

# Fundamentos da Programação

**RenatoHaddad**  
**Microsoft Most Valuable Professional**  
**Brasil**

Meu nome é Renato Haddad e sou Microsoft Most Valuable Professional no Brasil

# Objetivo

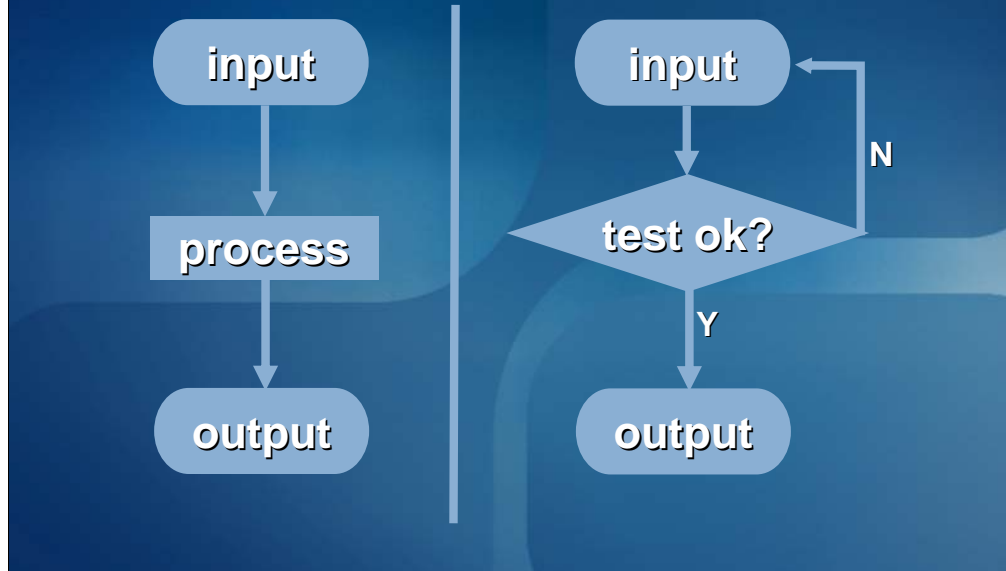
Mostrar os fundamentos da programação através de exemplos e práticas utilizadas no dia a dia do desenvolvimento de aplicações

O objetivo deste treinamento é mostrar os fundamentos da programação através de exemplos e práticas utilizadas no dia a dia do desenvolvimento de aplicações. Em muitos momentos estarei mostrando exemplos ilustrados com códigos e imagens para facilitar o bom entendimento e compreensão. Tais fundamentos são imprescindíveis para que você entenda que programação não é somente codificação, e sim, planejar uma aplicação de forma que seja de fácil produção, manutenção e usabilidade.

# Lógica de Programação

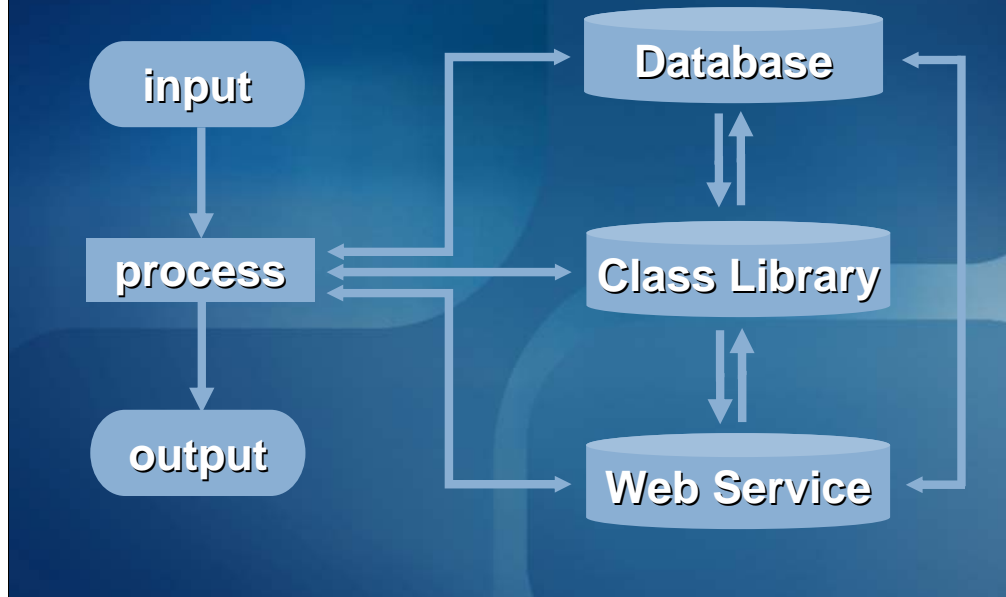
A lógica de programação empregada em um código determina diversos fatores como desempenho, fluxo de dados, estrutura do código para futuras manutenções e interoperabilidade.

# Lógica de Programação



Neste fluxos você percebe que há uma entrada e uma saída de dados. O processamento parece uma caixa preta, mas é aqui que você terá que se esforçar para aprender técnicas, metodologias, explorar a linguagem utilizada e como aplicar da melhor forma possível no código. Alguns fluxos de dados requerem condicionais, desvios, loopings, chamadas a rotinas e funções, enfim, é indicado que você pratique diversas vezes o código proposto da aplicação, afim de aprender e descobrir qual é a melhor forma. Opte sempre pelo simples e rápido, não reinvente a roda, use o que já existe pronto e com a experiência você saberá planejar um bom código. A entrada de dados pode ser através da Web, uma aplicação Windows, um smartphone, um pocket pc, enfim, qualquer meio que interaja com o usuário. Criar uma boa interface é um grande desafio. Já a saída de dados pode ser por meio impresso, voz, digital ou apenas exibido na tela.

