UFES 2019 / 2

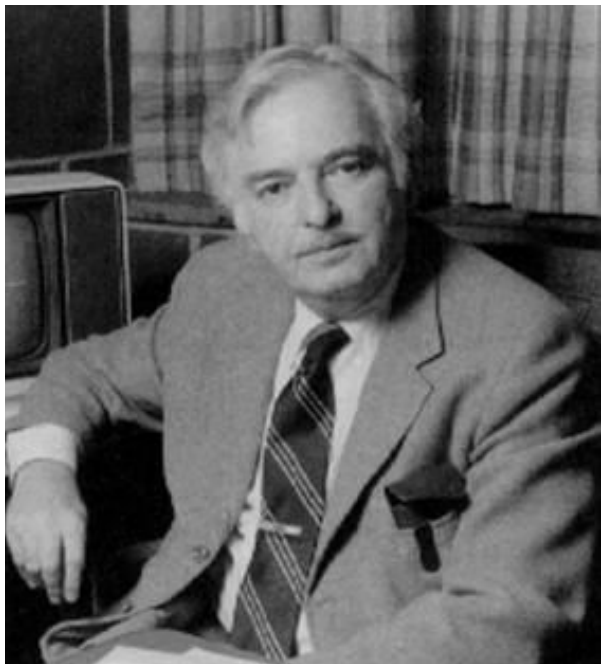
Linguagens de Programação

Visual Basic

Kaique
Leonardo
Mateus

História

Em 1963 foi criada a linguagem BASIC (*Beginners All-Purpose Symbolic Instruction Code*), no Darthmouth College (Hanover, New Hampshire)



John G. Kemeny

Thomas E. Kurtz

História

O objetivo era apresentar uma linguagem de fácil compreensão e uso, para que fosse ensinada a seus alunos.

FORTRAN e Assembly utilizavam rotinas de baixo nível.

História

Ao passar dos anos, junto da evolução do hardware das máquinas, surgiram novas variantes de BASIC

Atari BASIC ***(Atari 8-bit family)***

The standard cartridge-based interpreter for the Atari 400/800 personal computers and successors. On later machines, such as the Atari 800XL, this was built into the ROM.

Business Basic

name given collectively to BASIC variants which were specialized for business use on minicomputers in the 1970s.

Casio BASIC

used in Casio calculators

Famicom BASIC

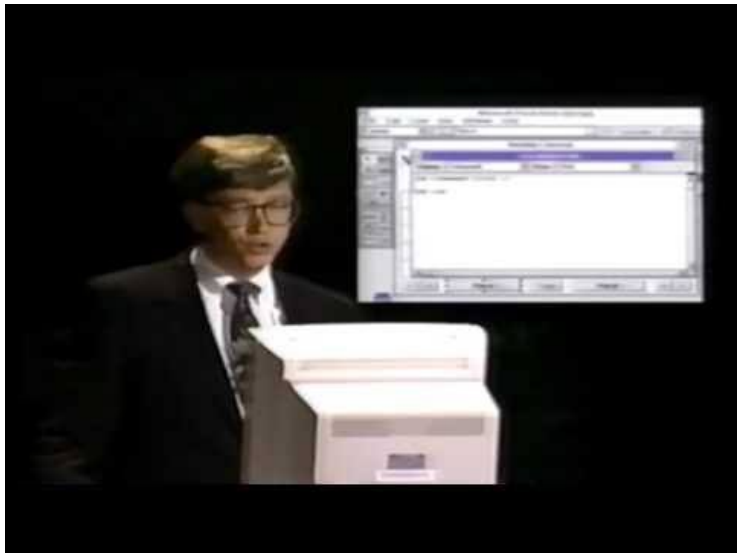
(Nintendo Entertainment System) — For the Nintendo Entertainment System.

História

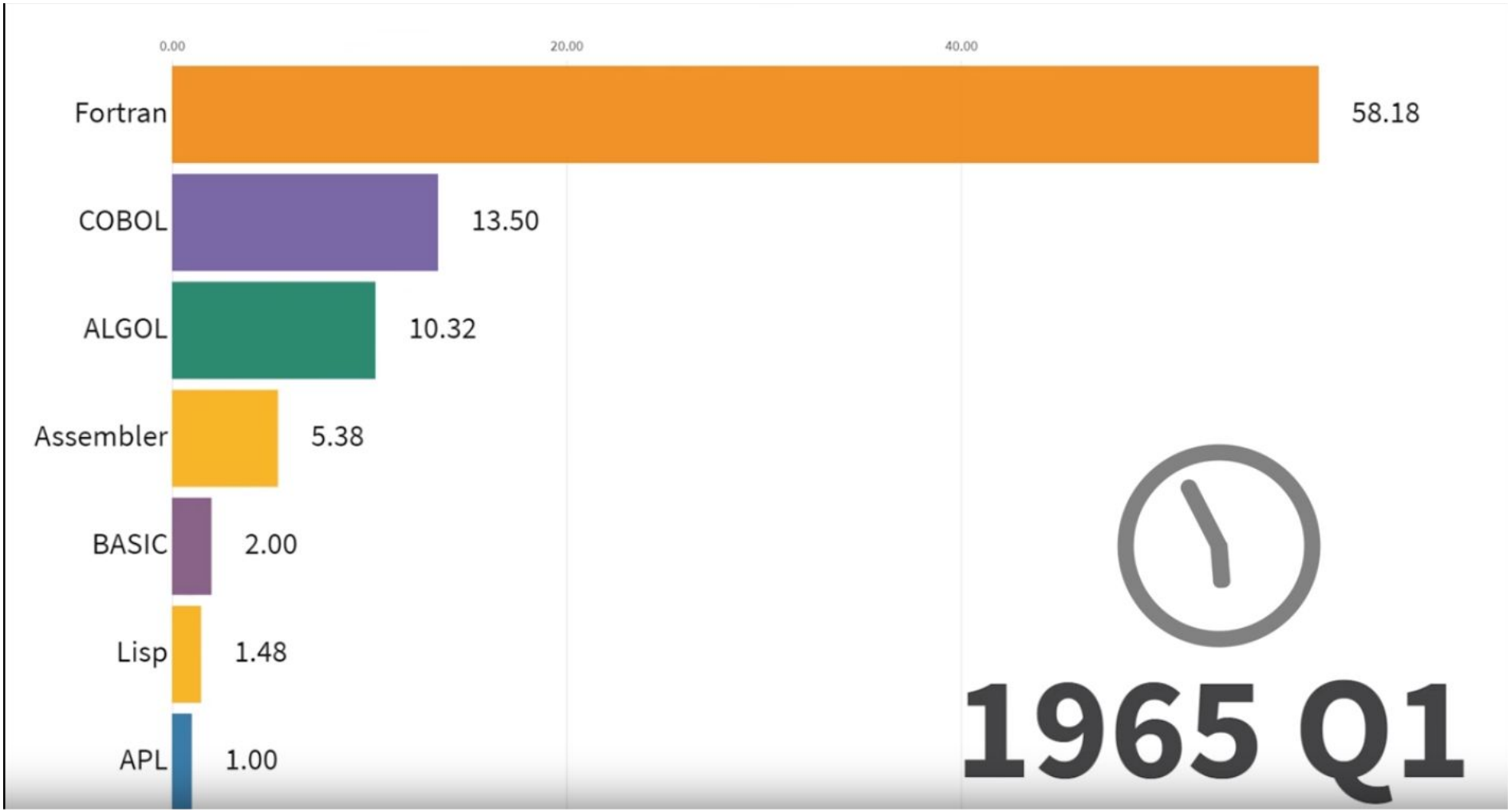
Em 1980, a Microsoft apresenta o GW-BASIC

2 anos depois ela lança o QuickBASIC

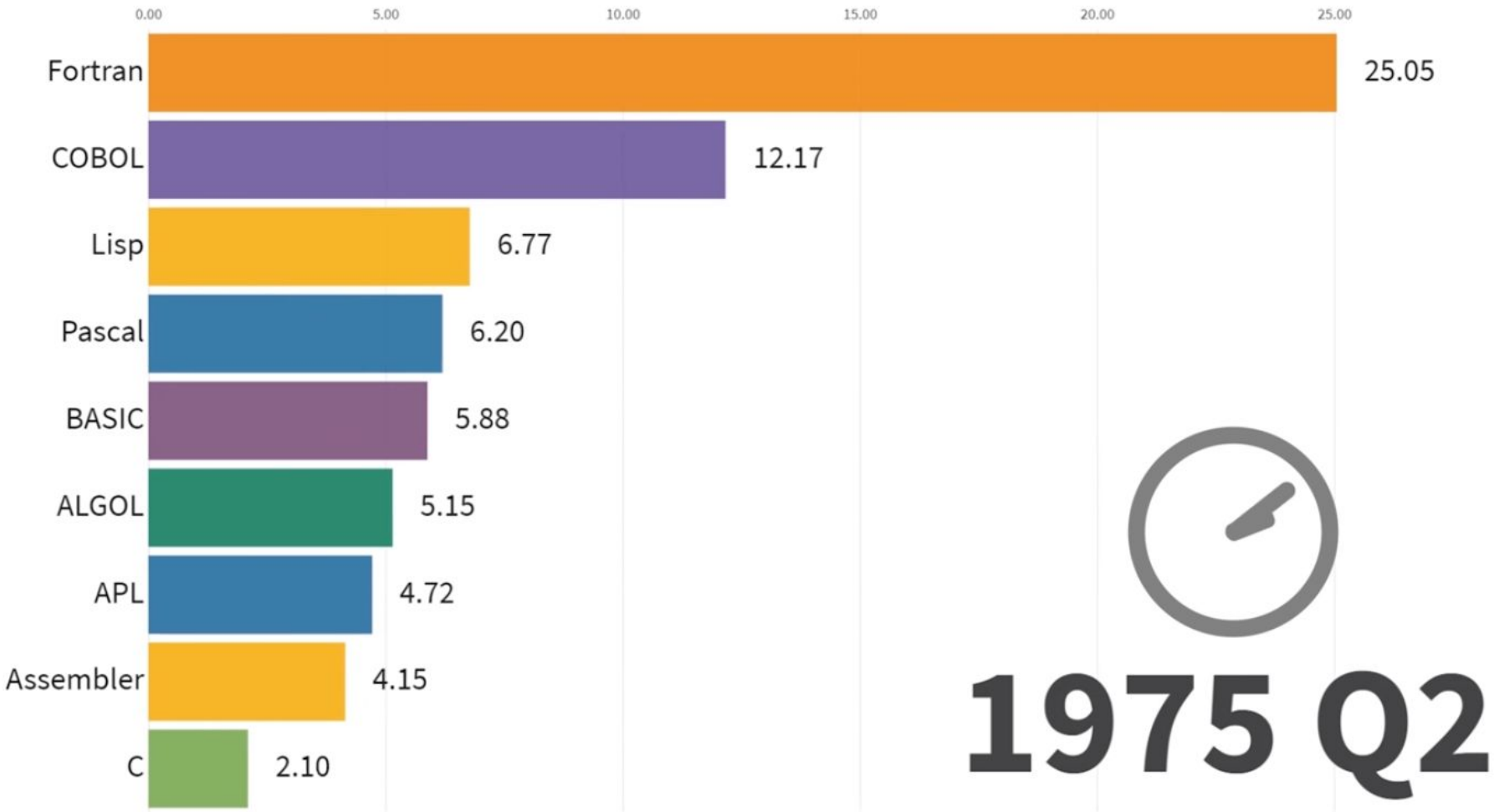
Até que em 1991, a Microsoft apresenta o Visual Basic



