**Linguagem de programação C#**

**Lucas Matheus Casarotti Rodrigues**

**Sumário**

[1. Introdução 4](#_Toc136523283)

[1.1. O que é C#? 5](#_Toc136523284)

[1.2. O que é uma linguagem de programação de alto nível? 5](#_Toc136523285)

[1.3. O que é uma linguagem de programação de baixo nível? 5](#_Toc136523286)

[1.4. O que é linguagem de programação compilada? 5](#_Toc136523287)

[1.5. O que é linguagem de programação interpretada? 5](#_Toc136523288)

[1.6. O que é tipagem de dados? 5](#_Toc136523289)

[1.7. Porque utilizar a linguagem de programação C#? 5](#_Toc136523290)

[1.8. Como funciona a linguagem de programação C#? 5](#_Toc136523291)

[2. Escopo de um programa em C# 4](#_Toc136523292)

[2.1. Definições 5](#_Toc136523293)

[2.2. O que é um namespace? 5](#_Toc136523294)

[2.3. O que é Using? 5](#_Toc136523295)

[2.4. Resumindo a execução do escopo 5](#_Toc136523296)

[3. Criando o nosso primeiro programa em C# 4](#_Toc136523297)

[3.1. Primeiro programa em c# 5](#_Toc136523298)

[4. C# Fundamental 4](#_Toc136523299)

[4.1. Variáveis e tipos de dados 4](#_Toc136523300)

[4.2. Constantes 5](#_Toc136523301)

[4.3. Nomes reservados 5](#_Toc136523302)

[4.4. Var 6](#_Toc136523303)

[4.5. Object 6](#_Toc136523304)

[4.6. Nullable Types 7](#_Toc136523305)

[4.7. Conversão de dados implícita 7](#_Toc136523306)

[4.8. Conversão de dados explicita 7](#_Toc136523307)

[4.9. Parse 8](#_Toc136523308)

[4.10. Convert 8](#_Toc136523309)

[4.11. Operadores aritméticos 9](#_Toc136523310)

[4.12. Operadores de atribuição 10](#_Toc136523311)

[4.13. Operadores de comparação 10](#_Toc136523312)

[4.14. Operadores lógicos 10](#_Toc136523313)

[4.15. Estrutura de condição IF e ELSE 11](#_Toc136523314)

[4.16. Estrutura de condição Switch Case 11](#_Toc136523315)

[4.17. Estrutura de repetição for 12](#_Toc136523316)

[4.18. Estrutura de repetição while 12](#_Toc136523317)

[4.19. Estrutura de repetição do-while 13](#_Toc136523318)

[4.20. Métodos e funções 13](#_Toc136523319)

[4.21. Value Types e Reference Types 16](#_Toc136523320)

# Introdução

## O que é C#?

É uma linguagem de programação de alto nível, compilada, de tipagem forte, desenvolvida pela Microsoft como parte da plataforma .NET.

## O que é uma linguagem de programação de alto nível?

A linguagem de programação de alto nível é mais próxima do programador do que do dispositivo, assemelhando-se mais à linguagem humana. Seu objetivo é facilitar a programação, sendo mais intuitiva e fácil de aprender.

## O que é uma linguagem de programação de baixo nível?

A linguagem de programação de baixo nível é mais voltada para o dispositivo, e tem como principal objetivo se comunicar com o computador de maneira mais rápida, ao contrário da linguagem de alto nível.

## O que é linguagem de programação compilada?

A linguagem compilada é uma linguagem de programação em que o código fonte é executado diretamente pelo sistema operacional ou pelo processador, após ser traduzido por meio de um processo chamado compilação, usando um programa de computador chamado compilador, para uma linguagem de baixo nível, como linguagem de montagem ou código de máquina.

## O que é linguagem de programação interpretada?

Linguagem interpretada é uma linguagem de programação em que o código fonte é executado por um programa de computador chamado interpretador, que por sua vez é executado pelo sistema operacional ou processador. Mesmo que um código em uma linguagem passe pelo processo de compilação, a linguagem ainda pode ser considerada interpretada se o programa resultante não for executado diretamente pelo sistema operacional ou processador.

## O que é tipagem de dados?

A linguagem de programação C# é uma linguagem de tipagem estática, o que significa que é obrigatório especificar o tipo de dado durante a declaração de variáveis, parâmetros de função e retornos de função. Isso permite um alto grau de verificação de tipo em tempo de compilação, o que ajuda a evitar erros de tipo e torna o código mais robusto e seguro.

Exemplo:

INT idade = declarando uma variável chamada idade do tipo numérico que aceita apenas números inteiros.

