

Notebook

**Curso C# Essencial (Com
LINQ, Net 7.0 e .NET 8.0)
Teoria**

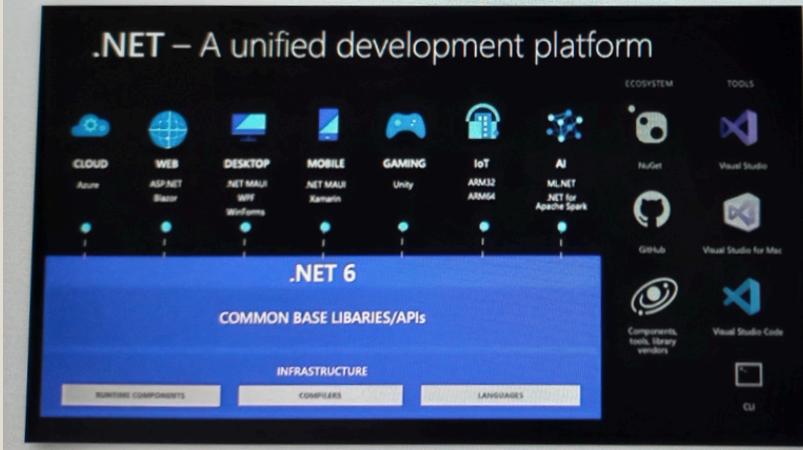
2024

22/05

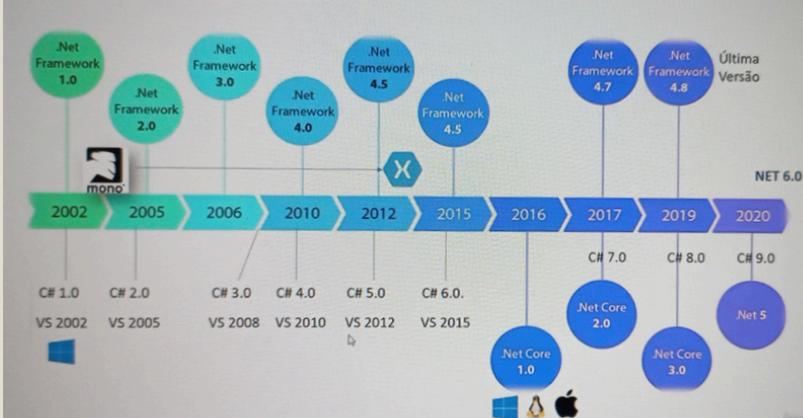
Seção 01: Introdução

01: Apresentação

A plataforma .NET ou apenas .NET



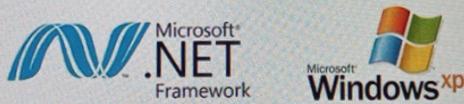
.NET - Histórico das releases



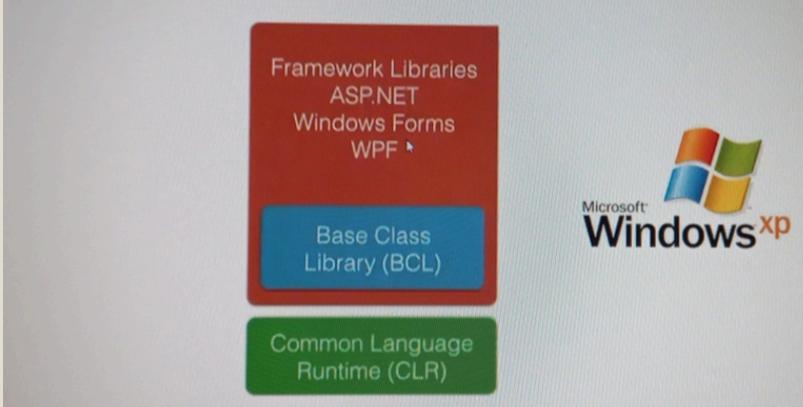
.NET Framework - 2002

O .NET Framework é um ambiente de execução da Microsoft e gerenciado para Windows que oferece uma série de serviços voltados ao desenvolvimento web e desktop reutilizando e reaproveitando códigos e possuindo componentes para criar código usando as linguagens C#, VB.NET e F#.

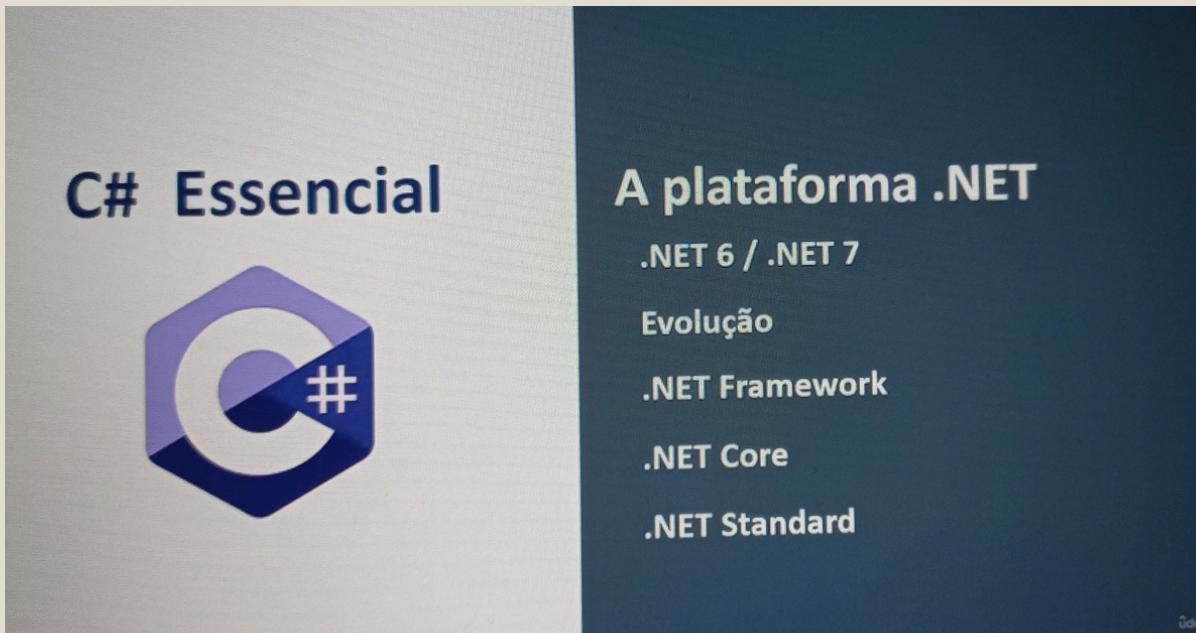
- CLR (Common Language Runtime): um mecanismo de execução que manipula aplicativos em execução (máquina virtual);
- BCL - Biblioteca de classes base : o .NET Framework oferece uma biblioteca de códigos testados e reutilizáveis que os desenvolvedores podem chamar de seus próprios aplicativos



.NET Framework – Desenvolvimento para Windows



02: Plataforma .net



.NET Core - 2016

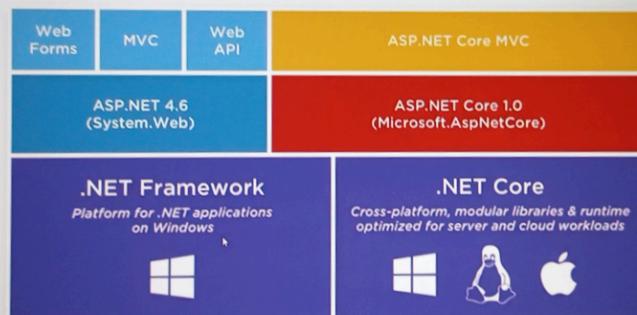
A .NET Core é uma plataforma para desenvolvimento de aplicações lançada em 2016 como um projeto de código aberto, sendo uma solução mais leve e modular que o .NET Framework e pode ser usada em diferentes sistemas operacionais como Windows, Mac e Linux.

Antes da unificação podíamos desenvolver aplicações usando a .NET Core ou o .NET Framework que é suportado apenas no Windows e cuja última versão é a versão 4.8.

Ambos os Frameworks compartilham muitos dos mesmos componentes e você podia compartilhar código entre os dois, no entanto, existiam diferenças fundamentais entre os dois e sua escolha ia depender do que você desejava realizar



.NET Framework e .NET Core

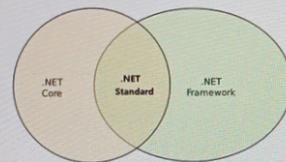


.NET Standard – O início da unificação

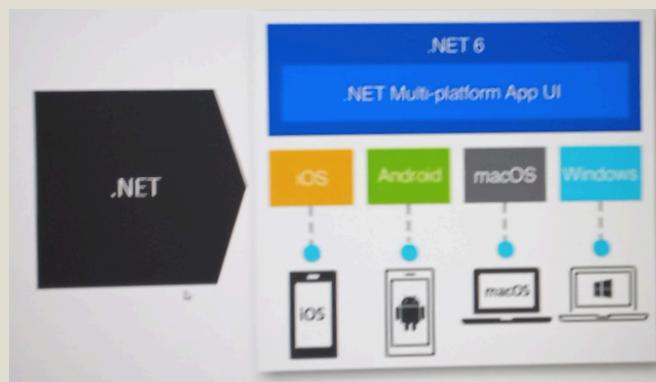
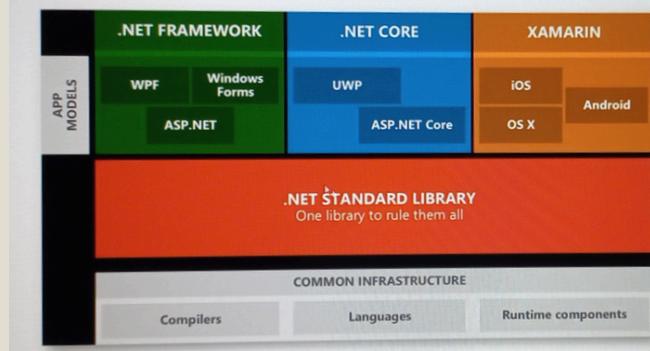
O [.NET Standard](#), atualmente na versão 2.1, surgiu para ser um meio termo entre as duas versões, ele é uma interface que define a lista de APIs que uma determinada função do .NET deve suportar.^b

Uma biblioteca escrita utilizando o .NET Standard pode ser suportada tanto por aplicações utilizando o .NET Core quanto o .NET Framework.

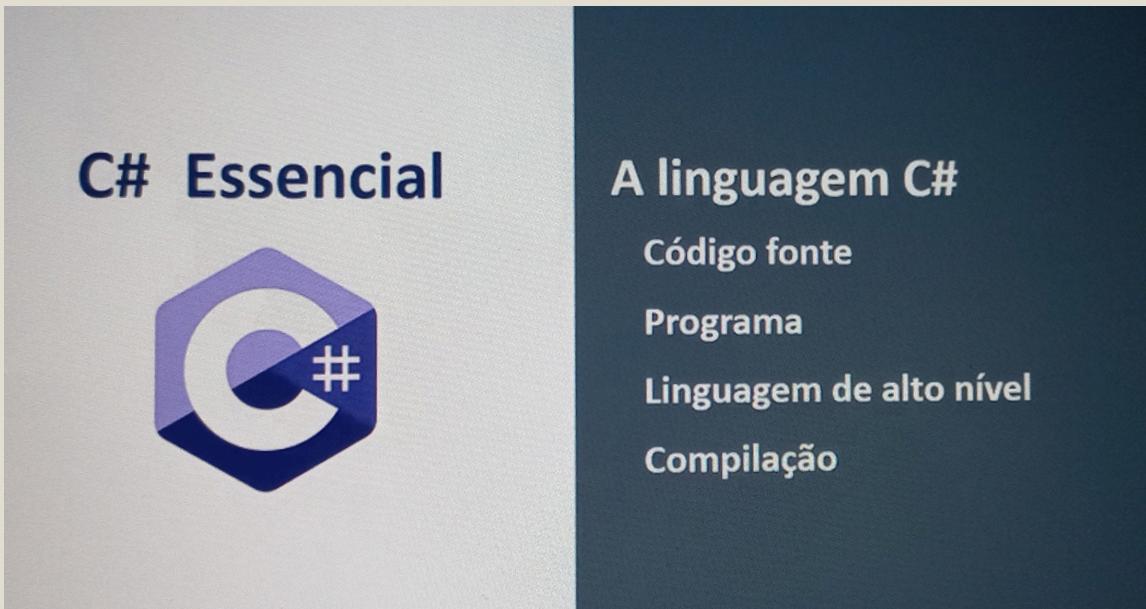
Ele foi criado para que esse compartilhamento fosse muito mais fácil e uniforme no ecossistema do .NET



.NET Standard



03: Linguagem C#



Linguagem de alto e baixo nível

```

1  using System;
2
3  Console.WriteLine("Hello World");
4
5  var dataAtual = DateTime.Now;
6  Console.WriteLine(dataAtual);
7
8  ExibirMensagem("Usando C# 9.0 - Instruções de nível superior");
9
10 //void ExibirMensagem(string mensagem)
11 //{
12 //    Console.WriteLine(mensagem);
13 //}
14
15 Console.ReadLine();

```

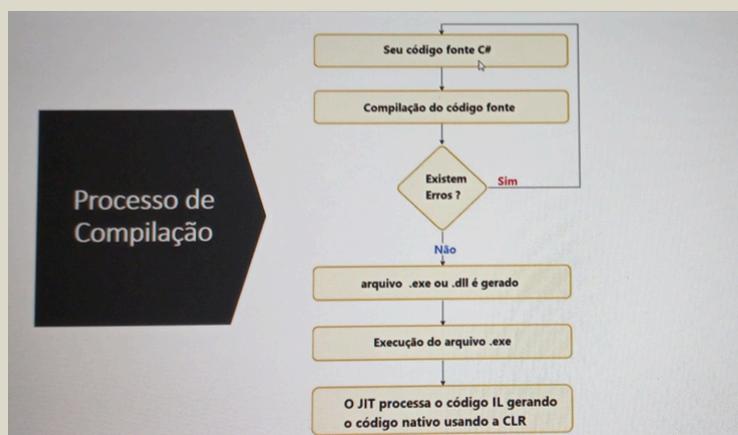
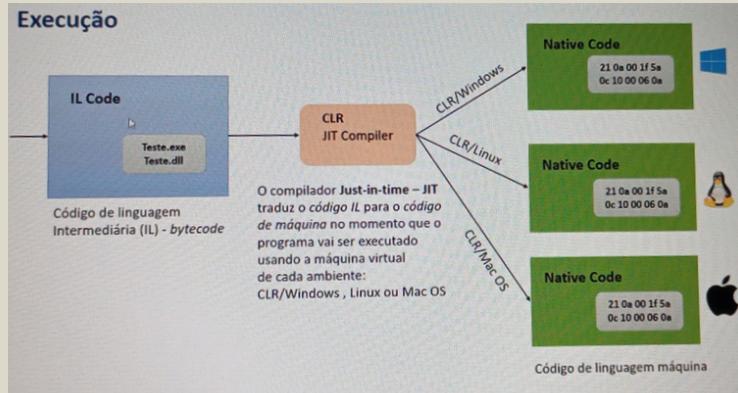
Código Fonte de um programa usando a linguagem C#
Linguagem de alto nível

```

C01A 45 0A      DECSC  INCH   RECEIVE NOT READY
C01B 84 80 05  LDA A  AC1A+1  GET CHAR
C019 84 7F      AND A  #57F   HOLE PARITY
C01B TE 00 79  JNP  OUTCH  ECBS & RTS
*****
* FUNCTION: INCH - INPUT HEX DIGIT
* INPUT:  INCH
* OUTPUT: digit in acc A
* CALLS:  INCH
* DECODES: acc A
* RETURNS to monitor if not HEX input
C01B BD 00      INCHX  BSR  INCH   GET A CHAR
C020 B1 30      CMP A  #'0  ZERO
C022 2B 11      BNE R  HEXERR  NOT HEX
C024 2B 19      CMP A  #'F  FIVE
C026 2F 0A      BLE R  HEXCTS  GOOD HEX
C028 B1 41      CMZ A  #'A  ZERO
C02A 2B 05      BNE R  HEXENDS  NOT HEX

```

Código Fonte de um programa feito em Assembly
Linguagem de baixo nível



04: C# características

C# Essencial



A linguagem C#

Características
Tipos de projetos
Palavras reservadas
Identificadores
Sintaxe básica

Tipos de projetos

Aplicações para Cloud e services
Aplicações Windows client
Bibliotecas Windows e componentes
Windows services
Aplicações Web
Web services e Web API
Aplicações mobile para iOS e Android
Backend services
Aplicações Azure cloud e services
Backend database usando ML/Data tools
Interoperabilidade com Office, SharePoint, SQL Server, etc.
Inteligência artificial e Machine learning
Blockchains e cryptocurrency
Internet das coisas (IoT) e dispositivos
Gaming consoles e gaming systems
Video games



Palavras reservadas da linguagem

- As palavras-chave são identificadores reservados predefinidos com significados especiais para o compilador.

abstract	do	in	protected	throw
as	double	int	public	true
base	else	interface	readonly	try
bool	enum	internal	ref	typeof
break	event	is	return	unit
byte	explicit	lock	sbyte	ulong
case	extern	long	sealed	unchecked
catch	false	namespace	short	unsafe
char	finally	new	sizeof	ushort
checked	fixed	null	stackalloc	using
class	float	object	static	using static
const	for	operator	string	virtual
continue	foreach	out	struct	void
decimal	goto	override	switch	volatile
default	if	params	this	while
delegate	implicit	private		

Identificadores

São nomes que atribuímos para identificar classes, propriedades, variáveis e outros recursos da linguagem C#

Regras para identificadores válidos

- Devem começar com uma letra ou sublinhado (_). Ex: valor1, _letra, taxa#20
- Não podem começar com um número. Ex: 1valor, 2letra -> **inválido**
- Não devem conter espaços em branco. Ex: taxa imposto -> **inválido**
- As palavras reservadas da não podem ser usadas como identificadores, a menos que incluam @ como prefixo. Ex: @as é um identificador válido, mas "as" não é.
- Os identificadores C# permitem caracteres Unicode.
- Os identificadores C# diferenciam maiúsculas de minúsculas. Ex: Pai é diferente de pai
- Os identificadores C# não podem conter mais de 512 caracteres.

Sintaxe Básica

// Comentário em uma linha.

/* */ Bloco de comentário.

{ } Delimitador de bloco de comandos.

