

Índice

MACROS.....	5
O QUE É UMA MACRO?	6
CRIAR MACROS.....	6
<i>Gravar uma Macro</i>	6
Exercício	8
Procedimento BackGround do Excel	9
<i>Programação em Visual Basic for Applications</i>	10
EXECUTAR UMA MACRO.....	12
<i>Tecla de Atalho – Shortcut Key</i>	12
<i>Botão na Toolbar</i>	13
Associar uma Macro a um Botão	13
Dissociar uma Macro de um Botão	15
<i>Run</i>	16
<i>Comando no Menu</i>	17
Associação de uma Macro a um Comando do Menu.....	17
Dissociação	19
<i>Editor de Visual Basic for Applications</i>	20
REMOVER MACROS	21
<i>Remoção de Macros em Ambiente Excel</i>	21
<i>Remoção de Macros no Editor de VBA</i>	21
EDITOR DE VISUAL BASIC FOR APPLICATIONS.....	22
PROJECT EXPLORER.....	24
PROPERTIES WINDOW.....	26
JANELA DE EDIÇÃO	27
OBJECT BROWSER	28
HELP ON-LINE	28
AS CORES DO VBA	29
FUNÇÕES E SUBROTINAS	30
SUBROTINAS.....	32
<i>Definição de SubRotinas</i>	32
<i>Trabalhar no Editor de VBA – Criar uma SubRotina</i>	33
<i>Execução de uma SubRotina</i>	34
FUNÇÕES	34
<i>Definição de Funções</i>	34
<i>Definição do tipo de parâmetros e do tipo da função</i>	35
<i>Trabalhar no Editor de VBA – Criar uma Função</i>	36
<i>Execução de uma Função</i>	37
Execução dentro de uma Célula.....	38
Execução dentro de uma Rotina.....	40
DIFERENÇAS ENTRE FUNÇÕES E ROTINAS	42
REGRAS PARA A PASSAGEM DE PARÂMETROS.....	42

Excel – Macros e Visual Basic for Applications

(versão Draft)

VARIÁVEIS	43
MANUSEAMENTO COM VARIÁVEIS	44
<i>O que são variáveis?</i>	44
<i>Associação de valores a variáveis:</i>	45
<i>Utilização de variáveis como se fossem valores:</i>	45
TIPOS DE VARIÁVEIS	46
DECLARAÇÃO DE VARIÁVEIS	47
VARIÁVEIS – VANTAGENS DA UTILIZAÇÃO	47
VARIÁVEIS DO TIPO OBJETO	48
<i>Declaração da Variável Objeto</i>	48
<i>Atribuição de uma variável Objeto</i>	48
<i>Utilização Genérica da Variável Objeto</i>	49
<i>Utilização Específica da Variável Objeto</i>	49
VARIÁVEIS – DECLARAÇÃO OPCIONAL E O TIPO VARIANT	51
<i>Os Prós e Contras da Utilização do tipo Variants</i>	51
Prós	51
Contras	51
<i>Variáveis – Declaração Forçada</i>	52
VARIÁVEIS – TIPOS DEFINIDOS PELO USUÁRIO - ESTRUTURAS	53
<i>Definição do Tipo</i>	53
<i>Utilização das Estruturas de Dados</i>	54
VARIÁVEIS – ARRAYS	55
<i>O que é um Array ?</i>	55
Array Uni-Dimensional	55
Array Bi-Dimensional	56
<i>Declaração de um array</i>	57
Arrays Uni-dimensionais	57
Arrays Bi-dimensionais	57
<i>Utilização de um Array</i>	58
Para acessar ao elemento	58
Atribuição de valores	58
<i>Option Base e Array Bounds</i>	59
CONSTANTES	60
O QUE SÃO CONSTANTES ?	61
INPUTBOX E MSGBOX	62
O QUE SÃO ?	63
INPUTBOX	63
<i>O que faz</i>	63
<i>Sintaxe</i>	63
<i>Parâmetros</i>	64
MSGBOX	65
<i>O que faz</i>	65
<i>Sintaxe</i>	65
<i>Parâmetros</i>	66
<i>Valores Produzidos</i>	69
DOMÍNIO DAS VARIÁVEIS, CONSTANTES E ROTINAS	70
O QUE É O DOMÍNIO ?	71
DOMÍNIO DAS VARIÁVEIS	71
<i>Âmbito do Procedimento</i>	72
<i>Âmbito do Módulo</i>	73
<i>Âmbito do Projecto</i>	74
DOMÍNIO DAS CONSTANTES	75
<i>Âmbito do Procedimento</i>	75

Excel – Macros e Visual Basic for Applications

(versão Draft)

Âmbito do Módulo.....	76
Âmbito do Projecto	76
DOMÍNIO DE SUBROTINAS E FUNÇÕES.....	77
ESTRUTURAS DE CONTROLO.....	78
O QUE SÃO ESTRUTURAS DE CONTROLO?	79
QUAIS AS ESTRUTURAS	79
IF-THEN-ELSE	80
<i>Função IF do Excel.....</i>	<i>80</i>
<i>Sintaxe da Estrutura If-Then-Else.....</i>	<i>80</i>
<i>Aplicação Prática</i>	<i>81</i>
<i>A instrução adicional ElseIf.....</i>	<i>83</i>
<i>Aplicação Prática</i>	<i>83</i>
FOR – NEXT	84
<i>Sintaxe.....</i>	<i>84</i>
<i>Aplicação Prática</i>	<i>84</i>
<i>A Função das Variáveis:.....</i>	<i>85</i>
<i>Construção do Ciclo:.....</i>	<i>85</i>
<i>Tradução Integral.....</i>	<i>86</i>
<i>Funcionamento do Ciclo:</i>	<i>86</i>
<i>Perigos associados à utilização do ciclo For-Next:.....</i>	<i>86</i>
<i>Outra Aplicação.....</i>	<i>87</i>
WHILE-WEND	88
<i>Sintaxe.....</i>	<i>88</i>
<i>Aplicação Prática</i>	<i>88</i>
<i>A Função das Variáveis:.....</i>	<i>89</i>
<i>Construção do Ciclo:.....</i>	<i>89</i>
<i>Tradução Integral.....</i>	<i>90</i>
<i>Funcionamento do Ciclo</i>	<i>90</i>
<i>Perigos associados à utilização do ciclo While-Wend.....</i>	<i>90</i>
<i>Outra Aplicação.....</i>	<i>91</i>
DO – LOOP	92
<i>Sintaxe.....</i>	<i>92</i>
<i>Aplicações Práticas.....</i>	<i>93</i>
SELECT CASE	95
<i>Sintaxe.....</i>	<i>95</i>
<i>Aplicação Prática</i>	<i>96</i>
<i>Construção da Estrutura</i>	<i>97</i>
FOR – EACH – NEXT	98
<i>Sintaxe.....</i>	<i>98</i>
<i>Aplicações Práticas.....</i>	<i>99</i>
<i>Utilizando Arrays.....</i>	<i>99</i>
<i>Construção do Ciclo.....</i>	<i>100</i>
<i>Utilizando Coleções de Objetos</i>	<i>101</i>
COLEÇÕES DE OBJETOS E OBJETOS	102
O QUE SÃO OBJETOS ?	103
OBJETOS: PROPRIEDADES, MÉTODOS E EVENTOS.....	103
<i>Propriedades</i>	<i>103</i>
<i>Métodos.....</i>	<i>103</i>
<i>Eventos.....</i>	<i>104</i>
OBJETOS MAIS UTILIZADOS NO EXCEL	105
Propriedades	105
Métodos	106
Propriedades	107
Métodos	108
Propriedades.....	109

Excel – Macros e Visual Basic for Applications

(versão Draft)

Métodos	110
Propriedades	111
Métodos	112
OBJETOS SINGULARES VS COLECÇÕES DE OBJETOS	113
INDEXAÇÃO DE COLECÇÕES POR NÚMERO OU NOME	114
<i>Indexação com Base em Números</i>	114
<i>Indexação com Base no Nome</i>	115
Vantagem	115
<i>O Objeto Range – uma excepção</i>	116
Tratamento como objeto:	116
Tratamento como colecção de objetos:	116
REFERÊNCIA IMPLÍCITA	117
<i>Declaração implícita da aplicação:</i>	117
<i>Declaração implícita do Workbook:</i>	118
<i>Declaração implícita da Worksheet:</i>	118
<i>Nível de referência a privilegiar</i>	119
MISCELLANEOUS	120
A INSTRUÇÃO WITH	121
<i>Aplicação Prática</i>	121
OUTRAS FUNÇÕES ÚTEIS DO VBA	122

Macros

O QUE É UMA MACRO?

Uma macro é um pequeno programa que contém uma lista de instruções a realizar no Excel. Como sendo um repositório de operações, uma macro pode executar um conjunto de tarefas através de um único procedimento o qual pode ser invocado rapidamente.

As instruções que formam o corpo da macro são escritas num código próprio para que o computador as possa entender, essa linguagem é designada por VBA – Visual Basic for Applications.

CRIAR MACROS

Existem duas possibilidades de criação de macros:

- Através do Gravador de Macros
- Utilizando o editor e programando em Visual Basic for Applications

Gravar uma Macro

1. Tools / Macro / Record New Macro. O Excel exibirá a caixa de diálogo da fig.1.

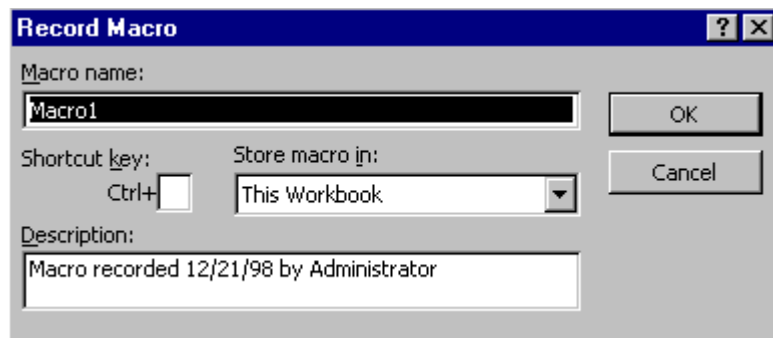


Fig.1 – caixa de diálogo para a gravação de macros.

Excel – Macros e Visual Basic for Applications

(versão Draft)

2. O nome da Macro será constituído por um conjunto de caracteres que identificarão a Macro e a sua funcionalidade.
3. Shortcut Key – é composto por uma combinação de teclas que poderão ser utilizadas para executar uma macro.
4. Opte por armazenar a macro numa das seguintes alternativas: This Workbook, New Workbook ou Personal Workbook, sendo que cada uma corresponde a macros de âmbito diferente. Assim deverão armazenar a macro na opção ThisWorkBook para que ela esteja activa sempre que o documento estiver aberto, e para que ela esteja armazenada no próprio arquivo não correndo o risco de ficar armazenada nas Macros do Excel. (na rede do ISCTE é impossível gravar a macro no Personal WorkBook.)
5. No campo Description, introduza um comentário à função – este poderá ser importante para que não seja esquecido o respectivo objectivo e funcionalidade.
6. Clique em OK para iniciar a gravação da macro – neste momento será exibida uma toolbar semelhante à da figura 2, e o ícone do mouse será transformado numa cassete, indicando o estado de gravação.



Fig.2 – Toolbar exibida para a gravação de macros.

7. Na toolbar Stop Record, existem dois botões: Stop Recording e Relative Reference – O botão de Stop Recording termina a gravação da macro, o botão de Relative Reference seleciona o modo de gravação da macro – se é feito com base em referências relativas (botão selecionado) ou referências absolutas.
8. No caso da toolbar Stop Record desaparecer, poderá voltar a exibi-la fazendo no menu a sequência: View / Toolbars / Stop Record (selecione a toolbar). Caso a toolbar não apareça listada a gravação de macros não está ativa.

Nota: Fique atento aos passos que dá quando está gravando pois tudo será registrado, o que significa que quando for executar a macro, esses procedimentos serão efetuados.

EXERCÍCIO

Objetivo: Gravar uma macro testando a diferença entre a execução de macros com referências absolutas e relativas.

1ª Fase: Macro com referências Absolutas

1. Acione a gravação da macro. Atribua-lhe o Short Key Ctrl+P
2. Certifique-se que o botão Relative Reference está desativado.
3. Clique na célula B3.
4. Escreva *ISCTE*
5. Formate a célula para Bold, Itálico, tamanho 18, Small Caps,... (utilize o Format / Font)
6. Na célula B4 escreva: *Av. Forças Armadas*
7. Na célula B5 escreva: *1700 Lisboa*
8. Pare a gravação da macro. – A macro está criada.
9. Apague tudo o que escreveu nas células da coluna B.
10. Clique na célula L8.
11. Aperte simultaneamente nas teclas Ctrl e P
12. O que aconteceu ?

2ª Fase: Macro com referências Relativas

1. Selecione a célula D5.
2. Acione a gravação da macro. Atribua-lhe o Short Key Ctrl+R
3. Selecione o botão Relative Reference.
4. Escreva *ISCTE* na célula que havia sido selecionada.
5. Formate a célula para Bold, Itálico, tamanho 18, Small Caps,... (utilize o Format / Font)
6. Na célula D6 escreva: *Av. Forças Armadas*
7. Na célula D7 escreva: *1700 Lisboa*
8. Pare a gravação da macro. – A macro está criada.
9. Apague tudo o que escreveu nas células da coluna D.
10. Clique na célula L8.
11. Carregue simultaneamente nas teclas Ctrl e R
12. O que aconteceu ? Porquê ?

PROCEDIMENTO BACKGROUND DO EXCEL

O Excel quando grava uma macro cria um objeto designado por *module* no workbook onde regista todas as operações gravadas em linguagem ***Visual Basic for Applications - VBA***. Este modulo não aparece no Excel com as restantes Sheets.

Para ser visualizado é necessário abrir o **Editor de Visual Basic for Applications**:

1. Tools / Macro / Macros
2. Selecciona-se a Macro e Clica-se no botão Edit
3. Poderá visualizar na área do lado direito o código VBA que está subjacente às macros que entretanto gravou. Aí poderá executar as mesmas tarefas que num editor de texto normal: escrever, apagar, copiar, mover,... mas instruções de VBA.
4. Tente fazer a leitura do que está escrito e compreenda o procedimento da macro.
5. Para regressar ao Excel basta File / Close and return to Microsoft Excel.

Programação em Visual Basic for Applications

1. Para acessar ao editor de Visual Basic for Applications: Tools / Macro / Visual Basic Editor (para se familiarizar mais com o editor consulte o capítulo *Editor Visual Basic for Applications* e o capítulo *Funções e sub-rotinas*)

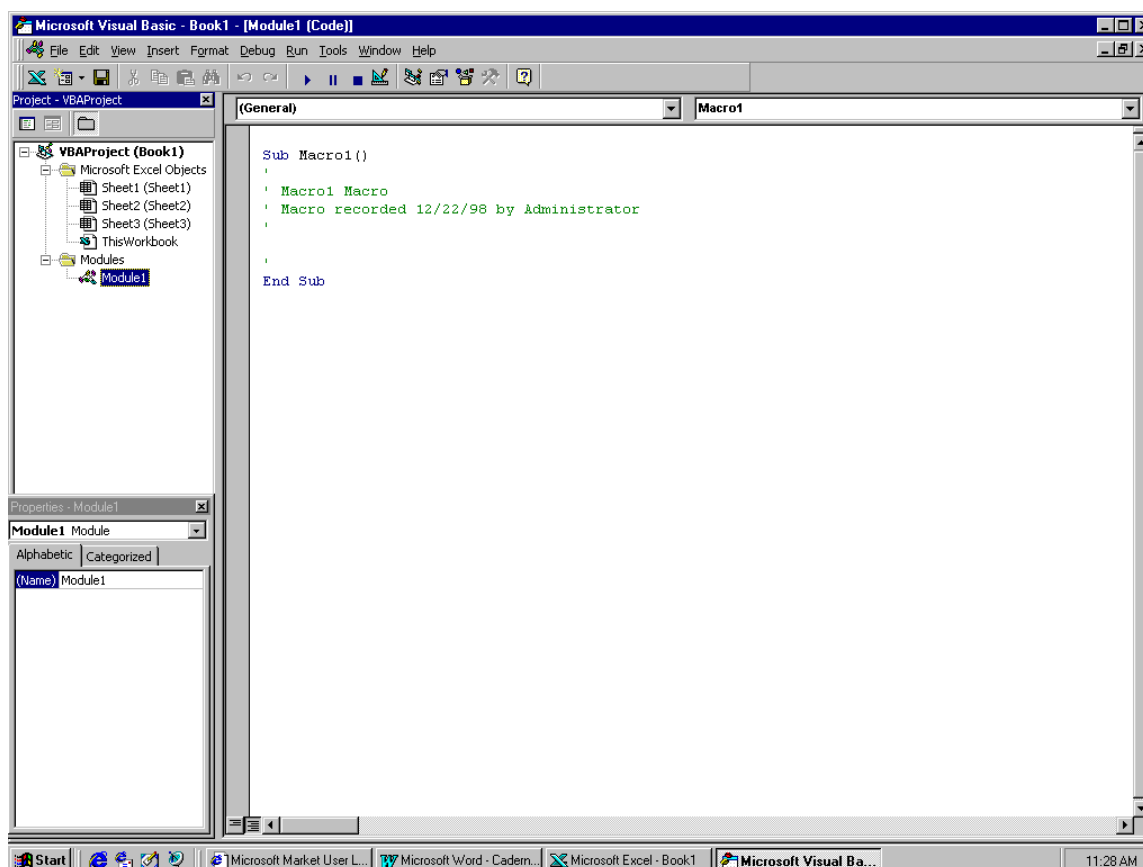


Figura 3 – Editor de Visual Basic for Applications

2. Para inserir um módulo faça Insert / Module – isto porque as macros que construir deverão ser escritas dentro de um módulo (repare na Figura 3 que o Module 1 está selecionado na janela de projeto e do lado direito tem-se uma área de edição onde poderão escrever as macros a executar)

Excel – Macros e Visual Basic for Applications

(versão Draft)

3. Pode agora programar os procedimentos manualmente ou com recurso a algumas funcionalidades do Editor:
 - a) Insert / Procedure – insere um procedimento para o qual deverá indicar o tipo de procedimento.
 - b) Quando estiver a introduzir as variáveis haverá de ser exibida a lista de tipos de variáveis possível.
4. Sempre que precisar construir uma macro mais ou menos complicada, se não conhecer muito de VBA poderá começar por gravar no Excel pequenas macros com operações muito simples e depois no editor de VBA tentar compreender a sua funcionalidade e assim construir uma macro cada vez mais complexa, completa e adequada às suas exigências.
5. Poderá em cada macro fazer uma chamada a uma outra macro, bastando para tal escrever o respectivo nome.
6. No Editor de Visual Basic for Applications poderá encontrar ajuda para o desenvolvimento do seu procedimento. Assim:
 - 6.1. Clique no Ícone do Object Browser ou View/Object Browser ou F2
 - 6.2. Na Caixa de Drop-Down onde aparece referido <All Libraries>, selecione a aplicação para a qual pretende ajuda – neste caso Excel. Convém referir que poderá utilizar as funções de toda e qualquer aplicação.
 - 6.3. Na área intitulada por Classes aparecem todos os objetos específicos para o manuseamento da aplicação selecionada. A classe especial designada por Globals refere-se às funções que estão disponíveis na aplicação independentemente dos objetos selecionados.
 - 6.4. Selecione um dos objetos e visualize do lado direito os Members of “<Elemento selecionado>”
 - 6.5. Para cada membro da classe dispõe de um help on-line que o esclarece sobre a respectiva função e funcionamento, dando exemplo que poderá testar. Para tal basta clicar sobre o botão de Help da janela do Object Browser.

EXECUTAR UMA MACRO

A execução de uma macro pode ser concretizada de diversas formas:

- Tecla de Atalho – Shortcut Key
- Botão na Toolbar
- Run
- Comando no Menu
- Editor de Visual Basic for Applications

Tecla de Atalho – Shortcut Key

A associação a teclas de atalho é realizada quando da criação da macro.

Botão na Toolbar

ASSOCIAR UMA MACRO A UM BOTÃO

1. View / Toolbars / Customize
2. Na janela do Customize selecione o Commands Tab
3. Selecione a categoria Macro (visualizará a caixa de diálogo exibida na Figura 4)

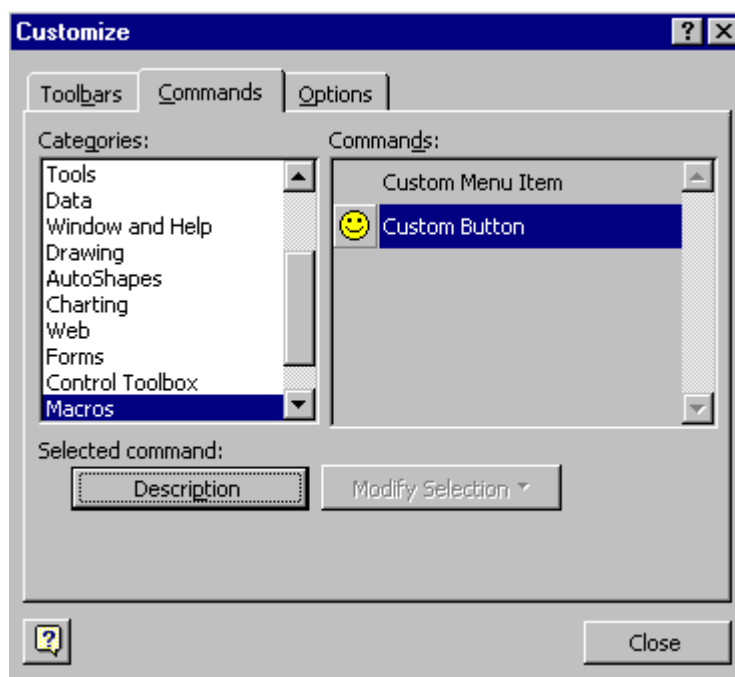


Fig.4 – caixa de diálogo para vinculação de botão a macros

Excel – Macros e Visual Basic for Applications

(versão Draft)

4. Na área respeitante aos Commands serão exibidos dois itens: Custom Menu Item e Custom Button. Selecione este segundo e arraste-o com o mouse até a Barra de Ferramentas onde pretende inseri-lo. Atenção só poderá inserir o botão quando o ponteiro do mouse ficar com a forma I. Nesse momento libertará o botão do mouse utilizado para o arrastamento e verificará que será criado um novo botão.
5. Na janela de Customize poderá ainda utilizar dois botões que se encontram na área do Selected Command:
 - a) Description – que exhibe um texto explicando o que o comando selecionado faz.
 - b) Modify Selection – semelhante ao clique sobre o botão criado, exhibe uma série de tarefas possíveis para configuração do botão (ver operações seguintes).

