

VITÓRIA SANTANA

Domine a programação com Dart e crie aplicativos incríveis!

ART BOSS

SEJA O CHEFÃO DO CÓDIGO MOBILE



EXPLORANDO DART - VITÓRIA SANTANA



Explorando Dart



Uma Jornada Simples e Poderosa

Se você está começando no desenvolvimento mobile ou busca novos desafios, este ebook é o seu guia para dominar o Dart. Vamos explorar os fundamentos dessa linguagem de forma simples e prática, com exemplos reais de código.

Ao longo dos capítulos, você aprenderá desde o básico até a programação orientada a objetos, sempre com explicações claras e aplicáveis no seu dia a dia de desenvolvedor.

Prepare-se para se tornar o chefão do código!

01

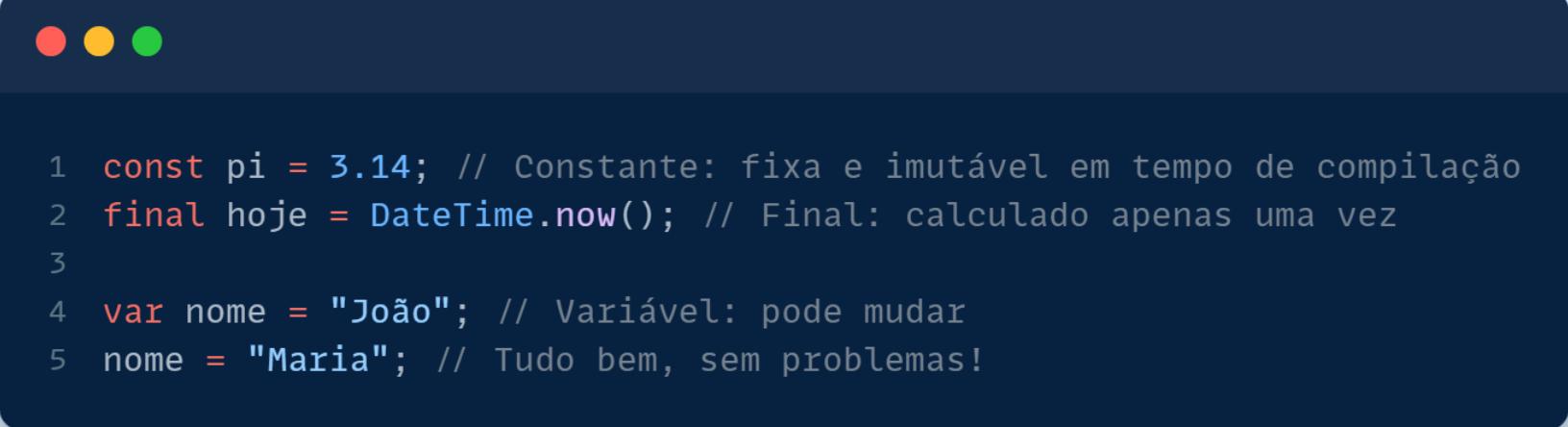
Fundamentos do Dart

Construindo a Base

Constantes e Variáveis

Aprenda a manipular os recursos essenciais para ser o chefão do Dart!

Em Dart, usamos **var** para variáveis que podem mudar. Já **final** e **const** são para valores fixos. A diferença é que **const** é usado para valores que já são conhecidos enquanto o código está sendo criado, e **final** é usado para valores que só são definidos quando o programa está rodando.



```
1 const pi = 3.14; // Constante: fixa e imutável em tempo de compilação
2 final hoje = DateTime.now(); // Final: calculado apenas uma vez
3
4 var nome = "João"; // Variável: pode mudar
5 nome = "Maria"; // Tudo bem, sem problemas!
```

Analogia

“Imagine que as constantes são como tatuagens: uma vez feitas, não podem ser mudadas. Já as variáveis são como adesivos: você pode trocar ou reposicionar sempre que precisar.”