;; Todos os comandos "simples" terminam com ";"

= Atribuição

== Comparação de igualdade

05: Organização do código

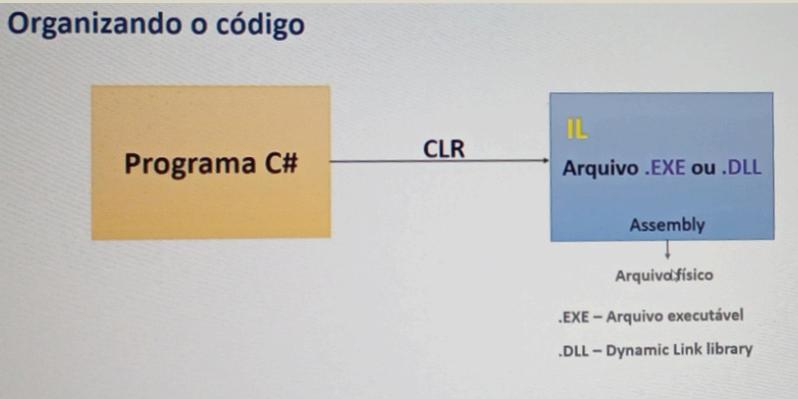
C# Essencial



Organização do Código

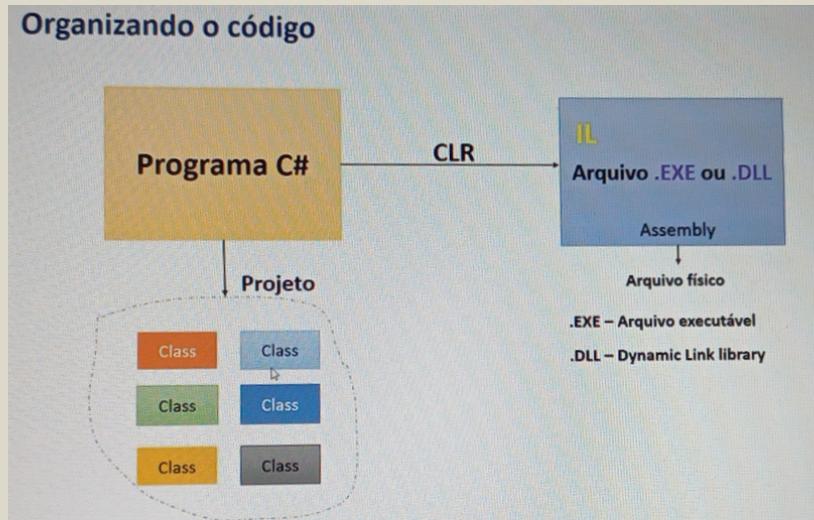
Assembly
Projeto
Classe
Namespace
Solução

Organizando o código

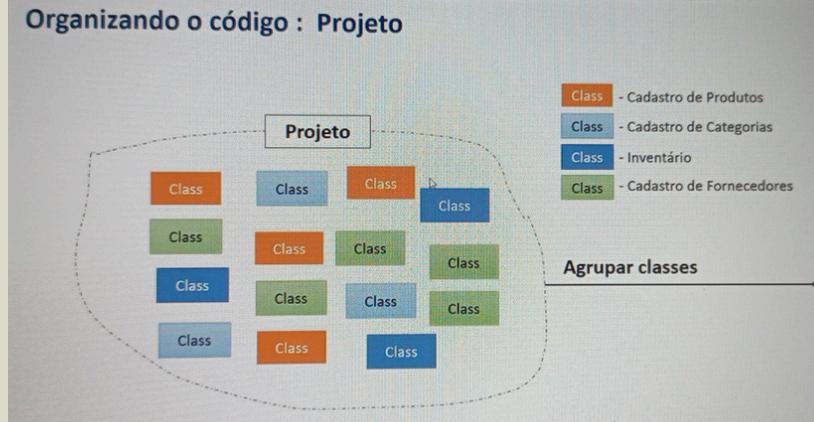


Como é
organizado o
projeto

Organizando o código



Organizando o código : Projeto



Organizando o código : namespace e diretiva using

nome_do_namespace.nome_da_classe → Fornecedores.Cadastro

Usando a diretiva using para simplificar o uso dos namespaces

using nome_do_namespace

nome_da_classe

```
using System;
using System.Collections;

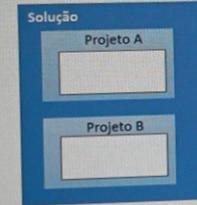
public class Fila
{
    //codigo
}
```

Organizando o código : Solução ou Solution

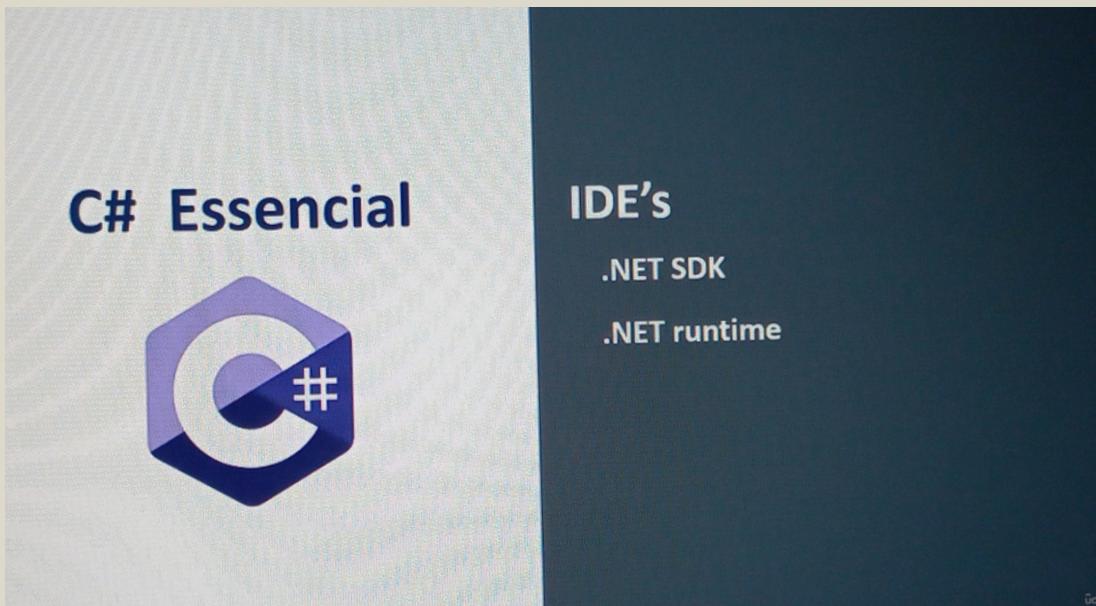
Ao criar um aplicativo no Visual Studio, você começa com um *projeto*

Uma **Solução** ou **Solution** é um *container* para os projetos criados no Visual Studio

Uma Solução pode conter um ou mais projetos

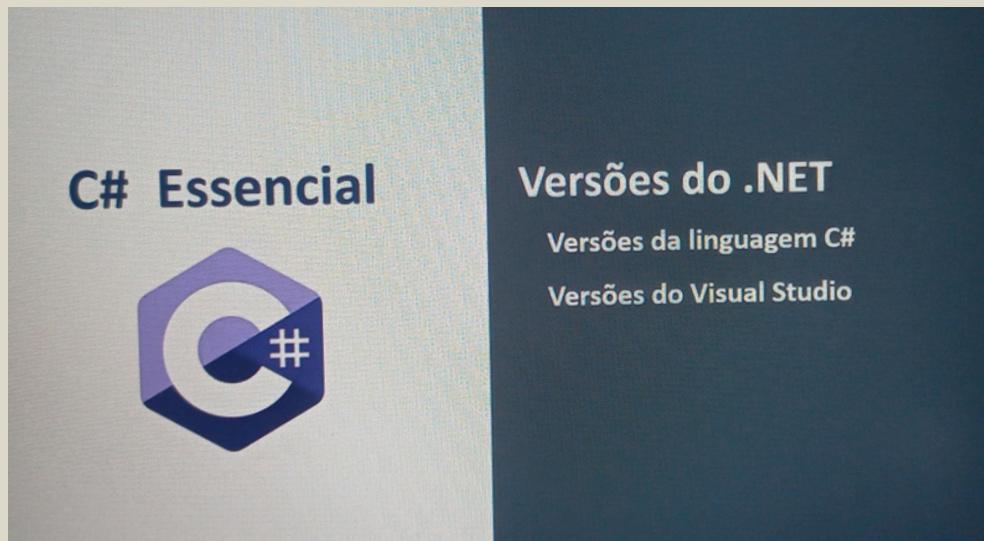


06: IDE ambiente dev



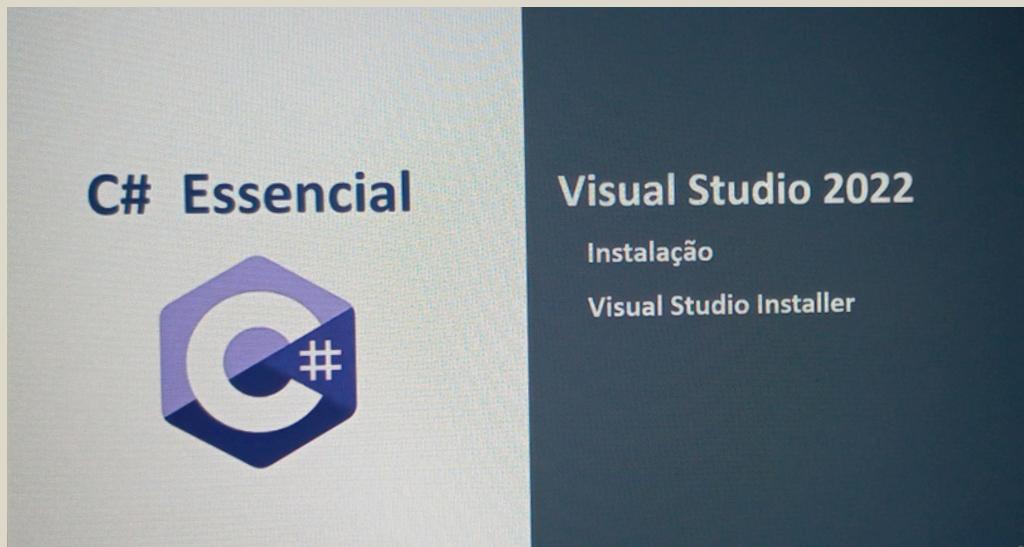
23/05

07: Versões do dotnet



23-24/05

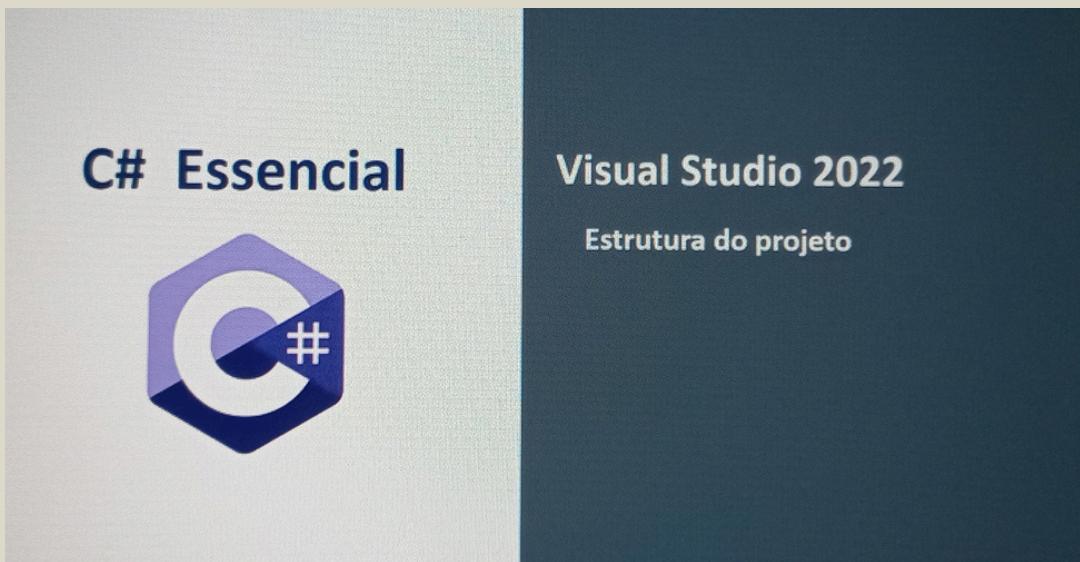
08 | 09: Instalação vs code



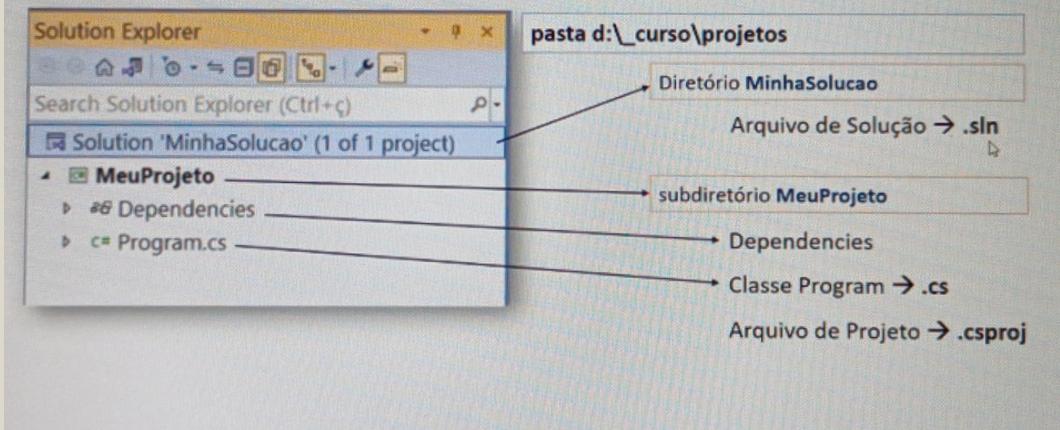
10: Template de projeto



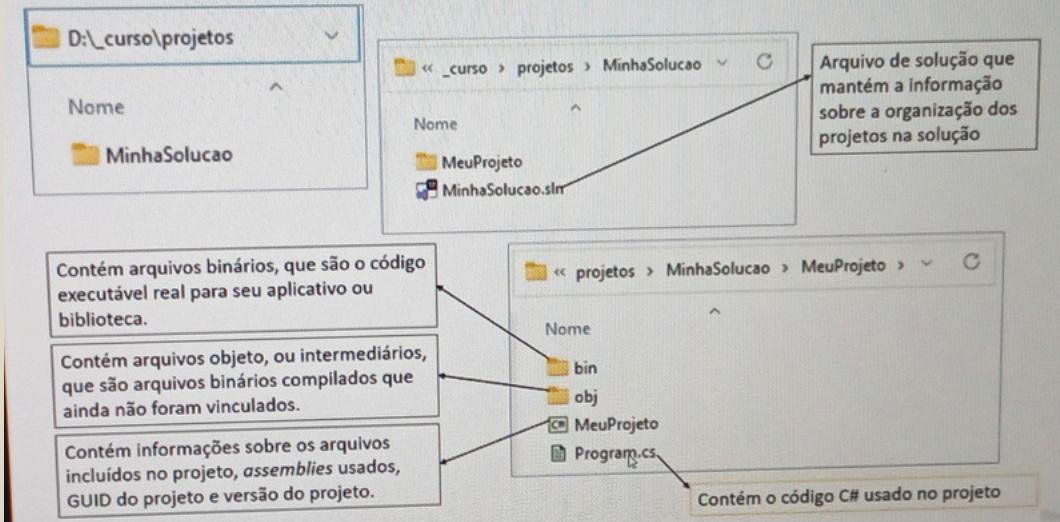
11: Estrutura do projeto



Visual Studio 2022 – estrutura do projeto



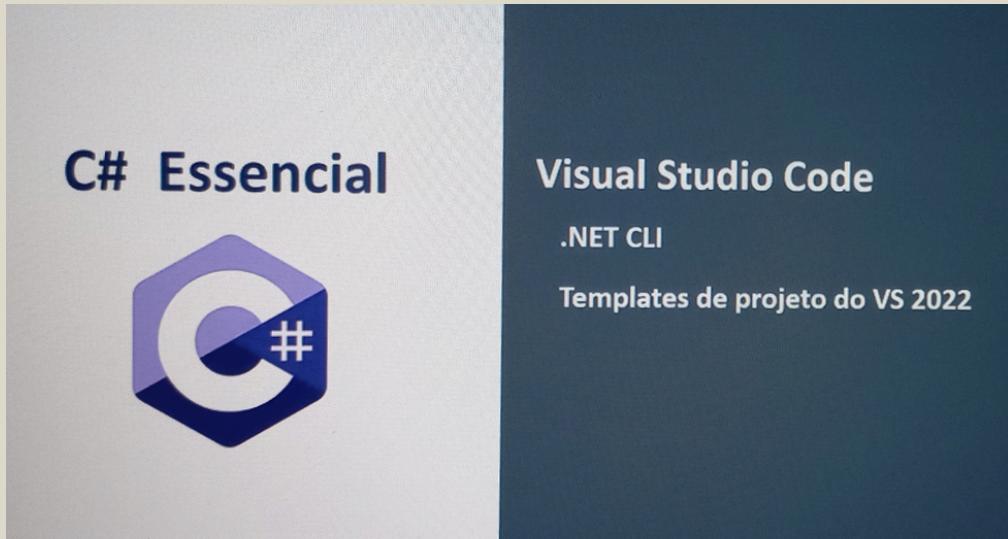
Visual Studio 2022 – estrutura do projeto



Como é estruturado a pasta do projeto
Descrição de cada parte do projeto

27/05

12: Vs code projetos



Criar projeto usando a NET CLI e os templates do Visual Studio

Usar a ferramenta de linha de comando .NET CLI para criar projetos

O NET CLI consiste no driver **dotnet** e das opções e argumentos de comandos

Exs: **dotnet --version** **dotnet new console** **dotnet run**

O driver **dotnet** executa um aplicativo ou um comando

Para emitir um comando podemos abrir um *terminal* ou *janela de prompt do DOS* ou janela do *PowerShell*

.NET CLI – Principais comandos

dotnet --info - Lista informações detalhadas do sistema
dotnet --version - Exibe a versão do .NET SDK atual
dotnet --list-sdks - Exibe a lista dos SDKs instalados
dotnet --list-runtimes - Exibe a lista dos runtimes instalados
dotnet new - Lista os principais templates de projetos;
dotnet new --list - Lista todos os templates de projetos;
dotnet new <nome_template> - Cria um projeto usando o template informado
dotnet run - Executa projetos;
dotnet restore - Restaurar os pacotes do projeto
dotnet test - Executa projetos de teste e testes de unidade;
dotnet publish - Usado para publicar projetos;
dotnet new sln - Cria uma nova solution;
dotnet sln add/remove - Adiciona projetos para uma solução;
dotnet add/remove reference - Adiciona referência de projetos para outros projetos;
dotnet add/remove package - Adiciona referência de pacotes Nuget para um projeto;

VS Code e .NET CLI - Criar projeto e solução igual ao Visual Studio

- Criar uma solução :

dotnet new sln -o <nome_solução>

- Criar um projeto dentro da solução (entrar na pasta da solução)

dotnet new <nome_template> -o <nome_projeto>

- Incluir o projeto criado na solução existente (a partir da pasta da solução)

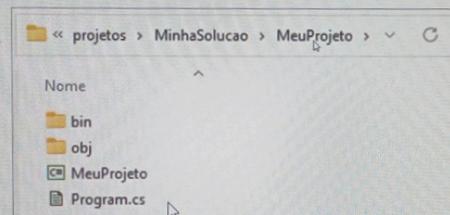
dotnet sln <nome_solução>.sln add <pasta_projeto>/<arquivo>.csproj

- Abrir o projeto no VS Code (estando na pasta da solução)

Digitar : code .

VS Code e .NET CLI

Cria a mesma estrutura do projeto criada pelo Visual Studio



13: Projeto - Configuração

C# Essencial



The C# logo icon is a purple hexagon containing a white 'C' and a white '#' symbol.

