# Visual Basic .NET

## A Linguagem: Operadores, Elementos Condicionais e Laços de Repetição

Professor: Danilo Giacobo

Página pessoal: www.danilogiacobo.eti.br
E-mail: danilogiacobo@gmail.com

# Objetivos da aula

- Quais são os elementos básicos da linguagem?
- Como declarar constantes?
- Como declarar variáveis?
- Como criar enumerações?
- Quais os tipos de dados disponíveis?
- Como converter valores em outros tipos?
- Como declarar vetores?
- Como trabalhar com Strings?
- Quais são os operadores da linguagem?
- Como comentar seu código?
- Como usar as estruturas condicionais IF..ELSE e SELECT CASE?
- Como trabalhar com Laços de Repetição (DO, FOR, WHILE)?
- Como trabalhar com Datas?
- Como trabalhar com valores monetários?

# Introdução

Para simplificar os inúmeros conceitos a serem vistos nesta aula, será usado a parte **Console** da aplicação.

Exemplo de uma aplicação **Console**:

```
Module Module1
    Sub Main()
        System.Console.WriteLine("Olá Turma de Desenvolvimento Visual!")
    End Sub
End Module
```

**Módulos** são designados para armazenar códigos. O procedimento **Main()** indica por onde o programa começa a ser executado. Para exibir um texto na tela (janela do DOS) é usado o método **WriteLine** da classe **System.Console**.

As palavras **Module**, **Sub** e **End** são conhecidas como **Palavras Reservadas** da linguagem VB .NET.

# Declarações

Uma declaração (**statement**) Visual Basic é uma instrução completa. Ela pode conter:

- Palavras reservadas
- Operadores
- Variáveis
- Valores literais
- Constantes
- Expressões

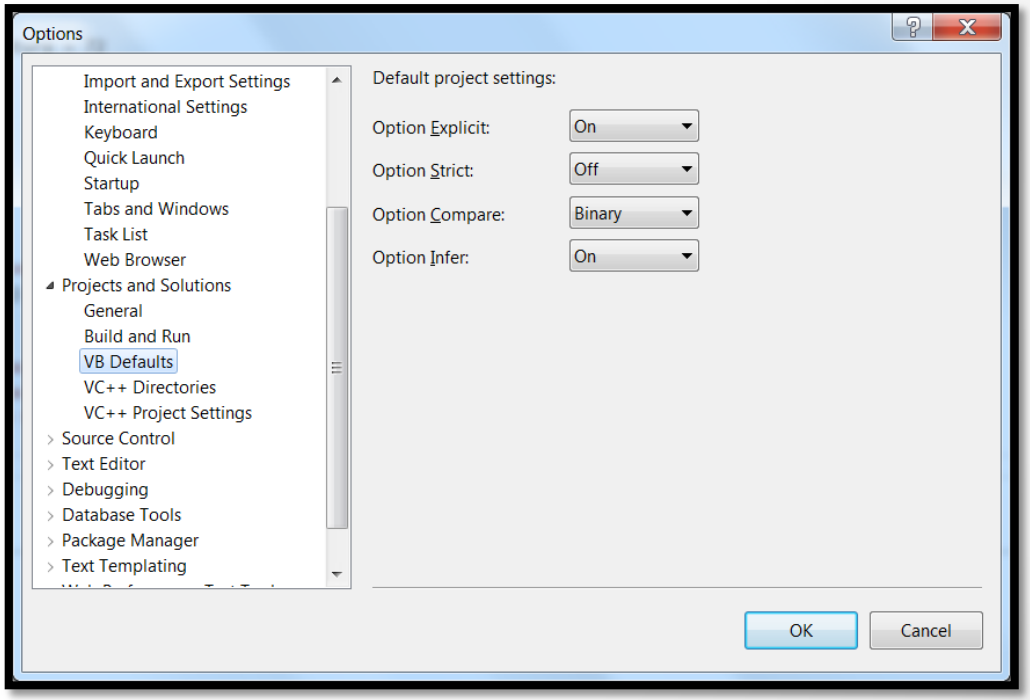
# Declarações

```
1  Module Module1
2      Sub Main()
3          Dim saldo As Single = -500.01
4          Dim temperatura As Integer : temperatura = 72
5          Dim temperatura2 As Integer
6
7          temperatura2 = 1 + 2 + 3 _
8          + 4 + 5 + 6 _
9          + 7 + 8 + 9 + 10
10
11         If (saldo < 0) Then
12             Dim strErro As String
13             strErro = "Seu saldo está negativo!"
14             System.Console.WriteLine(strErro)
15         End If
16
17         System.Console.WriteLine("Temperatura: " & temperatura)
18         System.Console.WriteLine("Temperatura 2: " & temperatura2)
19         System.Console.WriteLine("Esta é uma sentença longa que eu " _
20         & "quero mostrar para o usuário")
21     End Sub
22 End Module
```

# A declaração Option

A declaração **Option** define um número de opções para o restante do seu código. O uso dela no código previne erros de sintaxe e lógicos também. Algumas possibilidades de uso dela são:

- **Option Explicit**
- **Option Compare**
- **Option Strict**




Valores padrão das opções da declaração Option.

# A declaração Option

Você usa a opção **Option** na primeira linha do seu código.

Exemplo:

```
1   Option Explicit Off
2
3   Module Module1
4
5       Sub Main()
6           x = 1    'Variável sem a instrução Dim
7           System.Console.WriteLine("A variável 'x' não foi declarada!")
8           System.Console.WriteLine("Se eu colocar 'On' na opção 'Explicit' esse código não irá compilar!")
9       End Sub
10
11  End Module
```

Tente alterar o valor da opção **Explicit** para **On** e veja o resultado.