# Lógica de Programação



Já que estamos focados na plataforma .NET, este fluxo representa o processamento de códigos necessários para controlar o acesso as informações oriundas de um banco de dados ou de uma Class Library, que é uma DLL ou ainda de um Web Service. Se o Web Service for acessar um banco de dados, esta comunicação está pronta para isso. Com tantas opções de arquiteturas de soluções atualmente no mercado, com o tempo você saberá identificar quais são os melhores recursos a serem utilizados em cada situação.

# Fundamentos

Vamos ver alguns fundamentos básicos da programação

# Variável

- O que é uma variável?
- Em que situação usar uma variável?

Uma variável é um espaço alocado na memória RAM para ser utilizada no código. No .NET todas as variáveis devem ser obrigatoriamente declaradas e ter um tipo, no entanto, não é preciso declarar todas as variáveis no início do programa. Pelo ambiente do .NET tratar códigos seguros, garante a integridade da variável, ou seja, que os tipos atribuídos serão sempre de acordo ao definido. O conteúdo de uma variável poderá mudar no decorrer do programa, pois isto é comum de acontecer. Você não pode atribuir um valor a uma variável sem defini-la, isso causará um erro. As variáveis são usadas em situações em que você precisar armazenar uma informação temporariamente na memória.

# Variável

## Como declarar uma variável?

### C#

#### Tipo NomeDaVariável

```
double x2 = 3D;  
decimal moeda = 300.5m;  
int valor2 = 123;  
uint tipo = 4204868280;  
long valor3 = 4294967296;  
ulong valor4 = 9223372036854775808;  
short ind1 = 32767;  
ushort ind2 = 65535;  
bool desativado = true;  
string myCar = "jaguar";
```

Espaço  
na RAM

### VB.NET

#### Dim NomeDaVariável As Tipo

```
Dim myCar As String = "jaguar"  
Dim cost As Double = 12.5  
Dim quantity As Integer = 10  
Dim total As Double = cost * quantity  
Dim mouse, printer As String  
Dim V1, V2 As Integer, a1, a2 As Double
```



Na declaração de uma variável você deverá informar o nome e o tipo. Um variável não pode ser declarada sem o respectivo tipo, pois o .NET não permite isto. Veja alguns exemplos em C#, onde você declara o tipo e, em seguida o nome da variável seguido do respectivo conteúdo. Note os diversos tipos que o .NET permite. Já em VB.NET é o contrário, você informa a palavra chave DIM seguida do nome da variável AS tipo. É uma prática comum atribuir o valor diretamente na linha. No entanto, nem sempre isso será possível devido ao fluxo de informação do programa. Note ainda que uma variável pode conter o resultado de outras. Se você declarar na mesma linha duas variáveis com o mesmo tipo, ambas serão deste tipo. Note ainda que você pode declarar na mesma linha diversas variáveis com tipos diferentes, e a regra é que todas as variáveis serão do tipo declarado após o AS



# Variável

## Como nomear uma variável?

```
'VB.NET
Dim intValor As Integer
Dim strTexto As String
Dim dblPreco As Double
Dim objPessoa As Object
```

```
//C#
int inValor;
long lngValor = 4294967296;
double dblPreco = 124.50;
string strSQL = "Visual C#";
string strTools = "VS.NET";
string strBest = strTools + strSQL;
```

A nomeação de variáveis depende da metodologia e padrão que a sua empresa utiliza. Via de regra muitas equipes adotam um padrão que inicia com o tipo da variável seguido do nome, por exemplo, intValor, strTexto, dblSalario, objPessoa. Este tipo de nomenclatura também é adotada para os tipos de objetos usados nas aplicações, e é usada há anos em toda a comunidade. Dizemos que é uma simbologia húngariana. A facilidade proporcionada por esta nomenclatura ajuda em códigos muito extensos, mas não influencia em absolutamente nada em performance e desempenho. O importante é que você adote um padrão, senão acabará criando variáveis que nunca saberá a finalidade, e na hora de manutenção você perderá muito tempo descobrindo.

# Variável

## Tipos de variáveis

Visual Basic type	Common language runtime type structure	Nominal storage allocation	Value range
Boolean	System.Boolean	2 bytes	True or False.
Byte	System.Byte	1 byte	0 through 255 (unsigned).
Char	System.Char	2 bytes	0 through 65535 (unsigned).
Date	System.DateTime	8 bytes	0:00:00 on January 1, 0001 through 11:59:59 PM on December 31, 9999.
Decimal	System.Decimal	16 bytes	0 through +/- 79,228,162,514,264,337,593,543,950,335 with no decimal point; 0 through +/- 7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest nonzero number is +/-0.00000000000000000000000000000001 (+/-1E-28).
Double (double-precision floating-point)	System.Double	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324 for negative values; 4.94065645841246544E-324 through 1.79769313486231570E+308 for positive values.

As variáveis podem ser dos seguintes tipos, conforme a tabela. É importante ressaltar que a correta declaração do tipo influencia na boa programação e no desempenho da mesma, senão você terá que ficar convertendo os tipos durante o fluxo do programa, e isto demanda um certo tempo, além de economizar espaço de memória.