Visual Studio

Solution e Projeto

Propriedades e Configurações

Udemy

Visual Studio 2022

Open recent



▲ This week



MySolution.sln C:\Users\danie\Desktop\cursoC#\MySolution

24/05/2024 09:35

Get started



Clone a repository

Get code from an online repository like GitHub or Azure DevOps



Open a project or solution

Open a local Visual Studio project or .sln file



Open a local folder

Navigate and edit code within any folder



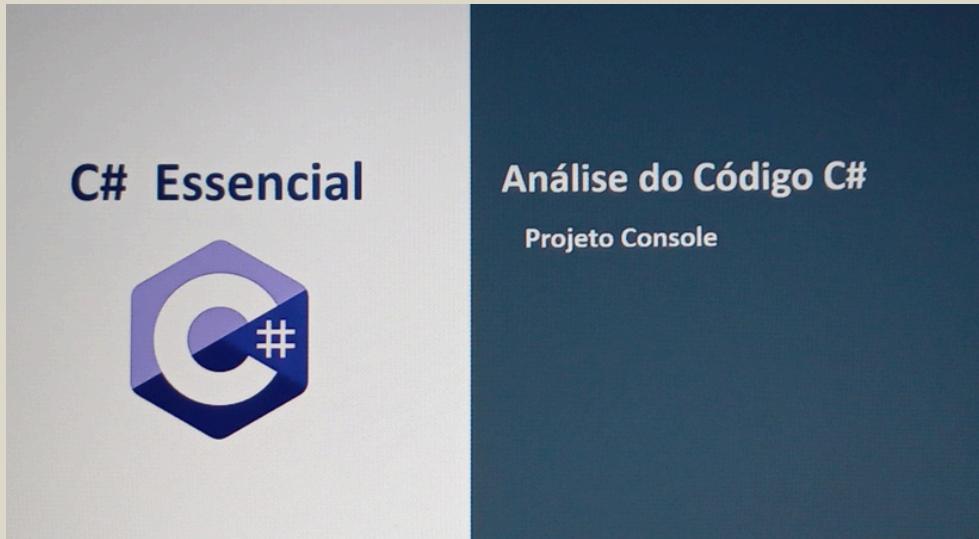
Create a new project

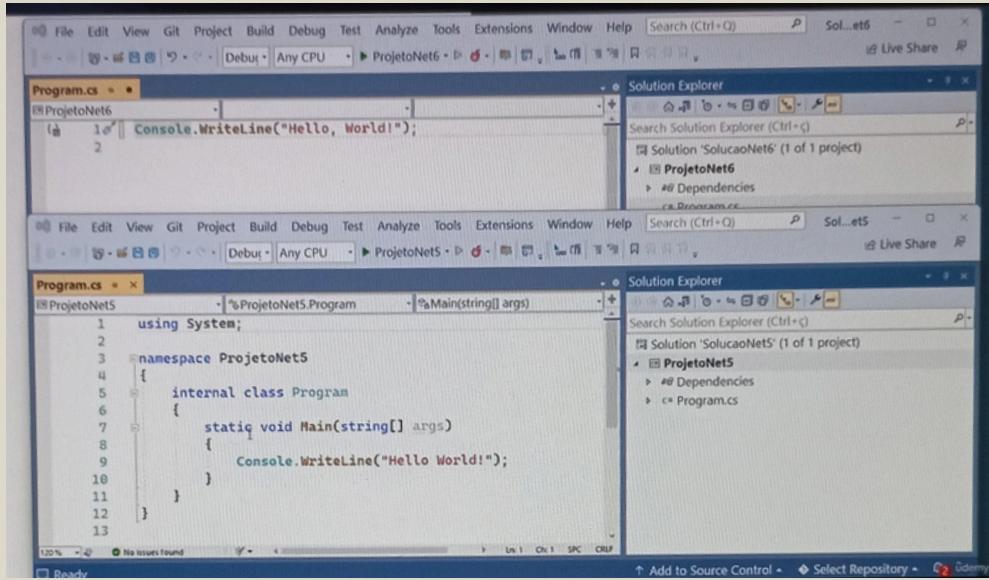
Choose a project template with code scaffolding to get started

[Continue without code →](#)

- Como criar um novo projeto
- Como pegar um projeto já criado
- Configurando
- Startando

14: Projeto console - análise de código



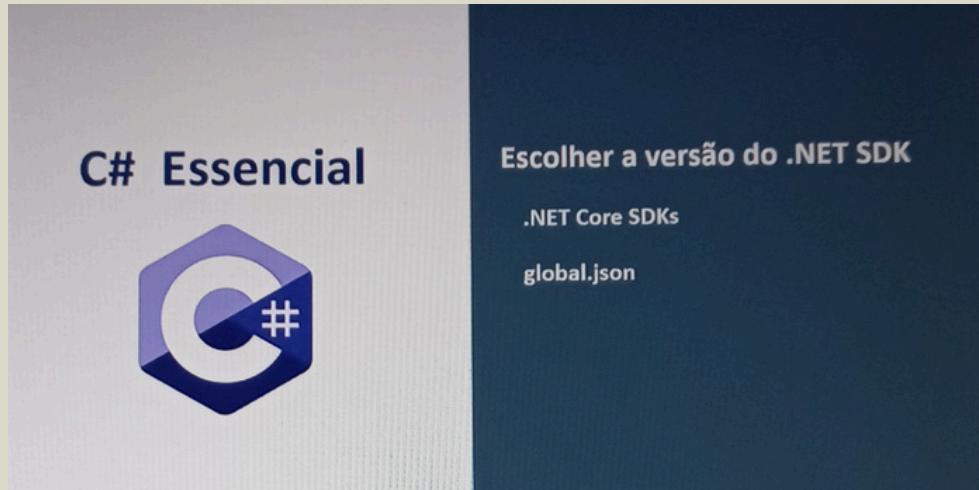


Um projeto feito com net6 e o outro com net5

Diferença do 5 para o 6

- A classe main fica exposta na versão 5
- A classe main fica oculta na versão 6

15: Como escolher a versão do SDK .net



.NET Core SDK

Cada versão do .NET SDK é instalada separada da versão anterior, e, assim, um novo SDK, não afeta o SDK instalado anteriormente.

Para inspecionar todos os .NET SDKs instalados na sua máquina acesse a pasta:
`X:\Program Files\dotnet\sdk`

As versões do .NET Runtime podem ser inspecionadas na pasta:
`X:\Program Files\dotnet\shared\Microsoft.AspNetCore.App`

- Onde fica o download do dotNet
- As versões
- Como listar
- Como criar um arq. .json

Listar SDKs , Runtimes e obter versão do .NET

Listar as versões dos SDKs instalados : `dotnet --list-sdks`

Listar as versões dos runtimes instalados : `dotnet --list-runtimes`

Para saber a versão do .NET usada : `dotnet --version`

Escolher a versão do .NET SDK

- Criar um arquivo `global.json` na raiz da pasta

```
{  
  "sdk": {  
    "version": "6.0.8"  
  }  
}
```

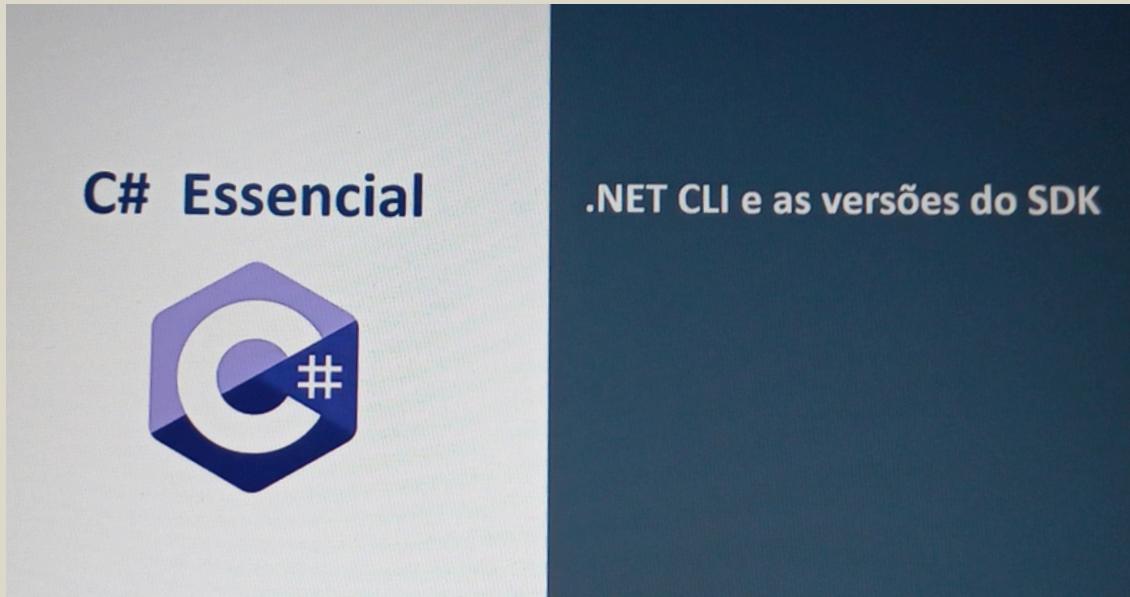
Se nenhum arquivo `global.json` for encontrado, ou o arquivo `global.json` não especificar uma versão do SDK, a versão mais recente do SDK instalada será usada.

Se `global.json` especificar uma versão do SDK:

- Se a versão do SDK especificada for encontrada na máquina, essa versão será usada;
- Se a versão do SDK especificada não puder ser encontrada na máquina, a versão mais recente do patch do SDK instalada dessa versão será usada.
- Se a versão do SDK especificada e uma versão de patch do SDK apropriada não puderem ser encontradas, será lançado um erro;

28/05

16: Net CLI - versão .net SDK



Escolher a versão do .NET SDK

- Criar um arquivo global.json

```
dotnet new globaljson --sdk-version 5.0.0 --force
```

```
D:\netversao>dir
O volume na unidade D não tem nome.
O Número de Série do Volume é 16C2-4B58

Pasta de D:\netversao

14/09/2022 07:54 <DIR> .
0 arquivos(s) 0 bytes
1 pasta(s) 3.832.892.710.912 bytes disponíveis

D:\netversao>dotnet new globaljson --sdk-version 3.1.423 --force
O modelo "arquivo global.json" foi criado com êxito.

D:\netversao>=
```

```
D:\MeuAppl>dir
O volume na unidade D não tem nome.
O Número de Série do Volume é 16C2-4B58

Pasta de D:\MeuAppl

05/04/2023 08:20 <DIR> .
05/04/2023 08:20 249 MeuAppl.csproj
05/04/2023 08:20 <DIR> obj
05/04/2023 08:20 105 Program.cs
2 arquivo(s) 354 bytes
2 pasta(s) 3.791.417.360.384 bytes disponíveis

D:\MeuAppl>type me|
```

Top Level statements

ou

Instruções de nível superior

```
D:\>dotnet --list-sdks
3.1.426 [C:\Program Files\dotnet\sdk]
5.0.408 [C:\Program Files\dotnet\sdk]
6.0.407 [C:\Program Files\dotnet\sdk]
7.0.104 [C:\Program Files\dotnet\sdk]
7.0.202 [C:\Program Files\dotnet\sdk]

D:\>dotnet --version
7.0.202

D:\>dotnet new console -o MeuAppl
O modelo "Aplicativo do Console" foi criado com êxito.

Processando ações pós-criação...
Restaurando D:\MeuAppl\MeuAppl.csproj:
Determinando os projetos a serem restaurados...
D:\MeuAppl\MeuAppl.csproj restaurado (em 61 ms).
A restauração foi bem-sucedida.

D:\>cd |
```

```
D:\MeuAppl>dotnet --list-sdks
3.1.426 [C:\Program Files\dotnet\sdk]
5.0.408 [C:\Program Files\dotnet\sdk]
6.0.407 [C:\Program Files\dotnet\sdk]
7.0.104 [C:\Program Files\dotnet\sdk]
7.0.202 [C:\Program Files\dotnet\sdk]

D:\MeuAppl>dotnet new console -o MeuApp2 -f net5.0
```

```
Pasta de D:\MeuAppl

05/04/2023 08:20 <DIR> .
05/04/2023 08:20 249 MeuAppl.csproj
05/04/2023 08:20 <DIR> obj
05/04/2023 08:20 105 Program.cs
2 arquivo(s) 354 bytes
2 pasta(s) 3.791.417.360.384 bytes disponíveis

D:\MeuAppl>type meuappl.csproj
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>net7.0</TargetFramework>
  <ImplicitUsings>enable</ImplicitUsings>
  <Nullable>enable</Nullable>
</PropertyGroup>

</Project>

D:\MeuAppl>
```

Tratando com as versões do .NET instaladas

Cria projeto especificando a versão instalada do .NET SDK

```
dotnet new console -o MeuApp1 -f .net5.0
```

```
dotnet new console -o MeuApp1 -f .net6.0
```

Cria projeto .NET 6/.NET 7 usando o método Main na classe Program

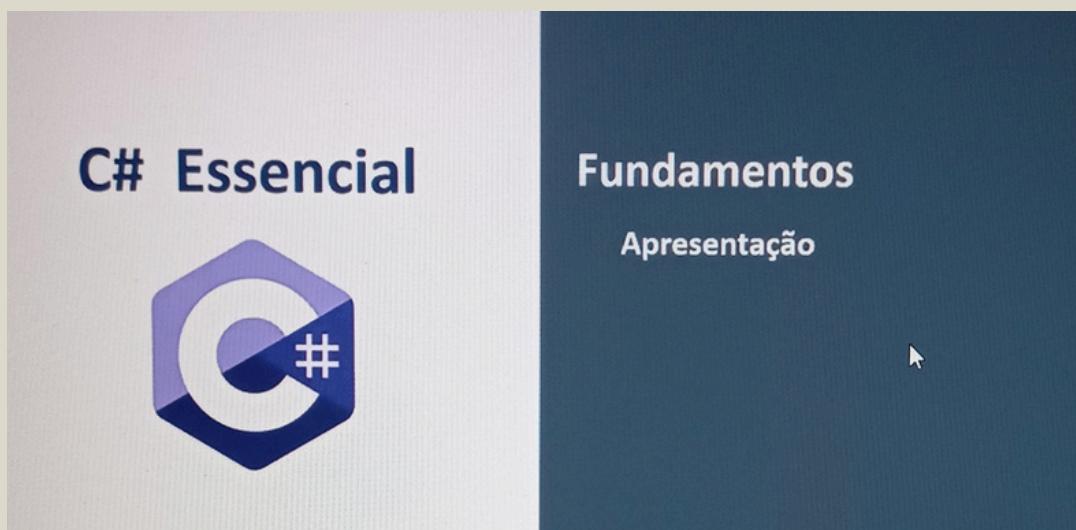
```
dotnet new console -o MeuApp1 --use-program-main
```

- Como escolher a versão via linha de comando
- Listando SDKs
- Criando/ entrando na pasta

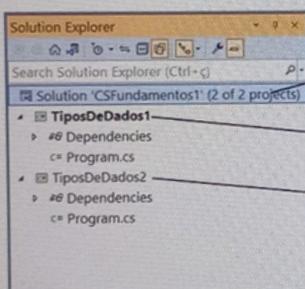
28/05

Seção 02: Fundamentos

17: Apresentação



Estrutura da solução e projetos criados no VS 2022



Para cada aula da seção teremos :

CSNome_da_Seção seguido de um número sequencial

→ Nome_do_Projeto1 → referente ao assunto tratado

→ Nome_do_Projeto2 → referente ao assunto tratado

Usar a notação Pascal Case

A primeira letra de cada palavra é iniciada com letra maiúscula
VariaveisEConstantes, TiposDeDados, ValoresPadrao,

Projetos criados usando o template Console App .NET 6.0 usando Top Level Statements

O que vai fazer na seção 2

Instruções usadas nos códigos dos programas

Console.WriteLine(variavel);

Escreve o valor da variável na janela do console (fluxo de saída padrão) seguido do pelo terminador de linha

Console.Write(variavel);

Escreve o valor da variável na janela do console (fluxo de saída padrão)

Console.ReadLine();

Lê a próxima linha de caracteres do fluxo de entrada padrão (console)

Dicas de atalhos usados no Visual Studio 2022

CTRL+F5

Executar o código sem depuração

Na depuração
podemos encontrar e
remover erros do
programa

F5

Executar o código com depuração

CTRL+K+D

Organizar o código

CTRL+D

Duplicar a linha onde o cursor está

CTRL+F

Localizar algo no documento atual

18: Tipos de dados

C# Essencial



The C# logo consists of a blue hexagon containing a white 'C' and a white '#' symbol.

Tipos de dados

- Conceito
- Os tipos de dados pré-definidos
- Variáveis e Constantes
- Declaração de variáveis

Tipos de dados

As informações em um computador são armazenadas inicialmente na memória RAM

A memória do computador se organiza como um armário com várias divisões. Sendo cada divisão identificada por um endereço

Como temos diversos tipos de informações, para otimizar o seu armazenamento , a linguagem C# divide essas informações em diferentes tipos de dados

Exemplo de tipos de dados : *números inteiros, números decimais, caracteres simples, textos, tipos complexos, estruturas de dados, etc.*

Tipos de dados

As informações em um computador são armazenadas inicialmente na memória RAM

A memória do computador se organiza como um armário com várias divisões. Sendo cada divisão identificada por um endereço

Como temos diversos tipos de informações, para otimizar o seu armazenamento , a linguagem C# divide essas informações em diferentes tipos de dados

Exemplo de tipos de dados : *números inteiros, números decimais, caracteres simples, textos, tipos complexos, estruturas de dados, etc.*

Essas informações são armazenadas em **variáveis ou constantes** na linguagem C#

Tipos de dados

A linguagem C# é uma linguagem de programação *fortemente tipada*

Na linguagem C# precisamos informar qual o *tipo de dados* que vamos usar toda vez que declararmos uma *variável ou constante*

Para isso a linguagem C# define *tipos pré-definidos* que podemos usar em nossos programas



Tipos de dados de referência e de valor

Tipos de referência : Não armazenam os dados diretamente e cada variável contém uma referência ao local da memória onde os dados estão armazenados

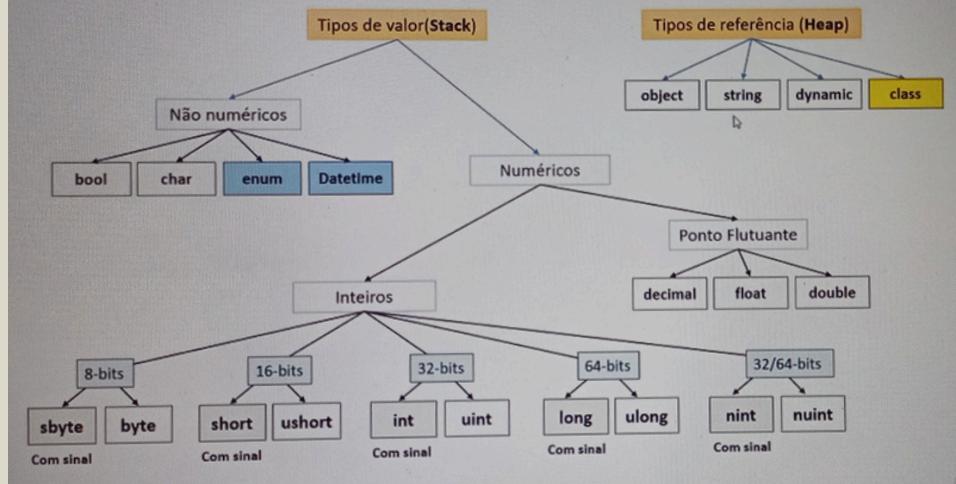
Os tipos de referência são armazenados na memória **Heap**

Tipos de valor : Armazenam diretamente seus dados e cada variável tem sua própria cópia dos dados

Os tipos de valor são armazenados na memória **Stack (LIFO – Last in First out)**



Tipos pré-definidos : de valor e de referência



Variáveis e constantes

Uma **variável** é um nome que representa a informação ou dados armazenados na memória durante a execução do programa

Cada variável em C# precisa ter **um tipo específico**, que determina o *tamanho e o uso da memória*

Uma **constante** é uma *variável cujo valor não se altera* durante o tempo de vida do programa.