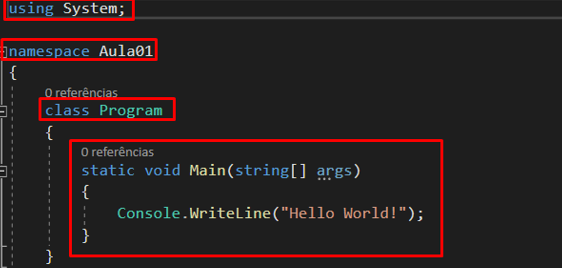
## Porque utilizar a linguagem de programação C#?

Porque é uma linguagem de programação completa, além de ser utilizada para o desenvolvimento web e mobile, pode ser utilizada também para o desenvolvimento de jogos, entre outros. Além disso, é uma tecnologia que está no mercado há um bom tempo, o que pode ser uma grande vantagem, pois consequentemente possui um amplo mercado de trabalho e muitos materiais de estudo disponíveis na internet.

## Como funciona a linguagem de programação C#?

Ela é uma linguagem de programação tipada, compilada e de código gerenciado. Sua execução depende de um gerenciador chamado "Runtime", que é responsável por limpar e alocar a memória, além de determinar como as informações são alocadas na memória. Dessa forma, podemos nos concentrar mais no desenvolvimento da linguagem em si, em vez de nos preocuparmos com os recursos de máquina.

# Escopo de um programa em C#



## Definições

* Importações
* Namespace
* Classe
* Método principal

## O que é um namespace?

Namespaces são divisões lógicas em uma linguagem de programação, onde não é permitido ter duas classes com o mesmo nome em um único namespace. Além disso, um namespace pode ser utilizado em diversos arquivos.

## O que é Using?

O 'using' é uma diretiva em algumas linguagens de programação que permite importar bibliotecas de código externas para serem utilizadas em um arquivo. Outros namespaces, sejam eles internos ou externos, podem ser importados usando o 'using'. Por padrão, apenas as bibliotecas básicas são incluídas, e outras bibliotecas precisam ser importadas explicitamente.

## Resumindo a execução do escopo

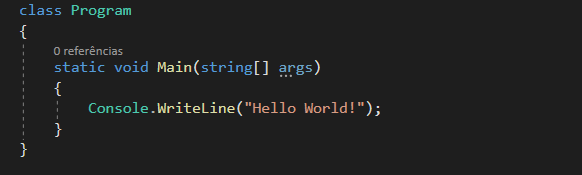
O arquivo 'Program.cs' contém a classe 'Program' e o método 'Main' (principal) em muitos projetos em C# e em outras linguagens de programação. O método 'Main' é o ponto de entrada do programa, onde a execução começa, e é geralmente o primeiro método a ser chamado quando um programa é executado. É responsável por iniciar a lógica do programa e chamar outros métodos e classes necessários para a execução correta do programa.

# Criando o nosso primeiro programa em C#

## Primeiro programa em c#

Neste primeiro programa em C#, utilizaremos o comando Console.WriteLine("Hello World"); para exibir a mensagem 'Hello World'. O método Console.WriteLine é usado para escrever uma linha de texto na saída padrão do console, que pode ser exibida no terminal ou na janela de saída de um ambiente de desenvolvimento integrado (IDE). Neste caso, ele imprimirá a mensagem 'Hello World' na saída padrão do console quando o programa for executado.

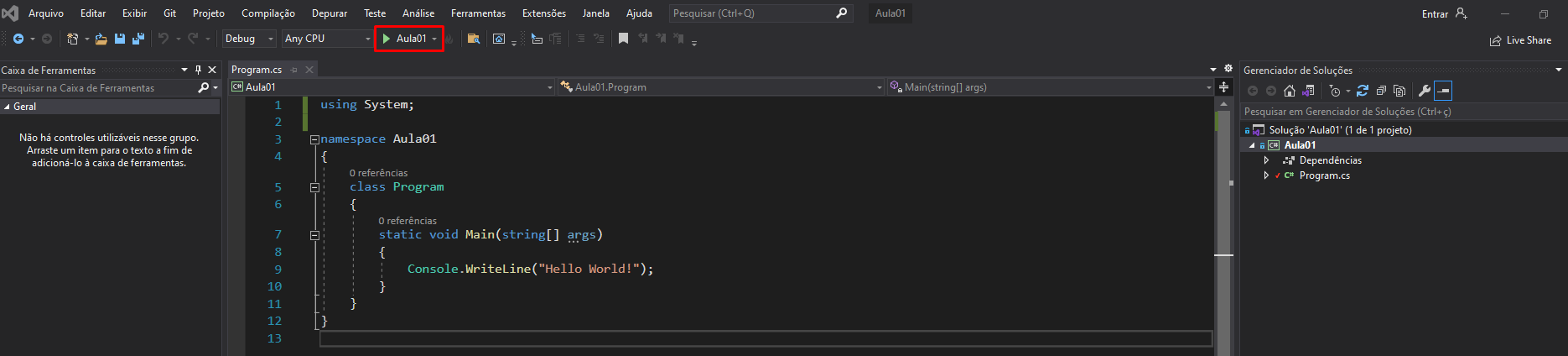
Exemplo:



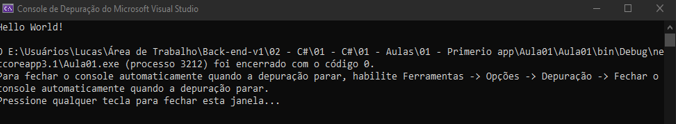
* 1. **Como executar o programa?**

Para executar o programa, é necessário clicar no botão verde que fica localizado na parte superior da IDE.

