FLUTTER

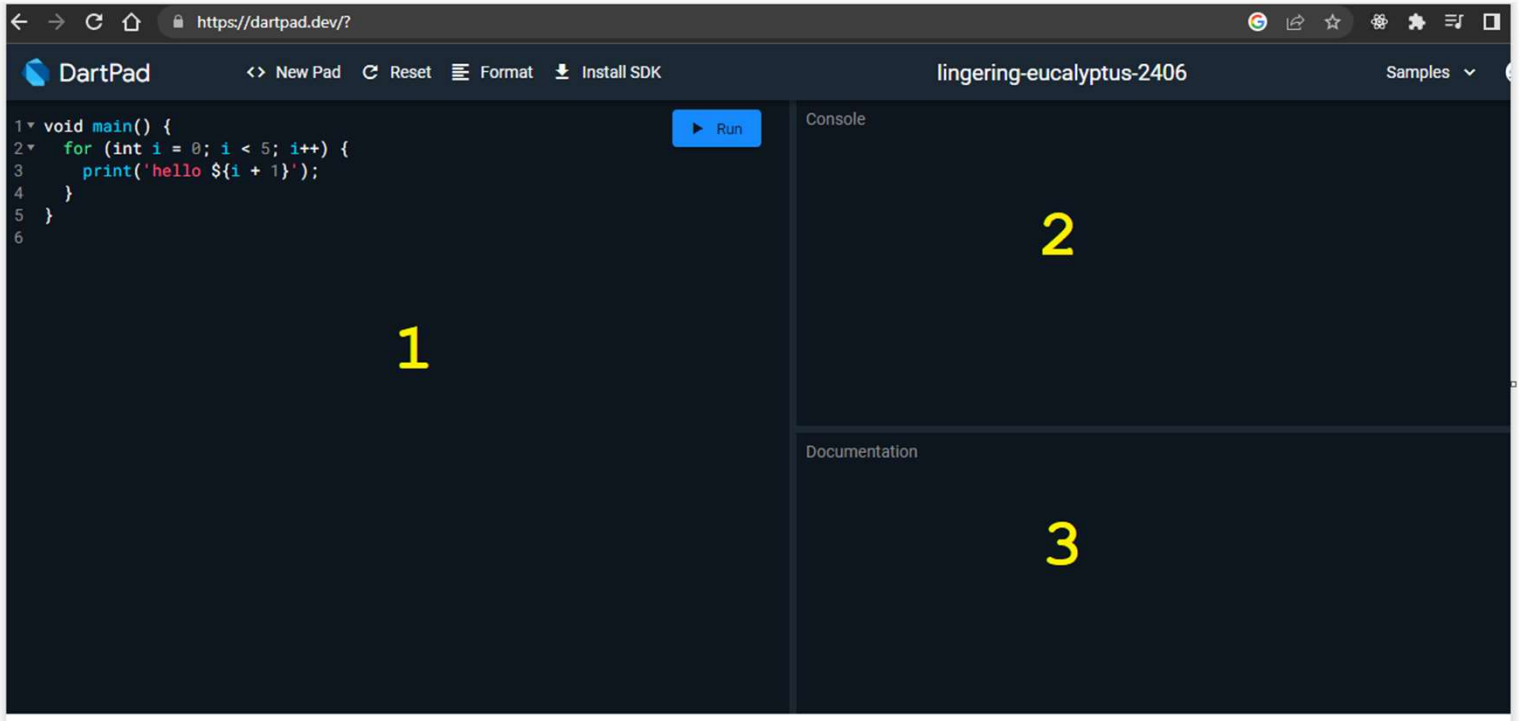
FLUTTER #10

Linguagem DART

Dart é uma linguagem de programação multi-paradigma utilizada para desenvolvimento de aplicações web, aplicações móveis e de desktop. A linguagem **Dart** permite que o código criado rode tanto no lado do cliente quanto no servidor.

