# Visual Basic for Applications

- ***Introdução***
  - É uma linguagem de programação baseada na conhecida linguagem BASIC
  - Está concebida para funcionar em conjunto com diferentes aplicações, de forma a potenciar a robustez das mesmas
  - Enquadra-se nos ambientes de programação baseados no processamento de sequência de eventos (*event-driven programming*)
- ***História***
  - Foi inicialmente integrada com o Excel 5 em 1994 e a partir daí a sua expansão para outras aplicações foi gradual
  - Foi com a saída do Office 97 em 1997 que a Microsoft concretizou um dos seus grandes objectivos: ter um ambiente de programação completamente integrado nos seus quatro produtos mais famosos: Word, Excel, Access e PowerPoint
  - Actualmente, o VBA é já por si só um produto independente, que outras companhias podem adoptar e incorporar nas suas aplicações

# Variáveis I

- ***Declaração explícita de variáveis***
  - Declarar uma variável **VAR**: `Dim VAR`
  - Declarar uma variável **VAR** como sendo do tipo **TYPE**: `Dim VAR As TYPE`
- ***Declaração implícita de variáveis***
  - Possibilidade de não declarar variáveis
  - Variáveis não declaradas ou sem declaração de tipo têm por defeito o tipo `Variant`
  - Não permitir o uso de variáveis implícitas: `Option Explicit`
- ***Visibilidade e longevidade de uma variável***
  - `Public`: visível em todos os módulos e durante toda a execução
  - `Private` ou `Dim`: visível dentro do seu módulo e durante toda a execução
  - `Dim`: visível dentro do seu procedimento e durante a sua execução
  - `Static`: visível dentro do seu procedimento e durante toda a execução

# Variáveis II

- *Tipo de variáveis*
  - `Variant` tipo genérico
  - `Boolean` True ou False
  - `Byte` 0 até 255
  - `Integer` -32.768 até 32.767
  - `Long` -2.147.483.648 até 2.147.483.647
  - `Single` -3,402823E38 até -1,401298E-45 (para valores negativos)
    1,401298E-45 até 3,402823E38 (para valores positivos)
  - `Double` -1,7976931348623 2E308 até -4,940656458412 47E-324 (negativos)
    4,940656458412 47E-324 até 1,7976931348623 2E308 (positivos)
  - `Currency` -922.337.203.685.477,5808 até 922.337.203.685.477,5807
  - `Decimal` +/-79.228.162.514.264.337.593.543.950.335 (sem casas decimais)
    +/-7,9228162514264337593543950335 (com casas decimais)
  - `Date` 1 de Janeiro de 100 até 31 de Dezembro de 9999
  - `String` 1 até aproximadamente 2 biliões de caracteres
    65.400 caracteres se tamanho fixo
  - `Type` definido pelo utilizador
  - `Object` referência a objectos

# Variáveis III

- ***Constantes***
  - System-defined constants: `True`; `False`; `Null`; `Empty`; `Nothing`
  - Intrinsic constants (bibliotecas do VBA): `Const LEFT_BUTTON = 1`
  - Symbolic constants: `Const PI = 3,14`
- ***Exemplos***

```
Dim val As Boolean
val = True
Dim aux
aux = 5
aux = “vba”
dummy = 1
Type Automovel
  modelo As String*30
  cilindrada As Integer
End Type
Dim meu_carro As Automovel
meucarro.modelo = “Ferrari”
meu_carro.cilindrada = 3000
```

# Operadores I

- ***Aritméticos***
  - `+` (adição)
  - `-` (subtracção e negação)
  - `*` (multiplicação)
  - `/` (divisão)
  - `\` (divisão inteira)
  - `^` (exponenciação)
  - `Mod` (resto da divisão)
- ***Lógicos***
  - `And` (e lógico)
  - `Or` (ou lógico)
  - `Not` (negação)
  - `Imp` (implicação)
  - `Xor` (ou exclusivo)
  - `Eqv` (equivalência)
- ***Texto***
  - `&` (concatenação)
- ***Relacionais***
  - `=` (igual a)
  - `>` (maior que)
  - `<` (menor que)
  - `<>` (diferente de)
  - `>=` (maior ou igual)
  - `<=` (menor ou igual)
  - `Like` (padrões de texto)
  - `Is` (referência de objectos)