# Variável

## Tipos de variáveis

Visual Basic type	Common language runtime type structure	Nominal storage allocation	Value range
<b>Long</b> (long integer)	<b>System.Int64</b>	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807.
<b>Object</b>	<b>System.Object</b> (class)	4 bytes	Any type can be stored in a variable of type <b>Object</b> .
<b>Short</b>	<b>System.Int16</b>	2 bytes	-32,768 through 32,767.
<b>Single</b> (single-precision floating-point)	<b>System.Single</b>	4 bytes	-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values.
<b>String</b> (variable-length)	<b>System.String</b> (class)	Depends on implementing platform	0 to approximately 2 billion Unicode characters.
<b>User-Defined Type</b> (structure)	(inherits from <b>System.ValueType</b> )	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members.

Aqui estão os tipos possíveis de serem declarados no .NET Framework. Qualquer dúvida que você tiver em relação a um determinado tipo, consulte o help do programa.

# Variável

## Tempo de vida de uma variável

```
// exemplo 1
int preco = 18;
if (preco >= 18)
{
    double indice = 2.3;
    int aumento = 100;
    Console.WriteLine(indice);
    Console.WriteLine(aumento);
}
else
{
    double indice = 2.4;
    int aumento = 120;
    Console.WriteLine(indice);
    Console.WriteLine(aumento);
}
//Console.WriteLine(indice);
//Console.WriteLine(aumento);
```

```
//exemplo 2
int preco = 18;
double indice2 = 0;
int aumento2 = 0;
if (preco >= 18)
{
    indice2 = 2.3;
    aumento2 = 100;
}
else
{
    indice2 = 2.4;
    aumento2 = 120;
}
Console.WriteLine(indice2);
Console.WriteLine(aumento2);
```

O tempo de vida de uma variável significa onde que ela poderá ser enxergada e utilizada dentro da estrutura do programa. Não necessariamente todas as variáveis devem ser declaradas no início do programa, pois há variáveis que nunca precisarão ser utilizadas no programa devido ao fluxo do programa. Por exemplo, em instruções condicionais, se você for utilizar a variável somente naquele bloco de código, então a declaração deverá ocorrer dentro do bloco. Note no exemplo 1 que as variáveis `indice` e `aumento` não podem ser enxergadas fora do bloco do `IF`, pois foram declaradas dentro do bloco. O `Console.WriteLine` é a forma de exibir dados ao usuário em uma Console Application, e neste caso as linhas fora do bloco do `IF` estão como comentários, senão ocorrerá um erro. Já no exemplo 2, estas variáveis foram declaradas antes do `IF`, e portanto, serão enxergadas tanto dentro quanto fora do bloco. Chamamos isso de escopo de uma variável.

# Variável

## Conversões

```
Dim age As Double = 21
Dim newAge As Int16 = Convert.ToInt32(age)
Dim salary As String = "3400.56"
Dim newSalary As Double = Convert.ToDouble(salary)
Dim name As String = "Renato"
Dim employee As String
employee = name + Convert.ToString(newSalary)

Dim myNumber As Long = 1000
Dim myNewType As Single
myNewType = CType(myNumber, Single)
Dim myVar As Double = CType(myNumber, Double)
Dim myTest As String = CType(salary, String)
```

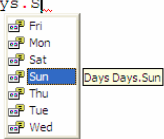
```
int number1 = 10;
object obj;
obj = number1; // boxing
number1 = (int)obj; // unboxing
string valor;
valor = "Valor: " + ((int)obj).ToString(); // unboxing
```

Existem situações onde a conversão de uma variável é necessária, principalmente quando interage com o usuário. O .NET dispõe da classe `Convert` o qual contém diversos métodos conforme os exemplos da figura, sendo `ToInt32`, `ToDouble`, `ToString`. Você pode ainda usar a função `Ctype` para converter um objeto, informando o conteúdo e o tipo de dado que deverá ser convertido. Lidando com objetos, você pode fazer o `boxing` e o `unboxing` que é a conversão implícita de um objeto/tipo ou tipo/objeto. Dizemos que isto representa um `CAST`.

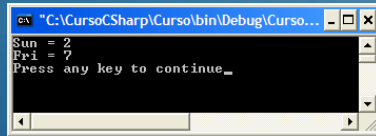
# Variável

## Tipos definidos pelo usuário

```
enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};  
public static void Main()  
{  
    int start = (int) Days.S  
}  
}
```

A screenshot of an IDE's Intellisense dropdown menu. It lists the members of the 'Days' enum: Fri, Mon, Sat, Sun, Thu, Tue, and Wed. The 'Sun' member is currently selected and highlighted in blue. To the right of the list, the text 'Days.Days.Sun' is visible.

```
enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};  
public static void Main()  
{  
    int start = (int) Days.Sun;  
    int finish = (int) Days.Fri;  
    Console.WriteLine("Sun = {0}", start);  
    Console.WriteLine("Fri = {0}", finish);  
}
```

A screenshot of a Windows console window. The title bar shows the file path 'C:\CursoCSharp\Curso\bin\Debug\Curso...'. The console output displays 'Sun = 2' and 'Fri = 7' on separate lines, followed by the prompt 'Press any key to continue\_'.

```
C:\CursoCSharp\Curso\bin\Debug\Curso...  
Sun = 2  
Fri = 7  
Press any key to continue_
```

Em alguns casos você pode criar os seus próprios tipos de dados através de Enumeração, tornando a visualização e a manipulação destes tipos diretamente no Intellisense, onde são exibidos todas as opções disponíveis.