DART

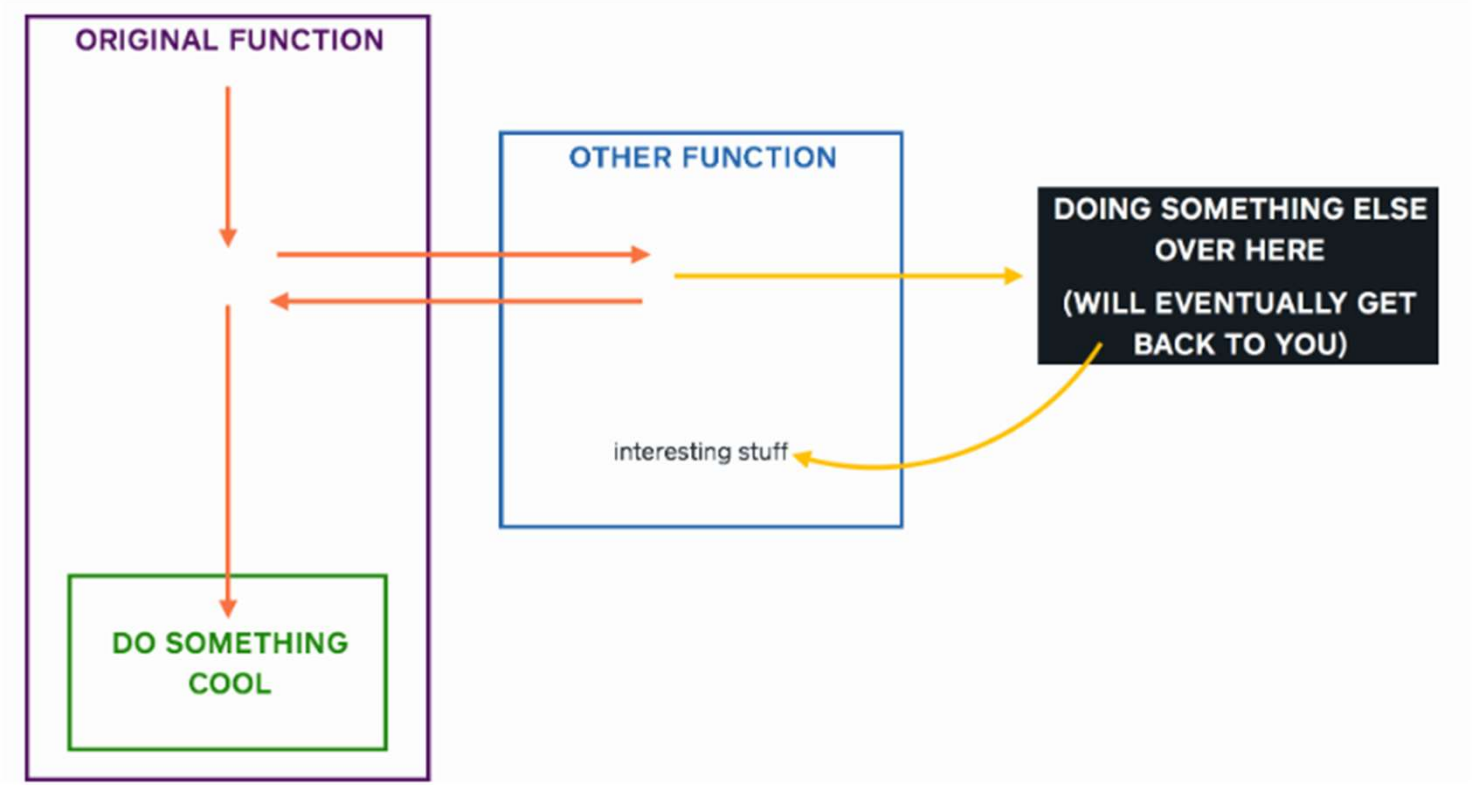
https://dartpad.dev/?



