Apostila de Funções em Dart/Flutter

Este material didático aborda os principais conceitos sobre funções na linguagem de programação Dart, comumente utilizada com o framework Flutter. Aqui você encontrará exemplos práticos para cada tipo de função, desde a mais simples até conceitos mais avançados como *higher-order functions*.

1. Função Simples (Sem Parâmetros, Sem Retorno)

Objetivo: Criar uma função que executa uma ação pré-definida sem a necessidade de receber dados externos e sem retornar nenhum valor.

Exercício: Crie uma função chamada exibirBoasVindas que não recebe nenhum parâmetro e, quando chamada, simplesmente imprime a mensagem "Bem-vindo ao mundo do Flutter!" no console.

Exemplo de Código:

```dart
// Declaração da função 'exibirBoasVindas'
void exibirBoasVindas() {
  print("Bem-vindo ao mundo do Flutter!");
}

void main() {
  // Chamada da função
  exibirBoasVindas();
}
```

Explicação: A palavra-chave *void* indica que a função não retorna nenhum valor. Como não há nada entre os parênteses *()*, a função não aceita parâmetros. A ação de imprimir no console é executada toda vez que *exibirBoasVindas()* é chamada.

2. Função com Parâmetros (Sem Retorno)

Objetivo: Escrever uma função que recebe dados (parâmetros) para processar, mas não retorna um valor.

Exercício: Escreva uma função chamada *saudacaoPersonalizada* que aceita um parâmetro do tipo *String* chamado *nome*. A função deve imprimir no console a mensagem "Olá, [nome]! Tenha um ótimo dia.".

Exemplo de Código:

```
// Função que aceita um parâmetro 'nome' do tipo String
void saudacaoPersonalizada(String nome) {
  print("Olá, $nome! Tenha um ótimo dia.");
}

void main() {
  // Chamando a função e passando o argumento "Ana"
  saudacaoPersonalizada("Ana");
}
```

Explicação: A função *saudacaoPersonalizada* tem um parâmetro *String nome* definido em sua assinatura. Ao chamá-la, você deve fornecer um argumento (um valor do tipo

String, como "Ana"), que será utilizado dentro da função. A interpolação de string *"$nome"* insere o valor da variável diretamente no texto.

3. Função com Retorno (Sem Parâmetros)

Objetivo: Criar uma função que, ao ser executada, retorna um valor que pode ser armazenado em uma variável ou usado em outras operações.

Exercício: Crie uma função chamada *obterAnoAtual* que não precisa de parâmetros, mas retorna o ano atual como um valor *int*. Chame a função e armazene o resultado em uma variável, depois imprima essa variável.

Exemplo de Código:

```
// Função que retorna um valor do tipo
int
int obterAnoAtual() {
  // Em uma aplicação real, você poderia
usar uma biblioteca de data/hora.
  // Para este exemplo, vamos retornar
um valor fixo.
  return 2025;
}

void main() {
  // A variável 'ano' recebe o valor
retornado pela função
  int ano = obterAnoAtual();
  print("O ano atual é: $ano");
}
```

Explicação: O tipo de retorno (*int*) é especificado antes do nome da função.

A palavra-chave *return* é usada para enviar o valor de volta para onde a função foi chamada.

4. Função com Parâmetros e Retorno

Objetivo: Desenvolver uma função que recebe dados, realiza um processamento com eles e retorna um resultado.

Exercício: Desenvolva uma função chamada *somarDoisNumeros* que recebe dois parâmetros do tipo *double*, *a* e *b*.

A função deve retornar a soma desses dois números.

Exemplo de Código:

```
// Função que recebe dois doubles e
retorna a soma
```

```
double somarDoisNumeros(double a, double
b) {
  return a + b;
}

void main() {
  double resultado =
somarDoisNumeros(10.5, 22.3);
  print("O resultado da soma é:
$resultado");
}
```

Explicação: Esta é a forma de função mais completa e comum. Ela combina a recepção de parâmetros (*double a*, *double b*) com o retorno de um valor (*double*). O resultado da operação *a* + *b* é retornado.

5. Função de Seta (Arrow Function)

Objetivo: Utilizar uma sintaxe mais concisa para funções que contêm apenas uma única expressão.

Exercício: Crie uma função *multiplicar* que aceita dois inteiros e retorna sua multiplicação, tudo em uma única linha usando a sintaxe =>.

Exemplo de Código:

```
// Arrow function para multiplicar dois
números
[cite_start]int multiplicar(int a, int
b) => a * b; [cite: 49]

void main() {
  int resultado = multiplicar(5, 10);
```

```
  print("O resultado da multiplicação é: $resultado");
}
```

Explicação: A sintaxe => *expressao* é um atalho para *{ return expressao;}*

É ideal para funções simples e de uma única linha, tornando o código mais limpo e legível.

6. Função com Parâmetros Nomeados Opcionais

Objetivo: Criar funções flexíveis onde alguns parâmetros não são obrigatórios e podem ser passados pelo nome, tornando a chamada da função mais clara.

