Aula 2 - Introdução ao Dart

Desenvolvimento de aplicações híbridas com Flutter

22-23 de Outubro/21

Configurando o ambiente

Configurando o ambiente

O Framework do Flutter já vem com um executável do Dart. Ele possúi algumas funcionalidades para trabalhar com projetos na linguagem.

- *run* -> Executa o nosso código na VM do Dart
- *analyse* -> Analíse o código por problemas
- *create* -> Cria um novo projeto em Dart puro.
- *compile* -> Compila o código em um executável nativo (.exe, .elf)

dartpad

Introdução ao Dart

- Dart é uma linguagem criada pelo Google, usada principalmente no Flutter.
- É compilada (*Ahead of Time*), mas pode ser executada *Just In Time*, por meio de uma VM.
- Possui vários recursos de linguagens modernas como Type Safety, e Null Safety.

dartpad

Expressões, variáveis e constantes

Expressões, variáveis e constantes

Expressões

Tudo que pode representar um valor:

```
10
```

```
3 - 1
```

```
x
```

```
'Meu texto'
```

Expressões, variáveis e constantes

Operadores Aritméticos

`+ , - , * , / , ~/ , %`

```
2 + 2;  // 4

7 / 3; // 2.333

7 ~/ 3; // 2

10 % 3; // 1
```

Expressões, variáveis e constantes

Constantes

Constantes possuem valor conhecido em tempo de compilação.

```
const number = 10;
```

Final

Finais não possuem valor conhecido em tempo de compilação, mas só podemos atribuir o valor a elas uma única vez.

```
final number = 10;
```

Expressões, variáveis e constantes

Variáveis

Dart possui Type Safety, então é necessário definir o tipo da variável em tempo de compilação, ou utilizar um tipo `dynamic` (Não recomendado).

```
int number = 10;

double pi = 13.14159;
```

Expressões, variáveis e constantes

Tipos

Tudo em Dart são objetos. Tanto `double` quanto `int` extendem de `num`, que extende de Object.

```
int number = 10;

double pi = 13.14159;

...

num my_number = 13;

my_number = 7.5;
```

Expressões, variáveis e constantes

Checando o tipo em Runtime

```
const int number = 10;

number is int // true

number is num // true

number is double // false

number.runtimeType // int
```

Expressões, variáveis e constantes

Type Casting

```
const num number = 10;
number.isEven // ERROR

number is int // true
(number as int).isEven
```

Strings

Strings

Introdução

Strings no Dart são sequencias ou *collections* de caracteres Unicode-16.

```
const message = 'This is a message';
```

Toda vez que modificamos uma String, o valor antigo é descartado, e uma nova String é criada.

```
const message = 'This is a message’;

message = 'This is a new message’;
```

Strings

Concatenação

Podemos concatenar Strings separadas, mas o resultado será uma string completamente nova.

```
const message = 'My Name is';

const name = 'João';

print(message + ' ' + name);
```

Strings

StringBuffer

StringsBuffers escrevem no mesmo local de memoria, sem criar uma String inteiramente nova.

```
final message = StringBuffer();

message.write('My Name Is');

message.write(' João');

print(message.toString());
```

Strings

Interpolação

```
const name = 'João';
const message = 'My Name Is $name';
…
const oneThird = 1 / 3;
final message = 'One third is ${oneThird.toStringAsFixed(3)}';
```

Strings

Múltiplas Linhas

```
final message = '''
  My Name is João
  ''';
```

Strings

Outros exemplos

```
final message = ' My name is \n João';
final message2 = 'I \u2764 Dart!’; // Out: I ❤ Dart!
final message3 = 'In the spot! \u{1F3AF}’; // Out: In the spot! 🎯
final messageWithoutSpecialCharacters = r'My name is \n João';
```

Controle de fluxo

Controle de fluxo

Booleano

Valores booleanos podem assumir apenas dois valores, `true` e `false`.

```
const yes = true;
```