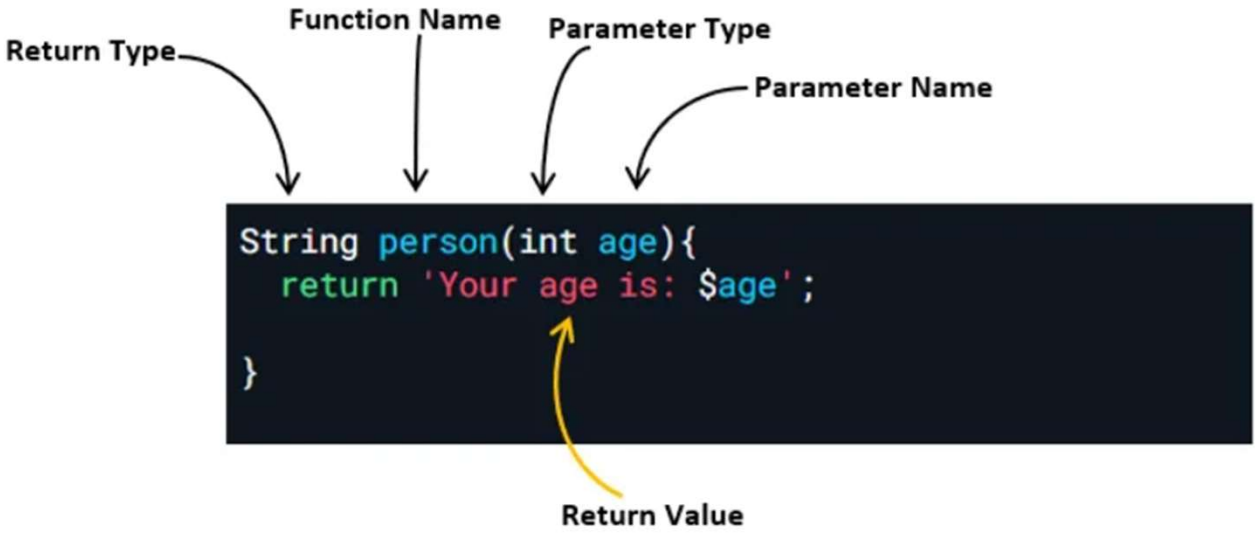
1 – Editor: Onde digitamos o código.
2 – Console: Onde vemos o resultado da execução do código.
3 – Documentação:

FUNÇÕES NO DART

Funções



Funções



Funções

Parâmetro *posicional* e obrigatório.	Parâmetro nomeado e opcional.	Parâmetro nomeado e obrigatório.	Parâmetro *posicional* e opcional.

void main(){
  teste('Julio');
}
 	
void teste(String a)
{
  print('$a');
}
	void main(){
	  teste(c: 'Cesar');
	}
	
	void teste({String? c})
	{
	  print('$c');
	}
		void main(){
		  teste(c: 'Cesar');
		}
		
		void teste({required String c})
		{
		  print('$c');
		}
			void main(){
			  teste('Julio');
			}
			
			void teste(String a, {String? c})
			{
			  print('$a $c');
			}

Funções

Parâmetro opcional com default

```
void main(){
 teste();
}

void teste({String c='Cesar'})
{
 print('$c');
}
```

Parâmetro function e obrigatório.

```
void main(){
        saudacoes('Zé mané', funcaox: digaAi);
}
void digaAi(){
  print('Ola...');
}
void saudacoes(String nome, {required Function funcaox})
{
        print('Saudações do $nome');
        funcaox();
}
```

EXPRESSÕES LAMBDA

DART

LAMBDA

Expressão Lambda	Com Lambda	Sem Lambda

void main() {
int sumOf(int a, int b) => a + b;
 	
print(sumOf(2,3));
 	
}
	// Arrow Syntax
	void sum(int x,int y) => print('sum is ${ x + y}’);
	
	void main(){
	sum(2,5);
	}
		void sum (int x, int y) {
		print('sum is ${x + y}');
		}
		
		void main (){
		sum(5,2);
		}

EXPRESSÃO TERNÁRIA

DART

EXPRESSÃO TERNÁRIA

```
// If and else
if (isClosed) {
   print('Store is closed');
}
else if (isOpen) {
   print('Store is open');
}
else if (isOutOfStock) {
   print('Item is out of stock');
}
else {
   print('Nothing matched');;
}
```

```
// Shorter way of if and else statement
isClosed ? askToOpen() : askToClose();
```

1 **2** **3**

COMPARISON		TRUE		FALSE
isClosed	?	askToOpen()	:	askToClose()

