A linguagem Visual Basic

Introdução

Os computadores guardam a informação (dados) em memória. O elemento básico de memória é o *byte* (conjunto de 8 *bits*, cada *bit* pode tomar o valor **1** ou **0**). Com um *byte* podemos ter combinações de *bits* que vão de *0000 0000* (decimal 0) a *1111 1111* (decimal **255**), dando um total de 256 combinações (com 8 *bits* temos $2^8 = 256$ combinações). A única coisa que podemos ter em memória são números (com um *byte* podemos ter números inteiros de **0** a **255**).

Como podemos então guardar em memória informação que não é numérica? A resposta é simples, guardamos um número inteiro que representa (codifica) essa informação. Por exemplo se tivermos um total de 256 cores e quisermos usar um *byte* para guardar o valor da cor, codificamos as cores (0 – branco, 1 – preto, 2 – amarelo...). Não guardamos a cor em si mas apenas um número que a representa. O mesmo se passa com as letras do alfabeto, os computadores não guardam letras apenas um número que as representa.

Em memória só temos números. A informação não é mais do que a maneira como esses números são interpretados.

I - Tipos de Dados

O ***tipo*** é uma classe bem definida que representa informação com as mesmas características.
Essas características são fundamentalmente :

- O conjunto de valores que essa informação pode tomar
- As operações que podem ser feitas sobre esses valores

Numa linguagem de programação, a informação está contida em variáveis. As ***variáveis*** terão um ***tipo*** associado, conforme as características da informação nelas guardadas. Dizemos por exemplo que uma variável é do tipo inteiro, quando nela podem ser guardados valores com as características dos números inteiros.
Os *tipos* podem ser divididos em dois grandes grupos :

Tipos elementares

Representam informação que já está na forma mais simples. Informação que não pode ser decomposta em informação mais simples.

Tipos estruturados

Representam informação que ainda pode ser decomposta em informação mais simples que por sua vez, ainda pode ser elementar ou estruturada

1 - Os tipos elementares

1.1 O tipo *integer*

Uma variável do tipo *integer* pode guardar números inteiros entre –32768 e 32767
Como se chegou a estes valores?
O VB usa dois bytes para guardar números inteiros. Temos pois um total de 16 bits que dão combinações desde *00000000 00000000* (decimal 0) a *11111111 11111111* (decimal 65535 --»»2^{16} -1). Os números inteiros também podem tomar valores negativos, pelo que foi necessário arranjar uma maneira de representar esses valores. Os computadores usam um dos bits do número (o bit mais à esquerda – maior peso) para diferenciar números inteiros positivos de negativos. Se esse bit é **0** é um inteiro positivo, se é **1** é um inteiro negativo. Assim o maior inteiro positivo será *0111 1111 1111 1111* (primeiro bit tem que ser **0** para o número ser positivo). Este número é o inteiro **32767**. Se a este número somarmos **1** teriamos *1000 0000 0000 0000* (decimal 32768). No entanto como o primeiro bit do número é **1,** o número é por convenção negativo. Temos pois o inteiro negativo **–32768.**

Conjunto de operadores válidos :

- **+** adição
- **-** subtracção
- ***** multiplicação
- **/** divisão (no entanto o resultado da divisão deixa de ser um número inteiro)
- **** divisão inteira (parte inteira da divisão)
- **MOD** resto da divisão inteira
- **^** potência

Exemplo : 7 \ 2 – resultado -» 3 7 MOD 2 – resultado -» 1

Algumas funções predefinidas em VB e que devolvem inteiros:

Sintax	Devolve	Exemplo	Devolve
Int(*numero*)	Parte inteira de *numero* *	**Int**(8.7)	**8**
		Int(-8.7)	**-9**
Fix(*numero*)	Parte inteira de *numero* *	**Fix**(8.7)	**8**
		Fix(-8.7)	**-8**
Round(*expressao* [**,**N*umCasasDecimais*]**)**	Um número arredondado às N*umCasasDecimaiss* . Se *NumCasasDecimais* for 0 ou não existente devolve o inteiro mais próximo	**Round**(17.267)	**17**
		Round(17.6)	**18**
		Round(17.267, 2)	**17.27**
Val(*string***)**	Um número representado por *string*	**Val**("123")	**123**
Cint (*string* ou *ExpressaoNumerica***)**	Um número convertido a partir de *string* ou E*xpressaoNumeric***	**Cint**(17.6)	**18**
		Cint("17.6")	**18**
Asc(*string*)	Inteiro que representa o código ASCII do primeiro char de *string*	**Asc**("A")	**65**
		Asc("Apple")	**65**
Abs(*numero*)	Valor absoluto de *numero*	**Abs**(-5)	**5**

* A diferença entre **Int** e **Fix** é que e o número for negativo, **Int** devolve o primeiro inteiro negativo menor ou igual a *numero*, enquanto **Fix** devolve o primeiro inteiro negativo maior ou igual a *numero.*
** Use **Clng** para converter para *long* e **Cdbl** para converter para *double*

Exemplo de declaração de uma variável do tipo *integer* e atribuição de um valor a essa variável
DIM x as integer
X = 112

1.2 O Tipo *long*

O tipo *long* tem todas as características do *integer* excepto que as variáveis deste tipo podem tomar valores entre –2147483648 e 2147483647
(O Visual Basic usa 4 bytes para guardar valores do tipo *long*)

Exemplo de declaração de uma variável do tipo *long* e atribuição de um valor a essa variável
DIM x as long
X = 342279

1.3 O tipo *single*

O tipo *Single* (single precision number) deve ser usado para guardar números com parte fraccionária. Variáveis deste tipo podem guardar números de -3.402823E38 a -1.401298E-45 para números negativos e de 1.401298E-45 a 3.402823E38 para números positivos.
Nota : 3.402823E38 significa 3.402823 x 10^{38} (a este tipo de representação chama-se representação em *notação cientifica* ou *virgula flutuante*) O número à esquerda do **E** tem o nome de *mantissa*, o número à direita tem o nome de *expoente*. Os números do tipo *single* tem uma precisão de 7 dígitos (número máximo de dígitos na mantissa)

Conjunto de operadores válidos :
+ adição
- subtracção
* multiplicação
/ divisão
^ potência

Exemplo de declaração de uma variável do tipo *single* e atribuição de um valor a essa variável
DIM r as single
R = 32.725

1.4 O tipo *double*

O tipo *double* (double precision number) tem as características dos *single*, mas pode ser usado para guardar números maiores e com precisão maior (15 dígitos).Os números podem ir de 1.79769313486231E308 a -4.94065645841247E-324 para números negativos e de 4.94065645841247E-324 a 1.79769313486232E308 para números positivos.

Algumas funções predefinidas em VB e que devolvem *doubles* ou *singles*

Sintax	Devolve	Exemplo	Devolve
Cos(*angulo*)	Coseno de *angulo**	**Cos**(0)	**1.0**
Sin(*angulo*)	Seno de *angulo**	**Sin**(0)	**0.0**
Tan(*angulo*)	Tangente de *angulo**	**Tan**(0)	**0.0**
Log(*numero*)	Logaritmo natural de *numero* (logaritmo base **e**)	**Log**(1)	**0.0**
Sqr(*numero*)	Raiz quadrada de *numero*	**Sqr**(4)	**2.0**
Rnd()	Um valor do tipo **single** menor do que 1 mas maior ou igual a 0	**Int**(1 +10***rnd)**	**Inteiro** entre 1 e 10
Val(*string*)	Um número representado por *string*	**Val**("123.17")	**123.17**
Abs(*numero*)	Valor absoluto de *numero*	**Abs**(-5.0)	**5.0**
** angulo é expresso em radianos*			

Exemplo de declaração de uma variável do tipo *double* e atribuição de um valor a essa variável

```
DIM r as double
    R = 32.7E100
```