# Estruturas de Programação

Vamos ver alguns conceitos de estruturas utilizadas na programação

# Estruturas de Decisão

## IF - VB.NET

**If** *condition* **Then**  
    *statements*

**Elseif** *elseifcondition* **Then**  
    *elseifstatements*

**Else**  
    *elsestatements*

**End If**

## IF - C#

**if** (*condition*)  
    *statements*

**else if** (*condition*)  
    *statements*

**else**  
    *statements*

A estrutura de decisão sem nenhuma dúvida é a mais utilizada na programação. O objetivo é identificar o conteúdo de uma condição e direcionar o fluxo do programa para um determinado cálculo, rotina, desvio, função, etc. O primeiro dele é o IF. Veja a sintaxe no VB.NET e no C Sharp. IF condição THEN então faça algo; caso contrário Else ou ainda você pode testar outra condição ElseIf até que o programa encontre o resultado correto. Cabe ressaltar que os Statements podem ser um bloco de código.



# IF – VB.NET

```
'IF 1
Dim Salary As Double = 2400
If Salary > 2000 Then
    Salary *= 1.1
End If
```

```
'IF 2
If Salary < 1000 Then
    Salary += 50
Else
    Salary += 100
End If
```

```
'IF 3
If Salary < 1000 Then
    Salary += 50
ElseIf Salary >= 1000 And Salary < 2000 Then
    Salary += 100
Else
    Salary += 500
End If
```

```
Dim linguagem As String = TextBox1.Text
If linguagem = "VB.NET" Then
    MessageBox.Show("Excelente")
ElseIf linguagem = "Visual C#" Then
    MessageBox.Show("Excelente")
ElseIf linguagem = "Delphi .NET" Then
    MessageBox.Show("boa escolha")
ElseIf linguagem = "Java" Then
    MessageBox.Show("boa sorte")
Else
    MessageBox.Show("pense bem")
End If
```

Veja este exemplo: existe uma variável chamada Salary que está definida e contém o valor 2400. Como toda condição retorna apenas verdadeiro ou falso, no IF1 é verificado se o valor é maior que 2000, e caso seja verdadeiro será somado 10% ao Salary. No IF2 é verificado se o valor é menor que 1000 e dependendo da condição é somado 50 ou 100, pois há o Else. Já no IF3 existem 3 condições, onde o primeiro IF soma 50, o segundo 100 e o ELSE 500. Note que é possível avaliar diversos ElseIfs com uma determinada expressão. Já no outro exemplo, usamos uma condicional para valores alfanuméricos com vários ElseIfs

# IF – C#

```
bool status = true;
double Salary = 1000;
if (status == true)
    Salary *= 1.1;

if (Salary >= 1800)
{
    double indice = 0.12;
    int bonus = 300;
    Salary *= indice + bonus;
    AumentaValor(Salary);
}
else
{
    double indice = 0.33;
    int bonus = 540;
    Salary *= indice + bonus;
    DiminuiValor(Salary);
}
```

```
Console.Write("Digite uma letra: ");
char c = (char) Console.Read();

if (Char.IsUpper(c))
    Console.WriteLine("A letra é maiúscula.");
else if (Char.IsLower(c))
    Console.WriteLine("A letra é minúscula.");
else if (Char.IsDigit(c))
    Console.WriteLine("Você digitou um número.");
else
    Console.WriteLine("Você não digitou uma letra.");
```

Veja este exemplo em C Sharp onde são avaliadas expressões lógicas, numéricas e strings. Note que no IF do Salary são definidas outras duas variáveis, é aplicada uma fórmula e são chamadas as respectivas rotinas. Desta forma, o IF desvia o fluxo do programa para uma determinada rotina.

# Estruturas de Decisão

Select Case – VB.NET

**Select Case** *testexpression*

**Case** *expressionlist*  
*statements*

**Case Else**  
*elsestatements*

**End Select**

switch – C#

**switch** (*testexpression*)

{

**case** *expressionlist*:  
*statements*

*break*:

**default**:

*statements*] }

As estruturas de Select Case deixa o código mais claro, limpo, fácil manutenção e entendimento. O funcionamento ocorre da seguinte maneira: a expressão é obtida no Select Case e para cada Case existe uma condição que será validada. Caso o Case seja verdadeiro, então a linha ou o bloco de código é executado. Se nenhum dos Cases for válido, você pode usar o Case Else. No caso do C Sharp, o Switch tem mesmo papel, no entanto, a cada Case é preciso declarar o break, senão o programa continua avaliando todos os Cases.

# Select Case – VB.NET

```
Dim DiaSemana As Integer = DateTime.Now.DayOfWeek
Dim Dia As String
Select Case DiaSemana
    Case 0, 6
        Dia = "Fim de semana"
    Case 1
        Dia = "Inicio de semana"
    Case 5
        Dia = "Sexta feira"
    Case Else
        Dia = "Meio da semana"
End Select
```

```
Dim country As String = TextBox1.Text
Dim sport As String
Select Case country
    Case "Brasil" Or "Argentina"
        sport = "Futebol"
    Case "EUA" Or "Australia" Or "England"
        sport = "Tennis"
    Case Else
        sport = "Golf"
End Select
```

```
<WebMethod(Description:="Calcula o Imposto de Renda")> _
Public Function ImpostoRenda(ByVal valor As Double) As Double
    Select Case valor
        Case Is < 900
            ImpostoRenda = 0
        Case 900 To 1800
            ImpostoRenda = (valor * 0.15) - 135
        Case Is > 1800
            ImpostoRenda = (valor * 0.25) - 250
    End Select
End Function
```

Veja o uso do Select Case nestes exemplos onde estamos avaliando um double e uma string. Veja como avaliar a expressão, por exemplo Case 0 vírgula 6 indica que pode ser 0 OU 6; Case IS < 900 indica que deverá ser menor que 900; Case 900 TO 1800 indica que o valor deverá estar dentro desta faixa; Case “Brasil” OR “Argentina” indica que pode ser um dos dois países. Note ainda o uso do Case Else, caso a condição não atender nenhum dos casos, o programa cai no Case Else. O uso do Case Else não é obrigatório, mas é uma boa prática em certos casos.