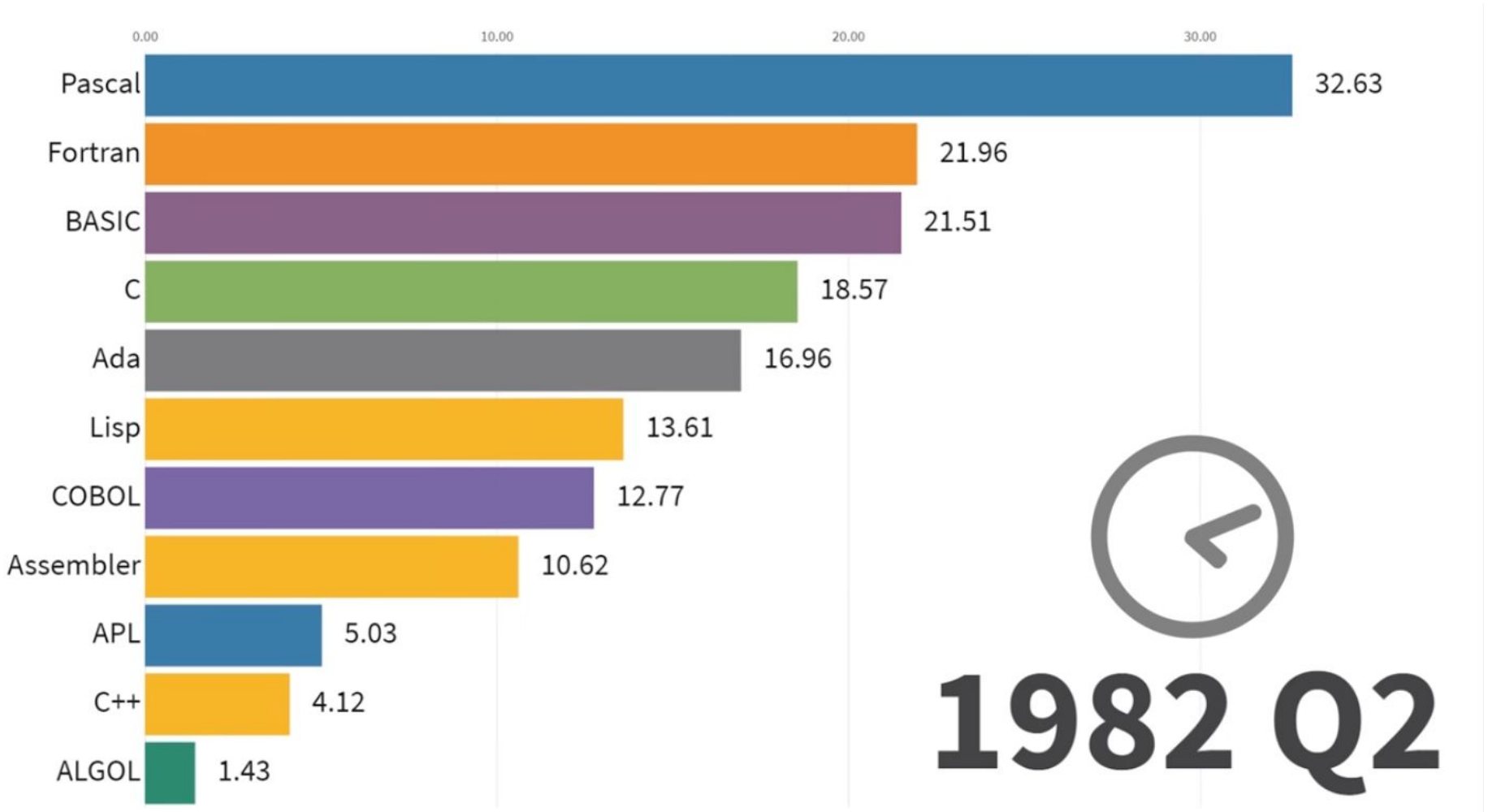
História


0.00
20.00
40.00
Fortran
58.18
COBOL
13.50
ALGOL
10.32
Assembler
5.38
BASIC
2.00
Lisp
1.48
APL
1.00
1965 Q1

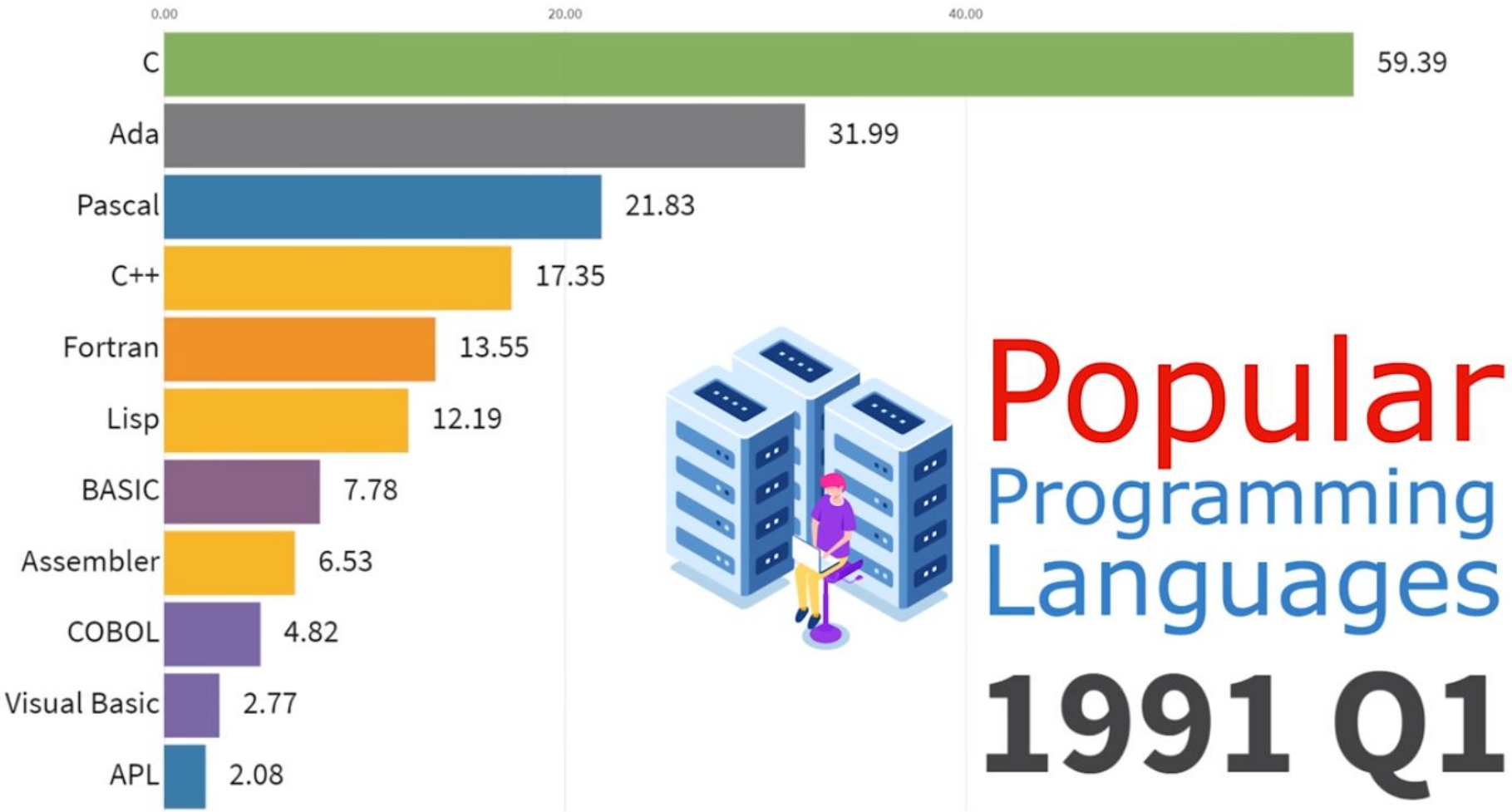
História


0.00
5.00
10.00
15.00
20.00
25.00
Fortran
25.05
COBOL
12.17
Lisp
6.77
Pascal
6.20
BASIC
5.88
ALGOL
5.15
APL
4.72
Assembler
4.15
C
2.10
1975 Q2

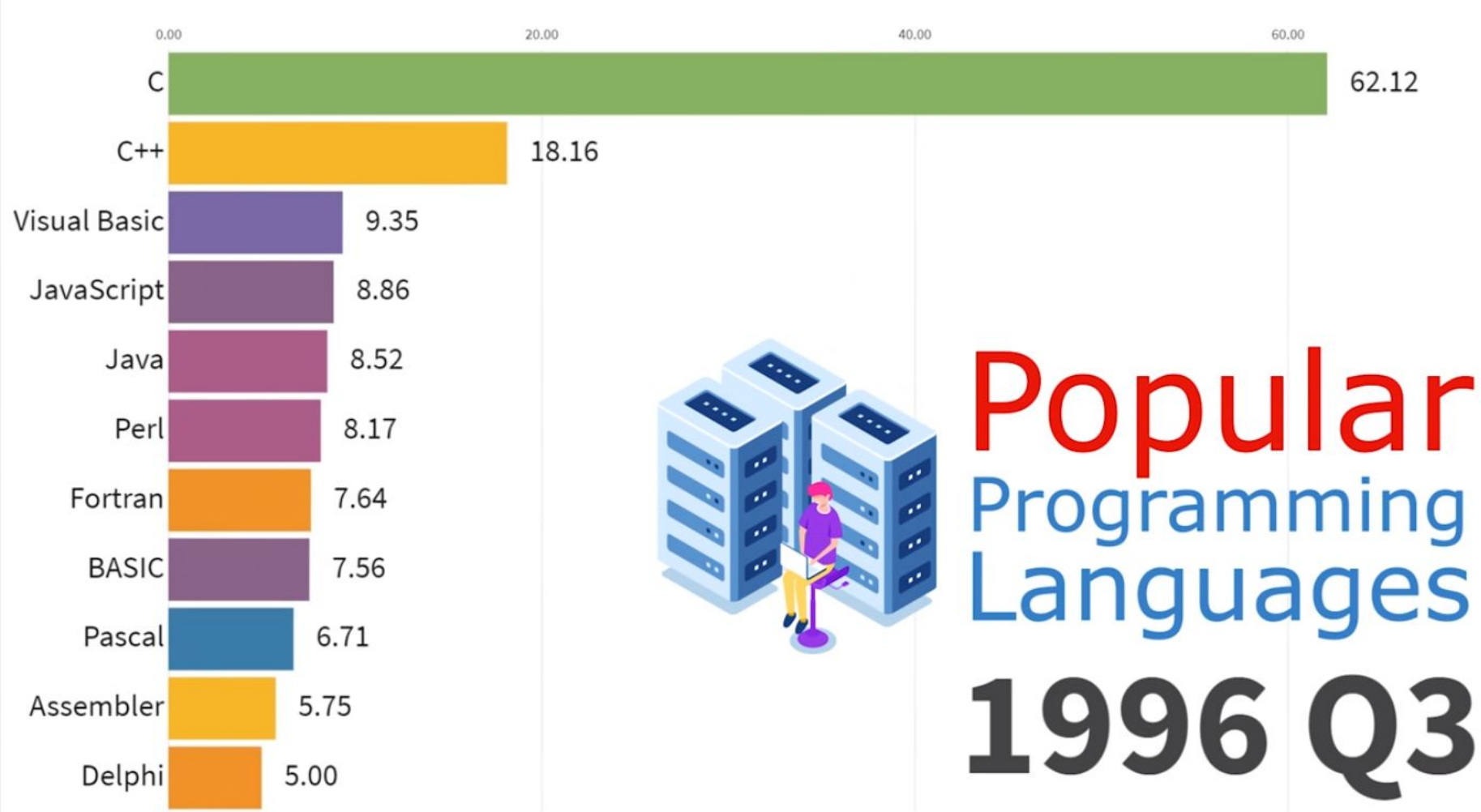
História


0.00
10.00
20.00
30.00
Pascal
32.63
Fortran
21.96
BASIC
21.51
C
18.57
Ada
16.96
Lisp
13.61
COBOL
12.77
Assembler
10.62
APL
5.03
C++
4.12
ALGOL
1.43
1982 Q2

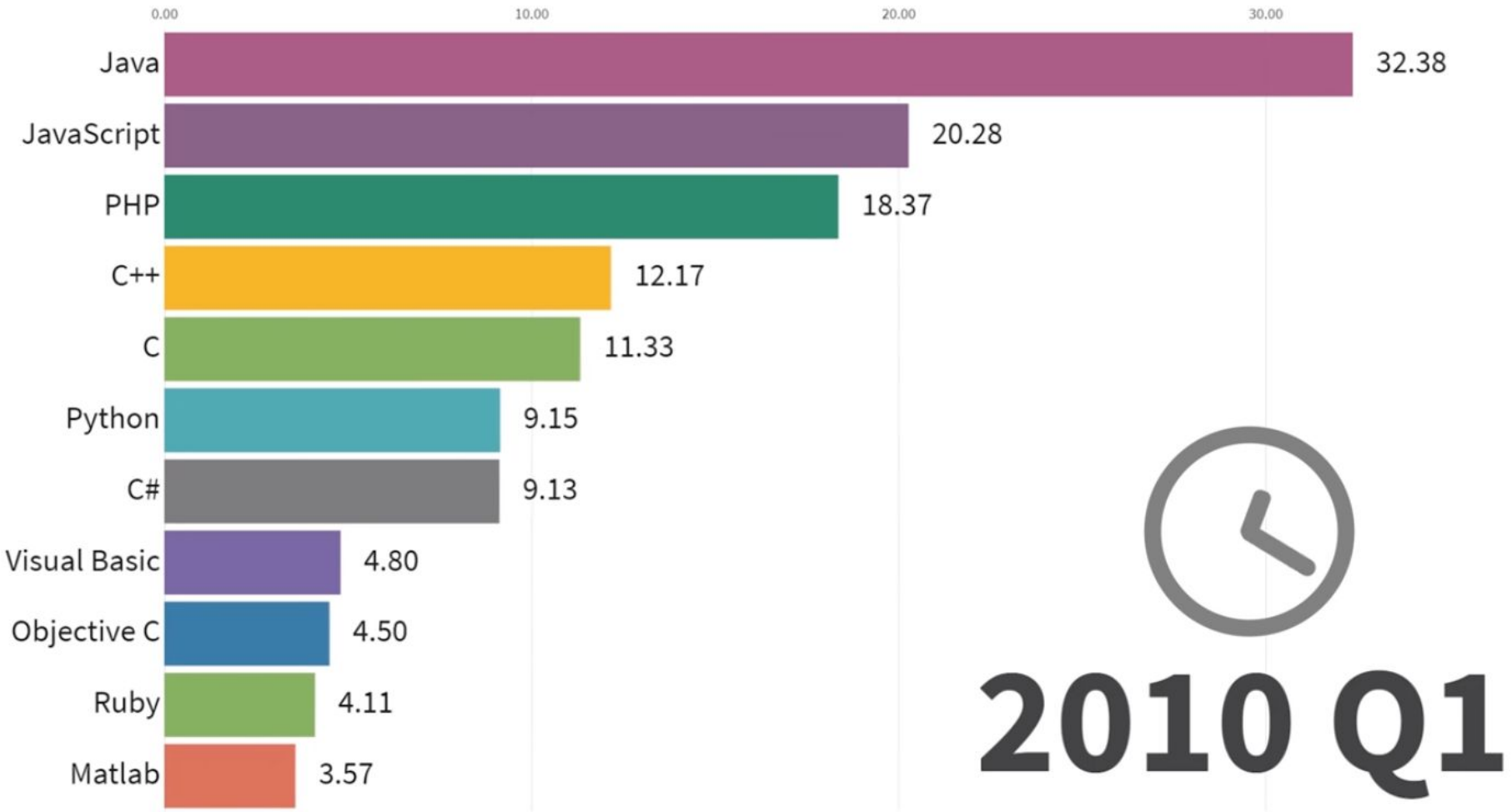
História


0.00
20.00
40.00
C
59.39
Ada
31.99
Pascal
21.83
C++
17.35
Fortran
13.55
Lisp
12.19
BASIC
7.78
Assembler
6.53
COBOL
4.82
Visual Basic
2.77
APL
2.08
Popular
Programming
Languages
1991 Q1