Exemplo:



Resultado:

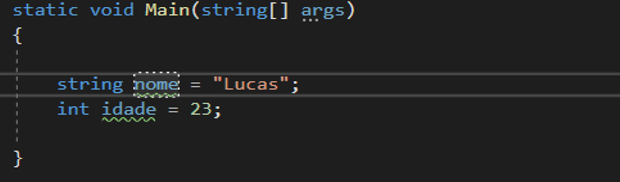


# C# Fundamental

## Variáveis e tipos de dados

Variáveis são espaços na memória utilizados para armazenar valores. Como o próprio nome sugere, elas são variáveis, ou seja, seu valor pode ser alterado a qualquer momento. Sempre que criarmos uma variável, é necessário informar o seu tipo.

Exemplo:



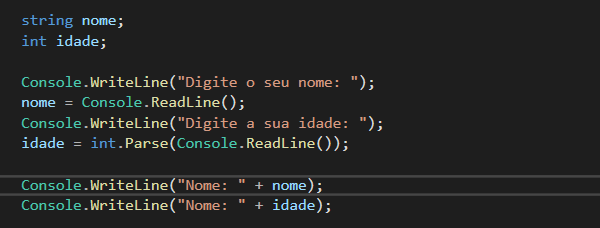
Neste outro exemplo iremos criar duas variáveis, uma do tipo string e outra do tipo inteiro para armazenar os dados que o usuário digitar.

Resumindo:

O comando “Console.WriteLine(“Mensagem”);” é utilizado para exibir a mensagem.

O comando “Console.RedLine();” é utilizado para ler o que foi digitado.

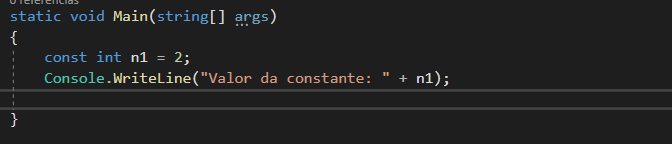
Obs: Observe que para a variável do tipo inteiro foi utilizado o comando “int.Parse(Console.RedLine());”, o motivo pelo qual foi utilizado esse comando é que por padrão o valor digitado pelo usuário é do tipo string, então para atribuir o valor a uma variável do tipo inteiro é necessário utilizar o comando “Parse” para realizar a conversão.



## Constantes

Também utilizamos constantes para armazenar valores. No entanto, ao contrário das variáveis, o valor atribuído a uma constante não pode ser modificado após a sua atribuição.

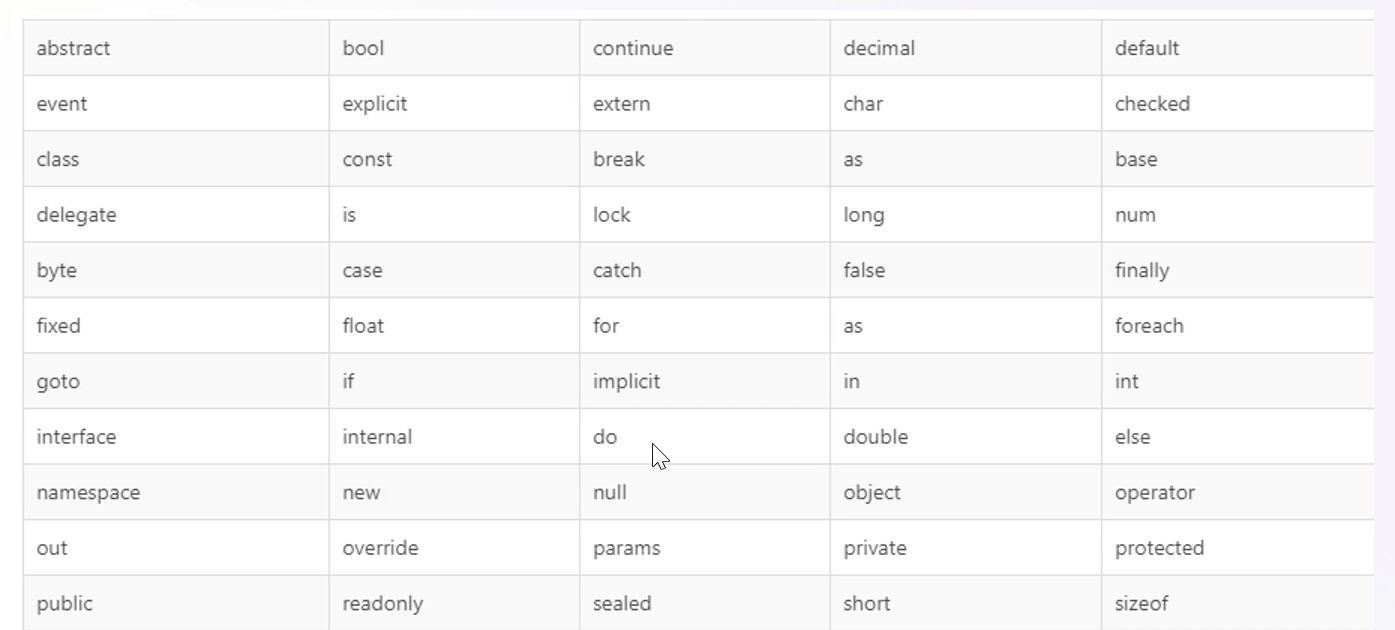
Exemplo:



## Nomes reservados

Também conhecidas como palavras-chave, são palavras exclusivas do C# que não podem ser utilizadas como nomes de variáveis.

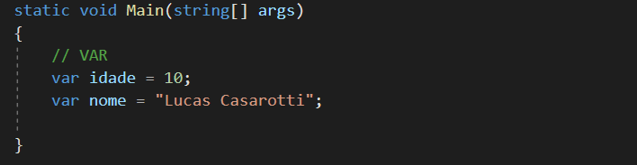
Exemplo:



## Var

"var" é utilizado para permitir que o compilador do C# infira automaticamente o tipo de dado de uma variável com base no valor atribuído a ela. Ou seja, o tipo da variável é determinado automaticamente pelo compilador com base no valor inicial atribuído, tornando desnecessário especificar explicitamente o tipo.

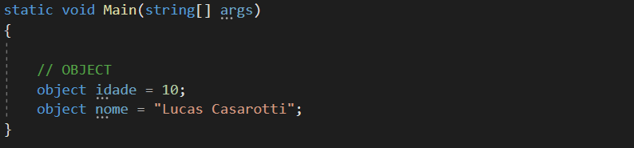
Exemplo:



## Object

O tipo de dado "object" em C# é um tipo de dado que representa um objeto genérico, ou seja, pode armazenar qualquer tipo de valor. É utilizado quando não se sabe previamente o tipo de dado que será armazenado em uma variável e pode ser usado como um tipo de dado genérico. No entanto, é importante ter em mente que o uso excessivo de "object" pode levar à perda de informações de tipo em tempo de compilação, o que pode resultar em erros de tempo de execução e dificultar a manutenção do código. Além disso, como mencionado, o "object" não possui Intelisense, que é uma funcionalidade de autocompletar e fornecer informações sobre tipos de dados disponível em alguns ambientes de desenvolvimento integrado (IDEs), o que pode tornar a codificação menos eficiente. Portanto, é recomendado o uso cuidadoso do tipo de dado "object" e preferir tipos de dados mais específicos sempre que possível.

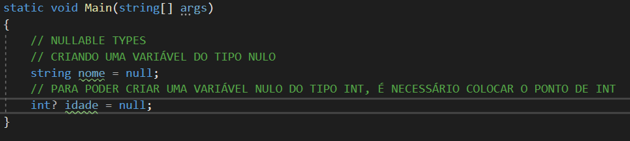
Exemplo:



## Nullable Types

Significa vazio, nulo, é diferente de zero ou uma string vazia, todo tipo de dado primitivo ou complexo pode receber o valor null.

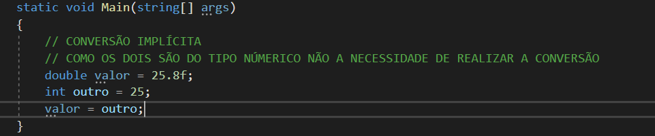
Exemplo:



## Conversão de dados implícita

A conversão implícita em C# ocorre quando os tipos são compatíveis e a conversão pode ser realizada automaticamente pelo compilador, sem a necessidade de uma sintaxe de "casting" explícito. Por exemplo, quando se atribui um valor de um tipo menor para um tipo maior, como int para double, não é necessário realizar uma conversão explícita, pois a conversão é feita automaticamente pelo compilador. Isso é chamado de conversão implícita, pois é feita de forma transparente e sem a necessidade de intervenção do programador. No entanto, é importante ter cuidado ao realizar conversões implícitas, pois pode haver perda de precisão ou outros efeitos indesejados, especialmente quando se trata de conversões entre tipos de dados com diferentes faixas de valores.