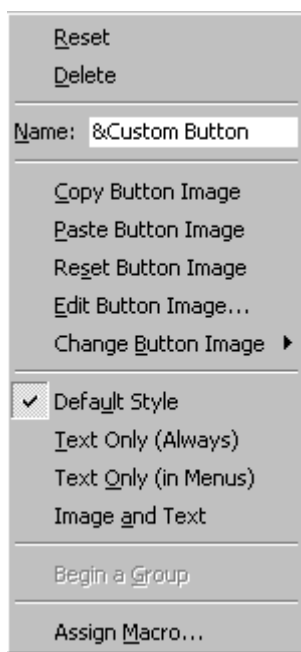


Fig.5 – Menu para configuração do botão da toolbar

Excel – Macros e Visual Basic for Applications

(versão Draft)

6. Clique sobre o botão Modify Selection- abrindo um menu de tarefas possíveis para a configuração do botão – Ver Figura 5
 - a) No último agrupamento de configurações possíveis existe uma opção designada por Assign Macro. Esta opção permite indicar qual a macro que deverá ser executada sempre que se clica no botão.
 - b) No terceiro agrupamento existem 4 estilos diferentes de exibir o botão: só com texto, com texto e imagem ou somente com imagem. Se seleccionar o estilo Texto e Imagem, será exibido no botão para além da imagem o nome associado ao botão.
 - c) Na opção Name indique o nome que pretende ver associado ao botão, por padrão aparece o nome da macro. (repare que aparece um & atrás da letra que aparece a sublinhado)
 - d) Para alterar a imagem associada ao botão poderá: optar por uma imagem diferente, alterar a que está a visualizar ou a construir a sua. Para isso clique sobre a tarefa de Change Button Image, selecione o logotipo pretendido, se nenhum o satisfizer e pretender criar o seu selecione o logotipo que representa um quadrado vazio. Para o poder (re)desenhar a imagem, na lista de tarefas disponibilizada pelo botão Modify Selection opte por Edit Button Image e crie a sua imagem que poderá reeditar.

DISSOCIAR UMA MACRO DE UM BOTÃO

1. Tools / Customize
2. Arraste o botão da toolbar até ao documento
3. Solte-o

Excel – Macros e Visual Basic for Applications

(versão Draft)

Run

1. Tools / Macro / Macros
2. Na caixa de diálogo Macros seleciona-se a macro pretendida na lista da Macro Name (Figura 6)
3. Clique sobre o botão Run

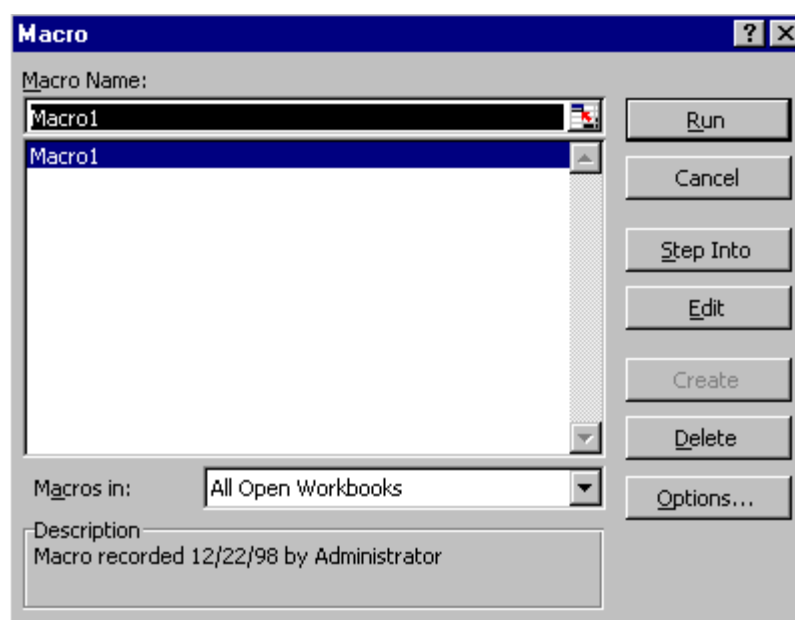


Fig.6– Janela para seleção da macro a executar

Comando no Menu

ASSOCIAÇÃO DE UMA MACRO A UM COMANDO DO MENU

1. View / Toolbars / Customize
2. Na janela do Customize encontra-se no Commands Tab
3. Selecione a categoria Macro
4. Na área relacionada aos Commands será exibido um item de Custom Menu Item, selecione-o e arraste-o com o mouse até a uma posição do menu que lhe interesse – por exemplo pode introduzir numa das listas de opções do menu (File, View,...) ou então criar como uma nova opção do Menu..

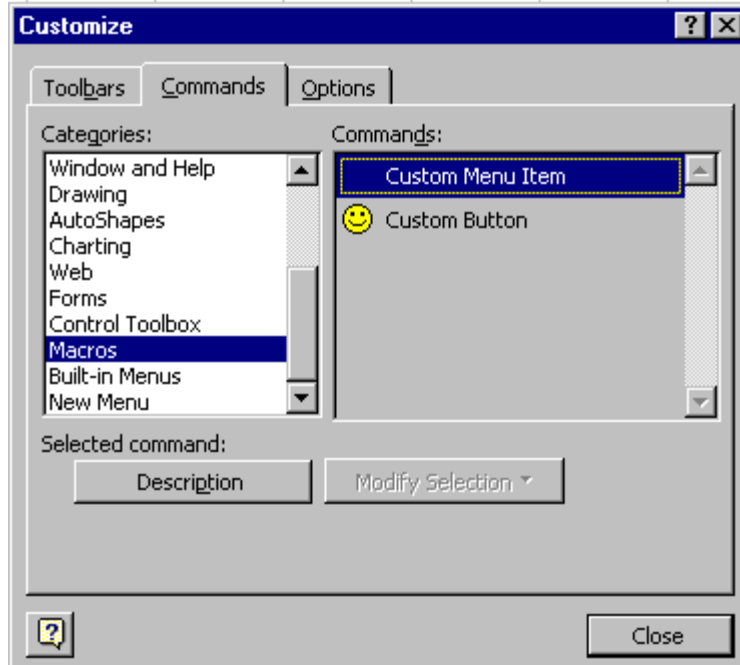


Fig.7 – Caixa de diálogo para atribuir uma macro a um comando do menu

Excel – Macros e Visual Basic for Applications

(versão Draft)

5. Se pretender criar uma nova lista no menu deverá :
- Selecionar a categoria New Menu
 - Na área dos Commands será exibida a opção New Menu, que deverá arrastar até à barra dos menus
 - Poderá alterar o seu nome clicando no botão de Modify Selection
 - Esta nova lista terá o mesmo comportamento que a outras

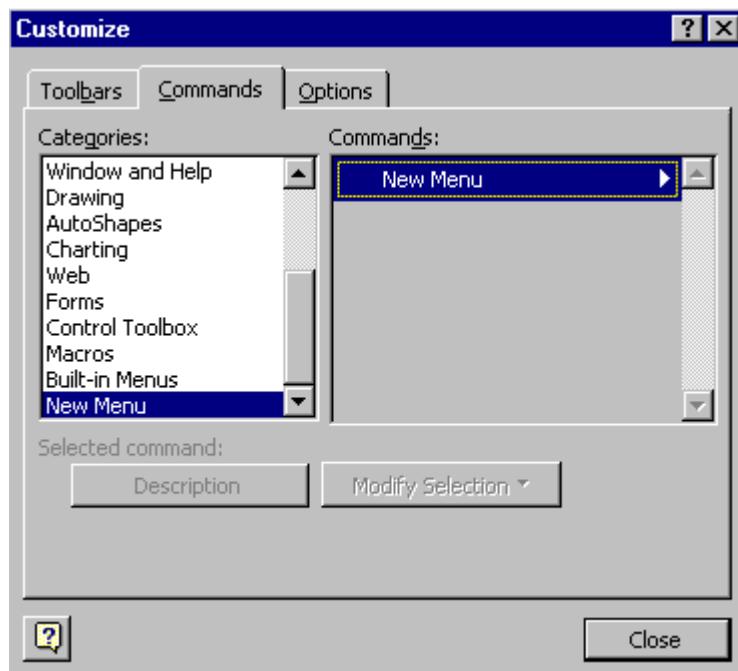


Fig.8 – Caixa de diálogo para criar um novo menu na barra dos menus

6. Na janela de Customize poderá ainda utilizar dois botões que se encontram na área do Selected Command:
- Description – que exhibe um texto explicando o que o comando selecionado faz.
 - Modify Selection – semelhante ao clique sobre o menu ou comando de menu criado, exhibe uma série de tarefas possíveis para configuração (Ver o item 6 do capítulo *Associar uma Macro a um Botão*)

DISSOCIAÇÃO

1. Tools / Customize
2. Arraste o Menu ou Comando do Menu até ao documento e solte-o

Editor de Visual Basic for Applications

1. Tools / Macro / Visual Basic Editor - para acessar ao Editor
2. Posicionando-se no corpo de uma macro, na janela do lado direito inferior, poderá executar a macro através de: Run / Run Sub-UserForms (figura 9) ou botão Run.

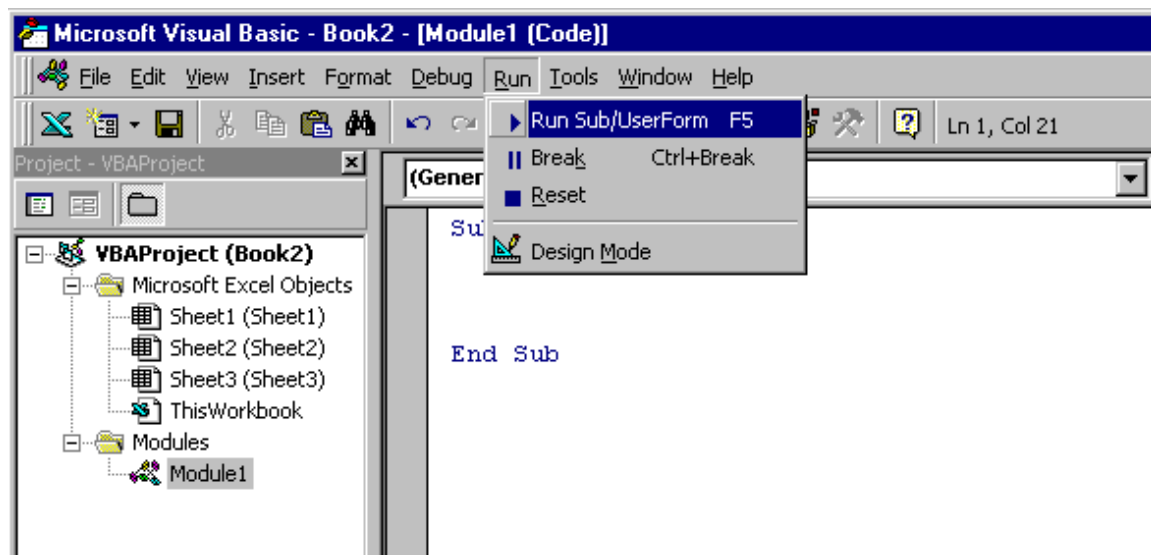


Fig.9 – Execução de uma macro no Editor de Visual Basic for Applications

REMOVER MACROS

A remoção das macros poderá ser feita:

- no ambiente Excel, ou
- no Editor de VBA

Remoção de Macros em Ambiente Excel

1. Tools / Macro / Macros
2. Seleciona-se a Macro a remover
3. Clica-se no botão Delete
4. Pede a confirmação e apaga se confirmar

Remoção de Macros no Editor de VBA

2. Tools / Macro / Visual Basic Editor - para acessar ao Editor
3. Podem-se apagar as macros que se encontrem na janela do lado direito inferior e que têm início com a palavra Sub e fim com as palavras End Sub (Ver capítulo de funções e procedimentos)

Editor de Visual Basic for Applications

Excel – Macros e Visual Basic for Applications (versão Draft)

Para acessar o editor de Visual Basic for Applications: Tools / Macro / Visual Basic Editor

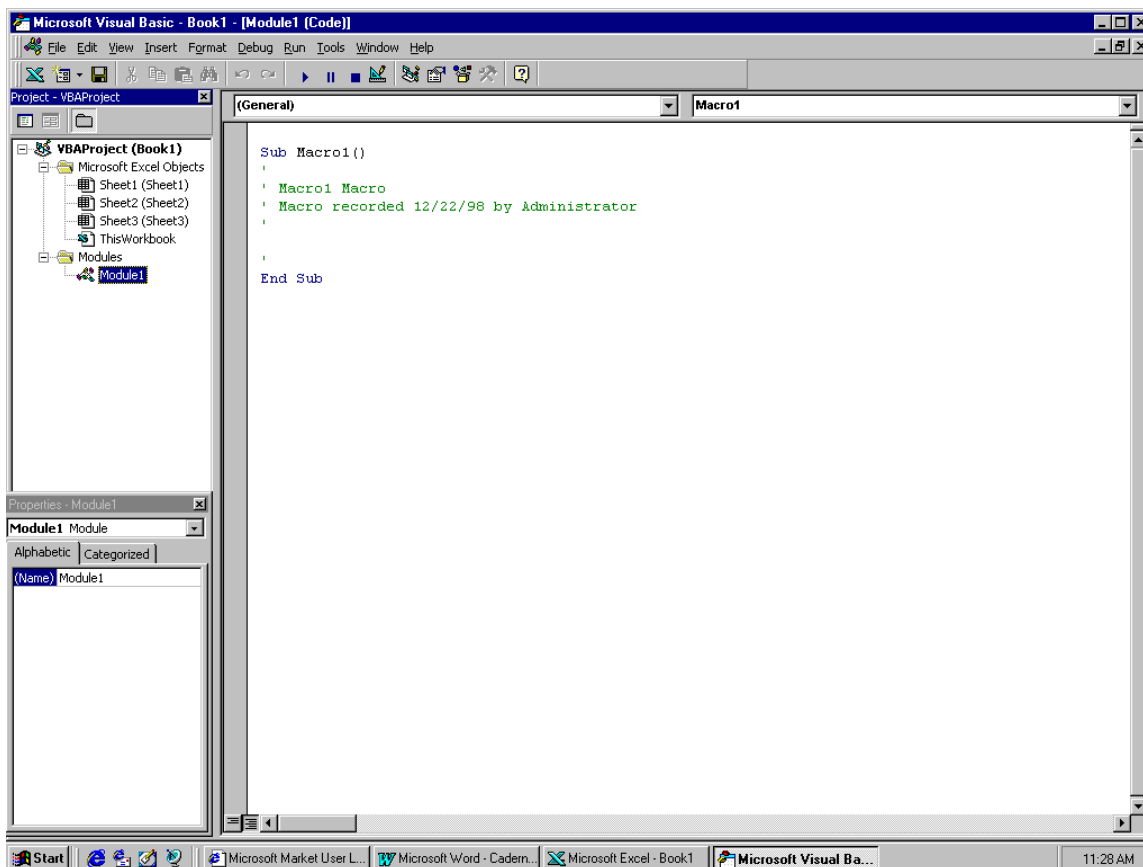


Figura 10 – Editor de Visual Basic for Applications

Encontrará a tela dividida em três grandes áreas: Project Explorer, Properties Window e do lado direito à janela de edição de texto.

PROJECT EXPLORER

Se a janela não estiver visível ative-a através do View / Project Explorer ou utilizando o botão ou combinações de letras associadas a esta tarefa.

Nesta janela poderá visualizar a hierarquia dos projetos de Visual Basic for Applications ativos nesse momento.

Entre eles deverá encontrar um cujo nome corresponde ao nome do Workbook do Excel com que está a trabalhar. E.g. VBAProject (Book1). É dentro deste projeto que deverá trabalhar para que todas as funcionalidades que implemente estejam nele ativas sempre que necessário.

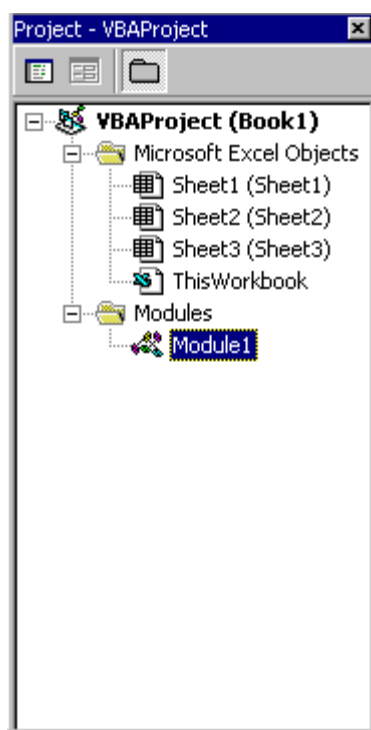


Figura 11 – Janela do Project Explorer

Excel – Macros e Visual Basic for Applications

(versão Draft)

Assim se fizer o desdobramento do seu VBAProject, encontrará uma pasta cuja designação é Microsoft Excel Objects, e uma outra designada Modules (se esta não aparecer significa que o seu projeto ainda não possui qualquer macro implementada. Para criar esta pasta deverá fazer: Insert / Module).

Na pasta do Microsoft Excel Objects, encontrará todos os objetos que fazem parte do seu documento: as WorkSheets e o WorkBook (que no fundo é o conjunto de WorkSheets). Se clicar duplamente em cada um destes objetos, uma nova janela será visualizada na área da direita, em cada uma dessas áreas poderá definir a ação a ser executada em função dos eventos de cada objeto.

Na Pasta Modules, aparecerá o conjunto de pastas (módulos) onde poderá programar as suas macros. Clicando duplamente em cada um dos módulos indicados poderá visualizar as macros, que o compõem, na janela da direita.

PROPERTIES WINDOW

Se a janela das propriedades não estiver visível ative-a através do View / Properties Window ou utilizando o botão ou combinações de letras associadas a esta tarefa.

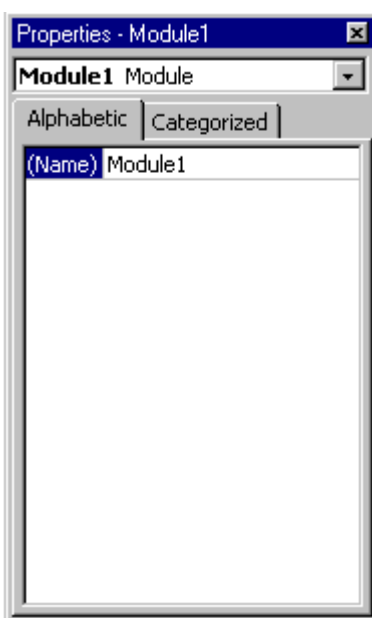


Figura 12 – Janela de propriedades

Nesta janela poderá visualizar e alterar as propriedades que definem cada objeto: o nome - neste caso.

JANELA DE EDIÇÃO

A janela de edição exibirá a cada momento o código em Visual Basic for Applications associado ao elemento selecionado na janela do Project Explorer.

