



Programação Orientada a Objetos

Módulo 1

Histórico de Linguagens Orientadas a Objetos

Revisão de Conceitos de Programação utilizando C#

Curso de Gestão da Informação

Prof. Dr. Daniel A. Furtado – FACOM/UFU

Conteúdo protegido por direito autoral, nos termos da Lei nº 9 610/98

A cópia, reprodução ou apropriação deste material, total ou parcialmente, é proibida pelo autor

Objetivos do Módulo

- Apresentar um breve histórico das linguagens de Programação Orientada a Objetos.
- Revisar conceitos básicos de programação utilizando a linguagem orientada a objetos C#.
- Introduzir o ambiente de desenvolvimento do Visual Studio Community, a linguagem C# e a plataforma .NET.
- Fazer a transição de Python para C#.
- Escrever os primeiros programas em C#.

Histórico da POO - Simula

- **Simula** é considerada a primeira linguagem orientada a objetos.
- Surgiu em **1967** na Noruega para simular eventos do mundo real.
- Introduziu conceitos básicos como **Classes** e **Objetos**.
- Introduziu a ideia de **Garbage Collection**, o mecanismo de limpeza automática de memória em programas modernos com memória gerenciada.

Histórico da POO – Smalltalk

- No início dos anos 1970 foi desenvolvida a **Smalltalk** no Xerox PARK.
- Traz a ideia de orientação à objeto pura – **tudo é objeto**.
- Popularizou a ideia de "**mensagens**" entre objetos (chamada de método).
- Introduziu os conceitos de **Encapsulamento**, **Herança** e **Polimorfismo**.
- Implementa o conceito de **máquina virtual** (antecessor da JVM).

Histórico da POO - C++

- No início dos anos 1980 surge o C++.
- Adiciona classes e objetos à linguagem C.
- Torna a POO viável no desenvolvimento de aplicações de alto desempenho e sistemas operacionais.
- Primeira linguagem orientada a objetos amplamente utilizada pela indústria.

Histórico da POO – Java

- Em 1995 a Sun Microsystems lança a linguagem **Java**
- Projetada para ser robusta e portátil
 - *"Write Once, Run Anywhere"*.
 - A Java Virtual Machine (JVM) permitiu que o mesmo código pudesse ser executado em diferentes sistemas operacionais.
- Popularização da POO em aplicações empresariais.

Histórico da POO – C#

- Lançada pela Microsoft em 2002 como parte do *framework* .NET.
- Projetada para ser uma **evolução** das linguagens Java e C++.
- **Puramente** orientada a objetos, mas com um design mais moderno.
- Código aberto e **multiplataforma**.
- Permite o desenvolvimento de aplicativos e sistemas de diversos tipos para Windows, macOS, Linux, iOS, Android e Web.

Histórico da POO - Atualidade

- Atualmente, a POO constitui o paradigma de programação dominante no desenvolvimento de software.
- Várias linguagens passaram a ter algum suporte à POO, mesmo sem ser puramente orientada a objetos:
 - JavaScript, Python, Swift, PHP, Kotlin etc.
 - Integração da POO com outros paradigmas como o funcional, procedural e outros.

Por que aprender Orientação a Objetos com C#?

- C# é uma linguagem orientada a objetos sólida e moderna.
- Suporta desde os conceitos mais básicos até os mais avançados de POO.
- Mais simples quando comparada a Java:
 - Requer menos código para expressar os mesmos conceitos de POO.
- Ecossistema de desenvolvimento amigável para iniciantes:
 - Com o Microsoft Visual Studio Community (gratuito).
- Linguagem versátil e em contínua evolução.
- Permite o desenvolvimento multiplataforma com o ecossistema .NET.
- Amplamente utilizada em sistemas empresariais, web APIs e ferramentas de análise.
- Mais adequada que o Python, pois os pilares da POO podem ser declarados de forma estrutural e explícita, como modificadores de acesso, interfaces e muitos outros conceitos fundamentais.

Framework .NET

- C# é utilizada em conjunto com o framework .NET, uma **plataforma** de desenvolvimento criada pela Microsoft que permite desenvolver aplicações de diversos tipos, para várias plataformas e arquiteturas.
- Permite construir:
 - **Aplicações Desktop**: softwares para Windows, Linux e macOS;
 - **Aplicações Móveis**: apps nativos para iOS e Android (com .NET MAUI);
 - **Aplicações Web**: APIs para a Web e serviços de *backend* (ASP.NET Core);
 - **Cloud e IA**: soluções de Machine Learning e Big Data.
- .NET é gratuito e de código aberto (open source), e suporta também as linguagens F# (funcional) e Visual Basic.