1.5 O tipo *boolean*

Uma variável do tipo *boolean* só pode tomar um de dois valores TRUE ou FALSE

Conjunto de operadores válidos :

AND conjunção
OR disjunção
NOT negação
XOR exclusão

Resultado das operações:

True AND True	**True**		**True OR True**	**True**
True AND False	**False**		**True OR False**	**True**
False AND True	**False**		**False OR True**	**True**
False AND False	**False**		**False OR False**	**False**
True XOR True	**False**		**NOT True**	**False**
True XOR False	**True**		**NOT False**	**True**
False XOR True	**True**			
False XOR False	**False**			

Examplos :

(5>2) AND (3=2)	resultado	**False**
(5>2) OR (3=2)	resultado	**True**
(5>2) XOR (3=2)	resultado	**True**
NOT (5>2)	resultado	**False**

Exemplo de declaração de uma variável do tipo *boolean* e atribuição de um valor a essa variável

```
DIM b as boolean
    b = FALSE
```

Nota: os operadores **AND**, **OR** e **XOR** também efectuam operações bit a bit em duas expressões numéricas e afectam o bit correspondente no *resultado* de acordo com a seguinte tabela:

Bit na expressão 1	**Bit na expressão 2**	**AND**	**OR**	**XOR**
		Bit correspondente no *resultado*		
1	**1**	**1**	**1**	**0**
1	**0**	**0**	**1**	**1**
0	**1**	**0**	**1**	**1**
0	**0**	**0**	**0**	**0**
Exemplo: 5 AND 3 resultado 1 ----» 101 AND 011 = 001 (em binario)				

O operador **NOT** inverte os valores dos bits de uma expressão numérica e afecta os bits correspondentes do *resultado* de acordo

2 – Os tipos estruturados

2.1 *strings*

Uma variável do tipo *string* pode guardar um ou mais caracteres. É o tipo ideal para guardar texto. Os valores são guardados entre aspas.

Há dois tipos de *strings* em VB :

- *strings* de comprimento variável (O comprimento é ajustado automaticamente quando o texto aumenta ou diminui)
- strings de comprimento fixo. (o número de caracteres é fixo e predeterminado)

Exemplo de declarações ilustrativas dos dois casos :

```
DIM nome as string / comprimento variável
DIM codigo as string*6 / string com 6 chars comprimento fixo
```

Concatenação de *strings*

Em VB o operador de concatenação é representado pelo símbolo **&**

Exemplo de aplicação:

```
DIM nome1 as string ,nome2 as string,nome3 as string
Nome1=”José “
Nome2=” Martins”
Nome3 = nome1 & nome2          / nome3 fica com o valor “José Martins”
```

Algumas funções predefinidas em VB para manipulação de strings

Sintaxe	Devolve	Exemplo	Devolve
Str(*numero*)	string correspondente a *numero*	**Str**(123)	**" 123"**
Len(*string*)	Integer correspondente ao número de caracteres de *string*	**Len**("CATOLICA")	**8**
Mid(*string, ini*[, *comp*])	Uma parcela de *string* começando no caracter de posição *ini* e *comp* caracteres a partir daí	**Mid**("CATOLICA",3,2)	**"TO"**
		Mid("CATOLICA",3)	**"TOLICA"**
Left(*string, comp*)	Os primeiros *comp* caracteres de *string*	**Left**("CATOLICA",3)	**"CAT"**
Right(*string,comp*)	Os últimos *comp* caracteres de *string*	**Right**("CATOLICA",4)	**"LICA"**
InStr(*string1*,*string2*)	Integer correspondente à posição onde é encontada a *string2* dentro de string1. Se não encontrar é devolvido o valor 0	**InStr**("CATOLICA","TO")	**3**
		InStr("CATOLICA","XP")	**0**
InStrRev(*string1*,*string2*)	Tal como *InStr* mas a procura começa a ser feita do fim para o principio	**InStrRev**("CATOLICA","CA")	**7**
LTrim(*string*)	copia de *string* retirando espaços do inicio	**Ltrim(" OLA")**	**"OLA"**
RTrim(*string*)	copia de *string* retirando espaços do fim	**Rtrim("OLA ")**	**"OLA"**
Trim(*string*)	copia de *string* retirando espaços iniciais e espaços finais	**Trim(" OLA ")**	**"OLA"**
Chr(*CodigoChar*)	String contendo o caracter associado ao código *CodigoChar*	**Chr**(65)	**"A"**

2.2 O arrays (tabelas)

Através de um *array* é possível representar, apenas com um identificador(nome), um conjunto de elementos do mesmo tipo. A *string*, por exemplo, não é mais que um array, de características especiais, cujos elementos são do tipo caracter. Embora um array represente um conjunto de elementos, a relevância da informação continua associada a cada um dos seus elementos e não ao todo. É isso que torna a *string* num *array* especial, pois neste caso particular, a relevância da informação está ao nível do conjunto de caracteres e não ao nível do caracter individual.

Declaração de *arrays*

Sintaxe : dim <identificador> (<indices>) as <identificador tipo>

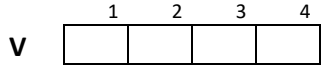
Pela declaração do índice do *array* conseguimos determinar, não só o número de elementos mas também o modo de acedermos a cada um. Cada elemento pode ser acedido fornecendo a posição do elemento dentro do *array*.

Se for indicado apenas um índice dizemos que o *array* é unidimensional (array de uma dimensão também chamado vector).

Se for indicado mais de um índice dizemos que é um *array* multidimensional

Exemplo de declaração de um array unidimensional de 4 elementos inteiros. O primeiro elemento é o de índice 1 e o último de índice 4

Dim v(1 to 4) as integer



Exemplo de atribuição de valores aos elementos:
V(1) = 7 / atribuição do valor 7 ao primeiro elemento
V(2) = 4
V(3) = 3
V(4) = 9

	1	2	3	4
V	**7**	**4**	**3**	**9**

Nos *arrays* multidimensionais cada elemento é identificado referindo qual a posição do elemento em cada uma das dimensões. O número total de elementos é obtido por multiplicação do número de elementos de cada uma das dimensões.
Por exemplo para declarar um *array* de duas dimensões representando uma matriz de 3 x 4
Dim m(1 to 3, 1 to 4) as integer / array de 12 elementos(3 x 4)
Podemos então considerar que o array representa 12 variáveis inteiras, que podem ser referenciadas pelos seguintes nomes :

M(1,1) M(1,2) M(1,3) M(1,4)
M(2,1) M(2,2) M(2,3) M(2,4)
M(3,1) M(3,2) M(3,3) M(3,4)

M

1	2	3	4	
				1
				2
				3

2.3 Registos
O registo permite representar só com um identificador(nome) vários elementos, mesmo que eles sejam de tipo diferente. Em VB é necessário criar primeiro um tipo definido pelo utilizador e depois criar variáveis desse tipo. Vamos supor a criação de um registo que represente informação sobre um aluno. As definições de *tipos definidos pelo utilizador* são feitas no inicio dos módulos, na zona de declarações desse módulo.

```
TYPE TpAluno
  Numero as string*9
  Nome as string
  Nota as integer
END TYPE
```

A partir daqui podem ser declaradas variáveis do tipo *TpAluno*. Não esquecer que os tipos apenas representam as características da informação. A informação existe a nível das variáveis.

```
Dim aluno as TpAluno
```

Os elementos do registo são acedidos à custa do identificador da variável, um ponto e o nome do elemento (campo) em causa.

```
Aluno.numero = "150199123"
Aluno.nome = "Carlos Mendes"
Aluno.nota = 17
```

Ou

```
With aluno
  .numero = "150199123"
  .nome = "Carlos Mendes"
  .nota = 17
end with
```

3 O tipo *variant*

Uma variável cujo tipo não é especificado é do tipo *variant*. Nestas variáveis poderá ser guardada informação de qualquer tipo. O VB tentará determinar o tipo tendo em conta os valores atribuídos à variável e as operações realizadas com ela.