# A declaração Imports

A declaração **Imports** permite que importar um namespace para que você não precise qualificar um item quando for utilizar o mesmo. Por exemplo, o procedimento **WriteLine** pertence ao namespace **System.Console**. Eu posso usar ele de duas formas:

```
1  Module Module1
2
3      Sub Main()
4          System.Console.WriteLine("Sem a declaração Imports")
5      End Sub
6
7  End Module
8
```

```
1   Imports System.Console
2
3   Module Module1
4
5       Sub Main()
6           WriteLine("Com a declaração Imports")
7       End Sub
8
9   End Module
10
```

# Constantes

Imagine a seguinte situação: você colocou em seu código milhares de valores numéricos iguais e agora precisa alterar eles. O que você faz? Abre todos os códigos e altera um a um? Uma tarefa hercúlea e fadada ao fracasso. A melhor abordagem nesse caso é a utilização de constantes. Ela funciona como uma variável mas o seu valor não pode ser alterado ao longo da execução do programa.

Como declarar uma constante no código? A sintaxe básica é:

**Const** *nome da constante* **As** *[tipo de dados] = valor inicial*

* O tipo de dados da constante é opcional.

# Constantes - Exemplo

O exemplo abaixo mostra como **declarar** e **usar** uma constante. A **constante** se chama **Pi**. Outras duas variáveis são criadas para exibir o cálculo da área de um círculo qualquer usando a constante declarada anteriormente.

```
1  Imports System.Console
2
3  Module Module1
4
5      Sub Main()
6          Const Pi = 3.14159
7          Dim Raio, Area As Single
8
9          Raio = 2
10         Area = Pi * Raio * Raio
11         WriteLine("Area = " & Str(Area))
12     End Sub
13
14 End Module
```

# Enumerações

Enumerações são usadas quando você precisa de várias constantes que pertencem a um mesmo grupo.

Como declarar uma enumeração no código? A sintaxe básica é:

**Enum** *nome da enumeração* **As** *[tipo de dados]*
*nome do 1° membro = valor inicial*
*nome do 2° membro = valor inicial*
.....
*nome do n° membro = valor inicial*
**End Enum**

* O tipo de dados da enumeração é opcional.

# Enumerações - Exemplo

O exemplo abaixo mostra como **declarar** e **usar** uma **enumeração** para armazenar os dias da semana e depois exibir um dia qualquer a partir da mesma.

```
1  Module Module1
2      Enum Dias_Semana
3          Domingo = 1
4          Segunda = 2
5          Terca = 3
6          Quarta = 4
7          Quinta = 5
8          Sexta = 6
9          Sabado = 7
10     End Enum
11
12     Sub Main()
13         System.Console.WriteLine("Quinta é dia " & Dias_Semana.Quinta & ".")
14     End Sub
15
16 End Module
```

Para usar a constante em uma enumeração você usa o nome da mesma mais o operador “.” e o nome da constante desejada.

# Variáveis

Se você precisa armazenar alguns valores no seu programa para trabalhar você precisa então de variáveis. No Visual Basic .NET é obrigatório declarar todas as variáveis antes de usar as mesmas. Você faz isso com a declaração **Dim**.

Como declarar uma variável no código? A sintaxe básica é:

**Dim** *nome da variável* **As** *[tipo de dados] = [valor inicial]*

* O tipo de dados e o valor inicial da variável são opcionais na declaração.
* O tipo de dados padrão (quando o mesmo não é definido) é Object.

Exemplos:

**Dim** *Numero* **As** *Integer = 1*
**Dim** *Nome* **As** *String = "Danilo Giacobo"*
**Dim** *Endereco* **As** *String*

# Variáveis - Valores padrão

Quando você declara uma variável sem definir um valor inicial para a mesma, a linguagem VB.NET coloca para ela um valor padrão. Dependendo do tipo de dados da variável este valor pode mudar. Exemplo:

```
1  Imports System.Console
2
3  Module Module1
4      Sub Main()
5          Dim b As Byte
6          Dim i As Integer
7          Dim c As Char
8          Dim s As String
9          Dim bo As Boolean
10         Dim data As Date
11
12         WriteLine("Valor padrão de uma variável do tipo Byte: " & b)
13         WriteLine("Valor padrão de uma variável do tipo Integer: " & i)
14         WriteLine("Valor padrão de uma variável do tipo Char: " & c)
15         WriteLine("Valor padrão de uma variável do tipo String: " & s)
16         WriteLine("Valor padrão de uma variável do tipo Boolean: " & bo)
17         WriteLine("Valor padrão de uma variável do tipo Date: " & data)
18     End Sub
19 End Module
```

# Variáveis - Outras considerações

Para criar um novo objeto é usada a palavra **New**. Exemplo:

```
Dim LinkLabel1 As New LinkLabel
```

Você não precisa necessariamente criar o objeto quando o declara. Você pode criar o mesmo apenas na hora em que for usá-lo. Exemplo:

```
Dim LinkLabel1 As LinkLabel
LinkLabel1 = New LinkLabel()
```

Você pode declarar múltiplas variáveis do mesmo tipo sem ter que repetir o nome do tipo de dados. Exemplo:

```
Dim count1, count2 As Integer          'Ambas as variáveis são do tipo inteiro.
```

# Variáveis - Prefixos

Uma boa prática a ser adotada em programação é usar o prefixo do tipo de dados da variável quando for nomear a mesma. Essa técnica é muito útil pois apenas olhando o prefixo da mesma em qualquer parte do código você sabe a qual tipo a mesma pertence.

| Tipo de Dados | Prefixo |
|---|---|
| Boolean | bln |
| Byte | byt |
| Collection object | col |
| Date (Time) | dtm |
| Double | dbl |
| Error | err |
| Integer | int |
| Long | lng |
| Object | obj |
| Single | sng |
| String | str |
| User-defined type | udt |

# Tipos de Dados

Quando você precisar criar uma variável duas perguntas precisam ser respondidas:

**- Qual é o seu nome?**
**- Qual e o seu tipo de dados?**

Lembrar dos tipos de dados de uma linguagem de programação é fácil o problema é saber qual o intervalo de valores que cada uma consegue armazenar.