Framework .NET

- O framework .NET é composto por dois componentes principais:
 - A **Common Language Runtime** (CLR): máquina virtual que gerencia e executa os programas, permitindo a portabilidade do código (similar à JVM do Java e à PVM do Python).
 - A **Class Library**: biblioteca de classes que fornecem funcionalidades essenciais às aplicações como manipulação de *strings*, acesso à banco de dados, manipulação de arquivos etc.

Framework .NET

- As aplicações .NET são primeiramente desenvolvidas utilizando uma de suas linguagens (C#, F# ou Visual Basic).
- Posteriormente, o código é compilado em uma linguagem **intermediária** denominada *Common Intermediate Language – CIL*
 - O resultado é armazenado em **assemblies** – arquivos **.exe** ou **.dll**
- Quando a aplicação é executada, a máquina virtual (CLR/runtime) usa um compilador *just-in-time* (JIT) para transformar o **assembly** em **código de máquina** conforme a arquitetura do computador.

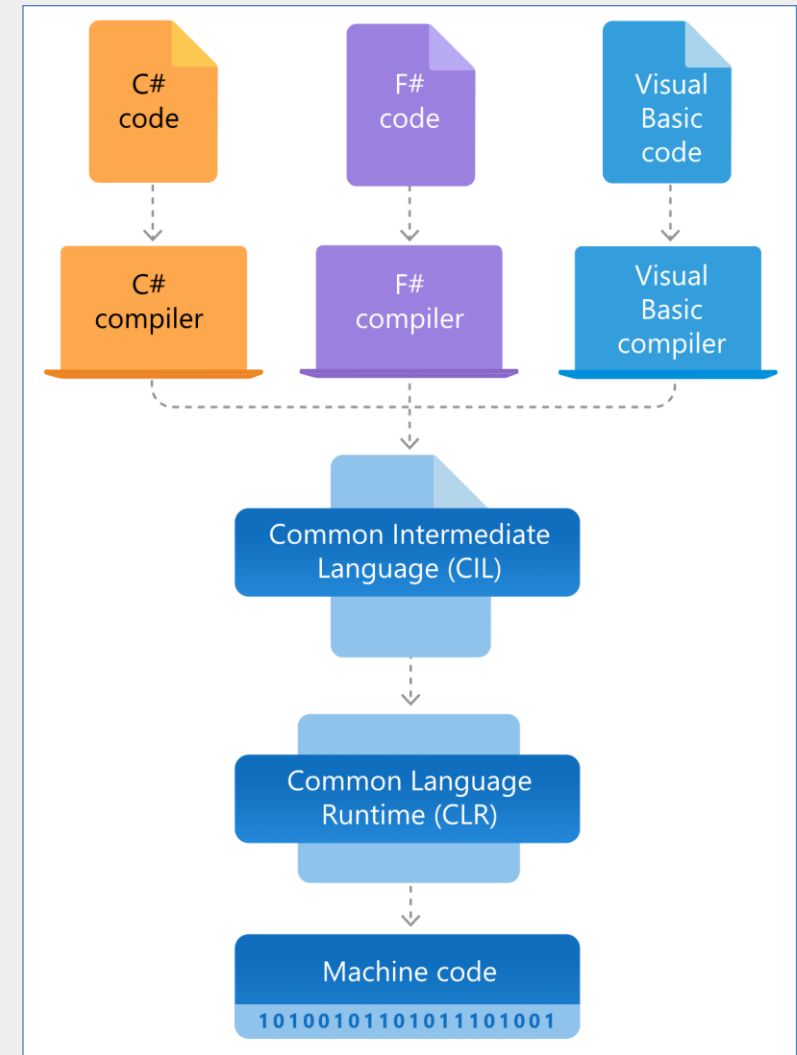


Figura adaptada de dotnet.microsoft.com

Ambiente de Desenvolvimento

- Recomenda-se a utilização do [Visual Studio 2026 Community](https://visualstudio.microsoft.com/pt-br/vs/community/) como ambiente de desenvolvimento C# (IDE - *Integrated Development Environment*).
- Pode ser baixado e utilizado de forma gratuita (estudantes, indivíduos).
- Atenção: **não** é o mesmo que o VS Code*
- Download: <https://visualstudio.microsoft.com/pt-br/vs/community/>
- Suporta a C# com o framework .NET 10 (versão mais recente).
- Durante a instalação, marque os *workloads* (necessário aprox. 20 GB):
 - **.NET desktop development.**
 - **.NET Multi-plataform App UI Development**

