Dominando o C# básico

Conceitos,
Exemplos e
Boas Práticas

Carlos Diones dos Santos

Introdução

Aprender a programar é mais do que decorar comandos ou copiar códigos prontos: é aprender a pensar de forma lógica e estruturada, capaz de transformar ideias em soluções reais.

Entre tantas linguagens disponíveis, o C# se destaca como uma das mais poderosas e versáteis, utilizada em aplicações web, APIs, jogos, softwares corporativos e muito mais. Por meio do .NET, ele permite criar projetos robustos e profissionais, mesmo para quem está começando.

Este eBook foi pensado para quem está dando os primeiros passos na programação. Aqui, você vai encontrar:

- Conceitos claros e diretos sobre C# e .NET.
- Exemplos comentados que você pode testar e adaptar.
- Boas práticas para escrever código limpo e fácil de manter.

Prepare seu editor de código, abra o Visual Studio ou VS Code, e vamos começar esta jornada.

Capítulo 1

Entendendo a Estrutura de um Programa C#

Antes de criar projetos maiores, é importante entender **como um programa em C# é organizado.**

Mesmo o código mais simples segue uma estrutura padrão que define **onde tudo começa.**

Exemplo de um programa básico

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Olá, C#!");
    }
}
```

O que significa cada parte

- **using System;** → Importa funcionalidades básicas, como Console.
- **class Program** → Toda aplicação em C# precisa estar dentro de uma classe.
- **static void Main()** → É o ponto de entrada do programa (onde tudo começa).
- **Console.WriteLine()** → Exibe mensagens no console.

Boas práticas

- Use **nomes claros** para classes e métodos.
- **Indente o código** corretamente para facilitar a leitura.
- Use **comentários curtos e úteis** para explicar partes importantes.

Exemplo com boas práticas

```
using System;

namespace MeuApp
{
    class Saudacao
    {
        static void Main()
        {
            // Exibe uma mensagem no console
            Console.WriteLine("Bem-vindo ao seu primeiro programa em
C#!");
        }
    }
}
```

Capítulo 2

Variáveis, tipos de dados e conversões

Em C#, **variáveis** são usadas para armazenar informações temporariamente na memória do computador.

Cada variável precisa ter **um tipo de dado**, que define o que ela pode guardar.

```
using System;

class Program
{
    static void Main()
    {
        string nome = "Carlos";
        int idade = 25;
        double altura = 1.75;
        bool ativo = true;

        Console.WriteLine($"{nome}, {idade} anos, {altura}m - 
Ativo: {ativo}");
    }
}
```

Conversões de tipo
Às vezes é preciso **converter valores** de um tipo para outro.

Dica: sempre valide a entrada antes de converter, para evitar erros.

```
int numero = int.Parse("10");          // converte texto em número
string texto = numero.ToString();     // converte número em texto
double valor = Convert.ToDouble("3.14");
```

Capítulo 3

Estruturas de Controle

As **estruturas de controle** permitem que seu programa **tome decisões** e **execute ações repetidas** conforme as condições.

If / Else
Use `if` para testar condições.
O bloco `else` é executado se a condição for falsa.

```
int nota = 80;

if (nota >= 70)
    Console.WriteLine("Aprovado!");
else
    Console.WriteLine("Reprovado!");
```

Else If

Permite testar várias condições em sequência.

```
int nota = 85;

if (nota >= 90)
    Console.WriteLine("Excelente!");
else if (nota >= 70)
    Console.WriteLine("Bom!");
else
    Console.WriteLine("Precisa melhorar.");
```

Switch

Usado quando há muitas opções para comparar.

```
string dia = "segunda";

switch (dia)
{
    case "segunda":
        Console.WriteLine("Início da semana!");
        break;
    case "sexta":
        Console.WriteLine("Quase fim de semana!");
        break;
    default:
        Console.WriteLine("Dia comum.");
        break;
}
```

Boas práticas

- Use `if` para decisões simples e `switch` para múltiplas opções.
- Indente sempre o código dentro dos blocos.
- Evite comparações desnecessárias – mantenha o código limpo.

Capítulo 4

Métodos e Funções

Os **métodos** (ou funções) servem para **organizar o código** em partes reutilizáveis.

Eles ajudam a evitar repetições e deixam o programa mais limpo e fácil de entender.

```
tipo_retorno NomeDoMetodo(parâmetros)
{
    // código
    return valor; // (se houver retorno)
}
```

```
using System;

class Program
{
    static void Main()
    {
        int resultado = Somar(3, 5);
        Console.WriteLine($"Resultado: {resultado}");
    }

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

O método `Somar` recebe dois números e devolve a soma.

Método sem retorno

Se o método apenas executa algo, use `void`:

```
static void ExibirMensagem(string nome)
{
    Console.WriteLine($"Olá, {nome}!");
}
```

Capítulo 5

Classes e Objetos

Em C#, quase tudo gira em torno de **classes e objetos**.
Eles são a base da **Programação Orientada a Objetos (POO)**, um dos pilares da linguagem.

