Aula 1. Conhecendo o DART e preparando nosso ambiente para o Flutter.

Na primeira aula vimos algumas características do DART, a linguagem que vai acompanhar a gente no Flutter.

A gente acessou o https://dartpad.dev/ e vimos os tipos de variáveis no dart.

```dart
void main() {
 // variables
 String howAmI = "Eu sou o Romulo";
 int myAge = 31;
 double myHeight = 1.80;
 bool isFlutterGood = true;
 num canBeAnInt = 20;
 num canBeADouble = 20.5;
 dynamic itCanBeAnything = "I here have super powers";
 var itShouldBeUsedCarefully = 5;
 List<dynamic> mixedVariablesList = ['Uma lista aqui cheia de coisas', 5, 8.7];
 Map<dynamic, dynamic> mixedVariablesMap = {'Meu valor é um int': 5, 8.0: 'isso aqui é um double'};
 var aSet = <String>{};

 var addToOuraSet = aSet.add(howAmI);


 print(howAmI);
 print(myAge);
 print(myHeight);
 print(isFlutterGood);
 print(canBeAnInt);
 print(canBeADouble);
 print(itCanBeAnything);
 print(itCanBeAnything.runtimeType);
 print(itShouldBeUsedCarefully);
 print(itShouldBeUsedCarefully.runtimeType);
 print(mixedVariablesList);
 print(mixedVariablesList[0]);
 print('tamanho da nossa lita ${mixedVariablesList.length}');
 print(mixedVariablesList.last);
 print(mixedVariablesMap);
 print(aSet);
 print(addToOuraSet);
}
```

Vimos, sem muito exemplo visual, final vs const vs late.

```dart
late String receiveValueLater;

void main() {
 //final vs const vs late

  final String aFinalString = 'usado mais em tempo de execução';
  const String aConstString = 'usada mais em tempo de compilação';
  receiveValueLater = 'usado quando a gente atribui um valor na variável após ser declarada';

  print(aFinalString);
  print(aConstString);
  print(receiveValueLater);

}
```

Usamos final ou const quando queremos falar para o computador que a variável não terá seu valor alterado, mas final está mais ligada ao tempo de execução (a variável está na cara da pessoa usuária) e const usada no momento de compilação – cuidado para não colocar “DateTime.Now()”, por exemplo.

Outro ponto importante é que instance variables (aquelas que são definidas numa classe) nunca podem ser const;

Late vem quando declaramos uma variável, mas ela vai ser inicializada após ser declarada. Ou quando iremos fazer uma inicialização “lazy”.

```
late String askedWithdrawal;

void main() {
 //final vs const vs late
 String withdrawalRequest(){
   return 'pediu um saque no valor de 32 reais';
 }

 //in this case, askedWithdrawal was initialized, therefore, our function is called
 askedWithdrawal = withdrawalRequest();

 print(askedWithdrawal);

}
```

If and else

```
void main() {

  //if and else

  bool flunked;
  String notFlunked = 'Passou de ano';

  if(notFlunked == 'Passou de ano'){
    flunked = false;
    print(flunked);
  } else {
    flunked = true;
    print(flunked);
  }
}
```

Operadores lógicos.

```dart
void main() {

  //logical operators

  /*
   * && and
   * || or
   * ! change to true or false
   */

  bool isAnxious;
  bool cantSleepAtNight = true;
  bool arrhythmia = true;

  bool notGoodForHealth;
  int sleepHours = 6;
  int hoursWorking = 8;

  bool isVeryTired = false;

  if(cantSleepAtNight && arrhythmia){
    isAnxious = true;
    print("é ansioso: $isAnxious");
  }

  if(sleepHours < 7 || hoursWorking > 8){
    notGoodForHealth = true;
    print('não é bom para saúde: $notGoodForHealth');
  }

  if(!isVeryTired){
    print('não estou muito cansado');
  }

}
```