Tipos Básicos

Em Dart, os tipos básicos são os tipos de dados que usamos para armazenar informações simples, como números ou texto.

```
 1 int idade = 25; // Números inteiros
 2 double altura = 1.75; // Números decimais
 3 String saudacao = "Olá, Dart!"; // Textos
 4 bool estaSatisfeito = true; // Valores verdadeiros ou falsos
 5
 6 // Listas: Coleções de elementos
 7 List<String> frutas = ["Maçã", "Banana", "Laranja"];
 8 print(frutas[1]); // Banana
 9
10 // Mapas: Conjuntos de pares chave-valor
11 Map<String, int> estoque = {
12   "Maçã": 10,
13   "Banana": 20
14 };
15 print(estoque["Maçã"]); // 10
```

Esses tipos avançados ajudam a organizar e lidar com dados de um jeito mais eficiente, principalmente em projetos maiores. Saber escolher o tipo certo pode deixar seu código mais claro e rápido.

Analogia

“Os tipos em Dart são as "etiquetas" que descrevem o que cada variável representa. Pense em um supermercado, onde cada corredor é identificado: números, textos, valores lógicos etc.”



Interpolação

Interpolação é uma maneira fácil de combinar texto com variáveis. Em Dart, você pode usar o símbolo `$` para inserir o valor de uma variável dentro de uma **String**.

```
1 String nome = "Ana";
2 int idade = 28;
3 print("Meu nome é $nome e eu tenho ${idade + 2} anos.");
4 // Resultado: Meu nome é Ana e eu tenho 30 anos.
```

A **interpolação** é especialmente útil para gerar mensagens dinâmicas ou formatar dados para exibição.

Analogia

“Interpolação é como fazer uma costura. É como juntar pedaços de tecido (texto e variáveis) com uma linha (interpolação) para criar uma peça completa, que faz sentido e se encaixa perfeitamente.”



Operadores

Operadores são símbolos que realizam operações sobre valores.

Operadores Aritméticos

Esses operadores são usados para realizar operações matemáticas.

```
1 int soma = 5 + 8;
2 int subtracao = 5 - 2;
3 int multiplicacao = 2 * 6;
4 double divisao = 3 / 8;
5 int divisaoInteira = 3 ~/ 3; // (divisão inteira, retorna o resultado sem casas decimais)
6 int modulo = 2 % 9; // módulo, retorna o resto da divisão
```

Operadores Relacionais

Esses operadores comparam valores e retornam um valor booleano (true ou false).

```
1 int a = 5;
2 int b = 12;
3 print(b == a); // Igual a           // Saída: false
4 print(b != a); // Diferente de      // Saída: true
5 print(a < b); // Maior que          // Saída: true
6 print(a > b); // Menor que          // Saída: false
7 print(b >= a); // Maior ou igual a      // Saída: true
8 print(b <= a); // Menor ou igual a      // Saída: false
```

Operadores Lógicos

Esses operadores são usados para combinar expressões booleanas.

```
1 var a = true;
2 var b = false;
3
4 print(a && b); // E lógico // Saída: false
5 print(a | b); // OU lógico // Saída: true
6 print(!a); // Não lógico // Saída: false
```

Operadores de Atribuição

Esses operadores são usados para atribuir valores às variáveis.

```
1 int a = 5;
2 int b = 12;
3 double c = 45;
4 double d = 78;
5 print(b = a); // Atribuição simples // 5
6 print(b += a); // Atribui o valor somado // 10
7 print(a -= b); // Atribui o valor subtraído // -5
8 print(a *= b); // Atribui o valor multiplicado // -50
9 print(c /= d); // Atribui o valor dividido // 0.576...
10 print(b %= a); // Atribui o valor do módulo // 10
```

Operadores de Incremento e Decremento

Esses operadores aumentam ou diminuem o valor de uma variável.

```
1 int a = 5;
2
3 a++; // Incremento, aumenta o valor de 1
4 print(a); // Saída: 6
5 a--; // Decremento. diminui o valor de 1
6 print(a); // Saída: 5
```

Operadores de Condicional (Ternário)

É uma maneira curta de fazer uma comparação e decidir o que fazer, tudo em uma única linha. Em outras palavras, seria uma forma compacta de escrever uma expressão condicional.

```
1 var idade = 18;
2
3 // condição ? valor_se_verdadeiro : valor_se_falso
4 var resultado = (idade >= 18) ? 'Maior de Idade' : 'Menor de Idade';
5 print(resultado); // Saída: Maior de Idade
```

Operadores de Tipo

Esses operadores servem para checar o tipo de uma variável ou garantir que ela é de um tipo específico. É como perguntar: 'Esse valor é realmente do tipo que eu espero?' ou 'Não é desse tipo, certo?'

```
1 var nome = 'Dart';
2 print(nome is String);
3 // Verifica se o Objeto é de um tipo específico
4 // Saída: true
5 print(nome is! int);
6 // Verifica se o Objeto não é de um tipo específico
7 // Saída: true
```