# Operadores II

- ***Precedências***

Maior precedência

| | |
|---|---|
| `^` | exponenciação |
| `-` | negação |
| `*` , `/` | multiplicação e divisão |
| `\` | divisão inteira |
| `Mod` | resto da divisão |
| `+` , `-` | adição e subtracção |
| `&` | concatenação |
| `=` , `>` , `<` , `<>` , `>=` , `<=` , `Like` , `Is` | comparação |
| `And` , `Or` , `Not` , `Imp` , `Xor` , `Eqv` | lógica |

Menor precedência

# Operadores III

- ***Símbolos de comparação***

| | |
|---|---|
| `?` | um qualquer caracter |
| `*` | zero ou mais caracteres |
| `#` | um qualquer dígito |
| `[lista]` | um qualquer na lista |
| `[!lista]` | um qualquer não na lista |

- ***Exemplo***

```
Dim var As String
Dim bool As Boolean
var = “aaaa1111”
bool = var Like “?a[a-z][!A-Z]#*”          ‘bool = True
```

- ***Outros caracteres***

| | |
|---|---|
| ‘ ou `Rem` | comentários |
| `:` | múltiplas instruções na mesma linha |
| `_` | uma instrução em múltiplas linhas |

# Arrays I

- *Declarar arrays*
  - `Dim nome (limite) As tipo`
  - `Dim nome (limite_inferior To limite_superior) As tipo`
  - `Option Base 1`
- *Exemplos*

```
Dim dias(6) As String
dias(0) = “Seg”
...
dias(6) = “Dom”
Dim dias(2 To 8) As String
dias(2) = “Seg”
...
dias(8) = “Dom”
Option Base 1
Dim dias(7) As String
dias(1) = “Seg”
...
dias(7) = “Dom”
```

# Arrays II

- ***Arrays com mais do que uma dimensão***
  - `Dim nome (limite_1,... ,limite_n) As tipo`
- ***Exemplo***

```
Dim matriz(10, 10) As Integer
matriz(0,0) = 0
...
matriz(10,10) = 100
```

- ***Arrays dinâmicos***
  - `ReDim`: redefine os limites de um dado array
  - `Redim Preserve`: redefine os limites de um dado array e preserva os valores nele existentes (é aplicável apenas quando se redefine a última dimensão)
  - `LBound`: devolve o limite inferior de um dado array
  - `UBound`: devolve o limite superior de um dado array

# Arrays III

- *Exemplos*

```
Dim vector()
...
ReDim vector(10)
vector(0) = 10
ReDim vector(UBound(vector) + 10)
aux = vector(0)                                  ‘aux = ?
vector(0) = 100
ReDim Preserve vector(UBound(vector) + 10)
aux = vector(0)                                  ‘aux = 100
ReDim vector(10,10)
vector(0,0) = 1000
ReDim vector(UBound(vector, 1), UBound(vector, 2) + 10)
aux = vector(0, 0)                               ‘aux = ?
```

# Procedimentos

- ***Procedimentos Sub***

```
[Public | Private] Sub nome ([argumentos])
   [...]
   [Exit Sub]
   [...]
End Sub
```

- ***Procedimentos Function***

```
[Public | Private] Function nome ([argumentos]) [As tipo]
   [...]
   [nome = expressão]
   [Exit Function]
   [...]
   [nome = expressão]
End Function
```

# Argumentos I

- ***Declarar argumentos***
  - `[Optional] [ByRef | ByVal] arg[( )] [As tipo] [= valor]`
- ***Exemplo***

```
Function area(comp As Integer, alt As Integer) As Integer
  area = comp * alt
End Function
```

- ***Passar e nomear argumentos***
  - Passar argumentos: `area(5, 4)`
  - Nomear argumentos: `area (alt:= 4, comp:= 5)`
- ***Opções de declaração***
  - `Optional`: declara que o argumento não é obrigatório (esta declaração implica que os restantes argumentos sejam igualmente declarados como `Optional`)
  - `ByRef`: declara que o argumento é passado por referência (o procedimento recebe o próprio endereço da variável passada como argumento)
  - `ByVal`: declara que o argumento é passado por valor (o procedimento apenas recebe o valor da variável passada como argumento)

