**C#**

Características:

**C#** é uma linguagem fortemente tipada, compilada e gerenciada, ou seja, possuí tipos de dados específicos, seu código sofre o processo de compilação para linguagem binária para ser interpretado pela máquina. CLR (Command Language Runtime) é quem gerencia o código ao executar e faz o serviço de consumir os recursos da máquina, como por exemplo segurança e alocação de memória.

Compilação e Gerenciamento:

No processo de compilação de qualquer linguagem do ecossistema .Net, o código é transformando primeiramente no IL (Intermediate Language), feito isso o JIT (Just In Time) tranforma o IL em código binário para finalizar.

IL:



Frameworks:

Framewrok é uma estrutura, um alicerce. Geralmente composto por um conjunto de bibliotecas, formando um ecossistema de desenvolvimento que oferece diversas funcionalidades que nos permitem escrever código de forma mais produtiva.

.NET Framework:

Framewrok mais antigo, teve sua primeira versão em meados de 2001. Atualmente está na versão 4.x e é compatível somente com windows. Hoje este framework é considerado legado.

NET Core:

Uma versão mais moderna do .NET Framework. Lançado em meados de 2015. Surgiu com o objetivo de suportar outros Sistemas Operacionais (Linux e Mac).

NET Standard:

.Net Framework e .NET Core coexistem. Podem ser instalados juntos e podem ser utilizados no mesmo projeto. Para garantir que o código vai funcionar em ambos os frameworks surge o .NET Standard, que é como uma "intersecção" dos dois frameworks. Tudo que for escrito em .NET Standard é compatível com ambos os frameworks.



NET 5:

Tem o objetivo de unificar os frameworks. o .Net Core chegou ao .NET Framework em nível de conteúdo, desta forma não faz mais sentido ter dois frameworks.

CLI:

Sigla para Command Line Interface. Alguns comandos:  
- dotnet --version

- dotnet --list-sdks

- dotnet --list-runtimes

- dotnet help

- dotnet new [Tipo\_Da\_Aplicação]

- dotnet restore --> restaura os pacotes que a aplicação precisa

- dotnet build --> compila a aplicação

- dotnet clean --> limpa as compilações anteriores

-dotnet run --> compila e executa

- dotnet run --environment=[Meu\_Ambiente] (Ex: development, staging, production)

Namespaces:

- Enquanto as pastas são divisões físicas, os Namespaces são divisões lógicas. Assim como não podemos ter dois arquivos com o mesmo nome nas pastas, não podemos ter duas classes com o mesmo nome em um namespace.

- O ideal é ter apenas um namespace e uma classe por arquivo.

- O escopo de um namespace é definido por CHAVES. Assim como CLASSES e MÉTODOS

- Um namespace pode ser reutilizado

- O ideal é que acomponhe o nome da "PASTA"

- Um namespace pode estar em diversos arquivos

Using:

- Precisamos importar o que desejamos para poder trabalhar, por padrão só o básico vem incluso

- Utilizamos a palavra reservada **using** para isto.

Variáveis:

- Uma variável é algo que utilizamos para armazenar uma informação

- Ser variváel significa que seu valor **pode** mudar a qualquer momento

- Utilize nomes coesos (Nunca utilize coisas como X ou Z)

- A primeira letra de uma variável é SEMPRE minúscula.

Constantes:

- Também serve para armazenar uma informação

- As constantes **não podem** ser alteradas

- São escritas em maiúsculas, separadas por "\_".

ex**: const int IDADE\_ADULTA = 18;**

Tipos Primitivos:

- Também são chamados de Value Types

- Armazenam o valor e não a referência para um item na memória

- São classificados em

- Tipos Simples (simple Types)

- Enumeradores(Enums)

-Estruturas(Structs)

-Tipos Nulos(Nullable Types)

Byte e SByte:

- **byte (8 bit) ->** 0 a 255

- **sbyte (8 bit) ->** -128 até 127 (permite valor negativo)

Ex: byte meuByte = 127;

Números Inteiros:

- Assim como temos o "signed" nos bytes, nos números por padrão são permitidos valores negativos. Então usamos o "unsigned" para definir que o mesmo não pode receber valores negativos

- Podem ser definidos pelos tipos:

- **short (16 bit) ->** -32,768 to 32,767

**- ushort (16 bit) ->** 0 to 65.535

**- int -> (32 bit) ->** -2,147,483,648 to 2,147,483,647

**- uint -> (32 bit) ->** 0 to 4,294,967,295

**- long -> (64 bit) ->** -9,223,372,036,854,775,808 até 9,223,372,036,854,775,807

**- ulong -> (64 bit) ->**  0 to 18,446,744,073,709,551,615

Números Reais:

- Possuem assimilação negativa e positiva por padrão, dispensando o uso do signed/unsigned em seus tipos

- Podem ser definidos pelos tipos:

- **float (32 bit) ->** -3.403823e38 to 3.403823e38

- **double (64 bit) ->** -1.79769313486232e308 to 1.79769313486232e308

- **decimal (128 bit) ->** (+ ou -) 1.0 x 10e-28 to 7.9 x 10e28

Ex:

- float salario = 2.500f; (tem que usar o **f** para saber que é um float)

-decimal salarioAnual = 25.000m; (tem que usar o **m** para saber que é um float)

Boolean:

- Armazena apenas **true** ou **false**

**- bool (8 bit) -> true/false**

Char:

- Armazena apenas UM caractere no formato Unicode.

- A atribuição de um valor char é dada por aspas simples

- **char (16 bit) ->** Qualquer caractere

Ex: char letra = 'C';

String:

- Armazena uma cadeia de caracteres.

- A atribuição de um valor string é dada por aspas duplas

Var:

- Substitui o nome de um tipo.

- Será do tipo do valor atribuído.

- É recomendado o seu uso.

Object:

- Funciona similar ao VAR, pois também vai assumir qualquer valor setado.

- É bom evitar usar

- Não possui Intelisense.

Nullable Types:

- Significa vazio,nada

- Todo tipo primitivo ou complexo pode receber o valor null

ex: **int? idade = null;**

Alias:

- Como o C# é Case Sensitive, tanto faz escrevermos um ou outro

- É recomendado utilizar sempre o alias

- ex:

**int idade = 25;** // Alias

I**nt32 idade = 25;** // Tipo

Valores Padrões:

- Todo tipo built-in, já possui um valor padrão

- Se nenhum valor for informado, seu valor padrão é utilizado

**- int --> 0**

**- float --> 0**

**- decimal --> 0**

**- bool --> false**

**- char --> '\0'**

**- string --> ""**

Conversão Implícita:

- É comum precisarmos converter um dado do tipo x para o tipo y

- Existem dois tipos de conversão, as implícitas e as explícitas

- As implícitas são:

- Podem ser executadas apenas com passagem de dados

- Possuem tipos compatíveis

ex:

**float valor = 25.8F;**

**int outro = 25;**

**valor = outro;** // Conversão implícita (o contrário não funcionaria)

Conversão Explícita:

- As conversões explícitas são:

- Ocorrem quando os tipos não são compatíveis

- É dada pelo uso do tipo entre parênteses antes da atribuição

- Segue as mesmas regras anteriores

ex:

**int inteiro = 100;**

**uint inteiroSemSinal = (uint)inteiro;** // Conversão explícita

Parse:

- Método presente em todo tipo primitivo no C#.

- Usado para converter um caractere ou string para o tipo em questão.

- Caso haja uma incompatibilidade, gera um erro.

ex:

**int inteiro = int.Parse("100");**

Convert:

- Similar ao **Parse,** porém permite converter vários tipos de valor e não apenas string.

- É uma classe com métodos estáticos.

ex:

**int inteiro = Convert.ToInt32("100");**

Operadores Aritméticos:

- Soma ----> +

- Subtração ----> -

- Multiplicação ----> \*

- Divisão ----> /

- Multiplicação e divisão são executadas primeiro, caso queira executar em uma ordem diferente, deve-se utilizar parênteses.

- Aceita os tipos: **short, int, float, double e decimal.**

Operadores de Atribuição:

- Utilizado para atribuir um valor.

- Podemos utilizar **=** junto a um operador aritmético.

ex: **x += 5**  é o mesmo que **x = x + 5.**

**x -= 1;**

**x \*= 10;**

**x /= 2;**

**x ++;**  é o mesmo que **x = x + 1. (increment)**

**x --;**  é o mesmo que **x = x - 1. (decrement)**

Operadores de Comparação:

- Igual ----> **==**

- Diferente ----> **!=**

- Maior que ----> **>**

- Menor que ----> **<**

- Maior ou igual a ----> **>=**

- Menor ou igual a ----> **<=**

Operadores Lógicos:

- Utilizado para operações condicionais.