Variáveis e constantes

Uma variável deve ser declarada antes de ser usada

A declaração de uma variável geralmente contém :

- 1 – O tipo de dados da variável
- 2 – O nome da variável
- 3 – A atribuição de um valor (*opcional*) usando o sinal = (igual)

Variáveis e constantes

↓
Tipo de dados
↓
int valor = 123;
↑
Nome da variável

int valor;

-A variável foi declarada mas não foi inicializada
-O compilador atribuir o valor padrão 0

const int valor = 123;

-Define uma constante cujo valor não se altera durante a execução

19: Tipos numéricos integrais

Tipos pré-definidos : Tipos numéricos integras

palavra-chave/tipo C#	Intervalo	Tamanho	Tipo .NET
sbyte	-128 a 127	Inteiro de 8 bits com sinal	System.SByte
byte	0 a 255	Inteiro de 8 bits sem sinal	System.Byte
short	-32.768 a 32.767	Inteiro de 16 bits com sinal	System.Int16
ushort	0 a 65.535	Inteiro de 16 bits sem sinal	System.UInt16
int	-2.147.483.648 a 2.147.483.647	Inteiro assinado de 32 bits	System.Int32
uint	0 a 4.294.967.295	Inteiro de 32 bits sem sinal	System.UInt32
long	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Inteiro assinado de 64 bits	System.Int64
ulong	0 a 18.446.744.073.709.551.615	Inteiro de 64 bits sem sinal	System.UInt64
nint	Depende da plataforma (computada em runtime)	Inteiro com sinal de 32 bits ou 64 bits	System.IntPtr
nuint	Depende da plataforma (computada em runtime)	Inteiro sem sinal de 32 bits ou 64 bits	System.UIntPtr

- São tipos por valor e dão suporte a operadores aritméticos de comparação e de igualdade

int valor = 123;
System.Int32 valor = 123;

O Valor padrão destes tipos é zero (0)
int valor;
System.Int32 valor;

20: Tipos de números de ponto flutuante

Tipos pré-definidos : Tipos numéricos de ponto flutuante

palavra-chave/tipo C#	Intervalo aproximado	Precisão	Tamanho	Tipo .NET
float	$\pm 1,5 \times 10^{-45}$ para $\pm 3,4 \times 10^{38}$	~6 a 9 dígitos	4 bytes	System.Single
double	$\pm 5,0 \times 10^{-324}$ to $\pm 1,7 \times 10^{308}$	~15 a 17 dígitos	8 bytes	System.Double
decimal	$\pm 1,0 \times 10^{-28}$ para $\pm 7,9228 \times 10^{28}$	28 a 29 dígitos	16 bytes	System.Decimal

- Representam números reais
- São tipos de valor armazenados na stack
- O valor padrão dos tipos de dados ponto flutuante é zero (0)
- Dão suporte a operadores aritméticos de comparação ($>$, $<$, $>=$, $<=$, $!=$) e de igualdade ($==$)
- O tipo double é usado para cálculos científicos e o decimal para cálculos financeiros
- São tipos que podemos inicializar usando literais (f-F , d-D, m-M)

```
double valor = 12.4;           float valor = 12.4F;           decimal valor = 12.4M;  
System.Double valor = 12.4;     System.Single valor = 12.4f;     System.Decimal valor = 12.4m;
```

21: Tipos não numéricos: bool e char

Tipos pré-definidos : Tipos não numéricos

palavra-chave/tipo C#	Intervalo	Tamanho	Tipo .NET
bool	true or false	8 bits	System.Boolean
char	U+0000 a U+FFFF	16 bits	System.Char

- São tipos de valor armazenados na stack
- O tipo bool pode ser obtido como resultado operações comparação e de igualdade
- O valor padrão do tipo bool é false
- O valor padrão do tipo char é '\0' (U+0000) - representação **unicode** para NUL

Tipos pré-definidos : Tipos não numéricos

palavra-chave/tipo C#	Intervalo	Tamanho	Tipo .NET
bool	true or false	8 bits	System.Boolean
char	U+0000 a U+FFFF	16 bits	System.Char

- São tipos de valor armazenados na stack
- O tipo bool pode ser obtido como resultado operações comparação e de igualdade
- O valor padrão do tipo bool é false
- O valor padrão do tipo char é '\0' (U+0000) - representação **unicode** para NUL

```
bool ativo = true;
System.Boolean ativo = true;
```

```
char letra = 'A';
System.Char letra= 'A';
```

22: Tipos de referência

Tipos pré-definidos : Tipos de referência	
Palavra-chave/tipo C#	Tipo .NET
string	System.String
object	System.Object
dynamic	System.Object

- São tipos de referência
- O valor padrão é null

```
object nota = 10;
object valor = 8.55m;
object nome = "Curso C#";
object ativo = true;
object letra = 'A';
```

```
string nome = "Curso Csharp Essencial";
System.String nome = "Curso Csharp Essencial";
```

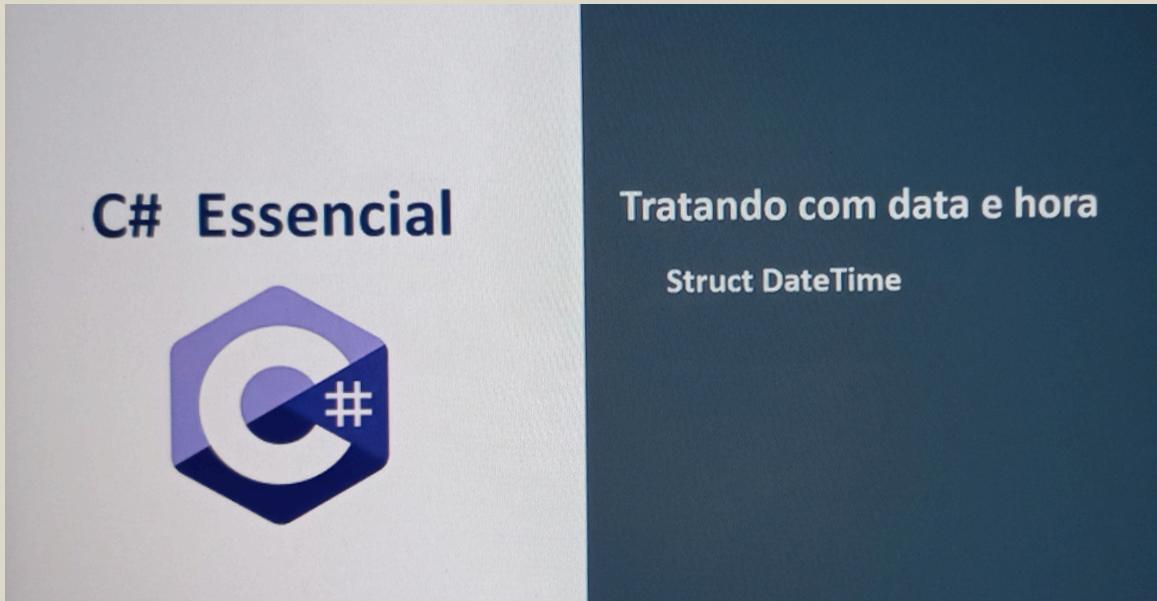
Reflection é usado para retornar metadados de tipos em tempo de execução

diferença de object e dynamic

DLR-
Dynamic
Language
Runtime

23: Tratamento de data e hora

DateTime



Tratando com data e hora - DateTime

Struct `DateTime`

Representa um momento no tempo expresso como uma **data e hora**

Uma variável do tipo `DateTime` é um *tipo de valor* e possui um *valor padrão*

O valor padrão de um `DateTime` é : **01/01/0001 00:00:00**

Ao usar `DateTime` a representação para o português do Brasil usa o formato :

dd/mm/aaaa hh:mm:ss

Obtendo uma data e hora

```
DateTime dataAtual = DateTime.Now;  
Console.WriteLine(dataAtual);
```

Exibe a representação da *data e hora local* formatada e expressa como:

04/09/2022 11:25:10

↓ ↓
dd/mm/aaaa hh:mm:ss

Criando uma data e hora específica

```
DateTime data = new DateTime(2022,09,04)  
Console.WriteLine(data);
```

Usa o operador `new` para criar uma data específica que deve ser especificada no formato **(aaaa,mm,dd)**

Para definir a data e a hora usamos o formato **(aaaa,mm,dd, hh, mm, ss)**

```
DateTime data = new DateTime(2022,09,04, 11, 10, 20)  
Console.WriteLine(data);
```

Operações com data e hora

1- Extrair informações como dia, mês, hora, ano, etc

`Year, Month, Day, Hour, Minute, Second, Millisecond`

2- Adicionar dias, horas, mês , anos, etc.

`AddDays, AddHours, AddMonths, AddYears`

3- Obter dia da semana e do ano

`DayOfWeek, DayOfYear`

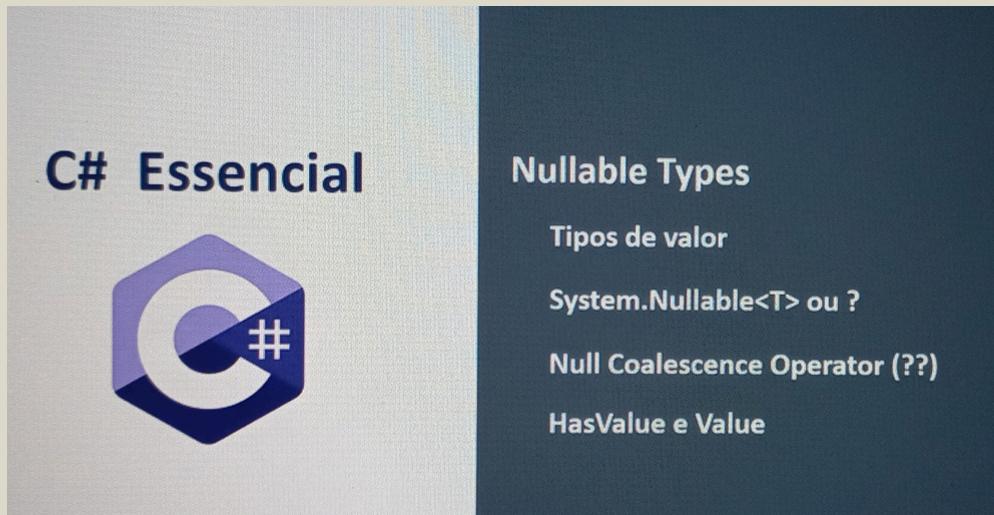
4- Expressar data no formato longo e abreviado

`ToLongDateString, ToShortDateString`

5- Expressar hora no formato longo e abreviado

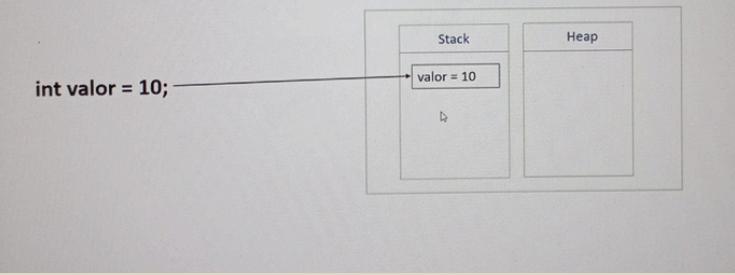
`ToLongHourString, ToShortHourString`

24: Nullable types



Tipos de valor

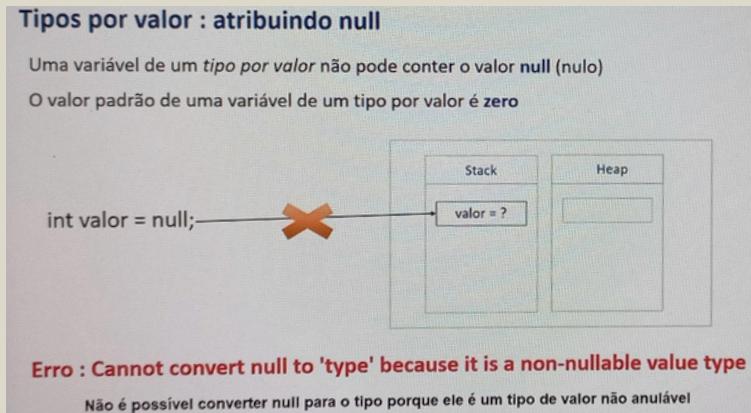
As variáveis de um tipo de valor (*tipos numéricos, char, bool, struct*) sempre tem um valor.
São armazenadas na memória Stack



Tipos por valor : atribuindo null

Uma variável de um *tipo por valor* não pode conter o valor **null** (nulo)

O valor padrão de uma variável de um tipo por valor é **zero**



não pode converter int null

??

Nullable Types ou Tipos Anuláveis

Um Nullable Type é um tipo de valor que pode receber um valor null

Os Nullable Types ou Tipos Anuláveis permitem atribuir um valor null a um tipo de valor

Nullable<T> <nome> = null; (T = int, double, float, bool, etc.)

Os Nullable Types suportam os valores do tipo mais o valor null

Ex: Nullable<bool> b = null; //suporta true, false e null

Nullable Types : declaração simplificada usando ?

Para simplificar a declaração podemos usar o operador ?

```
int? i = null;  
double? d = null;  
float? f = null;  
bool? b = null;
```

Nullable Types são diferentes dos tipos por valor

O Nullable Type int? é diferente do tipo int

int é um tipo não anulável ou Non-Nullable Type

int? é um Nullable Type

```
int? a = null;  
int b = a;  
Console.WriteLine(b);
```

Podemos atribuir o valor de uma variável de um tipo de valor para um Nullable Type mas não vice-versa

Cannot implicitly convert type int? to 'int'. An explicit conversion exists (are you missing a cast?)

Atribuir um Nullable Type a um tipo por valor usando operador ??

Use o operador '??' para atribuir um tipo anulável a um tipo não anulável.

```
int? a = null;  
int b = a ?? 0;  
Console.WriteLine(b);
```

O operador de coalescência nula ?? retorna o valor do operando esquerdo se ele não for null; caso contrário, ele avalia o operando direito e retornará seu resultado.

O operador ?? não avalia o operando do lado direito se o operando esquerdo for avaliado como não nulo.

Atribuir um Nullable Type a um tipo por valor : expressões

```
int? x = 4;  
int y = 3;  
int z = x * y;  
Console.WriteLine(z);  
Console.ReadLine();
```

Cannot implicitly convert type int? to 'int'. An explicit conversion exists (are you missing a cast?)

```
int? x = 4;  
int? y = 3;  
int? z = x * y;  
Console.WriteLine(z);  
Console.ReadLine();
```

Propriedades somente leitura: HasValue e Value

São usadas para examinar e obter um valor de uma variável de Nullable Type.

HasValue : true se tiver um valor false se não tiver um valor (null);

Value : Exibe o valor atribuído;

```
int? b = 100;  
if (b.HasValue)  
{  
    Console.WriteLine($"b = {b.Value}");  
}  
else  
{  
    Console.WriteLine("b não possui um valor");  
}
```

25: Nomenclatura: convenções

C# Essencial



Nomenclatura : Convenções

- Identificadores**
- Regras Gerais**
- Pascal Case**
- Camel Case**

Identificadores

Um **identificador** é o nome que você atribui a um tipo (classe, interface, struct, record, delegate ou enum), membro, propriedade, variável ou namespace.

Utilizar nomes significativos que permitam inferir o propósito do identificador.
Ex: calculoDoImpostoICMS, valorTotalComDesconto, dataVencimentoFatura

Não criar identificadores usando o mesmo nome mas alterando o uso de maiúsculas e minúsculas. Ex: valorTotal, ValorTotal, Valor_Total, valortotal

Case Sensitive : significa que é sensível a letras maiúsculas e minúsculas, ou seja, a mesma letra maiúscula e minúscula é considerada diferente

Regras gerais para identificadores válidos

Devem começar com letra ou sublinhado (_)

Não podem iniciar com um número ou caracteres especiais : #\$_@%&{|}!=+-*

Não podem conter espaços

Podem usar caracteres Unicode : `pre\u00E7o (\u00E7 -> ç)` (válido mas não recomendado)

Não pode ser uma palavra-reservada da linguagem : if, int, double, string, class,

Não podem exceder 512 caracteres

Regras gerais para identificadores válidos

Devem começar com letra ou sublinhado (_)

Não podem iniciar com um número ou caracteres especiais : #\$_@%&{|}!=+-*

Não podem conter espaços

Podem usar caracteres Unicode : `pre\u00E7o (\u00E7 -> ç)` (válido mas não recomendado)

Não pode ser uma palavra-reservada da linguagem : if, int, double, string, class,

Não podem exceder 512 caracteres

Válidos

```
int idade; ou int idade1;
string _imposto;
float valorTotal;
decimal pre\u00E7o (decimal preço)
```

Inválidos

```
int 9idade;
string $imposto;
float valor total;
decimal class;
```

Convenções

Camel Case - A primeira letra da *primeira palavra* é iniciada com *minúscula*. A *letra de cada palavra* seguinte deve ser iniciada com *maiúscula*. (*Não deve haver espaços entre as palavras*)

Ex: valorDoDesconto , nomeCompleto, valorDoImpostoSobreServiço

Usado em *nome de variáveis, parâmetros e campos internos privados*

Pascal Case - A *primeira letra* de cada palavra é iniciada com *maiúscula*. (*Não deve haver espaços entre as palavras*)

Ex: CalculaImpostoDeRenda, ValorDoDesconto, NomeCompleto

Usado em *nomes de classes, métodos, interfaces, propriedades*

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions>

Convenções

Constantes - Usar letras maiúsculas.

Ex: PI , DESCONTO, VALOR, IMPOSTO ,PESSOA_FISICA

Sublinhado (_) - Usado para campos internos privados e somente leitura (*camel case*)

Ex: _valorTotal, _calculoImposto, _precoComDesconto

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions>

26: Exercício - fase 01



1-6, 13 código git

7- Quais as diferenças entre os tipos por valor e os tipos por referência ?

- Tipos por valor – Armazenam os dados diretamente onde cada variável tem a cópia dos dados e são armazenados na memória Stack
- Tipos por referência – Não armazenam os dados diretamente e cada variável contém uma referência ao local da memória onde os dados estão. São armazenados na memória Heap

8- Inclua o tipo de dados correto entre os parênteses nas seguintes declarações de variáveis:

(int) x = 10;

(double) numero = 7.99;

(char) letra = 'C';

(float) temperatura = 27.4f;

(bool) ativo = false;

(string nome = "Manoel";

(Decimal) salario = 950.99m;

(DateTime) hoje = DateTime.Now;

9- Dada as variáveis declaradas a seguir classifique-as em tipos por valor(V) e tipos por referência (R) :

(v) int n = 1;

(r) string titulo = "A vida";

(v) float f = 12.45f;

(v) double d = 5.45;

(v) decimal valor = 10.99m;

(v) char sexo = 'M';

(r) object o = null;

10- O que é um nullable type e qual a sua utilidade ?

É um tipo de valor que pode receber um valor null. São usados para representar um valor indefinido/ausente para um tipo de valor ou para tratar com valores null em cenários onde podemos ter ou não valores atribuídos como informações de um banco de dados.

11- O que é Camel Case ? Dê um exemplo de sua aplicação.

Convenção de nomenclatura para nomes compostos onde a primeira letra da primeira palavra é iniciada com minúscula e a letra de cada palavra seguinte que compõe o nome deve ser iniciada com maiúscula.
Ex: taxaDeDesconto , impostoRendaPessoaFisica, descontoSobreValorTotal

12- O que é Pascal Case ? Dê um exemplo de sua aplicação.

É um tipo de valor que pode receber um valor null. São usados para representar um valor indefinido/ausente para um tipo de valor ou para tratar com valores null em cenários onde podemos ter ou não valores atribuídos como informações de um banco de dados.