Operadores de Spread (ou de Expansão)

Esses operadores ajudam a expandir ou combinar coleções.

```
1 List<int>? listaNula = null;
2 var lista1 = [1, 5, 10];
3 // '...' Espalha todos os elementos de uma coleção
4 var lista2 = [5, 7, ...lista1]; // Combina lista1 com lista2
5 print(lista2); // Saída: [5, 7, 1, 5, 10]
6
7 // Usando o operador ...? para evitar erros com uma lista null
8 var listaCombinada = [0, ...? listaNula];
9 print(listaCombinada); // Saída: [0]
```

Operadores de Cascade (.. e ?..)

Permitem fazer várias coisas com um objeto, uma após a outra, sem precisar repetir o nome dele.

```
1 void main() {
2   Pessoa? pessoa = null;
3
4   pessoa?.nome = 'Jão'..falar();
5   // '?..' Operações seguras, evita erro caso o objeto seja null
6
7   pessoa = Pessoa()
8     ..nome = 'Jao'
9     ..idade = 25
10    ..falar();
11   // ',,' Operações encadeadas
12
13   print(pessoa);
14 }
15
16 class Pessoa {
17   String? nome;
18   int? idade;
19
20   void falar() {
21     print('Meu nome é $nome e eu tenho $idade anos');
22   }
23 }
```

snappify.co

Notação Ponto

A notação ponto é simplesmente a forma de acessar ou interagir com as partes de um objeto, seja para pegar um valor (propriedade) ou pedir para ele realizar uma ação (método).

```
1 // Acessando Propriedades e Chamando Métodos
2 void main() {
3     var pessoa = Pessoa();
4
5     pessoa.falar(); // Chamando o método 'falar' do objeto 'pessoa'
6     print(pessoa.nome); // Acessando a propriedade 'nome' do objeto 'pessoa'
7 }
8
9 class Pessoa {
10     String nome = 'João';
11
12     void falar() {
13         print('Olá! João');
14     }
15 }
```

Analogia

“Os operadores são os pincéis que você usa para fazer ajustes no seu código (como somar, multiplicar), e a notação ponto é a tela onde você aplica esses ajustes em um objeto, como se estivesse criando uma pintura.”



Generics

Generics em Dart são como moldes reutilizáveis que ajudam a criar estruturas de dados ou funções que funcionam com diferentes tipos, sem precisar repetir o código para cada tipo específico.

```
1 // '<String>' Generics são como etiquetas
2 List<String> nomes = ["Ana", "João", "Maria"];
3 print(nomes[0]); // Saída: Ana
4
5 Map<String, int> pontuacao = {"Alice": 10, "Bob": 15};
6 print(pontuacao["Bob"]); // Saída: 15
```

Os generics são essenciais em projetos maiores, permitindo a criação de coleções reutilizáveis e adaptáveis.

Analogia

“Pense nos **generics** como caixas organizadoras que podem guardar qualquer tipo de item, seja um número, texto ou objeto. Quando você define uma caixa (classe, lista ou função), usa generics para especificar o que ela pode armazenar, garantindo segurança e flexibilidade ao mesmo tempo.”



02

Estruturas de Controle

Tomando Decisões

If-Else

If-Else é uma estrutura de controle usada para tomar decisões no código. Com ela, você verifica se uma condição é verdadeira e executa um bloco de código. Se a condição for falsa, pode executar outro bloco de código alternativo.

```
1 void main() {  
2     int idade = 18;  
3  
4     // Verifica se a idade é maior ou igual a 18  
5     if (idade >= 18) {  
6         print('Você é maior de idade.'); // Executa esta linha se a condição for verdadeira  
7     } else {  
8         print('Você é menor de idade.'); // Executa esta linha se a condição for falsa  
9     }  
10 }
```

O **if-else** funciona como uma escolha: "*Se isso for verdade, faça isso; senão, faça outra coisa.*" É essencial para criar programas que se adaptem a diferentes situações.

Analogia

“Se você está dirigindo e vê uma bifurcação, toma uma decisão baseado em uma condição. Esse é o "if-else" em ação.”