Exemplo de declaração de uma variável do tipo *variant* e atribuição de um valor a essa variável

```
DIM v as variant          / ou apenas DIM v
    V=10
```

Deve evitar, sempre que possível, utilizar *variants.*

- As operações com *variants* são significativamente mais lentas.
- A utilização de *variants* leva a programas menos claros.
- A possibilidade de utilização de operações não válidas para a informação que o *variant* representa, leva ao aparecimento de erros no programa.

II Variáveis

As variáveis são objectos da linguagem, cujo valor pode ser alterado durante a execução do programa.

1 - Características das variáveis

Âmbito

São as zonas do programa onde a variável é conhecida. Ou seja as zonas onde é conhecido o valor da variável. Uma variável que seja declarada dentro de um subprograma (procedimento ou função) só é conhecida dentro desse subprograma. Uma variável declarada a nível do módulo é conhecida em todo o módulo, excepto dentro de subprogramas que declarem, localmente, uma variável com o mesmo nome. Nesse caso e enquanto dentro desse subprograma, dizemos que a global perdeu o âmbito.

Tempo de vida

É o tempo durante o qual a variável está em memória. Variáveis declaradas dentro de subprogramas, nascem ao entrar dentro do subprograma e morrem(deixam de existir) ao sair do subprograma. Variáveis declaradas a nível do módulo continuam a existir ao entrar dentro de um subprograma, embora possam perder temporariamente o âmbito se houver outras declaradas dentro do módulo com o mesmo nome.

Tipo

Representa o conjunto de valores que a variável pode tomar e o conjunto de operações que podem ser feitos sobre esses valores

Valor

É a interpretação do que está guardado no endereço da variável. Por exemplo, o número (0) guardado no endereço de uma variável do tipo inteiro é interpretado como o valor inteiro 0, mas se guardado numa variável do tipo booleano é interpretado como o valor booleano FALSE.

2 – Declaração de variáveis

Em VB não é necessário declarar explicitamente variáveis. Se, durante o código, usar o nome de uma variável, o VB cria automaticamente essa variável**. *No entanto é boa prática de programação declarar explicitamente as variáveis***. O VB tem uma opção para obrigar a que todas as variáveis usadas, sejam explicitamente declaradas. Deve escrever no início de cada módulo:

Option explicit

Esta opção deve ser sempre usada pois leva à construção de programas melhor estruturados e nos quais é mais fácil detectar erros.

Exemplos de declarações :

```
Dim x as integer, y as integer / declaração de duas variáveis do tipo inteiro
Dim s as string                / declaração de uma variável do tipo string
Dim v(1 to 3) as integer       / declaração de um array de 3 elementos do tipo inteiro
```

Uma variável pode existir a nível de projeto (todos os módulos), se for declarada como pública em qualquer um dos módulos do projecto.

Exemplo:

```
Public x as integer / declaração de uma variável inteira pública
```

III - Constantes

Uma constante é um objecto cujo valor não varia durante a execução do programa. As linguagens de alto nível permitem atribuir um valor constante a um identificador e usá-lo em vez daquele sempre que for necessário.

Definição de constantes em VBA

A definição de constantes consiste na associação de um valor constante ao respectivo identificador.

Exemplo de declaração de constantes :

```
const   pi = 3.1415 ,  iva=0.23
const   ano = 2018
const   titulo = "Quadro"
```

Vantagens da utilização de constantes

Há 2 razões principais para usar constantes em vez de utilizar referências directas aos seus valores: 1 Aumento da legibilidade dos programas

É muito mais prático e evidente utilizar referências como "iva" em vez de 0.17 ao longo de um programa; para além de representar o valor, o identificador tem a vantagem de dar também o seu significado.

2 Manutenção mais fácil e mais rápida

Se for necessário alterar o valor de uma constante, a utilização de um identificador permite que a modificação fique limitada à definição da constante, em vez de ter que se alterar cada referência a essa constante ao longo do programa.

IV - Expressões

Uma expressão é composta por operandos e por operadores. Os operandos podem ser» constantes
» variáveis
» expressões
» valores gerados por funções
Os operadores podem ser unários se apenas necessitarem de um operando:
» exemplos: not, -
ou binários, se tiverem dois operandos and, or

Operadores relacionais em VBA

igual a	=
diferente de	<>
maior que	>
menor que	<
maior ou igual a	>=
menor ou igual a	<=

Exemplos:
» expressão1 1<2 ->**true**
» expressão2 2>3 ->**false**
» expressão1 and expressão2 ->**false**
» expressão1 or expressão2 ->**true**
As expressões seguintes não têm significado(uma vez que estão a usar operadores relacionais para estabelecer relações entre tipos incompativeis):

»true =2 false<7 43= "j"
Outros exemplos de expressões
»(a,b,c são variáveis)
»a
»a+b* sqr(c)
»-a+7 / 5

Prioridade dos operadores em VBA (começando pela prioridade mais alta)

- ( )
- Aritméticos
  - ^
  - - (negação)
  - * , /
  - \
  - mod
  - + , -
- &
- Relacionais e (like , is)
- Boleanos
  - not
  - and
  - or
  - xor

V – Estruturas de controle – Selecção

Um programa de computador precisa de ter mecanismos que lhe permitam tomar decisões e executar tarefas baseadas nessas decisões. Um programa deve ter a capacidade de decidir se uma instrução ou grupo de instruções deve ou não ser executado, dependendo de um valor de uma expressão.

1 – A instrução IF

Sintaxe da instrução IF:

```
IF <condição> THEN
        <instruções>
[ELSE                    ‘ parcelas entre [ ] são facultativas, podem ou não existir
        <instruções>]
 END IF
```

Exemplo 1 :

```
IF x=1 THEN
   ‘
   ‘ Este código entre o THEN e o END IF só é executado se x for igual a 1
   ‘
END IF
```

Exemplo 2 (com um if dependendo de outro if) :

```
IF x=1 THEN
  IF y = 1 THEN
        ‘
        ‘ Este código só é executado se x for igual a 1 e y for igual a 1
        ‘
  END IF
END IF
```

Quando uma instrução *IF* usa a clausula *ELSE*, um conjunto de instruções é executado se a condição for verdadeira, sendo executado outro conjunto se a condição for falsa. Em nenhuma ocasião os dois conjuntos de instruções serão executados em simultâneo.
Exemplo :

```
IF x = 1 THEN
  ‘
  ‘ Este código só é executado  se x tiver o valor 1
  ‘
ELSE
  ‘
  ‘ Este código só é executado  se x não tiver o valor 1
  ‘
END IF
```

2 – A instrução CASE
Podem ser tomadas decisões muito complexas usando a instrução IF. O código resultante pode, no entanto, tornar-se pouco claro e difícil de seguir. Existe em VB, outra instrução com a funcionalidade semelhante à do “IF THEN ELSE” mais flexível e permitindo criar código mais claro.
Sintaxe da instrução case :

SELECT CASE <*expressão*>
[**CASE** <lista constantes>
[<*instruções*>]] . . .
[**CASE ELSE**
[<*instruções*>]]

END SELECT

Exemplo :

```
SELECT CASE X
        CASE 0
          ‘
          ‘              este código é executado se x=0
          ‘
        CASE 1
          ‘
          ‘              este código é executado e x=1
          ‘
        CASE 2
          ‘
          ‘              este código é executado se x=2
          ‘
        CASE ELSE
          ‘
          ‘              (esta secção é opcional) o código aqui existente será executado se nenhuma das opções anteriores se verificar
          ‘
END SELECT
```

O computador testa a expressão (no exemplo, o valor da variável X) contra uma lista de constantes. O programa irá executar o primeiro conjunto de instruções, e só esse, que estiver associado à constante cujo valor é igual à expressão. As instruções a seguir ao *CASE ELSE* só serão executadas, se nenhuma das constantes tiver valor igual à expressão.