História


0.00
20.00
40.00
60.00
C
62.12
C++
18.16
Visual Basic
9.35
JavaScript
8.86
Java
8.52
Perl
8.17
Fortran
7.64
BASIC
7.56
Pascal
6.71
Assembler
5.75
Delphi
5.00
Popular
Programming
Languages
1996 Q3

História


0.00
10.00
20.00
30.00
Java
32.38
JavaScript
20.28
PHP
18.37
C++
12.17
C
11.33
Python
9.15
C#
9.13
Visual Basic
4.80
Objective C
4.50
Ruby
4.11
Matlab
3.57
2010 Q1

Introdução

O Visual Basic apresentado neste trabalho é o originalmente chamado Visual Basic .NET (VB.NET).

Introdução

VB.NET Surgiu em 2002 na versão 7 como o sucessor do Visual Basic original.

Alguns não consideram o VB.NET como uma nova versão do Visual Basic clássico, mas sim uma linguagem totalmente diferente.

Entre as muitas mudanças que ocorreram na transição para o .NET, destacam-se a inclusão de tratamento de exceções estruturado e curto circuito em expressões

Introdução

Em 2005 pararam de usar o .NET no nome da linguagem, ficando apenas Visual Basic, como era sua antecessora.

Progressão da linguagem

2002 (VB 7.0)

A primeira versão, Visual Basic .NET, depende da .NET Framework 1.0.

2003 (VB 7.1)

Visual Basic .NET 2003 foi lançada com o .NET Framework 1.1.

2005 (VB 8.0)

Depois do Visual Basic .NET 2003, a Microsoft retirou o ".NET" do nome do produto, chamando a versão de Visual Basic 2005.

2008 (VB 9.0)

Visual Basic 9.0 foi lançada com o .NET Framework 3.5 em novembro de 2007.

2010 (VB 10.0)

Em Abril de 2010, a Microsoft lançou o Visual Basic 2010.

Progressão da linguagem

2012 (VB 11.0)

Visual Basic 2012 foi lançada com o .NET Framework 4.5. Algumas mudanças significativas dessa versão foram:

- Programação assíncrona com o "async" e "await"
- Iteradores

2015 (VB 14.0)

Visual Basic 2015 foi lançada com o Visual Studio 2015.

2017 (VB 15.0)

Visual Basic 2017 foi lançada com o Visual Studio 2017.

Introdução

O Visual Basic atual (originalmente o VB.NET) é muito diferente do Visual Basic clássico. Apesar de ser o sucessor histórico, não é compatível com o Visual Basic 6.

É uma linguagem orientada a objetos, então ela suporta:

- Encapsulamento
- Polimorfismo
- Abstração
- Herança

Introdução

Paradigmas da linguagem:

- Estruturada
- Imperativa
- Orientada a objetos
- Declarativa
- Genérica
- Reflexiva
- Orientada a eventos

Introdução

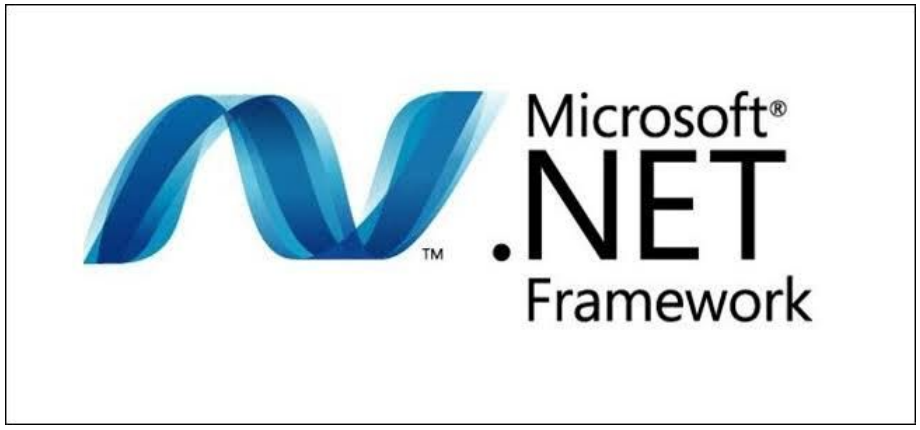
Ainda é uma linguagem de fácil compreensão até mesmo para iniciantes.

Visual Basic (assim chamaremos a partir de agora o originalmente VB.NET) roda no .NET Framework, o que faz com que programas feitos na linguagem sejam confiáveis e escaláveis.

Atualmente o Visual Basic está na versão 16, que foi lançada em 2019.

Breve comparação com C#

C# e Visual Basic foram as primeiras linguagens que a Microsoft para serem usadas com a framework .NET



As duas são desenvolvidas e administradas pela mesma equipe na Microsoft.

.NET

O .NET Framework é uma iniciativa da empresa Microsoft, que visa uma plataforma única para desenvolvimento e execução de sistemas e aplicações.

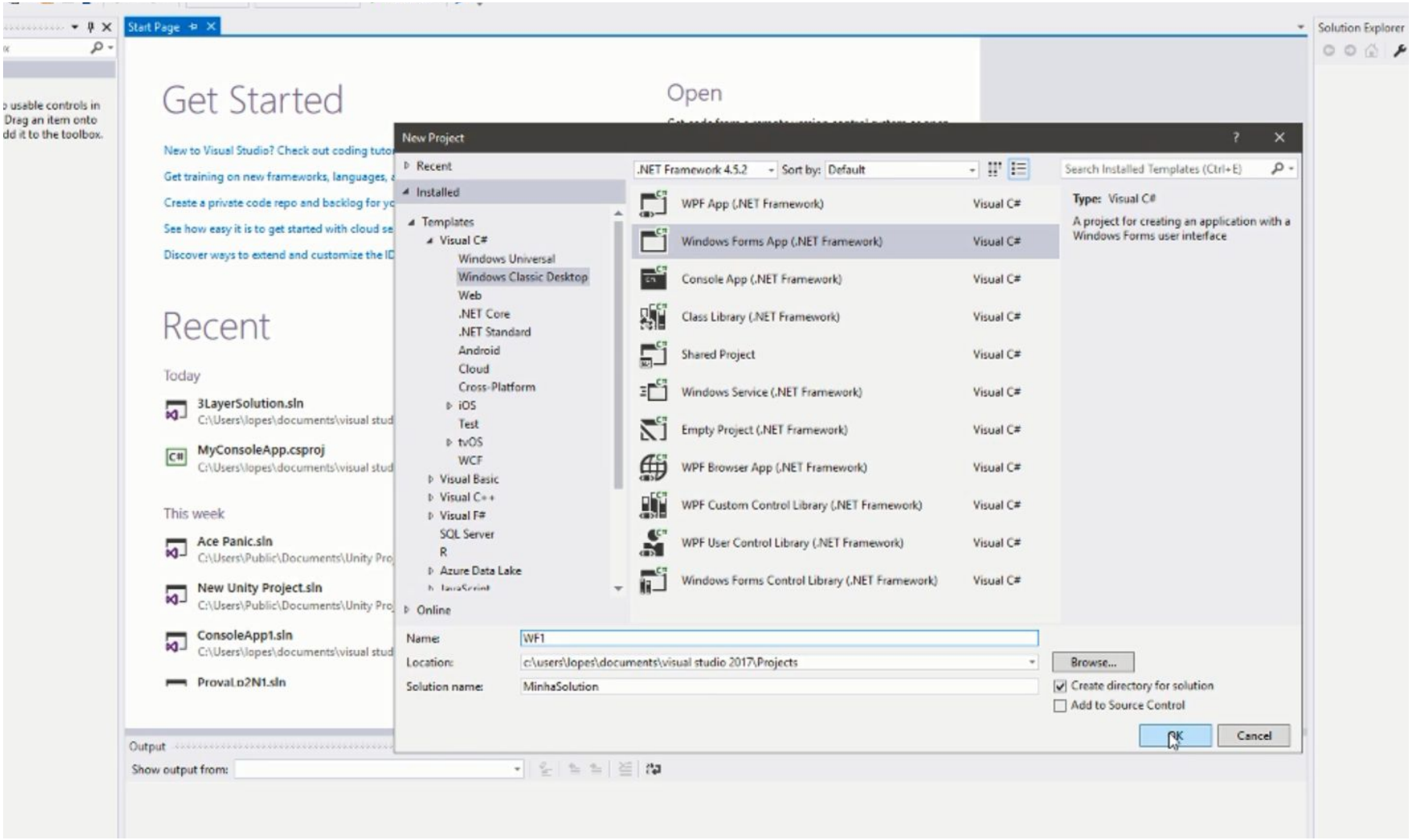
O .NET Framework consiste basicamente em dois componentes principai: ela é executada sobre uma Common Language Runtime - CLR (Ambiente de Execução Independente de Linguagem) interagindo com um Framework Class Library - FCL (Conjunto de Bibliotecas Unificadas).

.NET

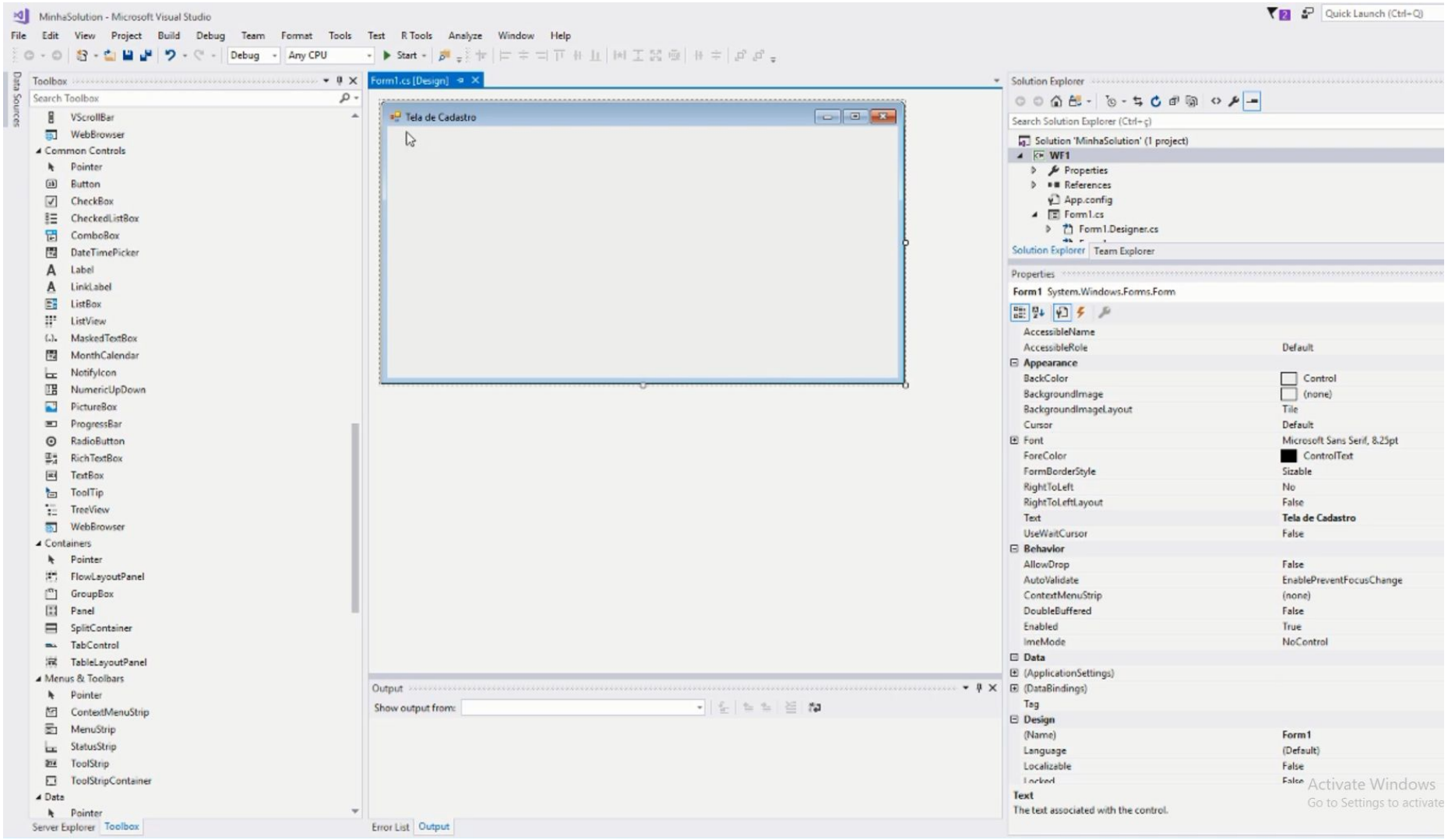
A CLR é capaz de executar através da Common Language Infrastructure, uma grande quantidade de linguagens de programação, interagindo entre si como se fossem uma única linguagem.

