Overview Dart

A seguir será apresentado um ***overview*** (ou melhor dizendo, um resumão) dos **principais comandos e estruturas mais básicas** da linguagem **Dart**, para que você possa ter tudo junto no mesmo lugar sempre precisar se lembrar deles.

Dart online

https://dart.dev/#try-dart

Dart VSCode

Para criar arquivos **Dart**, basta criar um arquivo com a extensão '**.dart**' e abrir o arquivo criado no **VSCode** ou dentro do **VSCode:**

view > Command Pallete ... > Dart: new Project > Simple Console Application

Hello World

```dart
void main() {
// hello world Dart
  print('hello World Dart');
}
```

Leitura de Dados

A leitura de dados via console não funciona corretametne em todos os compiladores online.

No VSCode:

```dart
import 'dart:io';

void main() {
  print('\x1B[2J\x1B[0;0H');
  print('Entre com seu nome: ');
  var nome = stdin.readLineSync();

  print('Entre com o ano do seu nascimento: ');
  var anoNasc = stdin.readLineSync();
  var idade = 2022 - int.parse(anoNasc!);

  print('Olá $nome, \nSua idade é: $idade anos');
}
```

Variáveis

Em outras linguagens existe uma diferenciação entre tipos primitivos como números, booleanos, strings e instâncias de classes. No **Dart**, por outro lado, tudo é um objeto. Até mesmo ***null*** e ***funções*** são objetos.

```dart
void main() {
  // Formas tradicionais de atribuição

  String nome = 'Pedro';
  int idade = 25;
  bool estudante = true;
  double nota = 9.5;

  if (estudante) {
    print('Nome aluno: $nome');
    print('Idade: $idade');
    print('Nota: $nota');
  }

  // Outras formas de atribuição:

  // Variável texto será do tipo String
  var texto = 'Maria';
  // Aceita novas atribuições desde que sejam do mesmo
  // tipo da atribuição inicial
  texto = 'Ana';
  print(texto);

  // Variável dynamic aceita novas atribuições
  // de qualquer tipo
  dynamic dinamica = 10;
  dinamica = 'José';
  dinamica = true;
  print(dinamica);
}
```

Constantes

As constantes em **Dart** podem ser declaradas utilizando as palavras chave ***const*** ou ***final***. A diferença entre elas é que ao se declarar uma variável como **const**, ela se tora imutável em tempo de execução. No caso de se utilizar o **final** apesar da definição do seu valor ser permitido uma única vez (assim como no caso de ***const***), essa definição pode ser efetuada em tempo de execução (o que não é possível no caso do ***const***).

```dart
void main() {
/*
 * A seguinte declaração daria erro uma vez que não é possível
 * alterar o valor de um const em tempo de execução
 */
  // const dayOne = DateTime.now();

/*
 * A próxima atribuição é permitida uma única vez,
 * e não causa nenhum erro, uma vez que o final permite
 * sua atribuição em tempo de execução
 */
  final dayTwo = DateTime.now();
  print(dayTwo);
}
```

Null safety

Documentação oficial: https://dart.dev/null-safety

Null safety codelab: https://dart.dev/codelabs/null-safety

Operador (?)

Os tipos de variáveis são **non-nullable** (não anuláveis) por padrão no **Dart**, ou seja, variáveis em **Dart** não podem receber valores nulos a não ser que o desenvolvedor defina que podem.

Esta permissão é dada ao inserir '**?**' na declaração da variável:

```dart
void main() {
  int a;
  a = null;
  print('a is $a.');
}
```

Para corrigir o código acima, basta incluir **?** na declaração da variável **a**:

```dart
void main() {
  int? a;
  a = null;
  print('a is $a.');
}
```

Operador (!)

Tendo certeza que uma variável não será nula, é possível utilizar o operador **'!'**. Desta forma o Dart "entenderá" que esta variável nunca será nula, permitindo sua dali pra frente de forma "**segura**" (responsabilidade do desenvolvedor).