Laço For

O laço for é uma estrutura de repetição usada para executar um bloco de código várias vezes. Ele é ideal quando você sabe quantas vezes a repetição deve ocorrer.

```
1 void main() {  
2     // O laço for começa aqui  
3     // i = 0: Inicializa a variável de controle 'i' com o valor 0  
4     // i < 5: Define a condição para continuar o laço (enquanto 'i' for menor que 5)  
5     // i++: Incrementa o valor de 'i' em 1 a cada iteração  
6     for (int i = 0; i < 5; i++) {  
7         print('Contagem: $i'); // Exibe o valor atual de 'i' no console  
8     }  
9 }
```

O for funciona como um cronograma: você define onde começa, onde termina e o que acontece a cada volta, permitindo repetir ações de forma controlada.

Analogia

“O **for** funciona como um cronograma: você define onde começa, onde termina e o que acontece a cada volta, permitindo repetir ações de forma controlada.



While

Enquanto houver condições para seguir em frente, o loop "while" continua. Ele é útil quando você não sabe exatamente quantas vezes precisará repetir uma ação.

```
1 void main() {  
2     int contador = 0;  
3  
4     while (contador < 5) {  
5         print('Contagem: $contador');  
6         contador++; // Incrementa o contador a cada repetição  
7     }  
8 }
```

Analogia

“O while é como um semáforo para pedestres: Enquanto o botão de atravessar está pressionado (condição verdadeira), o sinal fica vermelho para os carros. Quando o botão não está mais pressionado (condição falsa), o sinal volta ao normal.”



Switch

Para situações com várias opções, o "switch" é como escolher entre diferentes sabores de sorvete. Ele ajuda a lidar com múltiplos casos de maneira organizada.

```
1 void main() {  
2     String cor = "vermelho"; // A variável 'cor' é definida como "vermelho"  
3  
4     // O switch verifica o valor da variável 'cor' e executa o código correspondente  
5     switch (cor) {  
6         case "vermelho": // Se 'cor' for igual a "vermelho"  
7             print("Pare!"); // Exibe "Pare!"  
8             break; // Interrompe o switch, saindo da estrutura  
9         case "verde": // Se 'cor' for igual a "verde"  
10            print("Siga!"); // Exibe "Siga!"  
11            break; // Interrompe o switch, saindo da estrutura  
12        default: // Se 'cor' não corresponder a nenhum dos casos anteriores  
13            print("Atenção!"); // Exibe "Atenção!"  
14    }  
15 }
```

O bloco `default` é importante para lidar com casos não previstos, garantindo que o programa se comporte de forma robusta.

Analogias

“O **switch** é como um **cardápio de restaurante**. Cada prato no cardápio representa uma opção de escolha. Você escolhe o prato que deseja e o garçom traz exatamente o que corresponde à sua escolha. Se o prato que você quer não está no cardápio, o garçom traz a opção “Prato do dia” (o **default**). Assim, o **switch** facilita a escolha entre várias opções de forma rápida e organizada.”

“As estruturas de controle são como placas de sinalização em uma estrada: ajudam você a decidir que caminho seguir ou repetir um trajeto.”



03

Funções

As Pequenas Máquinas

Função Básica

Imagine um botão que você aperta para cumprimentar alguém. A função faz o trabalho.



```
1 void saudacao(String nome) {  
2   // Define a função 'saudacao' que recebe um parâmetro 'nome' do tipo String  
3   print("Olá, $nome!");  
4   // Exibe a mensagem "Olá, nome!" no console, substituindo $nome pelo valor passado  
5 }  
6  
7 main(){  
8   saudacao("Carlos");  
9 }  
10 // Chama a função 'saudacao' e passa "Carlos" como argumento para o parâmetro 'nome'
```

sna

Função com Retorno

Essas funções são como uma calculadora: você insere valores e recebe um resultado.

```
 1 void main() {  
 2   print(calcularArea(5, 3));  
 3   // Chama a função 'calcularArea' com os valores 5 e 3, e imprime o resultado, que será 15.0  
 4 }  
 5  
 6 double calcularArea(double largura, double altura) {  
 7   // Define a função 'calcularArea', que recebe dois parâmetros do tipo 'double' (largura e altura)  
 8   return largura * altura;  
 9   // Retorna o valor da área, multiplicando a largura pela altura  
10 }
```