- Visual Basic
- C#
- Managed JScript
- F#:

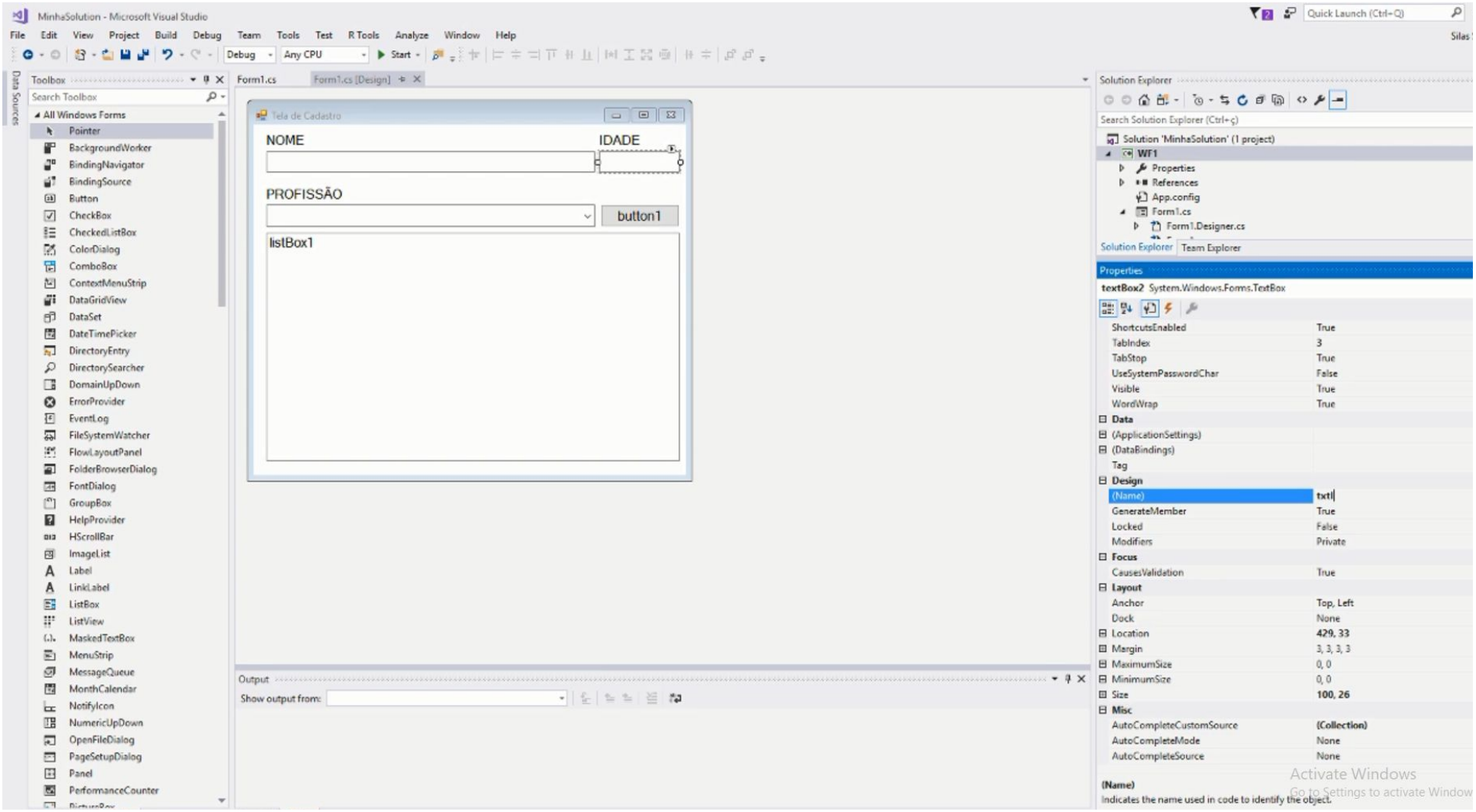
IDE


Start Page
Get Started
Open
Recent
Today
3LayerSolution.sln
MyConsoleApp.csproj
This week
Ace Panic.sln
New Unity Project.sln
ConsoleApp1.sln
Solution Explorer
New Project
Recent
Installed
Templates
Visual C#
Windows Universal
Windows Classic Desktop
Web
.NET Core
.NET Standard
Android
Cloud
Cross-Platform
iOS
Test
tvOS
WCF
Visual Basic
Visual C++
Visual F#
SQL Server
R
Azure Data Lake
Online
.NET Framework 4.5.2
Sort by: Default
Search Installed Templates (Ctrl+E)
WPF App (.NET Framework)
Windows Forms App (.NET Framework)
Console App (.NET Framework)
Class Library (.NET Framework)
Shared Project
Windows Service (.NET Framework)
Empty Project (.NET Framework)
WPF Browser App (.NET Framework)
WPF Custom Control Library (.NET Framework)
WPF User Control Library (.NET Framework)
Windows Forms Control Library (.NET Framework)
Visual C#
Type: Visual C#
A project for creating an application with a Windows Forms user interface
Name: WF1
Location: c:\users\lopes\documents\visual studio 2017\Projects
Solution name: MinhaSolution
Browse...
Create directory for solution
Add to Source Control
OK
Cancel
Output
Show output from:

IDE


MinhaSolution - Microsoft Visual Studio
File Edit View Project Build Debug Team Format Tools Test R Tools Analyze Window Help
Debug
Any CPU
Start
Quick Launch (Ctrl+Q)
Toolbox
Search Toolbox
VScrollBar
WebBrowser
Common Controls
Pointer
Button
CheckBox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MonthCalendar
NotifyIcon
NumericUpDown
PictureBox
ProgressBar
RadioButton
RichTextBox
TextBox
ToolTip
TreeView
WebBrowser
Containers
Pointer
FlowLayoutPanel
GroupBox
Panel
SplitContainer
TabControl
TableLayoutPanel
Menus & Toolbars
Pointer
ContextMenuStrip
MenuStrip
StatusStrip
ToolStrip
ToolStripContainer
Data
Pointer
Server Explorer
Toolbox
Form1.cs [Design]
Tela de Cadastro
Output
Show output from:
Error List
Output
Solution Explorer
Search Solution Explorer (Ctrl+ç)
Solution 'MinhaSolution' (1 project)
WF1
Properties
References
App.config
Form1.cs
Form1.Designer.cs
Solution Explorer
Team Explorer
Properties
Form1 System.Windows.Forms.Form
AccessibleName
AccessibleRole
Default
Appearance
BackColor
Control
BackgroundImage
(none)
BackgroundImageLayout
Tile
Cursor
Default
Font
Microsoft Sans Serif, 8.25pt
ForeColor
ControlText
FormBorderStyle
Sizable
RightToLeft
No
RightToLeftLayout
False
Text
Tela de Cadastro
UseWaitCursor
False
Behavior
AllowDrop
False
AutoValidate
EnablePreventFocusChange
ContextMenuStrip
(none)
DoubleBuffered
False
Enabled
True
ImeMode
NoControl
Data
(ApplicationSettings)
(DataBindings)
Tag
Design
(Name)
Form1
Language
(Default)
Localizable
False
Text
The text associated with the control.
Activate Windows
Go to Settings to activate

IDE


MinhaSolution - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test R Tools Analyze Window Help
Debug
Any CPU
Start
Quick Launch (Ctrl+Q)
Toolbox
Search Toolbox
All Windows Forms
Pointer
BackgroundWorker
BindingNavigator
BindingSource
Button
CheckBox
CheckedListBox
ColorDialog
ComboBox
ContextMenuStrip
DataGridView
DataSet
DateTimePicker
DirectoryEntry
DirectorySearcher
DomainUpDown
ErrorProvider
EventLog
FileSystemWatcher
FlowLayoutPanel
FolderBrowserDialog
FontDialog
GroupBox
HelpProvider
HScrollBar
ImageList
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MenuStrip
MessageQueue
MonthCalendar
NotifyIcon
NumericUpDown
OpenFileDialog
PageSetupDialog
Panel
PerformanceCounter
Form1.cs
Form1.cs [Design]
Tela de Cadastro
NOME
IDADE
PROFISSÃO
button1
listBox1
Output
Show output from:
Solution Explorer
Search Solution Explorer (Ctrl+ç)
Solution 'MinhaSolution' (1 project)
WF1
Properties
References
App.config
Form1.cs
Form1.Designer.cs
Solution Explorer
Team Explorer
Properties
textBox2 System.Windows.Forms.TextBox
ShortcutsEnabled True
TabIndex 3
TabStop True
UseSystemPasswordChar False
Visible True
WordWrap True
Data
(ApplicationSettings)
(DataBindings)
Tag
Design
(Name) txtI
GenerateMember True
Locked False
Modifiers Private
Focus
CausesValidation True
Layout
Anchor Top, Left
Dock None
Location 429, 33
Margin 3, 3, 3, 3
MaximumSize 0, 0
MinimumSize 0, 0
Size 100, 26
Misc
AutoCompleteCustomSource (Collection)
AutoCompleteMode None
AutoCompleteSource None
(Name)
Indicates the name used in code to identify the object.
Activate Windows
Go to Settings to activate Windows

Introdução

Um exemplo de *Console application* mostrando as características principais de um programa em VB:

```
Imports System
Module Module1

    'Prints Hello World
    Sub Main()

        Console.WriteLine("Hello World")
        Console.ReadKey()

    End Sub
End Module
```

Introdução

Declarações de namespace

```
Imports System
Module Module1

    'Prints Hello World
    Sub Main()

        Console.WriteLine("Hello World")
        Console.ReadKey()

    End Sub
End Module
```

Introdução

Módulos são uma divisão de código.Podem conter qualquer tipo de objeto, como constantes/variáveis, métodos, classes. Porém não podem ser instanciados.

```
Imports System
Module Module1

    'Prints Hello World
    Sub Main()

        Console.WriteLine("Hello World")
        Console.ReadKey()

    End Sub
End Module
```

Introdução

Linha de comentário. Os comentários são sinalizados pelo caractere ‘

```
Imports System
Module Module1

    'Prints Hello World
    Sub Main()

        Console.WriteLine("Hello World")
        Console.ReadKey()

    End Sub
End Module
```

Introdução

Declaração de um procedimento, ou sub-rotina, ou função. Cada módulo pode ter várias sub-rotinas

```
Imports System
Module Module1

    'Prints Hello World
    Sub Main()

        Console.WriteLine("Hello World")
        Console.ReadKey()

    End Sub
End Module
```

Introdução

Aqui é chamado um método da classe Console, definida dentro do namespace da System

```
Imports System
Module Module1

    'Prints Hello World
    Sub Main()

        Console.WriteLine("Hello World")
        Console.ReadKey()

    End Sub
End Module
```

Introdução

Para que a janela do terminal não feche automaticamente

```
Imports System
Module Module1

    'Prints Hello World
    Sub Main()

        Console.WriteLine("Hello World")
        Console.ReadKey()

    End Sub
End Module
```

Introdução

Fecha main e encerra o módulo

```
Imports System
Module Module1

    'Prints Hello World
    Sub Main()

        Console.WriteLine("Hello World")
        Console.ReadKey()

    End Sub
End Module
```

Compilando

Você pode compilar pelo terminal se não quiser usar a IDE

• Compilar File. vb e criar arquivo. exe	vbc -reference:Microsoft.VisualBasic.dll File.vb
• Compilar File. vb e criar File. dll	vbc -target:library File.vb
• Compilar File. vb e criar My. exe	vbc -out:My.exe File.vb
• Compila todos os arquivos de VB do diretório, Com otimizações ligadas e DEBUG definido, produzindo File2.exe	vbc -define:DEBUG=1 -optimize -out:File2.exe *.vb