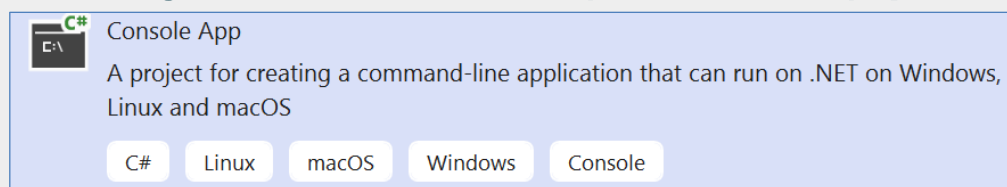
* É possível executar o código C# a partir do VS Code utilizando extensões (como a C# Dev Kit) e o terminal do VS Code (conforme apresentado a seguir). No entanto, essa forma não é recomendada neste curso por oferecer suporte limitado aos recursos da linguagem e do framework, e por não oferecer as mesmas facilidades de uso do Microsoft Visual Studio 2026 Community.

Em último caso, quando a instalação da IDE não é possível, pode-se utilizar compiladores online:

<https://www.programiz.com/csharp-programming/online-compiler/>

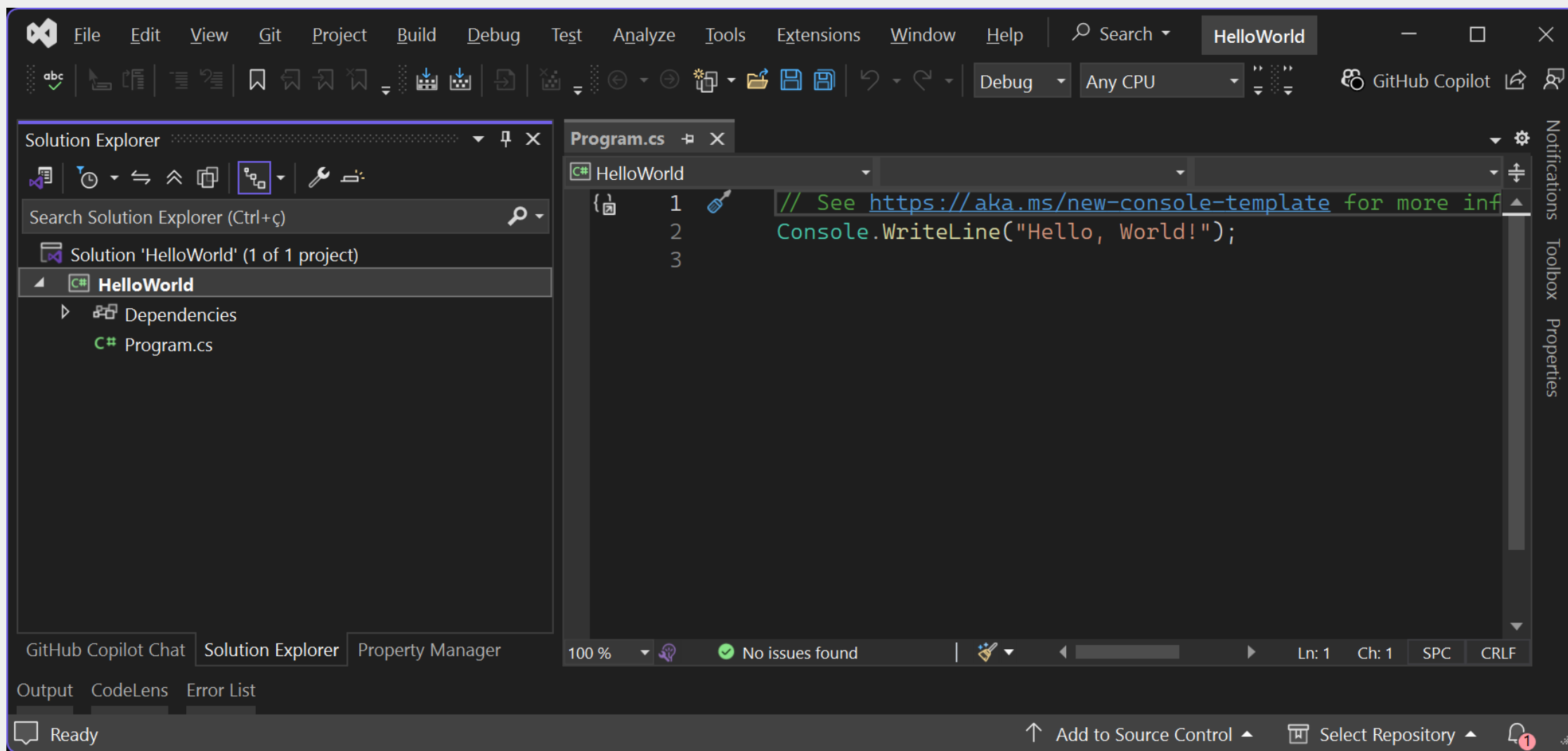
Primeiro Projeto no Visual Studio Community