Os tipos de dados mais utilizados em VB .NET são:

**- Boolean**
**- Date**
**- Double**
**- Integer**
**- String**

# Conversão entre Tipos de Dados

Veja o seguinte código:

```
1   Option Strict On
2
3   Module Module1
4
5       Sub Main()
6           Dim dblNumero As Double
7           Dim intNumero As Integer
8           dblNumero = 3.14159
9           intNumero = dblNumero
10          System.Console.WriteLine("intNumero = " & Str(intNumero))
11          System.Console.WriteLine("intNumero = {0}, dblNumero = {1}", Str(intNumero), Str(dblNumero))
12      End Sub
13
14  End Module
```

O programa acima irá compilar e executar com sucesso?

Quando a opção **Strict** está ligada (**On**) é necessário especificar a o tipo de conversão de dados.

# Conversão entre Tipos de Dados

Veja agora o código corrigido:

```
1  Option Strict On
2
3  Module Module2
4
5      Sub Main()
6          Dim dblNumero As Double
7          Dim intNumero As Integer
8          dblNumero = 3.14159
9          intNumero = CInt(dblNumero)
10         System.Console.WriteLine("intNumero = " & Str(intNumero))
11         System.Console.WriteLine("intNumero = {0}, dblNumero = {1}", Str(intNumero), Str(dblNumero))
12     End Sub
13
14 End Module
```

No próximo slide é apresentada a lista de funções de conversão que você pode usar em seus programas VB .NET.

# Conversão entre Tipos de Dados

- **CBool:** Converte o valor para o tipo de dados **Boolean**.
- **CByte:** Converte o valor para o tipo de dados **Byte**.
- **CChar:** Converte o valor para o tipo de dados **Char**.
- **CDate:** Converte o valor para o tipo de dados **Date**.
- **CDbl:** Converte o valor para o tipo de dados **Double**.
- **CDec:** Converte o valor para o tipo de dados **Decimal**.
- **CInt:** Converte o valor para o tipo de dados **Int**.
- **CLng:** Converte o valor para o tipo de dados **Long**.
- **CObj:** Converte o valor para o tipo de dados **Object**.
- **CShort:** Converte o valor para o tipo de dados **Short**.
- **CSng:** Converte o valor para o tipo de dados **Single**.
- **CStr:** Converte o valor para o tipo de dados **String**.

# Conversão entre Tipos de Dados

Se você não lembrar de um função de conversão de dados em particular, você pode também usar a função **CType**, que permite especificar um tipo de dados a ser convertido. Exemplo:

```
1  Option Strict On
2
3  Module Module3
4
5      Sub Main()
6          Dim dblNumero As Double
7          Dim intNumero As Integer
8          dblNumero = 3.14159
9          intNumero = CType(dblNumero, Integer)
10         System.Console.WriteLine("intNumero = " & Str(intNumero))
11         System.Console.WriteLine("intNumero = {0}, dblNumero = {1}", Str(intNumero), Str(dblNumero))
12     End Sub
13
14 End Module
```

Em termos de performance de código, a função **CType** é mais rápida que as outras funções de conversão apresentadas.

# Conversão entre Tipos de Dados

A linguagem VB .NET possui várias maneiras de converter um tipo de dados em outro e até mesmo formatar a sua exibição na tela.

| Para converter | Use isso |
|---|---|
| Código ASCII do caractere para caractere | Chr |
| Texto para maiúsculo ou minúsculo | Format, LCase, UCase, String.ToUpper, String.ToLower, String.Format |
| Data para um número | DateSerial, DateValue |
| Número decimal pata outras bases | Hex, Oct |
| Número para texto | Format, Str |
| Um tipo de dados para outro | CBool, CByte, CDate, CDbl, CInt, CLng, CObj, CSng, CShort, CStr, Fix, Int |
| Caractere para o código ASCII do caractere | Asc |
| Texto para número | Val |
| Tempo para número serial | TimeSerial, TimeValue |

# Verificando Tipos de Dados

A linguagem VB .NET possui um número de funções utilizadas para verificar um determinado tipo de dados. Você pode usar elas para saber o tipo de um determinado objeto.

| Função | Faz isso |
|---|---|
| IsArray() | Retorna True se a variável fornecida for um Array |
| IsDate | Retorna True se a variável fornecida for uma Data |
| IsDBNull | Retorna True se for passado um valor NULL do banco de dados, nesse caso, um valor System.DBNull |
| IsError | Retorna True se o valor fornecido for um erro |
| IsNumeric | Retorna True se o valor fornecido for um número |
| IsReference | Retorna True se a variável do tipo Object passada como parâmetro não possuir um objeto atribuído; caso contrário retorna False. |

Dica: Você também pode usar a função **TypeOf** retorna o tipo de um determinado objeto.