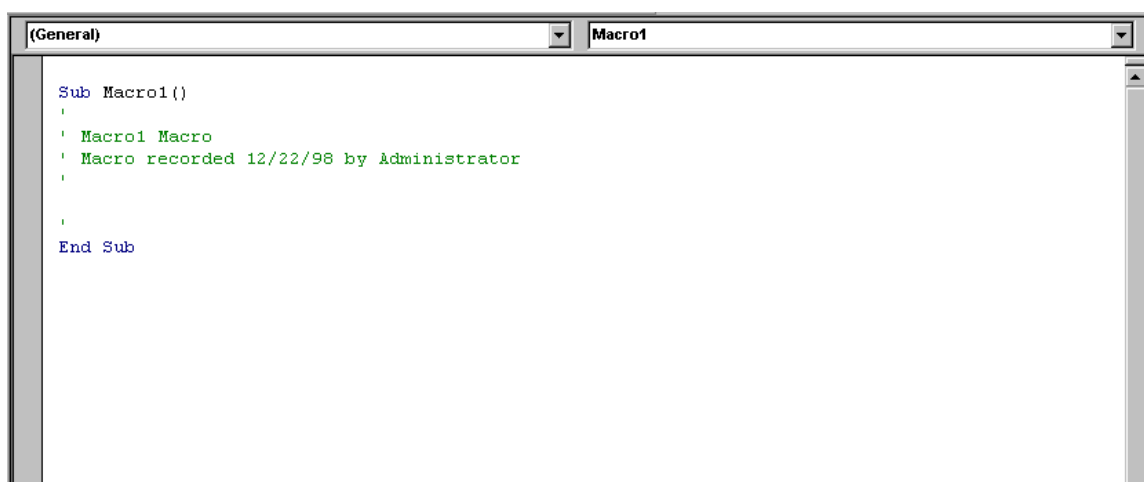


Figura 13 – Janela de edição

OBJECT BROWSER

No Editor de Visual Basic for Applications poderá encontrar ajuda para o desenvolvimento do seu procedimento. Assim:

- Clique no Ícone do Object Browser ou View/Object Browser ou F2
- Na Caixa de Drop-Down onde aparece referido <All Libraries>, selecione a aplicação para a qual pretende ajuda – neste caso Excel. Convém referir que poderá utilizar as funções de toda e qualquer aplicação.
- Na área intitulada por Classes aparecem todos os objetos específicos para o manuseamento da aplicação selecionada. A classe especial designada por Globals refere-se às funções que estão disponíveis na aplicação independentemente dos objetos selecionados.
- Selecione um dos objetos e visualize do lado direito os Members of “ <Elemento selecionado>”
- Para cada membro da classe dispõe de um help on-line que o esclarece sobre a respectiva função e funcionamento, dando um exemplo que poderá testar. Para tal basta clicar sobre o botão de Help da janela do Object Browser.

HELP ON-LINE

O Editor de Visual Basic for Applications, disponibiliza ao usuário um sistema de ajuda constante. Assim, quando se constroem procedimentos, na janela de edição, à medida que a linguagem é escrita o editor abre oportunamente listas de opções para ajudar a escrever o código.

As CORES DO VBA

Na Janela de Edição são programados, armazenados e exibidos os procedimentos VBA criados.

Porém, o texto envolvido em cada procedimento é dotado de uma série de cores diferentes, sendo que cada uma tem significado diferente.

Cor	Significado
Azul	Palavras-Chave da linguagem. Ex: Sub, End Sub, Function, If, Then, Else, While, Loop,...
Vermelho	Sempre que escreve na janela uma linha em linguagem VBA, o editor vai retificar a sintaxe da linguagem por forma a indicar se existe ou não algum erro de escrita. Se for detectado um erro a linha fica a vermelho e é exibida uma mensagem de erro, senão todas as palavras são reconhecidas, coloridas da cor da respectiva categoria e alteradas para letras maiúsculas ou minúsculas, conforme esteja pré-definido.
Preto	Nome de variáveis, procedimentos, valores, operadores,...
Verde	Comentários introduzidos no meio dos procedimentos. Estes comentários servem para o usuário poder associar algumas explicações aos procedimentos realizados. De referir que as palavras com esta cor são ignoradas no procedimento, i.e., não produzem qualquer efeito na sua execução. Para introduzir comentários bastará que o caractere ‘ ‘ anteceda o texto a introduzir.
Amarelo	Um sombreado amarelo poderá aparecer sobre a linha que identifica um procedimento. Esta cor simboliza a ocorrência de um erro na execução do respectivo procedimento e o estado de execução do mesmo, i.e., o procedimento iniciou a execução, durante a qual detectou um erro e agora está parado, mas ainda em execução. Quando isto acontecer não deverá voltar a dar ordem de execução do procedimento, sem antes parar (Stop) a execução mal sucedida.

Funções e Sub Rotinas

Excel – Macros e Visual Basic for Applications

(versão Draft)

Como foi referido anteriormente, quando se grava uma macro no Excel, este tem um comportamento em background que realiza a respectiva codificação para Visual Basic for Applications. Este resultado é bem visível quando procedemos à edição de uma macro no Editor de Visual Basic for Applications. Cada macro que se cria tem um comportamento concreto e autónomo relativamente a outras macros implementadas, e tem como objetivo executar um determinado número de instruções¹ que respondam às exigências do usuário.

Cada macro criada dá origem a um procedimento ou rotina. Existem dois tipos de rotinas:

- as sub-rotinas ou rotinas Sub, e
- as funções.

¹ Entenda-se instrução como uma tarefa a executar que corresponde a uma linha de código.

SUB ROTINAS

Definição de Sub Rotinas

As Sub Rotinas são aquelas cuja definição é delimitada pelas palavras-chave *Sub* e *EndSub*. Assim se reparar todas as macros que grava no Excel são deste tipo. Repare ainda como é que são definidas:

Sub <nome_da_macro> ()

<corpo_da_macro>

End Sub

Estas Sub Rotinas são designadas pelo nome² que lhe atribuímos e não recebem parâmetros³ do exterior, têm como função desempenhar um conjunto de tarefas que compõem o seu corpo. O corpo da macro, é assim composto por um conjunto de instruções, sendo que cada instrução diferente necessita de estar numa linha diferente. Contudo, quando se trata de instruções demasiado grandes o editor faz a sua partição por diversas linhas, recorrendo ao operador “_”, por forma a facilitar a leitura.

² O nome da rotina pode ser qualquer um desde que não contenha espaços, comece por caracteres alfa

³ Para ter a noção do que são parâmetros recorde-se do funcionamento das funções do Excel, que para executarem determinada função necessitavam de receber parâmetros que colocamos entre parêntesis separados por vírgulas.

Trabalhar no Editor de VBA – Criar uma Sub Rotina

Para criar uma Sub Rotina é necessário que exista um módulo onde se possa escrever. Uma vez posicionado nesse módulo poderá:

- Escrever a macro integralmente, ou
- Recorrer ao Insert / Procedure para que o Visual Basic for Applications lhe crie a estrutura (Figura 14)

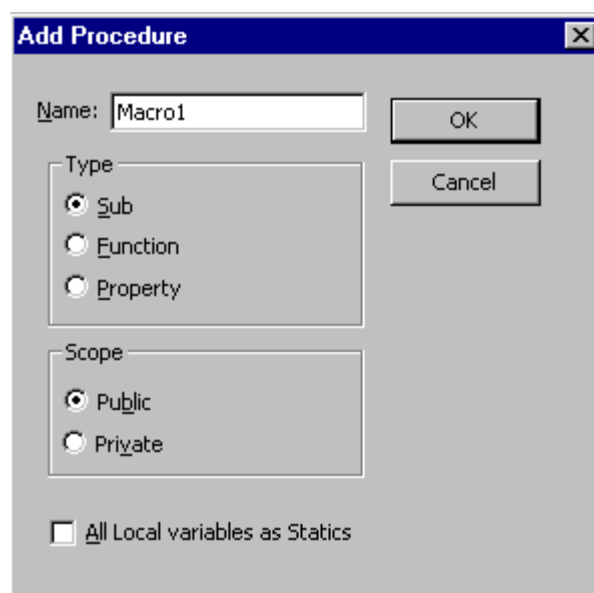


Figura 14 – Caixa de Diálogo para a criação de uma nova rotina

Assim deverá indicar o tipo de rotina a criar – Sub e o nome que pretende dar à macro (ou rotina ou procedimento ou Sub Rotina) ⁴. Automaticamente ele criará a estrutura da rotina, neste caso ficaria:

```
Sub Macro1( )
```

```
End Sub
```

⁴ Nomes alternativos para fazer referência a um conjunto de instruções.

Execução de uma Sub Rotina

Sub Rotina e Macro são duas designações para a mesma realidade, portanto tudo o que foi referido relativamente a Macros é válido também para as Sub Rotinas. (Consultar capítulo das Macros)

Neste contexto a execução de Sub Rotinas segue o mesmo mecanismo das macros. Porém neste momento já se pode referir uma nova forma de executar as macros ou Sub Rotinas – dentro de outras rotinas, i.e., quando se escreve dentro de uma rotina o nome de outra a execução da rotina tem continuidade na execução da rotina que está a ser invocada.

FUNÇÕES

Definição de Funções

Funções são rotinas cuja definição começa com a palavra-chave *Function* e termina com as palavras *End Function*. Todas as funções que utiliza no Excel são deste tipo de rotina. A sua definição tem a estrutura seguinte:

Function <Nome da Função> (<parametro1>, <parametro2>,...)

...

<Nome da Função> = <Valor / Expressão>

...

End Function

A função é identificada pelo nome, pelo número e tipo de parâmetros recebidos, e tem como objetivo executar um conjunto de instruções e produzir um valor final. Isto é, sempre que se pretender executar uma função é sabido à priori que ela produzirá um valor. Recorde-se como exemplo a função SUM, esta recebe por parâmetro um conjunto de valores que se pretendem somar, sabe-se que o resultado da aplicação dessa função ao conjunto de valores será o respectivo somatório.

Para definir o valor produzido por uma função basta no seu interior, atribuir ao nome da função um determinado valor ou expressão.

Definição do tipo de parâmetros e do tipo da função

Todos os elementos de input e output de uma função têm um tipo de dados atribuído. Assim os parâmetros deverão ser definidos com o tipo de dados respectivo e a função deverá ser definida do tipo de dados que ela envie para o exterior.

Após isto poder-se-á refinar a definição de uma função:

Function <Nome da Função> (<parametro1> **As** <Tipo>, ...) **As** <Tipo>

...
<Nome da Função> = <Valor / Expressão>
...

End Function

Nota: Se os tipos não forem definidos será assumido por padrão como sendo do tipo **Variant**

Trabalhar no Editor de VBA – Criar uma Função

Para criar uma Função é necessário que exista um módulo onde se possa escrever. Uma vez posicionado nesse módulo poderá:

- Escrever a macro integralmente, ou
- Recorrer ao Insert / Procedure para que o Visual Basic for Applications lhe crie a estrutura (Figura 15)

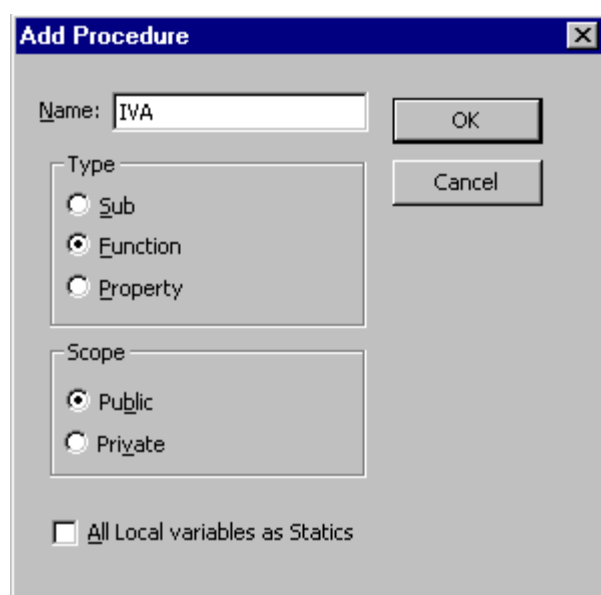


Figura 15 – Caixa de Diálogo para a criação de uma nova rotina - função

Nesta caixa de diálogo deverá indicar o tipo de rotina a criar – Function – e o nome que pretende dar à função. Automaticamente o Visual Basic for Applications criará a estrutura da rotina, que neste caso ficaria:

```
Function IVA( )
```

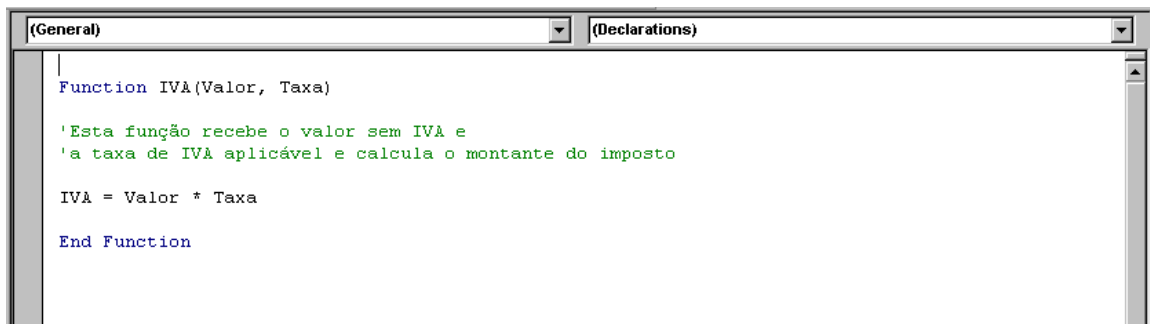
```
End Function
```

Execução de uma Função

Uma vez que uma função produz um valor ela poderá ser executada:

- dentro de uma célula numa WorkSheet, à semelhança de qualquer uma outra função do Excel
- dentro de qualquer outra função ou SubRotina.

Com base na seguinte função, analisemos as formas de execução com mais detalhadamente:



The image shows a screenshot of the Visual Basic Editor window. The window has two tabs at the top: '(General)' and '(Declarations)'. The '(Declarations)' tab is selected. The main area of the window contains the following VBA code:

```
Function IVA(Valor, Taxa)

'Esta função recebe o valor sem IVA e
'a taxa de IVA aplicável e calcula o montante do imposto

IVA = Valor * Taxa

End Function
```

Figura 16 – Janela de Edição com a função IVA

EXECUÇÃO DENTRO DE UMA CÉLULA

1. Posicione-se na célula onde pretende inserir a função
2. Insert / Function
3. Selecione a categoria *User Defined* – repare que aparece listada a função que acabou de criar IVA (Figura 17)

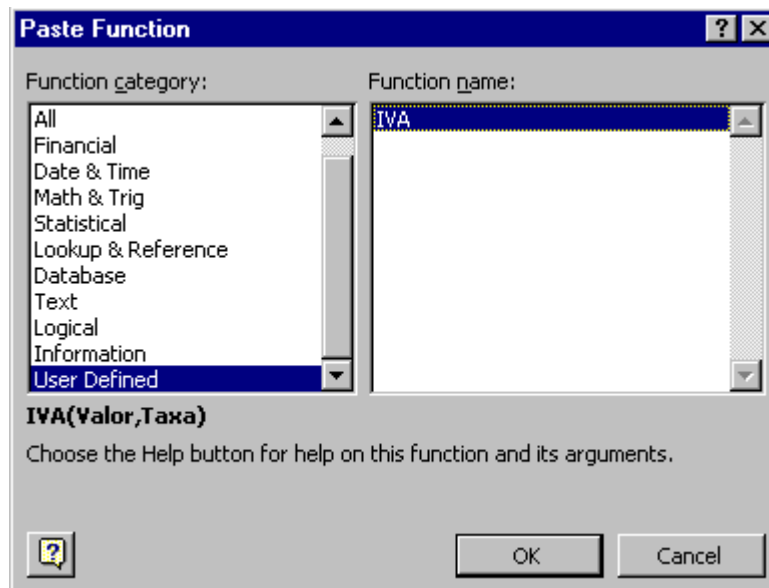


Figura 17 – Caixa de Diálogo para introdução da função

Excel – Macros e Visual Basic for Applications

(versão Draft)

4. Clique em OK e de imediato uma janela de ajuda virá auxiliar a utilização da sua função (Figura 18)

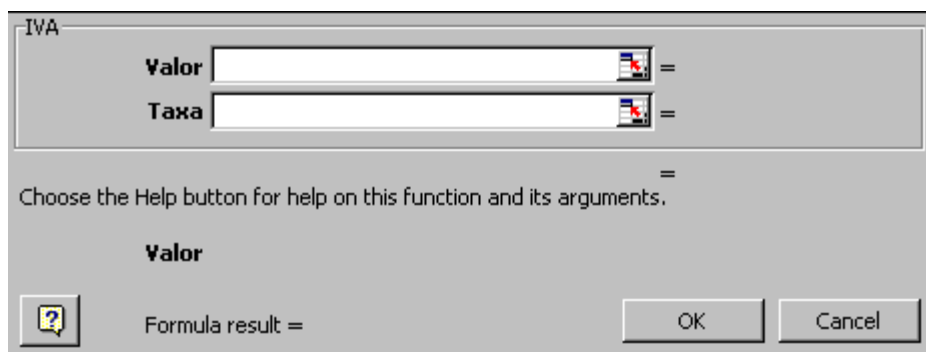


Figura 18 – Caixa de Diálogo para apoio à utilização da função

5. Introduza os parâmetros e clique em OK (Figura 19)

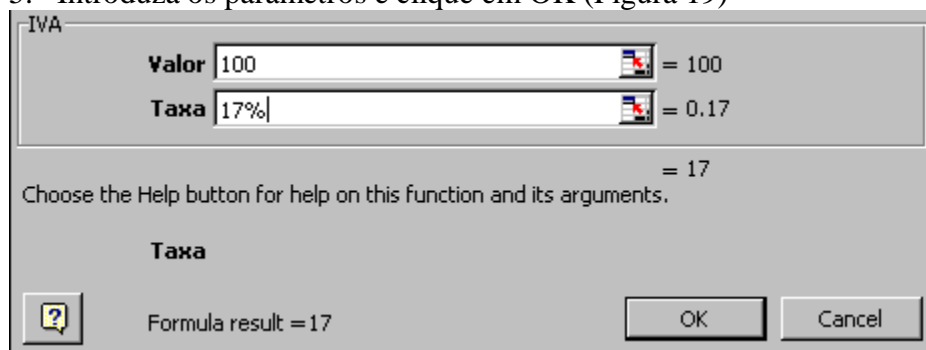


Figura 19 – Caixa de Diálogo para apoio à utilização da função – introdução de valores

Em suma:

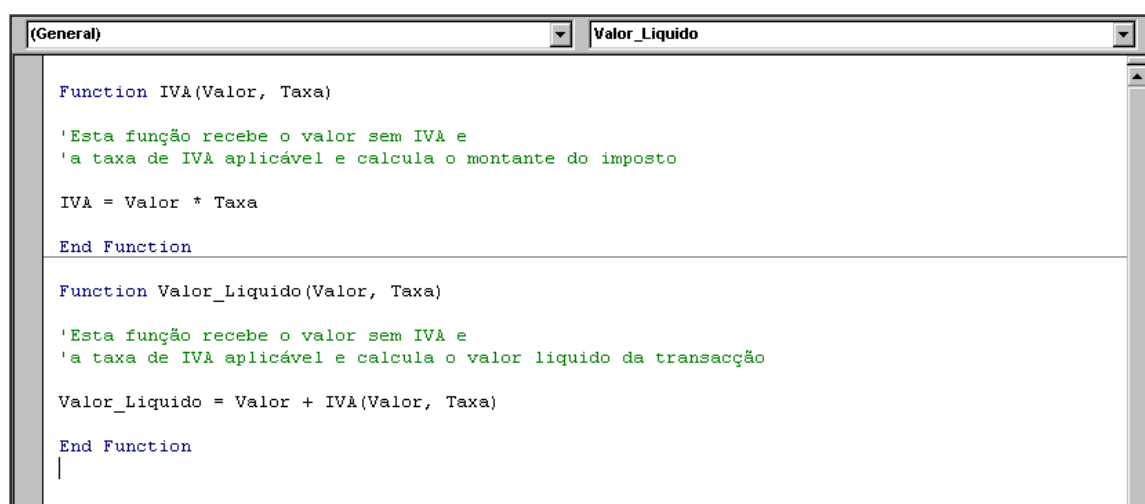
Qualquer função poderá ser chamada a partir da WorkSheet do Excel, sendo que a sua chamada será realizada à semelhança de qualquer outra função: numa célula

= <nome da função> (<Param_1> , <Param_2>)

EXECUÇÃO DENTRO DE UMA ROTINA

Quando dentro de uma rotina se faz referência ao nome de uma outra rotina a execução da primeira passa pela execução daquela que está a ser invocada.