Funções

A sintaxe para declarar um procedimento Function é a seguinte:

```
[Modifiers] Function FunctionName [(ParameterList)] As ReturnType
    [Statements]
End Function
```

Funções

Exemplos

```
Function yesterday() As Date
End Function

Function findSqrt(ByVal radicand As Single) As Single
End Function
```

Funções

Retornando valores

```
Function FunctionName [(ParameterList)] As ReturnType
    ' The following statement immediately transfers control back
    ' to the calling code and returns the value of Expression.
    Return Expression
End Function
```

Funções

Declaração e chamada

```
Function Hypotenuse(side1 As Double, side2 As Double) As Double
    Return Math.Sqrt((side1 ^ 2) + (side2 ^ 2))
End Function


Dim testLength, testHypotenuse As Double
testHypotenuse = Hypotenuse(testLength, 10.7)
```

Sintaxe do Dim

Para declarações de variáveis, usa-se a expressão *Dim*

```
Dim myVar, nextVar, thirdVar
```

Essa linha cria 3 variáveis de um tipo Variant

Sintaxe do Dim

Agora a variável “myAnswer” recebe o tipo String

```
Dim myAnswer As String
```

Sintaxe do Dim

Se você quiser declarar mais de uma variável na mesma linha, precisa declarar o tipo em cada uma delas

```
Dim x As Integer, y As Integer, z As Integer
```

Sintaxe do Dim

Nesse exemplo apenas z foi declarado como Integer, x e y são Variants

```
Dim x, y, z As Integer
```

Sintaxe do Dim

Mas se você quiser um atalho para diminuir a declaração, pode fazer desse jeito

```
Dim x%, y%, z as Integer
```

Sintaxe do Dim

Mas se você quiser um atalho para diminuir a declaração, pode fazer desse jeito

```
Dim x%, y%, z as Integer
```

Usam-se diferentes atalhos para diferentes tipos:

% -integer
& -long
@ -currency

-double
! -single
$ -string

Sintaxe do Dim

Para declarar arrays, é necessário o uso de parênteses

```
Dim myArray()
```

Declarando constantes

```
Public Const conAge As Integer = 34
```

Nomes de elementos declarados

Um nome de elemento em Visual Basic deve observar as seguintes regras:

- Deve começar com um caractere alfabético ou um sublinhado (_).
- Deve conter apenas caracteres alfabéticos, dígitos decimais e sublinhados.
- Deve conter pelo menos um caractere alfabético ou dígito decimal se começar com um sublinhado.
- Não deve ter mais de 255 caracteres.

Nomes de elemento declarados

O exemplo a seguir mostra dois nomes de elemento válidos.

aB123__45

_567

Nomes de elemento declarados

O exemplo a seguir mostra alguns nomes de elementos inválidos.

_

12ABC

xyz$wv

Nomes de elemento declarados

O exemplo a seguir mostra alguns nomes de elementos inválidos.

_

12ABC

xyz$wv

Nomes de elementos que começam com um sublinhado (_) não fazem parte da independência de linguagem e dos componentes independentes de linguagem (CLS), portanto, o código em conformidade com CLS não pode usar um componente que define tais nomes. No entanto, um sublinhado em qualquer outra posição em um nome de elemento é compatível com CLS.

Distinção de maiúsculas e minúsculas

Os nomes de elementos em Visual Basic não diferenciam maiúsculas de minúsculas. Isso significa que, quando o compilador compara dois nomes que diferem somente em maiúsculas e minúsculas, ele os interpreta como o mesmo nome. Por exemplo, ele considera ABC e abc para fazer referência ao mesmo elemento declarado.

Distinção de maiúsculas e minúsculas

No entanto, o Common Language Runtime (CLR) usa a associação que diferencia maiúsculas de minúsculas.

Por exemplo, se você definir uma classe com um elemento chamado ABC e outros assemblies fizerem uso de sua classe por meio da Common Language Runtime, eles deverão se referir ao elemento como ABC.

Se, posteriormente, você recompilar sua classe e alterar o nome do elemento para abc, os outros assemblies que usam sua classe não poderão mais acessar esse elemento.

Palavras reservadas

AddHandler	AddressOf	Alias	And
AndAlso	As	Boolean	ByRef
Byte	ByVal	Call	Case
Catch	CBool	CByte	CChar
CDate	CDbl	CDec	Char
CInt	Class Constraint	Class Statement	CLng
CObj	Const	Continue	CSByte
CShort	CSng	CStr	CType
CUInt	CULng	CUShort	Date
Decimal	Declare	Default	Delegate
Dim	DirectCast	Do	Double
Each	Else	ElseIf	End Statement

Palavras reservadas

End <keyword>	`EndIf`	Enum	Erase
Error	Event	Exit	False
Finally	For (in For...Next)	For Each...Next	Friend
Function	Get	GetType	GetXMLNamespace
Global	`GoSub`	GoTo	Handles
If	If()	Implements	Implements Statement
Imports (.NET Namespace and Type)	Imports (XML Namespace)	In	In (Generic Modifier)
Inherits	Integer	Interface	Is
IsNot	Let	Lib	Like
Long	Loop	Me	Mod
Module	Module Statement	MustInherit	MustOverride
MyBase	MyClass	NameOf	Namespace

Palavras reservadas

Narrowing	New Constraint	New Operator	Next
Next (in Resume)	Not	Nothing	NotInheritable
NotOverridable	Object	Of	On
Operator	Option	Optional	Or
OrElse	Out (Generic Modifier)	Overloads	Overridable
Overrides	ParamArray	Partial	Private
Property	Protected	Public	RaiseEvent
ReadOnly	ReDim	REM	RemoveHandler
Resume	Return	SByte	Select
Set	Shadows	Shared	Short
Single	Static	Step	Stop
String	Structure Constraint	Structure Statement	Sub

Palavras reservadas

SyncLock	Then	Throw	To
True	Try	TryCast	TypeOf...Is
UInteger	ULong	UShort	Using
`Variant`	`Wend`	When	While
Widening	With	WithEvents	WriteOnly
Xor	#Const	#Else	#ElseIf
#End	#If	=	&
&=	*	*=	/
/=	\	\=	^
^=	+	+=	-
-=	>> Operator	>>= Operator	<<
<<=			

Tipos de dados

Boolean	Varia	True ou False
Byte	1 byte	0 a 255 (unsigned)
Char	2 bytes	65535 (unsigned)
Date	8 bytes	from 0:00:00 January 1, 0001 to 11:59:59 PM of December 31, 9999
Integer	4 bytes	-2,147,483,648 to 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (signed)
String	Varia	0 até cerca de 2 bilhões de caracteres Unicode.
Variant	16 bytes para números 24 bytes + tamanho da string, para strings	Mesmo do Double e da String

Tipos de dados

SByte	1 byte	True ou False
Object	4 ou 8 bytes	Qualquer tipo pode ser armazenado em uma variável do tipo Object
Short	2 bytes	-32.768 a 32.767
Single	4 bytes	-3.4028235 e + 38 a-1.401298 E-45 † para valores negativos; 1.401298 e-45 a 3.4028235 E + 38 † para valores positivos
UInteger	4 bytes	0 a 4.294.967.295
Definido pelo usuário (estrutura)	Varia	Cada membro da estrutura tem um intervalo determinado por seu tipo de dados e independente dos intervalos dos outros membros

Tipos de dados

Double	8 bytes	-1.79769313486231570 e + 308 a-4.94065645841246544 E-324 † para valores negativos; 4.94065645841246544 e-324 a 1.79769313486231570 E + 308 † para valores positivos
Decimal	16 bytes	0 a +/-79228162514264337593543 950335 (+/-7.9...E + 28) † sem ponto decimal; 0 a +/-7.92281625142643375935 43950335 com 28 casas à direita do decimal; o menor número diferente de zero é +/-0, 1 (+/-1E-28) †

Tipos de dados - Enum

```
Public Class Egg
    Enum EggSizeEnum
        Jumbo
        ExtraLarge
        Large
        Medium
        Small
    End Enum

    Public Sub Poach()
        Dim size As EggSizeEnum

        size = EggSizeEnum.Medium
        ' Continue processing...
    End Sub
End Class
```

Tipos de dados - Conversão

CBool (expression): converte o parâmetro para um Boolean.

CDate(expression): converte o parâmetro para um Date.

CDbl(expression): converte o parâmetro para um Double.

CByte (expression): converte o parâmetro para um byte.

CChar(expression): converte o parâmetro para um Char.

CLng(expression): converte o parâmetro para um Long.

CDec(expression): converte o parâmetro para um Decimal.

CInt(expression): converte o parâmetro para umn Integer.

CObj(expression): converte o parâmetro para umn Object.

CStr(expression): converte o parâmetro para um String.

CSByte(expression): converte o parâmetro para um Byte.

CShort(expression): converte o parâmetro para um Short.

Tipos de dados - Verificação

Você pode verificar os tipos dos dados com as seguintes funções:

- IsArray
- IsDate
- IsEmpty
- IsError
- IsMissing
- IsNull
- IsNumeric
- IsObject

Variáveis e constantes

Toda variável é inicializada para um valor padrão.

```
Module Program
   Sub Main()
      Dim var As Integer
      Console.WriteLine(var)
   End Sub
End Module
```

```
Output: 0
```

Variáveis e constantes

Linguagem não permite manipulação de ponteiros.

Trabalha com referências, permite variáveis sinônimas.

```
Module Program
    Sub Main()
        Dim classe = New With {.Atributo1 = "str1",
            .Atributo2 = "str2"}
        Dim var = classe
        var.Atributo1 = "modificado"
        Console.WriteLine(var)
    End Sub
End Module
```

```
Output: { Atributo1 = modificado, Atributo2 = str2 }
```

Variáveis e constantes

Tipos de variáveis locais podem ser declarados ou inferidos.

```
Module Program
    Sub Main()
        Dim inteiro1 As Integer
        inteiro1 = 1
        Dim inteiro2 = 2
        Console.WriteLine(inteiro1.GetType().ToString()
            + " " + inteiro2.GetType().ToString())
    End Sub
End Module
```

```
Output: System.Int32 System.Int32
```

Variáveis e constantes

Modificadores de acesso:

```
Module Program
    Structure Estrutura
        Public x As Integer
    End Structure
    Sub Main()
        Dim var As Estrutura
        var.x = 1
        Console.WriteLine(var.x)
    End Sub
End Module
```

```
Output: 1
```

Variáveis e constantes

Operador **`Static`**:

```
Module Program
    Sub Main()
        Dim var As Integer = 0
        For i As Integer = 0 To 10
            Resgatar(var)
        Next
        Console.WriteLine(var)
    End Sub
    Sub Resgatar(ByRef var As Integer)
        Static x As Integer = -1
        x += 1
        var = x
    End Sub
End Module
```

Output: 10

Variáveis e constantes - Escopo

Existem dois tipos de escopo de visibilidade para variáveis:

Local

Dentro de procedimentos.

Módulo

Dentro de módulos e fora de procedimentos.

Variáveis e constantes - Escopo

Exemplo:

```
Module Program
    Dim s1 As String = "str1"

    Sub P1()
        Console.WriteLine(s1)  ' Output: str1
    End Sub
    Sub P2()
        Console.WriteLine(s2)  ' Erro, não compila
    End Sub
    Sub Main()
        Dim s2 As String = "str2"
        P1()
        P2()
    End Sub
End Module
```

Variáveis e constantes - Tempo de vida

O tempo de vida de uma variável **inicia** com o início do procedimento no qual ela é inicializada.

O tempo de vida **finaliza** com o término do procedimento, com algumas exceções:

`Static`

Variável de alocação estática. Só é liberada ao término do aplicativo.

`Shared`

Atributo compartilhado entre instâncias de uma classe. Só é liberada ao término do aplicativo.

Variáveis e constantes - Tempo de vida

Exemplo:

```
Module Program
    Class Classe
        Public Shared s As String = "str"
    End Class
    Sub P1()
        Dim obj1 As Classe
        Console.WriteLine(obj1.s)
        obj1.s += "_modificado"
    End Sub
    Sub Main()
        P1()
        Dim obj2 As Classe
        Console.WriteLine(obj2.s)
    End Sub
End Module
```

```
Output:   str
          str_modificado
```

Variáveis e constantes - Constantes

Constantes são criadas com o uso da instrução **`Const`**.

O escopo de visibilidade e tempo de vida é o mesmo de uma variável em seu lugar.

Constantes predefinidas:

- Compilação condicional. Ex: **`VBC_VER`**;
- Impressão e exibição. Ex: **`vbLf`**;
- Enumerações. Ex: **`FirstDayOfWeek`**.

Variáveis e constantes - Coletor de lixo

A linguagem possui coletor de lixo.

Coletor gerencia alocação e liberação de memória.

Momento de coleta é decidido com base nas alocações.

Mecanismo de gerações.

Pode ser forçado chamando **`GC.Collect()`**.