# Declarando Arrays

**Arrays** são estruturas que permitem você trabalhar com vários dados de um determinado tipo sem precisar criar inúmeras variáveis para este fim. Para criar um array padrão você usa a palavra **Dim** e para criar um do tipo dinâmico você usar a palavra **ReDim**. Todo array em VB .NET começa com em 0 e este índice é sempre do tipo numérico inteiro.

Exemplos:

**Dim** Data(30)
**Dim** Strings(10) **As String**
**Dim** Matriz(20, 40) **As Integer**
**Dim** Intervalos(10, 100)

O primeiro elemento do **array Data** é referenciado no código como **Data(0)**, e o último elemento está em **Data(29)**.

# Declarando Arrays Padrão

Veja no exemplo abaixo como declarar e usar um array em sua forma mais simples:

```
1  Module Module1
2
3      Sub Main()
4          Dim Data(30)
5          Dim Strings(10) As String
6          Dim Matriz(20, 40) As Integer
7          Dim Intervalos(10, 100)
8          Dim MeuArray() = {10, 3, 2}
9
10         Data(0) = 2
11         Data(29) = "Ultimo Valor do Array!"
12         Strings(3) = "Este é um texto!"
13         Matriz(0, 0) = 10
14         System.Console.WriteLine("Quarto elemento do array: " & Strings(3))
15         System.Console.WriteLine("Primeiro elemento da matriz: " & Matriz(0, 0))
16         System.Console.WriteLine("Primeiro elemento do array Data: " & Data(0))
17         System.Console.WriteLine("Ultimo elemento do array Data: " & Data(29))
18         System.Console.WriteLine("Elementos de MeuArray: {0}, {1} e {2}.", MeuArray(0), MeuArray(1), MeuArray(2))
19     End Sub
20
21 End Module
```

**Dica:** Você pode inicializar um array com valores sem precisar especificar seu tamanho (veja a linha 8).

# Declarando Arrays Dinâmicos

Veja pode usar a declaração **Dim** para declarar um array dinâmico. Para isto basta deixar os parênteses vazios. Arrays dinâmicos podem ser dimensionados ou redimensionados com a declaração **ReDim**. A sintaxe dessa instrução é:

**ReDim** [Preserve] **As** *nome da variável(nova dimensão)*

Você usa a palavra **Preserve** para manter os dados no array quando você troca o tamanho do mesmo.

Você pode usar a função **UBound** para encontrar o **limite sup**erior de um array. Essa função é útil para percorrer os elementos de um array usando um laço de repetição.

# Declarando Arrays Dinâmicos

Vamos a um exemplo:

```
1  Module Module1
2
3      Sub Main()
4          Dim Dinamico() As String
5          ReDim Dinamico(10)
6          Dinamico(0) = "Texto 0"
7
8          'Preciso de mais espaço!
9          ReDim Dinamico(100)
10         Dinamico(50) = "Texto 50"
11
12         Dim Tamanho = UBound(Dinamico)
13
14         System.Console.WriteLine("Elemento da posicao 0: " & Dinamico(0))
15         System.Console.WriteLine("Elemento da posicao 50: " & Dinamico(50))
16         System.Console.WriteLine("Tamanho do Vetor: " & Tamanho)
17     End Sub
18
19 End Module
```

Quando este código é executado o que aparece na tela? O programa está exibindo os valores do array corretamente?

# Trabalhando com Strings

Para trabalhar com dados que envolvem palavras, frases e afins é necessário usar a classe **String** do namespace **System**. Essa classe contém diversas funções para tratamento deste tipo de dado.

Para **declarar** uma **String** você pode usar qualquer uma das seguintes formas:

**Dim** strTexto **As String**

**Dim** MeuTexto **As String** = "Bem-vindo ao Visual Basic .NET"

Exemplos de funções:

- Para dividir um texto em partes menores você pode usar as funções **Left**, **Mid** e **Right**.
- Para encontrar a quantidade de caracteres de um texto você usa a função **Len**.

# Trabalhando com Strings

No exemplo abaixo é utilizada a função **ToUpper** para converter um texto em letras maiúsculas e uma outra função similar chamada **UCase** para fazer o mesmo processo. A diferença entre as duas é que a primeira pertence a classe **String** e a outra é herdada de versões anteriores da linguagem VB.

```
1   Option Strict On
2
3   Module Module1
4
5       Sub Main()
6           Dim strTexto1 As String = "bem-vindo ao visual basic"
7           Dim strTexto2 As String
8           Dim strTexto3 As String
9           strTexto2 = UCase(strTexto1)
10          strTexto3 = strTexto1.ToUpper
11          System.Console.WriteLine(strTexto1)
12          System.Console.WriteLine(strTexto2)
13          System.Console.WriteLine(strTexto3)
14      End Sub
15
16  End Module
```

# Trabalhando com Strings

No exemplo abaixo é utilizada a função **Substring** para obter parte de um texto e uma outra função similar chamada **Mid** para fazer o mesmo processo. A diferença entre as duas é que a primeira pertence a classe **String** e a outra é herdada de versões anteriores da linguagem VB.

```
1  Module Module1
2
3      Sub Main()
4          Dim strTexto1 As String = "Oi, olhe aqui!"
5          Dim strTexto2 As String
6          Dim strTexto3 As String
7          strTexto2 = Mid(strTexto1, 5, 4)
8          strTexto3 = strTexto1.Substring(4, 4)
9          System.Console.WriteLine(strTexto1)
10         System.Console.WriteLine(strTexto2)
11         System.Console.WriteLine(strTexto3)
12     End Sub
13
14 End Module
```

# Convertendo Strings para Números

É comum na linguagem Visual Basic ter que converter valores que são números para texto e vice-versa. A função **Str** converte um **número** para a representação **textual** do mesmo e a função **Val** converte uma **String** para a sua representação **numérica**. Exemplo:

```
1  Module Module1
2
3      Sub Main()
4          Dim strTexto1 As String = "1234"
5          Dim intValor1 As Integer
6          intValor1 = Val(strTexto1)
7          strTexto1 = Str(intValor1)
8          System.Console.WriteLine(intValor1)
9          System.Console.WriteLine(Trim(strTexto1))
10     End Sub
11
12 End Module
```

**Dica:** Você pode usar também o método **Format** e a função **String.Format** para exibir números e textos formatados.