Classe: um molde (modelo) que define características e comportamentos.
Objeto: uma instância (cópia) dessa classe em uso.

```
using System;

class Pessoa
{
    public string Nome;
    public int Idade;

    public void Apresentar()
    {
        Console.WriteLine($"Olá! Meu nome é {Nome} e tenho {Idade} anos.");
    }
}

class Program
{
    static void Main()
    {
        Pessoa pessoa1 = new Pessoa();
        pessoa1.Nome = "Carlos";
        pessoa1.Idade = 30;

        pessoa1.Apresentar();
    }
}
```

O que acontece:

- Criamos a classe **Pessoa** com duas variáveis (Nome, Idade) e um método (Apresentar).

Depois, criamos um **objeto** pessoa1 e usamos os valores.

Boas práticas

- Use **nomes no singular** para classes (Pessoa, Carro).

- Mantenha os atributos **privados** e acesse por **propriedades** (conceito chamado *encapsulamento*).

Cada classe deve representar **um único conceito.**

Capítulo 6

Boas Práticas no Dia a Dia

Escrever código que **funciona** é bom.
Mas escrever código que **é fácil de entender, manter e melhorar** é o que diferencia um bom desenvolvedor.
Aqui estão práticas simples que fazem diferença desde o início.

1. Use nomes claros
Evite abreviações e nomes genéricos.
Um bom nome explica sozinho o que o código faz.
Exemplo ruim:`int x = 10;`
Exemplo bom:`int idadeDoUsuario = 10;`

2. Mantenha o código limpo

- Use **espaços e quebras de linha** para separar partes lógicas.

- Evite blocos muito grandes dentro de um único método.

Dica: métodos curtos e focados facilitam testes e manutenção.

3. Evite repetição (princípio DRY – Don’t Repeat Yourself)
Se você perceber que está escrevendo o mesmo código várias vezes, transforme-o em um método reutilizável.

```
static void ExibirMensagem(string
texto){    Console.WriteLine(texto);
```

4. Comente com propósito
Comentários devem explicar **por que** o código existe, não apenas **o que** ele faz.

```
// Verifica se o usuário tem idade
suficiente para acessar o conteúdo
if (idade >= 18) Console.WriteLine("Acesso
permitido");
```

5. Teste sempre o seu código
Mesmo em exemplos simples, execute o programa e valide se o resultado é o esperado.
O hábito de testar desde cedo ajuda a evitar erros futuros.

Capítulo 7

Próximos Passos

Você chegou até aqui!

Agora já entende os **fundamentos do C#**: como estruturar um programa, usar variáveis, controlar o fluxo, criar métodos e classes – e aplicar boas práticas desde o início.Mas o aprendizado não termina aqui. Este é apenas o **primeiro degrau** da sua jornada como desenvolvedor .NET.

Continue praticando
A melhor forma de aprender programação é **escrevendo código todos os dias.**
Experimente:

- Recriar exemplos deste eBook de memória.

- Fazer pequenas modificações (trocar tipos de dados, adicionar novas funções).

Criar miniaplicativos de console para resolver problemas simples do dia a dia.

Explore o ecossistema .NET
Depois de dominar o básico, você pode avançar para:

- **ASP.NET Core:** desenvolvimento de sites e APIs.
- **Entity Framework:** manipulação de bancos de dados com C#.
- **Windows Forms e WPF:** criação de interfaces gráficas.
- **Blazor e MAUI:** aplicações web e multiplataforma com C#.

Cada etapa vai ampliar suas possibilidades dentro do .NET.

Participe da comunidade
Entre em grupos, fóruns e eventos sobre .NET e C#.
Alguns lugares para começar:

- dev.to
- Comunidades no Discord e LinkedIn voltadas a desenvolvedores iniciantes.

Aprender com outras pessoas acelera o crescimento e traz novas oportunidades.

Conclusão

Aprender a programar é uma jornada — e cada linha de código escrita é um passo adiante.

Com este eBook, você deu os primeiros passos no universo do **C# e do .NET**, compreendendo os fundamentos que sustentam qualquer aplicação: estrutura, lógica, métodos, classes e boas práticas.Mais importante do que decorar comandos, é **entender como pensar como um desenvolvedor**: analisar problemas, buscar soluções simples e manter o código claro e organizado.

Essas são as bases que vão acompanhar você em qualquer linguagem, projeto ou tecnologia.A partir daqui, o caminho é de **exploração e prática**.

Desafie-se a criar seus próprios projetos, participe da comunidade, troque experiências e mantenha a curiosidade viva.
O .NET é um ecossistema vasto — e você já domina o essencial para crescer dentro dele.

O próximo grande projeto pode começar com a próxima linha de código que você escrever.