Variáveis e constantes - Persistência

Escrita e leitura de arquivo binário com as classes **`BinaryWriter`** e **`BinaryReader`** do namespace **`System.IO`**:

```
Imports System.IO

Module Program
    Structure Ponto
        Dim X As Double
        Dim Y As Double
    End Structure
    Sub Main()
        Dim P1 As Ponto
        P1.X = 0.5
        P1.Y = 3.1
```

Variáveis e constantes - Persistência

Cria arquivo binário de nome "MeuPonto" guardando informações da estrutura:

```
Try
    Dim MeuPonto As FileStream =
        New FileStream("MeuPonto", FileMode.Create)
    BW = New BinaryWriter(MeuPonto)
    BW.Write(P1.X)
    BW.Write(P1.Y)
    BW.Close()
Catch ex As IOException
    Console.WriteLine(ex.Message())
End Try
```

Variáveis e constantes - Persistência

Lê arquivo binário de nome "MeuPonto" e atribui valores em P2:

```
Dim P2 As Ponto
Dim BR As BinaryReader
Try
    Dim MeuPonto As FileStream =
        New FileStream("MeuPonto", FileMode.Open)
    BR = New BinaryReader(MeuPonto)
    P2.X = BR.ReadDouble()
    P2.Y = BR.ReadDouble()
    Console.WriteLine(P2.X)
    Console.WriteLine(P2.Y)
    BR.Close()
Catch ex As IOException
    Console.WriteLine(ex.Message())
End Try
```

```
Output:  0.5
         3.1
```

Variáveis e constantes - Persistência

A linguagem possui suporte para serialização.

Serialização **binária** `(System.Runtime.Serialization)`

- Compacta
- Armazenamento
- Fluxos de rede

Serialização **XML** `(System.Xml.Serialization)`

- Legibilidade
- Flexibilidade

Expressões e comandos - Operadores

Operações aritméticas:

```
Dim x As Integer
x = 67 + 34   'x = 101
x = 32 - 12   'x = 20
```

```
Dim y As Double
y = 45 * 55.23   'y = 2485.35
y = 32 / 23      'y = 1.391...
```

```
Dim x As Integer = 65
Dim y As Integer
y = -x          'y = -65
```

```
Dim z As Double
z = 23 ^ 3       'z = 12167
```

Expressões e comandos - Operadores

Operações aritméticas:

```
Dim k As Integer
k = 23 \ 5    'k = 4
```

```
Dim x As Integer = 100
Dim y As Integer = 6
Dim z As Integer
z = x Mod y   'z = 4
```

```
Dim a As Double = 100.3
Dim b As Double = 4.13
Dim c As Double
c = a Mod b   'c = 1.17999...
```

Expressões e comandos - Operadores

Operações aritméticas:

Divisão por zero com tipos integrais (Integer, Long, Byte, etc.) lança **`DivideByZeroException`**.

Dividendo	Divisor	Valor dividendo	Resultado
Double	Double	0	`NaN`
Double	Double	> 0	`PositiveInfinity`
Double	Double	< 0	`NegativeInfinity`

Expressões e comandos - Operadores

Operações aritméticas:

```
Dim lResult, rResult As Integer
Dim pattern As Integer = 12   '0000 1100.
lResult = pattern << 3        'lResult = 96.
rResult = pattern >> 2        'rResult = 3.
```

O tipo de dados do operando de padrão deve ser `SByte`, `Byte`, `Short`, `UShort`, `Integer`, `UInteger`, `Long` ou `ULong`.

Expressões e comandos - Operadores

Operações de comparação:

```
Console.WriteLine(23 = 44)        'False
Console.WriteLine(23 = 23)        'True

Console.WriteLine(23 <> 44)       'True
Console.WriteLine(23 <> 23)       'False

Console.WriteLine(23 < 44)        'True
Console.WriteLine(23 < 23)        'False

Console.WriteLine(23 > 44)        'False
Console.WriteLine(23 > 12)        'True
```

Expressões e comandos - Operadores

Operações de comparação:

```
Console.WriteLine(23 <= 44)          'True
Console.WriteLine(23 <= 23)          'True

Console.WriteLine(23 >= 44)          'False
Console.WriteLine(23 >= 23)          'True

Console.WriteLine("73" < "9")        'True
Console.WriteLine("734" = "734")     'True
Console.WriteLine("aaa" > "aa")      'True
```

Ordem configurada por instrução de compilação.

Expressões e comandos - Operadores

Operações de comparação:

Ao comparar objetos, são usados os operadores **`Is`** e **`IsNot`**

```
Dim x As New customer()
Dim y As New customer()
If x Is y Then
    ' Código para caso apontam para a mesma instância.
End If
If x IsNot y Then
    ' Código para caso apontam para instâncias diferentes.
End If
```

Expressões e comandos - Operadores

Operações de comparação:

Comparação de tipos pode ser feita com **`TypeOf`**

```
Dim x As System.Windows.Forms.Button
x = New System.Windows.Forms.Button()
If TypeOf x Is System.Windows.Forms.Control Then
    ' Código caso True
End If
```

Retorna True se instância for de classe derivada da classe especificada ou se implementar interface especificada.

Expressões e comandos - Operadores

Operações de concatenação:

```
Dim a As String = "abc"
Dim z As String = a & "def"       'abcdef
Dim w As String = a + "def"       'abcdef
```

Expressões e comandos - Operadores

Operações lógicas e bit a bit:

Operador unário **`Not`**

```
Dim x, y As Boolean
x = Not 23 > 14          'False
y = Not 23 > 67          'True
```

Expressões e comandos - Operadores

Operações lógicas e bit a bit:

Operadores binários **`And`**, **`Or`** e **`Xor`**

```
Dim a, b, c, d, e, f, g As Boolean

a = 23 > 14 And 11 > 8     'True
b = 14 > 23 And 11 > 8     'False

c = 23 > 14 Or 8 > 11      'True
d = 23 > 67 Or 8 > 11      'False

e = 23 > 67 Xor 11 > 8     'True
f = 23 > 14 Xor 11 > 8     'False
```

Expressões e comandos - Operadores

Operações lógicas e bit a bit:

Operadores com curto-circuito **`AndAlso`** e **`OrElse`**

```
Dim quantidade As Integer = 12
Dim limite As Integer = 45

If quantidade > limite And ehValido(quantidade) Then
    ' A expressão chama o procedimento ehValido
End If

If quantidade > limite AndAlso ehValido(quantidade) Then
    ' A expressão não chama o procedimento ehValido
End If
```

Expressões e comandos - Operadores

Operações lógicas e bit a bit:

```
'quantidade = 12
'limite = 45

If quantidade < limite Or ehValido(quantidade) Then
    ' A expressão chama o procedimento ehValido
End If

If quantidade < limite OrElse ehValido(quantidade) Then
    ' A expressão não chama o procedimento ehValido
End If
```

Expressões e comandos - Operadores

Operações lógicas e bit a bit:

```
Dim x As Integer
x = 3 And 5      '1
```

Compara bits em posições correspondentes e atribui valores com base na comparação.

Expressões e comandos - Operadores

Operador **`If`**:

```
Dim num = 3
Console.WriteLine( If(num >= 0, "Positivo", "Negativo") )
' Positivo

num = -1
Console.WriteLine( If(num >= 0, "Positivo", "Negativo") )
' Negativo
```

Expressões e comandos - Operadores

Sobrecarga de operadores:

```
Public Structure altura
    Private metros As Integer
    Private centimetros As Double
    Public Overloads Function ToString() As String
        Return Me.metros & " m " & Me.centimetros & " cm"
    End Function
    Public Shared Operator +(ByVal a1 As altura,
                             ByVal a2 As altura) As altura
        ' Código para calcular soma de alturas
    End Operator
End Structure
```

Expressões e comandos - Funções

Funções de conversão de tipo:

```
Dim s As Single = 173.7619
Dim d As Double = s

Dim i1 As Integer = CInt(Fix(s))              ' 173
Dim b1 As Byte = CByte(Int(d))                ' 173
Dim s1 As Short = CShort(Math.Truncate(s))    ' 173
Dim i2 As Integer = CInt(Math.Ceiling(d))     ' 174
Dim i3 As Integer = CInt(Math.Round(s))       ' 174
```

Expressões e comandos - Atribuições

Operador = funciona tanto para atribuição quanto para comparação.

Atribuições compostas:

`^=`	`+=`
`*=`	`-=`
`/=`	`<<=`
`\=`	`>>=`
`&=`	

Não há atribuição unária como `++` ou `--`.

Expressões e comandos - Comandos

Delimitadores de bloco são obrigatórios.

Caso especial em operador **`If...Then...Else`**:

```
If True Then Console.WriteLine("Teste")
' OK

If True Then
    Console.WriteLine("Teste")
' Não compila

If True Then
    Console.WriteLine("Teste")
End If
' OK
```

Expressões e comandos - Comandos

Comando **GoTo**:

```
    Sub GoToDemo()
        Dim num As Integer = 1
        Dim str As String
        If num = 1 Then GoTo Linha1 Else GoTo Linha2
Linha1:
            str = "Igual a 1"
            GoTo UltimaLinha
Linha2:
            str = "Igual a 2"
UltimaLinha:
            Debug.WriteLine(str)
    End Sub
```

Expressões e comandos - Comandos

Comando **`For Each...Next`**:

```
Dim lista As New List(Of String) _
    From {"abc", "def", "ghi"}
For Each item As String In lista
    Debug.Write(item & " ")
Next
Debug.WriteLine("")
```

```
Output:  abc def ghi
```

Expressões e comandos - Comandos

Comandos **Exit** e **Continue**:

```
Dim indice As Integer = 0
Do While indice <= 100
    If indice > 10 Then
        Exit Do
    End If

    Debug.Write(indice.ToString & " ")
    indice += 1
Loop
```

```
Output:  0 1 2 3 4 5 6 7 8 9 10
```

Modularização - Módulos

Instrução **`Module`** apresenta definições de:

- variáveis;
- propriedades;
- eventos;
- procedimentos.

Modularização - Módulos

Propriedades:

```
Dim primeiroNome, ultimoNome As String
Property nome() As String
    Get
        If ultimoNome = "" Then
            Return primeiroNome
        Else
            Return primeiroNome & " " & ultimoNome
        End If
    End Get
```

Modularização - Módulos

Propriedades:

```
    Set(ByVal Valor As String)
        Dim espaco As Integer = Valor.IndexOf(" ")
        If espaco < 0 Then
            primeiroNome = Valor
            ultimoNome = ""
        Else
            primeiroNome = Valor.Substring(0, espaco)
            ultimoNome = Valor.Substring(espaco + 1)
        End If
    End Set
End Property
```

Modularização - Módulos

Eventos:

```
Event UmEvento(ByVal Valor As Integer)

RaiseEvent UmEvento(Valor)

Dim WithEvents EClass As New EventClass
Sub EClass_EventHandler() Handles EClass.XEvent
    Console.WriteLine("Evento recebido.")
End Sub

AddHandler EClass.XEvent, AddressOf EClass_EventHandler
```

Modularização - Módulos

Procedimentos:

```
Sub Main()
    Dim nome As String = Console.ReadLine()
    Console.WriteLine("Olá, " & nome)
End Sub
```

Modularização - Abstração

Procedimentos vs. Funções:

```
Sub LerNome()
    Dim nome As String = Console.ReadLine()
    Console.WriteLine("Olá, " & nome)
End Sub

Function ObterNome() As String
    Dim nome As String = Console.ReadLine()
    return nome
End Function
```

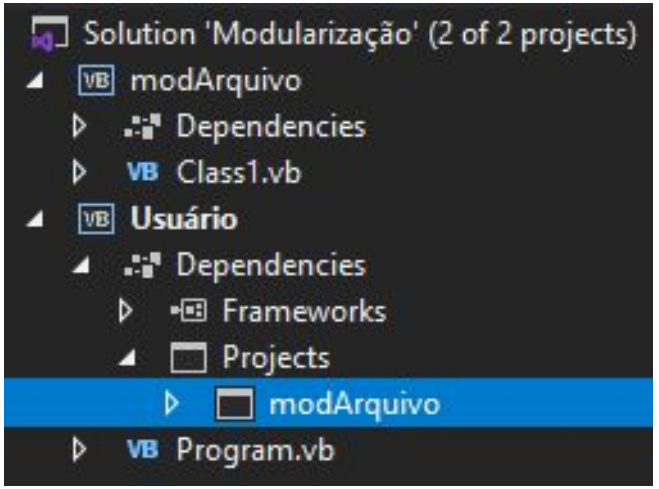
Modularização - Abstração

Instrução **`Imports`** permite referenciar nomes de tipos sem especificar namespace.

```
Namespace modNamespace
    Public Module modModule1
        Sub modProcedure()
            Console.WriteLine("Teste 1")
        End Sub
    End Module
    Public Module modModule2
        Sub modProcedure()
            Console.WriteLine("Teste 2")
        End Sub
    End Module
End Namespace
```

Modularização - Abstração

```
Imports modArquivo.modNamespace

Module Program
    Sub Main()
        modModule1.modProcedure()
        modModule2.modProcedure()
    End Sub
End Module
```



Modularização - Parâmetros

Passagem de argumento por referência (bidirecional do objeto ou unidirecional da referência):

- palavra-chave **`ByRef`**

```
Sub Trim(ByRef str As String)
    ' Modifica string
End Sub
```

Passagem de argumento por valor (unidirecional):

- palavra-chave **`ByVal`**

```
Function IsUpper(ByVal str As String) As Boolean
    ' Verifica string e retorna boolean
End Function
```

Modularização - Parâmetros

Correspondência entre parâmetros reais e formais

```
Sub Main()
    P1("abc", 123)
    P1(arg1:="abc", arg2:=123)
    P1(arg2:=123, arg1:="abc")
End Sub
Sub P1(arg1 As String, arg2 As Integer)
    Console.WriteLine(arg1 & arg2)
End Sub
```

```
Output:  abc123
         abc123
         abc123
```

Modularização - Parâmetros

Parâmetros opcionais (últimos da lista)

```
Sub Main()
    P1("abc")
    P1("abc", 321)
    P1(arg1:="abc", arg2:=321)
    P1(arg2:=321, arg1:="abc")
End Sub
Sub P1(arg1 As String, Optional arg2 As Integer = 123)
    Console.WriteLine(arg1 & arg2)
End Sub
```

```
Output:  abc123
         abc321
         abc321
         abc321
```

Modularização - Parâmetros

Lista de parâmetros variáveis

```
Sub Main()
    P1("abc", "def", "ghi")
End Sub

Sub P1(ParamArray args() As String)
    For Each arg In args
        Console.Write(arg & " ")
    Next
End Sub
```

```
Output:  abc def ghi
```

Modularização - Parâmetros

Momento da passagem é normal por padrão.