# Convertendo Caracteres

Os caracteres que um programa armazena internamente são guardados como caracteres **Unicode**. Por exemplo, o código **65** representa o caractere **'A'**. Você pode converter um código de um caractere para a sua representação real e vice-versa fazendo uso das funções **Asc** e **Chr**. Exemplo:

```
1  Module Module1
2
3      Sub Main()
4          Dim Codigo As Integer = 120
5          Dim Caractere As Char = "D"
6
7          System.Console.WriteLine("Caractere representado pelo codigo {0}: {1}.", Codigo, Chr(Codigo))
8          System.Console.WriteLine("Codigo ASCII do caractere '{0}': {1}.", Caractere, Asc(Caractere))
9      End Sub
10
11 End Module
```

# Operadores Aritméticos

Os **operadores aritméticos** da linguagem VB são:

- ^ (Exponenciação)
- * (Multiplicação)
- / (Divisão)
- \ (Divisão Inteira)
- **Mod** (Módulo)
- **+** (Adição)
- **-** (Subtração)

```
1  Module Module1
2
3      Sub Main()
4          Dim intVariavel1 As Integer = 1234
5          Dim intVariavel2 As Integer = 2345
6          Dim intVariavel3 As Integer
7
8          intVariavel3 = intVariavel1 + intVariavel2
9
10         System.Console.WriteLine("{0} + {1} = {2}.", _
11                                  intVariavel1, intVariavel2, intVariavel3)
12     End Sub
13
14 End Module
```

# Operadores de Atribuição

Os **operadores de atribuição de valores** da linguagem VB são:

- **=** (Atribuição)
- **^=** (Exponenciação seguida de atribuição)
- ***=** (Multiplicação seguida de atribuição)
- **/=** (Divisão seguida de atribuição)
- **\=** (Divisão Inteira seguida de atribuição)
- **+=** (Adição seguida de atribuição)
- **-=** (Subtração seguida de atribuição)
- **&=** (Concatenação seguida de atribuição)

```
1  Module Module1
2      Sub Main()
3          Dim intVariavel1 As Integer = 10
4          Dim intVariavel2 As Integer = 4
5
6          intVariavel1 /= 3
7          intVariavel2 ^= 2
8
9          System.Console.WriteLine("intVariavel1 = {0}.", intVariavel1)
10         System.Console.WriteLine("intVariavel2 = {0}.", intVariavel2)
11     End Sub
12 End Module
```

# Operadores de Comparação

Considere a expressão: **5 > 4**
- Os valores em verde são chamados de **operandos**
- O sinal em vermelho é chamado de **operador de comparação**

Os **operadores de comparação de valores** da linguagem VB são:

- **<** (Menor que) - **True** se **operando 1** for **menor** que o **operando 2**
- **<=** (Menor ou igual a) - **True** se **operando 1** for **menor ou igual** ao **operando 2**
- **>** (Maior que) - **True** se **operando 1** for **maior** que o **operando 2**
- **>=** (Maior ou igual a) - **True** se **operando 1** for **maior ou igual** ao **operando 2**
- **=** (Igual a) - **True** se **operando 1** for **igual** ao **operando 2**
- **<>** (Diferente) - **True** se **operando 1** for **diferente** do **operando 2**
- **Is** - **True** se duas referências de objetos se referirem ao mesmo objeto
- **Like** - Verifica se uma sequencia corresponde a um padrão

# Operadores Lógicos

**Operadores lógicos** são operadores que trabalham apenas com **valores lógicos**, isto é, **verdadeiro** e **falso**.

Os **operadores lógicos** da linguagem VB são:

- **And** (e)
  - **True** se ambos os operandos forem verdadeiros;
  - **False** se um dos operandos for falso.
- **Or** (ou)
  - **False** se ambos os operandos forem falsos;
  - **True** se um dos operandos for verdadeiro.
- **Not** (não)
  - **False** se o operando for verdadeiro;
  - **True** se o operando for falso.

# Precedência dos Operadores

Veja o código abaixo.
Qual vai ser o valor da média depois do cálculo?

```
1  Module Module1
2      Sub Main()
3          Dim intNota1, intNota2, intNota3, intNumeroEstudantes As Integer
4          Dim dblMedia As Double
5
6          intNota1 = 60
7          intNota2 = 70
8          intNota3 = 80
9          intNumeroEstudantes = 3
10         dblMedia = intNota1 + intNota2 + intNota3 / intNumeroEstudantes
11
12         System.Console.WriteLine("Media = " & dblMedia)
13     End Sub
14 End Module
```

**Média é igual a 70 ou 156,666666667?**

# Comentários

De uma maneira geral, você deve adicionar comentários ao seu código para torná-lo fácil de ler por você e por outros programadores. Comentários em VB começam com o caractere de apóstrofo (‘) e fazem com que o compilador ignore o que vem depois dele na linha de código. Exemplo:

```
1  Module Module1
2      'Procedimento Main: Por onde o código é iniciado
3      Sub Main()
4          'Declaração de variáveis
5          Dim intVariavel
6
7          'Atribuição de valores
8          intVariavel = 10
9
10         'Exibição de valores na tela do Console
11         System.Console.Write("Valor armazenado na variavel intVariavel: {0}", intVariavel)
12     End Sub
13 End Module
```

**Dica:** No Visual Studio você pode selecionar as linhas a serem comentadas/descomentadas e clicar no botão para a ferramenta fazer esta tarefa para você. As teclas de atalho são: Ctrl+C+E (comentar) e Ctrl+C+U (descomentar).