Flow control

Switch case

```dart
late String askedWithdrawal;

void main() {
  //Switch case

  String metabolicDiseases = 'Diabetes';

  switch(metabolicDiseases){
    case 'Diabetes':
    print('é uma doença metabólica');
    break;
    case 'Pressão arterial alta':
    print('é uma doença metabólica');
    break;
    default:
    print('não é metabólica');
  }
}
```

For

```dart
void main() {

//for

  List<int> listOfInt = [1,2,3,4,5,6,7,8,9,10];

  for(var interger in listOfInt){
    print(interger+=1);
  }
}
```

While

```dart
void main() {
  //while
  var i = 1;
  while (i <= 10) {
    if(i%2==0){
       print(i);
    }
    i++;
  }
}
```

Aqui vimos simples exemplos de funções. Vimos como fazer named parameters e o uso do arrow function. Na arrow function não podemos colocar um if statement nele, mas podemos usar expressões de condição. Como o null-check.

```dart
void main() {
//Functions

  isOddOrEven(int value){
    if(value % 2 == 0){
      print('$value é par');
    } else{
      print('$value é ímpar');
    }

  }

  isOddOrEven(2);

  String checkNullAware(String? value) => value ?? 'valor veio null, então esse texto veio no lugar';

  var result = checkNullAware('25');

  print(result);

}
```

Mais uma vez, aqui vimos algo que já estamos bem acostumados. Criar classes, colocar um método nela e instanciar nosso objeto. Lembrando quando temos o “null-safety” podemos colocar um “?” dessa forma conseguimos prosseguir com nossa classe.

```
//Classes, methods and objects

class Person {

  String? name;
  int? age;

  void presentYourself(){
    print('meu nome é $name e minha idade é de $age');
  }
 }


void main() {
  var person = Person();
  print(person.name = 'Rômulo');
  print(person.age = 31);
  person.presentYourself();
}
```

Nessa parte a gente viu os construtores. Que são passados para nossa classe com o “this”. Você poderia printar de forma direta o “presentYourSelf()”, desta forma:

```
void main() {
  Person person = Person(name: 'Rômulo', age: 31);

  print(person.presentYourSelf());
}
```

```
//Constructors

class Person{
  final String? name;
  final int? age;
  Person({required this.name, required this.age});

  String presentYourSelf(){
    return 'meu nome é $name e minha idade é de $age';
  }
}



void main() {
  Person person = Person(name: 'Rômulo', age: 31);
  var mySelf = person.presentYourSelf();
  print(mySelf);
}
```

Um pouco de getters e setters;

```dart
class Heart{

  final bool systole;
  final bool dystole;
  String _isHealth = 'está saudável';

  String get howIsYourHeart => _isHealth;

  set setValue(String isHealth) => _isHealth = isHealth;

  Heart(this.systole, this.dystole);

}



void main() {

  Heart myHeart = Heart(true, false);
  myHeart.setValue = 'está bom';
  print(myHeart._isHealth);

}
```

Vimos que no dart não temos interface, aqui fazemos nossos contratos via classes abstratas. E o “implements” é responsável por fazer a implementação desse contrato.

```dart
//OOP
abstract class Artery {

  double diameter();
  String arteryFunction();

}

class FemoralArtery implements Artery {
  @override
  double diameter(){
    return 6.6;
  }
   @override
  String arteryFunction(){
   return 'levar o sangue para a grande cirucalação';
  }
 }



void main() {
  FemoralArtery femoralArtery = FemoralArtery();
  print(femoralArtery.diameter());
  print(femoralArtery.arteryFunction());
}
```

Já o extends, tem mais a função de herda alguma característica de uma super classe. Veja que aqui fizemos um extends de uma classe também abstrata.

```dart
//OOP
abstract class Mamal{
  String characteristics();

}

class Person extends Mamal{
  final String? name;

  Person(this.name);

  @override
  String characteristics(){
    return 'essa pessoa por nome de $name é um mamífero';
  }
}


void main() {
  Person person = Person('Rômulo');
  var characteristic = person.characteristics();
  print(characteristic);

}
```