- Retorna SEMPRE **true** ou **false.**

&&:

- Deve atender TODAS as condições

**||:**

- Se atender uma condição já retorna verdadeiro.

**!:**

- Negação

Operadores Condicionais:

**IF**:

- Utilizado para a tomada de decisão

**ELSE IF:**

-(Se não se)

**ELSE:**

- (Se não)

Estrutura Condicional SWITCH:

- Utilizado quando temos muitas decisões.

- Executado em cascata

- Devemos parar manualmente a execução com o comando **break.**

- Possui uma execução padrão chamada default

**ex:**

**int valor = 1;**

**switch (valor)**

**{**

**case 1: Console.WriteLine("1"); break;**

**case 2: Console.WriteLine("2"); break;**

**case 3: Console.WriteLine("3"); break;**

**case 4: Console.WriteLine("4"); break;**

**default: Console.WriteLine("6"); break;**

**}**

Laços de repetição FOR:

- Executa uma determinada ação várias vezes.

- Exige 3 parâmetros:

- Contador

- Condição

- Incremento

Laços de repetição WHILE:

- Executa enquanto UMA condição for verdadeira.

- Exige uma expressão booleana.

Laços de repetição DO WHILE:

- Executa enquanto UMA condição for verdadeira.

- Exige uma expressão booleana.

- Checa a condição DEPOIS de executar.

Funções ou Métodos:

- Podemos segmentar nosso programa em funções.

- Também conhecidos como métodos.

- O **main** é um exemplo de método.

- Possui um retorno, um nome e parâmetros.

- Esta composição é camada de assinatura.

- Começam **sempre** com maiúscula.

-----------------------------------------------------------------------------------------------------

Heap e Stack:

- A memória é dividida em duas partes, **Heap** e  **Stack**

- **Heap** armazena os dados.

- **Stack** armazena as referências para os dados.

- No .Net qualquer tipo é tradado como:

- Tipo de Referência (**Reference Type**)

- Tipo de Valor (**Value Type**)

Value Types:

- Tipos de valor armazenam dados

- São armazenados em um local da memória chamada **Stack**.

- Quando armazenamos um valor, a memória é alocada.

- Este espaço armazena o dado criado

- Nossa variável acessa este dados diretamente

- Se assimilarmos uma variável do tipo de valor a outra, o valor será **copidado** e ambas serão independentes.

- Alguns types do tipo de valor: Built-in, Structs, Enums.

- **Garbage Collector** não acessa o stack.

Ex:

**int x = 25;**

**int y = x;** // Y é uma cópia de X

**Console.WriteLine(x);**  // 25

**Console.WriteLine(y);** // 25

**x = 32;**  // Somente X foi alterado

**Console.WriteLine(x);** // 32

**Console.WriteLine(y);** // 25

Reference Types:

- Armazenam o endereço do objeto que contém os dados.

- Não armazenam os dados em si.

- São armazenados em um local da memória chamado de **Heap.**

- Ao assimilar um variável será criada uma **referência**, esta aponta para a **mesma** informação, desta forma, elas **não** são independentes.

- Quando não são mais utilizados são marcados para exclusão.

- **Garbage Collector** passa removendo todos eles,

- Alguns types do tipo de referência: Classes, Objects, Arrays...

Ex:

**var arr = new string[2];**

**arr[0] = "Item 1";**

**var arr2 = arr;**  // Não cria uma cópia

**Console.WriteLine(arr[0]);**  // "Item 1"

**Console.WriteLine(arr2[0]);** // "Item 1"

// Alterando as duas listas

**arr[0] = "Item alterado";**

**Console.WriteLine(arr[0]);**  // "Item alterado"

**Console.WriteLine(arr2[0]);** // "Item alterado"

Structs:

- Tipos de dado estruturado.

- Apenas a estrutura, o esqueleto.

- Tipo de valor.

- Armazenam apenas outros tipos de dados.

- Definido pela palavra **struct**

- Composto de **propriedades** e **métodos.**

- Nome **sempre** com maiúsculo.

- Criado a partir da palavra **new.**

Enum:

- Usado para fornecer uma melhor visualização do código.

- Substituem o uso de inteiros.

- Usados em listas curtas.

- Usados em dados fixos (**Hard Coded**).

- **Sempre** em maiúsculo.

-Começar com a letra **E**