Convenção de nomenclatura para nomes compostos onde a primeira letra da primeira palavra é iniciada com minúscula e a letra de cada palavra seguinte que compõe o nome deve ser iniciada com maiúscula.
Ex: taxaDeDesconto , impostoRendaPessoaFisica, descontoSobreValorTotal
Convenção de nomenclatura para nomes compostos onde a primeira letra de cada palavra que compõe o nome deve ser iniciada com maiúscula Ex: NomeCompletoAssinante , ValorImpostoSobreServico , TotalDesconto

14- Quais os valores padrões dos tipos de dados bool, char, int, double, float, decimal e string.

int x = 77 , y = 66 ;

Console.WriteLine(x+y);

bool → false

char → '\0' ou (U+000)

int, double, float e decimal → zero (0)

string → null

15- Indique verdadeiro(V) ou falso (F) para as seguintes declarações de variáveis considerando a nomenclatura usada:

(f) double 1valor = 12.45;

(f) string #nome = "Pedro";

(v) float _temperatura = 12.45f;

(f) double int = 5;

(f) decimal renda extra = 91.45m;

(f) bool status\$conta = false;

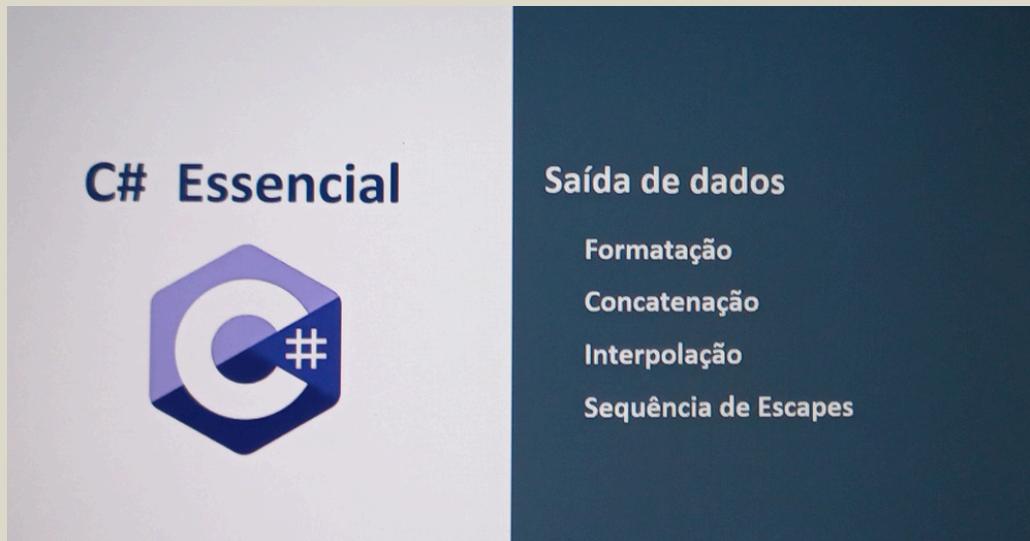
(v) string titulo3 = "Tópico 1";

(v) float salario_mensal = 1999.55f;

(v) int percentualValorDesconto = 5;

(v) const bool MENSALIDADE_EM_DIA = true;

27: Formatar saída de dados



Saída de dados : Formatação

Concatenação - É o processo de acrescentar uma cadeia de caracteres ao final de outra cadeia de caracteres. Para isso usamos o operador +.

```
Console.WriteLine(nome + " tem " + idade + " anos");
```

Interpolação - Usa objetos e expressões para realizar uma *interpolação de strings*. Para isso usa o operador \$ para indicar a interpolação e {} para conter as variáveis a serem substituídas

```
Console.WriteLine($"{nome} tem {idade} anos");
```

Place Holders – Realiza a concatenação de expressões usando objetos. Para isso usa {} e uma numeração iniciada em 0 para indicar a ordem de substituição das variáveis que devem ser informadas na ordem a serem exibidas.

```
Console.WriteLine("{0} tem {1} anos", idade, nome);
```

28: Formatação - sequência de escape

Formatação usando sequencias Escapes

As sequências de escape são combinações de caracteres consistindo de uma barra invertida (\) seguida por uma letra ou por uma combinação de dígitos.

Sequência Escape	Representação
\a	Sinal sonoro (alerta)
\b	Backspace
\f	Alimentação de formulário
\n	Nova linha
\r	Carriage return
\t	Tabulação horizontal
\v	Tabulação vertical
'	Aspas simples
"	Aspas duplas
\\\	Barra invertida
\?	Interrogação
\u 000	Caractere ASCII na notação unicode
\x hh	Caractere ASCII na notação hexadecimal

```
string local = "c:\\dados\\\\poesias.txt";
string frase = "Ele falou:\"Não fui eu\"";
string pizza = "\\nPizz\\nde\\nMussarela";
string bolo = "\\nBolo\\tde\\tChocolate";
```

29-30: Conversão I e II

The image shows a presentation slide with a light gray header and a dark blue footer. In the header, the text "C# Essencial" is displayed in a blue sans-serif font, followed by a large purple hexagonal logo containing a white "C#" symbol. The footer contains a list of four items in white text: "Conversão de Tipos", "Conversão Implícita", "Conversão Explícita", and "Método ToString()". Below these is another item: "Métodos da classe Convert".

C# Essencial

Conversão de Tipos

Conversão Implícita

Conversão Explícita

Método ToString()

Métodos da classe Convert

Conversão entre os tipos

A linguagem C# é **estaticamente tipada** em tempo de compilação

Após uma variável ser declarada ela *não pode ser declarada novamente*

Nem pode ser usada para armazenar valores de outro tipo de dados

A menos que este tipo de dados seja convertível para o tipo de dados da variável

O processo de converter um valor de um tipo de dados para outro tipo de dados é chamado de **conversão de tipos**

Conversão de tipos usando a classe Convert()

Fornece diversos métodos para converter um tipo de dados em outro tipo de dados.

Método	Descrição
ToBoolean()	converte um tipo para um valor Boolean
ToChar()	converte um tipo para o tipo char
ToDouble()	converte um tipo para o tipo double
ToInt16()	converte um tipo para o tipo 16-bit
ToInt32()	converte um tipo para o tipo 32-bit
ToString()	converte um tipo para uma string

Esta classe pertence ao namespace System

Tipos de conversão entre tipos de dados

Conversão Implícita

O compilador C# converte automaticamente um tipo de dados em outro tipo (*Quando a conversão entre os tipos for compatível*)

Conversão Explícita

A conversão tem que ser feita manualmente de forma explícita

Conversão de tipos usando a classe Convert()

As conversões de *ampliação ou estreitamento* entre dois tipos de dados que não resultarem em *perda de dados*, terão êxito e o método retornará um valor do tipo de destino

Quando uma conversão de *estreitamento de dados* resultar em perda de dados vai ocorrer uma **OverflowException**

```
int varInt = 10000;  
Console.WriteLine(Convert.ToInt32(varInt));
```

Conversão para String usando método ToString()

O método **ToString()** da classe **Object** retorna uma string que representa o objeto atual

Converte um objeto em sua *representação de cadeia de caracteres para exibição*

Como **Object** é pai de todos os objetos na linguagem C# todos os objetos herdam o método **ToString()** da classe **Object**.

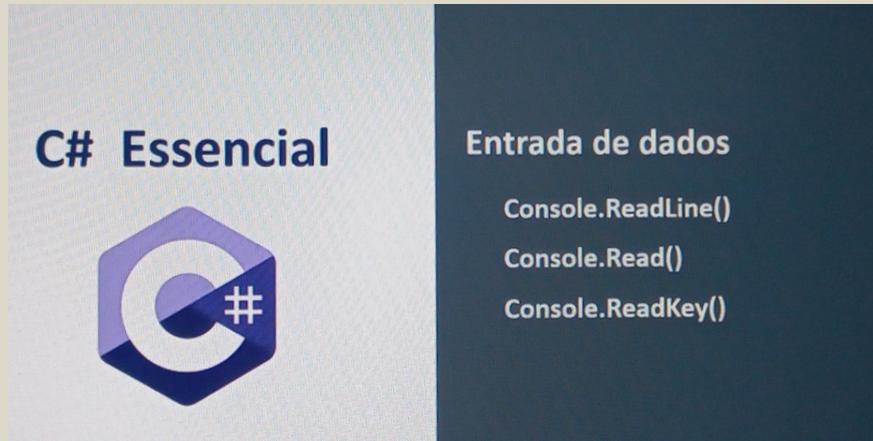
Tipos de conversões implícitas possíveis

sbyte	short, int, long, float, double, decimal, or nint
byte	short, ushort, int, uint, long, ulong, float, double, decimal, nint, or nuint
short	int, long, float, double, or decimal, or nint
ushort	int, uint, long, ulong, float, double, or decimal, nint, or nuint
int	long, float, double, or decimal, nint
uint	long, ulong, float, double, or decimal, or nuint
long	float, double, or decimal
ulong	float, double, or decimal
float	double
nint	long, float, double, or decimal
nuint	ulong, float, double, or decimal

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/numeric-conversions>

//byte -> 1 byte
//short -> 2 bytes
//int -> 4 bytes
//long -> 8 bytes
//float -> 4 bytes
//decimal -> 16 bytes

31: Entrada de dados



Entrada de dados

`Console.ReadLine()`

`Console.Read()`

`Console.ReadKey()`

Entrada de dados : métodos da classe Console

ReadLine(): lê uma única linha de entrada do fluxo de entrada padrão.
Retorna a mesma string.

↳

Read(): lê apenas um único caractere do fluxo de entrada padrão.
Retorna o valor ASCII do caractere.

.ReadKey(): lê apenas um único caractere do fluxo de entrada padrão.
Obtém a próxima tecla pressionada pelo usuário.
Retorna um tipo `ConsoleKeyInfo`.
É usado para segurar a tela até que o usuário pressione uma tecla.

32: Operadores Aritméticos e class

C# Essencial

Operadores Aritméticos

Operadores binários
+, -, *, / e %

Classe Math

Operadores binários

Operação	Operador	Exemplo
adição	+	$x + y$
subtração	-	$x - y$
multiplicação	*	$x * y$
divisão	/	x / y
módulo(*)	%	$x \% y$

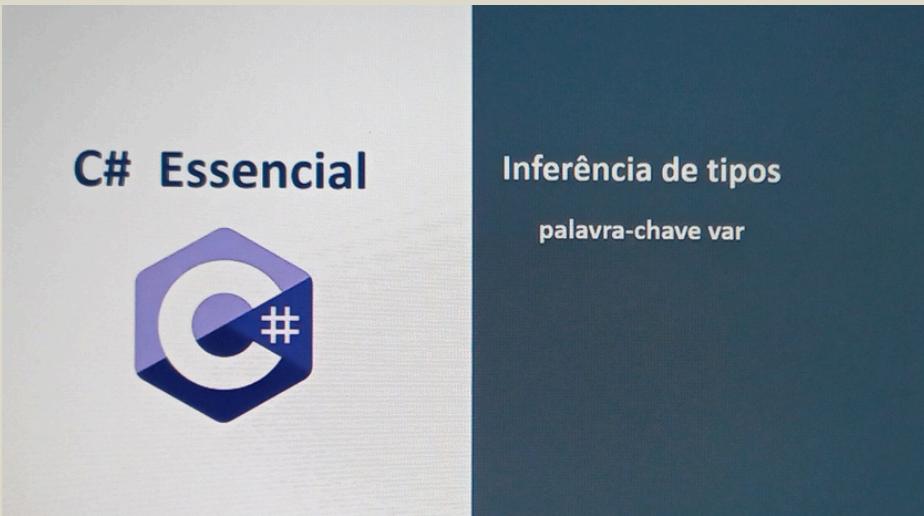
(*) Retorna o resto da divisão

Classe Math

Possui constantes e diversos **métodos estáticos** usados para cálculos matemáticos

Método ou Constante	Resultado	Exemplo
Math.PI	Representa o número Pi	$\text{Pi} = 3.141592653589793...$
Math.E	Representa a base e do logaritmo natural	$\text{E} = 2.7182818284590451$
Cos(x)	Obtém o cosseno de x	$\text{Cos}(2.0) = 0.416151187233593...$
Sin(x)	Obtém o seno de x	$\text{Sin}(2) = 0.909297426826574...$
Tan(x)	Obtém a tangente de x	$\text{Tan}(1.5) = 14.1014925359038...$
Sqrt(x)	Calcula a raiz quadrada de x	$\text{Sqrt}(169) = 13$
Pow(x,y)	Obtém o valor de x elevado a y	$\text{Pow}(2,4) = 16$
Abs(x)	Fornecê o valor absoluto de x	$\text{Abs}(-4.5) = 4.5;$
Max(x,y)	Obtém o maior valor entre dois números	$\text{Max}(2.46,2.56) = 2.56;$
Min(x,y)	Obtém o menor valor entre dois números	$\text{Min}(1.92,1.89) = 1.89;$
Log10(x)	Calcula o logaritmo de x na base 10	$\text{Log10}(3.0) = 0.47712...$
Log(x)	Calcula o logaritmo de x na base e	$\text{Log}(3.0) = 1.098612...$
Exp(x)	Retorna o exponencial (e elevado a x)	$\text{Exp}(5.0) = 148.413159102576...$

33: Interferência de tipos



Linguagem fortemente tipada

A linguagem C# é *fortemente tipada* e a declaração de tipo padrão é o *tipo explícito*

`string nome = "Maria";` ~~`nome = Maria;`~~
`int idade = 10;` ~~`idade = 10;`~~

`int x = 13;` Declara e atribui o valor imediatamente

`int y;` Declara e atribui o valor mais tarde
`y = 13;`

`decimal salario, imposto, total;`

Inferência de tipos : var

A partir da versão 3.0 da linguagem C# as variáveis que forem declaradas no escopo do método podem possuir *um tipo implícito var*

Usar a palavra-chave `var` para instruir o compilador para *deduzir o tipo da variável da expressão a partir do lado direito da instrução de inicialização*

O tipo inferido pode ser um *tipo interno*, um *tipo anônimo*, um *tipo definido pelo usuário* ou um *tipo definido na biblioteca de classes da plataforma .NET*

Inferência de tipos - palavra-chave var

`int x = 0 ;` → Definição explícita ou direta do tipo de dados da variável x

`var x = 0 ;` → Definição implícita ou indireta do tipo de dados da variável x

int

`var x;` → ERRO Implicitly-typed variables must be initialized

Inferência de tipos - palavra-chave var : limitações

A palavra-chave `var` só pode ser usada quando uma variável local é declarada e inicializada na mesma instrução;
(Implicitly-typed variables must be initialized)

Não é possível inicializar a variável como `null`
(Cannot assign <Null> to implicitly-typed variable)

Múltiplas variáveis de *tipo implícito* não podem ser inicializadas na mesma instrução;
(Implicitly-typed variables cannot have multiple declarators)

Não podemos alterar o tipo da variável `var` depois de inicializada.

Inferência de tipos - palavra-chave var : Usos

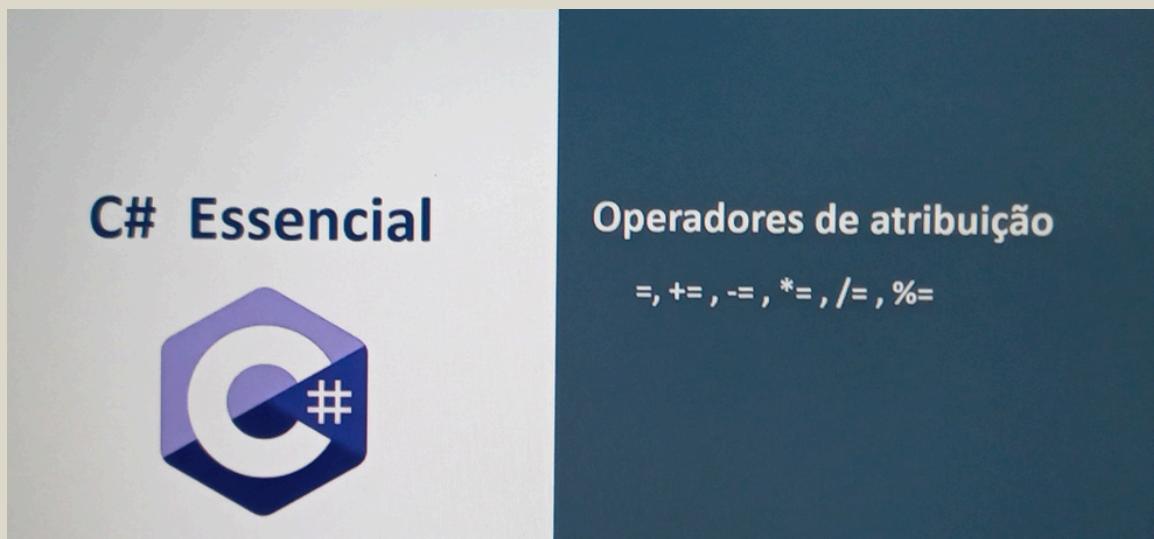
Considerado uma açúcar sintático (sugar syntax)

Usada para declarar tipos anônimos

Usada em laços `for` e `foreach`

Usada em instruções `using`

34: Operadores de atribuição



Operadores de atribuição

Operador	Exemplo	Significado
=	x = 10;	x recebe o valor 10
+=	x += 5;	x recebe o valor de x+5 → (x = x + 5)
-=	x -= 5;	x recebe o valor de x-5 → (x = x - 5)
*=	x *= 5;	x recebe o valor de x*5 → (x = x * 5)
/=	x /= 5;	x recebe o valor de x/5 → (x = x / 5)
%=	x %= 5;	x recebe o valor de x%5 → (x = x % 5)

Operadores de atribuição : Tipos numéricos

<code>var x = 10;</code>	valor de x é 10
<code>x += 5;</code>	valor de x é 15 ou seja → <code>x = x + 5;</code>
<code>x -= 3;</code>	valor de x é 12 ou seja → <code>x = x - 3;</code>
<code>x *= 4;</code>	valor de x é 48 ou seja → <code>x = x * 4;</code>
<code>x /= 5;</code>	valor de x é 9 ou seja → <code>x = x / 5;</code>
<code>x %= 5;</code>	valor de x é 4 ou seja → <code>x = x % 5;</code>

Operador + e += com strings

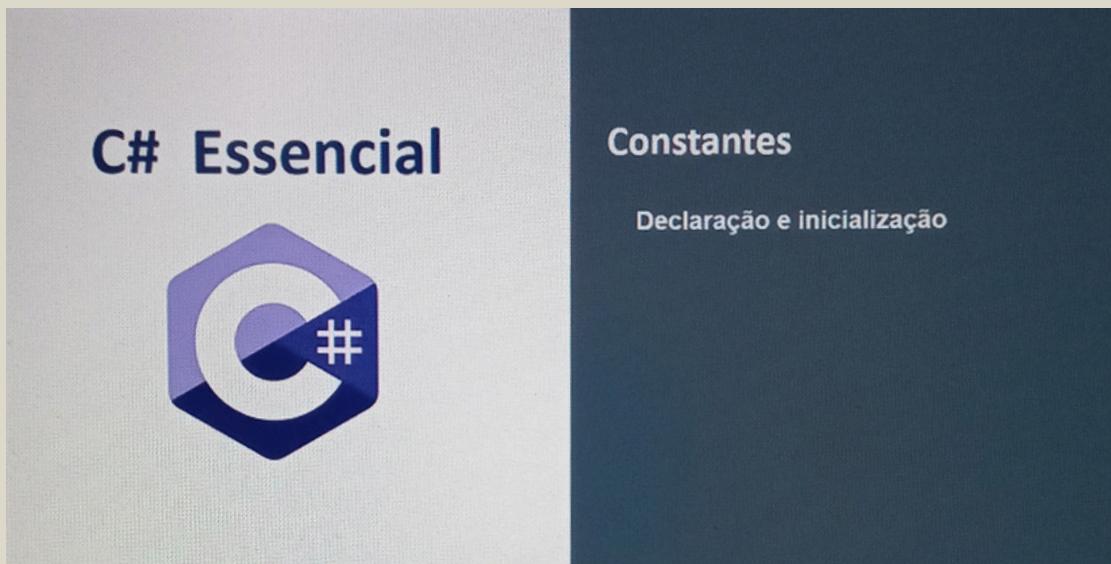
`var y = "123";` valor inicial de y é “123”

`y += "456";` valor de y é “123456” → `y = y + "456"`

Quando um ou ambos os operandos forem do tipo **string** o operador **+** vai *concatenar* as *strings* de seus operandos

O valor **null** representa uma **string** ou *cadeia de caracteres vazia* (“”)

35: Constantes



Constantes

Constantes são valores imutáveis que são conhecidos em *tempo de compilação* e não mudam durante a vida útil do programa.