# Estrutura Condicional - If...Else

A sintaxe e o funcionamento da estrutura **If...Else** na linguagem VB .NET é:

**If** *condição* **Then**
    [comandos]
[**ElseIf** *condição-n* **Then**
    [comandos] ...]
[**Else**
    [comandos]]
**End If**

* O que está entre [] é opcional.

Importante:

O operador de comparação de valores na linguagem VB .NET é o "=".

# Estrutura Condicional - If...Else

Um exemplo que lê e testa um valor inteiro e mostra uma mensagem usando a estrutura condicional **If...Else** é mostrado abaixo:

```
1  Module Module1
2      Sub Main()
3          Dim intNumero
4          System.Console.Write("Entre com um número: ")
5          intNumero = Val(System.Console.ReadLine())
6          If intNumero = 1 Then
7              System.Console.WriteLine("Obrigado.")
8          ElseIf intNumero = 2 Then
9              System.Console.WriteLine("Está bom.")
10         ElseIf intNumero = 3 Then
11             System.Console.WriteLine("Muito grande.")
12         Else
13             System.Console.WriteLine("Não é um número.")
14         End If
15     End Sub
16 End Module
```

**Dica:** Você pode usar as palavras reservadas **TypeOf** e **Is** para verificar o tipo de dados de um objeto na cláusula **If** desta forma:

```
If (TypeOf Err.GetException() Is OverflowException) Then
    System.WriteLine("Overflow error!")
End If
```

# Estrutura Condicional - Select Case

A sintaxe e o funcionamento da estrutura **Select Case** na linguagem VB .NET é:

**Select Case** *expressão de teste*
    [**Case** *expressão-n*
        [*comandos-n*]] ...
    [**Case Else**
        [*comandos-else*]]
**End Select**

* O que está entre [] é opcional.

Importante:

A estrutura Select Case é recomendada quando você quer testar o valor de uma variável contra várias condições sem ter que escrever vários elementos If...Else aninhados.

# Estrutura Condicional - Select Case

Um exemplo que lê e testa uma variável do tipo inteiro e mostra uma mensagem usando a estrutura condicional **Select Case** é mostrado abaixo:

```
1  Module Module1
2      Sub Main()
3          Dim intNumero
4          System.Console.Write("Entre com um número: ")
5          intNumero = Val(System.Console.ReadLine())
6          Select Case intNumero
7              Case 1
8                  System.Console.WriteLine("Obrigado.")
9              Case 2
10                 System.Console.WriteLine("Está bom.")
11             Case 3
12                 System.Console.WriteLine("OK.")
13             Case 4 To 7
14                 System.Console.WriteLine("Entre 4 e 7.")
15             Case Is > 7
16                 System.Console.WriteLine("Definitivamente grande.")
17             Case Else
18                 System.Console.WriteLine("Não é um número.")
19         End Select
20     End Sub
21 End Module
```

# Estrutura Condicional - IIf

A função **IIf** avalia uma expressão (se é verdadeira ou falsa) e retorna um valor do tipo **Object** de acordo com o resultado avaliado. A sua sintaxe é:

**IIf**(*expressão, parte verdadeira, parte falsa*)

Exemplo:

```
1  Module Module1
2      Sub Main()
3          Dim intNumero = 2, intValorAbsoluto As Integer
4
5          intValorAbsoluto = IIf(intNumero < 0, -1 * intNumero, intNumero)
6
7          System.Console.WriteLine("Valor absoluto: " & intValorAbsoluto)
8      End Sub
9  End Module
```

**Dica:** Você pode usar o **operador de negação (-)** para mudar a expressão **-1 * intNumero** para **-intNumero** apenas.

# Estrutura Condicional - Choose

Você pode usar a função **Choose** para retornar um valor de um número de escolhas baseado em um índice. A sua sintaxe é:

**Choose**(*índice, escolha-1* [, *escolha-2*, ... [, *escolha-n*]])

* O que está entre [] é opcional.

Exemplo:

```
1  Module Module1
2      Sub Main()
3          Dim intID As Integer = 2
4          Dim strNome As String
5
6          strNome = Choose(intID, "Danilo", "Daniel", "Fernanda")
7
8          System.Console.WriteLine("Nome escolhido: " & strNome)
9      End Sub
10 End Module
```

# Estrutura de Repetição - Do

O laço de repetição (loop) **Do** continua executando os comandos que estão dentro do seu bloco enquanto uma condição for verdadeira (**While**) ou até que uma condição seja falsa (**Until**). Ele possui duas sintaxes:

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

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

* O que está entre [] é opcional e o que está entre {} significa que você deve escolher um.