# Argumentos II

- *Exemplos*

```
Function area(comp As Integer, Optional larg As Integer = 1)
  area = comp * larg
End Function
...
aux = area (5, 4)                                    'aux = 20
aux = area (5)                                       'aux = 5

Function area(Optional comp As Integer = 1, larg As Integer)
  area = comp * larg
End Function                                         'compile error

Function area(Optional comp As Integer = 1,
              Optional larg As Integer = 1)
  area = comp * larg
End Function
...
aux = area (5, 4)                                    'aux = 20
aux = area (5)                                       'aux = 5
aux = area ()                                        'aux = 1
```

# Argumentos III

- *Exemplos*

```
Sub dobro(ByVal var As Integer)
  var = 2 * var
End Sub
...
aux = 10
dobro aux                                   ‘aux = 10
Sub dobro(ByRef var As Integer)
  var = 2 * var
End Sub
...
aux = 10
dobro aux                                   ‘aux = 20
Sub dobro(var As Integer)
  var = 2 * var
End Sub
...
aux = 10
dobro aux                                   ‘aux = 20
```

# Estruturas de decisão I

- *Execução condicional*

```
If condição_1 Then
  [...]
...
[ElseIf condição_n Then
  [...]]
[Else
  [...]]
End If
```

- *Exemplo*

```
If num = 0 Then
  msg = “zero”
ElseIf num < 0 then
  msg = “negativo”
Else
  msg = “positivo”
End If
```

# Estruturas de decisão II

- *Múltiplos testes*

```
Select Case expressão_a_testar
  Case lista_de_expresões_1
    [...]
  ...
  [Case lista_de_expressões_n
    [...]]
  [Case Else
    [...]]
End Select
```

- *Lista de expressões válidas*
  - expressão_1 [, ..., expressão_n}
  - expressão_1 To expressão_2
  - Is operador_de_comparação expressão

- *Exemplo*

```
Select Case num
  Case 0
    msg = “zero”
  Case Is < 0
    msg = “negativo”
  Case Else
    msg = “positivo”
End Select
```

# Código em ciclo I

- ***Ciclos condicionais***

```
Do {While | Until} condição
   [...]
   [Exit Do]
   [...]
Loop

Do
   [...]
   [Exit Do]
   [...]
Loop {While | Until} condição
```

- ***Exemplo***

```
Dim matriz(5,5), i, j
i = 1
Do
   j = 1
   Do
      matriz(i,j) = i + j
      j = j + 1
   Loop While j <= 5
   i = i + 1
Loop Until i > 5
```

- ***Condições de paragem***
  - `While`: executa o ciclo enquanto a condição for verdade
  - `Until`: executa o ciclo enquanto a condição for falsa

# Código em ciclo II

- ***Ciclos numeráveis***

```
For contador = início To fim [Step incremento]
   [...]
   [Exit For]
   [...]
Next [contador]
```

- ***Condições de paragem***
  - **Step**: por defeito o valor de **incremento** é 1
  - Se o **incremento** for positivo ou zero, o ciclo termina assim que **contador** seja maior do que **fim**
  - Se for negativo, termina assim que **contador** seja menor do que **fim**

- ***Exemplo***

```
Dim matriz(5,5), i, j
For i = 1 To 5
  For j = 5 to 1 Step -1
    matriz(i,j) = i + j
  Next j
Next i
```

# Código em ciclo III

- ***Ciclos numeráveis***

```
For Each elemento In colecção
   [...]
   [Exit For]
   [...]
Next [elemento]
```

- ***Condições de paragem***
  - O ciclo termina assim que percorrer todos os elementos em **`colecção`**
  - É útil em situações em que o número de elementos em **`colecção`** é variável ou desconhecido

- ***Exemplo***

```
Dim soma, elem
soma = 0
For Each elem In matriz
    soma = soma + elem
Next elem
```

# Funções básicas I

- *Caixas de mensagem*
  - MsgBox (mensagem)
  - InputBox (mensagem)
- *Conversão de dados*
  - CBool (expressão)
  - CByte (expressão)
  - CInt (expressão)
  - CLng (expressão)
  - CSng (expressão)
  - CDbl (expressão)
  - CCur (expressão)
  - CDate (expressão)
  - CStr (expressão)
  - CVar (expressão)