EXPRESSÃO REDUCE

DART: reduce ()

[1,4,5] -> 10

DART: reduce ()

```dart
void main() {
  var products = [123, 12, 35, 34, 23, 45, 132];
  final sum = products.reduce((i, j) => i+j);
  print(sum); // 404
}
```

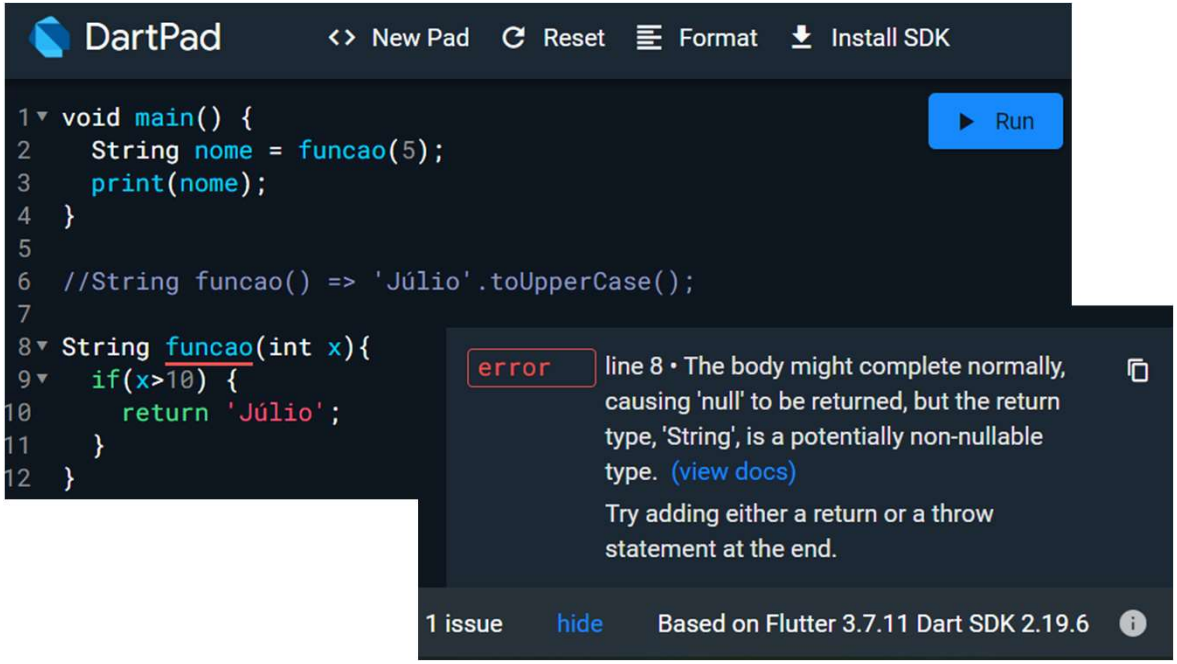
FUNÇÃO NULLABLE

DART

Retorno de funções com Nullables

Retorno nullables

Por que está dando erro na Linha 8?