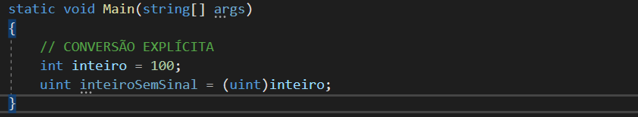
Exemplo:



## Conversão de dados explicita

A conversão explícita em C# ocorre quando os tipos não são diretamente compatíveis e é necessária uma conversão mais específica. Isso é feito usando uma sintaxe de "casting", que envolve o uso do tipo de dado desejado entre parênteses antes da atribuição. Por exemplo, para converter um valor de ponto flutuante em um inteiro, pode-se usar a sintaxe de casting da seguinte forma: (int)valorFloat, onde (int)indica a conversão explicitamente para o tipo inteiro. É importante observar que a conversão explícita pode resultar em perda de dados se os tipos não forem compatíveis, e deve ser usado com cuidado.

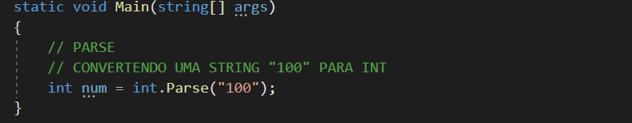
Exemplo:



## Parse

Essa extensão é usada para converter um caractere ou uma string para qualquer tipo específico.

Exemplo:

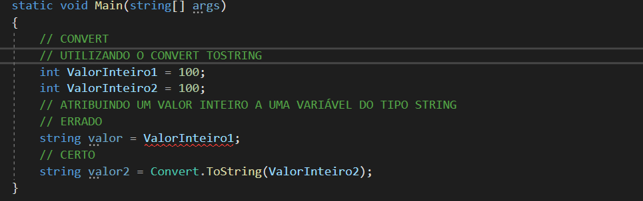


## Convert

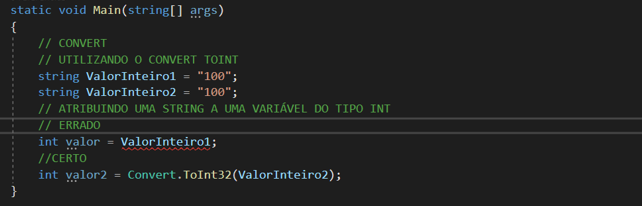
Similar ao parse visto anteriormente, porém permite converter vários tipos de valores. Devemos informar o tipo na chamada da conversão.

Exemplo:

Convertendo inteiro para string



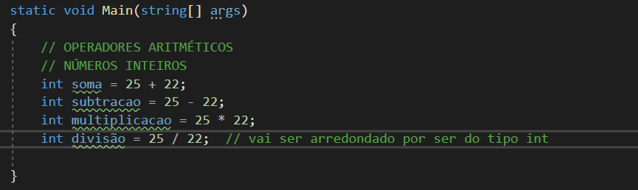
Convertendo string para inteiro



## Operadores aritméticos

São operadores matemáticos dentro do c#, como soma, subtração, divisão e multiplicação.

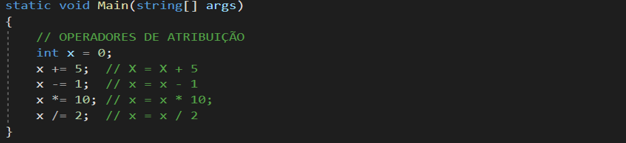
Exemplo:



## Operadores de atribuição

É utilizado quando queremos atribuir um valor a uma variável.

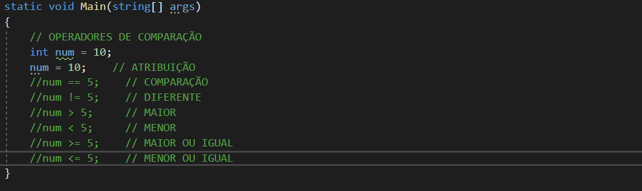
Exemplo:

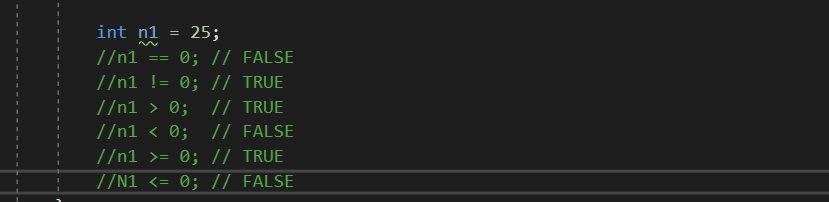


## Operadores de comparação

É utilizado para comparar dados entre si, como números, strings, bytes, etc. A comparação sempre retorna um valor booleano, verdadeiro (True) ou falso (False).

Exemplo:

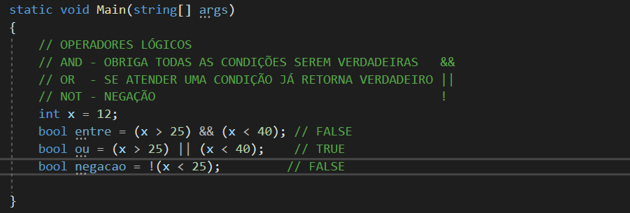




## Operadores lógicos

Os operadores condicionais são utilizados para executar instruções com base em condições, retornando um valor booleano, verdadeiro (True) ou falso (False).