## switch - C#

```
Console.Write("Qual o curso? ");
string curso = Console.ReadLine();
switch (curso)
{
    case "ASP":
        Console.WriteLine("Active Server Pages");
        break;
    case "ASP.NET":
        Console.WriteLine("ASP.NET");
        break;
    case "VB.NET":
        Console.WriteLine("VB.NET");
        break;
    default:
        Console.WriteLine("curso inválido");
        break;
}

Console.Write("Qual é o valor? ");
string s = Console.ReadLine();
int valor = int.Parse(s);
switch (valor)
{
    case 100:
        Console.WriteLine("O valor selecionado é 100");
        break;
    case 150:
        Console.WriteLine("O valor selecionado é 150");
        break;
    case 200:
        Console.WriteLine("O valor selecionado é 200");
        break;
    default:
        Console.WriteLine("Por favor selecione um valor");
        break;
}
```

Veja no C Sharp com é o uso do switch, onde cada expressão deverá conter o break assim que for executada. O uso do Default ocorre se nenhuma expressão acima foi válida. No C Sharp não é possível usar o Case 100 TO 200 para determinar o intervalo da expressão como usamos no VB, sendo preciso montar uma lógica diferente. O fato é que tanto o uso do Select Case quanto o Switch deixa o código mais estruturado em relação ao uso de vários ElseIF.

# Estruturas de Looping

- For Next
- ForEach
- While
- Do While

O uso de loopings nos programas é uma prática comum, pois em muitos casos é preciso percorrer uma determinada coleção de dados, um conjunto de registros, valores de matrizes, etc. Estes quatro comandos serão detalhados a seguir.

# For Next – VB.NET

**For** *counter* = *start* **To** *end* **Step**

*statements*

**Exit For**

**Next**

```
Dim contador As Integer
For contador = 1 To 20
    Console.WriteLine(contador)
Next
'-----
For contador2 As Integer = 1 To 20
    Console.WriteLine(contador2)
Next
'-----
For contador3 As Integer = 1 To 20 Step 4
    Console.WriteLine(contador3)
Next
'-----
For contador4 As Integer = 100 To 0 Step -20
    Console.WriteLine(contador4)
Next
'-----
For contador5 As Integer = 0 To 20
    Console.WriteLine(contador5)
    If contador5 = 10 Then Exit For
Next
```

```
For contador As Integer = 0 To lstLinguagens.Items.Count - 1
    If lstLinguagens.Items(contador).Selected Then
        lblLinguagens.Text += lstLinguagens.Items(contador).Text & " - "
    End If
Next
```

O For Next precisa de um contador que normalmente é uma variável e o looping vai do início Start até TO o fim END. Veja nestes exemplos que a variável contador por ser inicializada antes do For ou na própria declaração. Em certos loopings você pode usar o Step que é o incremento do looping, podendo ser positivo ou negativo. Se durante o processamento você quiser abandonar o looping, terá que usar o Exit For.

# For – C#

*for (counter; expression; iterators)  
statement*

```
for (int i = 1; i <=10; i++)  
    Console.WriteLine(i);  
  
//-----  
for (int i = 1; i <= 10; i++)  
{  
    if (i == 8)  
        break;  
    Console.WriteLine(i);  
}
```

```
Label1.Text = "Datos seleccionadas: <br>";  
for (int i=0; i <= Calendar1.SelectedDates.Count-1; i++)  
{  
    if (Calendar1.SelectedDates[i].Day != 0)  
    {  
        Label1.Text += Calendar1.SelectedDates[i] + "<br>";  
    }  
}
```

Em C Sharp o funcionamento é igual ao VB, mudando apenas a sintaxe na hora de declarar o FOR. Note que para abandonar o looping é o Break.



# For Each – VB.NET

For Each *element* In *group*  
    *statements*  
Exit For  
Next

```
Dim odd As Integer = 0
Dim even As Integer = 0
Dim arrayData() As Integer = {0, 1, 2, 5, 7, 8, 11}
Dim number As Integer
For Each number In arrayData
    If number Mod 2 = 0 Then
        even += 1
    Else
        odd += 1
    End If
Next
Console.WriteLine("Founded {0} even and {1} odd", _
    even.ToString(), odd.ToString())
```

```
Dim Arqs As String() = Directory.GetFiles("c:\windows")
Dim file As String
For Each file In Arqs
    Console.WriteLine(file)
Next
```

```
'checa os itens selecionados na lista
lblLinguagens.Text = String.Empty
Dim objItem As ListItem
For Each objItem In lstLinguagens.Items
    If objItem.Selected Then
        lblLinguagens.Text += objItem.Text & " - "
    End If
Next
```

O For Each é usado em casos onde você precisa percorrer uma coleção de dados. É mais fácil, simples e claro em relação ao For Next. No For Each você não precisa verificar o limite do looping, pois isto já está implícito na própria coleção, ou seja, você se preocupa apenas em manipular o objeto atual. Caso tenha dúvida no uso, pense da seguinte forma: Para cada elemento contido na coleção, faça algo. Veja nestes exemplos onde percorremos uma lista de Arrays com números ou com os nomes dos arquivos em uma pasta, ou ainda um controle ListBox para saber quais itens estão selecionados.

