

Revisão de Fundamentos de Programação Estruturada: Por que Começar por Aqui?

Se você quer construir uma casa sólida, começa pela fundação. Em programação, não é diferente: **dominar os fundamentos estruturados é o que permite crescer com segurança para conceitos mais avançados, como a Programação Orientada a Objetos (POO).**

Por que revisar fundamentos, mesmo que você já tenha estudado programação antes?

- Porque revisar os conceitos básicos **reforça a lógica** necessária para entender estruturas complexas.
- Porque os detalhes de tipos de dados, estruturas de controle e funções mudam de linguagem para linguagem — e, no C#, há peculiaridades que podem fazer diferença no seu código.
- Porque o mercado valoriza profissionais que **sabem o porquê** das coisas e evitam erros comuns ao escrever código eficiente e legível.
- Porque POO se apoia totalmente nesses fundamentos — e muita gente tropeça em orientação a objetos por falhas ou "lacunas" na base.



Provocações para Engajamento

Reflexões para programadores

Você consegue explicar para um colega o que é uma variável ou o que faz um comando 'if'? Tente escrever uma explicação curta e simples, usando suas próprias palavras.

Veja o meme abaixo: Reflita: Por que tanta gente pensa que programar é simples?



Ativando seu conhecimento prévio:

- Liste, sem consultar o material, nomes de tipos de dados que você já ouviu falar ou usou.
- Pense em uma situação do seu dia em que você tomou uma decisão (ex: "Se está chovendo, levo guarda-chuva"). Escreva um exemplo dessa decisão em linguagem natural — depois, tente imaginar como seria em código.

Contextualização para POO e Mercado

Tudo o que você vai revisar aqui será fundamental para aprender conceitos como classes, objetos, métodos e herança. **Uma base forte facilita a caminhada para conteúdos mais avançados e para o mercado de trabalho!**



Fundamentos

Tipos de dados, estruturas de controle e funções

Programação Estruturada

Algoritmos e lógica de programação



Programação Orientada a Objetos

Classes, objetos, herança e polimorfismo

Mercado de Trabalho

Aplicação prática dos conhecimentos

Sua Missão nesta Jornada

Sua missão nesta jornada: Revisar, exercitar e realmente dominar os fundamentos. Não só para passar de fase, mas para construir uma carreira sólida em programação!

Revisar

Relembrar e consolidar conceitos fundamentais de programação estruturada

Exercitar

Praticar com exemplos e desafios para fixar o conhecimento

Dominar

Alcançar fluência nos fundamentos para avançar com segurança

Tipos de Dados Primitivos em C#

Os tipos de dados primitivos são os blocos fundamentais para construir qualquer programa em C#. Eles definem como a informação é armazenada e manipulada pelo computador.

#

Numéricos

int, float, double - para representar valores numéricos inteiros e decimais

T

Texto

char, string - para representar caracteres individuais e sequências de texto



Lógico

bool - para representar valores verdadeiro ou falso

Introdução e Engajamento Específicos

Imagine que você está organizando uma caixa de ferramentas: cada ferramenta tem um propósito específico, ocupa um espaço diferente e serve melhor para determinados problemas. **Tipos de dados em programação funcionam assim:** cada tipo serve para representar uma categoria de informação — números, textos, verdadeiro/falso, entre outros.

Saber escolher o tipo de dado certo deixa seu programa mais eficiente, fácil de entender e evita muitos erros. Em C#, como em outras linguagens, esse conhecimento é a base para toda lógica, algoritmos e — mais adiante — a POO.



Provocações para Engajamento

Desafio:

Pense nos exemplos abaixo e tente decidir qual tipo de dado seria mais adequado para representar cada um: Escreva suas respostas antes de consultar o material!

1 Idade de uma pessoa

2 Nome completo

3 Nota de uma prova

4 Presença ou ausência em uma aula

5 Letra inicial do nome

Reflexão:

Quando, no dia a dia, você usaria uma régua e quando usaria uma balança? Por que ferramentas diferentes? Da mesma forma, em programação, escolhemos "ferramentas" (tipos) para diferentes necessidades. Pense nisso!

O que são Tipos de Dados Primitivos?

São **categorias básicas de dados** que uma linguagem de programação entende e armazena diretamente na memória. Eles determinam:

1 O que pode ser feito com aquele dado (operações).

2 Quanto espaço ocupa na memória.

3 O resultado de operações, conversões e comparações.

Em C#, os principais tipos primitivos são: int, float, double, char, bool, string.