No exemplo que se segue, a função Valor_Liquido faz uma chamada à função IVA por forma a que, o valor por esta produzido, seja adicionado à variável Valor, e assim produzir o output final da função Valor_Liquido.



```
Function IVA(Valor, Taxa)

'Esta função recebe o valor sem IVA e
'a taxa de IVA aplicável e calcula o montante do imposto

IVA = Valor * Taxa

End Function

Function Valor_Liquido(Valor, Taxa)

'Esta função recebe o valor sem IVA e
'a taxa de IVA aplicável e calcula o valor liquido da transacção

Valor_Liquido = Valor + IVA(Valor, Taxa)

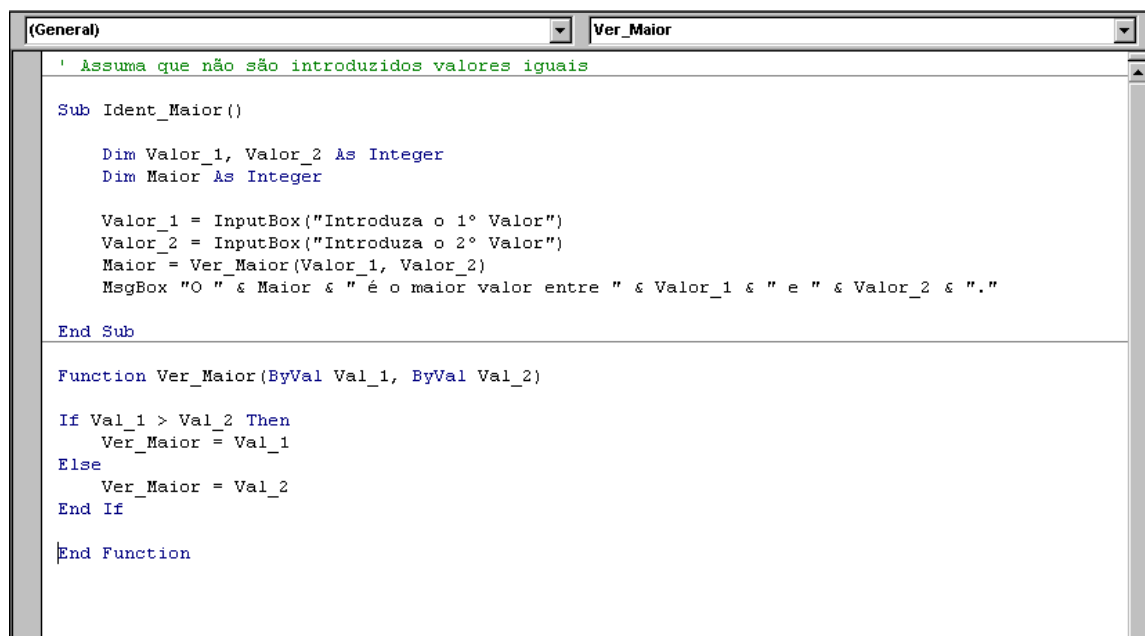
End Function
```

Figura 20 – Função Valor_Liquido ao ser executada dá ordens de execução à função IVA

Excel – Macros e Visual Basic for Applications

(versão Draft)

Um outro exemplo poderá elucidar melhor:



```
(General) Ver_Maior

' Assuma que não são introduzidos valores iguais

Sub Ident_Maior()

    Dim Valor_1, Valor_2 As Integer
    Dim Maior As Integer

    Valor_1 = InputBox("Introduza o 1º Valor")
    Valor_2 = InputBox("Introduza o 2º Valor")
    Maior = Ver_Maior(Valor_1, Valor_2)
    MsgBox "O " & Maior & " é o maior valor entre " & Valor_1 & " e " & Valor_2 & "."

End Sub

Function Ver_Maior(ByVal Val_1, ByVal Val_2)

    If Val_1 > Val_2 Then
        Ver_Maior = Val_1
    Else
        Ver_Maior = Val_2
    End If

End Function
```

Figura 21 – A Subrotina Ident_Maior ao ser executada dá ordens de execução à função Ver_Maior

DIFERENÇAS ENTRE FUNÇÕES E ROTINAS

As funções são similares às sub rotinas, existem simplesmente três diferenças:

1. As Funções Começam com a palavra-chave *Function* e terminam com as palavras *End Function*
2. As Funções podem ser chamadas a partir de fórmulas introduzidas numa WorkSheet
3. As funções retornam valores para as fórmulas ou sub rotinas que as chamarem

REGRAS PARA A PASSAGEM DE PARÂMETROS

Regra 1: Como uma função retorna um valor, esta deverá ser utilizada numa expressão. Quando uma função é utilizada do lado direito de uma associação, ou como argumento de uma outra rotina, se deverão passar os parâmetros dentro de parêntesis

Regra 2: Pode-se chamar uma função ou sub rotina com a palavra-chave Call, neste caso se deverão colocar todos os parâmetros entre parêntesis.

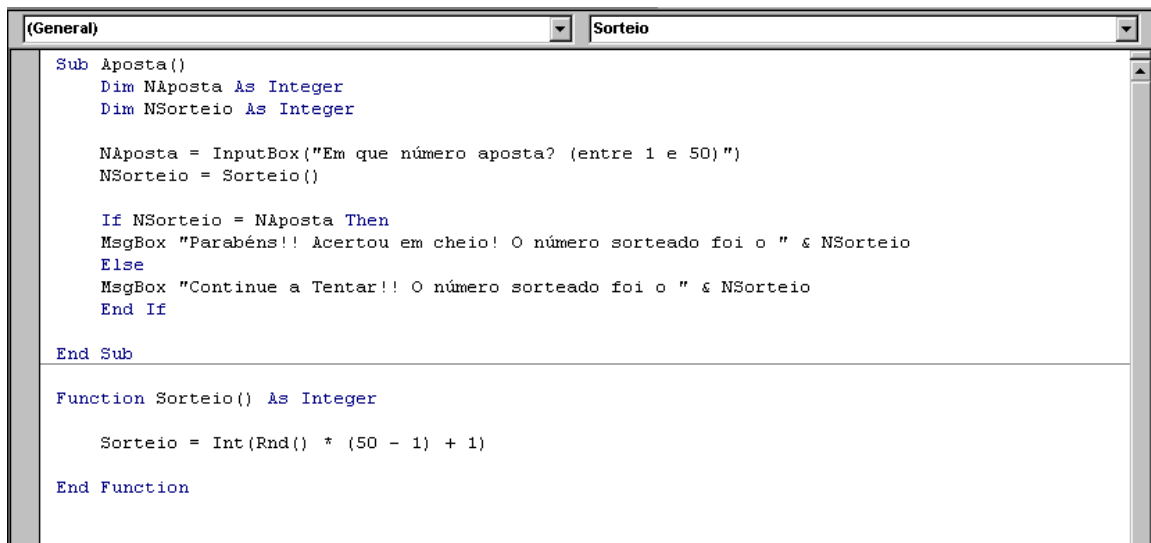
Regra 3: Quando uma rotina é chamada de forma isolada e sem a palavra-chave Call, não se deve utilizar parêntesis.

Variáveis

MANUSEAMENTO COM VARIÁVEIS

O que são variáveis?

As variáveis constituem repositórios temporários de dados, podendo ser utilizadas para diversos fins.



```
(General) | Sorteio  
  
Sub Aposta()  
    Dim Nposta As Integer  
    Dim NSorteio As Integer  
  
    Nposta = InputBox("Em que número aposta? (entre 1 e 50)")  
    NSorteio = Sorteio()  
  
    If NSorteio = Nposta Then  
        MsgBox "Parabéns!! Acertou em cheio! O número sorteado foi o " & NSorteio  
    Else  
        MsgBox "Continue a Tentar!! O número sorteado foi o " & NSorteio  
    End If  
  
End Sub  
  
Function Sorteio() As Integer  
  
    Sorteio = Int(Rnd() * (50 - 1) + 1)  
  
End Function
```

Figura 22 –Manuseamento de Variáveis

Associação de valores a variáveis:

Quando se pretende atribuir valores a variáveis deve-se indicar o nome da variável, o operador "=" e o valor que se pretende que a variável armazene.

<Nome_Variável> = <Valor>

Pela observação do procedimento Adição - Figura 22- podemos verificar que as variáveis *Parcela_1* e *Parcela_2* armazenam os valores introduzidos pelo usuário através das *InputBox* (Ver capítulo "*InputBox e MsgBox*"). Também à variável *Total* vai ser atribuído o valor resultante da adição das duas parcelas - *Total = Parcela_1 + Parcela_2*.

Utilização de variáveis como se fossem valores:

O nome da variável representa o conteúdo da mesma, i.e., sempre que mencionar o nome da variável é o seu conteúdo que será considerado.

No exemplo da figura 22, pode-se constatar que na expressão:

Total = Parcela_1 + Parcela_2

Parcela_1 representa o primeiro valor introduzido e *Parcela_2* representa o segundo valor, não se trata de adicionar o nome de duas variáveis, mas adicionar os conteúdos que elas armazenam.

TIPOS DE VARIÁVEIS

O tipo de variável está associado ao gênero de informação que esta tem hipótese de armazenar

Boolean – 2 bytes – Permite armazenar valores Booleanos – True ou False

Byte – 1 Byte – permite armazenar números sem sinal entre 0 e 255

Currency - 8 bytes – permite armazenar moeda

Date – 8 Bytes – permite armazenar datas

Double – 8 bytes – permite armazenar um real desde -1.79769313486232E308 até -4.94065645841247E-324 para valores negativos, e desde 1.79769313486232E308 até 4.94065645841247E-324 para valores positivos.

Single – 4 bytes – permite armazenar um real desde -3.402823E38 até -1.4011298E-45, para valores negativos e desde 3.402823E38 até 1.4011298E-45, para valores positivos

Integer – 2 bytes - permite armazenar números inteiros entre -32.768 e 32767

Long – 4 bytes – permite armazenar números inteiros entre -2 147 483 648 e 2 147 483 648

Object – 4 bytes – utilizado para fazer referência a um objeto do Excel

String – 1 byte por caractere – permite armazenar conjuntos de caracteres

Variant – 16 bytes - permite armazenar qualquer tipo de dados

User-Defined – permite armazenar valores de tipos diferentes

DECLARAÇÃO DE VARIÁVEIS

Dim I_Numero As Integer

Onde:

Dim – Palavra chave que indica uma declaração de variáveis (abreviatura de dimensão)

I_Numero - nome da variável a utilizar

As – palavra chave utilizada para separar o nome da variável do tipo de dados

Integer – tipo de dados atribuído à variável

É possível na mesma declaração de variáveis declarar variáveis de diversos tipos:

Dim var1 As Integer, var2 As Date, var3 As Double

Para declarar diversas variáveis do mesmo tipo:

Dim var_1, var_2, var_3 As Currency

VARIÁVEIS – VANTAGENS DA UTILIZAÇÃO

- ♦ Simplificam a codificação, principalmente quando se necessita de utilizar um valor específico inúmeras vezes
- ♦ Com variáveis o código é mais rápido

VARIÁVEIS DO TIPO OBJETO

Uma variável objeto representa uma referência a um objeto. Uma variável de extrema importância que facilita a codificação e melhora a performance da sub rotina.

Declaração da Variável Objeto

Dim <Var_Objeto> **As Object**

Atribuição de uma variável Objeto

Set <Var_Objeto> = <Objeto>

Onde:

Set – palavra chave que indica a associação de uma variável objeto

<Var_Objeto> - Variável Objeto

= - Operador de associação

<Objeto> - Objeto a ser atribuído à variável

Utilização Genérica da Variável Objeto

A utilização genérica do tipo Objeto serve para suportar qualquer tipo de objeto Excel (WorkBook, WorkSheet, Range,...)

Exemplo:

```
Dim Range_1 as Object
Range_1 = Worksheet(1).Range("A1")
Range_1.Value = 10
```

Utilização Especifica da Variável Objeto

Utiliza-se o tipo exato do objeto que se pretende atribuir à variável. Esse tipo específico de objetos coincide com o nome dos objetos em EXCEL.

```
Dim Range_1 As Range
Dim WB_1 As Workbook
Dim WS_1 As WorkSheet
Dim XL As Aplicativo
```

Exemplo:

```
Dim Range_1 as Range
Range_1 = Worksheet(1).Range("A1")
Range_1.Value = 10
```

Excel – Macros e Visual Basic for Applications

(versão Draft)

Contudo estas declarações também podem ser feitas da seguinte forma genérica:

```
Dim Range_1 As Object  
Dim WB_1 As Object  
Dim WS_1 As Object  
Dim XL As Object
```

Então qual o interesse de definir as variáveis de forma específica se o podemos fazer de forma genérica?

Por uma questão de performance, se utilizar um objeto genérico, o VBA antes de executar qualquer função com o objeto tem que primeiramente o identificar (perdendo tempo) – em sub rotinas simples essa diferença não é substancial mas quando se trata de grandes sub rotinas já se denotam diferenças significativas.

VARIÁVEIS – DECLARAÇÃO OPCIONAL E O TIPO VARIANT

A declaração de variáveis é opcional, se as variáveis não forem declaradas o VBA faz a sua declaração por padrão. Assim sempre que a instrução do Dim é omitida para uma variável, essa assume o tipo Variant.

Os Prós e Contras da Utilização do tipo Variants

PRÓS

- Diminui o número de linhas de código
- Não é necessário estar preocupado se a variável está ou não declarada porque o VBA automaticamente o faz

CONTRAS

- Aumenta o tempo de execução – o VBA primeiro precisa reconhecer o tipo de dados com os quais está trabalhando.
- Este tipo de dados consome mais memória (uma vez que tem que alocar espaço para qualquer tipo de dados que lhe seja atribuído) – 16 bytes mais um byte por caractere se for String => problemas de performance para sub rotinas grandes.
- Não é possível saber o tipo de dados que uma determinada variável contém – dificultando a detecção de erros.

Variáveis – Declaração Forçada

Para que o VBA detecte um erro sempre que uma variável não seja declarada deverá fazer:

- Tools/Options
- Editor Tab
- Activar Require Variable Declaration

Ou então, escrever no início de cada módulo **Option Explicit**

Neste caso sempre que seja detectada uma variável que ainda não foi declarada dá uma mensagem de erro - ***Variable Not Defined***

VARIÁVEIS – TIPOS DEFINIDOS PELO USUÁRIO - ESTRUTURAS

Definição do Tipo

A primeira fase é constituída pela definição do tipo:

Type Dados_Pessoais

Nome **As String**

Idade **As Integer**

DataNascimento **As Date**

BI **As Long**

End Type

Criou-se um tipo de dados que representa uma estrutura com dados de diferentes tipos. Esta definição deverá ocorrer no início do módulo VBA.

Onde:

Type	Palavra-Chave que indica a definição de um tipo de dados criado pelo usuário.
Dados_Pessoais	Nome atribuído ao tipo de dados.
Nome As String	Primeiro elemento da estrutura de dados definida.
Idade As Integer	Segundo elemento da estrutura de dados definida.
DataNascimento As Date	Terceiro elemento da estrutura de dados definida.
BI As Long	Quarto elemento da estrutura de dados definida.
End Type	Palavra-Chave que indica o fim da definição da estrutura de dados.

Utilização das Estruturas de Dados

Como utilizar as estruturas de dados:

Sub Tipos_definidos_Usuário()

Dim Pessoa **As** Dados_Pessoais

Pessoa.Nome = “Francisco”

Pessoa.DataNascimento = #8/7/73#

Pessoa.Idade = WorksheetFunction.Year(Date)_

– WorksheetFunction.Year(Pessoa.DataNascimento)

Pessoa.BI = 103582915

MsgBox ⁵Pessoa.Nome & Chr(13) & “, Idade “ & Pessoa.Idade & Chr(13) & _
“, Data de Nascimento ” & Pessoa.DataNascimento & Chr(13) & _
“, com o BI número “_ & Pessoa.BI

End Sub

⁵ Ver capítulo *InputBox e MsgBox*

VARIÁVEIS – ARRAYS

O que é um Array ?

Um Array é uma variável que representa um conjunto de variáveis do mesmo tipo.

Os Arrays podem ser multi-dimensionais, onde todas as dimensões são indexadas numericamente.

ARRAY UNI-DIMENSIONAL

	0
	1
	2
	3
	4
	5

Um array uni-dimensional é constituído por uma única lista de elementos indexáveis. Esta lista tem um elemento inicial e um outro final sendo que a cada elemento da lista corresponde um único índice, tradução do lugar que ocupa na lista, que o identifica univocamente.

ARRAY BI-DIMENSIONAL

0	1	2	
			0
			1
			2
			3

Um array bi-dimensional é um pouco mais complexo e é constituído por um conjunto de listas do mesmo comprimento, este formato é normalmente conhecido como array ou matriz. É portanto constituída por linhas e colunas e cada elemento é identificado por um índice composto pela interseção dos números da linha e da coluna.

Declaração de um array

ARRAYS UNI-DIMENSIONAIS

Dim ArrayNumerico(10) As Integer

Array_Numerico é o nome da variável array, o número entre parêntesis indica o número de elementos que o array pode armazenar, isto é

Array_Numerico	
	0
	1
	2
	3
	4
	5
	6
	7
	8
	9

Em que cada elemento é do tipo Integer.

ARRAYS BI-DIMENSIONAIS

Dim Tabela_Textual (5, 4) As String

Tabela_textual é o nome da variável, os número entre parêntesis indicam que a tabela irá ter 5 linhas e 4 colunas, podendo assim armazenar 20 elementos do tipo String.

Tabela_Textual			
	0	1	2

Utilização de um Array

PARA ACESSAR AO ELEMENTO

<Nome_do_Array>(<Indice1_do_Elemento>[,<Indice2_do_Elemento>,...])

ATRIBUIÇÃO DE VALORES

<Nome_do_Array>(<Indice1_do_Elemento>[,<Indice2_do_Elemento>,...]) = <Valor>

Exemplo 1:

Sub Países()

Dim Países(3) **As String**

Países (0) = "Portugal"

Países(1) = "Brasil"

Países(2) = "Moçambique"

MsgBox "Países Armazenados:" & Chr(13) & Países(0) & Chr(13) & Países(1) & Chr(13) & Países(2)

End Sub

Após a atribuição de valores a cada elemento do array, este passa a ter o conteúdo seguinte:

Países

Portugal	0
Brasil	1
Moçambique	2

Valor estes que serão exibidos através da MsgBox.

Exemplo 2:

Option Base 1

Sub Utilizacao_Array()

Dim Lotaria(3) As Integer

Lotaria(1)=int(10000*Rnd())

Lotaria(2)=int(10000*Rnd())

Lotaria(3)=int(10000*Rnd())

MsgBox “Números da lotaria: ” & Lotaria(1) & “, “& Lotaria(2) & “, “& Lotaria(3)

End Sub

Option Base e Array Bounds

A indexação de um array por padrão tem início em 0, sendo que os respectivos índices vão de 0 a **dimensão-1**. Para alterar a base de indexação por forma a ter início em 1 basta colocar no início do módulo **Option Base 1**

Contudo se pretender que um array comece em outro número que não seja 0 ou 1, é necessário especificar os limites inferior e superior do índice quando da declaração do array.

Exemplo:

Sub Utilizacao_Array()

Dim Lotaria(4 To 5) As Integer

Lotaria(4)=int(10000*Rnd())

Lotaria(5)=int(10000*Rnd())

MsgBox “Números da lotaria: ” & Lotaria(4) & “, “& Lotaria(5)

End Sub

Constantes

O QUE SÃO CONSTANTES ?

Constantes são valores que não alteram durante a execução de uma rotina. São declaradas da mesma forma que as variáveis, a única diferença reside no fato da atribuição ser feita na mesma instrução da declaração, e só poder ser feita uma única vez.

Const <Nome_Constante> **As** <Tipo> = <Valor>

Const <Nome_Constante> **As** <Tipo> = <Expressão de cálculo>

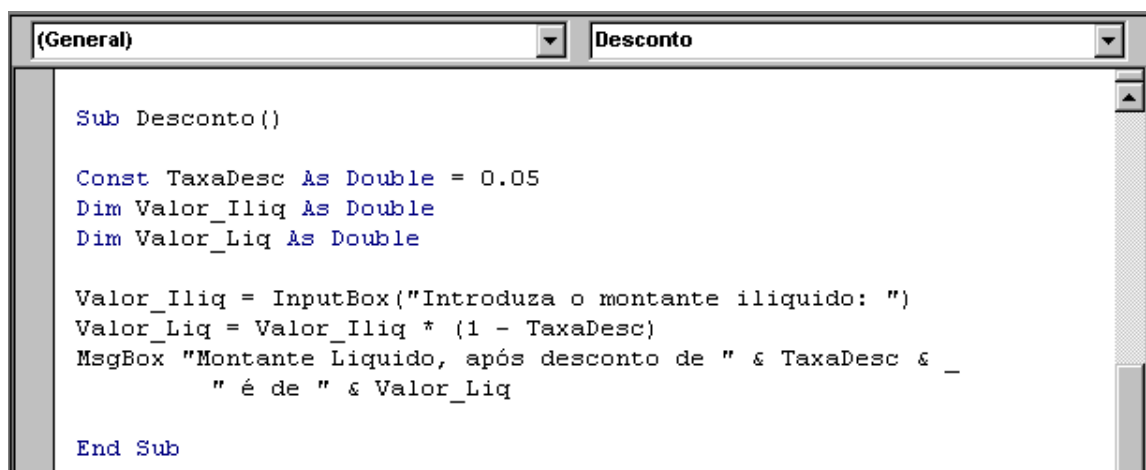


Figura 23 –Manuseamento de Constantes

InputBox e MsgBox

O QUE SÃO ?

Para haver interação entre o usuário e uma macro ou programa é necessário que exista um interface de comunicação. Este poderá ser mais ou menos complexo e completo, contudo existem dois elementos básicos para estabelecer esta ligação: InputBox e MsgBox.

Neste contexto a InputBox é uma função que permite ao usuário introduzir dados no programa – é portanto um mecanismo de input. O MsgBox é um mecanismo de Output e permite ao usuário visualizar os dados produzidos pelo programa.

INPUTBOX

O que faz...

1. Exibe na tela uma janela com uma caixa – text box – para a inserção de dados.
2. Espera que o usuário introduza os dados e/ou acione um dos botões.
3. Como é uma função produz um valor final. Este consiste nos dados inseridos pelo usuário na forma textual - String.

Sintaxe

InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])

Numa primeira avaliação da função, pode-se dizer que a mesma possui diversos parâmetros, mas somente o primeiro é obrigatório, sendo que todos os outros quando ignorados assumem valores atribuídos por padrão.