Caso o desenvolvedor não tenha certeza do que está fazendo, uma exceção pode ser lançada caso um valor nulo seja atribuído a esta variável.

```dart
int? couldReturnNullButDoesnt() => -3;

void main() {
  int? couldBeNullButIsnt = 1;
  List<int?> listThatCouldHoldNulls = [2, null, 4];

  int a = couldBeNullButIsnt;
  int b = listThatCouldHoldNulls.first; // first item in the list
  int c = couldReturnNullButDoesnt().abs(); // absolute value

  print('a is $a.');
  print('b is $b.');
  print('c is $c.');
}
```

Para corrigir o código acima:

```dart
int? couldReturnNullButDoesnt() => -3;

void main() {
  int? couldBeNullButIsnt = 1;
  List<int?> listThatCouldHoldNulls = [2, null, 4];

  int a = couldBeNullButIsnt;
  int b = listThatCouldHoldNulls.first!; // first item in the list
  int c = couldReturnNullButDoesnt()!.abs(); // absolute value

  print('a is $a.');
  print('b is $b.');
  print('c is $c.');
}
```

Palavra Chave: `late`

Quando a variável deve ser non-nullable (não anulável) mas não receberá seu valor imediatamente na sua declaração, é possível utilizar a palavra chave **`late`**.

A palavra **`late`** deve ser usada nos seguintes casos (responsabilidade do desenvolvedor):

- Quando não é desejado declarar um valor inicial para a variável
- Quando se deseja atribuir posteriormente o valor para a variável
- Quando se tem a certeza que a variável não será utilizada antes que seu valor seja devidamente atribuído

Caso o desenvolvedor se engane, será lançado um erro como exceção

```
class Meal {
  late String _description;

  set description(String desc) {
    _description = 'Meal description: $desc';
  }

  String get description => _description;
}

void main() {
  final myMeal = Meal();
  myMeal.description = 'Feijoada!';
  print(myMeal.description);
}
```

Listas

```
// criando uma Lista/array glogal tipada
List<String> letras = ['a', 'b', 'c', 'd', 'e'];

void main() {
  print(letras.first); // mostra primeiro elemento
  print(letras.last); // mostra o último elemento
  print(letras.isEmpty); // verifica se array está vazio
  print(letras.length); // mostra o tamanho do array

  // Acrescentando valores
  letras.add('f');
  letras.add('g');
  print(letras);

  // remove elemento na posição especificada
  letras.removeAt(2);
  print(letras);

  // verifica a existência de uma letra
  print(letras.contains('c'));

  // Acessando elementos pelos índices
  letras[0] = 'H';
```

```
  letras[1] = 'I';
  letras[2] = 'J';
  letras[3] = 'K';
  letras[4] = 'L';

  print(letras);

  // percorre e imprime cada elemento da lista pelo seu índice
  print('Letras na lista:');
  for (var i = 0; i < letras.length; i++) {
    print(letras[i]);
  }

  // percorre e imprime cada elemento da lista
  print('Letras na lista:');
  for (var letra in letras) {
    print(letra);
  }
}
```

Mapas

```
// criando um Map glogal tipado
Map<String, dynamic> contatos = {};

void main() {

  contatos['Ana'] = '9999-0000';
  contatos['Pedro'] = '8888-9898';
  contatos['João'] = '7878-1234';
  contatos['Maria'] = '9898-5151';

  print(contatos.values);
  print(contatos.keys);
  print(contatos);

  // verifica se existe o contato
  print(contatos.containsKey('Ana'));

  // verifica se existe o telefone especificado
  print(contatos.containsValue('9999-0000'));

  // adiciona um novo contato
  contatos['Jose'] = '7777-2525';

  // remove um contato existente
  contatos.remove('Pedro');

  print(contatos);

  // para percorrer um mapa e imprimir suas chaves
  print('Imprimindo as chaves de contatos');
  for (var contato in contatos.keys) {
    print(contato);
  }

  // para percorrer um mapa e imprimir seus valores
```

```
  print('Imprimindo os valores de cada chave de contatos');
  for (var contato in contatos.values) {
    print(contato);
  }

  print('Imprimindo chaves e valores de contatos');
  contatos.forEach((chave, valor) {
    print('$chave: $valor');
  });
}
```

Condicionais e operadores lógicos

Exemplo 1: Faça um programa que receba três valores A, B e C. O programa verifica se estes valores podem ser os comprimentos dos lados de um triângulo e, se forem, verificar se estes compõem um triângulo equilátero, isósceles ou escaleno. O algoritmo também deverá informar se não for um triângulo. Sabe-se que:

- O comprimento de cada lado de um triângulo é menor do que a soma dos comprimentos dos outros dois lados;
- O triângulo equilátero possui os comprimentos dos três lados iguais;
- O triângulo isósceles possui os comprimentos de dois lados iguais;
- O triângulo escaleno possui os comprimentos de seus três lados diferentes.

```
void main() {
  var ladoA = 10;
  var ladoB = 10;
  var ladoC = 10;

  if ((ladoA < ladoB + ladoC) &&
      (ladoB < ladoA + ladoC) &&
      (ladoC < ladoA + ladoB)) {
    if ((ladoA == ladoB) && (ladoA == ladoC)) {
      print('Equilátero');
    } else if ((ladoA == ladoB) || (ladoA == ladoC) || (ladoB == ladoC)) {
      print('Isósceles');
    } else {
      print('Escaleno');
    }
  } else {
    print('Não é triângulo!');
  }
}
```

Exemplo 2: Faça um programa que receba e apresente três valores inteiros em ordem crescente (do menor para o maior).

```dart
void main() {
  var a = 75;
  var b = 12;
  var c = 20;

  if (a < b) {
    if (b < c) {
      print('$a, $b, $c');
    } else if (a < c) {
      print('$a, $c, $b');
    } else {
      print('$c, $a, $b');
    }
  } else if (b < c) {
    if (a < c) {
      print('$b, $a, $c');
    } else {
      print('$b, $c, $a');
    }
  } else {
    print('$c, $b, $a');
  }
}
```

Exemplo 3: No dart é permitida a utilização do operador ternário:

```dart
void main() {
  var lampada = true;
  var estado = lampada ? 'Acesa' : 'Apagada';
  print(estado);
}
```

Exemplo 4: O Dart também oferece a estrutura switch-case:

```dart
switch (variavel) {
    case 1:
        // executa algo
    break;
    case 2:
        // executa algo
    break;
    default:
        // Não sendo case 1 ou case 2
    break;
}
```

Repetições

Exemplo 1: Faça um algoritmo que mostre a soma dos valores pares do intervalo de 0 a 10 utilizando ***for*** como estrutura de repetição.

```
void main() {
  var soma = 0;
  for (var i = 0; i < 11; i += 2) {
    soma = soma + i;
    print(i);
  }
  print('Soma dos pares até 10 = ' + soma.toString());
}
```

Exemplo 2: Repita o exemplo anterior utilizando ***while*** como estrutura de repetição.

```
void main() {
  var soma = 0;
  var i = 0;
  while (i < 11) {
    if (i % 2 == 0) {
      soma = soma + i;
      print(i);
    }
    i++;
  }
  print('Soma dos pares até 10 = ' + soma.toString());
}
```

Funções

Exemplo 1: Faça um programa que possua as seguintes funções

1. Calcule a soma de 3 números
2. Calcule a área de um círculo
3. Calcule a raiz quadrada de um número

```
import 'dart:math';

void main() {
  soma(1.5, 2.3, 3.4);
  areaCirculo(5);
  raiz(25);
}

void soma(double a, double b, double c) {
  var soma = a + b + c;
```

```
  print('A soma de $a + $b + $c = ' + soma.toStringAsFixed(2));
}

void areaCirculo(double a) {
  var area = pi * pow(a, 2);
  print('A área do circulo de raio $a = ' + area.toStringAsFixed(2));
}

void raiz(double a) {
  var raiz = sqrt(a);
  print('A raiz quadrada de $a = ' + raiz.toStringAsFixed(2));
}
```

Exemplo 2: Funções com parâmetros opcionais

```
void main() {
  var rendimento;

  // cálculo do rendimento com taxa padrão de 1,5%
  rendimento = calcularRendimento(100);
  print('Previsão de rendimento = R\$ ${rendimento.toStringAsFixed(2)}');

  // cálculo do rendimento com taxa alterada para 2,5%
  rendimento = calcularRendimento(100, 2.5);
  print('Previsão de rendimento = R\$ ${rendimento.toStringAsFixed(2)}');
}

double calcularRendimento(double valorAplicado, [double taxa = 1.5]) {
  return valorAplicado * taxa / 100;
}
```