Exercício: Crie uma função *exibirInfoProduto* que aceite um *String* obrigatório para o nome do produto e dois parâmetros nomeados opcionais: *preco* (um *double*) e *categoria* (uma *String*). A função deve imprimir as informações do produto, tratando os casos em que os valores opcionais não são fornecidos.

Exemplo de Código:

```
// Uso de {} para parâmetros nomeados e '?' para indicar que são anuláveis (opcionais)
void exibirInfoProduto(String nome, {double? preco, String? categoria}) {
  String precoInfo = preco != null ? "R\$ ${preco.toStringAsFixed(2)}" : "Não informado";
  String categoriaInfo = categoria ?? "Não informada"; // Outra forma de tratar nulos

  print("Produto: $nome, Preço: $precoInfo, Categoria: $categoriaInfo");
```

```
}

void main() {
  exibirInfoProduto("Caneta", preco:
2.50, categoria: "Papelaria");
  exibirInfoProduto("Caderno", preco:
15.00);
  exibirInfoProduto("Borracha");
}
```

Explicação: Parâmetros envolvidos por chaves

{} são nomeados. O *?* após o tipo (*double?)* indica que a variável pode ter um valor nulo (*null*), tornando-a opcional. Ao chamar a função, você especifica o nome do parâmetro (ex: *preco: 2.50*), o que aumenta a clareza.

7. Função com Parâmetros Posicionais Opcionais

Objetivo: Definir funções onde os parâmetros opcionais são passados pela sua posição na lista de argumentos, e não pelo nome.

Exercício: Escreva uma função *criarResumo* que aceite *titulo* e *autor* como *String* e um parâmetro posicional opcional resenha. A função deve retornar uma

String formatada de maneiras diferentes, dependendo se a resenha foi ou não fornecida.

Exemplo de Código:

```
// Uso de [] para parâmetros posicionais
opcionais
String criarResumo(String titulo, String
autor, [String? resenha]) {
  String resultado = "'$titulo' por
$autor.";
  if (resenha != null) {
    resultado += " $resenha";
```

```
  }
  return resultado;
}

void main() {
  String resumo1 = criarResumo("O Senhor dos Anéis", "J.R.R. Tolkien", "Uma obra-prima da fantasia.");
  String resumo2 = criarResumo("1984", "George Orwell");

  print(resumo1);
  print(resumo2);
}
```

Explicação: Parâmetros envolvidos por colchetes *[]* são posicionais e opcionais. Eles devem ser os últimos na lista de parâmetros. A chamada da função não exige o nome do parâmetro, apenas o valor na posição correta.

8. Função Anônima (Closure)

Objetivo: Criar uma função sem um nome específico, geralmente para ser passada como argumento para outra função.

Exercício: Crie uma lista de *String* com alguns nomes. Utilize o método

forEach da lista para imprimir cada nome em maiúsculas, passando para ele uma função anônima.

Exemplo de Código:

```
void main() {
  List<String> nomes = ["ana", "bruno", "carla"];
```

```
  // 'forEach' espera uma função como
argumento
  nomes.forEach((nome) {
    print(nome.toUpperCase());
  });
}
```

Explicação: A parte *(nome) { print(nome.toUpperCase());}* é uma função anônima. Ela não tem nome e é definida diretamente no local onde é usada (como argumento para o

forEach). O *forEach* executa essa função para cada item da lista.

9. Função como Parâmetro (Higher-Order Function)

Objetivo: Escrever uma função que aceita outra função como parâmetro, permitindo a criação de código mais genérico e reutilizável.

Exercício: Crie uma função *filtrarLista* que aceite uma lista de inteiros e uma "função de teste". Essa função de teste deve receber um

int e retornar um *bool*. A função *filtrarLista* deve retornar uma nova lista contendo apenas os números que passam no teste. Teste-a com uma lista de 1 a 8, passando uma função que verifica se um número é par.

Exemplo de Código:

```
// 'filtrarLista' é uma Higher-Order
Function porque aceita 'teste' (uma
função) como parâmetro.
List<int> filtrarLista(List<int> lista,
bool Function(int) teste) {
  List<int> listaFiltrada = [];
  for (int numero in lista) {
    if (teste(numero)) {
      listaFiltrada.add(numero);
    }
```

```
  }
  return listaFiltrada;
}

// Função que será usada como "teste"
bool ehPar(int numero) {
  return numero % 2 == 0;
}

void main() {
  List<int> numeros = [1, 2, 3, 4, 5, 6,
7, 8];

  // Passando a função 'ehPar' como
argumento para 'filtrarLista'
  List<int> numerosPares =
filtrarLista(numeros, ehPar);

  print("Lista original: $numeros");
  print("Lista de números pares:
$numerosPares");
}
```

Explicação:

filtrarLista é uma *higher-order function* porque um de seus parâmetros, *teste*, é do tipo *Function*.

Especificamente, *bool Function(int)* define que a função teste deve aceitar um *int* e retornar um *bool*. Isso permite que a lógica de filtragem seja customizável. Poderíamos passar qualquer outra função de teste (como

ehImpar, *ehMaiorQueCinco*, etc.) sem alterar a função *filtrarLista*.