Exemplo de utilização de uma instrução IF e CASE para classificar uma nota de 0 a 20 nos escalões (MAU,MEDIOCRE,SUFICIENTE,BOM, MUITO BOM)
Supondo que foi declarada uma variável inteira (nota) que contém o valor da nota, e uma variável de tipo string (escalao) que se pretende que fique com informação correspondente ao escalão. Supõe-se também que o valor de nota está efectivamente contido entre 0 e 20.

```
IF nota <=4 THEN
  Escalao = “MAU”
ELSE
  IF nota <=9 THEN
    Escalao = “MEDIOCRE”
  ELSE
    IF nota <=13 THEN
      Escalao = “SUFICIENTE”
    ELSE
      IF nota <=16 THEN
        Escalao = “BOM”
      ELSE
        Escalao = “MUITO BOM”
      END IF
    END IF
   END IF
 END IF
```

```
SELECT CASE nota
  CASE 0 TO 4
    Escalao= “MAU”
  CASE 5 TO 9
     Escalao = “MEDIOCRE”
  CASE 10 TO 13
     Escalao = “SUFICIENTE”
  CASE 14 TO 16
     Escalao = “BOM”
  CASE ELSE
     Escalao = “MUITO BOM”
END SELCT
```

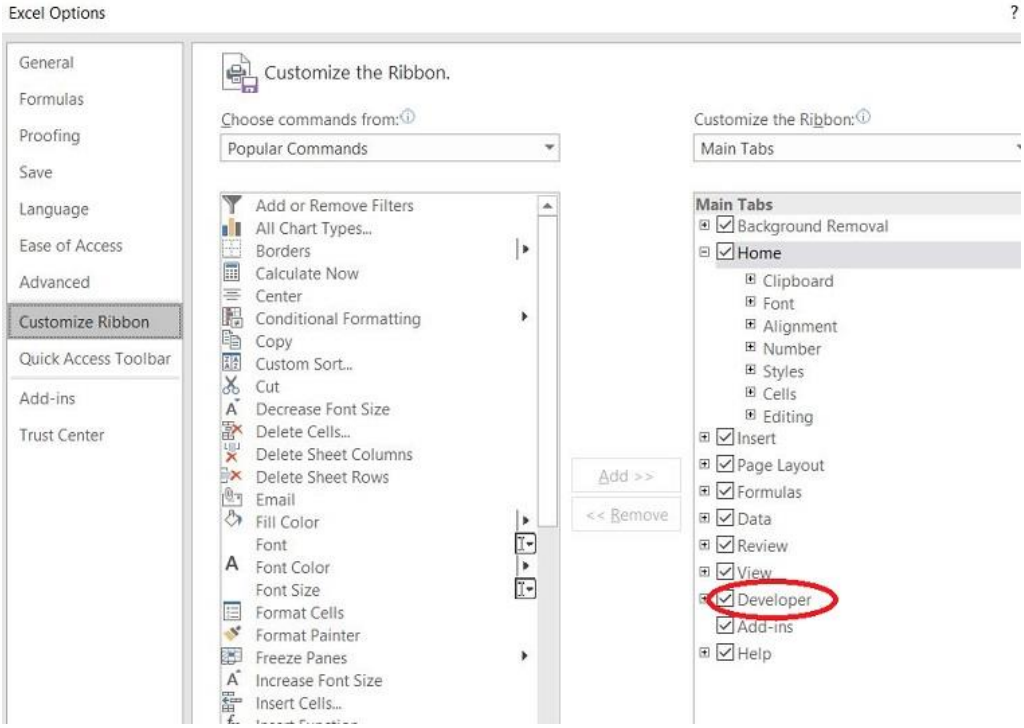
Embora os dois conjuntos de instruções tenham exactamente o mesmo efeito, é fácil notar que a instrução CASE se traduziu num código, muito mais claro e fácil de seguir.

Fazer em Excel :

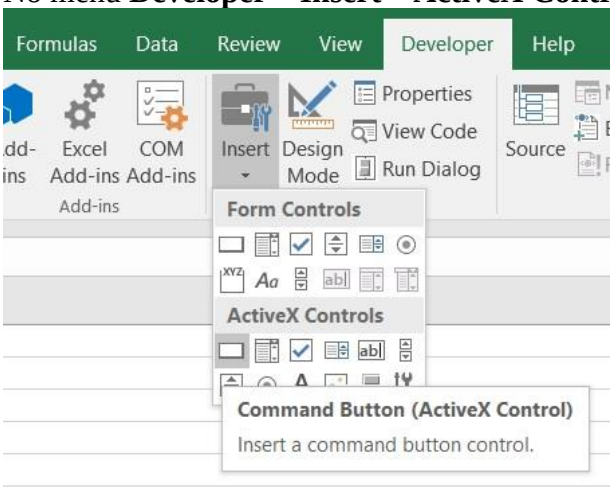
Criar um botão com o nome "ESCALAO".
Quando se carregar neste botão, o programa pedirá ao utilizador para introduzir o valor da nota. Em seguida o programa calcula o escalão e dará uma mensagem mostrando o seu valor.

Resolução (figuras segundo Excel 2007 / 2016):

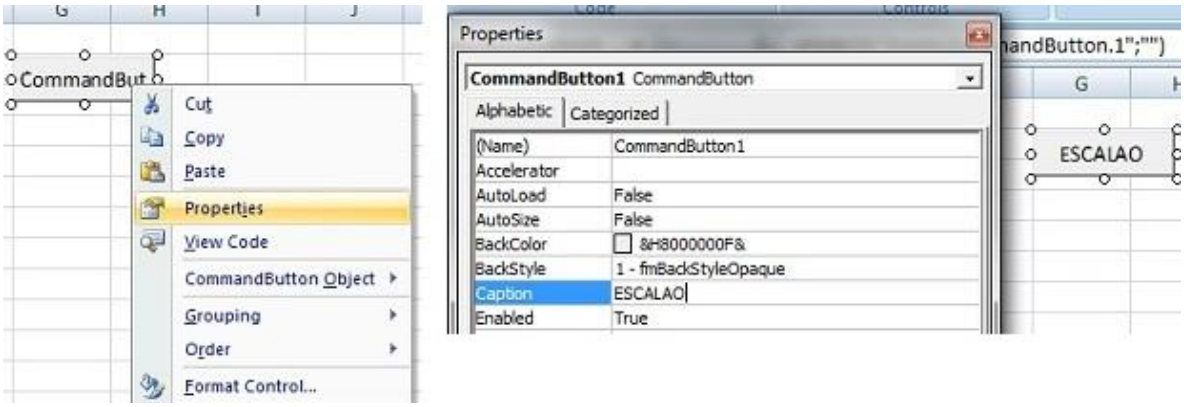
- Se não tiver o menu *Developer* acessivel, vá a **File** – **Excel Options – Customize Ribbon**