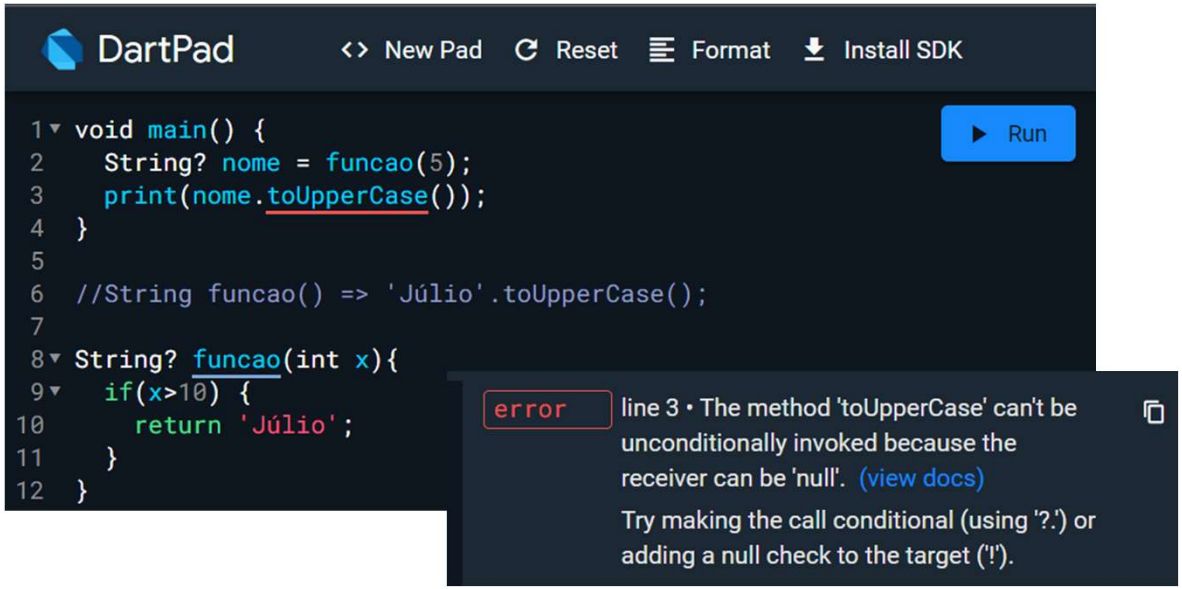
`Nullsafety00a.dart`

A função retorna **null**, quando não
É especificado um retorno.

Solução: Use o operador ?

Retorno nullables

Por que está dando erro na Linha 3?



`Nullsafety00b.dart`

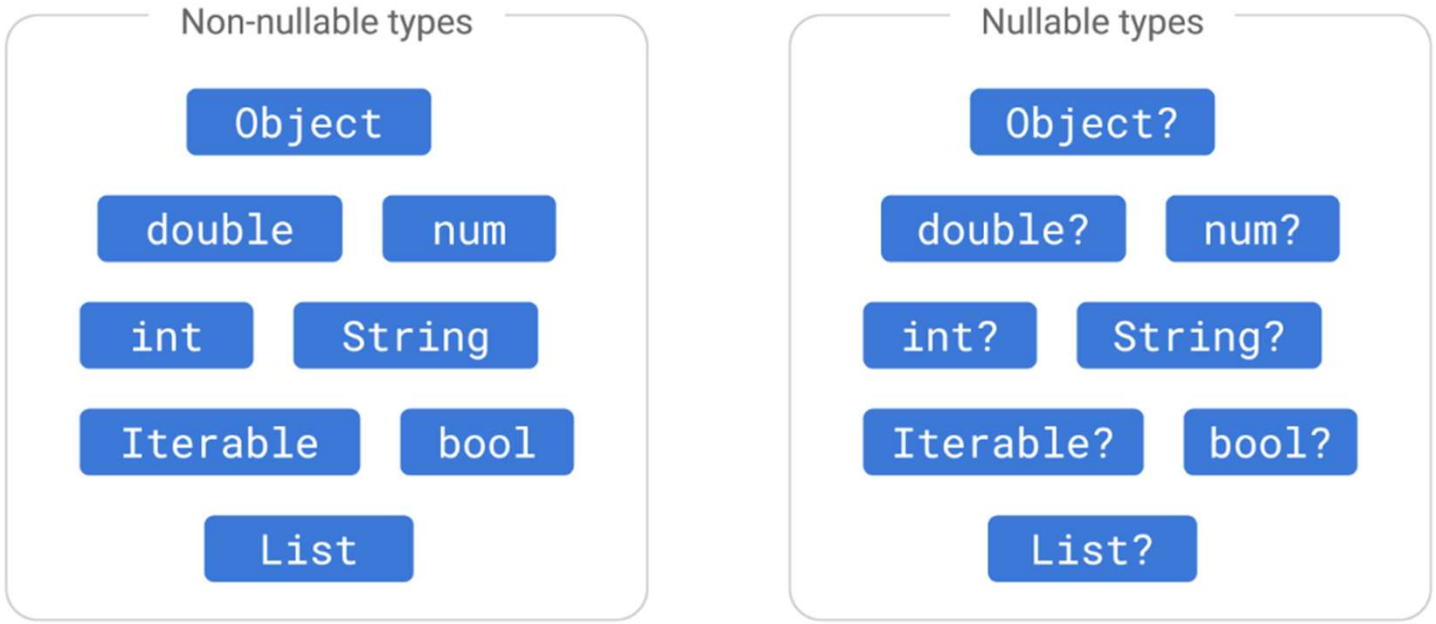
A função toUpperCase retorna letras, maiúsculas, Porém não é possível isso com null.