As constantes são declaradas usando modificador **const** e devem ser inicializadas na sua declaração

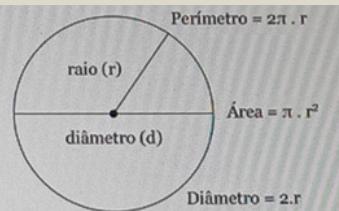
```
const int ANO = 12;  
  
const int MES = 30, SEMANA = 7, QUINZENA = 15;  
  
const int MESES_ANO = 12;  
const int DIAS_ANO = 365;  
  
const float DIAS_POR_MES = (float)DIAS_ANO / (float)MESES_ANO;
```

Constantes

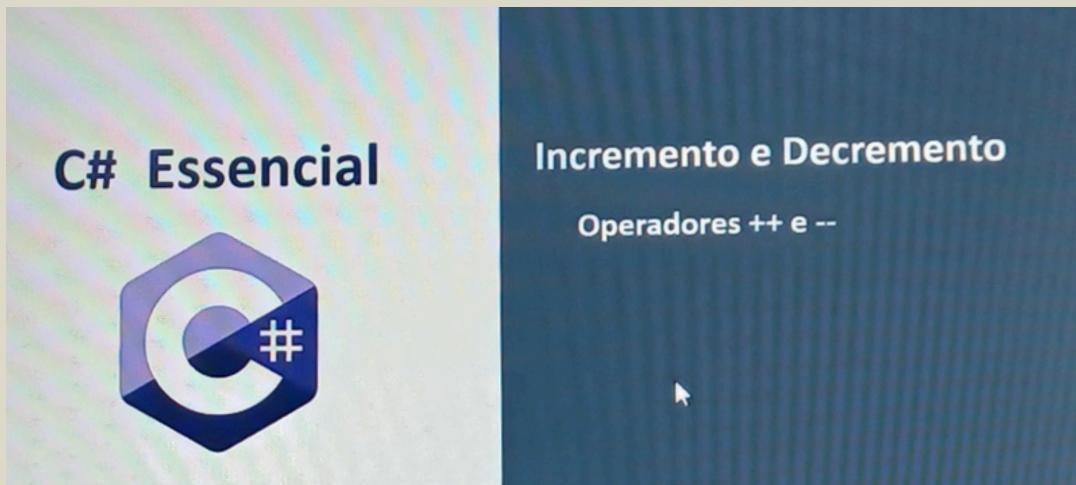
Calcular a área e o perímetro de um círculo

Perímetro = $2 * \pi * \text{raio}$; **Área** = $\pi * \text{raio} * \text{raio}$;

```
double raio, perimetro, area;  
const double PI = 3.14;  
  
Console.WriteLine("Informe o raio do circulo : ");  
raio = Convert.ToDouble(Console.ReadLine());  
  
perimetro = 2 * PI * raio;  
area = PI * raio * raio;  
  
Console.WriteLine($"Perímetro = {perimetro}");  
Console.WriteLine($"Área      = {area}");
```



36: Operadores de incrementos e decrementos



Operadores de incremento (++) e decremento(--)

Operador de incremento ++

Aumenta de uma unidade (1) o valor de uma variável

int x = 10;
x++;  **11** **x = x + 1;**

Operador de decremento --

Diminui de uma unidade (1) o valor de uma variável

int x = 10;
 x--;

Operadores de incremento (++) e decremento(--)

Operador de incremento ++

Aumenta de uma unidade (1) o valor de uma variável

int x = 10;
x++;  **11** **x = x + 1;**

Operador de decremento --

Diminui de uma unidade (1) o valor de uma variável

int x = 10;
x--;  **9** **x = x - 1;**

Operadores incrementais e decrementais

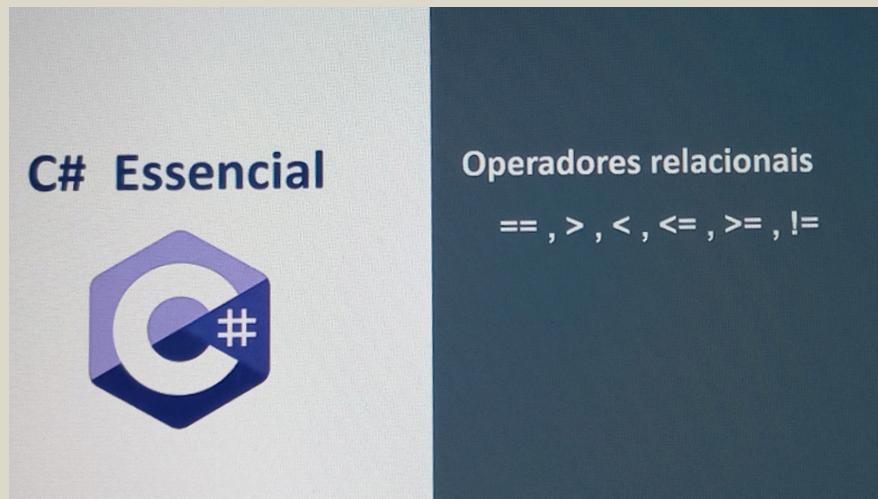
Objetivo : Aumentar ou Diminuir exatamente em uma unidade o valor de uma variável

Operador de Incremento : ++

Operador de Decremento : --

Operador	Exemplo	Significado
++	x++; ou ++x;	Incrementa x de uma unidade $\rightarrow x = x + 1;$
--	x--; ou --x;	Decrementa x de uma unidade $\rightarrow x = x - 1;$

37: Operadores relacionais



Operadores Relacionais

A característica dos operadores relacionais é que o resultado de uma operação relacional terá como resultado true ou false (verdadeiro ou falso)

Operador	Significado	Exemplo para : int x=10; e int y=20;
<code>==</code>	Igualdade	<code>x == y</code> → resultado será False
<code>></code>	Maior que	<code>x > y</code> → resultado será False
<code><</code>	Menor que	<code>x < y</code> → resultado será True
<code>>=</code>	Maior ou igual	<code>x >= y</code> → resultado será False
<code><=</code>	Menor ou igual	<code>x <= y</code> → resultado será True
<code>!=</code>	Não Igual ou diferente de	<code>x != y</code> → resultado será True

Operadores Relacionais

```
int x = 10;  
int y = 20;  
Console.WriteLine( x == y );      // False
```

```
int x = 10;  
int y = 20;  
bool resultado = x == y ;  
Console.WriteLine( resultado ); // False
```

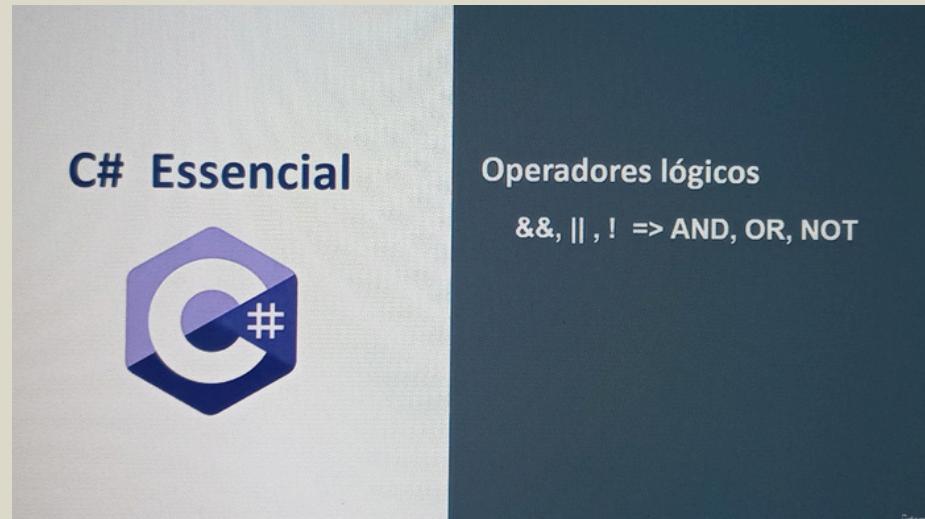
Operador relacional de igualdade (==) com string

```
string a = "curso";  
string b = "Curso";  
Console.WriteLine( a == b );      // False  
Console.WriteLine( a.Equals(b) ); // False
```

O método Equals é usado para determinar se duas strings possuem o mesmo valor ou não

```
string a = "curso";  
string b = "Curso";  
bool resultado = a == b ;      → bool resultado = a.Equals(b) ;  
Console.WriteLine( resultado ); // False
```

38: Operadores lógicos



Operadores Lógicos condicionais

São usados em expressões lógicas e trabalham com operandos booleanos e seu resultado será **true ou false (verdadeiro ou falso)**

Operador	Significado	$c1 = \text{true}; \text{ e } c2 = \text{false};$	Comportamento
&&	AND = E	$c1 \&\& c2 \rightarrow \text{false}$	Retornará False se apenas uma das condições for False
 	OR = OU	$c1 c2 \rightarrow \text{true}$	Retorna True se apenas uma das condições for True
!	NOT = Não	$!(c1 \&\& c2) \rightarrow \text{true}$	Inverte o resultado, retorna False se o resultado da expressão for True e vice-versa.

Operadores Lógicos – Tabela verdade

Condição1 – C1	Condição2 – C2	$C1 \&\& C2, C1 \text{ AND } C2$
F	F	F
F	V	F
V	F	F
V	V	V

Se qualquer condição se tornar falsa, o operador lógico **&&** ou **AND** retornará falso.

Condição1 – C1	Condição2 – C2	$C1 C2, C1 \text{ OR } C2$
F	F	F
F	V	V
V	F	V
V	V	V

Se qualquer condição se tornar verdadeira, o operador lógico **||** ou **OR** retornará verdadeiro

Condição	NOT
True	False
False	True

O operador lógico **!** ou **NOT** sempre retorna o inverso do valor do operando da condição

Operador Lógicos

```
bool c1 = 5 >= 7;           → False
```

```
bool c2 = 9 != 8;           → True
```

```
bool resultado;
```

```
resultado = c1 && c2;
```

```
Console.WriteLine("operador AND (&&) : " + resultado);
```

c1= False e c2=True

Resultado => False

```
resultado = c1 || c2;
```

```
Console.WriteLine("operador OR (||) : " + resultado);
```

c1= False e c2=True

Resultado => True

```
resultado = !c1;
```

```
Console.WriteLine("operador NOT (!) : " + resultado);
```

c1= False

Resultado => True

39: Precedência e associatividade de operadores

C# Essencial



The C# logo consists of a purple hexagon containing a white 'C' and a blue hexagon containing a white '#'. The two hexagons overlap.

Precedência e associatividade de operadores

Regras

Tabela de precedência e associatividade

Precedência e associatividade

A precedência do operador é um conjunto de regras que define como uma expressão é avaliada.

No C#, cada operador tem uma prioridade atribuída, e, com base nessa prioridade, a expressão é avaliada.

As expressões com operadores de maior precedência são avaliadas primeiro.

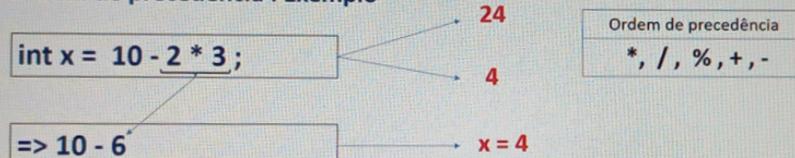
Quando dois operadores tiverem a mesma precedência eles são avaliados com base na associatividade do operador que pode ser da direita para esquerda ou da esquerda para a direita.

Tabela de Precedência e associatividade

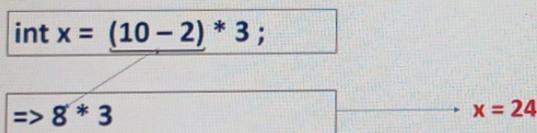
Operadores	Ordem Decrescente de Precedência	Associatividade
Aritméticos	* , / , % , + , -	Da esquerda para a direita
Atribuição	= , *= , /= , %=, += , -=	Da direita para a esquerda
Incremento/Decremento	++, -- (prefixo) ++a, --a ++, -- (sufixo) a++, a--	Da direita para a esquerda Da esquerda para a direita
Relacionais	< , > , <= , >= , == , !=	Da esquerda para a direita
Lógicos	! , && ,	! -> Da direita para a esquerda Da esquerda para a direita

Os operadores [] e () possuem a maior ordem de precedência, nesta ordem, e podem ser usados para alterar a ordem de precedência.

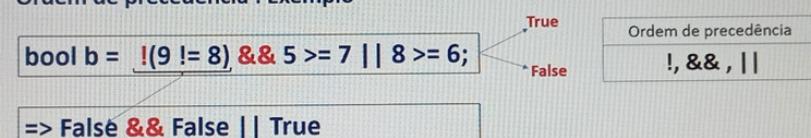
Ordem de precedência : Exemplo



Alterando a ordem de precedência usando [] ou ()



Ordem de precedência : Exemplo



Alterando a ordem de precedência usando ()

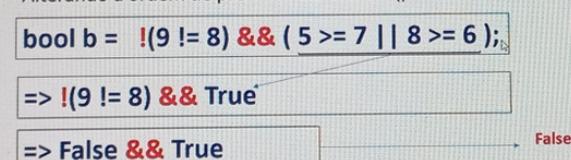


Tabela Geral de Precedência e associatividade

Ordem Decrescente de Precedência	Associatividade
[] , (), ++, -- (sufixo)	Da esquerda para a direita
! , ++ , -- (prefixo)	Da direita para a esquerda
* , /, %,	Da esquerda para a direita
+ , -	Da esquerda para a direita
< , > , <= , >=	Da esquerda para a direita
== , !=	Da esquerda para a direita
&&	Da esquerda para a direita
	Da esquerda para a direita
= , *= , /= , %=, += , -=	Da direita para a esquerda

Veja tabela completa em:

<https://docs.microsoft.com/pt-br/cpp/c-language/precedence-and-order-of-evaluation?view=msvc-170>

Ordem de precedência : Exemplo

int a = 5, b=6, c= 4;

int r = --a * b - ++c ;

Ordem de precedência

++(prefixo), --(prefixo) , * e -

→ int r = ((--a) * b) - (++c) ;

=> --5 * 6 - ++4 ;

=> 4 * 6 - ++4 ;

=> 4 * 6 - 5 ;

=> 24 - 5

↳

→ r = 19

Precedência e associatividade

int a = 5, b = 6, c = 3;
int resultado = a = b = c;

A associatividade do operador = é da direita para a esquerda

=> a = b = 3

=> a = 3

=> 3 → Resultado = 3;

↳

O valor de c (ou seja, 3) é atribuído a b, e então o valor de b é atribuído a a.

Depois de executar esta instrução, os valores de a, b e c serão igual 3.

40: Nullable reference types

The slide is divided into two main sections. The left section, titled "C# Essencial" with a large blue hexagonal logo containing a white "C#" and a smaller "Essencial" text below it, contains a list of three features: "Null e Nullable", "<Nullable>enable</Nullabe>", and "Null Conditional Operator (?.)". The right section, titled "Nullable Reference types", also lists these three features.

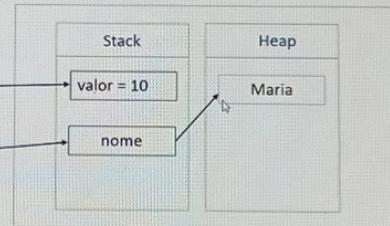
C# Essencial	Nullable Reference types
Null e Nullable	Null e Nullable
<Nullable>enable</Nullabe>	<Nullable>enable</Nullabe>
Null Conditional Operator (?.)	Null Conditional Operator (?.)

Tipos de valor e tipos de referência

As variáveis de um tipo de valor (*tipos numéricos, char, bool, struct*) sempre tem um valor e são armazenadas na memória Stack

As variáveis de um tipo de referência (*object, string, class*) contém uma referência ao local onde o valor pode estar armazenado e são armazenadas na memória Heap.

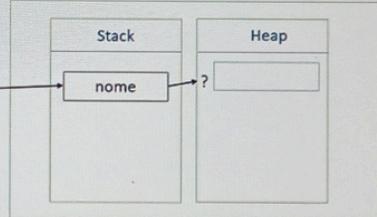
```
int valor = 10;  
string nome = "Maria";
```



Tipos por referência

O valor padrão de uma variável de um tipo por referência é **null**

```
string nome = null;
```



Alerta: Converting null literal or possible null value to non-nullable type

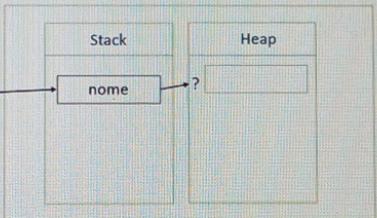
Tipos por referência

Um tipo de referência pode não ter nenhuma referência

Isso é expresso com o valor **null** (nulo)

Se uma variável de um tipo de referência for igual a **null** ela *não tem nenhuma referência* a um valor na memória Heap

```
string nome = null;
```



Isso também ocorre quando declaramos a variável de um tipo de referência mas não atribuímos um valor a ela

Tipos por referência : NullReferenceException

```
string nome = null;           → Converting null literal or possible null value to non-nullable type
Console.WriteLine(nome);

Console.WriteLine(nome.ToUpper());
```

↓

System.NullReferenceException: 'Object reference not set to an instance of an object.'

Nullable Reference Types

Nullable Reference Types – Alerta ativado

São um recurso de tempo de compilação

Este recurso nos alerta sobre a possibilidade de ocorrer um erro envolvendo a manipulação de uma referência nula

A finalidade dos *tipos de referência anuláveis* é minimizar a chance de seu aplicativo lançar um **System.NullReferenceException** quando executado

O alerta é ativado quando o elemento **<Nullable>** é definido como **enable** no arquivo de projeto :

```
<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>net6.0</TargetFramework>
  <ImplicitUsings>enable</ImplicitUsings>
  <Nullable>enable</Nullable>
</PropertyGroup>
```

Nullable Reference Types – Evitar o NullReferenceException

Para inibir o alerta podemos usar o tipo **Nullable (?)** ao atribuir o valor **null**

E empregar o **Null Conditional Operador (?.)** ao acessar a referência

```
string? nome = null;
Console.WriteLine(nome?.ToUpper());
```

Ou podemos inicializar as variáveis dos tipos de referência corretamente

```
string nome = "";
Console.WriteLine(nome?.ToUpper());
```

41: Operador ternário

<p>C# Essencial</p> 	<p>Operador Unitário e Ternário</p> <p>Uniário (+) e Uniário(-)</p> <p>Ternário (? :)</p>
--	--

Operadores uniários , binários e operador condicional ternário

Os operadores uniários atuam em um operando em uma expressão.

++ , -- , ! , + , -

Os operadores binários atuam em dois operandos em uma expressão.