1. Abra o Visual Studio.
2. Clique em **File** → **Create a new project**.
3. Na caixa de seleção **All Languages**, escolha **C#**
4. Na caixa de seleção **All Project Types**, escolha **Console**
5. Em seguida, escolha a primeira opção **Console App** da listagem:





6. Clique em **Next**, dê um nome ao projeto ("HelloWorld"), escolha o local de instalação e marque a opção **Place solution and Project in the same directory**.
7. Clique em **Next**. Na escolha do framework, escolha a opção **.NET 10** e clique em **Create** para finalizar.

Tela do Primeiro Projeto no Visual Studio Community



Execução do Programa

- Para executar o programa sem depurá-lo basta teclar **Ctrl-F5** ou clicar no ícone **Play** da barra de ferramentas (sem o fundo verde - ).
- Para executar o programa em modo de **depuração**, pode-se teclar **F5** ou clicar no ícone **Play** com fundo totalmente verde - .
- Em C# utilizamos `Console.WriteLine("Mensagem")` para imprimir mensagens na janela de console da aplicação, com quebra de linha no final.
- Para imprimir sem a quebra de linha, utilize `Console.Write("Mensagem")`.
- Tal comando tem efeito análogo à função `print("Mensagem")` da linguagem Python.

Ambiente de Desenvolvimento Alternativo – VS Code

- Caso utilize Linux ou, por algum motivo, não consiga instalar o Visual Studio Community, recomenda-se, como alternativa, utilizar o **Microsoft VS Code**, seguindo os passos:
 1. Instale o VS Code: <https://code.visualstudio.com/download>
 2. Abra o VS Code, clique na aba **Extensões** na barra lateral esquerda e instale a extensão **C# Dev Kit** da Microsoft;
 3. Crie uma pasta para o projeto C# (ex. D:\Temp\Trab1);
 4. Abra o terminal do VS Code: menu **Terminal** → **Novo Terminal**;
 5. No terminal, navegue até a pasta criada digitando: **cd D:\Temp\Trab1**;
 6. Digite **dotnet new console** para criar um **novo projeto** na pasta;
 7. Digite "**code .**" para abrir o projeto em uma nova janela do VS Code;
 8. Abra novamente o terminal e digite **dotnet run** para executar o código.
 9. Para adicionar classes, acrescente um novo arquivo .cs no projeto utilizando o painel **Explorer** do VS Code.

Observações Gerais sobre a Linguagem

- Em C#, as declarações precisam terminar com o **ponto-e-vírgula**.
 - Diferente do Python, que utiliza quebra de linha e indentação.
- C# diferencia maiúsculas e minúsculas (case sensitive).
- Comentários de linha: `// comentário de linha`
- Comentários de bloco: `/* comentário de bloco */`

Declaração de Variáveis

- Em C#, os tipos das variáveis são definidos em tempo de compilação e não podem ser alterados posteriormente.
 - Tipagem estática e fixa.
- Para declarar uma variável, coloca-se o seu **tipo** seguido do seu **nome**:
 - `tipoDaVariavel nomeDaVariavel;`
- Exemplos de variáveis em C#:
 - `int a; // declara a variável a para armazenar números inteiros.`
 - `int idade = 20; // declara e inicializa a variável b com o valor 5.`
 - `string nome = "Fulano"; // declara e inicializa uma string`
 - `float altura = 1.75f; // declara e inicializa uma variável de ponto flutuante`
- Em algumas situações, pode-se utilizar **var** para que o tipo seja **inferido** pelo compilador a partir do valor sendo atribuído:
 - `var nome = "Ciclano"; // nome será do tipo string`
 - `var idade = 20; // idade será do tipo int`

Tipos mais Comuns

■ Inteiros

- **short** – para inteiros pequenos (2 bytes, -32,768 a 32,767).
- **int** – para números inteiros (4 bytes, -2,147,483,648 a 2,147,483,647).
- **long** – para inteiros longos (8 bytes).

■ Decimais

- **float** – para números decimais de menor precisão (4 bytes).
- **double** – para números decimais de dupla precisão (8 bytes).
- **decimal** – permite a representação exata de valores monetários (porém com maior custo computacional).