Solução: Use o operador ??

String **nome** = função(5) **??** 'Não informado';

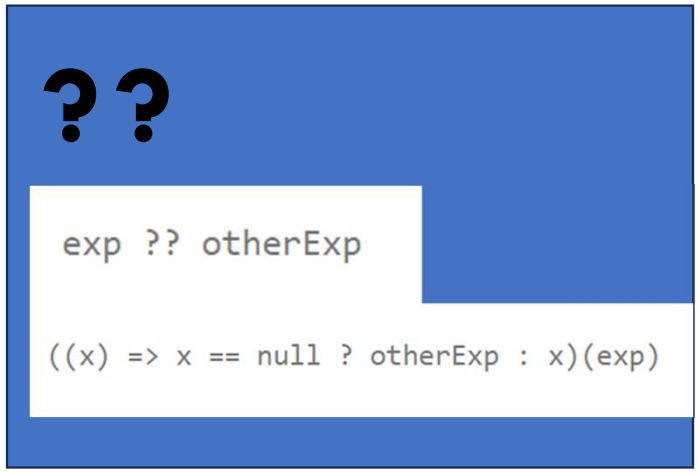
Caso o **nome** não exista, um valor default será assumido.

Resumo:

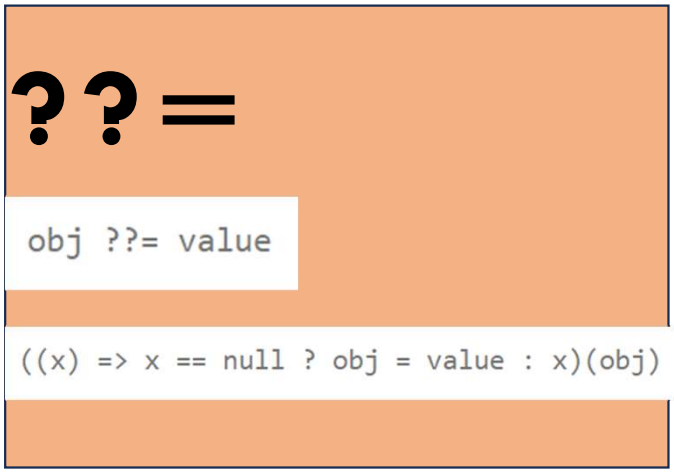


DART

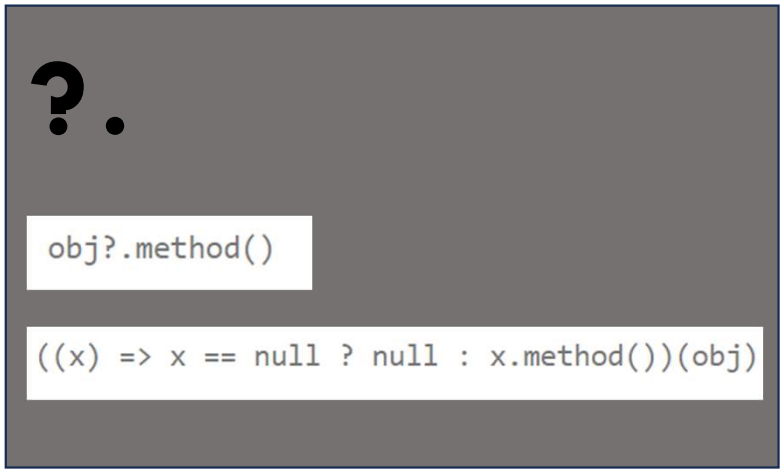
DART



Se for null recebe o valor exp
Senão recebe otherExp



Se for null recebe o valor



Chama o método se não for null

DART

```dart
// assign y to x, unless y is null,
//   otherwise assign z
x = y ?? z;

// assign y to x if x is null
x ??= y;

// call foo() only if x is not null
x?.foo();
```

FLUTTER

https://medium.com/@frojho/o-que-significa-e-no-dart-334599ed5f95

Copyright © 2023 Accenture. All rights reserved.