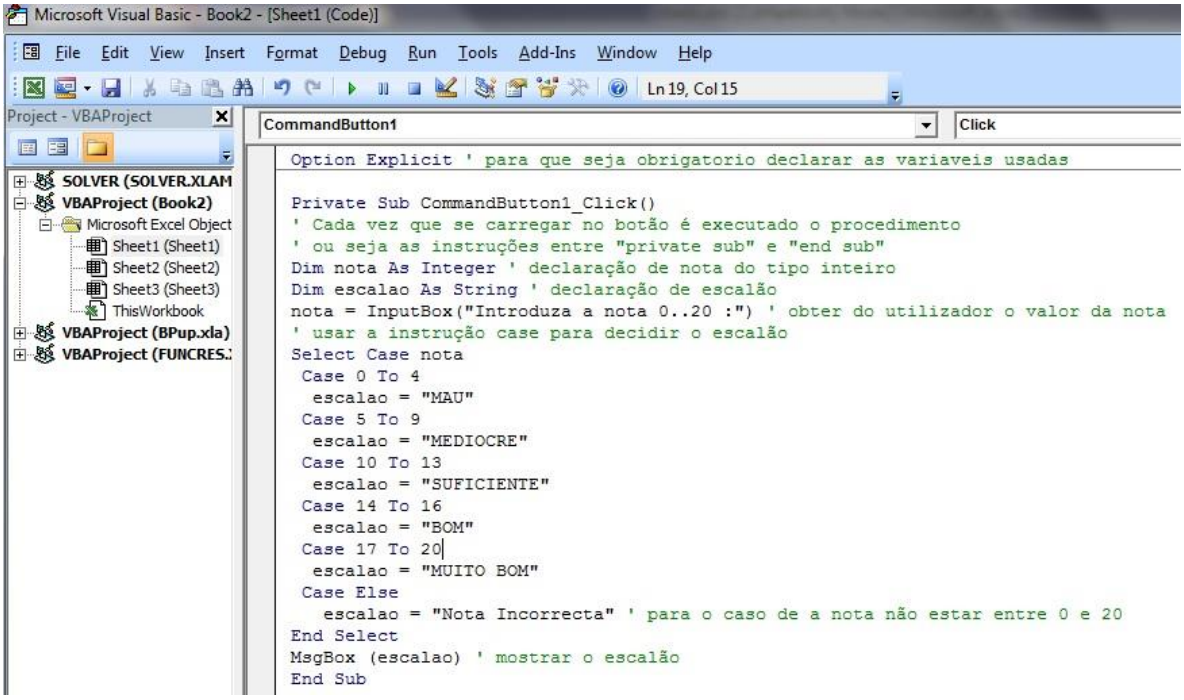
No menu **Developer** – **Insert – ActiveX Controls – Command Button**



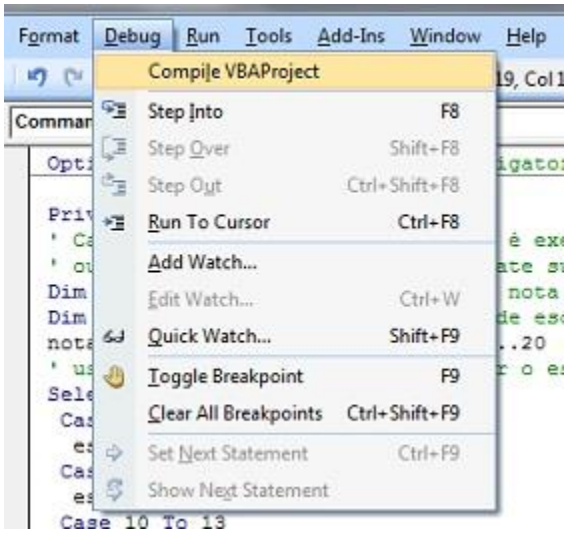
- Faça um click dentro do **Sheet1** para colocar o botão.
- Carregue em cima do botão usando o botão da direita do rato. Escolha a opção *Properties*. Altere a propriedade *Caption* para "ESCALAO". Este passará a ser o nome que aparece escrito no botão.



- Faça duplo click, com o botão da direita do rato, em cima do botão. Isto fará abrir um módulo onde pode ser escrito o seu código. É criado também, automaticamente, o cabeçalho de um procedimento (sub) que será chamado sempre que o utilizador carregar no botão.
- Neste módulo escreva as seguintes instruções (o texto a seguir à plica, a verde, são comentários para esclarecimento do código e não necessita ser passado)



- Depois de escrever o código verifique os erros de sintaxe – **Debug – Compile VBAProject** (emende os erros de sintaxe eventualmente assinalados)



- Na folha excel – **Developer – Design Mode** carregue no esquadro para sair de *design mode*. Cada vez que se carregar no botão será executado o código associado ao mesmo.

VI - Estruturas de controle – Repetição

Estas estruturas permitem repetir um conjunto de instruções, um certo número de vezes. O número de vezes pode ser fixo ou depender de uma condição. Em programação, uma sequência de instruções executada repetidamente é chamada um *ciclo*.

1 – O ciclo FOR

Este tipo de ciclo executa um conjunto de instruções um número fixo de vezes. Deve ser usado quando se conhece, à partida, o número de repetições a ser feita.

Sintaxe : o contido entre [] é opcional

```
FOR <var de controle> = <valor inicial> TO  <valor final> [ STEP <incremento> ]
        <instruções>
NEXT
```

O ciclo é controlado por uma variável numérica (*var de controle*). É feita uma repetição das *instruções* para cada valor da *var de controle*, desde o seu *valor inicial* até ao seu *valor final.* A *var de controle* é incrementada automaticamente, no fim de cada repetição, do valor de *incremento*. Se STEP não for incluído, o incremento toma por defeito, o valor 1.

Exemplos :

```
FOR X = 1 TO 10          / X vai tomar os valores 1,2,3,4.....9,10
  ‘
  ‘ este código vai ser executado 10 vezes
  ‘
NEXT

FOR X = 0 TO 10 STEP 2         / X vai tomar os valores 0,2,4.....10
  ‘
  ‘ este código vai ser executado 6 vezes
  ‘
NEXT

FOR X = 10 TO 1 STEP -1        / X vai tomar os valores 10,9,8,7.....1
  ‘
  ‘ este código vai ser executado 10 vezes
  ‘
NEXT
```

2 – O ciclo DO WHILE ... LOOP

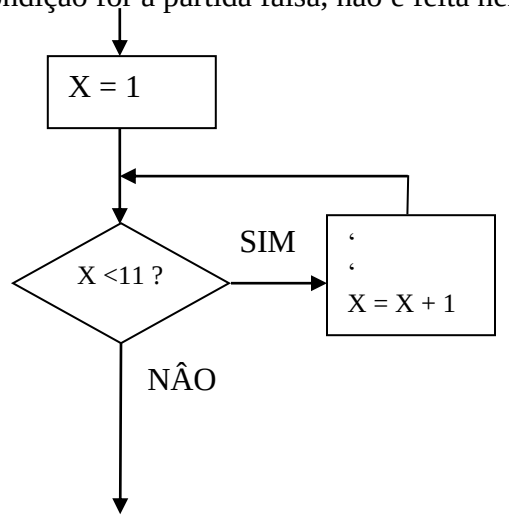
O ciclo *while* repete um conjunto de instruções enquanto uma dada condição for verdadeira. A condição é analisada à cabeça, o que significa que se a condição for à partida falsa, não é feita nenhuma repetição.

Sintaxe :

```
DO WHILE <condição>
        <instruções>
LOOP
```

Exemplo :

```
X = 1
DO WHILE X<11
  ‘
  ‘ este código é executado 10 vezes
  ‘ X = X + 1
LOOP
```



2 – O ciclo DO LOOP ... UNTIL

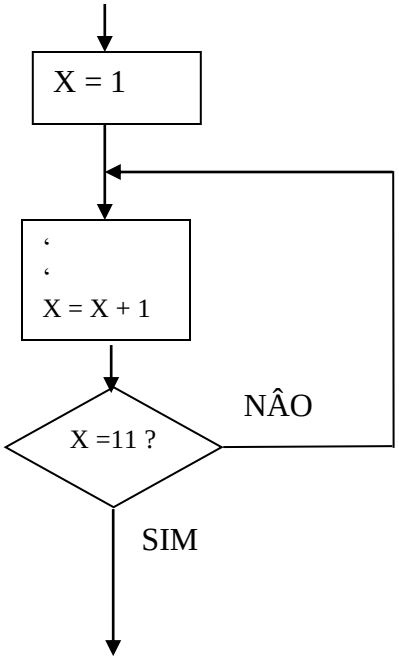
O ciclo *Do Loop Until* repete um conjunto de instruções até que uma condição seja verdadeira. A condição só é analisada no fim de cada repetição pelo que as instruções a repetir são executadas pelo menos uma vez.