Avaliação pode ser modificada para preguiçosa com a classe **`Lazy<T>`** ou **`LazyInitializer`**.

Modularização - Dados

Orientação a objeto

```
Class Ponto
    Public x As Single
    Public y As Single
    Public Sub New(x As Single, y As Single)
        Me.x = x
        Me.y = y
    End Sub
    Public Overrides Function ToString() As String
        Return $"({x.ToString("n1")}, {y})"
    End Function
End Class
Sub Main()
    Dim p As Ponto = New Ponto(1.0, 3.2)
    Console.WriteLine(p)
End Sub
```

```
Output:  (1.0, 3.2)
```

Modularização - Dados

Classes e métodos abstratos e interfaces

```
Public MustInherit Class forma
    Public valor As Double
    Public MustOverride Function area() As Double
End Class

Public Class circulo : Inherits forma
    Public Overrides Function area() As Double
        Return Math.PI * (valor ^ 2)
    End Function
End Class
```

Modularização - Dados

Classes e métodos abstratos e interfaces

```
Interface Interface1
    Sub sub1(ByVal i As Integer)
End Interface

Public Class Classe1
    Implements Interface1
    Sub Sub1(ByVal i As Integer) Implements Interface1.sub1
        ' Código de implementação.
    End Sub
End Class
```

Polimorfismo - Verificação de tipos

Configuração **`Option Strict`** ativa/desativa checagem de tipos mais rigorosa.

- Restringe conversões implícitas;
- Não permite amarração tardia;
- Impede tipagem implícita para **`Object`**.

```
Dim x As Long = 5
Dim y As Integer = x        ' Erro de compilação

Dim x                       ' Erro de compilação
```

Polimorfismo - Verificação de tipos

Configuração **`Option Infer`** ativa/desativa inferência de tipos

```
Dim x As Long = 5
Dim y = x                    ' On:   OK
                             ' Off:  Erro de compilação
```

Polimorfismo - Ad Hoc

Coerção

```
Sub P1(x As Double)
    Console.WriteLine(x.GetType)
End Sub

Sub Main()
    Dim x As Long = 1000
    P1(x)
End Sub
```

```
Output:  System.Double
```

Polimorfismo - Ad Hoc

Sobrecarga

```
Sub Proc(x As Double)
    Console.WriteLine(x.GetType)
End Sub

Sub Proc(x As Long)
    Console.WriteLine(x.GetType)
End Sub

Sub Main()
    Dim x As Long = 1000
    Proc(x)
End Sub
```

```
Output:  System.Int64
```

Polimorfismo - Universal

Paramétrico

```
Public Class classeGenerica(Of t)
    Public Sub procedimento(ByVal item As t)
        Dim var As t
        ' Código
    End Sub
End Class

Public classe As New classeGenerica(Of Integer)
```

Polimorfismo - Universal

Inclusão

```
Class ClasseBase
    Public Overridable Function CalcularFrete(
        ByVal Dist As Double, ByVal Taxa As Double) As Double
            Return Dist * Taxa
    End Function
End Class

Class ClasseDerivada
    Inherits ClasseBase
    Public Overrides Function CalcularFrete(
        ByVal Dist As Double, ByVal Taxa As Double) As Double
            Return MyBase.CalcularFrete(Dist, Taxa) * 2
    End Function
End Class
```

Polimorfismo - Ampliação

```
Sub Main()
    Dim classe As New ClasseDerivada
    P1(classe)
    P2(classe)
End Sub

Sub P1(arg As ClasseDerivada)
    Console.WriteLine(arg.CalcularFrete(1.0, 1.0))
    Console.WriteLine(arg.GetType())
End Sub

Sub P2(arg As ClasseBase)
    Console.WriteLine(arg.CalcularFrete(1.0, 1.0))
    Console.WriteLine(arg.GetType())
End Sub
```

```
Output:    2
           Abstraçao.Program+ClasseDerivada
           2
           Abstraçao.Program+ClasseDerivada
```

Exceções

No .NET, uma exceção é um objeto herdado da classe **System.Exception**. Uma exceção é lançada de uma área do código em que ocorreu um problema. A exceção é passada pilha acima até que o aplicativo trate dela ou o programa seja encerrado.

Exceções

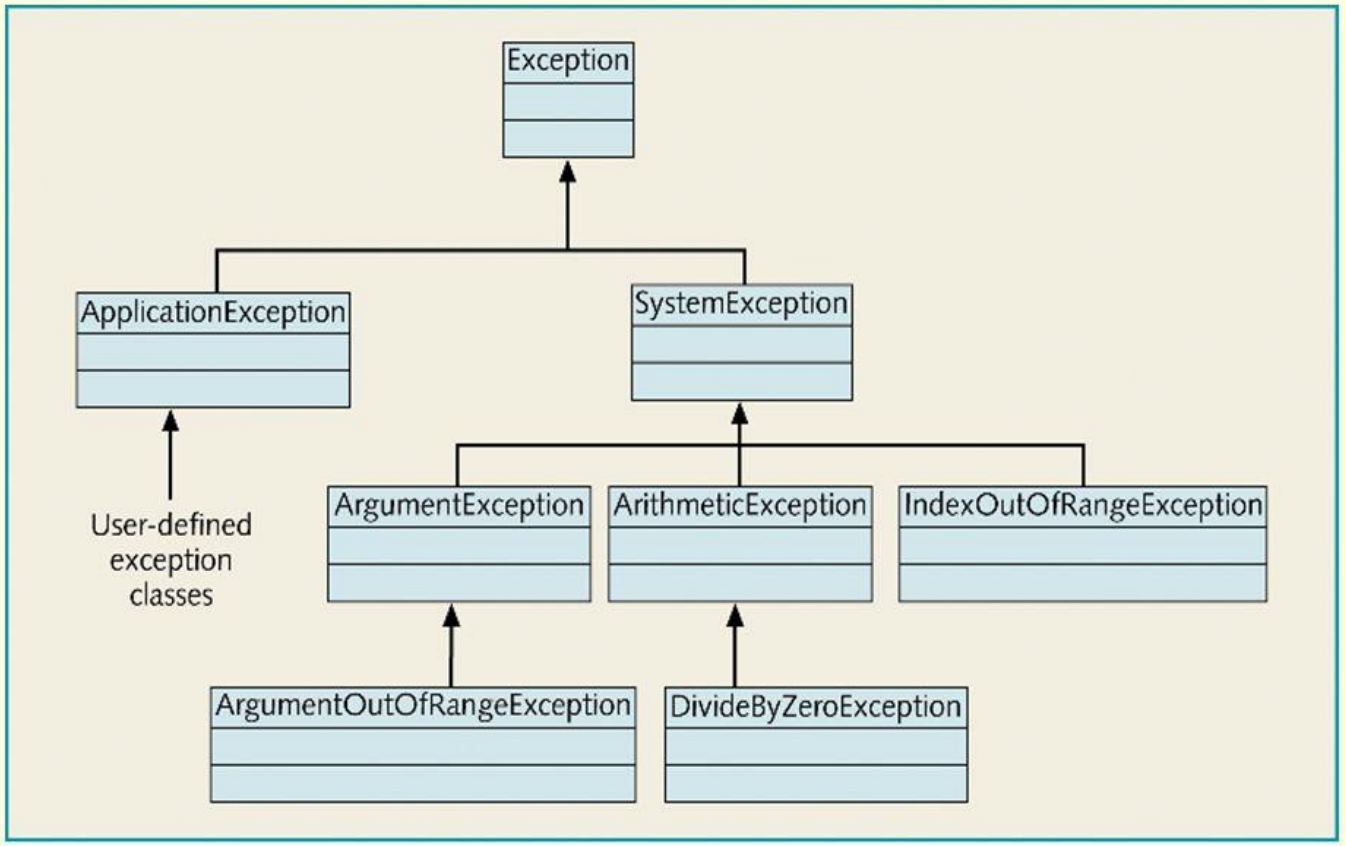


Figure 12-3 Exception class hierarchy

Exceções

A classe **System.ApplicationException** oferece suporte a exceções geradas por programas de aplicativos. Portanto, as exceções definidas pelos programadores devem derivar dessa classe.

A classe **System.SystemException** é a classe base para todas as exceções predefinidas do sistema.

Exceções

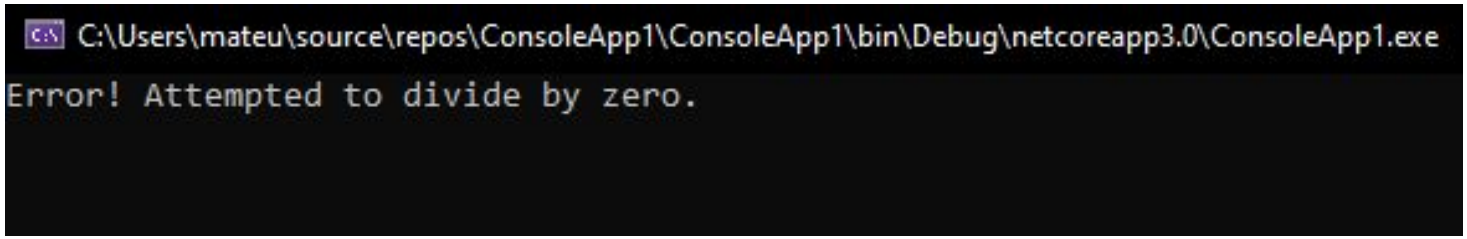
```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

Exceções

```
Sub Main(args As String())
    Try
        Dim x As Integer = 2
        Dim y As Integer = 0
        Dim z As Integer = 0
        z = x / y
    Catch ex As DivideByZeroException
        Console.WriteLine("Error! {0}" ex.Message)
    Finally
        Console.ReadLine()
    End Try
    End Sub
```

Exceções

Resultado:



Exceções

Criando suas próprias exceções:

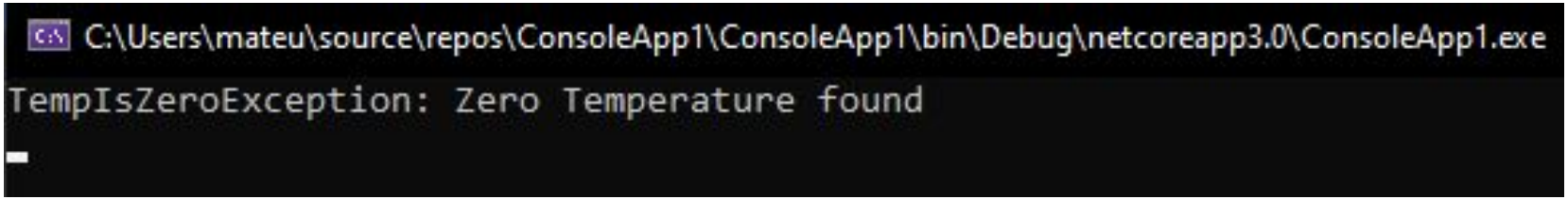
```
Public Class TempIsZeroException : Inherits ApplicationException
    Public Sub New
        MyBase.New("Zero Temperature found")
    End Sub
    End Class

Public Class Temperature
    Dim temperature As Integer = 0
    Sub showTemp()
        If (temperature = 0) Then
            Throw (New TempIsZeroException)
        Else
            Console.WriteLine("Temperature: {0}", temperature)
        End If
    End Sub
    End Class
```

Exceções

Criando suas próprias exceções:

```
Sub Main()
    Dim temp As Temperature = New Temperature()
    Try
        temp.showTemp()
    Catch e As TempIsZeroException
        Console.WriteLine("TempIsZeroException: {0}", e.Message)
    End Try
    Console.ReadKey()
    End Sub
```

Resultado:



Exceções

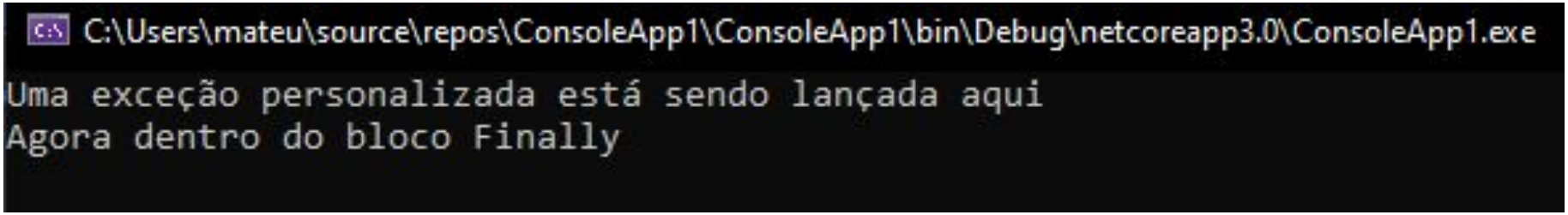
Lançando objetos:

- O Visual Basic permite ao programador lançar um objeto desde que ele seja derivado direta ou indiretamente da classe **System.Exception**
- Você pode usar a instrução **Throw** para interceptar erros em seu código porque Visual Basic move para cima a pilha de chamadas até encontrar o código de tratamento de exceção apropriado.

Exceções

```
Sub Main()
        Try
                Throw New ApplicationException("Uma exceção personalizada está
sendo lançada aqui")
        Catch e As Exception
                Console.WriteLine(e.Message)
        Finally
                Console.WriteLine("Agora dentro do bloco Finally")
        End Try
        Console.ReadKey()
        End Sub
```

Resultado:



Exceções

Instrução On Error:

- Mantida apenas para garantir a compatibilidade com projetos desenvolvidos em Visual Basic 6
- Sintaxe:

On Error { GoTo [line | 0 | -1] | Resume Next }

  - line: Número da linha
  - 0: Desabilita o manipulador de erro habilitado no procedimento atual e o redefine como Nothing;
  - -1: Desabilita a exceção habilitada no procedimento atual e a redefine como Nothing.
  - Resume Next: Especifica que quando ocorrer um erro em tempo de execução, o controle vai para a instrução imediatamente após a instrução em que ocorreu o erro e a execução continua desse ponto.

Concorrência

- Possui Threads como o C#

Concorrência

Programação assíncrona com Async e Await:

- O Visual Studio 2012 apresenta uma abordagem simplificada, programação assíncrona, que aproveita o suporte assíncrono no .NET Framework 4.5 e superior, bem como no Windows Runtime.

Concorrência

```
' Three things to note about writing an Async Function:
'   - The function has an Async modifier.
'   - Its return type is Task or Task(Of T). (See "Return Types" section.)
'   - As a matter of convention, its name ends in "Async".
Async Function AccessTheWebAsync() As Task(Of Integer)
    Using client As New HttpClient()
        ' Call and await separately.
        '   - AccessTheWebAsync can do other things while GetStringAsync is also running.
        '   - getStringTask stores the task we get from the call to GetStringAsync.
        '   - Task(Of String) means it is a task which returns a String when it is done.
        Dim getStringTask As Task(Of String) =
            client.GetStringAsync("https://docs.microsoft.com/dotnet")
        ' You can do other work here that doesn't rely on the string from GetStringAsync
        DoIndependentWork()
        ' The Await operator suspends AccessTheWebAsync.
        '   - AccessTheWebAsync does not continue until getStringTask is complete.
        '   - Meanwhile, control returns to the caller of AccessTheWebAsync.
        '   - Control resumes here when getStringTask is complete.
        '   - The Await operator then retrieves the String result from getStringTask.
        Dim urlContents As String = Await getStringTask
        ' The Return statement specifies an Integer result.
        ' A method which awaits AccessTheWebAsync receives the Length value.
        Return urlContents.Length

    End Using

End Function
```

Concorrência

As características a seguir resumem o que torna o exemplo anterior um método assíncrono:

- A assinatura do método inclui um modificador Async.
- O nome de um método assíncrono, por convenção, termina com um sufixo "Async".
- O tipo de retorno é um dos seguintes tipos:
  - Task<TResult> se o método possui uma instrução de retorno em que o operando tem o tipo TResult.
  - Task se o método não possui instrução de retorno alguma ou se ele possui uma instrução de retorno sem operando.
  - Sub se você estiver escrevendo um manipulador de eventos assíncronos.

Concorrência

- O método geralmente inclui pelo menos uma expressão await, a qual marca um ponto onde o método não pode continuar até que a operação assíncrona aguardada seja concluída. Enquanto isso, o método é suspenso e o controle retorna para o chamador do método. A próxima seção deste tópico ilustra o que acontece no ponto de suspensão.
- As palavras-chave Async e Await não fazem com que threads adicionais sejam criados. Os métodos assíncronos não exigem multithreading porque um método Async não é executado em seu próprio thread. O método é executado no contexto de sincronização atual e usa tempo no thread somente quando o método está ativo.
- A abordagem baseada em async para a programação assíncrona é preferível às abordagens existentes em quase todos os casos.

Documentação

Avaliação da linguagem

Critérios Gerais	C	C++	Java	Visual Basic
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Sim
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Estruturada, imperativa, OO e declarativa
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Avaliação da linguagem

Critérios Gerais	C	C++	Java	Visual Basic
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Sim
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Estruturada, imperativa, OO e declarativa
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Avaliação da linguagem

Critérios Gerais	C	C++	Java	Visual Basic
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	**Não**	**Não**	**Não**	**Sim**

Sintaxe simples, sem necessidade de manipulação de memória e ponteiros. Visual Studio. Fortemente tipada projetada para ser fácil de aprender

Método de Projeto	Estruturado	Estruturado e OO	OO	Estruturada, imperativa, OO e declarativa
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Avaliação da linguagem

Critérios Gerais	C	C++	Java	Visual Basic
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Sim
Eficiência	**Sim**	**Sim**	**Parcial**	**Parcial**

Assim como JAVA, VISUAL BASIC impõe que os índices de vetores sejam verificados em todos os acessos durante a execução dos programas. Isso implica necessidade de fazer um teste antes de qualquer acesso aos vetores

Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Avaliação da linguagem

Critérios Gerais	C	C++	Java	Visual Basic
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Sim
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	**Não**	**Não**	**Sim**	**Sim**

Assim como Java, é executada por modo híbrido compilada e depois interpretada pela máquina virtual Java, isso possibilita portabilidade para execução em SO

Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Avaliação da linguagem

Critérios Gerais	C	C++	Java	Visual Basic
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Sim
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	**Estruturado**	**Estruturado e OO**	**OO**	**Estruturada, imperativa, OO e declarativa**
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Avaliação da linguagem

Critérios Gerais	C	C++	Java	Visual Basic
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Sim
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Estruturada, imperativa, OO e declarativa
Evolutibilidade	**Não**	**Parcial**	**Sim**	**Sim**

Orientação a Objetos facilita a evolutibilidade. Facilidade de gerar documentação o código com XML (parecido com o JavaDoc)

Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

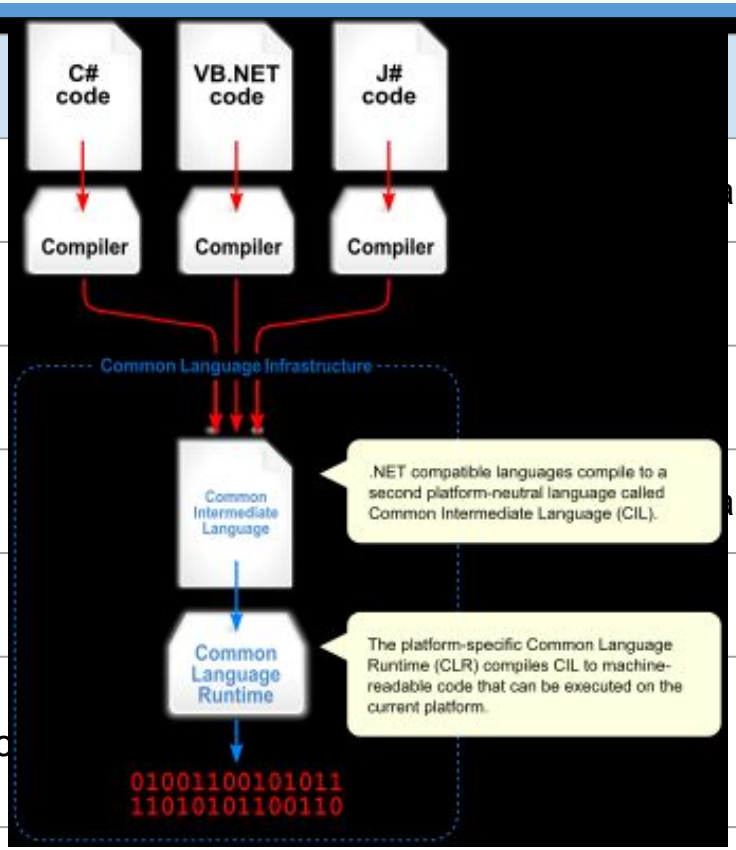
Avaliação da linguagem

Critérios Gerais	C	C++	Java	Visual Basic
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Sim
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Estruturada, imperativa, OO e declarativa
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	**Parcial**	**Sim**	**Sim**	**Sim**

Por exemplo, em Visual Basic, permite ao programador modularizar o seu código por módulos ou classes e ainda permite programação de dll e criação de componentes.

Avaliação da linguagem

Critérios Gerais	C			Visual Basic
Aplicabilidade	Sim			Parcial
Confiabilidade	Não			Sim
Aprendizado	Não			Sim
Eficiência	Sim			Parcial
Portabilidade	Não			Sim
Método de Projeto	Estruturado			Estruturada, imperativa, OO e declarativa



Integração através do .NET. Possui interoperabilidade entre linguagens. Ex: C#
CLI (Common Language Infrastructure): A especificação define um ambiente que permite a utilização de múltiplas linguagens de alto nível em diferentes plataformas sem a necessidade de serem reescritas para uma arquitetura específica.

Integração	**Sim**	**Sim**	**Parcial**	**Parcial**
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Avaliação da linguagem

Critérios Gerais	C	C++	Java	Visual Basic
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Sim
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Estruturada, imperativa, OO e declarativa
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	**Depende da ferramenta**	**Depende da ferramenta**	**Depende da ferramenta**	**Depende da ferramenta**

Avaliação da linguagem

Critérios Específicos	C	C++	Java	Visual Basic
Escopo	**Sim**	**Sim**	**Sim**	**Sim**
Expressões e Comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência dos dados	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização	LINQ, Entity Framework, Serialização, NHibernate
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e pro referência	Lista variável, por valor e por cópia de referência	Varargs, default, por nome e por posição

Avaliação da linguagem

Critérios Específicos	C	C++	Java	Visual Basic
Escopo	Sim	Sim	Sim	Sim
Expressões e Comandos	**Sim**	**Sim**	**Sim**	**Sim**
	Todas oferecem uma ampla variedade de expressões e comandos.			
Persistência dos dados	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização	LINQ, Entity Framework, Serialização, NHibernate
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e pro referência	Lista variável, por valor e por cópia de referência	Varargs, default, por nome e por posição

Avaliação da linguagem

Critérios Específicos	C	C++	Java	Visual Basic
Escopo	Sim	Sim	Sim	Sim
Expressões e Comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	**Sim**	**Sim**	**Sim**	**Sim**

Todas oferecem ampla variedade de tipos primitivos e compostos. Visual Basic possui 11 tipos primitivos no total.

Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e pro referência	Lista variável, por valor e por cópia de referência	Varargs, default, por nome e por posição

Avaliação da linguagem

Critérios Específicos	C	C++	Java	Visual Basic
Escopo	Sim	Sim	Sim	Sim
Expressões e Comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	**Programador**	**Programador**	**Sistema**	**Sistema**

Utiliza o coletor de lixo do .NET. Toda vez que você cria um novo objeto, o Common Language Runtime aloca memória para o objeto do heap gerenciado.

Avaliação da linguagem

Critérios Específicos	C	C++	Java	Visual Basic
Escopo	Sim	Sim	Sim	Sim
Expressões e Comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência dos dados	**Biblioteca de funções**	**Biblioteca de classes e funções**	**JDBC, biblioteca de classes, serialização**	**LINQ, Entity Framework, Serialização, NHibernate**
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e pro referência	Lista variável, por valor e por cópia de referência	Varargs, default, por nome e por posição

Avaliação da linguagem

Critérios Específicos	C	C++	Java	Visual Basic
Escopo	Sim	Sim	Sim	Sim
Expressões e Comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência dos dados	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização	LINQ, Entity Framework, Serialização, NHibernate
Passagem de parâmetros	**Lista variável e por valor**	**Lista variável, default, por valor e pro referência**	**Lista variável, por valor e por cópia de referência**	**Varargs, default, por nome e por posição**

Linguagens de Programação

Visual Basic

Fim

Kaique
Leonardo
Mateus