Exemplo:



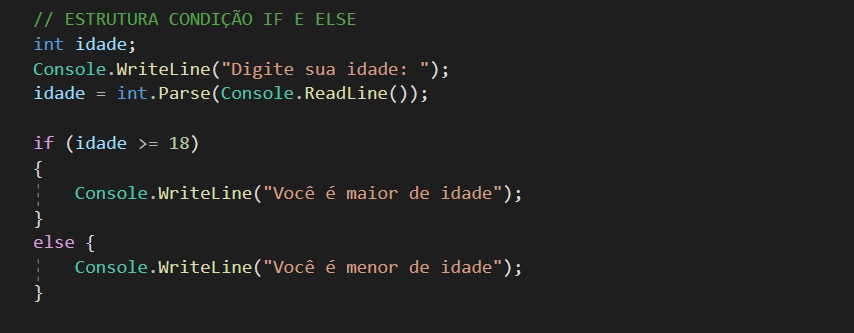
## Estrutura de condição IF e ELSE

A estrutura de condição é utilizada para verificar uma ou mais condições e executar um bloco de código se a condição for verdadeira (True) e outro bloco de código se a condição for falsa (False).

Exemplo:

Se a condição de que a variável idade for maior ou igual a 18 for verdadeira, retorna True e mostra a mensagem "Você é maior de idade". Caso contrário, retorna False e exibe a mensagem "Você é menor de idade".

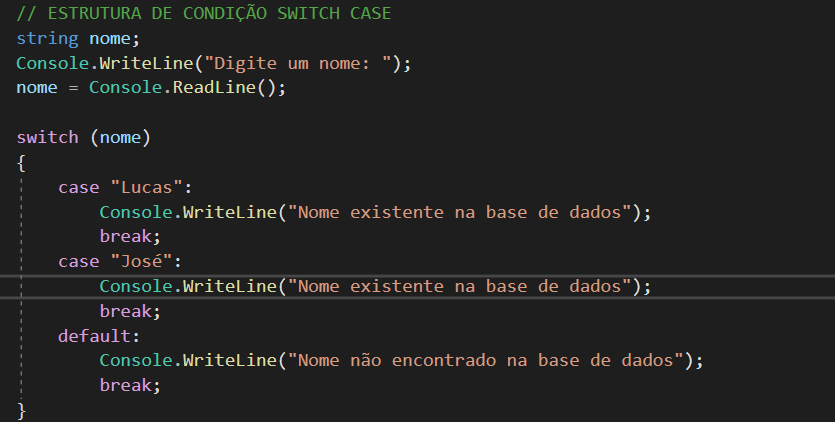
Exemplo:



## Estrutura de condição Switch Case

É uma forma de reduzir a complexidade de vários if e else encadeados. É muito utilizado, principalmente em estruturas de menu. O conteúdo de uma variável é comparado com um valor constante, e caso a comparação seja verdadeira, um determinado comando é executado.

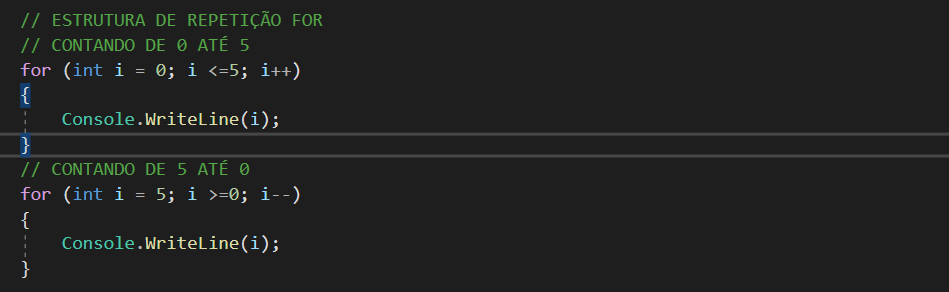
Exemplo:

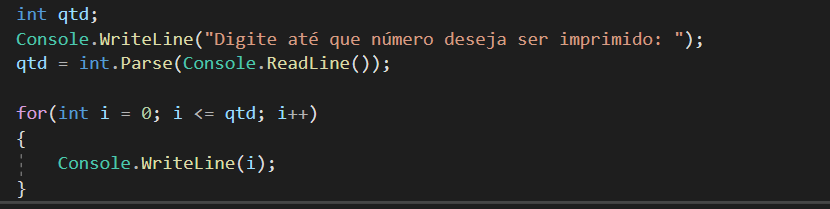


## Estrutura de repetição for

Na tradução “para”, o laço de repetição "for" é muito utilizado em programação para executar uma determinada ação várias vezes. Ele é definido por três parâmetros: o contador, a condição de repetição e o incremento.