Excel – Macros e Visual Basic for Applications

(versão Draft)

Parâmetros

Parâmetro	Comentário
Prompt (Obrigatório)	Expressão textual exibida como mensagem na janela de input. A dimensão máxima é de 1024 caracteres. Se pretender construir uma mensagem com mais do que uma linha poderá utilizar o caractere Enter - Chr(13). A junção dos elementos que constituem a mensagem é realizada através do operador &. Exemplo: "A Soma de 3 com 5 é :" & Chr(13) & " 8 "
Title (Facultativo)	Titulo da janela de input. Se este for omitido, aparecerá por padrão o nome da aplicação.
Default (Facultativo)	Expressão inserida por padrão na caixa de inserção de dados e constituirá a resposta por padrão se o usuário não introduzir outra. Se este parâmetro for omitido aparecerá uma text box vazia.
Xpos (Facultativo)	Número que identifica a distância horizontal entre o lado esquerdo da tela e a janela de input. Se este valor for omitido a janela aparecerá centrada horizontalmente.
Ypos (Facultativo)	Número que identifica a distância vertical entre o lado superior da tela e a janela de input. Se este valor for omitido a janela ficará posicionada a 1/3 da parte inferior da tela
HelpFile (Facultativo)	Nome do arquivo de Help que será utilizado para dar apoio ao preenchimento desta janela. Se for indicado este parâmetro o seguinte é obrigatório.
Context (Facultativo)	Número do índice do tópico de Help constante no arquivo HelpFile, e que corresponde à janela em questão.

Atenção: Se pretender enviar mais que um parâmetro optativo respeite a ordem através de “,” (virgulas)

Exemplo:

InputBox(“Introduza o Nome da Aplicação:”, , “Excel”)

Na janela de input será exibida a mensagem “Introduza o Nome da Aplicação:”, o título da caixa será o definido por padrão e o valor na caixa de inserção será “Excel”.

MsgBox

O que faz...

1. Exibe na tela uma janela com uma mensagem.
2. Espera que o usuário acione um dos botões.
3. Como é uma função produz um valor final. Devolve um número inteiro indicando o botão que foi clicado.

Sintaxe

MsgBox(prompt[, buttons] [, title] [, helpfile, context])

À semelhança da InputBox , pode-se dizer que a mesma possui diversos parâmetros, mas somente o primeiro é obrigatório, sendo que todos os outros quando ignorados assumem valores atribuídos por padrão.

Parâmetros

Parâmetro	Comentário
Prompt (Obrigatório)	Expressão textual exibida como mensagem na janela de input. A dimensão máxima é de 1024 caracteres. Se pretender construir uma mensagem com mais do que uma linha poderá utilizar o caractere Enter Chr(13) Exemplo: "A Soma de 3 com 5 é :" & Chr(13) & " 8 "
Buttons (Facultativo)	Número que identifica o tipo de botões que se pretende visualizar na janela de output. Ver tabela seguinte. Se for omitido assumirá o valor 0 por padrão.
Title (Facultativo)	Titulo da janela de input. Se este for omitido, aparecerá por padrão o nome da aplicação.
HelpFile (Facultativo)	Nome do arquivo de Help que será utilizado para dar apoio ao preenchimento desta janela. Se for indicado este parâmetro o seguinte é obrigatório.
Context (Facultativo)	Número do índice do tópico de Help constante no arquivo HelpFile, e que corresponde à janela em questão.

Atenção: Se pretender enviar mais que um parâmetro optativo respeite a ordem através de “,” (virgulas)

Exemplo:

MsgBox(“Erro de Sintaxe !!!” , “Mensagem de Erro”)

Na janela de output será exibida a mensagem “Erro de Sintaxe”, o botão exibido será o de OK (padrão) e o titulo da janela será “Mensagem de Erro”.

Excel – Macros e Visual Basic for Applications

(versão Draft)

Constante de VBA	Valor	Descrição
VbOKOnly	0	Exibe somente o botão de OK.
VbOKCancel	1	Exibe os botões OK e Cancel.
VbAbortRetryIgnore	2	Exibe os botões Abort, Retry, e Ignore.
VbYesNoCancel	3	Exibe os botões Yes, No, e Cancel .
VbYesNo	4	Exibe os botões Yes e No.
VbRetryCancel	5	Exibe os botões Retry e Cancel.
VbCritical	16	Exibe o ícone de Critical Message.
VbQuestion	32	Exibe o ícone de Warning Query.
VbExclamation	48	Exibe o ícone de Warning Message.
VbInformation	64	Exibe o ícone de Information Message.
VbDefaultButton1	0	O primeiro botão é o selecionado por padrão.
VbDefaultButton2	256	O segundo botão é o selecionado por padrão.
VbDefaultButton3	512	O terceiro botão é o selecionado por padrão.
VbDefaultButton4	768	O quarto botão é o selecionado por padrão.
VbApplicationModal	0	Application modal – o usuário só poderá dar continuidade ao trabalho depois de responder à MsgBox na aplicação corrente.
VbSystemModal	4096	System modal - o usuário só poderá dar continuidade ao trabalho depois de responder à MsgBox em qualquer aplicação em curso no sistema.

Pela análise desta tabela pode-se constatar que existem diferentes agrupamentos de códigos: para definir o tipo de botões (0-5), para definir o tipo de ícones (16,32,48,64), para definir o botão selecionado por padrão (0,256,512,768) e para indicar o modo de execução (0 e 4096). Pode-se adicionar os códigos e assim fazer combinações entre diversas opções destes 4 grupos, contudo nunca se deve adicionar mais do que um código por agrupamento.

Exemplos:

Para a instrução:

MsgBox "Erro de Sintaxe!!!", 2 + 48 + 512 + 4096, "Mensagem de Erro"

É exibida a seguinte janela:

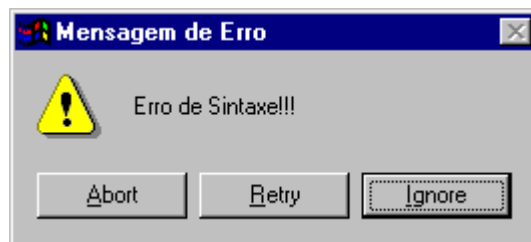


Figura 24 –MsgBox

Para a instrução:

MsgBox "Erro de Sintaxe!!!", 5 + 64 + 256 + 4096, "Mensagem de Erro"

É exibida a seguinte janela:



Figura 25 –MsgBox

Valores Produzidos...

Como já foi referido, a função MsgBox produz um valor em função do botão acionado, assim produzirá como output um dos valores constantes da tabela seguinte:

Constante de VBA	Valor	Botão Accionado
vbOK	1	OK
vbCancel	2	Cancel
vbAbort	3	Abort
vbRetry	4	Retry
vbIgnore	5	Ignore
vbYes	6	Yes
vbNo	7	No

Domínio das variáveis, constantes e rotinas

O QUE É O DOMÍNIO?

Scope ou domínio de um elemento refere-se à área na aplicação na qual esse mesmo elemento pode ser acessado e utilizado, ou seja onde é que o elemento é reconhecido.

DOMÍNIO DAS VARIÁVEIS

Refere-se à área onde a variável permanece ativa, mantendo o valor que lhe vai sendo atribuído.

Existem 3 níveis de domínio para as variáveis:

- Âmbito do Procedimento ou Procedimental
- Âmbito do Modulo ou Modular
- Âmbito do Projeto

As declarações de tipos User Defined que são escritas no Declarations do módulo têm domínio de Projeto.

Âmbito do Procedimento

Estas variáveis são declaradas no corpo do procedimento com recurso à palavra chave **Dim**.

São variáveis criadas quando da execução do procedimento e automaticamente destruídas quando o procedimento termina, sendo que **só são reconhecidas durante o procedimento que as declarou**. Assim sendo, qualquer tentativa realizada por um procedimento no sentido de trabalhar com variáveis definidas no corpo de um outro procedimento não terá êxito.

Exemplo

```
Sub Ambito_Procedimento()  
    Dim Var_1 As Integer  
    Var_1 = InputBox (“Introduza um número Inteiro”)  
    MsgBox “Foi este o número que introduziu: ” & Var_1  
    Ambito_Procedimento_2  
End Sub  
  
Sub Ambito_Procedimento_2()  
    MsgBox “Foi este o número que introduziu: ” & Var_1  
End Sub
```

A execução destas rotinas resultará no seguinte:

A **primeira mensagem** (MsgBox no Procedimento Ambito_Procedimento)exibirá o número introduzido pelo usuário que fora armazenado na variável Var_1.

A **segunda mensagem**, oriunda da instrução MsgBox do Procedimento Ambito_Procedimento_2 não exibirá o valor introduzido, porque a variável Var_1 nele referida é considerada como uma variável interna do processo, e portanto criada dentro do procedimento Ambito_Procedimento_2. Como tal, embora possua o mesmo nome que a variável da rotina Ambito_Procedimento_1 não existe qualquer ligação entre elas, são duas variáveis distintas.

Neste caso se pretendesse que o valor da primeira variável fosse reconhecido no procedimento chamado, teria de o passar por parâmetro.

Âmbito do Módulo

Uma variável como Âmbito do Módulo pode ser acessada por **todas as rotinas existentes no módulo** onde a variável é declarada.

Para declarar uma variável do nível modular, deverá declará-la com a palavra chave **Dim** na seção **Declarations do respectivo módulo VBA**. Neste caso qualquer variável aí declarada será considerada de nível modular por padrão, para tornar esse fato mais explícito poder-se utilizar a palavra chave **Private** na declaração:

Ex:
Private Var_2 As String

Exemplo:

Dim Var_1 As Integer ‘ ou *Private Var_1 As Integer*

```
Sub Ambito_Modulo()  
    Var_1 = InputBox (“Introduza um número Inteiro”)  
    Ambito_Modulo_2  
End Sub
```

```
Sub Ambito_Modulo_2()  
    MsgBox “Foi este o número que introduziu: ” & Var_1  
End Sub
```

Neste caso como a variável é reconhecida no módulo por qualquer rotina nele existente, desde que seja referida sempre pelo mesmo nome, o procedimento Ambito_Modulo_2 irá exibir o valor introduzido pelo usuário no procedimento Ambito_Modulo.

Âmbito do Projeto

As variáveis assim definidas têm o âmbito correspondente a todo o projeto, isto é podem ser **acessadas e alteradas em todos e quaisquer módulos**. Para declarar variáveis deste tipo deve-se fazê-lo na seção **Declarations de qualquer módulo**, para tal deverá utilizar a palavra chave **Public**.

Exemplo:

No Module_1 tem-se:

```
Public Var_1 As Integer
```

```
Sub Ambito_Projecto()
```

```
    Var_1 = InputBox (“Introduza um número Inteiro”)
```

```
    Ambito_Projecto _2
```

```
End Sub
```

No Module_2 tem-se:

```
Sub Ambito_Projecto _2()
```

```
    MsgBox “Foi este o número que introduziu: ” & Var_1
```

```
End Sub
```

A execução do procedimento Ambito_Projecto no Module_1 mandou executar o procedimento Ambito_Projecto do Module_2, e o valor atribuído à variável Var_1 foi acedido posteriormente noutro procedimento de outro módulo, dado tratar-se de uma variável global.

DOMÍNIO DAS CONSTANTES

À semelhança das variáveis também as constantes têm 3 níveis de domínio:

- Âmbito do Procedimento ou Procedimental
- Âmbito do Modulo ou Modular
- Âmbito do Projeto

Âmbito do Procedimento

Estas constantes são **declaradas no corpo do procedimento** com recurso à palavra chave Const. Só **têm existência dentro do procedimento** onde são declaradas.

Exemplo:

```
Sub Ambito_Procedimento()
```

```
    Const Taxa_Desc As Single = 0.05
```

```
    Dim Desconto As Double
```

```
    Desconto = InputBox (“Introduza o montante das Compras”) * Taxa_Desc
```

```
    MsgBox “O desconto é de : ” & Desconto
```

```
End Sub
```

Âmbito do Módulo

Uma constante com Âmbito do Módulo pode **ser utilizada por todas as rotinas existentes no módulo onde é definida.**

Para declarar uma constante a nível modular, deverá declará-la com a palavra chave **Const** na **secção Declarations do respectivo módulo VBA**. Neste caso qualquer constante aí declarada será considerada de nível modular por padrão, para tornar esse fato mais explícito poder-se utilizar a palavra chave **Private** na declaração:

Ex:
Private Const Const_1 As String

Âmbito do Projeto

As constantes assim definidas têm o **âmbito correspondente a todo o projeto**, isto é podem ser utilizadas em todo e qualquer módulo. Para definir constantes deste tipo deverá fazê-lo na **secção Declarations de qualquer módulo**, para tal deverá utilizar a palavra chave **Public**.

Ex:
Public Const Const_1 As String

DOMÍNIO DE SUB ROTINAS E FUNÇÕES

Estas só têm dois níveis de Scope: **o nível do projeto e o nível do módulo.**

Por padrão as rotinas são de âmbito do projeto sem qualquer indicação adicional. Contudo poderá tornar este fato mais explícito pela introdução da Palavra-Chave Public, que em termos operacionais não adicionará qualquer funcionalidade, mas em termos de leitura dará uma maior percepção.

Exemplo:

```
Public Sub Ambito_Procedimento()  
    Const Taxa_Desc As Single = 0.05  
    Dim Desconto As Double  
  
    Desconto = InputBox (“Introduza o montante das Compras”) * Taxa_Desc  
    MsgBox “O desconto é de : ” & Desconto  
End Sub
```

Para que uma rotina tenha o âmbito do módulo onde está definida, deverá ser antecedida pela palavra chave Private.

Estruturas de Controle

O QUE SÃO ESTRUTURAS DE CONTROLE?

O VBA disponibiliza algumas estruturas que pode utilizar para controlar o decurso da execução da rotina. Estas estruturas dão ao programador um poder enorme para construir rotinas bastante complexas e flexíveis.

QUAIS AS ESTRUTURAS...

VBA Control	
If - Then - Else	Testa uma condição e executa um determinado conjunto de instruções consoante o resultado dessa avaliação
For – Next	Executa uma determinada tarefa um determinado número de vezes.
While-Wend	Executa uma determinada tarefa enquanto que uma determinada condição permaneça verdadeira, i.e., com o valor True.
Do – Loop	Executa uma determinada tarefa enquanto que a avaliação de uma condição permaneça True ou então até que seja True.
Select - Case	Seleciona um dos segmentos de código a processar mediante a avaliação consecutiva de condições.
For – Each – Next	Realiza uma determinada tarefa repetitiva em cada objeto de uma coleção ou em cada item de um array.

IF-THEN-ELSE

Função IF do Excel

Recorrendo à função IF do Excel, recorde:

=IF(<condição>, <se condição verdadeira>, <se condição falsa>)

A função IF tinha o seguinte comportamento:

1. Avalia a condição, que deverá ser uma expressão booleana colocada como primeiro parâmetro;
2. Se a condição for verdadeira, então (**then**) realiza as operações colocadas no segundo parâmetro;
3. Caso contrário (**else**), realiza as operações que formam o terceiro parâmetro

A estrutura IF do VBA tem o mesmo tipo de funcionamento, o que difere é a sintaxe.

Sintaxe da Estrutura If-Then-Else

```
If <Condição> Then  
    <se condição verdadeira>  
[ Else  
    <se condição falsa> ]  
End If
```

A palavra Else é opcional num If-Then-Else Statement, sendo que no caso de ser omitida, a avaliação negativa da condição implica uma saída automática da Instrução If.

Aplicação Prática

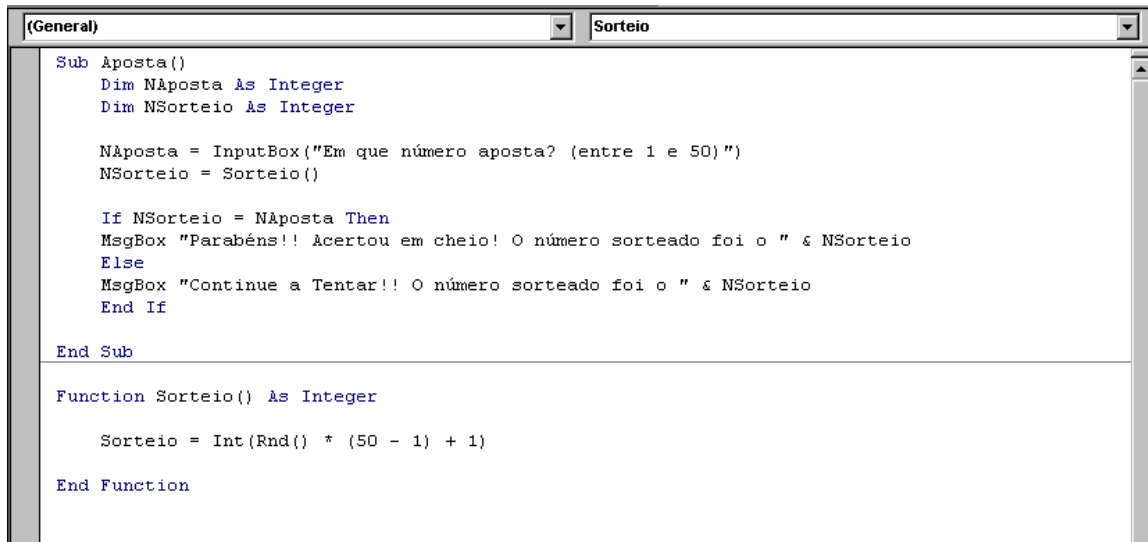


Figura 26 – Aplicação Prática com um IF

A rotina Aposta recebe uma aposta do usuário e mediante o Sorteio a realizar pela respectiva função, verifica se o jogador ganhou ou não a aposta, comunicando-lhe esse fato.

Excel – Macros e Visual Basic for Applications

(versão Draft)

Para uma maior clarificação do funcionamento do IF, atenda ao quadro seguinte:

If	Palavra chave que indica o início de uma instrução If-Then-Else
Nsorteio = NAposta	A condição a testar. Serve para determinar a sequência a dar à execução da rotina. Da avaliação desta condição pode-se obter um de dois valores True ou False, sendo que depende deste resultado o caminho a prosseguir. Se for True executará as instruções que seguirem a palavra-chave Then até encontrar a palavra chave Else, não executando mais nada dentro do IF, caso contrário executará o código que se seguir à palavra chave Else até ao End If.
Then	Palavra chave que determina o fim da condição teste. Todas as instruções que têm início nesta palavra-chave até à palavra-chave Else serão executadas se a condição for verdadeira.
MsgBox “Parabéns!! Acertou em cheio! O número sorteado foi o ” &NSorteio	Instruções a executar se a condição for verdadeira.
Else	Palavra-chave que determina o término de execução das instruções quando o resultado da avaliação for True, e que determina o início das instruções a executar se o resultado da condição for False.
MsgBox “Continue a Tentar!! O número sorteado foi o ” &NSorteio	Instruções a executar se a condição for falsa.
End If	Palavra-chave que indica o fim do controlo de If-Then-Else e como tal onde se deve retomar as instruções para prosseguir a execução do procedimento.

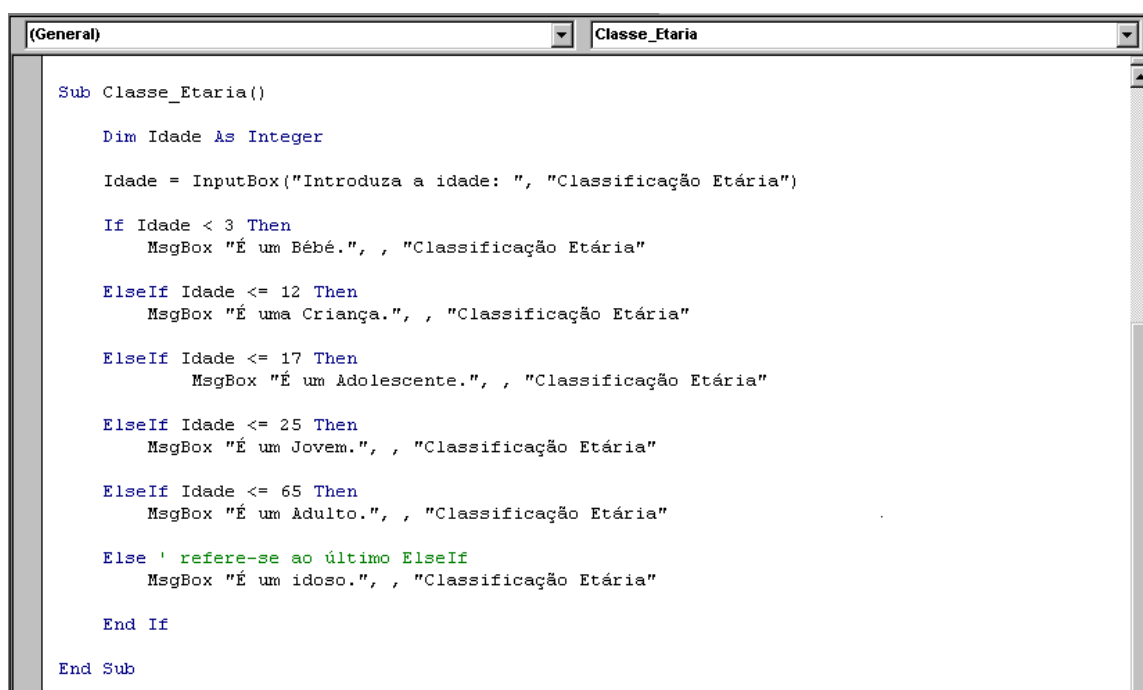
A instrução adicional ElseIf

Esta instrução propõe uma condição alternativa se o teste da condição anterior tiver tido um resultado negativo.