**&& , || , != , >, <, >=, <=, *=, /=, %=, +=, -=
==, =, *, +, -, /, %**

O operador ternário utiliza **3 argumentos** e avalia uma expressão booleana

? :

Operador uniário -

O operador uniário menos (-) produz o **negativo do seu operando** (*deve ser de um tipo aritmético*)

```
Console.WriteLine("Informe o numero : \n");
var n = Convert.ToInt32(Console.ReadLine());

Console.WriteLine($"O negativo de {n} é igual a {-n}");
```

Operador uniário +

O operador positivo (+) não tem efeito na expressão com a qual é usado, e, retorna o valor do seu operando. (*deve ser de um tipo aritmético*)

```
int positivo = 1;
int resultado;

resultado = +positivo;
Console.WriteLine(resultado); —— 1
```

Operador condicional ternário (?:)

Avalia uma expressão booleana e retorna o resultado de uma das duas expressões dependendo se a expressão booleana é avaliada como true ou false.

O operador ternário (?:) é usado para validar uma condição.

```
condição ? expressão1_se_true : expressão2_se_false
```

Condição : Qualquer expressão booleana.

Expressão1_se_true : Uma expressão avaliada se a condição for verdadeira (true).

Expressão2_se_false : Uma expressão avaliada se a condição for falsa (false)

Operador condicional ternário (?:)

```
Console.WriteLine("Informe o temperatura: \n");
var temp = Convert.ToDouble(Console.ReadLine());

var resultado = temp > 27 ? "Quente" : "Normal" ;

Console.WriteLine($"O tempo está {resultado}");
```

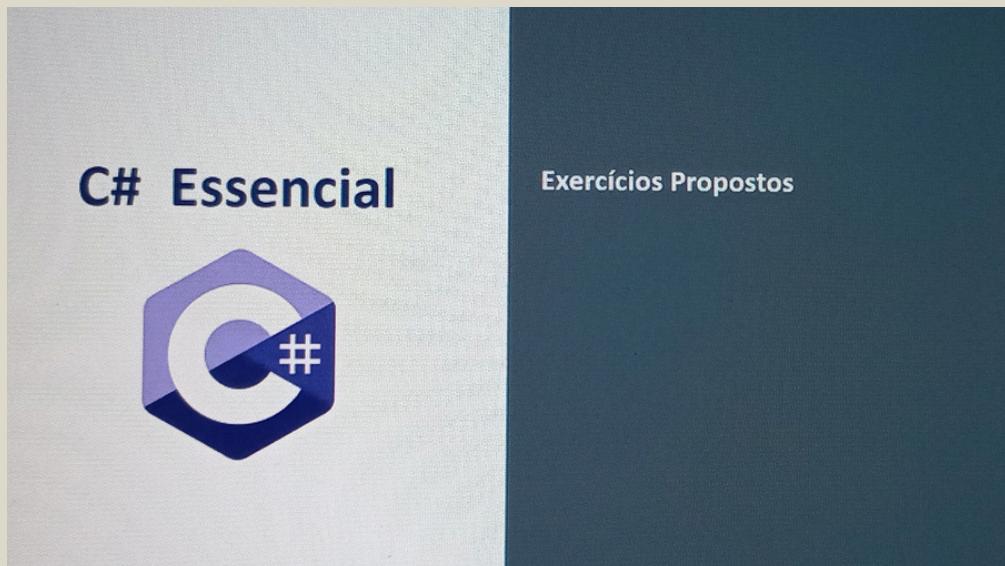
Operador condicional ternário (?:) - aninhando operações

```
Console.WriteLine("Informe o valor de x");
int x = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Informe o valor de y");
int y = Convert.ToInt32(Console.ReadLine());

string resultado = x > y ? "x é maior que y" :
                     x < y ? "x é menor que y" :
                     x == y ? "x é igual a y" : "Sem resultado";

Console.WriteLine(resultado);
```

42: Exercício - fase 02



Exercícios fase 2

3- Para qual tipo de dados você pode converter um **float implicitamente** ?

- () int
- double
- () long
- decimal

4- Em qual conversão numérica você precisaria realizar o casting (*conversão forçada*) ?

- () int para long
- double para long
- double para float
- decimal para float
- long para int
- double para decimal

6- Marque verdadeiro(V) ou falso(F) para os códigos abaixo:

- (F) long resultado = 1.32;
- (V) var nome = "Maria";
- (V) string resultado = 100.ToString();
- (F) A sequência de escape \n inclui uma nova linha
- (F) float f = 5.45;
- (V) decimal valor = (decimal) 10.99f;
- (F) var status = null;
- (V) object o = 12.45m;
- (V) string titulo = true.ToString();
- (F) A sequencia \t inclui uma tabulação vertical

10- Indique verdadeiro(V) ou falso (F) para as seguintes declarações de variáveis considerando a nomenclatura usada:

- (✓) **string? nome;** é um exemplo de *nullable reference type*;
- (F) para x igual a 0 , **Console.WriteLine(x++);** imprime o valor 1
- (F) A ordem de precedência dos operadores lógicos é : ! , || e && (NOT, OR e AND)
- (✓) para y igual a 0 , **Console.WriteLine(++y);** imprime o valor 1
- (F) **(10 % 2 == 0) ? "Par" : "Impar";** Vai retornar "Impar"
- (F) Para x=25 e y=5 , a expressão **(y >= x) && (y <= x);** retorna true;
- (✓) Os **nullable reference types** emitem um alerta em tempo de compilação para uso do **null**
- (F) Para x = 10 , o código **Console.WriteLine(x+=x);** imprime o valor 10;
- (F) Para x= 5 , a expressão **!(9 >= x) && x <= 7 || x == 5;** retorna false;
- (✓) O operador **?.** permite verificar se um objeto é **null** e evitar o **NullReferenceException**
- (✓) Qualquer tipo de dado pode ser declarado como **anulável** com a ajuda do operador "?"

13- Considere o seguinte trecho de código:

```
int y = 5 ;
y = (y++)+y+(++y);      ou    y = y++ + y + ++y;
Console.WriteLine(y);
```

Retorna o 5 incrementa para o 6
6 vai incrementar para 7 e depois resolver
 $y=(5+6+7)=18$

15 - Escolha a opção que representa a exibição do resultado para o código usando os *operadores de decremento e incremento* (pré e pós) :

```
var numero = 5;
Console.WriteLine(numero++);
numero = 1;
Console.WriteLine(++numero);
numero = 2;
Console.WriteLine(numero--);
numero = 3;
Console.WriteLine(--numero);

Console.ReadKey();
```

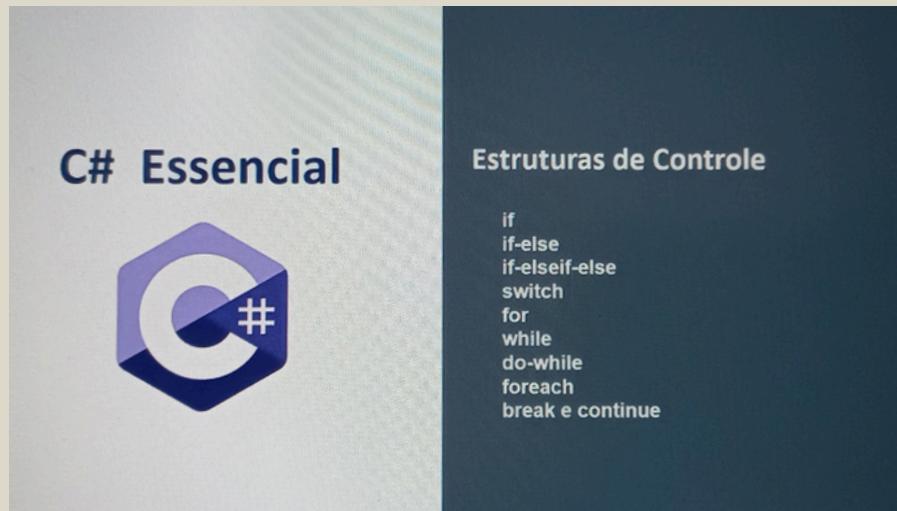
(✓)	()	()	()
5	6	5	6
2	3	2	2
2	2	1	1
2	2	2	3

43: PDFs dos projetos e respostas dos exercícios

06/06

Seção 03: Estrutura de Controle

44: Apresentação



45: Instrução if

The image shows a slide from a C# tutorial. The title '45: Instrução if' is at the top. On the left, there's a logo consisting of a blue hexagon with a white 'C#' symbol inside. The main content area has a light gray background. It features the text 'C# Essencial' and a list of four items under the heading 'Estrutura de Controle : if': 'A instrução if', 'Conceito', 'Sintaxe', and 'Usos'. The right side of the slide has a dark blue sidebar.

C# Essencial

Estrutura de Controle : if

- A instrução if
- Conceito
- Sintaxe
- Usos

Instrução if

Seleciona um bloco ou uma instrução para execução com base no valor de uma expressão booleana (true ou false)

Usamos a instrução if para especificar um bloco de código que deverá ser executado, se uma condição for verdadeira.

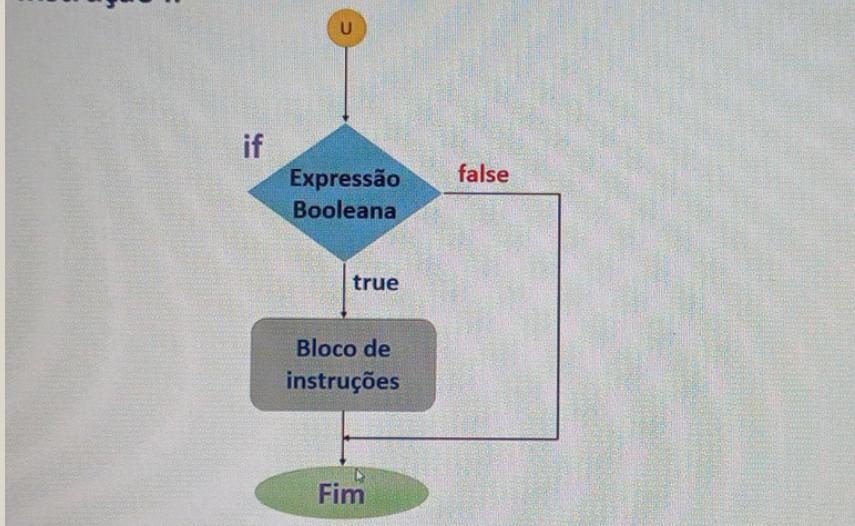
Sintaxe

```
if (expressão booleana)
{
    //executa o bloco de instrução se expressão for true
    //instrução 1
    //instrução 2
}
```

```
if (expressão booleana)
    //executa apenas uma instrução se expressão for true
```

Se o bloco possuir apenas um comando o uso das chaves {} é opcional

Instrução if



Instrução if - Exemplo : Cliente especial tem desconto

```
bool clienteEspecial = false;
Console.WriteLine("Cliente Especial (S/N)\t");
var resposta = Console.ReadLine();

if(resposta == "S")
{
    clienteEspecial = true;
}

if(clienteEspecial)
{
    Console.WriteLine("Desconto de 10%");
}

Console.ReadKey();
```

46: Instrução if - else

The image shows a presentation slide with a light gray background. On the left side, there is a dark blue vertical sidebar containing the title 'Estrutura de Controle : if-else' and four bullet points: 'A instrução if-else', 'Conceito', 'Sintaxe', and 'Usos'. On the right side, there is a large white area with the text 'C# Essencial' and a purple hexagonal logo featuring a white 'C#' symbol.

C# Essencial

Estrutura de Controle : if-else

- A instrução if-else
- Conceito
- Sintaxe
- Usos

Instrução if-else

A instrução if-else é composta por dois blocos de instruções : o bloco de instruções if e o bloco de instruções do else

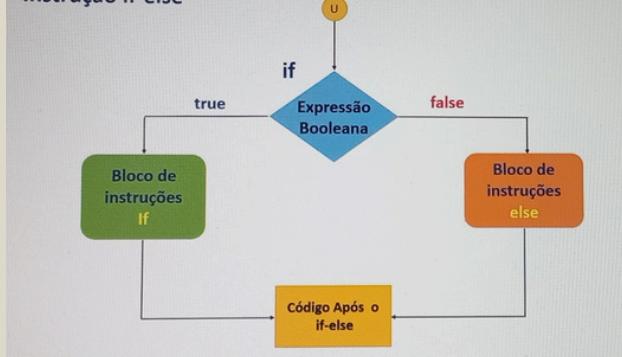
Sintaxe

```
if (expressão booleana)
{
    //executa for true
    //instrução 1
    //instrução 2
}
else
{
    //executa for false
    //instrução 1
    //instrução 2
}
```

As instruções do bloco if serão executadas se a condição definida na expressão booleana for true

As instruções do bloco else serão executadas se a condição definida na expressão booleana for false

Instrução if-else



Instrução if - Exemplo : Aluno aprovado ou reprovado

Informar a NOTA do aluno

SE NOTA É MAIOR que 5 → if (NOTA > 5)
 ALUNO APROVADO → {
 }
 SENÃO → else
 ALUNO REPROVADO → {
 }

```
if (NOTA > 5)
{
    ALUNO APROVADO
}
else
{
    ALUNO REPROVADO
}
```

Instrução if - Exemplo : Aluno aprovado ou reprovado

```
Console.WriteLine("Informe a nota do aluno \t");
var nota = Convert.ToInt32(Console.ReadLine());
```

```
false if( nota > 5) true
{
    Console.WriteLine("Aluno Aprovado");
}
else
{
    Console.WriteLine("Aluno Reprovado");
}
```

Instrução if - Exemplo : Comparar se x é maior, menor ou igual a y

```
Console.Write("\nInforme o valor de x\t");
int x = Convert.ToInt32(Console.ReadLine());

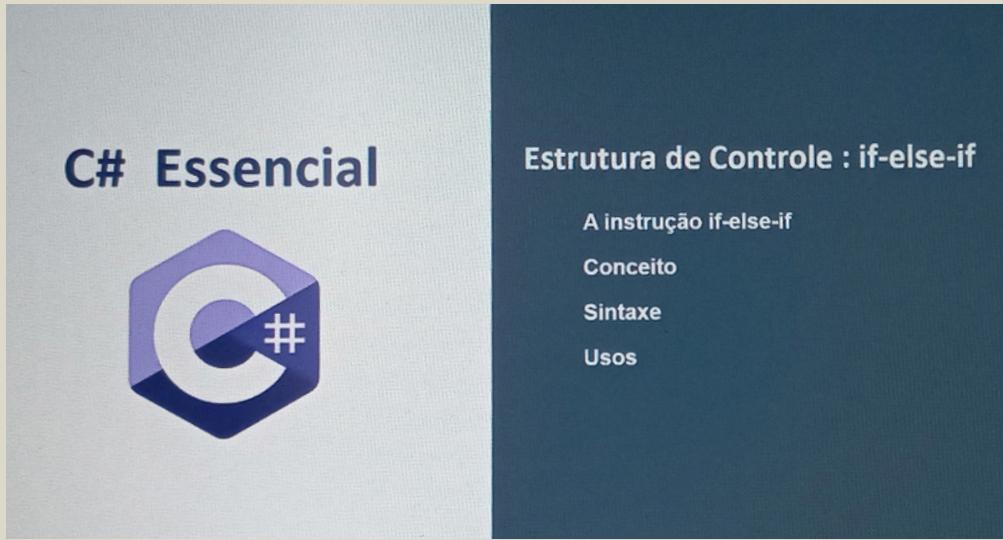
Console.Write("\nInforme o valor de y \t");
int y = Convert.ToInt32(Console.ReadLine());

false if( x > y) true
{
    Console.WriteLine("x é maior que y");
}
else
{
    false if( x < y) true
    {
        Console.WriteLine("x é menor que y");
    }
    else
    {
        Console.WriteLine("x é igual a y");
    }
}

Console.ReadKey();
```

If-else aninhados

47: Instrução if - else - if



C# Essencial

Estrutura de Controle : if-else-if

- A instrução if-else-if
- Conceito
- Sintaxe
- Usos

Instrução if-else-if

Podemos usar instruções else if após a instrução if para avaliar mais de uma condição

As instruções else if somente serão executadas se a condição na instrução if for falsa

Portanto ou a instrução if será executada ou uma das instruções else if será executada, mas não ambas

A instrução if pode ter várias cláusulas else if onde cada cláusula tem uma condição

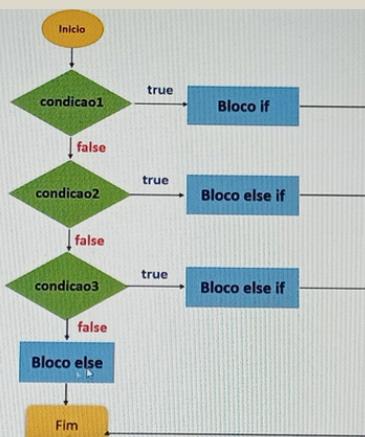
A instrução else if verifica cada condição de cima para baixo sequencialmente, e, se uma condição for verdadeira, o bloco correspondente é executado e as demais condições não serão avaliadas.

Se nenhuma condição for verdadeira, o bloco da cláusula else é executado. (A cláusula else é opcional.)

Instrução if-else-if - sintaxe

```
if (condicao1)
{
    //executa se condicao1 for true
    //instrução
}
else if (condicao2)
{
    //executa se condicao1 for false e condicao2 for true
    //instrução
}
else if (condicao3)
{
    //executa se condicao1 for false e condicao3 for true
    //instrução
}
else
{
    //executa se condicao1, condicao2 e condicao3 for false
    //instrução
}
```

Instrução if-else-if



Instrução else-if - Ex: Aluno reprovado, recuperação, aprovado e aprovado com distinção

Informar a NOTA do aluno

```
SE NOTA MENOR QUE 5 → if (NOTA < 5 )
    {
        ALUNO REPROVADO
    }
SENÃO SE NOTA MAIOR OU IGUAL A 5 E MENOR QUE 6 → else if ( nota >= 5 && nota < 6 )
    {
        ALUNO EM RECUPERAÇÃO
    }
SENÃO SE NOTA MAIOR OU IGUAL A 6 E MENOR OU IGUAL A 9 → else if ( nota >= 6 && nota <= 9 )
    {
        ALUNO APROVADO
    }
SENÃO SE NOTA MAIOR QUE 9 → else if ( nota > 9 )
    {
        ALUNO APROVADO COM DISTINÇÃO
    }
```

48/49: Instrução switch I e II

The image shows a split-screen presentation slide. The left side, titled "C# Essencial", features the C# logo (a purple hexagon with a white "C" and "#"). The right side, titled "Estrutura de Controle : switch", lists concepts: "A instrução switch", "Conceito", "Sintaxe", and "Usos".