Sintaxe :

```
DO
  <instruções>
LOOP UNTIL <condição>
```

Exemplo :

```
X = 1
DO
  ‘
  ‘ este código é executado 10 vezes
  ‘ X = X + 1
LOOP UNTIL X=11
```



Fazer em Excel :

Criar um botão com o nome “DIGITOS”.

Quando se carregar neste botão, o programa pedirá ao utilizador para introduzir um inteiro. Em seguida o programa calcula o número de dígitos desse inteiro, informando o utilizador através duma mensagem.

Resolução :

- No menu **Developer** – **Insert – ActiveX Controls – Command Button**
- Carregue em cima do botão usando o botão da direita do rato. Escolha a opção *Properties*. Altere a propriedade *Caption* para “DIGITO”. Este passará a ser o nome que aparece escrito no botão.

- Faça duplo click, com o botão da direita do rato, em cima do botão. Isto fará abrir um módulo onde pode ser escrito o seu código. É criado também, automaticamente, o cabeçalho de um procedimento (sub) que será chamado sempre que o utilizador carregar no botão.

Neste módulo escreva as seguintes instruções (o texto a seguir à plica são comentários para esclarecimento do código e não necessita ser passado)

```
Option Explicit ' para que seja obrigatorio declarar as variaveis usadas
Private Sub CommandButton1_Click() ' cabeçalho criado automaticamente
‘ Cada vez que se carregar no botão é executado o procedimento
‘ ou seja as instruções entre “private sub” e “end sub”
Dim s As String
Dim x  As Long, aux As Long
Dim dig As Integer
s = InputBox("Introduza um número inteiro : ") ‘ obter o número do utilizador
x = Val(s) ‘converter a string obtida em inteiro
aux = x ‘ guardar o valor obtido noutra variável
dig = 0 ‘ variável que vai contar o número de dígitos
DO
  aux = aux \ 10 ‘ retirar o último dígito ao número
  dig = dig + 1 ‘ contra +1 por cada dígito retirado
LOOP UNTIL aux = 0 ‘ terminar o ciclo quando aux chegar a 0
MsgBox (Str(x) & " tem " & Str(dig) & " digitos") ‘ mostrar número de dígitos
End Sub
```

VII – Subprogramas

Um subprograma é um conjunto de instruções ao qual se deu um nome. Os subprogramas permitem a divisão em vários módulos, cada um dos quais pode ser desenvolvido separada e independentemente dos outros. A interligação dos vários módulos levará a um programa com a resolução total do problema.

1 – Procedimentos

Procedimentos são como pequenos programas independentes e contidos dentro de um programa maior. O objectivo de um *procedimento* é o de realizar uma tarefa especifica.

Sintaxe :

```
SUB <identificador> [ ( < lista de argumentos > ) ]
        <instruções>
 END SUB
```

Um subprograma deve ser estanque, comunicando com o resto do programa apenas através da *lista de argumentos* (parâmetros). Os parâmetros são os canais privilegiados de comunicação com o exterior, para receber ou devolver informação. Aos parâmetros usados na declaração do subprograma dá-se o nome de *parâmetros formais*. Aos parâmetros usados na chamada do subprograma dá-se o nome de *parâmetros actuais*. Na chamada do subprograma, o número de *parâmetros actuais* tem que ser igual ao número de *parâmetros formais*. Cada *parâmetro formal* terá sempre um *parâmetro actual*, do mesmo tipo, que lhe corresponde.

1.1 Parâmetros passados por valor (parâmetros de entrada)

Quando um parâmetro é usado apenas para transmitir informação ao subprograma, dizemos que esse parâmetro é passado por valor. Na altura da chamada, o *parâmetro formal* recebe o valor transmitido pelo respectivo *parâmetro actual*. Em VB, para declarar um parâmetro por valor, é necessário escrever ***BYVAL***, antes do nome desse parâmetro.

Se dentro do subprograma, for alterado o *parâmetro formal*, essa alteração não é transmitida ao *parâmetro actual.*

Exemplo : (procedimento com um parâmetro x passado por valor)

```
SUB teste1 (ByVal x as integer)
  ‘
  ‘
  ‘
END SUB
```

1.2 Parâmetros passados por referencia (parâmetros de entrada/saída)

Quando um parâmetro tem possibilidade de devolver informação ao programa que o chama, dizemos que esse parâmetro é passado por referencia. Na altura da chamada, o *parâmetro formal* recebe (partilha o endereço de memória) do *parâmetro actual*. Em VB os parâmetros são, por defeito, passados por referencia.

Se, dentro do subprograma for alterado o *parâmetro formal*, essa alteração é transmitida ao *parâmetro actual* correspondente.

Exemplo : (procedimento com um parâmetro x passado por referencia)

```
SUB teste2 (x as integer)          / seria equivalente escrever ByRef x as integer
  ‘
  ‘
  ‘
END SUB
```

Um subprograma pode ter declarado qualquer número de parâmetros, sendo cada um deles passado por *valor* ou por *referencia*, conforme o desejado.

Exemplo : (procedimento com 2 parâmetros x, s passados por valor e z passado por feferencia)

```
SUB teste3 (ByVal x as integer,ByVal s as string, ByRef z as integer)
  x= x+ Len(s)
  z= z + x \ 3
END SUB
```

1.3 Chamada de procedimentos

Em VB os procedimentos podem ser chamados de duas maneiras. Por exemplo para chamar o procedimento teste1 : (os parâmetros actuais usados, são apenas exemplos, embora tenham que respeitar o tipo dos formais que lhes correspondem)

```
Call teste1(10)
        Ou
Teste1 10
```

Chamada do procedimento teste3
Considere o seguinte troço de programa :

```
DIM a as integer, c as integer
a=1
c=1
Call teste3(a,"Rui Dias",c)          ' ou  Teste3 a, "Rui Dias", c
Call MsgBox(a)        ' [1] Passagem por valor, x recebe o valor de a, uma alteração em x não se reflecte em a
Call MsgBox(c)        ' [4] Passagem por referencia, z recebe o valor de c, uma alteração em z tem reflexo imediato em c
```

2 – Funções
As funções são quase como procedimentos, com a excepção de devolverem um valor. Assim a função será chamada na perspectiva do valor que vai devolver.
Sintaxe :

```
FUNCTION <identificador>[ ( < lista de argumentos > ) ] AS <identificador tipo>
  <instruções>
 END FUNCTION
```

Como a função devolve um valor, deve ser especificado o tipo do valor devolvido, a seguir à clausula ***AS*.**

Exemplo : (declarar uma função de nome *f1* com dois parâmetros inteiros e que devolve um inteiro)

```
FUNCTION f1 (<byval x AS integer, byval y AS integer) AS integer
        f1 = x + y
END FUNCTION
```

A função devolve o valor que for por último atribuído ao nome da função. No caso do exemplo, a função devolverá o valor correspondente a x + y.

2.1 Chamada de funções
As funções declaradas pelo utilizador são chamadas da mesma forma que qualquer uma das predefinidas em VB.

Para chamar a função F1 : (supondo que a variável C foi declarada como inteiro)

```
C = f1(10,20)
```

A variável ***C*** ficará com o valor 30, uma vez que o *parâmetro formal **X*** recebe o valor 10, o *parâmetro formal **Y*** recebe o valor 20 e a função devolve o valor de ***X* + *Y***.

Fazer em Excel :
Criar uma função que devolva o número de dígitos de um dado inteiro.