CONCORÊNCIA NO DART

Linguagem Assíncrona

Dart é uma linguagem de programação moderna e nova, com vários recursos legais que já são populares e estão validados em outras linguagens, como por exemplo a execução assíncrona.

FLUTTER

Execução Sincrona

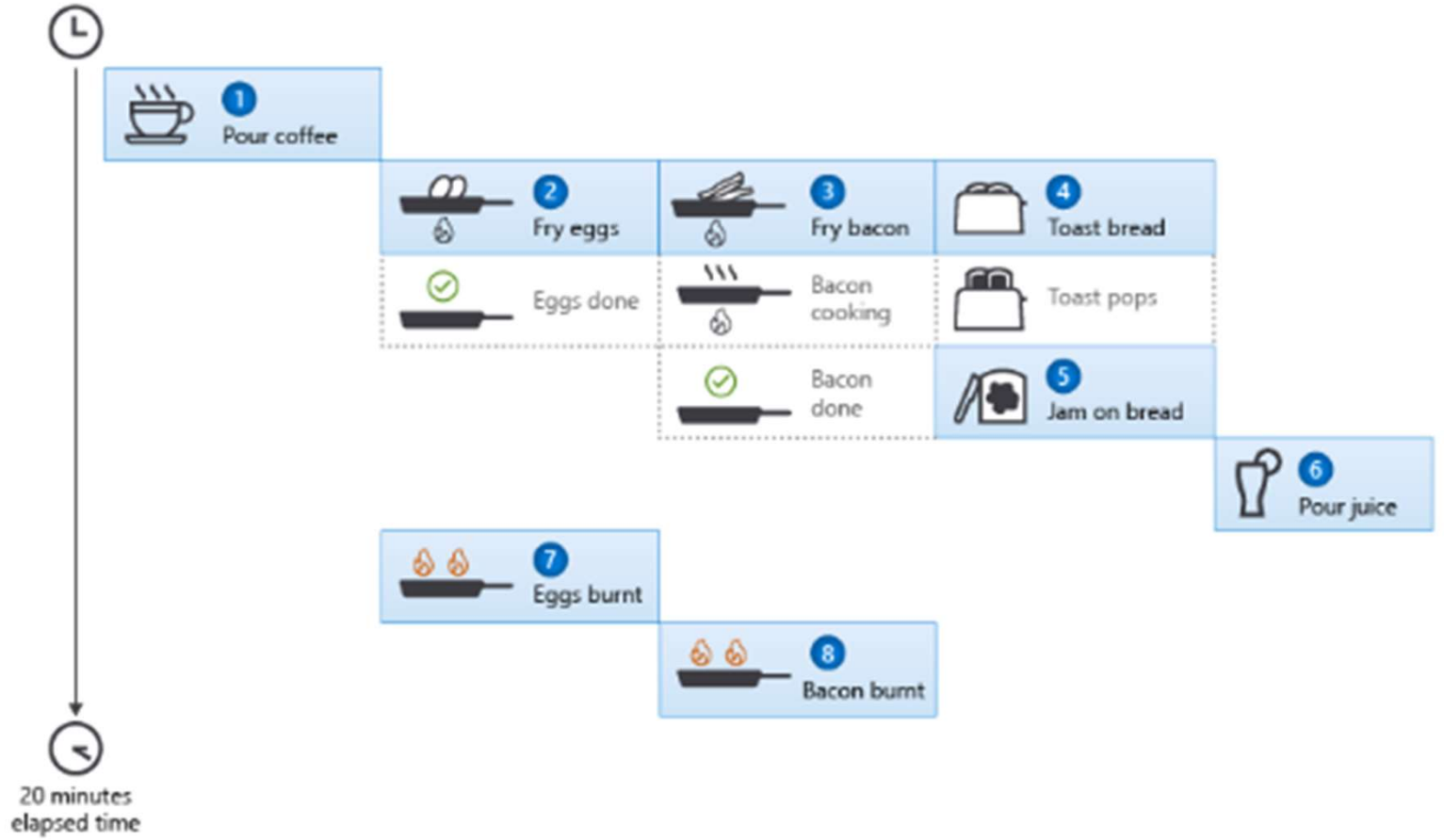
No **Dart** os commandos são executados de forma sequencial um após o outro.