APLICAÇÃO PRÁTICA

Pretende-se criar uma macro que classifique um indivíduo por faixa etária em função da sua idade. A classificação pretendida é a seguinte:

Idade	Classe Etária
Menos de 3 anos	Bebé
Dos 3 aos 12	Criança
Dos 13 aos 17	Adolescente
Dos 18 aos 25	Jovem
Dos 26 aos 65	Adulto
Mais de 65	Idoso



```
(General) Classe_Etaria

Sub Classe_Etaria()

    Dim Idade As Integer

    Idade = InputBox("Introduza a idade: ", "Classificação Etária")

    If Idade < 3 Then
        MsgBox "É um Bébé.", , "Classificação Etária"
    ElseIf Idade <= 12 Then
        MsgBox "É uma Criança.", , "Classificação Etária"
    ElseIf Idade <= 17 Then
        MsgBox "É um Adolescente.", , "Classificação Etária"
    ElseIf Idade <= 25 Then
        MsgBox "É um Jovem.", , "Classificação Etária"
    ElseIf Idade <= 65 Then
        MsgBox "É um Adulto.", , "Classificação Etária"
    Else ' refere-se ao último ElseIf
        MsgBox "É um idoso.", , "Classificação Etária"
    End If

End Sub
```

Figura 27 – Aplicação Prática com ElseIf

FOR – NEXT

Permite a execução de uma tarefa durante um determinado número de vezes.

Sintaxe

For <Inicialização do Contador> **To** <Valor > [**Step** <Valor a Incrementar>]
 <Instruções a realizar em cada iteração>
Next

Aplicação Prática

Pretende-se criar uma rotina que recebendo a base e a potência calcule o valor respectivo.

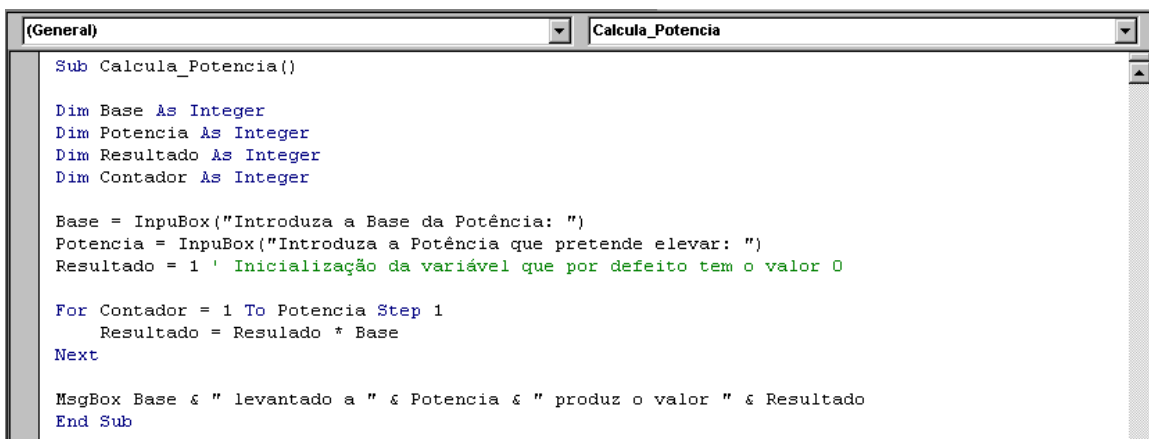


Figura 28 – Procedimento que calcula a potência de um número

A instrução For-Next tem como função calcular a potência. O mesmo efeito poderia ser obtido recorrendo à expressão $\text{Resultado} = \text{Base} ^ \text{Potência}$, contudo para fins de demonstração de funcionamento este exemplo é bastante simples.

A FUNÇÃO DAS VARIÁVEIS:

Variável	Função
Base	Elemento a elevar.
Potência	Número de vezes a multiplicar a base.
Contador	Conta o número de vezes que a base já foi multiplicada, é uma variável que será automaticamente incrementada em cada looping do ciclo.
Resultado	Variável que armazena o resultado sucessivo por cada vez que se multiplica.

CONSTRUÇÃO DO CICLO:

For	Palavra-chave que indica o início do ciclo For-Next
Contador = 1 To Potência	<p>Expressão que contém a inicialização do contador, indicando o valor de início e de fim. Assim, o número de vezes que o ciclo é executado será: Valor_Fim – Valor_Início + 1.</p> <p>A palavra To é utilizada para separar o valor do contador no momento inicial do valor no momento final. (Tradução: O contador iniciar-se-á a um e atingirá o valor máximo traduzido pela variável Potência)</p>
Step 1	Palavra chave utilizada para especificar o valor a incrementar ou decrementar ao contador do ciclo por cada vez que o loop é concretizado. Normalmente o valor a incrementar é um, contudo qualquer outro valor pode ser utilizado, desde números decimais, a números negativos (Provocando assim a decrementação). A palavra-chave Step é opcional, sempre que for omitida é assumido 1 como o valor a incrementar por padrão.
Resultado = Resultado * Base	Instrução a realizar de cada vez que o ciclo for executado. Neste caso a instrução é única, contudo poder-se-ão adicionar outras instruções.
Next	Palavra Chave que indica o fim de um ciclo For-Next . Sempre que a execução do ciclo chega à instrução Next incrementa a variável contador e volta ao início do ciclo.

TRADUÇÃO INTEGRAL

For Contador = 1 To Potência Step 1
*Resultado = Resultado * Base*
Next

Para o número de vezes, a iniciar em 1 até que atinja, o valor Potência, pelo incremento de 1 na execução de cada ciclo, deverá multiplicar sucessivamente o resultado acumulado, pela base.

Funcionamento do Ciclo:

A **primeira execução do ciclo** distingue-se das restantes por a ela estar associada a inicialização do contador, sendo o restante procedimento semelhante a qualquer outra execução.

No **início de cada execução do ciclo**, a variável contador é comparada com o valor final de execução. **Se o Step for um valor positivo** (incrementar) e o valor do contador for superior ao valor final significa que o ciclo já foi realizado o número de vezes pretendido, e então o código acabará a execução da instrução For-Next e seguirá na linha de código que esteja situada imediatamente a seguir, caso contrário executa uma vez mais o ciclo e incrementa a variável contador. Por outro lado, **se o Step contiver valor negativo** (subtrair) e o valor do contador for inferior ao valor final significa que o ciclo já foi realizado o número de vezes pretendido, e então o código acabará a execução da instrução For-Next e seguirá na linha de código que esteja situada imediatamente a seguir, caso contrário executa uma vez mais o ciclo e subtrai a variável contador.

Perigos associados à utilização do ciclo For-Next:

- Não definir o limite de execução (não atribuir valor à variável que o define)
 - Definir erroneamente o Step, por forma a que nunca seja obtido o valor que determina o fim da execução
- ⇒ estas condições implicarão que o ciclo não tenha fim – **Ciclos Infinitos**

Outra Aplicação

Pretende-se criar uma rotina para calcular um fatorial.

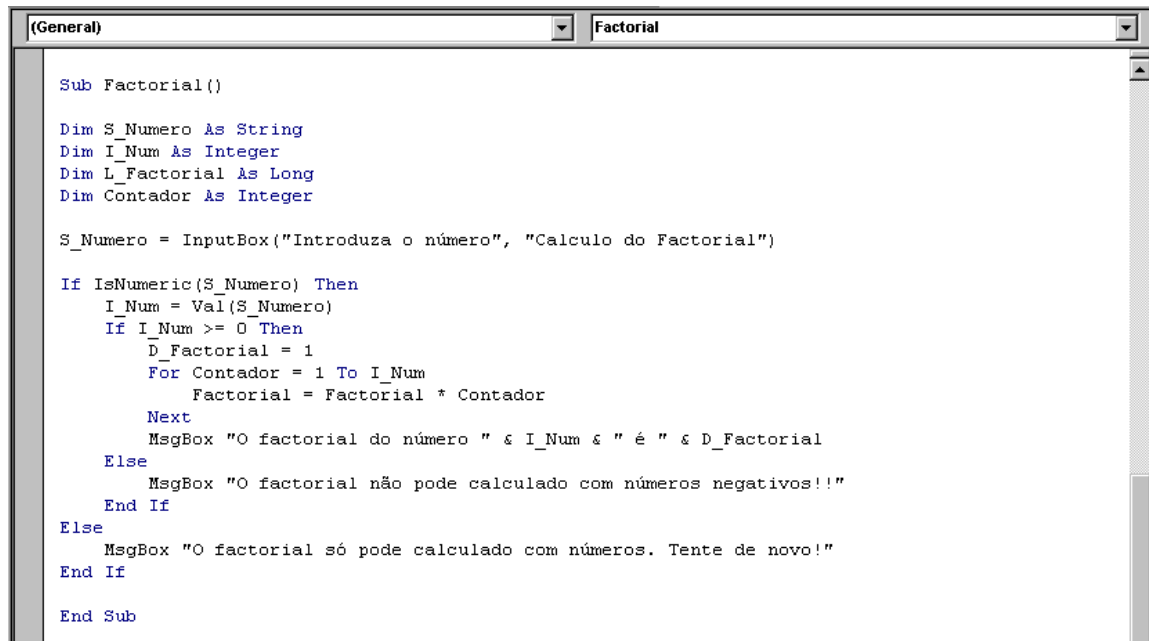


Figura 29 – Procedimento que calcula o fatorial

WHILE-WEND

A estrutura While-Wend tem um funcionamento similar ao For-Next. Realiza um looping um determinado número de vezes, até que uma determinada condição seja verdadeira.

Sintaxe

```
While <Condição>  
    <Instruções a realizar em cada iteração>  
Wend
```

Aplicação Prática

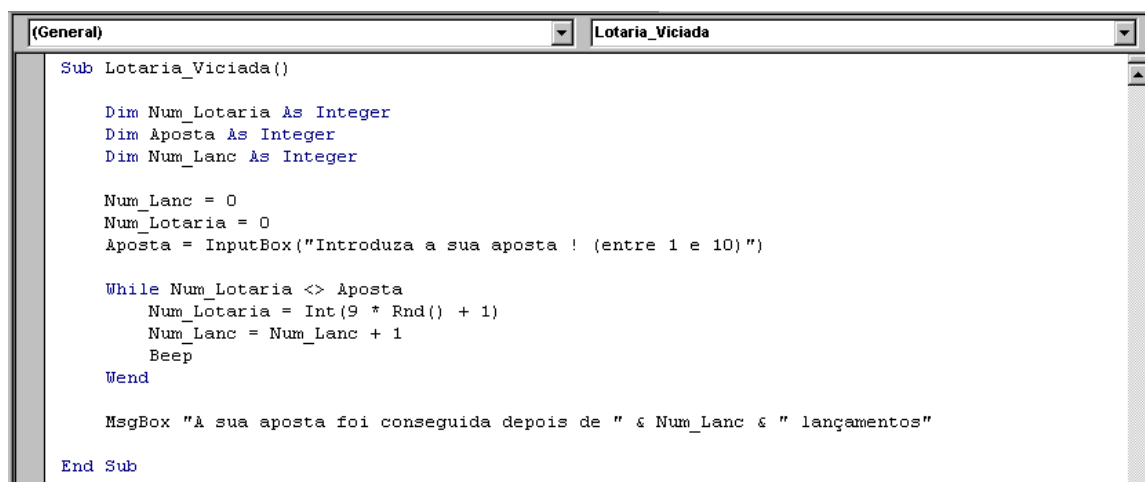


Figura 30 – Procedimento “Lotaria Viciada”

A instrução While-Wend tem como função gerar números aleatórios entre 1 e 10 por forma a encontrar o número da aposta, e saber qual o número de lançamentos necessários para que aquele valor fosse obtido.

A FUNÇÃO DAS VARIÁVEIS:

Variável	Função
Num_Lotaria	Número sorteado. Este será iniciado a 0 por forma a que não corresponda a nenhum valor introduzido pelo usuário e assim possa realizar o primeiro sorteio.
Aposta	Número em que o usuário pretende apostar.
Num_Lanc	Número de lançamentos realizados até obter o resultado da aposta. Este valor tem de ser incrementado cada vez que é realizado um sorteio.

CONSTRUÇÃO DO CICLO:

While	Palavra-chave que indica o início do ciclo While-Wend
Num_Lotaria <> Aposta	Condição teste utilizada para determinar o terminus da realização do ciclo. Se esta condição for Verdadeira executa as instruções que estão dentro do While-Wend, se for Falsa a execução do ciclo é terminada tendo o programa sequência nas instruções que seguem a palavra chave Wend.
Num_Lotaria = Int (9 * Rnd() + 1)	Instrução a realizar de cada vez que o ciclo é executado. Tem como função gerar números aleatórios entre 1 e 10.
Num_Lanc = Num_Lanc +1	Instrução a realizar de cada vez que o ciclo é executado. Tem como função fazer a contagem de quantos lançamentos foram realizados até se obter o valor da aposta.
Beep	Instrução a realizar de cada vez que o ciclo é executado. Tem como função apitar em cada sorteio.
Wend	Palavra Chave que indica o fim de um ciclo While-Wend.

TRADUÇÃO INTEGRAL

```
While Num_Lotaria <> Aposta  
    Num_Lotaria = Int ( 9 * Rnd() + 1 )  
    Num_Lanc = Num_Lanc + 1  
    Beep  
Wend
```

Enquanto o número sorteado não for igual ao valor da aposta, o sorteio continua, o que implica sortear um número contabilizar o número de sorteios realizados e apitar para que o usuário tenha a percepção do que está a ser realizado.

Funcionamento do Ciclo

Existe uma **fase de inicialização** das variáveis envolvidas na condição – Teste para garantir o correto funcionamento do ciclo.

Avalia a condição teste e se for verdadeira executa todas as instruções até à palavra-chave *Wend* voltando de novo à avaliação da condição, se for falsa prossegue a execução da rotina nas instruções que se localizam depois da palavra-chave *Wend*.

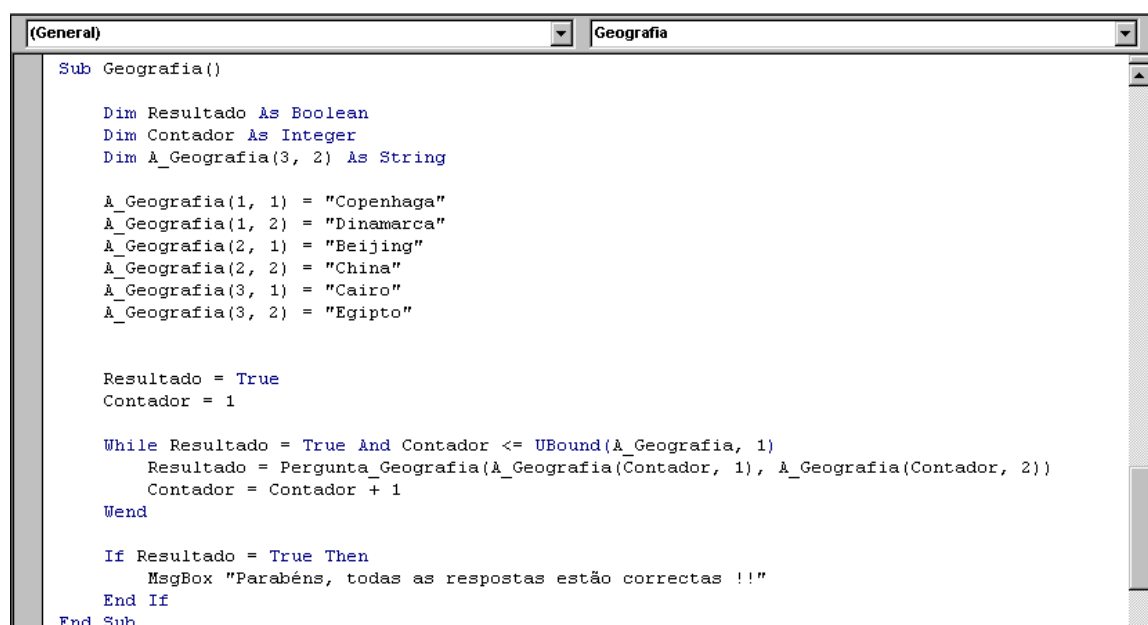
Perigos associados à utilização do ciclo While-Wend

- Má ou inexistência de inicialização das variáveis envolvidas na condição-teste.
- Garantir que as variáveis envolvidas na condição teste poderão ter valores diferentes por cada vez que o ciclo seja executado.
- Garantir que em algum momento a condição teste é falsa e o ciclo termina a sua execução.

⇒ A não verificação destas condições implicará que o ciclo não tenha fim – **Ciclos Infinitos**

Outra Aplicação

Pretende-se realizar um jogo de geografia. Tente compreender o seu funcionamento.



```
(General) | Geografia

Sub Geografia()

    Dim Resultado As Boolean
    Dim Contador As Integer
    Dim A_Geografia(3, 2) As String

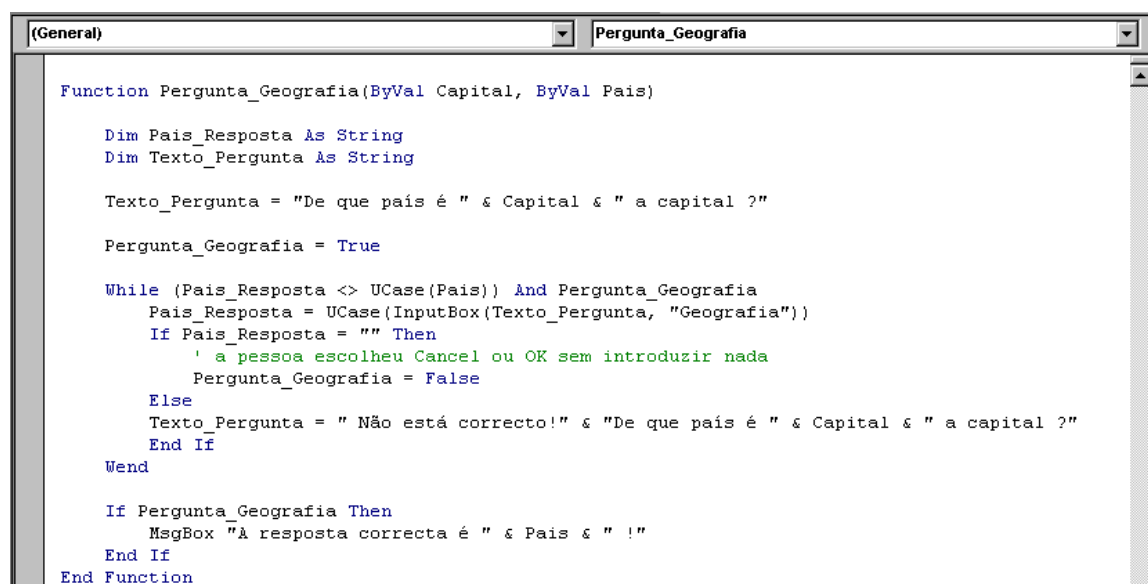
    A_Geografia(1, 1) = "Copenhaga"
    A_Geografia(1, 2) = "Dinamarca"
    A_Geografia(2, 1) = "Beijing"
    A_Geografia(2, 2) = "China"
    A_Geografia(3, 1) = "Cairo"
    A_Geografia(3, 2) = "Egipto"

    Resultado = True
    Contador = 1

    While Resultado = True And Contador <= UBound(A_Geografia, 1)
        Resultado = Pergunta_Geografia(A_Geografia(Contador, 1), A_Geografia(Contador, 2))
        Contador = Contador + 1
    Wend

    If Resultado = True Then
        MsgBox "Parabéns, todas as respostas estão correctas !!"
    End If
End Sub
```

Figura 31 – Corpo do jogo



```
(General) | Pergunta_Geografia

Function Pergunta_Geografia(ByVal Capital, ByVal Pais)

    Dim Pais_Resposta As String
    Dim Texto_Pergunta As String

    Texto_Pergunta = "De que país é " & Capital & " a capital ?"

    Pergunta_Geografia = True

    While (Pais_Resposta <> UCase(Pais)) And Pergunta_Geografia
        Pais_Resposta = UCase(InputBox(Texto_Pergunta, "Geografia"))
        If Pais_Resposta = "" Then
            ' a pessoa escolheu Cancel ou OK sem introduzir nada
            Pergunta_Geografia = False
        Else
            Texto_Pergunta = " Não está correcto!" & "De que país é " & Capital & " a capital ?"
        End If
    Wend

    If Pergunta_Geografia Then
        MsgBox "A resposta correcta é " & Pais & " !"
    End If
End Function
```

Figura 32 – Função Auxiliar

Do – Loop

Esta estrutura é similar à estrutura do While-Wend. Contudo fornece duas possibilidades que estão limitadas àquela estrutura:

- Do Loop permite **posicionar a condição teste no início ou fim do loop**, a condição no fim do Loop evita uma inicialização prévia do valor das variáveis envolvidas na condição teste, dado que essa inicialização pode ser feita no decurso do ciclo com valores reais.
- Do Loop permite ainda especificar se o loop se vai realizar enquanto (**while**) uma expressão for verdadeira ou até que (**until**) a condição seja verdadeira (facilidade conseguida através do operador Not)

Sintaxe

Poderá ser:

Do [{ **While** | **Until** } <condição>]
 <Instruções a realizar em cada iteração>

Loop

Ou então:

Do
 <Instruções a realizar em cada iteração>

Loop[{ **While** | **Until** } <condição>]