Resolução :
- Vá para o Editor do Visual Basic, Menu **Developer – Visual Basic**
- Menu **Insert** – **Module** (só necessário se o seu projecto ainda não tiver nenhum módulo criado)

Neste módulo escreva as seguintes instruções (o texto a seguir à plica são comentários para esclarecimento do código e não necessita ser passado)

```
Option Explicit ' para que seja obrigatorio declarar as variaveis usadas

Function Digito(x as long) as integer
Dim c  As integer
 c = 0 ‘ c vai ser usada para contar os dígitos
 DO
  x = x \ 10 ‘ retirar o último dígito ao número
  c = c + 1 ‘ conta +1 por cada dígito retirado
 LOOP UNTIL x = 0 ‘ terminar o ciclo quando x chegar a 0
 Digito = c ‘ devolve o número de dígitos
End Function
```

Nota : pode usar o mesmo módulo para criar várias funções.

VIII - Objectos Fundamentais do Excel

O Objecto *Application*

O objecto ***Application*** representa todos os objectos da aplicação Excel tais como, *WorkBooks AddIns, Windows...*
Application.ActiveWorkBook ou **ActiveWorkBook** – representa o *WorkBook* activo (seleccionado)
Application.ActiveCell ou **ActiveCell** – representa a celula activa na janela activa (a janela de topo)
Application.Quit – Fecha (termina) o Microsoft Excel
O objecto ***Application*** pode também ser usado para aceder à maioria das funções do Excel

Application.WorksheetFunction.Average(range("A1:A20")) – expressão que devolve a media dos valores contidos na range "A1:A20"
Pode também ser usado apenas **Application.Average(range("A1:A20"))**
A versão sem o ***WorksheetFunction*** é, na maioria dos casos preferível, pois é possível controlar, dentro do VBA, se a função Excel devolve um erro.

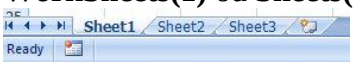
```
Dim v as variant
v = Application.Match("Pedro";A1:A100;0)
if IsError(v) then / Match fail
```

O objecto *WorkBook*

O objecto ***WorkBook*** representa um WorkBook individual dentro do Excel.
Através da colecção ***WorkBooks*** pode aceder a todos os Workbooks abertos, ou individualmente a cada um deles.
WorkBooks(1) – Primeiro WorkBook aberto, **WorkBooks(2)** – Segundo WorkBook aberto
Também se pode referir a um WorkBook pelo seu nome:
WorkBooks("test.xlsx") – representa um WorkBook aberto cujo nome é *test.xlsx*
ThisWorkBook ou **Application.ThisWorkBook** – representa o WorkBook que contém o modulo onde o código está a ser executado
ActiveWorkBook or **Application.ActiveWorkBook** – representa o *WorkBook* activo (seleccionado)

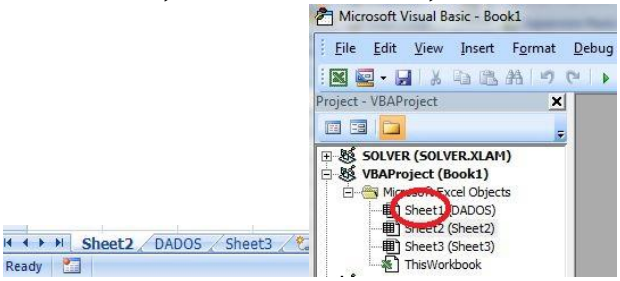
O objecto *WorkSheet*

O objecto ***WorkSheet*** pertence à colecção ***WorkSheets***. A colecção ***WorkSheets*** (ou apenas ***Sheets***) representa todos os WorkSheets num dado WorkBook. Através da colecção ***WorkSheets*** (ou ***Sheets***) pode aceder a todos os WorkSheets, ou individualmente a cada um deles.
Nota: se omitir o WorkBook é assumido que o WorkSheet pertence ao ***ActiveWorkBook*** (**Sheets(1)** ou **ActiveWorkBook.Sheets(1)** é o mesmo)
WorkSheets(1) ou **Sheets(1)** – O primeiro WorkSheet na ordem do tabulador de escolha de WorkSheets.



Pode aceder ao WorkSheet pelo nome:
WorkSheets("Sheet1"), or **Sheets("Sheet1")** – representa o WorkSheet de nome *Sheet1*
WorkBooks(2).Sheets(1) – representa o 1º WorkSheet pertencente ao 2º WorkBook aberto

Também pode aceder a um WorkSheet pelo seu nome de código (code name), este nome "nasce" quando é criado o WorkSheet e é independente da ordem no tabulador, ou do nome do próprio WorkSheet. Mesmo que a ordem mude, ou se altere o nome, o nome de código permanece.



Neste exemplo o tabulador mostra 3 WorkSheets. O primeiro da ordem mudou o nome para "DADOS" e depois trocou de ordem com o segundo. Teremos:

Sheets(1).name ---» "Sheet2", **Sheets(2).name** ---» "DADOS", **Sheet(3).name** ---» "Sheet3"

Estas alterações não irão ter influencia no nome de código. O nome de código pode ser visto no editor do VBA. Assim, o nome de código do WorkSheet "DADOS" continua a ser Sheet1.

Sheet1.name ---» "DADOS", **Sheet2.name** ---» "Sheet2", **Sheet3.name** ---» "Sheet3"

ActiveSheet – representa o WorkSheet activo (aquele cuja janela está no topo)

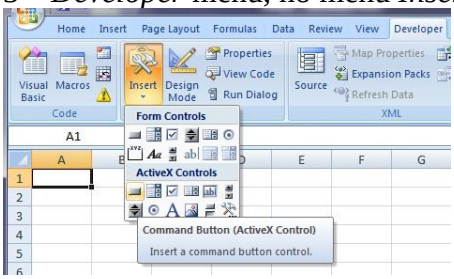
Pode usar a propriedade ***Count*** para aceder ao número de elementos em qualquer colecção. A Expressão ***Sheets.Count*** devolve o número de WorkSheets no ***ActiveWorkBook***

Exercício – Mostrar o nome de todos os WorkSheets no **ActiveWorkBook**

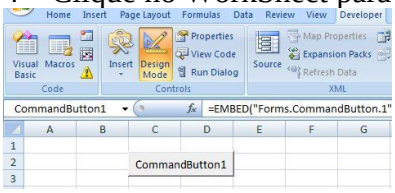
1 – Abra um novo WorkBook

2 – Seleccione *Sheet1*

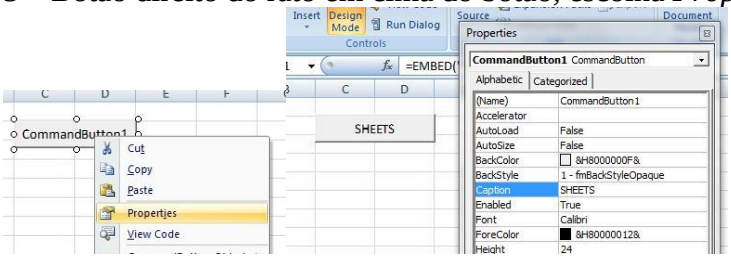
3 – *Developer* menu, no menu *Insert* escolha *Command Button (ActiveX Control)*



4 – Clique no WorkSheet para posicionar o botão

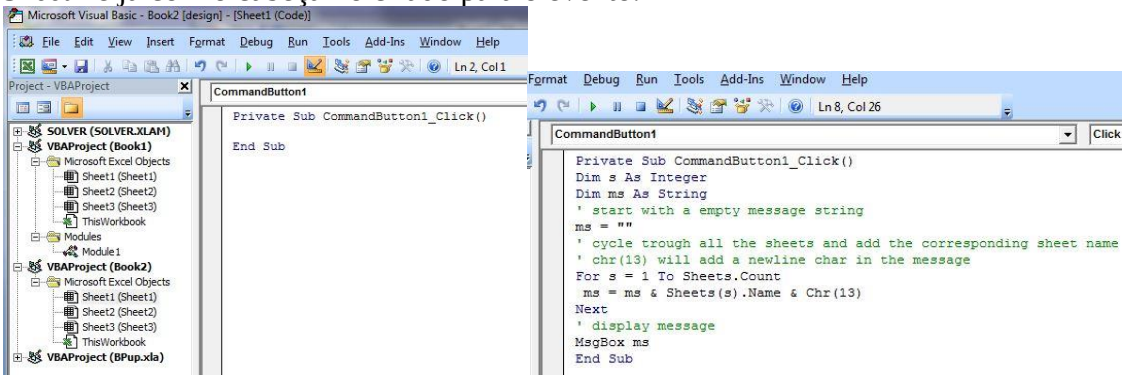


5 – Botão direito do rato em cima do botão, escolha *Properties*, mude *Caption* para "SHEETS"