■ Lógico e textual

- **bool** – para valores lógicos **true** ou **false** (**bool logado = true**).
- **char** – para um único caractere; aspas simples (**char sexo = 'm'**).
- **string** – para cadeia de caracteres; aspas duplas (**string nome = "João"**).

Strings em C#

- Em C# as *strings* são representadas como objetos (e não tipos primitivos).
- Caracteres individuais podem ser acessados com colchetes []
 - `string nome = "Fulano";`
 - `Console.WriteLine(nome[0]); // mostra a letra 'F'`
 - `Console.WriteLine(nome[1]); // mostra a letra 'u'`
- São imutáveis. Portanto, não podem ser alteradas de fato (qualquer alteração resulta na **criação** de uma nova string).

O tipo `string` em C# é, na verdade, um *alias* para a classe `System.String` do framework .NET.

Concatenando Strings com o Operador +

- `string primNome = "João";`
- `string ultNome = "Silva";`
- `string nomeCompleto = primNome + " " + ultNome;`

Interpolação de Variáveis em Strings

- Para inserir nomes de variáveis na string, pode-se utilizar o **operador \$** antes das aspas e inserir os nomes das variáveis entre **chaves { }**
- Essa forma é mais prática do que utilizar o operador de concatenação **+**

```
int a = 1;
int b = 2;
int c = 3;
int delta = b*b - 4*a*c;

string msg = $"O discrim. da eq. com coeficientes {a}, {b}, e {c} é {delta}";
Console.WriteLine(msg);
```

Entrada de Dados na Janela de Console

Para receber dados do usuário pela janela de console pode-se utilizar o método `Console.ReadLine()`

```
Console.Write("Informe o seu nome: ");  
string nome = Console.ReadLine();  
Console.WriteLine($"Bem vindo, {nome}");
```

OBS: No código acima percebe-se um *warning* do compilador destacando a expressão `Console.ReadLine` (sublinhado verde). Isso acontece porque em situações especiais esse método pode retornar `null`, valor que não está previsto para a `string nome`. (O C# 8.0 introduziu o conceito de *nullable reference types*, recomendando que o programador indique explicitamente que uma variável poderá receber o valor `null`).

Operador ??

- Uma forma de eliminar o *warning* do compilador no exemplo anterior é utilizando o operador ?? do C#.
- O operador ?? permite atribuir um valor diferente à variável caso a expressão que o antecede resulte em `null`.
- No exemplo a seguir, caso `Console.ReadLine` retorne um valor diferente de `null`, ele será atribuído normalmente à variável `nome`. Caso contrário, será atribuído a string vazia "".

```
Console.WriteLine("Informe o seu nome: ");  
string nome = Console.ReadLine() ?? "";  
Console.WriteLine($"Bem vindo, {nome}");
```

Operador ? na Definição de Tipos

Outra forma de eliminar o *warning* do compilador no exemplo anterior é utilizando o operador ? logo depois de `string`:

```
Console.Write("Informe o seu nome: ");  
string? nome = Console.ReadLine();  
if (nome != null)  
    Console.WriteLine(nome);
```

O operador ? nesse contexto é uma forma do programador dizer ao compilador que está **ciente** de que a variável de referência poderá receber o valor `null` e que tomará as devidas medidas para evitar eventuais erros causados por essa atribuição.

Obtendo o Tamanho da String

O número de caracteres da string (seu tamanho) pode ser obtido por meio da propriedade `Length` da string.

```
Console.Write("Informe o seu nome: ");  
string nome = Console.ReadLine() ?? "";  
Console.WriteLine($"O nome informado tem {nome.Length} caracteres!");
```

Obtendo Substrings

- O método `Substring` permite resgatar uma parte específica da `string`
- Por exemplo, `nome.Substring(0, 3)` retorna a substring de `nome` que começa na posição `0` e tem `3` caracteres de comprimento
- Exemplo prático

```
Console.WriteLine("Informe o seu nome: ");  
string nome = Console.ReadLine() ?? "";  
string iniciais = nome.Substring(0, 2);  
Console.WriteLine($"As duas letras iniciais do seu nome são {iniciais}");
```

OBS: Outra forma de obter uma substring é utilizando o operador ponto-ponto (`..`)
Exemplo: `string iniciais = nome[0..2];`

Conversão de String em int/float/double

- O método `Console.ReadLine` sempre retornará uma `string`. Se o usuário digitar o número `20`, por exemplo, o método retornará a `string` `"20"`.
- Portanto, se é esperado um número do usuário, deve-se fazer a devida conversão antes de executar operações matemáticas.
- Os métodos `int.Parse`, `float.Parse`, e `double.Parse` permitem converter uma `string` para `inteiro`, `float` e `double`, respectivamente.

```
Console.WriteLine("Informe a sua idade: ");  
string entrada = Console.ReadLine() ?? "";  
int idade = int.Parse(entrada);  
Console.WriteLine($"Em 5 anos você estará com {idade+5}");
```

OBS: Neste código o método `int.Parse` lançará uma exceção (levando a um erro na execução do código) caso o usuário digite um valor que não seja um número inteiro. Veja uma forma de tratar essa situação no próximo slide.

