📚 Glossário Completo de C# para Iniciantes

Este glossário foi criado para quem deseja iniciar na linguagem **C#**. Com explicações claras, exemplos práticos e foco nos fundamentos essenciais da linguagem, ele é ideal para quem deseja desenvolver aplicações desktop, web, jogos (Unity), entre outros.

📌 Variáveis

São usadas para armazenar dados na memória.

```
csharp
CopiarEditar
int idade = 30;
double altura = 1.80;
string nome = "Lucas";
bool ativo = true;
```

📌 Constantes

Valor fixo que não pode ser alterado.

```
const double PI = 3.1415;
```

📌 Tipos de Dados

Principais tipos usados em C#:

- `int` — números inteiros
- `double` — números decimais
- `char` — caractere único (`'A'`)
- `bool` — verdadeiro ou falso
- `string` — texto

📌 Operadores

Utilizados para realizar operações matemáticas, relacionais e lógicas.

```
int soma = 5 + 3;
bool resultado = (10 > 5) && (5 < 8);
```

📌 Entrada e Saída

```
Console.WriteLine("Digite seu nome:");
string nome = Console.ReadLine();
Console.WriteLine("Olá, " + nome);
```

📌 Condicionais

Permitem executar diferentes blocos de código dependendo da condição.

```
if (idade >= 18)
{
    Console.WriteLine("Maior de idade");
}
else
{
    Console.WriteLine("Menor de idade");
}
```

📌 Estrutura `switch`

Permite testar múltiplos valores.

```
switch (opcao)
{
    case 1:
        Console.WriteLine("Opção 1");
        break;
    case 2:
        Console.WriteLine("Opção 2");
        break;
    default:
        Console.WriteLine("Outra opção");
        break;
}
```

📌 Laços de Repetição (Loops)

For:

```
for (int i = 0; i < 5; i++)
{
    Console.WriteLine(i);
}
```

While:

```
int contador = 0;
while (contador < 5)
{
    Console.WriteLine(contador);
    contador++;
}
```

📌 Arrays

Estruturas para armazenar múltiplos valores do mesmo tipo.

```
int[] numeros = {1, 2, 3, 4, 5};
```

📌 Listas (Coleções)

Permitem manipular listas de objetos dinamicamente.

```
List<string> nomes = new List<string>() {"Ana", "Bruno", "Carlos"};
nomes.Add("Daniela");
```

📌 Funções (Métodos)

Blocos reutilizáveis de código.

```
int Somar(int a, int b)
{
    return a + b;
}
```

📌 Função Principal

Ponto de entrada de um programa C#.

```
static void Main(string[] args)
{
    Console.WriteLine("Olá, Mundo!");
}
```

📌 Classes

Estrutura básica para Programação Orientada a Objetos.

```
class Pessoa
{
    public string Nome;
```

```
    public void Falar()
    {
        Console.WriteLine("Olá, meu nome é " + Nome);
    }
}
```

📌 Objetos

Instâncias de uma classe.

```
Pessoa p = new Pessoa();
p.Nome = "Maria";
p.Falar();
```

📌 Construtores

Métodos especiais chamados na criação de objetos.

```
class Produto
{
    public string Nome;

    public Produto(string nome)
    {
        Nome = nome;
    }
}
```

📌 Encapsulamento (Getters e Setters)

Controla o acesso aos atributos.

```
class Conta
{
    private double saldo;

    public void SetSaldo(double valor)
    {
        saldo = valor;
    }

    public double GetSaldo()
    {
        return saldo;
    }
}
```

📌 Herança

Permite criar classes derivadas de outras.

```
class Animal
{
    public void Comer() => Console.WriteLine("Comendo...");
}

class Cachorro : Animal
{
    public void Latir() => Console.WriteLine("Au au");
}
```

📌 Polimorfismo

Permite reescrever métodos de uma classe base.

```
class Animal
{
    public virtual void Som() => Console.WriteLine("Som genérico");
}

class Gato : Animal
{
    public override void Som() => Console.WriteLine("Miau");
}
```

📌 Interfaces

Contratos para classes implementarem métodos obrigatórios.

```
interface IVeiculo
{
    void Acelerar();
}

class Carro : IVeiculo
{
    public void Acelerar() => Console.WriteLine("Carro acelerando");
}
```

📌 Try-Catch (Tratamento de Exceções)

Utilizado para tratar erros.

```
try
{
    int numero = int.Parse("abc"); // Vai gerar erro
}
catch (FormatException)
{
    Console.WriteLine("Erro: formato inválido.");
```

```
}
```

📌 Nullable Types (Tipos Nulos)

Permitem que variáveis armazenem `null`.

```
int? idade = null;
```

📌 Lambda Expressions (Expressões Lambda)

Funções curtas e anônimas.

```
Func<int, int, int> soma = (a, b) => a + b;
Console.WriteLine(soma(3, 4));
```

📌 LINQ

Consulta de dados em coleções.

```
var numeros = new List<int>() {1, 2, 3, 4, 5};
var pares = numeros.Where(n => n % 2 == 0);
```

📌 Namespaces

Organização de classes em grupos.

```
namespace MeuProjeto
{
    class MinhaClasse { }
}
```

📌 Bibliotecas Importantes

- `System` — funcionalidades básicas
- `System.Collections.Generic` — listas, dicionários
- `System.Linq` — LINQ
- `System.IO` — arquivos
- `System.Threading` — threads e tarefas assíncronas