Controle de fluxo

Operadores Lógicos

! , == , != , <= , >= , &&, ||

```
const exampleAnd = (1 < 2) && (2 < 4); // true
const negation = !exampleAnd; // false
const exampleOr = (1 < 3) || (4 < 2); // false
const stringEquals = 'Dog' == 'Cat' // false
```

Controle de fluxo

If Else

Podemos encadear if’s e else’s no nosso código:

```
const condition1 = true;
const condition2 = true;
if (condition1) {
    print('Condition 1 is true’);
} else if (condition2) {
    print('Condition 2 is true');
} else {
    print('Nothing is true');
}
```

Controle de fluxo

Operador Ternário

O Operador Ternário é útil para simplificar comparações de If Else

```
const condition1 = true;

const result = (condition1) ? true : false;
```

Controle de fluxo

Switches

```
const animal = 'Dog';
switch (animal) {
  case 'Dog':
    print('It\'s a dog');
    break;
  case 'Cat':
    print('It\’s a cat');
    break;
  default:
    print('I don\’t know what it is');
}
```

Controle de fluxo

Switches com Enums

```
const animal = DomesticAnimals.cat;
switch (animal) {
  case DomesticAnimals.dog:
    print('It\'s a dog');
    break;
  case DomesticAnimals.cat:
    print('It\'s a cat');
    break;
  case DomesticAnimals.turtle:
    print('It\'s a turtle');
}
```

```
enum DomesticAnimals {
  dog,
  cat,
  turtle
}
```

Controle de fluxo

For Loop

Similar à sintaxe em C:

```
for (int i = 0; i < 10; i++) {
  print(i);
}
```

Controle de fluxo

For-In Loop

Usado para iterar por coleções:

```
const name = 'João';
for (var character in name.codeUnits) {
  print(character); // Prints each character code in the String
}
```

Controle de fluxo

For-Each Loop

Método de coleções para iterar sobre elas:

```
const name = 'João';
name.codeUnits.forEach((element) {
  print(element);
});
```

Controle de fluxo

While e Do-While Loops

O While loop também é similar ao C.

```
bool shouldLoopForever = true;

while (shouldLoopForever) {
  print('Looping');
}

do {
  print('Running');
} while(shouldLoopForever);
```

Funções

Funções

Introdução

Anatomia de uma função em Dart:

```
String message(String firstName, String lastName) {
  return 'Hi $firstName $lastName!’;
}
```

Uma Função que retorna uma String, e possui dois parâmetros posicionais.

Funções

Recomendações

Algumas boas praticas são recomendadas quando utilizamos funções:

- Evitar Side-Effects (Pure functions)
- Fazer apenas uma coisa (Principio da responsabilidade única)
- Escolher bons nomes.
    - Verbos para funções não puras. Ex: updateDatabase()
    - Pronomes para funções puras. Ex: temperature(), e não getTemperature()

Funções

Parâmetros com valores default

Podemos definir valores default para parâmetros de funções em Dart:

```
String message(String firstName, [String lastName = '', int? birthYear]) {
  final message = 'Hi $firstName $lastName!';
  return (birthYear == null) ? message : message + ', you were born in $birthYear’;
}

message('João', 'Victor', 1995); // VALIDO!
message('João', 'Victor'); // VALIDO!
message('João', 1995); // NÃO VALIDO
```

Funções

Parâmetros nomeados.

Parametros nomeados podem ser inseridos em qualquer ordem.

```
String message(String firstName, {String lastName = '', int? birthYear}) {
  final message = 'Hi $firstName $lastName!';
  return (birthYear == null) ? message : message + ', you were born in $birthYear’;
}

message('João', lastName: 'Victor', birthYear: 1995); // VALIDO!
message('João', lastName: 'Victor'); // VALIDO!
message('João', birthYear: 1995); // VALIDO!
```

Funções

Funções Anônimas.