# Foreach – C#

**foreach** (*element in group*)  
*statement*

```
DataSet ds = new DataSet();
FileStream arquivo = new FileStream("c:\\produtosXML.xml",
    FileMode.Open, FileAccess.Read);
ds.ReadXml(arquivo);
arquivo.Close();
DataTable dt = ds.Tables[0];
foreach (DataRow row in dt.Rows)
{
    TextBox1.Text += row["ProductName"] + " - " +
        row["UnitPrice"] + "\n";
}
```

```
lstArquivos.Items.Clear();
txtInfos.Text = string.Empty;
DirectoryInfo diretorio = new DirectoryInfo(lstDiretorios.SelectedValue);
FileInfo[] arquivos = diretorio.GetFiles();
foreach (FileInfo arq in arquivos)
{
    lstArquivos.Items.Add(new ListItem(arq.FullName.ToString()));
}
```

No C Sharp a sintaxe é mais simples ainda. Neste exemplo percorremos todas as linhas de uma DataTable oriundas de um arquivo XML, mostrando todos os dados, e no outro exemplo são listados todos os arquivos contidos em uma determinada pasta. A idéia de coleções é fantástica porque você não precisa saber quantos registros existem no DataTable, assim como quantos arquivos existem na pasta, pois o Foreach é que encarrega disto.

## VB.NET

**While** *condition*  
*statements*  
**End While**

## While

```
dr = cmd.ExecuteReader();  
TextBox1.Text = string.Empty;  
while(dr.Read())  
{  
    TextBox1.Text += string.Format("{0} ({1:c})\n",  
        dr["ProductName"], dr["UnitPrice"]);  
}
```

**Do { While | Until }** *condition*  
*statements*  
**Exit Do**  
**Loop**

**Do**  
*statements*  
**Exit Do**  
**Loop { While | Until }** *condition*

## C#

**while** (*expression*) *statement*  
**do** *statement* **while** (*expression*);

```
Dim i As Integer  
i = 1  
While i <= 20  
    Console.WriteLine(i)  
    i += 1  
End While  
  
i = 1  
Do While i <= 20  
    Console.WriteLine(i)  
    i += 2  
Loop  
  
i = 1  
Do  
    Console.WriteLine(i)  
    i += 2  
Loop Until i >= 20  
  
i = 1  
Do  
    Console.WriteLine(i)  
    i += 2  
Loop While i <= 20
```

O looping While é executado sempre associado a uma condição, ou seja, a cada passagem pelo looping a condição é avaliada. Dependendo da situação, a condição deve ser colocada no início ou no final do looping. Se for no início, é avaliada logo na primeira vez; se for no final, o looping é executado pelo menos a primeira vez, pois a condição será avaliada no final da primeira passagem pelo looping. Caso precise abandonar o looping, use o Exit Do. Nestes exemplos temos diversas condicionais e uma das situações em que o looping é muito utilizado é na leitura de dados de um DataReader para preencher um determinado controle com os dados de uma tabela, por exemplo, preencher um TextBox com alguns dados da tabela de produtos.

# Rotinas e Objetos

Vamos ver o uso de rotinas e objetos

# Procedures / Rotinas

- O que é uma procedure?

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click  
    'códigos  
End Sub
```

- Argumentos e tipos usados

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click  
    NewProcedure(".NET", 2005)  
End Sub  
  
Private Sub NewProcedure(ByVal tools As String, ByVal year As Integer)  
    MessageBox.Show(tools, year)  
End Sub
```

- Argumento opcional

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click  
    NewProcedure(".NET", 2005)  
    NewProcedure(".NET", 2005, 10)  
End Sub  
  
Private Sub NewProcedure(ByVal tools As String, _  
    ByVal year As Integer, Optional ByVal month As Integer = 12)  
    MessageBox.Show(tools, year & "/" & month)  
End Sub
```

Procedure é um conjunto de códigos a serem executados para uma determinada finalidade. A característica principal de uma procedure é que ela nunca retorna nada, simplesmente executa o código.

Quando você adiciona um botão em um formulário e cria um código, é criado automaticamente um evento associado a este botão que vai executar a procedure. Uma procedure pode conter diversos argumentos com respectivos tipos diferentes. Neste exemplo é chamada a procedure NewProcedure passando os devidos parâmetros. O mais importante é que a ordem e os tipos dos parâmetros devem ser respeitados, pois os mesmos estão declarados na lista de argumentos da procedure. Alguns argumentos são opcionais e devem ser declarados na procedure com a palavra chave Optional contendo o valor default, assim, quando a procedure for chamada, se este parâmetro não for fornecido, é assumido o valor default, caso contrário é assumido o valor fornecido. Para chamar uma procedure basta digitar Call Procedure ou somente o nome da Procedure.

# Erros comuns na chamada

Fig 1

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    NewProcedure(".NET", 2005)
    NewProcedure(".NET", 2005, 10)
End Sub

Private Sub NewProcedure(ByVal tools As String, _
    ByVal year As Integer, Optional ByVal month As Integer = 12)
    MessageBox.Show(tools, year & "/" & month)
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button2.Click
    NewProcedure("Express Editions", 2005, 6)
End Sub
```

Fig 2

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    NewProcedure(".NET", 2005)
    NewProcedure(".NET", 2005, 10)
End Sub

Private Sub NewProcedure(ByVal tools As String, _
    ByVal year As Integer, Optional ByVal month As Integer = 12)
    MessageBox.Show(tools, year & "/" & month)
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button2.Click
    NewProcedure(".NET", 2005)
    NewProcedure(".NET", 2005, 10)
End Sub
```

Fig 3

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click, Button2.Click
    NewProcedure(".NET", 2005)
    NewProcedure(".NET", 2005, 10)
End Sub
```

Observe na figura 1 que existe no Botão 2 uma chamada para a mesma NewProcedure contendo argumentos diferentes. Esta situação não está errada devido aos argumentos. Já na figura 2 onde os argumentos são idênticos, apesar de funcionar sem problemas, conceitualmente está errado repetir o mesmo conteúdo de ambos botões. O .NET permite atribuir ao Handles a chamada da Procedure. Então, o correto é como está na figura 3, onde o Handles contém os dois botões + evento. Isto se aplica a qualquer objeto existente, basta informar o objeto + evento. Utilizei um simples exemplo, mas é comum termos grandes rotinas a serem executadas diversas vezes.