Aplicações Práticas

- ◆ Utilizando a **condição teste no início do Loop** e com a palavra **While**

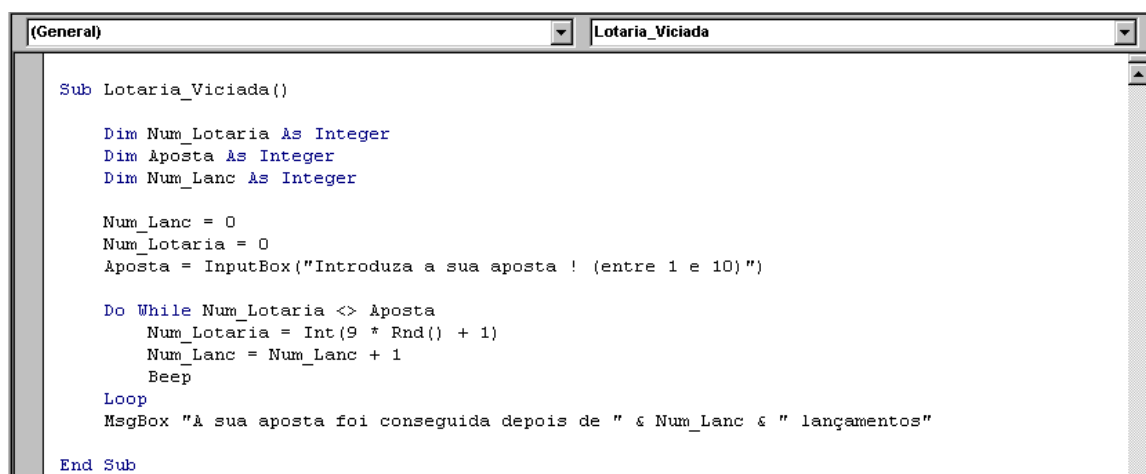


Figura 33 – Condição teste no início – com While

- ◆ Utilizando a **condição teste no início do Loop** e com a palavra **Until**

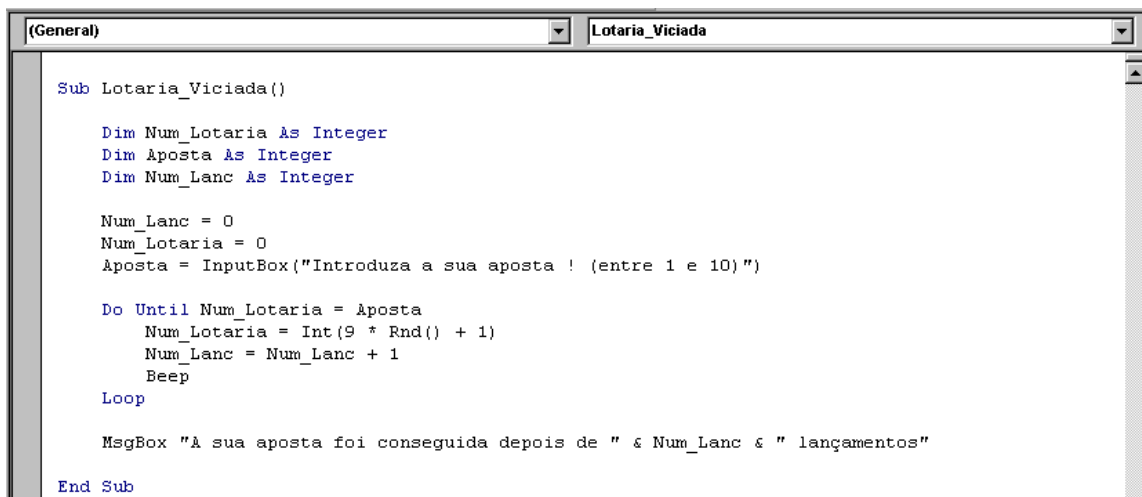
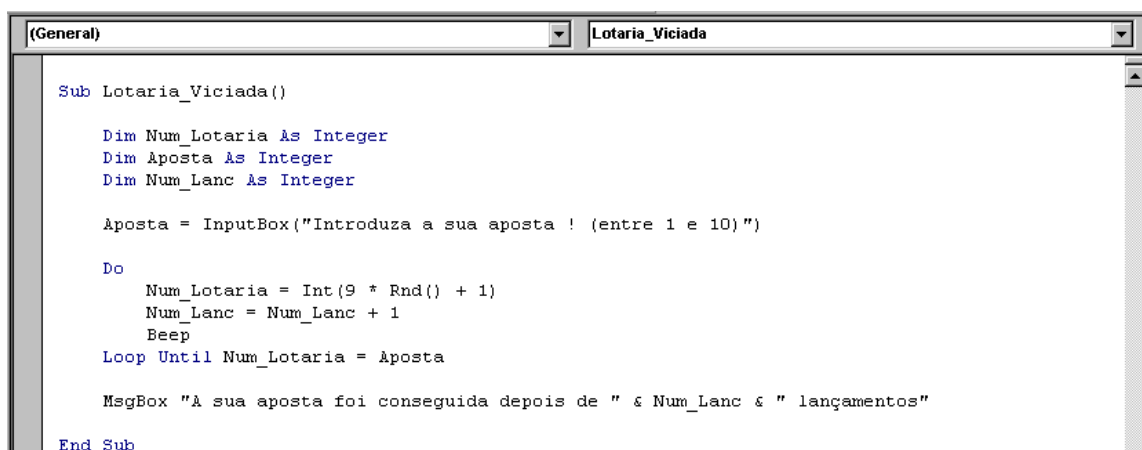


Figura 34 – Condição teste no início – com Until

Excel – Macros e Visual Basic for Applications (versão Draft)

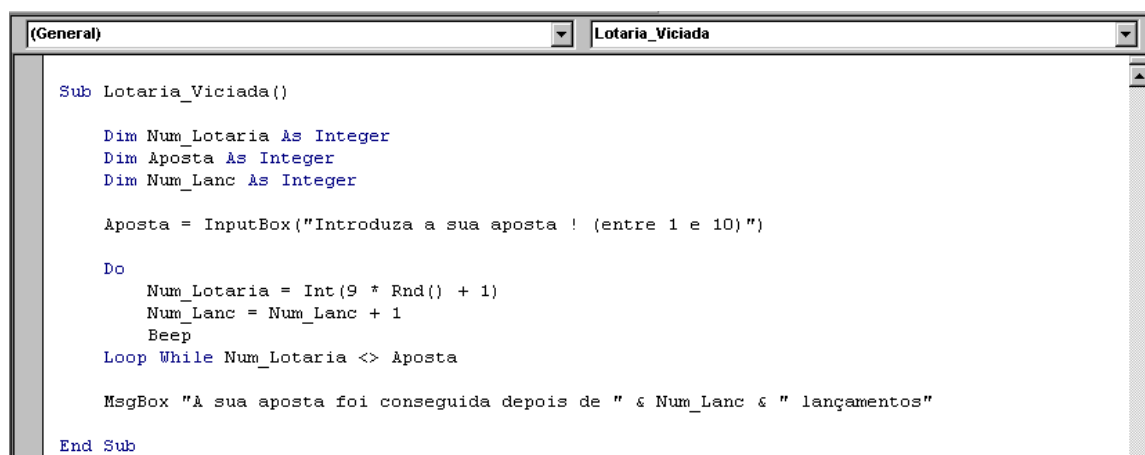
- ♦ Utilizando a **condição teste no fim do Loop** e com a palavra **Until**



```
Sub Lotaria_Viciada()  
  
    Dim Num_Lotaria As Integer  
    Dim Aposta As Integer  
    Dim Num_Lanc As Integer  
  
    Aposta = InputBox("Introduza a sua aposta ! (entre 1 e 10)")  
  
    Do  
        Num_Lotaria = Int(9 * Rnd() + 1)  
        Num_Lanc = Num_Lanc + 1  
        Beep  
    Loop Until Num_Lotaria = Aposta  
  
    MsgBox "A sua aposta foi conseguida depois de " & Num_Lanc & " lançamentos"  
  
End Sub
```

Figura 35 – Condição teste no fim – com Until

- ♦ Utilizando a **condição teste no fim do Loop** e com a palavra **While**



```
Sub Lotaria_Viciada()  
  
    Dim Num_Lotaria As Integer  
    Dim Aposta As Integer  
    Dim Num_Lanc As Integer  
  
    Aposta = InputBox("Introduza a sua aposta ! (entre 1 e 10)")  
  
    Do  
        Num_Lotaria = Int(9 * Rnd() + 1)  
        Num_Lanc = Num_Lanc + 1  
        Beep  
    Loop While Num_Lotaria <> Aposta  
  
    MsgBox "A sua aposta foi conseguida depois de " & Num_Lanc & " lançamentos"  
  
End Sub
```

Figura 36 – Condição teste no fim – com While

SELECT CASE

Permite a escolha de um percurso mediante a avaliação de n condições. É de extrema utilidade para evitar os If's encadeados, dando um maior grau de legibilidade e simplicidade ao código construído.

Sintaxe

Select Case <Expressão a ser avaliada>

[**Case** <Valor da Expressão>
[Instruções a realizar]]

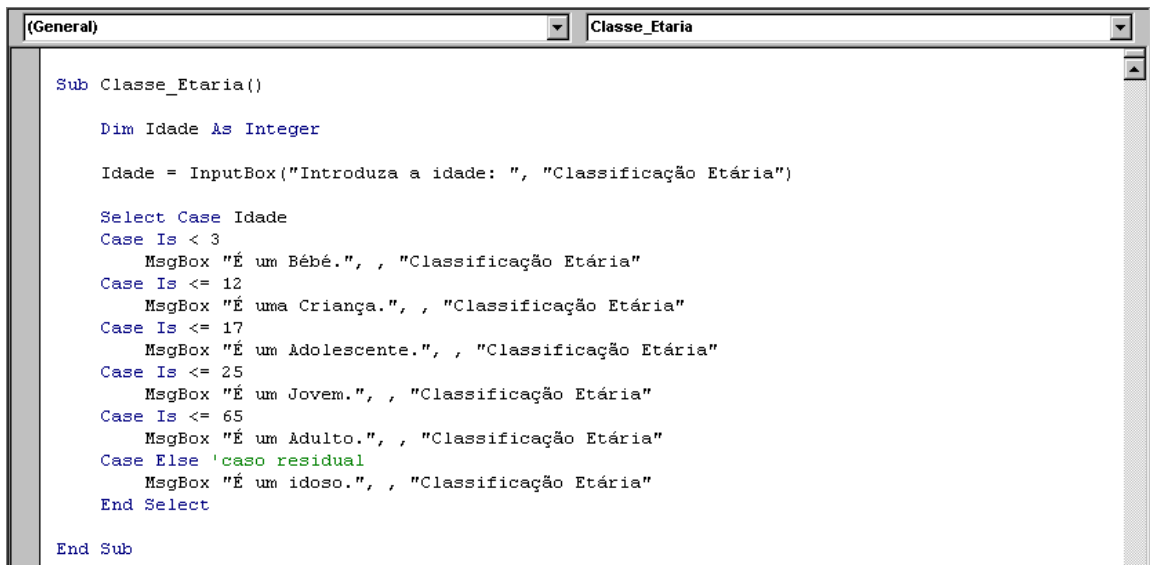
...

[**Case Else**
[Instruções a realizar na situação residual]]

End Select

Aplicação Prática

Recordem o processo resolvido com recurso a **If –Then –Else – ElseIf** (figura 23)
Mais facilmente seria resolvido com recurso à estrutura **Select Case**



```
Sub Classe_Etaria()  
  
    Dim Idade As Integer  
  
    Idade = InputBox("Introduza a idade: ", "Classificação Etária")  
  
    Select Case Idade  
    Case Is < 3  
        MsgBox "É um Bêbé.", , "Classificação Etária"  
    Case Is <= 12  
        MsgBox "É uma Criança.", , "Classificação Etária"  
    Case Is <= 17  
        MsgBox "É um Adolescente.", , "Classificação Etária"  
    Case Is <= 25  
        MsgBox "É um Jovem.", , "Classificação Etária"  
    Case Is <= 65  
        MsgBox "É um Adulto.", , "Classificação Etária"  
    Case Else 'caso residual  
        MsgBox "É um idoso.", , "Classificação Etária"  
    End Select  
  
End Sub
```

Figura 37 – Aplicação da estrutura Select Case à rotina Classe_Etaria

CONSTRUÇÃO DA ESTRUTURA

Select Case	Palavras-Chave que indicam o início de um controlo Select Case
Idade	Expressão sujeita a teste, i.e., variável cujo conteúdo está a ser avaliado. Esta variável vai ser comparada sucessivamente pelos valores alternativos apresentados nas instruções Case <Valor>, se encontrar o valor nalguma dessas opções Case executará as linhas de código que aí terão início até à opção de Case seguinte. Caso o valor da variável a ser comparada não corresponda a nenhum valor apresentado nas opções Case, existe uma opção Case especial - Case Select - para os restantes valores, neste caso serão executadas todas as instruções que se localizem entre o Case Else e o Case Select.
Case Is<3 ou Case Is<=12 ou Case Is<=17 ou Case Is<=25 ou Case Is<=65 ou	Expressões Case. Se o valor da variável for igual a qualquer um dos valores apresentados em cada uma destas expressões, o fluxo de execução terá continuidade na linha abaixo da expressão case que faz o matching, até que uma nova expressão case seja encontrada. Sendo que nessa altura termina o controlo Select Case dando continuidade ao programa nas instruções que se seguirem ao End Select.
Case Else	Será a instrução Case residual, selecionada somente se nenhuma das outras o tiver sido. Neste caso serão realizadas todas as instruções de código que se lhe seguirem até à expressão End Select. Findo o qual seguirá todas as instruções após o controlo Select case.
End Select	Palavra-Chave que indica o fim do controlo Select Case.

FOR – EACH – NEXT

A estrutura For-each-next é de longe a mais potente do VBA. De fato permite executar uma determinada instrução em todos os elementos de uma coleção de objetos, ou em todos os elementos de um array.

Esta estrutura **quando aplicada em arrays** não funciona para a atualização dos valores do array, mas somente para a extração do seu conteúdo. Contudo quando aplicada a coleções de objetos pode sê-lo para alteração das suas propriedades ou extração de valores.

Sintaxe

For Each <Variável do tipo dos elementos do grupo> **In** <Grupo>

<Instruções a realizar para cada elemento do grupo>

Next

Aplicações Práticas

UTILIZANDO ARRAYS

Pretende-se iniciar um array com um conjunto de 5 países e posteriormente visualizar os elementos introduzidos.

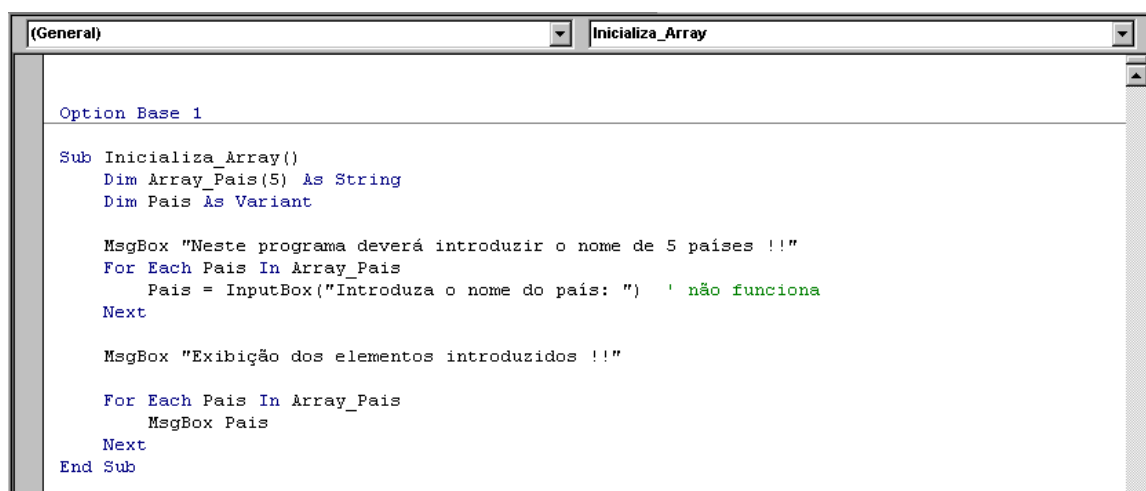


Figura 38 – Exemplo não-funcional

Repare que o exemplo, poderá estar conceitualmente correto, mas não funciona devido à restrição do For-Each-Next para a alteração de valores de arrays – tarefa impossível de realizar. A alternativa poderia ser, a exibida na figura 39:

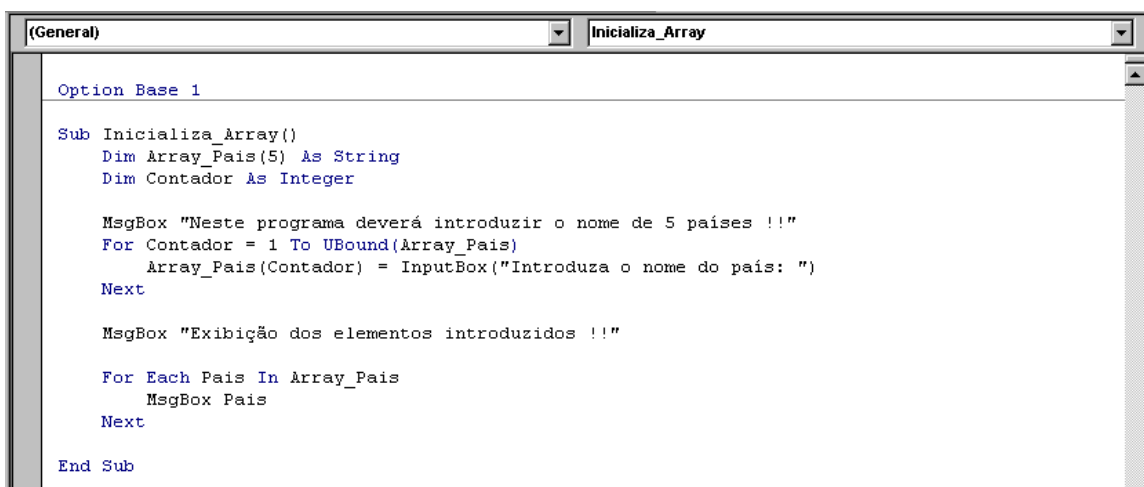
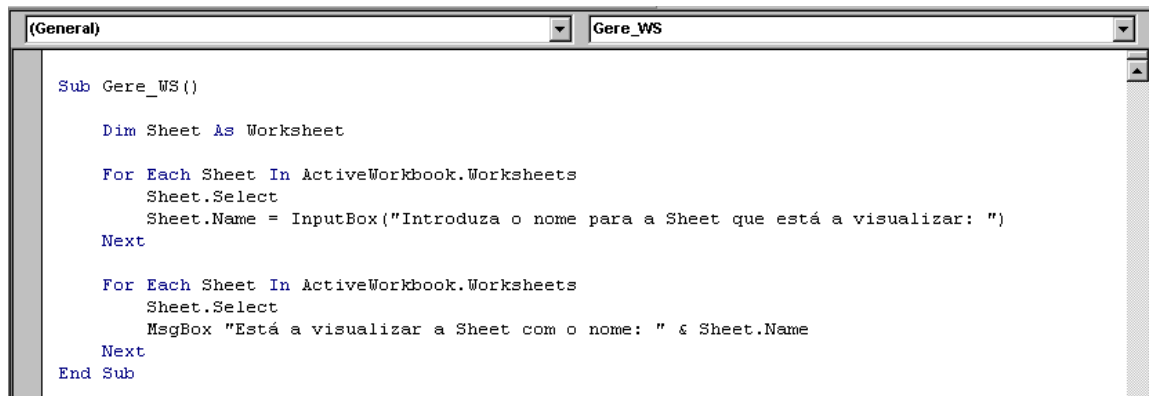


Figura 39 – Alteração da sub-rotina Inicializa_Array

Construção do Ciclo

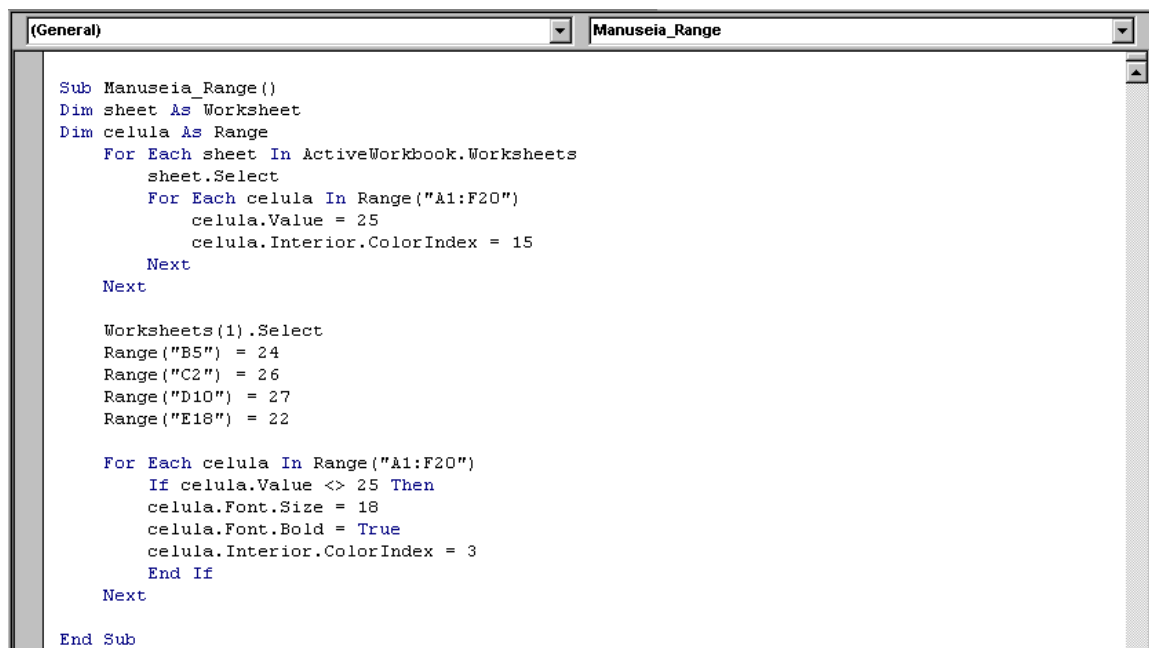
For Each	Palavras-Chave que indicam o início de um controlo For-Each
Pais	Variável à qual vão ser atribuídos sucessivamente todos os elementos do grupo de objetos. Este grupo pode ser constituído por uma array ou uma coleção de objetos. Sendo que se se tratar de um array esta variável deverá ser do tipo Variant . Se tratar de uma coleção, esta variável poderá assumir o tipo Variant, o tipo Object genérico ou o tipo Object específico que corresponde ao tipo de objetos a que a coleção remete.
In	Palavra-chave que separa a variável do grupo.
Array_Pais	Grupo de elementos a tratar. Poderá ser um array ou uma coleção de objetos. O ciclo será executado tantas vezes quantos os elementos constantes do grupo. Na primeira iteração, a variável assume o valor do primeiro item do grupo, como tal em cada loop este valor será atualizado pelo item seguinte.
MsgBox Pais	Instrução a realizar em cada volta do ciclo. Sendo que em cada volta a variável Pais terá um elemento diferente.
Next	Palavra-chave que indica o fim do loop. Neste momento o programa terá continuação na instrução For Each correspondente se a variável não corresponder ao último item do grupo, caso contrário sairá da instrução For Each-Next prosseguindo a execução da rotina na linha de código abaixo do Next.