Conversão de String em int/float/double

```
Console.Write("Informe a sua idade: ");  
string entrada = Console.ReadLine() ?? "";  
_ = int.TryParse(entrada, out int idade);  
Console.WriteLine($"Em 5 anos você estará com {idade + 5} ");
```

Neste exemplo, o método `int.TryParse` tentará fazer a conversão da string de entrada para um valor inteiro. Em caso de sucesso, `TryParse` retornará `true` e o valor convertido (`idade`) é retornado no parâmetro de saída `idade`. Caso contrário, `TryParse` retornará `false` e o parâmetro `idade` terá o valor padrão `0`. Neste exemplo simples o valor lógico retornado não é utilizado e por esse motivo o caractere `_` foi colocado no início da linha, descartando o valor lógico de retorno.

Constantes

- Uma **constante** é uma variável cujo valor não pode ser alterado.
- Para defini-las, basta utilizar a palavra reservada **const** antes no nome da variável.
- Exemplo:
 - `const double PI = 3.14159;`

Operadores Aritméticos, Relacionais e Lógicos

Operadores Aritméticos e de Atribuição

Operador	Significado
+	Adição (e concatenação)
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira
++	Incremento
--	Decremento
=	Atribuição
+=	Atribuição com soma ou concatenação
-=	Atribuição com sub.

Operadores Relacionais e Lógicos

Operador	Significado
==	Comparação por igualdade
!=	Diferente
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a
&&	“E” lógico
	“Ou” lógico
!	Negação lógica

Exercícios

1. Escreva um programa em C# que receba as medidas de altura e peso de uma pessoa e calcule e mostre o seu IMC utilizando a fórmula a seguir. O resultado deve ser apresentado em uma mensagem formatada da seguinte forma: *"O IMC de uma pessoa com peso X e altura Y é Z"*, onde X, Y e Z representam valores fornecidos e calculados.

$$\text{IMC} = \frac{\text{Peso}}{\text{Altura}^2}$$

Estrutura Condicional if-else

- A estrutura **if-else** permite executar diferentes blocos de código dependendo de uma condição ser **verdadeira** ou **falsa**.
- A lógica é a mesma do Python, porém é necessário utilizar parênteses envolvendo a condição.
- Se o bloco de código contém mais de uma sentença, é necessário colocá-lo entre chaves.
- O bloco **else** é opcional.

```
float notaFinal = 78.5f;  
int faltas = 4;  
if (notaFinal ≥ 60 && faltas ≤ 18)  
    Console.WriteLine("Aprovado!");  
else  
    Console.WriteLine("Repvoadado!");
```

```
if (notaFinal ≥ 60 && faltas ≤ 18)  
{  
    Console.WriteLine("Aprovado!");  
    Console.WriteLine("Parabéns!");  
}
```

Estrutura Condicional if-else

- É possível aninhar vários blocos **if-else** caso haja necessidade.
- C# não possui a palavra reservada **elif** como no Python.

```
if (notaFinal ≥ 60 && faltas ≤ 18)
    Console.WriteLine("Aprovado!");
else if (faltas < 18)
    Console.WriteLine("Reprovado por nota!");
else
    Console.WriteLine("Reprovado por falta!");
```

Operador Ternário "?"

- Permite atribuir valor às variáveis com base em condições.
- Sintaxe:
 - `tipo variavel = condicao ? valorSeVerdade : valorSeFalso;`
- Várias sentenças podem ser aninhadas como em um `if-else`.
- Exemplos:
 - `string situacao = nota >= 70 ? "Aprovado" : "Reprovado";`
 - `string conceito = nota >= 80 ? "A" : nota >= 60 ? "B" : "C";`

A variável `situacao` receberá a string `"Aprovado"` ou a string `"Reprovado"`, dependendo da condição `nota >= 70` ser `verdadeira` ou `falsa`, respectivamente.