# Função

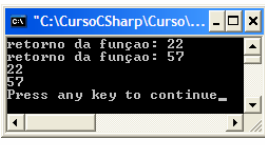
- O que é uma função?
- Argumentos e tipos usados
- Tipos retornados

```
static string FormatData(string valor)
{
    return string.Format("retorno da função: {0}", valor);
}

static void Main()
{
    Console.WriteLine(FormatData(Convert.ToString(Soma(10,12))));
    Console.WriteLine(FormatData(Convert.ToString(Subtrai(80,23))));
    Console.WriteLine(Soma(10,12));
    Console.WriteLine(Subtrai(80,23));
}

static int Soma(int n1, int n2)
{
    return n1 + n2;
}

static int Subtrai(int n1, int n2)
{
    return n1 - n2;
}
```



Uma função é uma rotina a ser executada que sempre irá retornar algo de um tipo, por exemplo, uma string, um integer, um DataSet, etc. Uma função pode ou não conter argumentos. Caso tenha, é preciso definir os nomes e os tipos. Neste exemplo temos 3 funções: a soma e a subtrai que recebem dois inteiros como argumento, efetua o cálculo e retorna um inteiro. A função FormatData recebe uma String, efetua a formatação e retorna uma String. Perceba na chamada desta função que foi preciso converter o resultado da função que é inteiro para String, afinal o argumento é String. Note ainda que chamamos uma função dentro de outra sem nenhum problema.

# Função

```
Public Function GetData(ByVal sql As String) As DataSet
    Dim conn As New SqlConnection(conexao)
    Dim adapter As New SqlDataAdapter(sql, conn)
    Dim ds As New DataSet
    conn.Open()
    adapter.Fill(ds, "dados")
    conn.Close()
    Return ds
End Function

Public Function QtdeEstoque(ByVal cod As Integer) As String
    Dim conn As New SqlConnection(conexao)
    Dim sql As String = "Select UnitsInStock FROM Products Where"
    conn.Open()
    Dim cmd As New SqlCommand(sql, conn)
    Return cmd.ExecuteScalar()
    conn.Close()
End Function

Public Function Faturamento() As Double
    Dim conn As New SqlConnection(conexao)
    Dim sql As String = "Select Sum(UnitsInStock * UnitPrice)"
    conn.Open()
    Dim cmd As New SqlCommand(sql, conn)
    Return cmd.ExecuteScalar()
    conn.Close()
End Function
```

Veja o trecho de código de uma rotina que retorna um DataSet, uma String e um Double. Estas rotinas podem ser chamadas a partir de qualquer objeto, desde que consigam enxergá-las. Cabe ressaltar que o escopo da função ocorre somente no local onde ela foi criada, exceto casos de componentização e Web Services.



# Função

## Funções internas

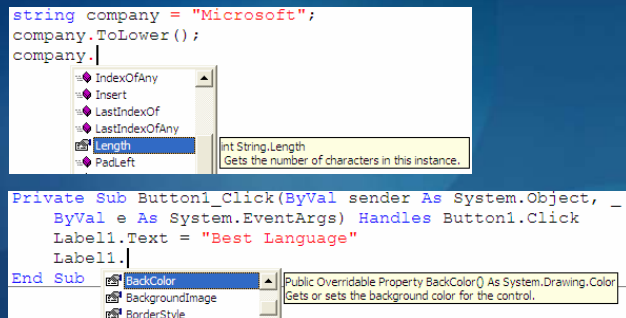
```
Dim juros As Double = txtJuros.Text / 100
Dim parcelas As Double = numParcelas.Text
Dim valor As Double = txtValor.Text
Dim resultado As Double = 0
If rdPagto.Checked Then
    If txtEntrada.Text > 0 Then
        valor -= txtEntrada.Text
    End If
    resultado = -Pmt(juros, parcelas, valor)
ElseIf rdFuturo.Checked Then
    resultado = -FV(juros, parcelas, 1, valor)
ElseIf rdPresente.Checked Then
    resultado = -PV(juros, parcelas, 1, valor)
    resultado = -PV(|
End If
```

PV (Rate As Double, NPer As Double, Pmt As Double, [FV As Double = 0.0],

Se uma função já existe internamente do Framework, devemos utilizá-la ao invés de criar uma customizada, pois nada de reinventar a roda. Existem centenas de funções prontas no Framework e vale a pena pesquisar qual função lhe atende em determinada situação.

# Objeto

## O que é um objeto?



Já que estamos focados no .NET Framework, tudo no .NET são objetos, seja um controle, uma variável, um componente, etc. Cada objeto contém seus métodos e propriedades.

# Propriedades

## Propriedades de um objeto



As propriedades de um objeto definem as características do mesmo, por exemplo, um carro tem a cor azul, direção hidráulica, vidros verdes, 4 portas, rodas de ferro, etc. Já uma moto pode ter algumas propriedades similares, por exemplo, cor vermelha, 2 rodas de alumínio, etc. Apesar das propriedades similares aos objetos, cada um contém as respectivas.