Funções Lambda

Rápidas e diretas, como notas adesivas para tarefas simples. Funções lambda são muito úteis para realizar tarefas simples ou passar funções como argumentos. Por exemplo, imagine que você quer filtrar uma lista ou realizar uma ação específica em cada elemento:

```
1 void main() {  
2  
3     var soma = (int a, int b) => a + b;  
4     // Define uma função anônima que recebe dois parâmetros (a e b) e retorna a soma deles  
5     print(soma(3, 4));  
6     // Chama a função 'soma' com os valores 3 e 4, e imprime o resultado, que será 7  
7  
8     List<int> numeros = [1, 2, 3, 4];  
9     // Cria uma lista chamada 'numeros' com os valores 1, 2, 3 e 4  
10    var numerosDuplicados = numeros.map((n) => n * 2).toList();  
11    // Usa o método 'map' para multiplicar cada número da lista por 2,  
12    // e converte o resultado de volta para uma lista  
13    print(numerosDuplicados);  
14    // Imprime a lista 'numerosDuplicados', que será [2, 4, 6, 8]  
15 }
```

Nesse exemplo, usamos uma função lambda para duplicar os números de uma lista. Isso mostra como lambdas podem facilitar a escrita de códigos mais concisos e expressivos, especialmente em operações funcionais como mapear ou filtrar elementos.

Além disso, lambdas são frequentemente usadas em frameworks para callbacks, como ações a serem executadas após eventos.

Analogia

“Funções são como pequenos robôs programados para realizar tarefas específicas sempre que você precisar. Elas tornam o código mais modular e reutilizável.”



04

Programação Orientada a Objetos

Moldando o Mundo

O que são Classes e Objetos?

Pense em uma classe como um molde para criar objetos. A classe define o que um objeto deve ter (atributos) e o que ele pode fazer (métodos). Já o objeto é uma instância de uma classe, ou seja, a materialização desse molde na prática.

- Atributos (ou propriedades) são as características do objeto. Eles podem ser, por exemplo, o nome de uma pessoa ou a cor de um carro.
- Métodos são as ações que os objetos podem realizar, como andar, falar ou mover-se.



```
1 class Carro {  
2     String cor; // Atributo: cor do carro  
3     String modelo; // Atributo: modelo do carro  
4  
5     // Método: define o que o carro pode fazer  
6     void acelerar() {  
7         print('O carro está acelerando!');  
8     }  
9  
10    // Método: define outra ação que o carro pode fazer  
11    void buzinar() {  
12        print('Beep beep!');  
13    }  
14 }  
15  
16 void main() {  
17     var meuCarro = Carro(); // Criação de um objeto da classe Carro  
18     meuCarro.cor = 'Vermelho'; // Atribuindo um valor ao atributo 'cor'  
19     meuCarro.modelo = 'Fusca'; // Atribuindo um valor ao atributo 'modelo'  
20  
21     // Imprime as características do carro  
22     print('Meu carro é um ${meuCarro.modelo} de cor ${meuCarro.cor}');  
23     meuCarro.acelerar(); // Chama o método 'acelerar'  
24     meuCarro.buzinar(); // Chama o método 'buzinar'  
25 }
```

EXPLORANDO DART - VITÓRIA SANTANA

Analogia

“Se a classe for uma receita de bolo, o objeto é o bolo que você realmente assa. A receita define como o bolo deve ser feito, mas o bolo é o produto final.”



Construtores

Os construtores são como recepcionistas que ajudam a criar novos objetos. Eles inicializam os atributos dos objetos quando criados. A maneira mais comum de usar um construtor é fornecer valores para os atributos no momento da criação do objeto.

```
1 class Carro {  
2     String cor;  
3     String modelo;  
4  
5     // Construtor: inicializa os atributos com valores passados  
6     Carro(this.cor, this.modelo);  
7  
8     void acelerar() {  
9         print('O carro $modelo está acelerando!');  
10    }  
11 }  
12  
13 void main() {  
14     var meuCarro = Carro('Azul', 'Civic');  
15     // Usando o construtor para inicializar atributos  
16     meuCarro.acelerar();  
17     // Chama o método 'acelerar'  
18 }
```