Estrutura Condicional switch-case

- Permite testar uma variável com relação a múltiplos valores.
- Similar ao `match-case` do Python (versão ≥ 3.10).

```
Console.WriteLine("Informe uma opção: ");
string entrada = Console.ReadLine() ?? "0";
int opcao = int.Parse(entrada);

switch (opcao)
{
    case 1:
        Console.WriteLine("Opção 1 selecionada");
        break;
    case 2:
        Console.WriteLine("Opção 2 selecionada");
        break;
    case 3:
        Console.WriteLine("Opção 3 selecionada");
        break;
    default:
        Console.WriteLine("Opção inválida!");
        break;
}
```

Estrutura de Repetição for

- Permite executar um bloco de código várias vezes. Normalmente utilizada quando se deseja repetir o código um número de vezes **determinado**.
- Sintaxe:

```
for (inicialização; condição; atualização)
{
    // bloco de código a ser repetido
}
```

inicialização

- Executada uma única vez, antes do laço iniciar.
- Normalmente declara e inicializa uma variável.

condição

- Testada antes da execução de cada iteração.
- Se verdadeira, o bloco de código é executado. Se falsa, o laço é encerrado.

atualização

- Executada depois de cada iteração do laço.
- Normalmente atualiza a variável do laço com incremento ou decremento.

Estrutura de Repetição for

```
for (int i = 0; i < 10; i++)  
{  
    Console.WriteLine("Hello World!");  
}
```

Mostra "Hello World!" 10 vezes.

```
for (int i = 0; i < 20; i+=2)  
{  
    Console.WriteLine(i);  
}
```

Mostra os números pares não negativos menores que 20.

Estrutura de Repetição for

```
for (int i = 10; i ≥ 0; i--)  
{  
    Console.WriteLine(i);  
}
```

Mostra os inteiros de 0 a 10 na ordem contrária.

```
Console.Write("Informe o seu nome: ");  
string nome = Console.ReadLine() ?? "";  
for (int i = nome.Length - 1; i >= 0; i--)  
    Console.Write(nome[i]);
```

Apresenta o nome do usuário ao contrário.

Estrutura de Repetição for

```
Console.WriteLine("Informe um inteiro >= 0: ");  
string entrada = Console.ReadLine() ?? "0";  
int n = int.Parse(entrada);  
  
int fat = 1;  
for (int i = 2; i <= n; i++)  
    fat = fat * i;  
  
Console.WriteLine($"{n} fatorial é {fat}");
```

Calcula o fatorial de um número solicitado ao usuário.

Estrutura de Repetição for – Exercício

Quantas iterações serão executadas caso o usuário informe o valor 0? E se ele informar 1?

```
Console.WriteLine("Informe um inteiro >= 0: ");  
string entrada = Console.ReadLine() ?? "0";  
int n = int.Parse(entrada);  
  
int fat = 1;  
for (int i = 2; i <= n; i++)  
    fat = fat * i;  
  
Console.WriteLine($"{n} fatorial é {fat}");
```

Exercícios

1. Escreva um programa em C# que receba um inteiro `n` do usuário e mostre na tela todos os números **ímpares positivos** menores que `n`.
2. Escreva um programa em C# que receba uma string do usuário e contabilize o número de letras `'a'` presentes na string.
 - A verificação deve ser feita caracter por caracter da string.
 - A quantidade deve ser informada ao usuário.

Estrutura de Repetição `while`

- Utilizada quando se deseja repetir um bloco de código **enquanto** uma determinada condição permanecer **verdadeira**.
- Geralmente não se sabe, de antemão, quantas vezes o código precisa ser repetido. Sua sintaxe geral é:

```
while (condição)
{
    // bloco de código a ser repetido
}
```

- Se a condição é inicialmente **falsa**, o bloco nem é executado.
- Se a condição é **verdadeira**, o bloco de código é então executado. Em seguida, a condição é novamente testada. Se continua verdadeira, o bloco é novamente executado e processo se repete. Se falsa, o processo se encerra (sem nova execução).

Estrutura de Repetição while

```
Console.Write("Informe o seu nome: ");
string nome = Console.ReadLine() ?? "";
while (nome == "")
{
    Console.Write("Nome inválido. Informe um nome não vazio: ");
    nome = Console.ReadLine() ?? "";
}
```

Neste exemplo o `while` é utilizado para solicitar um nome ao usuário até que seja informado um valor diferente da string vazia `""`.