C# Essencial	Estrutura de Controle : switch
	A instrução switch
	Conceito
	Sintaxe
	Usos

Instrução switch

O bloco **switch-case** é uma estrutura de condição que define o código a ser executado com base em uma comparação de valores

A instrução **switch** pode ser usada para substituir a instrução **if...else if**

Diferente das instruções **if else if** a instrução **switch** não avalia uma expressão booleana que retorna **true ou false** mas avalia o valor da variável ou expressão da instrução **switch**

Instrução switch - sintaxe

```
switch (variável/expressão)
{
    case valor1:
        // código 1
        break;
    case valor2:
        // código 2
        break;
    default:
        // código 3
        break;
}
```

O valor da variável/expressão na instrução **switch** é comparado com os valores de cada **case**, e, se houver uma correspondência, o bloco de código associado ao **case** é executado

O comando **break** é utilizado para especificar a última linha de código a ser executada dentro do **case**. Se não for declarado, os códigos definidos dentro dos **cases** subsequentes serão executados

O operador **default** é usado para definir um fluxo alternativo para as situações em que o valor contido no **switch** não seja atendido por nenhum dos **cases** especificados. (Equivalente ao bloco **else**)

Instrução switch – avaliando o valor de uma expressão : Par ou Ímpar

```
Console.WriteLine("Informe um número inteiro: \n");
int numero = Convert.ToInt32(Console.ReadLine());

switch (numero % 2)
{
    case 0: Console.WriteLine("\n" + numero + " é par");
        break;

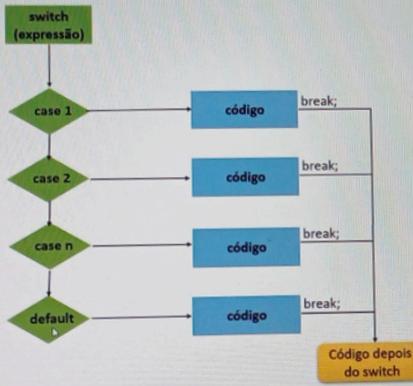
    case 1: Console.WriteLine("\n" + numero + " é ímpar");
        break;
}

Console.WriteLine("\nFim do processamento...");
Console.ReadKey();
```

Calcula o valor da expressão e o utiliza para verificar cada case

Par = 2 * n
Ímpar = 2 * n + 1

switch



Instrução switch – Número de prestações

Informar o número de prestações (1 a n)

AVALIA o valor informado

CASO VALOR IGUAL A 1

PRESTAÇÃO 300
SAI

CASO VALOR IGUAL A 2

PRESTAÇÃO 150
SAI

CASO VALOR IGUAL A 3

PRESTAÇÃO 100
SAI

PADRÃO

VALOR INVÁLIDO
SAI

```
switch (VALOR)
{
    case 1:
        Prestacao R$ 300
        break;
    case 2:
        Prestacao R$ 150
        break;
    case 3:
        Prestacao R$ 100
        break;
    default:
        Valor Inválido
        break;
}
```

Instrução switch – aninhadas

```
switch (expressão)
{
    case 1:
        // código
        break;
    case 2:
        // código
        switch (expressão2)
        {
            case 1:
                // código
                break;
            case 2:
                // código
                break;
        }
        break;
    default:
        // código
        break;
}
```

Instrução switch dentro de outra instrução switch

Instrução switch – aninhadas

```
int cargo = 0;
int funcao = 0;
Console.WriteLine("Você é Gerente(1) ou Programador(2) ?\t");
cargo = Convert.ToInt32(Console.ReadLine());

if (cargo == 2)
{
    Console.WriteLine("Você é Junior(1) ou Senior(2) ?\t");
    funcao = Convert.ToInt32(Console.ReadLine());
}

switch (cargo)
{
    case 1:
        Console.WriteLine("\nBem-Vindo Gerente.");
        break;
    case 2:
        Console.WriteLine("\nBem-Vindo Programador.");
        switch (funcao)
        {
            case 1:
                Console.WriteLine("\nVocê é Junior.");
                break;
            case 2:
                Console.WriteLine("\nVocê é Senior.");
                break;
            default:
                Console.WriteLine("\nFunção desconhecida");
                break;
        }
        break;
    default:
        Console.WriteLine("\nNão consigo te identificar");
        break;
}

Console.WriteLine("\nFim do processamento...");
Console.ReadKey();
```

50: Estrutura de repetição

The image shows a screenshot of a presentation slide. On the left, there is a light gray sidebar with the text "C# Essencial" and the Microsoft C# logo (a purple hexagon with a white "C#" inside). The main content area has a dark blue background. At the top, it says "Estrutura de repetição". Below that, there is a list of four items: "A instrução goto e label", "Conceito", "Sintaxe", and "Usos".

C# Essencial

Estrutura de repetição

- A instrução goto e label
- Conceito
- Sintaxe
- Usos

As estruturas de repetição são usadas para repetir instruções ou blocos de código.

A decisão de repetir o código é baseada na avaliação de uma expressão lógica. Se a expressão for verdadeira, o código é executado

Uma estrutura de repetição permite especificar que uma ação seja repetida várias vezes, dependendo do valor de uma condição.

Estrutura de repetição : Instrução goto e label

A instrução **goto** pode ser usada para *transferir o controle de uma parte para outra parte* do programa com a ajuda de um identificador chamado de **label**

Podemos usar a instrução **goto** para criar uma estrutura de repetição :

- Definimos uma **label** que é usada como um identificador
- A seguir definimos o código que desejamos repetir após a **label**
- Definimos uma condição que será avaliada para continuar repetindo ou não o código
- Usando a instrução **goto** transferimos a execução do código para a **label** repetindo o código

Estrutura de repetição : Instrução goto e label

label: ←
 ↳
 //código a ser executado

condicao

goto label; ——————

Podemos também fazer assim:
condicao
 goto label;
 ...
label:
 //instrução

Estrutura de repetição : Instrução goto e label

Imprimir o valor de uma variável i iniciando com o valor 1 e incrementando o seu valor até 10

```
int i = 1;  
repetir:  
    Console.WriteLine($"i = {i} ");  
    i++;  
    if (i <= 10)  
        goto repetir;  
Console.WriteLine("Fim do processamento...");  
Console.ReadKey();
```

Laço
loop

Estruturas de repetição

while
do while
for
for each
break
continue

51/52: Instrução while I e II

The image shows a slide from a presentation. On the left, there is a light gray background with the text "C# Essencial" in dark blue and a large purple hexagonal logo containing a white "C#" symbol. On the right, there is a dark blue background with the text "Instrução while" in white, followed by three white bullet points: "Conceito", "Sintaxe", and "Uso".

C# Essencial

Instrução while

- Conceito
- Sintaxe
- Uso

Instrução while

A instrução while executa uma instrução ou um bloco de instruções enquanto uma expressão booleana especificada for avaliada como true .

Como essa expressão é avaliada antes de cada execução do loop, um loop while pode executar zero ou mais vezes

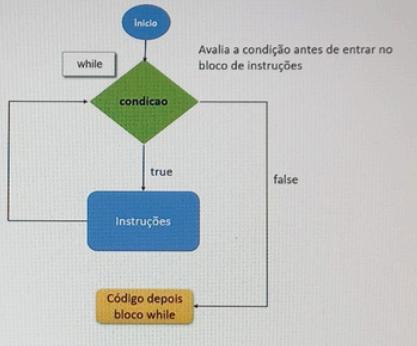
Instrução while - syntaxe

```
while(condição)
{
    //bloco de código
}
```

O loop/laço inicia com a instrução while e deve incluir uma expressão condicional booleana (*condição*) que retorna true ou false .

O bloco de código será executado enquanto a condição for true

while



Instrução while – Imprimir o valor de i de 1 a 10 no console

```
var i = 1;
while (i <=10) —> Enquanto i for menor ou igual a 10 repita a execução do bloco de código
{
    Console.WriteLine($"i = {i}");
    i++;
}
```

Cuidado com o laço ou loop infinito !!!

Instrução while – Imprimir o valor de i de 1 a 10 no console

```
var i = 10;
while (i > 0)
{
    Console.WriteLine($"i = {i}");
    i--;
}
```

Outra abordagem !!!

Instrução while – Imprimir a tabuada de um número maior que zero

- Declarar e inicializar as variáveis
- Solicitar a entrada do número
- Verificar se o número é maior que zero
- Se o número for maior que zero
 - criar um laço while para imprimir a tabuada do número definindo a condição de execução do loop
 - Se o número for menor ou igual a zero
 - Encerrar o processamento

Instrução while – Imprimir a tabuada de um número maior que zero

- Declarar e inicializar as variáveis
- Solicitar a entrada do número
- Verificar se o número é maior que zero
- Se o número for maior que zero
 - criar um laço while para imprimir a tabuada do número definindo a condição de execução do loop
 - Se o número for menor ou igual a zero
 - Encerrar o processamento

Tabuada do 8

8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80

Instrução while – Saindo do loop com break

A instrução break pode ser usada para encerrar uma instrução switch ou um loop (while, for, if, etc.) em uma determinada condição.

```
while (condição1)
{
    if(condição2) ————— Quando condição2 for true (Condição de saída)
    {
        break; ————— A instrução break será acionada e a execução do loop
    }
    instruções;
}
```

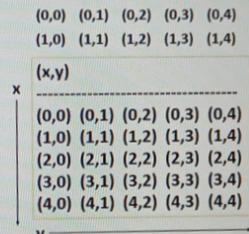
Instrução while aninhadas

```
while (condição)
{
    instruções;
}
```

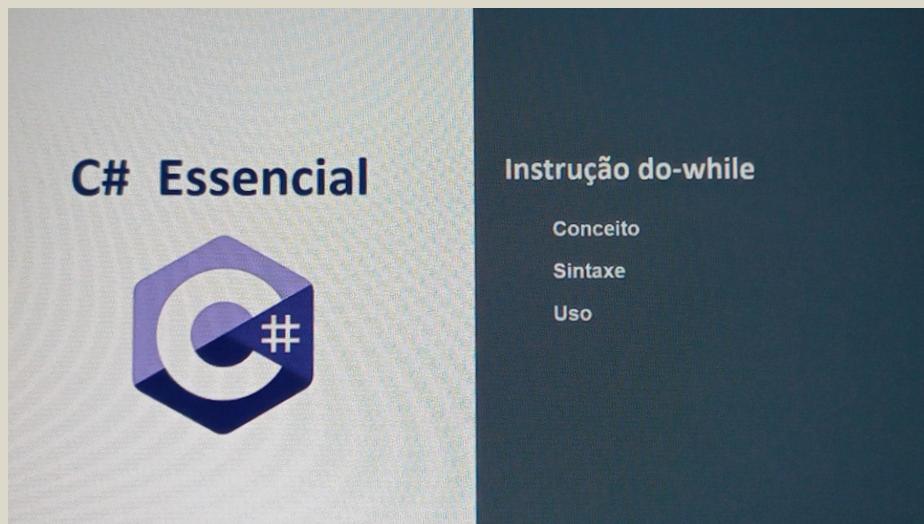
```
while (condição1)
{
    while (condição2)
    {
        instruções;
    }
    instruções;
}
```

Instrução while aninhadas : Exemplo coordenadas do plano (x,y)

```
int x=0;
while (x<5)
{
    int y=0;
    while (y<5)
    {
        Console.WriteLine($"{(x},{y}) ");
        y++;
    }
    x++;
    Console.WriteLine();
}
```



53: Instrução do-while



Instrução do-while

O loop do-while é uma variante do loop while e possui o mesmo comportamento

A diferença é que o loop do-while executa o bloco de código pelo menos uma vez, antes de avaliar a condição, e a seguir repete o loop enquanto a condição for verdade/true

Instrução do-while : sintaxe

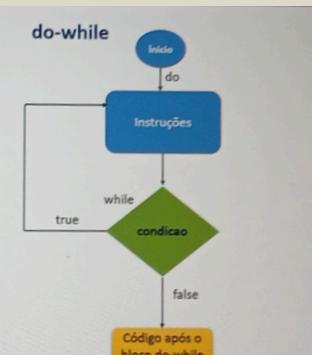
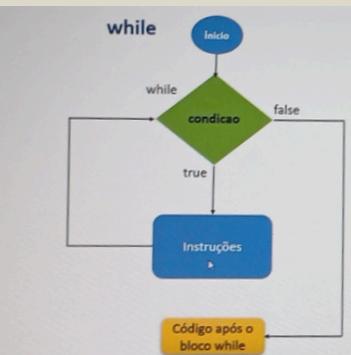
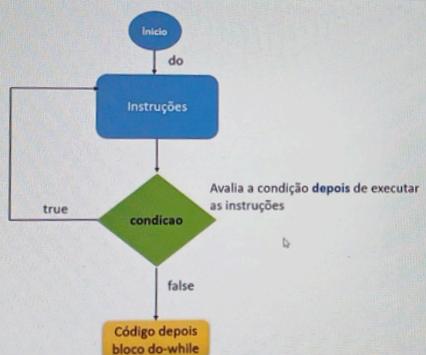
```
do
{
    //bloco de código
}
while(condição);
```

O loop inicia com a instrução do seguida por um bloco de código que será executado e a seguir avalia uma expressão booleana com a instrução while

A execução será interrompida quando uma condição booleana for avaliada como falsa/false

Note que a expressão condicional aparece no final do loop, logo, a(s) instrução(ões) no loop são executadas pelo menos uma vez antes que a condição seja testada

do-while



Loop do-while – instrução break

- Imprimir o valor da variável i de 1 a 10 e sair do loop quando i for maior que 7

```
var i = 1;
do
{
    Console.WriteLine($"i = {i}");
    i++;
    if( i > 7)           Condição de saída do loop do-while
        break;
}
while (i <=10);
```

Sai do loop

Loop infinito com do-while e com while

```
do
{
    //bloco de código
}
while(true);
```

while(true)
{
 //bloco de código
}

Para sair do loop infinito é necessário definir a condição de saída e usar a instrução break para interromper o loop

CTRL+C —> Força a interrupção do programa em loop infinito

Instrução do-while aninhadas

```
do
{
    instruções;
}
while (condição);
```

do-while aninhados

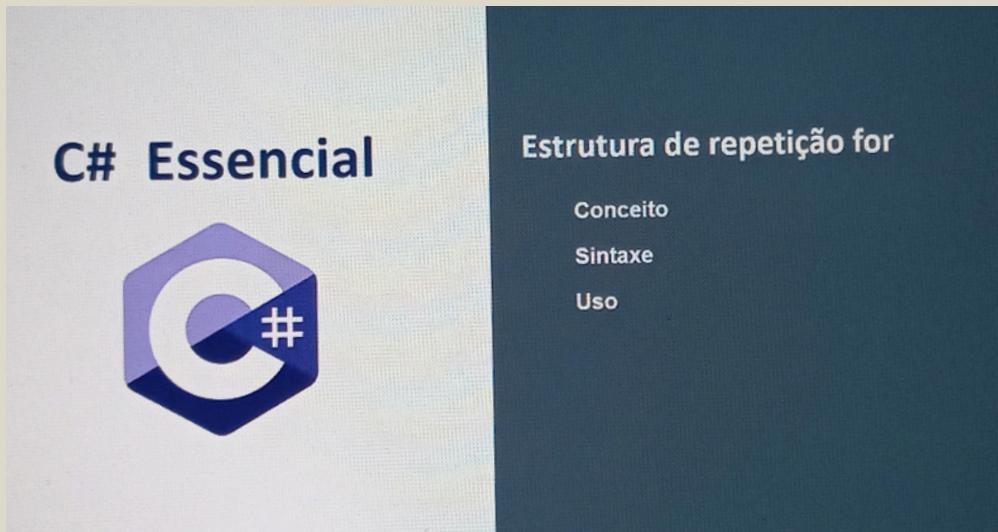
```
do
{
    do
    {
        instruções;
    }
    while (condição2);
    instruções;
}
while (condição1);
```

Instrução while aninhadas : Exemplo: exibir coordenadas do plano (x,y)

```
int x=0;
do
{
    int y=0;
    do
    {
        Console.WriteLine($"{(x},{y}) ");
        y++;
    }
    while (y<5);
    x++;
    Console.WriteLine();
}
while(x<5);
```

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(x,y)				
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

54/55: Instrução for I e II



loop for

Executa um bloco de instruções repetidamente até que a condição especificada retorne false.

```
for ( inicialização; condição; iteração )
{
    //instruções
}
```

Inicialização: É usado para declarar e inicializar uma variável de controle que será local e não pode ser acessada fora do loop. É executada primeiro e apenas uma vez.

Condição: É uma expressão booleana que retorna true ou false. Se a expressão for avaliada como true, ela executará o loop novamente; caso contrário, o loop é encerrado.

Iteração: É usado para incrementar ou decrementar a variável de controle do loop.

loop for

```
int i = 1;
while( i<=10 )
{
    Console.WriteLine($"i = {i}");
    i++;
}

for ( int i = 1 ; i <=10 ; i++ )
{
    Console.WriteLine($"i = {i}");
}
```

loop for

```
for ( inicializador; condição; iterador )
{
    //instruções
}
```

- A instrução de **inicialização** é executada primeiro e apenas uma vez, declarando e atribuindo o valor para a variável de controle
- A condição é avaliada
- Se a condição for true
 - As instruções dentro do loop são executadas
 - A instrução do **iterador** é executada e altera o valor da variável de controle
 - A condição é avaliada novamente
 - O processo é repetido até que a condição seja false
- Se a condição for false o loop é encerrado

loop for

```
int i = 1;
while( i<=10 )
{
    Console.WriteLine($"i = {i}");
    i++;
}

for ( inicio ; condição ; incremento )
{
    //instruções
}
```

inicio
while(condição)
{
 //instruções
 incremento

Instrução for

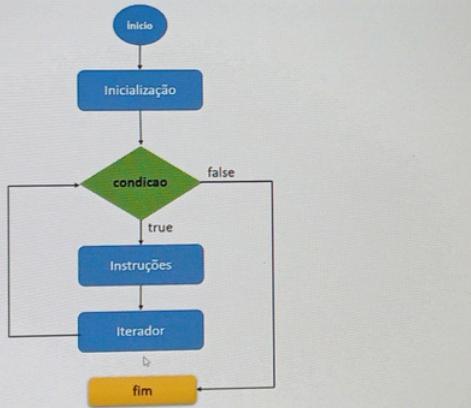
Criar um programa C# usando o laço for para exibir a tabuada de multiplicação de número inteiros

```
int resultado,numero;
Console.Write("Informe o número inteiro : \t");
numero = Convert.ToInt32(Console.ReadLine());
Iniciador: int i=1;
Condição : i<=10;
Iterador : i++

for ( int i = 1; i <= 10; i++ )
{
    resultado = numero * i;
    Console.WriteLine(numero + " X " + i + " = " + resultado);
}

Console.WriteLine("\nFim do processamento...");
Console.ReadKey();
```

loop for



Instrução for – Múltiplas expressões

Podemos usar várias expressões dentro de um loop for. Isso significa que podemos ter mais de uma *instrução de inicialização* e/ou iterador dentro de um loop for.

```
for ( int i = 0, j = 0; i+j <= 10; i++, j++ )
{
    Console.WriteLine($"i = {i} e j = {j}");
}
```

Inicialização	Condição	Iterador
int i = 0, j = 0;	i + j <= 10;	i++, j++

Tanto i como j são incrementados em cada iteração

Instrução for – Seções opcionais

A inicialização, a condição e a instrução do iterador são opcionais em um loop for e assim podemos executar um loop for sem essas instruções.

1- Loop for sem definir a inicialização e a instrução de iteração

```
int i = 1;
for ( ; ; i <= 5 )
{
    Console.WriteLine($"Loop for: Iteração {i}");
    i++;
}
```

2- Loop nenhum a seção (loop infinito)

```
int i = 1;
for ( ; ; i++ )
{
    Console.WriteLine($"Loop for: Iteração {i}");
    i++;
}
```

loop for aninhados

Da mesma forma que os *loop while* e *do-while* também podemos ter loop for aninhados

```
for ( inicialização; condição; iteração )
{
    for (inicialização; condição; iteração)
    {
        instruções;
    }
    instruções;
}
```

```
while (condição)
{
    for (inicialização; condição; iteração)
    {
        instruções;
    }
    instruções;
}
```

56: Instrução break e continue

The image shows a screenshot of a presentation slide. On the left, there is a light gray sidebar with the text "C# Essencial" and a large blue hexagonal logo containing a white "C#" symbol. The main content area has a dark blue background. At the top, it says "Instruções break e continue". Below that is a vertical list of three items: "Conceito", "Sintaxe", and "Uso".

C# Essencial

Instruções break e continue

Conceito

Sintaxe

Uso

57: Lista de exercício

C# Essencial



Exercícios

Estruturas de controle e
Repetição

58: Projeto das aulas e respostas dos exercícios

06/06

Seção 04: Classes e métodos

Estudo C# Código