Exemplo 3: Funções com parâmetros nomeados

```
void main() {
  var rendimento;

  // cálculo do rendimento com taxa padrão de 1,5%
  // observar que o valor aplicado é obrigatório (required)
  rendimento = calcularRendimento(valorAplicado: 100);
  print('Previsão de rendimento = R\$ ${rendimento.toStringAsFixed(2)}');

  // cálculo do rendimento com taxa alterada para 2,5%
  // observar que por serem nomeados os parâmetros pode ficar fora de ordem
  rendimento = calcularRendimento(taxa: 2.5, valorAplicado: 100);
  print('Previsão de rendimento = R\$ ${rendimento.toStringAsFixed(2)}');
}

double calcularRendimento({required double valorAplicado, double taxa = 1.5}) {
  return valorAplicado * taxa / 100;
}
```

Exemplo 4: Funções anônimas com parâmetros opcionais

```
void main() {
  // exemplo de função anônima modo verboso
  var rendimento1 = (double valorAplicado, [double taxa = 1.5]) {
    return valorAplicado * taxa / 100;
  };
  print('Previsão de rendimento = R\$ ${rendimento1(100).toStringAsFixed(2)}');

  // exemplo da mesma função anônima mais simplificada utilizando arrow function
  var rendimento2 = (valorAplicado, [taxa = 1.5]) => valorAplicado * taxa / 100;

  print(
      'Previsão de rendimento = R\$ ${rendimento2(100, 2.5).toStringAsFixed(2)}');
}
```

Classes

Exemplo 1: Declaração simples de uma classe

```
class Data {
  int? dia;
  int? mes;
  int? ano;
}

void main() {
  var data = Data();
  data.dia = 1;
  data.mes = 1;
  data.ano = 2000;

  print('${data.dia}/${data.mes}/${data.ano}');
}
```

Exemplo 2: Construtor de uma classe – modo tradicional

```
class Data {
  int? dia;
  int? mes;
  int? ano;

  Data(int dia, int mes, int ano) {
    this.dia = dia;
    this.mes = mes;
    this.ano = ano;
  }
}

void main() {
  var data = Data(1, 1, 2000);
  print('${data.dia}/${data.mes}/${data.ano}');
}
```

Exemplo 3: Construtor de uma classe – modo compacto (Dart)

```dart
class Data {
  int? dia;
  int? mes;
  int? ano;

  Data(this.dia, this.mes, this.ano);
}
void main() {
  var data = Data(1, 1, 2000);
  print('${data.dia}/${data.mes}/${data.ano}');
}
```

Exemplo 4: Construtor de uma classe – parâmetros opcionais

```dart
class Data {
  int? dia;
  int? mes;
  int? ano;
  Data([this.dia = 1, this.mes = 1, this.ano = 2000]);
}
void main() {
  var data = Data();
  print('${data.dia}/${data.mes}/${data.ano}');

  data = Data(10);
  print('${data.dia}/${data.mes}/${data.ano}');

  data = Data(10, 10);
  print('${data.dia}/${data.mes}/${data.ano}');

  data = Data(10, 10, 2020);
  print('${data.dia}/${data.mes}/${data.ano}');
}
```

Exemplo 5: Construtor de uma classe – parâmetros requeridos e opcionais nomeados

```dart
class Data {
  int? dia;
  int? mes;
  int? ano;
  Data({required this.dia, this.mes = 1, this.ano = 2000});
}
void main() {
  var data = Data(dia: 10);
  print('${data.dia}/${data.mes}/${data.ano}');

  data = Data(ano: 2020, mes: 10, dia: 10);
  print('${data.dia}/${data.mes}/${data.ano}');
}
```

Repare que no caso do parâmetro obrigatório (**required**) não deve ser atribuído um valor opcional como nos demais parâmetros **mês** e **ano**.

Exemplo 6: getters e setters

```
class Data {
  int? _dia;
  int? _mes;
  int? _ano;

  get dia => _dia;
  get mes => _mes;
  get ano => _ano;

  set dia(valor) => _dia = valor;
  set mes(valor) => _mes = valor;
  set ano(valor) => _ano = valor;
}

void main() {
  var data = Data();
  data.dia = 10;
  data.mes = 10;
  data.ano = 2000;

  print('${data.dia}/${data.mes}/${data.ano}');
}
```

Obs. Para que os atributos da classe fiquem realmente protegidos, devem ser precedidos de underline e não podem estar sendo acessados dentro do mesmo arquivo da classe.