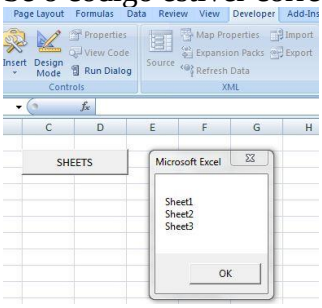


Pode, se o pretender, mudar outras propriedades tais como *color, font, size…*
Deixe a propriedade *name* como está. Esta propriedade é o nome de código do botão, o nome pelo qual nos referimos ao botão no programa.
Ao carregar neste botão serão mostrados os nomes de todos os WorkSheets.

6 – Em *Design Mode* faça um duplo clique no botão. Isto abrirá o editor do VBA, no modulo correspondente ao *Sheet1* e já com o cabeçalho criado para o evento.

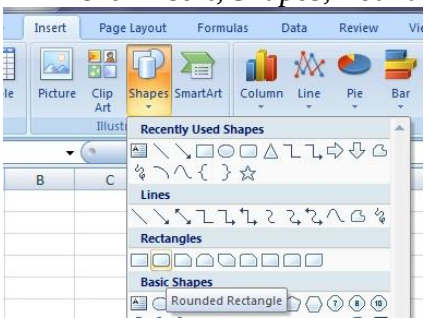


Precisamos agora de escrever o código correcto entre Sub – End Sub.
O texto a verde representa comentários.
7 – Vá a *Debug – Compile VBAProject*, se existirem erros de sintaxe serão assinalados.
Corrija os erros, se existirem, e *compile* novamente.
Se não existirem erros de sintaxe estamos prontos a experimentar o programa.
8 – Volte ao *sheet1*, saia de *Design Mode* e carregue no botão.
Se o código estiver correcto deve ver esta mensagem no ecrã:



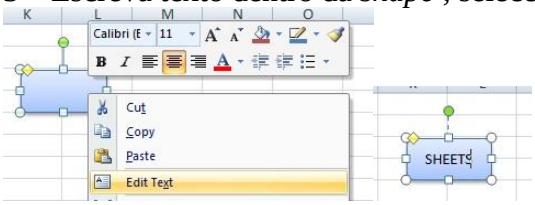
Como utilizar botões mais "interessantes"

1 – Menu *Insert*, *Shapes*, *Rounded Rectangle*



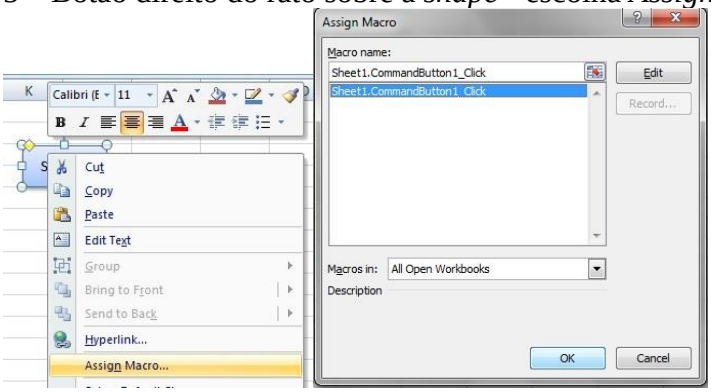
2 – Insira o rectangulo no WorkSheet e vá a *Shape Style* para dar um ar mais "tridimensional" ao botão

3 – Escreva texto dentro da *shape* , seleccionar a *shape*, botão direito do rato - *Edit Text*



4 – Vá até à janela de código do *Sheet1* e remova a palavra « Private » do subprograma (sub) CommandButton1_Click() (Desta forma o subprograma pode ser acedido pelos outros módulos)

5 – Botão direito do rato sobre a *shape* - escolha *Assign Macro*



6 – Escolha a macro *Sheet1.CommandButton1_Click*, a partir de agora o código é corrido sempre que se clicar a shape. O nome da macro só é visível depois de ter efectuado o passo 4.

Os objectos Range, Cells, Rows e Columns

Range e ***Cells*** serão os objectos que, provavelmente, mais irá utilizar. Pode utilizar ***Cells*** para aceder a uma célula individual e ***Range*** para aceder a uma célula ou a um conjunto de células. Usando ***Cells*** está a aceder a um elemento do WorkSheet como se fosse um array de duas dimensões (matriz). A primeira dimensão representa a linha e a segunda a coluna.
Nota: Uma referencia a ***Range*** ou ***Cells***, dentro de um "Sheet Module", sem referir o WorkSheet, significa que são células ou ranges dentro do respectivo WorkSheet (aquele onde estamos a escrever o código). Se for num módulo genérico (não associado a um WorkSheet) então estamos a referir células ou ranges no ***ActiveWorkSheet***

Cells(3,1).value = 7 – célula "A3" recebe o valor 7
value é a propriedade por defeito para o objecto **Range** e **Cells**, podendo assim ser omitido. (**Cells(3,1).value=7** é **Cells(3,1)=7** é exactamente o mesmo)
Pode fazer o mesmo usando o objecto ***Range***
Range("A3") = 7 – celula "A3" recebe o valor 7
Use o objecto ***Range*** para aceder a um grupo de células
Range("A1:C3") = 7 ou **Range("A1","C3") = 7** ou **Range(cells(1,1),cells(3,3)) = 7** – todas as células da range "A1:C3" recebem o valor 7
Range("A1:C3").Clear – limpa (apaga conteúdo) a range "A1:C3" , incluindo formatações
Range("A1:C3").ClearContents – limpa os valores na range "A1:C3", a formatação é mantida
Pode referir explicitamente o WorkSheet
ActiveSheet.Range("A3") = 7 – celula "A3" pertencente ao ***ActiveSheet*** recebe o valor 7
Sheets("Sheet2").Range("A3") = 7 – celula "A3" pertencente ao WorkSheet cujo nome é "Sheet2" recebe o valor 7
Pode referir múltiplas areas:
Range("A1:C3,A5:C7") = 7 – a range "A1:C3" e a range "A5:C7" recebem o valor 7
Até pode usar o objecto ***Range*** para aceder a toda uma linha ou coluna.
Range("C:C").Insert – é inserida uma nova coluna antes da coluna "C" (3)
Pode fazer o mesmo com o usando o objecto ***Columns***
Columns(3).Insert
Range("3:3").Delete – linha 3 é apagada, todas as linhas posteriores sobem
Ou pode usar o objecto ***Rows***
Rows(3).Delete
Também pode usar o objecto ***Range*** para aceder a múltiplas linhas ou colunas
Range("1:1,3:3,5:5").Interior.Colorindex=5 – background das linhas 1,3 e 5 muda para azul
Range("1:7").Font.Size = 14 or **Rows("1:7").Font.Size =14** – o tamanho da fonte nas linhas 1 a 7 passa a ser 14
Número de linhas e colunas numa range:
Range("A1:C7").Rows.Count – devolve o número de linhas na range "A1:C7" (7)
Range("A1:C7").Columns.Count – devolve o número de colunas na range "A1:C7" (3)

Bibliografia

Paul Perry - Teach yourself Visual Basic
SAMS Publishing

J. Pavão Martins - Introdução à Programação usando o PASCAL
McGraw Hill

Microsoft Excel – Help do Visual Basic