Funções que por algum motivo, não precisam ser nomeadas:

```
String message(String firstName) {
  return 'Hi $firstName’;
}
(String firstName) {
  return 'Hi $firstName';
}
```

Funções

First-class citizens

No Dart, funções podem ser passadas para outras funções ou atribuídas à variáveis.

```
Function message = (String firstName) {
  return 'Hi $firstName';
};
print(message('João'));
```

Funções

Arrow Functions

Simplifica funções anonimas quando precisamos de apenas uma expressão:

```
Function message = (String firstName) => 'Hi $firstName';

print(message('João'));
```

Ou

```
String message(String firstName) =>  'Hi $firstName';
```

Funções

Escopo

O escopo é definido pelas chaves. Então tanto funções quanto loops e Ifs definem escopos.

```
const shouldPrint = true;
if (shouldPrint) {
  const variableInsideIf = ''; // Lives inside the if block scope
}
print(variableInsideIf) // ERROR
…
String message(String firstName) {
  final text = 'Hi $firstName'; // Lives in the function scope
  return text;
}
```

Classes

Classes

Uma classe simples

```
class User {
  int Id = 0;
  String name = '';
}

final user = User();
user.id = 1;
user.name = 'João';
```

Classes

Adicionando métodos

```
class User {
  int Id = 0;
  String name = '';

  String greet() => 'Hi, my name is $name';
}

final user = User();
print(user.greet());
```

Classes

Construtores

Existem varias formas de definir construtores para uma classe:

```
class User {
  int id = 0;
  String name = '';

  User(int id, String name) {
    this.id = id;
    this.name = name;
  }
}
```

Forma Padrão

Classes

Construtores

```
class User {
  int id = 0;
  String name = '';

  User(this.id, this.name);
}
```

Forma simplificada

Classes

Construtores

```
class User {
  final int id;
  final String name;

  const User(this.id, this.name);
}
```

Construtor constante. O Dart só criará uma instancia desse objeto.

Classes

Propriedades privadas

```
class User {
  final int _id;
  final String _name;

  const User(int id, String name): _id = id, _name = name;
}
```

Classes

Propriedades estáticas

```
class User {
  final int _id;
  final String _name;

  static const int TAG = "UserClass";

  const User(int id, String name): assert(id < 10), _id = id, _name = name;
```

Acessamos utilizando `User.TAG;`

Classes

Herança

```
class Person {
  final int id;
  final String name;

  const Person(this.id, this.name);
}

class Student extends Person {
  final int grade;
  Student(this.grade, int id, String name): super(id, name);
}
```

Linguagem Dart

- Mixin:

```dart
abstract class Person {
  void live(){}
}
mixin Runner {
  void run() {}
}
mixin Swimer {
  void swim() {}
}

class Joao extends Person with Runner, Swimer {
  void act() {
    live();
    run();
    swim();
  }
}
```

dartpad

Linguagem Dart

- Null safe dart:

```dart
String? content = null;
String? test;

printContentLowerCase() {
  print(content?.toLowerCase());
  print(content!.toLowerCase());
}

printContent() {
  print(content ?? "empty");
}

fixContent() {
  content ??= "";
}
```

dartpad

Listas

Linguagem Dart

- List, Set e Map:

```dart
List<String> list = ["a", "a", "b", "c"];
Set<String> set = {"1", "2", "3"};
Map<int, String> map = {1: "1", 2: "2"};

List list2 = ["a", true, "b", "c"];
var set2 = {"1", "2", "3", 4};
Map map2 = {1: "1", 2: false};
```

dartpad

Linguagem Dart

- Cascade:

```
list..add("5")
..add("6")
..add("7")
..add("8");
```

dartpad

Linguagem Dart

- Spread, filter, map, reduce:

```
List list3 = [...list, "d"];
```

```
var list = [1, 2, 3, 4, 5, 6];
var newList = list.where((element) => element< 4);

var strList = list.map((e) => e.toString());

var total = list.reduce((value, element) => value + element);
```

dartpad

Exercício 3

Criar um programa com uma lista de notas e filtrar pelas notas maiores que 8.5, passando uma função(com arrow function) já criada bool *minimum* que recebe um valor de ponto flutuante para nota.

dartpad

Linguagem Dart

- Tour na linguagem Dart
- Dart cheatsheet codelab
- Site da linguagem Dart