Exemplo:

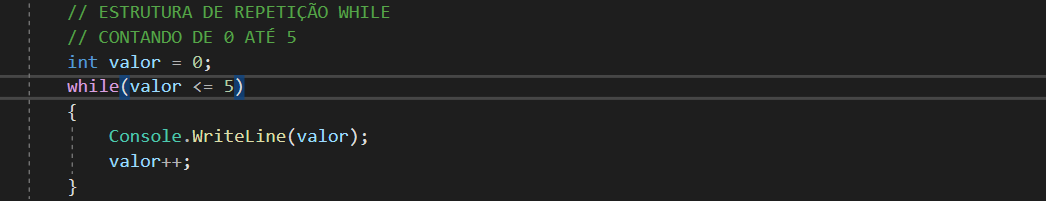




## Estrutura de repetição while

Na tradução "enquanto", o laço de repetição "while" é utilizado para executar um bloco de código enquanto uma determinada condição for verdadeira. O código será executado repetidamente até que a condição se torne falsa.

Exemplo:

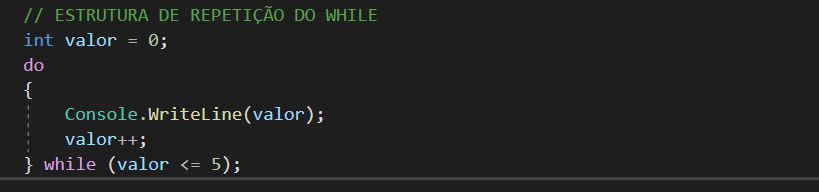


Resumidamente, no começo do loop "while", a condição é definida como "enquanto a variável 'valor' for menor ou igual a 5", ou seja, a condição é verdadeira (true). A cada iteração do loop, o valor da variável 'valor' é incrementado em 1. Após executar o loop 6 vezes consecutivas, o valor da variável 'valor' será maior do que 5, tornando a condição falsa (false) e assim interrompendo a execução do loop "while".

## Estrutura de repetição do-while

Na tradução "Faça", o loop "do-while" é utilizado para executar uma ação enquanto uma condição seja verdadeira, porém a condição é verificada após a execução do código, ao contrário do "while" que verifica a condição no início. Isso significa que, mesmo que a condição seja falsa, o código dentro do "do-while" será executado pelo menos uma vez antes de verificar a condição.

Exemplo:



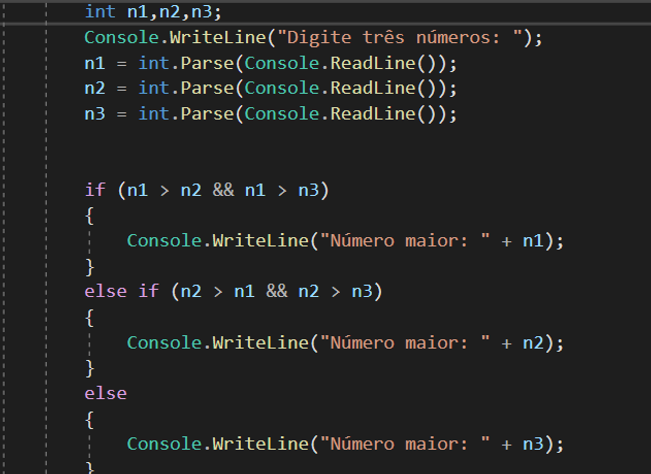
## Métodos e funções

No contexto da programação, uma "função" é um conjunto de comandos que realiza uma tarefa específica e pode ser reutilizado em diferentes partes de um programa. Funções são como pequenos programas independentes que podem ser chamados por outros programas para realizar uma determinada ação. Elas ajudam a modularizar e organizar o código, tornando-o mais legível, manutenível e reutilizável. As funções podem receber argumentos, processar dados e retornar um resultado, permitindo uma abordagem modular e estruturada na construção de software.