- *Exemplos*

```
Dim valor
valor = InputBox("Insira valor")
MsgBox("O valor inserido foi "
       & valor & "!")
```

```
CBool(1)                  'True
CBool(0)                  'False
CByte(125.5678)           '126
CInt(2345.5678)           '2346
CCur(543.214588)          '543.2146
CDate("23-2-69")          '23-02-1969
CDate(#2/23/69#)          '23-02-1969
CStr(437.324)             '"437.324"
CDbl(CDate("4:30"))       '0,1875
```

# Funções básicas II

- ***Testes sobre os dados***
  - **IsArray (variável)**: testa se **variável** é um array
  - **IsDate (expressão)**: testa se **expressão** pode ser convertida numa data
  - **IsNumeric (expressão)**: testa se **expressão** é um valor numérico
  - **IsMissing (argumento)**: testa se um **argumento** do tipo **Optional** foi passado ao procedimento corrente
- ***Exemplos***

```
IsDate(#2/12/69#)                                   'True
IsDate("12 de Janeiro de 1998")                     'False
IsNumeric("459.95")                                 'True
IsNumeric("45 Help")                                'False

Function dobro(Optional a As Integer)
   If IsMissing(a) Then dobro = 0 Else dobro = a * 2
End Function
dobro(2)                                            '4
dobro()                                             '0
```

# Funções básicas III

- ***Testes sobre os dados***
  - **IsEmpty (variável)**: testa se **variável** já foi iniciada
  - **IsNull (variável)**: testa se **variável** é **Null**
- ***Exemplos***

```
Dim var1
IsEmpty(var1)                    'True
var1 = Null
IsEmpty(var1)                    'False
var1 = Empty
IsEmpty(var1)                    'True

Dim var2
IsNull(var2)                     'False
var2 = ""
IsNull(var2)                     'False
var2 = Null
IsNull(var2)                     'True
```

# Funções básicas IV

- *Manipulação de strings*
  - `Len (string)`
  - `LCase (string)`
  - `UCase (string)`
  - `Left (string, comprimento)`
  - `Right (string, comprimento)`
  - `Mid (string, início [, comprimento])`
- *Exemplos*

```
Len("Hello World")                    '11
Lcase("Hello World")                  '"hello world"
Ucase("Hello World")                  '"HELLO WORLD"
Left("Hello World", 1)                '"H"
Right("Hello World", 3)               '"rld"
Mid("Hello World", 7)                 '"World"
Mid("Hello World", 7, 2)              '"Wo"
```

# Funções básicas V

- *Manipulação de strings*
  - `LTrim (string)`
  - `RTrim (string)`
  - `Trim (string)`
  - `InStr ([início,] string_geral, string_procura)`
  - `StrComp (string1, string2)`
- *Exemplos*

```
LTrim(“  <-  ->  ”)                    ‘“<-  ->  ”
RTrim(“  <-  ->  ”)                    ‘“  <-  ->”
Trim(“  <-  ->  ”)                     ‘“<-  ->”
InStr(1, “Hello World”, “o”)           ‘5
InStr(6, “Hello World”, “o”)           ‘8
StrComp(“abc”, “abc”)                  ‘0 (string1 = string2)
StrComp(“abc”, “ABC”)                  ‘1 (string1 > string2)
StrComp(“ABC”, “abc”)                  ‘-1 (string1 < string2)
```

# Funções básicas VI

- *Manipulação de datas e horas*
  - Date
  - Time
  - Now
  - Year (data)
  - Month (data)
  - Day (data)
  - Hour (hora)
  - Minute (hora)
  - Second (hora)
  - Weekday (data)
  - DateSerial (ano, mês, dia)
  - TimeSerial (hora, minuto, segundo)

- *Exemplos*

```
Date                    ‘31-12-1999
Time                    ‘15:30:00
Now                     ‘31-12-1999 15:30:00
Year(Date)              ‘1999
Hour(Time)              ‘15
WeekDay(Date)           ‘6 (Domingo é 1)
DateSerial(1999, 12, 31)  ‘31-12-1999
TimeSerial(15, 30, 0)     ‘15:30:00
```