**Dica:** Você pode usar o comando **Exit Do** para interromper um loop a qualquer momento.

# Estrutura de Repetição - Do

O exemplo abaixo mostra um programa que continua a exibir uma mensagem "O que eu devo fazer?" até que o usuário digite a palavra "Pare". A função **UCase** é usada para ignorar se a pessoa digita a palavra em letras maiúsculas ou minúsculas.

```
1  Module Module1
2      Sub Main()
3          Dim strPalavra As String = ""
4          Do Until UCase(strPalavra) = "PARE"
5              System.Console.WriteLine("O que eu devo fazer?")
6              strPalavra = System.Console.ReadLine()
7          Loop
8      End Sub
9  End Module
```

**Dica:** A segunda forma do loop Do assegura que o corpo do loop será executado ao menos uma vez.

# Estrutura de Repetição - For

O laço de repetição (loop) **For** é o laço de repetição mais popular. Ele precisa de um índice para controlar o número de iterações realizadas. A sua sintaxe é:

```
For índice = inicio To fim [Step passo]
    [comandos]
    [Exit For]
    [comandos]
Next [índice]
```

* O que está entre [] é opcional.

**Dicas:**

**-** Você pode usar o comando **Exit For** para interromper um loop a qualquer momento.

- Se você não informar o valor de "passo" o padrão para o mesmo então será **1**.

# Estrutura de Repetição - For

O exemplo abaixo mostra um programa que exibe uma frase 4 vezes utilizando o laço de repetição **For**. Ele mostra também a utilização do controle de passos (*steps*) para exibir um número menor de vezes a mesma mensagem.

```
1  Module Module1
2      Sub Main()
3          Dim intIndice As Integer
4
5          'Sem a declaração Step
6          For intIndice = 0 To 3
7              System.Console.WriteLine("Olá Turma de Desenvolvimento Visual!")
8          Next intIndice
9
10         'Com a declaração Step
11         For intIndice = 0 To 3 Step 2
12             System.Console.WriteLine("Olá Turma de Desenvolvimento Visual!")
13         Next intIndice
14     End Sub
15 End Module
```

# Estrutura de Repetição - For Each...Next

O laço de repetição (loop) **For Each..Next** é usado para percorrer elementos em um array. Você não precisa se preocupar com índices ou com o tamanho do array. A sua sintaxe é:

```
For Each elemento In grupo
    [comandos]
    [Exit For]
    [comandos]
Next [elemento]
```

* O que está entre [] é opcional.

**Dicas:**

**-** Você pode usar o comando **Exit For** para interromper um loop a qualquer momento.

# Estrutura de Repetição - For Each...Next

O exemplo abaixo mostra um programa que exibe todos os elementos de um determinado array.

```
1  Module Module1
2      Sub Main()
3          Dim intGrupo(3), intElemento As Integer
4
5          intGrupo(0) = 0
6          intGrupo(1) = 1
7          intGrupo(2) = 2
8          intGrupo(3) = 3
9
10         For Each intElemento In intGrupo
11             System.Console.WriteLine(intElemento)
12         Next intElemento
13     End Sub
14 End Module
```

# Estrutura de Repetição - While

O laço de repetição (loop) **While** continua a sua execução enquanto a condição permanecer **verdadeira**. Se a condição nunca se tornar **falsa** ele vai ser executado eternamente causando famoso **"loop infinito"**. A sua sintaxe é:

```
While condição
    [comandos]
End While
```

* O que está entre [] é opcional.

# Estrutura de Repetição - While

O exemplo abaixo mostra um programa que faz uso do laço de repetição **While** para fazer uma contagem simples de valores.

```
1  Module Module1
2      Sub Main()
3          VerificaWhile()
4      End Sub
5
6      Sub VerificaWhile()
7          Dim intContador As Integer = 0
8          Dim intNumero As Integer = 10
9
10         While intNumero > 6
11             intNumero -= 1
12             intContador += 1
13         End While
14
15         MsgBox("O loop executou " & intContador & " vezes.")
16     End Sub
17 End Module
```

# A declaração With

A declaração não é um laço de repetição mas pode funcionar como um. Ele é útil em casos em que você precisa acessar várias propriedades de um mesmo objeto sem ter que escrever várias vezes o nome do mesmo. A sua sintaxe é:

```
With objeto
    [comandos]
End With
```

* O que está entre [] é opcional.

Exemplo:

```
1  Public Class Form1
2      Private Sub Form1_Load(sender As Object, e As EventArgs) Handles Me.Load
3          Dim txtCampo As New TextBox
4
5          With txtCampo
6              .Height = 1000
7              .Width = 3000
8              .Text = "Bem-vindo ao Visual Basic"
9          End With
10     End Sub
11 End Class
```

# Trabalhando com o namespace Math

No **namespace System.Math** estão os métodos que facilitam o cálculo de várias expressões matemáticas, desde arredondamento de valores e cálculo da raiz quadrada até funções matemática usadas em astrofísica.

| Função | Descrição |
|---|---|
| Abs | Calcula o valor absoluto de um número. |
| Atan | Produz um valor duplo contendo o ângulo cuja tangente é o número dado. |
| Cos | Calcula o cosseno de um determinado ângulo. |
| Exp | Produz um valor duplo contendo e (a base dos logaritmos naturais) elevado à potência dada. |
| Log | Calcula o logaritmo de um determinado número. |
| Round | Calcula o valor arredondado de um número. |
| Sign | Produz um valor indicando o sinal de um número (1: positivo, -1: negativo). |
| Sin | Calcula o seno de um determinado ângulo. |
| Sqrt | Calcula a raiz quadrada de um número. |
| Tan | Calcula a tangente de um determinado ângulo. |