Estrutura de Repetição do-while

- Utilizada quando se deseja repetir um bloco de código **ao menos uma vez** e **enquanto** uma determinada condição permanecer **verdadeira**.
- Geralmente não se sabe, de antemão, quantas vezes o código precisa ser repetido, mas sabe-se que é necessário executá-lo ao menos uma vez. Sua sintaxe geral é:

```
do
{
    // bloco de código a ser repetido
} while (condição)
```

- Diferentemente do **while**, com a estrutura **do-while** a condição é verificada no final (e não no início).
 - Por isso o bloco de código é executado pelo menos uma vez.
- Se a condição é **verdadeira**, o bloco de código é executado novamente e o processo se repete. Se **falsa**, o processo se encerra.

Estrutura de Repetição do-while - Exemplo

```
string nome;  
do  
{  
    Console.WriteLine("Informe o seu nome: ");  
    nome = Console.ReadLine() ?? "";  
} while (nome == "");
```

Este exemplo solicitará repetidamente o **nome** ao usuário até que um valor diferente de vazio seja fornecido. Repare que ao utilizar o **do-while**, evitamos a repetição dos comandos `Console.WriteLine` e `Console.ReadLine`, como no último exemplo.

Estrutura de Repetição do-while

```
int opcao;  
do  
{  
    Console.WriteLine("1 - Fazer cálculo A");  
    Console.WriteLine("2 - Fazer cálculo B");  
    Console.WriteLine("3 - Sair");  
    Console.Write("Informe a opção desejada: ");  
    string entrada = Console.ReadLine() ?? "";  
    opcao = int.Parse(entrada);  
  
    // Executa as operações conforme opção escolhida...  
} while (opcao != 3);
```

Exemplo de um menu de opções utilizando a estrutura **do-while**. O usuário poderá acessar o menu repetidas vezes, até que seja escolhida a opção **3 -Sair**. Repare que nesta situação o **do-while** é mais apropriado, uma vez que precisamos apresentar a opções pelo menos uma vez. Se um simples **while** fosse utilizado, teríamos que duplicar a apresentação do menu (uma vez antes do **while** e outra dentro do **while**).

Arrays em C#

- São estruturas básicas que armazenam itens do mesmo **tipo** (array de inteiros, array de *floats*, array de objetos (referências) etc.).
- Possuem tamanho fixo, definido no momento da criação.
- São alocados em blocos contíguos de memória, permitindo acesso rápido e direto.
- Os itens do array podem ser acessados por índice e começam na posição 0.

Arrays em C#

- Podem ser declarados e posteriormente instanciados

```
int[] notas;  
notas = new int[5];  
notas[0] = 80; notas[1] = 85;
```

- Podem ser declarados e instanciados na mesma linha

```
int[] notas = new int[5];
```

- Declaração e instanciação usando `var`

```
var notas = new int[5];
```

- Declaração, instanciação e inicialização na mesma linha

```
string[] nomes = { "Fulano", "Ciclano", "Beltrano" };  
int[] notas = { 80, 85, 90 };  
notas[2] = 95; // altera a última nota de 90 para 95  
Console.WriteLine(notas[2]); // mostra a última nota atualizada
```

Percorrendo Array com Estrutura for

```
int[] pares = { 0, 2, 4, 6, 8, 10 };  
for (int i = 0; i < pares.Length; i++)  
{  
    Console.WriteLine($"Elemento {i}: {pares[i]}");  
}
```

Percorre item por item do array de números pares e apresenta cada elemento na tela utilizando um laço **for** simples. Observe que o tamanho do array é retornado pela propriedade **Length**.

Percorrendo Array com Estrutura foreach

```
int[] pares = { 0, 2, 4, 6, 8, 10 };  
foreach (int par in pares)  
{  
    Console.WriteLine(par);  
}
```

Percorre o array de pares utilizando a estrutura **foreach**. A cada iteração a variável **par** receberá automaticamente o próximo elemento do array **pares**. Observe que o acesso aos elementos é mais simples e intuitivo. Porém, não há a variável de índice (**i**), como no exemplo anterior, inviabilizando a apresentação das posições dos elementos.

Exercício

- Escreva um programa em C# que receba 10 strings do usuário e as armazene em um array de strings de 10 posições. Depois de solicitar todas as strings o programa deverá apresentá-las na ordem em que foram fornecidas utilizando a estrutura `foreach`, e na ordem contrária utilizando a estrutura `for`. O programa deve ter o menor número possível de linhas.

Referências

- Joseph Albahari. **C# 12 in a Nutshell**: The Definitive Reference. 1ª ed., 2023.
- Mark J. Price. **C# 9 and .NET 5 – Modern Cross-Platform Development**: Build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code, 5ª ed., 2020.
- learn.microsoft.com/en-us/dotnet/csharp/fundamentals