Tipos, Sintaxe, Exemplos e Quando Usar

int – Números Inteiros

Definição: Representa números inteiros (positivos, negativos, zero), sem parte decimal.

Sintaxe: int idade = 20; int saldo = -100;

Uso típico: contadores, idades, quantidades, índices de listas.

Curiosidade: O int ocupa 4 bytes (32 bits) e vai de -2.147.483.648 até 2.147.483.647.

Dica: Use int sempre que não precisa de casas decimais. Se o valor pode ser muito grande (ex: população mundial), use long.

float e double – Números Reais (Ponto Flutuante)

float:

- Menor precisão, ocupa 4 bytes.
- Exemplo: float temperatura = 25.6f; (o f é obrigatório para indicar float!)

double:

- Maior precisão, ocupa 8 bytes.
- Exemplo: double salario = 1350.75;

Quando usar?

- float: medições menos precisas, economia de memória.
- double: operações financeiras, cálculos que exigem precisão.

Atenção: Para valores financeiros, o ideal é o tipo decimal, que é ainda mais preciso.

bool – Valores Lógicos

Definição: Representa verdadeiro (true) ou falso (false).

Exemplo: bool aprovado = true; bool maiorDeIdade = idade >= 18;

Quando usar? Em verificações, condições, flags.

char – Caracteres Individuais

Definição: Um único símbolo — pode ser letra, número, símbolo especial.

Exemplo: char letra = 'A'; char digito = '7'; char simbolo = '\$'; (Obs: char sempre entre aspas simples!)

Common type pi



string – Sequência de Caracteres

Definição: Cadeias de caracteres, ou seja, textos.

Exemplo: string nome = "Maria"; string mensagem = "Bem-vindo!";

Quando usar? Para nomes, frases, qualquer texto.

Boas Práticas e Dicas Profissionais

1 Escolha o tipo certo

Evita bugs e uso excessivo de memória.

2 Declaração explícita

Prefira declarar o tipo, a menos que use var com clareza no contexto.

3 Atenção à precisão

Para cálculos financeiros, prefira decimal ao invés de float/double.

4 Conversões (casting)

```
int x = 10;  
double y = x; // ok (int -> double)  
int z = (int) y; // pode perder parte decimal!
```

Armadilhas e Erros Comuns

Principais armadilhas ao trabalhar com tipos de dados

1 Overflow

Tentar armazenar um valor maior que o limite do tipo.

2 Precisão

Comparar floats/doubles diretamente pode dar resultado inesperado.

```
double a = 0.1 + 0.2;  
Console.WriteLine(a == 0.3); // false!
```

3 Strings não são chars

```
string nome = 'A'; // ERRO!  
char letra = "A"; // ERRO!
```

Exercícios Práticos

1 Declare variáveis

Para: idade de uma pessoa, salário, letra inicial do nome, aprovado em disciplina, frase de boas-vindas.

2 Identifique qual tipo usar

Para cada situação:

- Nota de uma prova
- Quantidade de alunos
- Nome do curso
- Valor da mensalidade
- Status de pagamento (pago ou não)

3 Experimente somar dois char

No C#. O que acontece? Anote sua explicação.

4 Teste o código

```
double preco = 9.99;  
int unidades = 3;  
double total = preco * unidades;
```

Agora, troque preco para int. O que muda?

Resumo Visual (Quadro-Resumo)

Tipo	Exemplo	Ocupa (bytes)	Usos típicos
int	10, -4, 0	4	Idade, contador
float	1.7f, -3.2f	4	Medidas, sensores
double	10.75, -2.3	8	Salário, média
char	'A', '8', '#'	2	Letra, dígito
bool	true, false	1	Aprovação, ativo
string	"João", "ABC"	varia	Nome, frase, mensagem

Conexão com POO

Em Programação Orientada a Objetos, cada objeto manipula atributos que têm tipos definidos. **Se você domina bem os tipos primitivos, vai entender com mais facilidade como definir e usar propriedades nas suas classes**, evitando bugs e erros de lógica!



Tipos Primitivos

Base para definição de atributos

Classes

Estruturas que agrupam atributos e métodos

Objetos

Instâncias de classes com valores específicos

