, Questão, Crítico, etc).
- o Título, que aparece na barra superior azul da caixa
- o Ficheiro de Ajuda e o Contexto, são opções avançadas que não serão contempladas aqui

De referir que quando se utilizam argumentos opcionais, como os acima descritos, há alguns pormenores a ter em conta. Por exemplo, se o programador desejar definir apenas a mensagem e o título da caixa, e desejar utilizar o estilo por defeito deve chamar a função MsgBox da seguinte forma, deixando em branco o argumento para definição do estilo, utilizando uma vírgula.

MsgBox “Tem a certeza que quer sair do programa?”, , “Programa de teste”

Manuseamento de “Strings”

Função Len

Devolve o número de caracteres de uma **String**.

Dim n **As Integer**

Dim Palavra **As String**

n = Len(Palavra)

Função Mid

Selecciona um certo número de caracteres de uma **String** a partir de um determinado caractere. Útil para extrair informação de uma **String**.

```
Dim NumeroBI As String
```

```
Dim VectorNumeros(1 To 8) As String
```

```
Dim i As Integer
```

```
NumeroBI = "10252299"
```

```
For i = 1 To Len(NumeroBI)
```

```
    VectorNumeros (i) = Mid(NumeroBI, i, 1)
```

```
Next
```

```
'VectorNumeros(1) = "1"
```

```
'VectorNumeros(2) = "0"
```

```
'VectorNumeros(3) = "2"
```

```
'VectorNumeros(4) = "5"
```

```
'...
```

Função Split

Separa uma String a partir de um delimitador pré-definido e devolve um vector com o número de blocos que se encontravam separados pelo delimitador. Útil para, por exemplo, separar uma linha de caracteres separados por uma vírgula.

```
Dim Linha As String
```

```
Dim Colunas() As Integer
```

```
Linha = "23, 345, 32, 1, 90"
```

```
Colunas = Split(Linha, ",")
```

```
'Colunas(0) = 23
```

```
'Colunas(1) = 345
```

```
'Colunas(2) = 32
```

```
'Colunas(3) = 1
```

```
'Colunas(4) = 90
```

Juntar Strings

É possível juntar várias Strings numa só. Se se quiser juntar várias variáveis numa só String é possível utilizar o operador "+".

```
Dim Nome, Apelido, NomeCompleto As String
```

```
Nome = "Luís"
```

```
Apelido = "Fernandes"  
NomeCompleto = Nome + " " + Apelido
```

```
'NomeCompleto = "Luís Fernandes"
```

Note-se que os caracteres " " que definem um espaço entre o "Nome" e o "Apelido" podem ser substituídos pela função "Space", que recebe por argumento o número de espaços que se deseja introduzir na **String**. (e.g. Space(3) = " ")

Se as variáveis forem de tipos diferentes, terá então de ser utilizar o operador "&".

```
Dim Nome, NomeNumero As String  
Dim Numero As Integer
```

```
Nome = "Luís"  
Numero = 10  
NomeNumero = Nome + " " & Numero
```

```
'NomeNumero = "Luís 10"
```

Conversão entre tipos de variáveis

Função Str

Converte qualquer tipo de variável numa variável do tipo **String**.

```
Dim i As Integer  
Dim iStr As String
```

```
i = 10  
iStr = Str(i)  
'iStr = "10"
```

Função Int

Converte qualquer tipo de variável numérica para uma variável do tipo **Integer**.

```
Dim i As Integer  
Dim iStr As String
```

```
iStr = "10"  
i = Int(iStr)  
'i = 10
```