# Métodos

## Métodos de um objeto



Métodos de um objeto são as ações referentes ao mesmo. Alguns objetos podem ter métodos similares, por exemplo, um carro e uma moto tem os métodos acelerar, frear, abastecer, trocar a marcha, etc.

# Coleções

O que é uma coleção?

The screenshot shows a Windows application window titled 'Form1'. It features a data grid with the following columns: ProductID, ProductName, CategoryID, UnitPrice, UnitsInStock, and SupplierID. The grid displays data for 'Dairy Products' (CategoryID: 4). The data is as follows:

ProductID	ProductName	CategoryID	UnitPrice	UnitsInStock	SupplierID
11	Queso Cabrales	4	21	22	5
12	Queso Manchego La Pastora	4	38	86	5
31	Gorgonzola Telino	4	12.5	0	14
32	Mascarpone Fabioli	4	32	9	14
33	Gellost	4	2.5	112	15
59	Raclette Courdavault	4	55	79	28
60	Camembert Pierrot	4	34	19	28

Coleções é uma maneira estruturada de agrupar e gerenciar objetos de tipos semelhantes com a finalidade de facilitar o gerenciamento destes objetos. O uso é comum quando se lida com banco de dados, por exemplo, uma tabela contém diversas linhas e colunas e para manipular basta ler estas coleções. Outro exemplo clássico são os objetos inseridos em um formulário. Quando se deseja pesquisar certos objetos no formulário, basta ler a coleção de objetos e identificar o objeto requerido.

# OOP

## Programação Orientada a Objetos - OOP

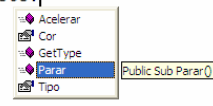
```
Public Class clsCarro
    'property
    Private myCor As String
    Public Property Cor() As String
        Get
            Return myCor
        End Get
        Set(ByVal Value As String)
            myCor = Value
        End Set
    End Property

    'method
    Public Sub Acelerar()
        'acelera o carro
    End Sub
    Public Sub Parar()
        'para o carro
    End Sub
End Class
```

```
Public Class clsMoto
    Inherits clsCarro
    Private myTipo As String
    Public Property Tipo() As String
        Get
            Return myTipo
        End Get
        Set(ByVal Value As String)
            myTipo = Value
        End Set
    End Property
End Class

Sub Main()
    Dim objCarro As New clsCarro
    objCarro.Cor = "verde"
    objCarro.Acelerar()
    objCarro.Parar()

    Dim objMoto As New clsMoto
    objMoto.Cor = "azul"
    objMoto.Tipo = "trial"
    objMoto.Acelerar()
    objMoto.|
End Sub
```



O termo Programação Orientada a Objetos – OOP se tornou comum na plataforma .NET, visto que qualquer coisa é um objeto contendo métodos e propriedades. Você pode criar um objeto e definir as propriedades e métodos conforme a necessidade. Este modelo de programação é diferente do modelo usado em programação estruturada ou procedural, pois com OOP você consegue produtividade, escrever menos códigos e ter domínio sobre os objetos e classes. Neste exemplo temos uma classe chamada clsCarro contendo uma propriedade chamada Cor para definir a cor do carro. Em seguida, temos os métodos Acelerar e Parar. Depois temos uma nova classe chamada clsMoto, o qual herda a classe clsCarro e define mais uma propriedade. Este recurso de herdar é fantástico no OOP, afinal se a classe Carro já tem características comuns a classe Moto, porque recriar, basta herdar o objeto. Quando você for usar o objeto Moto, irá perceber que as propriedades e os métodos do Carro estão disponíveis.

# Glossário

Veja o glossário dos termos comuns encontrados no meio da programação

## Termos comuns

- Variable
- Types
- Index
- Constants
- Collection
- Array
- Procedure
- n-tier
- business class
- web services

Variable – espaço volátil de memória RAM para armazenar dados temporários

Types – pode ser tipos de dados, objetos, controles ou projetos

Index – índice de uma coleção, tabela, matriz ou objeto

Constants – representação de uma variável criada ou da própria linguagem que não sofrerá alterações

Collection – coleção de objetos

Array – matriz de dados ou objetos

Procedure – rotina definida pelo programa ou desenvolvedor

n-tier – desenvolvimento em camadas (banco de dados, classe de negócio e interface)

business class – classe de negócio contendo as regras de uma rotina ou dados

web services – métodos a serem expostos via http para quem quiser consumir



## Termos comuns

- Build
- Deploy
- OOP
- Components
- Class library
- Breakpoint
- Debug
- Solution
- Project

build – é o processo de compilação de um projeto

deploy – é o processo de instalação do aplicativo, classe ou componente

OOP – programação orientada à objeto

Components – podem ser programas externos, códigos compilados externos à aplicação

class library – biblioteca de classes sendo uma camada DLL com rotinas a serem consumidas por outras rotinas

Breakpoint – ponto de interrupção para rastrear o programa e descobrir erros ou verificar o fluxo da informação

Debug – processo de rastrear um erro ou confirmação de dados no programa

Solution – arquivo de solução que pode conter um ou vários projetos

Project – arquivo de projeto contendo os respectivos programas

## Termos comuns

- Stored procedure
- Query
- Provider
- Datatable
- Dataset

stored procedure – rotinas internas no SQL Server ou Oracle para melhorar a performance da aplicação

query – consulta representada por uma instrução SQL

provider – driver usado para acessar fontes de dados

datatable – tabela de dados na memória, oriunda ou não de uma fonte de dados

dataset – espaço na memória para acomodar as tabelas