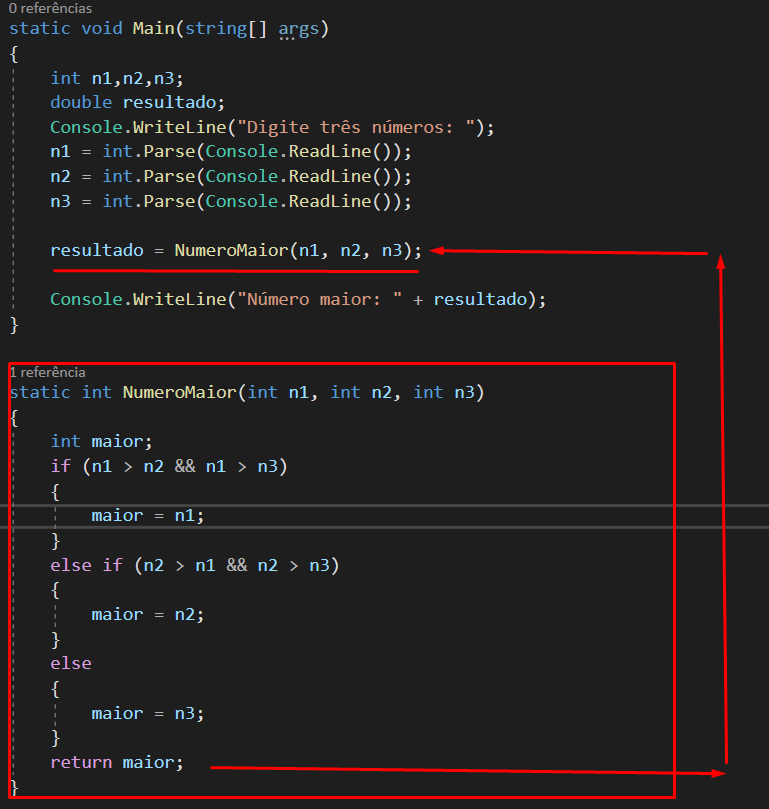
Exemplo:

1 – Crie um programa que leia três números e mostre o maior número digitado

Na imagem abaixo, o exercício foi realizado sem a utilização de uma função. A desvantagem é que toda vez que for necessário ler três números e mostrar o maior número digitado, será preciso repetir a verificação utilizando if-else. Isso pode resultar em duplicação de código, tornando o programa mais extenso e difícil de manter.



Na imagem abaixo, foi utilizada a função criada "NumeroMaior" para fazer a verificação do maior número entre três valores. A vantagem é que, em outra parte do código, se for necessário fazer a mesma verificação, basta chamar a função e passar os parâmetros de entrada. Dessa forma, evita-se a duplicação de código e torna-se mais modular e reutilizável.



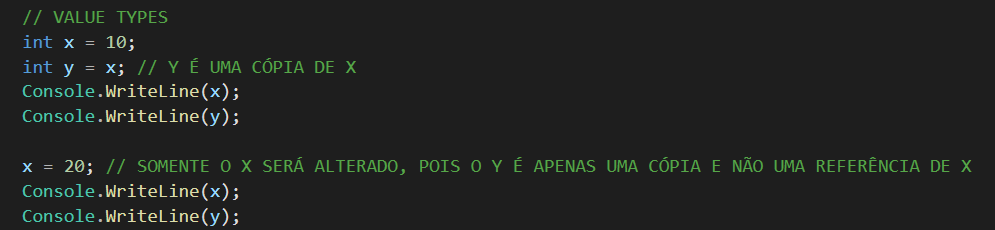
A variável "resultado" está recebendo o valor retornado pela função "NumeroMaior", que é chamada com os parâmetros de entrada "n1", "n2" e "n3". Dentro da função, esses parâmetros são verificados para determinar qual deles é o maior número, e em seguida é utilizado o comando "return maior" para retornar o valor do maior número encontrado.

## Value Types e Reference Types

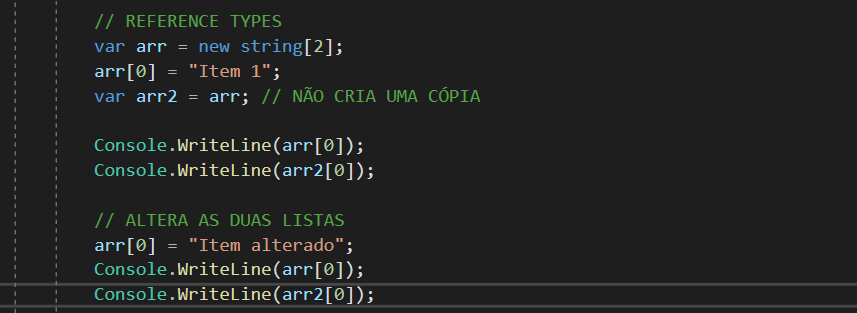
Value Types e Reference Types são dois conceitos importantes na programação, principalmente em linguagens que suportam tipagem forte. Eles se referem à forma como os dados são armazenados e manipulados na memória.

Value Types (tipos de valor) são tipos de dados que são armazenados diretamente no local em que são declarados. Isso significa que, ao atribuir um valor a uma variável de um tipo de valor, o valor em si é armazenado na variável. Reference Types (tipos de referência), por outro lado, são tipos de dados que são armazenados em uma área de memória separada chamada heap. Em vez de armazenar o valor diretamente na variável, uma referência ao valor é armazenada na variável. Isso significa que a variável não contém o valor real, mas sim um endereço de memória que aponta para o local onde o valor está armazenado.

Exemplo:



Y contínua com o valor 10, pois é uma cópia armazenada no stack da memória, onde não temos referência. Diferente do Reference Types que armazena o endereço do objeto que contém os dados.



Classes, objetos e arrays são do tipo Reference Types.