- Sincrono

Copyright © 2023 Accenture. All rights reserved.

FLUTTER

- Execução Assincrona

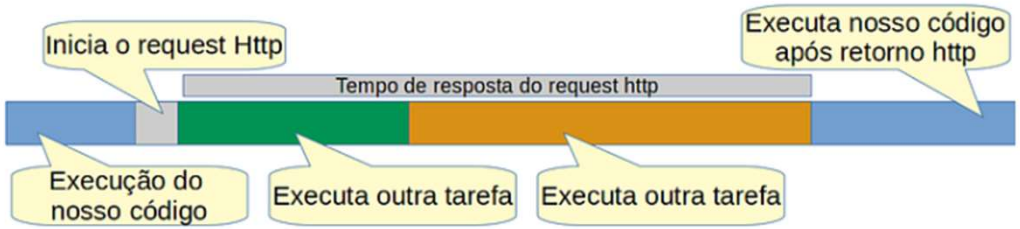


FLUTTER

- Execução Sincrona

Na programação síncrona, uma tarefa só é executada quando outra é finalizada, como podemos ver na imagem abaixo:

- Execução Assincrona



Copyright © 2023 Accenture. All rights reserved.

FLUTTER

- Execução Sincrona

Na programação síncrona, uma tarefa só é executada quando outra é finalizada, como podemos ver na imagem abaixo:

- Execução Assincrona

Já na programação assíncrona, diversas tarefas podem ser executadas em "paralelo", dividindo o tempo com a execução de diferentes tarefas:

FUNÇÃO FUTURE

DART

Await/Async/Future

1 – Editor: Onde digitamos o código.
2 – Console: Onde vemos o resultado da execução do código.
3 – Documentação:

Dart é uma linguagem ***single-thread****, ***non-bloking***.
Isso significa que diferentemente de linguagens como C e Java, onde há recursos para *programação paralela* e *multi-thread*, por padrão isso não ocorrem em **Dart** nem em JS.

Non-bloking significa que essa linguagem oferece recursos para ***execução assíncrona*** de tarefas.

O conceito de assíncronismo é muito importante no Flutter/Dart pois há casos em que um método levará um certo tempo para ser executado e o **app não pode** "**travar**" até que este retorno aconteça.

Await/Async/Future

- O ***async*** determina que um método será assíncrono, ou seja, não irá retornar algo imediatamente, então o aplicativo pode continuar a execução de outras tarefas enquanto o processamento não é finalizado.

- O ***await*** serve para determinar que o aplicativo deve esperar uma resposta de uma função antes de continuar a execução. Isso é muito importante pois há casos em que uma função depende do retorno de outra.

- Já o ***Future*** determina que uma função irá retornar algo no “futuro”, ou seja, é uma função que levará um tempo até ser finalizada.

Await/Async/Future

```
Future<List> getAllPets() async {

  final dataList = await DbUtil.getData('pets');
  .....
  return _petList;

}
```

No código, estamos determinando que o método ***getAllPets()*** irá retornar uma lista no "futuro", por isso o uso do **Future**. Além disso, essa função será assíncrona, sendo assim, precisamos indicar na assinatura do método a palavra **async**.

Para retornar a lista de pets, precisamos buscar estes dados de um banco de dados. Porém, esta consulta não é imediata, levando um certo tempo para ser concluída.

Para que o Flutter "espere" até que alguma informação seja retornada do banco de dados, utilizamos o **await**, indicando que o restante do método só poderá ser executado quando houver um retorno do método DbUtil.getData('pets')