# Trabalhando com o namespace Math

O programa abaixo mostra algumas funções matemáticas sendo utilizadas.

```
1  Imports System.Math
2  Imports System.Console
3
4  Module Module1
5      Sub Main()
6          Dim dblNumero As Double = 10.55
7
8          WriteLine("Raiz quadrada de {0} = {1}.", dblNumero, Sqrt(dblNumero))
9          WriteLine("Valor {0} arredondado = {1}.", dblNumero, Round(dblNumero))
10         WriteLine("Logaritmo de {0} = {1}.", dblNumero, Log(dblNumero))
11         WriteLine("Sinal indicativo de {0} = {1}.", dblNumero, Sign(dblNumero))
12     End Sub
13 End Module
```

**Dica:** Praticamente todas as funções do **namespace Math** trabalham com variáveis do tipo **Double**.

# Trabalhando com Datas e Horas

Uma das maiores dores de cabeça em qualquer linguagem de programação é ter que trabalhar com datas (anos, mês, dias, horas, minutos e segundos). Felizmente a linguagem VB .NET possui um leque de funções prontas para ajudar nessa árdua tarefa.

Exemplo:

```
1  Module Module1
2      Sub Main()
3          Dim dtData1, dtData2 As Date
4
5          dtData1 = #3/13/2014#
6          dtData2 = #12/31/2014#
7
8          System.Console.WriteLine("Nova data 1: " & DateAdd(DateInterval.Month, 22, dtData1))
9          System.Console.WriteLine("Nova data 2: " & DateAdd(DateInterval.Year, -2, dtData2))
10         System.Console.WriteLine("Diferença..: " & DateDiff(DateInterval.Day, dtData1, dtData2))
11     End Sub
12 End Module
```

**Dica:** Você pode usar a função **Format** para escolher a formatação de valores do tipo Data e Hora.

# Trabalhando com Datas e Horas

A tabela a seguir contém a relação de propriedades da linguagem VB .NET para trabalhar com Datas e Horas:

| Para fazer isso | Use isso |
|---|---|
| Obter a date e hora corrente | Today, Now, TimeofDay, DateString, TimeString |
| Realizar cálculos entre datas | DateAdd, DateDiff, DatePart |
| Retornar um valor do tipo Data | DateSerial, DateValue |
| Retornar um valor do tipo Tempo | TimeSerial, TimeValue |
| Ajustar a Data e Hora | Today, TimeofDay |
| Temporizar um processo | Timer |

Exemplo:

```
1  Module Module2
2      Sub Main()
3          System.Console.WriteLine("Data corrente: " & Today())
4          System.Console.WriteLine("Hora corrente: " & TimeOfDay())
5      End Sub
6  End Module
```

# Trabalhando com Datas e Horas

A tabela a seguir contém a relação de formatos da linguagem VB .NET para formatar e exibir Datas e Horas:

| Expressão | Produz isso |
|---|---|
| Format(Now, “M-d-yy”) | 3-3-14 |
| Format(Now, “M/d/yy”) | 3/3/14 |
| Format(Now, “MM - d - yy”) | 03 - 3 - 14 |
| Format(Now, “ddd, MMMMM d, yyy”) | seg, março 3, 2014 |
| Format(Now, “d MMM, yyy”) | 3 mar, 2014 |
| Format(Now, “hh:mm:ss MM/dd/yy”) | 10:44:42 03/03/14 |

Exemplo:

```
1  Module Module2
2      Sub Main()
3          System.Console.WriteLine("{0}", Format(Now, "M-d-yy"))
4          System.Console.WriteLine("{0}", Format(Now, "M/d/yy"))
5          System.Console.WriteLine("{0}", Format(Now, "MM - d - yy"))
6          System.Console.WriteLine("{0}", Format(Now, "ddd, MMMMM d, yyy"))
7          System.Console.WriteLine("{0}", Format(Now, "d MMM, yyy"))
8          System.Console.WriteLine("{0}", Format(Now, "hh:mm:ss MM/dd/yy"))
9      End Sub
10 End Module
```

# Trabalhando com dados financeiros

A tabela a seguir contém a relação de funções da linguagem VB .NET para trabalhar com dados referentes a área financeira (bancária, por exemplo):

| Para fazer isso | Use isso |
|---|---|
| Calcular depreciação | DDB, SLN, SYD |
| Calcular valor futuro | FV |
| Calcular taxa de juros | Rate |
| Calcula taxa interna de retorno | IRR, MIRR |
| Calcular número de períodos | NPer |
| Calcular pagamentos | IPmt, Pmt, PPmt |
| Calcular valor presente | NPV, PV |

# Finalizando um programa a qualquer momento

Existem momentos que você quer finalizar um programa sem mais delongas – por exemplo, quando ocorrer um erro no programa em vez do mesmo continuar executando você pode mostrar uma mensagem de erro, gravar em um log e finalizar o mesmo. Para essa situação e tantas outras você pode usar o comando **End**. Ele para a execução do programa. Exemplo:

```
1  Module Module1
2      Sub Main()
3          Dim strPalavra As String
4
5          While True
6              System.Console.Write("Informe um palavra: ")
7              strPalavra = System.Console.ReadLine()
8
9              If strPalavra = "Finalizar" Then
10                 End
11             End If
12         End While
13     End Sub
14 End Module
```

**Dica:** O comando **Stop** é similar ao **End** exceto que ele coloca seu programa em modo debug.

# Referências Bibliográficas


- HOLZNER, Steven. **Visual Basic .NET: Black Book**. Arizona: Coriolis Group Books, 2002. xxxviii, 1144 p ISBN 1-57610-835-X.