Herança

A herança permite que uma classe herde atributos e métodos de outra classe. Isso facilita a reutilização de código e a criação de hierarquias. Pense em herança como passar uma receita de bolo de geração em geração, mas com a possibilidade de adicionar novos ingredientes.

```
 1 class Veiculo {  
 2     String modelo;  
 3  
 4     Veiculo(this.modelo);  
 5  
 6     void mover() {  
 7         print('O veículo $modelo está se movendo!');  
 8     }  
 9 }  
10  
11 class Carro extends Veiculo {  
12     Carro(String modelo) : super(modelo);  
13     // Chama o construtor da classe pai  
14  
15     void buzinar() {  
16         print('Bip Bip!');  
17     }  
18 }  
19  
20 void main() {  
21     var carro = Carro('Fusca');  
22     carro.mover(); // Método herdado da classe Veiculo  
23     carro.buzinar(); // Método específico da classe Carro  
24 }
```

Encapsulamento

Encapsulamento significa esconder os detalhes internos de um objeto e fornecer apenas uma interface pública para interagir com ele. Isso ajuda a proteger os dados e a controlar o acesso a eles.



```
1 class ContaBancaria {  
2     double saldo = 0;  
3  
4     // Método para depositar dinheiro, só pode ser feito por esse método  
5     void depositar(double valor) {  
6         if (valor > 0) {  
7             saldo += valor;  
8         }  
9     }  
10  
11    // Método para verificar o saldo, não permite alterar diretamente  
12    double consultarSaldo() {  
13        return saldo;  
14    }  
15 }  
16  
17 void main() {  
18     var conta = ContaBancaria();  
19     conta.depositar(100); // Deposita 100 reais  
20     print('Saldo atual: ${conta.consultarSaldo()}'); // Exibe o saldo  
21 }
```

Polimorfismo

Polimorfismo permite que um objeto se comporte de maneiras diferentes, dependendo de como é utilizado. É como ter um único botão que, ao ser pressionado, pode fazer várias coisas diferentes, dependendo da situação.

```
 1 class Animal {  
 2     // Define o método 'fazerSom' que será usado por qualquer classe que herde de 'Animal'  
 3     void fazerSom() {  
 4         print('Som de animal'); // Exibe uma mensagem genérica  
 5     }  
 6 }  
 7  
 8 class Cachorro extends Animal {  
 9     // A classe 'Cachorro' herda de 'Animal' e sobrescreve o método 'fazerSom'  
10     @override  
11     void fazerSom() {  
12         print('Latido!'); // Exibe uma mensagem específica para o cachorro  
13     }  
14 }  
15  
16 class Gato extends Animal {  
17     // A classe 'Gato' também herda de 'Animal' e sobrescreve o método 'fazerSom'  
18     @override  
19     void fazerSom() {  
20         print('Miau!'); // Exibe uma mensagem específica para o gato  
21     }  
22 }  
23  
24 void main() {  
25     var meuCachorro = Cachorro(); // Cria um objeto da classe 'Cachorro'  
26     var meuGato = Gato(); // Cria um objeto da classe 'Gato'  
27  
28     meuCachorro.fazerSom(); // Chama o método 'fazerSom' do objeto 'meuCachorro', que exibe "Latido!"  
29     meuGato.fazerSom(); // Chama o método 'fazerSom' do objeto 'meuGato', que exibe "Miau!"  
30 }
```

Analogia

“A Programação Orientada a Objetos (POO) é como construir um mundo em que tudo é um “objeto”, com suas próprias características e comportamentos. Em vez de tratar tudo como uma sequência de comandos, você organiza o código em partes que representam entidades do mundo real. Essas partes são classes e objetos.”



AGRADECIMENTOS



OBRIGADA POR LER ATÉ AQUI



Este Ebook foi gerado por IA, revisado e diagramado por humano.

Gostaria de expressar minha sincera gratidão ao Felipe Aguiar, da DIO, pelas aulas incríveis e pelas valiosas dicas. Agradeço também à DIO, CAIXA e Microsoft, pelo BootCamp repleto de conteúdo rico e relevante. E, claro, um agradecimento especial à minha maravilhosa namorada Júlia, cuja visão e dicas de design tornaram este e-book ainda mais bonito e agradável de ler.

Muito obrigado a todos pelo apoio e contribuição e muito obrigada a você que chegou até aqui!



MEU [GITHUB](#)



MEU [LINKEDIN](#)

IAs utilizadas

- ✓ <https://chatgpt.com/>
- ✓ <https://www.bing.com/images/create?cc=br>
- ✓ <https://www.imagine.art/>