UTILIZANDO COLEÇÕES DE OBJETOS



```
Sub Gere_WS()  
  
    Dim Sheet As Worksheet  
  
    For Each Sheet In ActiveWorkbook.Worksheets  
        Sheet.Select  
        Sheet.Name = InputBox("Introduza o nome para a Sheet que está a visualizar: ")  
    Next  
  
    For Each Sheet In ActiveWorkbook.Worksheets  
        Sheet.Select  
        MsgBox "Está a visualizar a Sheet com o nome: " & Sheet.Name  
    Next  
End Sub
```

Figura 40 – Rotina para atribuir nomes às WorkSheets do Excel



```
Sub Manuseia_Range()  
    Dim sheet As Worksheet  
    Dim celula As Range  
    For Each sheet In ActiveWorkbook.Worksheets  
        sheet.Select  
        For Each celula In Range("A1:F20")  
            celula.Value = 25  
            celula.Interior.ColorIndex = 15  
        Next  
    Next  
  
    Worksheets(1).Select  
    Range("B5") = 24  
    Range("C2") = 26  
    Range("D10") = 27  
    Range("E18") = 22  
  
    For Each celula In Range("A1:F20")  
        If celula.Value <> 25 Then  
            celula.Font.Size = 18  
            celula.Font.Bold = True  
            celula.Interior.ColorIndex = 3  
        End If  
    Next  
End Sub
```

Figura 41 – Rotina para demonstrar o manuseamento de ranges

Coleções de Objetos e Objetos

O QUE SÃO OBJETOS ?

Objetos são elementos caracterizados por um conjunto de propriedades, e que têm subjacente um determinado comportamento. Por exemplo, uma janela do windows é um objeto, caracterizada por um nome, um título, uma dimensão, um posicionamento dentro da tela..., e tem um comportamento inerente, pode ser aberta, fechada, minimizada, maximizada, escondida, redimensionada,...

Seguindo esta lógica podemos identificar alguns objetos do ambiente de trabalho Windows: o botão, o gráfico, o menu, o documento, a imagem, a caixa de texto, a fórmula, o workbook, a worksheet, a célula, o conjunto de células de uma worksheet,...

OBJETOS: PROPRIEDADES, MÉTODOS E EVENTOS

Propriedades

As propriedades dos objetos constituem o conjunto de características que o definem. Por exemplo: nome, cor, dimensão, designação, valor contido, ...

Métodos

Os métodos traduzem o comportamento de um objeto. Estes métodos representam procedimentos que executam uma determinada tarefa, que pode ser complementada através da passagem de argumentos ou parâmetros.

Eventos

Eventos ou acontecimentos, representam todas as atividades que envolvam o objeto e que normalmente direta ou indiretamente são disparadas pelo usuário. Por exemplo: abrir ou fechar um workbook, clicar sobre um botão ou worksheet, alterar o conteúdo de um elemento, ...

Estes eventos servem para que possamos ativar uma determinada tarefa quando da sua ocorrência.

Exemplo:

Suponha que pretende executar uma macro quando abre o seu workbook.

1º Crie a macro que pretende executar.

2º No Editor de Visual Basic, na janela de Projeto – *Project Window* – selecione o objeto *ThisWorkBook*. Na janela de edição repare nas duas caixinhas que se encontram na parte superior. A do lado esquerdo indica *general* clique nela e selecione o elemento *workbook*, na caixinha da direita selecione o evento *Open*.

3º Automaticamente aparecerá um procedimento na janela de edição cujo nome será *Workbook_Open*, tudo o que escrever no seu conteúdo será executado quando o documento for aberto, neste caso indique o nome da macro que criou anteriormente.

OBJETOS MAIS UTILIZADOS NO EXCEL

Os Objetos mais utilizados no Excel são: Application, WorkBook, WorkSheet e Range

Application

Application é o objeto de topo hierárquico, representa o próprio Excel.

PROPRIEDADES

Propriedades	
Caption	Menção exibida na barra de títulos do Excel
DisplayAlerts	TRUE – as mensagens de alerta são exibidas durante a execução da sub rotina. False caso contrário.
Path	Nome do diretório onde o Excel está instalado.
ScreenUpdating	True – altera a tela consoante a sub rotina que executa. Caso contrário, a tela não é alterada durante a execução da sub rotina.
WindoWorkSheetState	O estado da janela da aplicação: XINormal – janela tamanho normal XIMaximized – janela tamanho maximizado XIMinimized – janela com tamanho minimizado
DisplayStatusBar	Mostra ou esconde a StatusBar. True – exhibe
DisplayFormulaBar	Mostra ou esconde a Barra de Formulas True – exhibe

MÉTODOS

Métodos	
Calculate	Implica o cálculo de todas as fórmulas de todas as WorkSheet e WorkBookS abertos. Argumentos : não tem
Help	Exibe um tópico do Help de um determinado arquivo de Help. Argumentos : Helpfile: nome do arquivo, incluindo a path se necessário Helpcontextid: número que faz referência ao índice de help
Quit	Fecha aplicação Excel. (se a propriedade Display alerts estiver com o valor False, o Excel não proporá a gravação de alterações nos arquivos) Argumentos : não tem
Run	Utilizado para correr uma sub rotina de VB ou uma Macro do XL Argumentos : Macro: nome da macro ou sub rotina a executar Argumentos que a sub rotina necessita para ser executada – arg1:=<valor 1>, arg2:=<valor>,...

WorkBook

O Objeto WorkBook, na hierarquia de objetos segue de imediato o objeto application e representa um arquivo de Excel.

PROPRIEDADES

Propriedades	
Name	Nome do arquivo. Não permite a alteração do nome, para isso é necessário proceder ao Save/As
Path	Diretório onde o arquivo se encontra
Saved	True – se não houve nenhuma alteração no arquivo desde a última gravação False – caso contrário Exemplo: If not(activeWorkbook.Saved) Then ActiveWorkbook.Save End If

MÉTODOS

Métodos	
Activate	Ativa um documento aberto. Equivalente a ir ao Window e seleccionar um documento aberto, trazendo-o para a janela principal do Excel. Argumentos: não tem
Close	Fecha um documento. Argumentos: SaveChanges – se True, o documento é gravado antes de ser fechado; False caso contrário FileName – se o argumento SaveChanges estiver em TRUE, o WorkBook é gravado com o nome aqui indicado RoutWorkBook – se TRUE e o WorkBook tiver endereços para envio atribuídos, envia o arquivo por e-mail antes de fechar, caso contrário não.
Protect	Protege um documento contra qualquer tipo de alteração Argumentos: Password: Senha a utilizar na proteção Structure: True protege também a estrutura WindoWorksheet: True a estrutura do WorkBook na janela é protegida
Save	Grava o WorkBook. Argumentos: Não Tem
SaveCopyAs	Cria uma cópia do documento em questão Argumentos: FileName: nome da cópia pretendida para o arquivo

WorkSheet

Na hierarquia situa-se abaixo do objeto Workbook, uma vez que um Workbook é constituído por um conjunto de Worksheets.

PROPRIEDADES

Propriedades	
Index	Índice de uma Worksheet num Workbook
Name	Nome da Worksheet.
UsedRange	Traduz o range na Worksheet que contém dados.
Visible	True – está visível False – está escondida, mas o usuário pode visualizá-la recorrendo ao Menu Format XlVeryHidden – está escondida e ninguém a pode visualizar, a menos que volte a aplicar uma sub rotina que a coloque visível.

MÉTODOS

Métodos	
Activate	Ativa uma determinada WorkSheet. Equivalente a trabalharmos num WorkBook e clicarmos nela para visualizarmos o seu conteúdo. Argumentos: Não Tem
Calculate	Provoca o cálculo de todas as fórmulas constantes da WorkSheet Argumentos: Não Tem
Delete	Apaga uma WorkSheet do WorkBook. Argumentos: Não Tem
Protect	Protege uma WorkSheet contra qualquer tipo de alteração Argumentos: Password: Senha a utilizar na proteção DrawingObjects: True- protege os objetos gráficos Contents – True- protege as células e conteúdos Scenarios – True – protege os cenários afetos à WorkSheet UserInterfaceOnly – True – protege as interfaces contra alterações apesar de poder alterar as sub rotinas
Cell	Referência uma célula, através dos seus índices numéricos. Argumentos: Número da Linha Número da Coluna Ex: Célula C5 será representado por Cells(5, 3)

Range

Objeto utilizado para representar uma ou mais células de uma WorkSheet.

PROPIEDADES

Propriedades	
Count	Número de Células num Range. Read-Only
Dependents	Retorna um range, que contém todos os dependentes (valores indexados pelas fórmulas) do range em questão. Read-Only
Name	Nome de um range. Read/Write
Value	Valor constante de um range (célula ou conjunto de células). Read/Write
Formula	Traduz a fórmula contida num range como uma string. Read/Write
Text	Busca o conteúdo de uma célula mas em formato de texto.

MÉTODOS

Métodos	
Calculate	Provoca o cálculo da fórmula constantes do range Argumentos: Não Tem
ClearContents	Apaga o conteúdo (fórmulas e valores) de uma célula, deixando os formatos. Argumentos: Não Tem
Copy	Copia o conteúdo de um range para um outro de igual dimensão ou então para o clipboard. Argumentos: Destination – range para o qual os valores vão ser copiados (na ausência deste parâmetro a cópia é feita para o clipboard)
Offset	Provoca um deslocamento de um determinado número de linhas e de colunas, tendo como base o range ao qual este método está a ser aplicado. Argumentos: RowOffset – número de linhas que se desloca ColumnOffset – número de colunas que se desloca
EntireRow	Faz referência à(s) linha(s) indicadas por um determinado range. Argumentos: Não Tem
Select	Seleciona o range em questão. Argumentos: Replace- (Opcional)
Cell	Referência uma célula, através dos seus índices numéricos. Argumentos: Número da Linha Número da Coluna Ex: Célula C5 será representado por Cells(5, 3)

OBJETOS SINGULARES VS COLEÇÕES DE OBJETOS

Objeto Singular – refere um único objeto que pode ser referenciado pelo nome.

Coleções de Objetos – constituem conjuntos de objetos singulares que são referenciados pelo índice que os identifica na coleção.

As coleções de objetos também podem ser considerados como objetos.

Exemplo:

WorkBooks(“Book1.XLS”) é um conjunto de objetos do tipo WorkSheet, mas também é um objeto do tipo Workbook.

INDEXAÇÃO DE COLEÇÕES POR NÚMERO OU NOME

Perante uma coleção de objetos é necessário identificar cada um dos diferentes itens que a compõem para que a eles seja possível acessar.

Por analogia poder-se mencionar o exemplo do array, este é uma variável plural, dado que é composto por um conjunto de variáveis. Quando se pretende acessar a uma posição do array utiliza-se o nome do array e o índice da posição requerida.

Assim, a indexação de coleções pode ser realizada com base:

- em números ou
- em nomes.

Indexação com Base em Números

Os números prendem-se com a ordem pela qual o objeto está inserido na coleção (começa em 1).

Exemplo:

WorkSheets(3).Name="Terceiro"

Indexação com Base no Nome

Seleciona-se um objeto numa coleção pelo nome que está associado ao objeto.

Exemplo:

```
Worksheets("Sheet3").Name="Terceiro"
```

VANTAGEM

Não é necessário saber a ordem pela qual foi inserido na coleção

DESVANTAGEM

Alteração do nome da sheet provoca erros

Exemplo:

```
Worksheets("Sheet3").Name="Terceiro"
```

```
Worksheets("Sheet3").Visible=False" ‘ o objeto não é reconhecido
```

O Objeto Range – uma exceção

O objeto range é referido da mesma forma para ser utilizado como um objeto singular ou coleção de objetos.

TRATAMENTO COMO OBJETO:

```
Range(“A1”).Value=1
```

Equivalente a colocar na primeira célula da Sheet o valor 1.

TRATAMENTO COMO COLEÇÃO DE OBJETOS:

```
Range (“A1:F20”).Value= 1
```

Equivalente a colocar em todas as células do range A1 a F20 o valor 1.

Ou então:

```
Range (“A1:F20”).Name= “Conjunto”
```

```
Range (“Conjunto”).Value= 1
```

Onde, na primeira instrução se atribui ao range “A1:F20” o nome “Conjunto”, e na última instrução se utiliza essa designação para referenciar o respectivo conjunto de células e atribuir-lhe o valor 1.

REFERÊNCIA IMPLÍCITA

Quando se faz referência a uma célula da worksheet, pode-se fazê-lo de diversas formas equivalentes. No quadro seguinte é exibida a equivalência entre a expressão mais completa e a mais reduzida, sendo que ambas têm a mesma função (colocar na célula A1 o valor 1):

<p><code>Application.Workbooks(1).Worksheets(1).Range("A1").Value=1</code> ⇔ <code>Range("A1").Value=1</code></p>

A diferença entre ambas as formas de acesso está no fato da segunda forma (`Range("A1").Value = 1`) admitir que se está a trabalhar no workbook e na worksheet que nesse momento estão ativas no Excel, enquanto que na primeira forma são indicadas as referências identificadoras do workbook e da worksheet onde se pretende trabalhar.

Assim, pode-se afirmar que a segunda forma faz uma alusão implícita à aplicação, ao workbook e à worksheet onde se trabalha.

Declaração implícita da aplicação:

Para fazer a alusão implícita da aplicação, basta não a indicar, e por padrão o Excel assume que se está a trabalhar no seu contexto:

<p><code>Workbooks(1).Worksheets(1).Range("A1").Value=1</code></p>

Declaração implícita do WorkBook:

Omitir a referência ao workbook, é semelhante a assumir o workbook ativo como ambiente de trabalho. Neste contexto, as expressões abaixo indicadas assumem por padrão que se está a trabalhar no Excel, e no workbook que nesse momento estiver ativo.

**ActiveWorkbook.Worksheets(1).Range("A1").Value=1 ⇔
Worksheets(1).Range("A1").Value=1 ⇔**

Declaração implícita da WorkSheet:

O mesmo se aplica relativamente às worksheets.

**ActiveSheet.Range("A1").Value=1 ⇔
Range("A1").Value=1 ⇔**

Nível de referência a privilegiar

Temos aqui representados 7 níveis de codificação:

Então, se existem tantas formas de referenciar uma célula, qual a forma privilegiar?

Application.Workbooks(1).Worksheets(1).Range(“A1”).Value=1 ⇔

Workbooks(1).Worksheets(1).Range(“A1”).Value=1 ⇔

ActiveWorkbook.Worksheets(1).Range(“A1”).Value=1 ⇔

Worksheets(1).Range(“A1”).Value=1 ⇔

Activsheet.Range(“A1”).Value=1 ⇔

Range(“A1”).Value=1 ⇔

Range(“A1”)=1

O ideal seria escrever na forma completa, mas existem algumas **desvantagens**:

- **código muito denso** – dificuldades em escrever e ler
- **maior probabilidade de ocorrência de erros** – quando não se sabe precisamente o contexto em que a aplicação vai ser executada é perigoso descrever uma path certinha

Miscellaneous

A INSTRUÇÃO WITH

A instrução With permite abreviar referências a objetos. Não faz sentido utilizá-lo quando se pretende utilizar só uma propriedade ou método, mas quando pretendemos utilizar bastantes.

Aplicação Prática

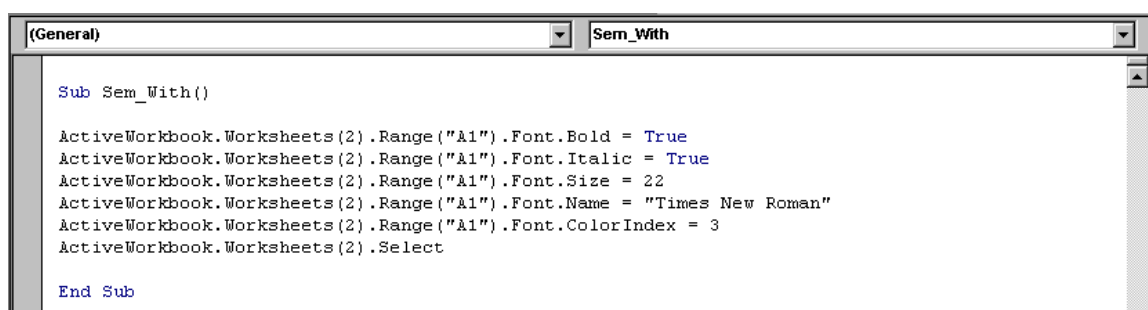


Figura 38 – Rotina para demonstrar o manuseamento de objetos sem a instrução With

Ou Então:

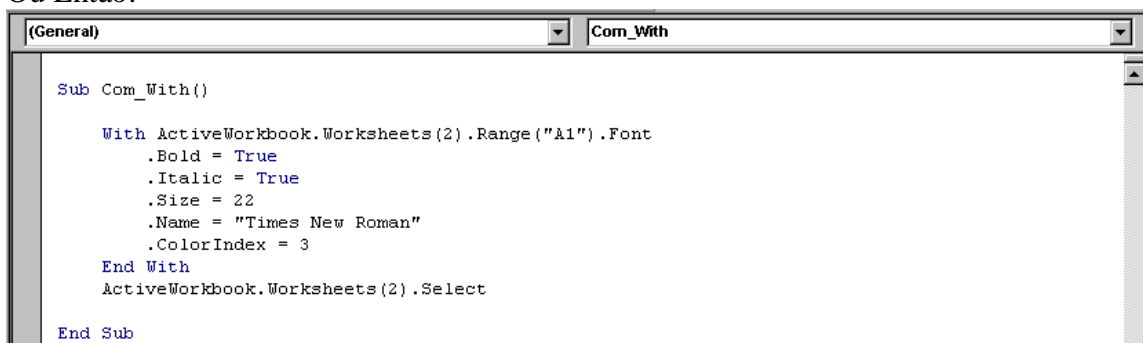


Figura 39 – Rotina para demonstrar o manuseamento de objetos com a instrução With

OUTRAS FUNÇÕES ÚTEIS DO VBA

Abs	Valor absoluto de um número.
CurDir	Diretório MS-DOS corrente.
Date	Data do sistema.
Exit Do	Interrompe a execução de uma ciclo Do – Loop
Exit For	Interrompe a execução de um ciclo For – Next ou For Each – Next
Exit Function	Provoca a interrupção da execução de uma função.
Exit Sub	Provoca a interrupção da execução de uma sub-rotina.
Fix	Arredonda um número decimal positivo para baixo, e um negativo para cima. Ex 3,9 ->3 e -3,9 -> -3
Int	Arredonda para cima um número decimal positivo ou negativo. Ex 3,9 ->4 e -3,9 -> -4
Is Array	True se a expressão é um array. False caso contrário.
IsDate	True se a expressão é do tipo Date. False caso contrário.
IsEmpty	True se nenhum valor foi atribuído à variável.
IsError	True se a expressão contiver um erro.
IsNull	True se a expressão representar o valor NULL.
IsNumeric	True se a expressão for numérica.
IsObject	True se se tratar de um objeto.
Len	Retorna a dimensão de uma String.
Now	Retorna o valor da data e da hora actual.
Shell	Corre um programa executável.
Sqr	Retorna a raiz quadrada de um número.
Str	Retorna a representação String de um número.
StrComp	Realiza a comparação de strings, produzindo True ou False conforme sejam ou não iguais.
Time	Produz a hora actual.
TypeName	Retorna o tipo de dados de uma variável.
Cint	Converte uma expressão de forma numérica ou textual para um valor de tipo inteiro.
Cbool	Converte uma expressão de forma numérica ou textual para um valor de tipo booleano.
Cdate	Converte uma expressão de forma numérica ou textual para um valor de tipo data.
CLng	Converte uma expressão de forma numérica ou textual para um valor de tipo Long.
CStr	Converte uma expressão de forma numérica ou textual para um valor de tipo String.