



Introdução a Dart e Flutter

Conceitos da linguagem Dart e Desenvolvendo os meus primeiros projetos

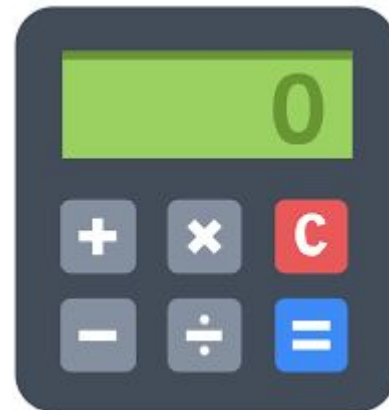
Fernando Duwe

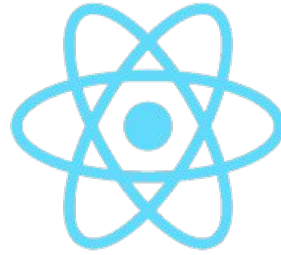


Precisamos desenvolver uma calculadora...

... qual tecnologia usaremos?...

...e em qual plataforma ela será executada?



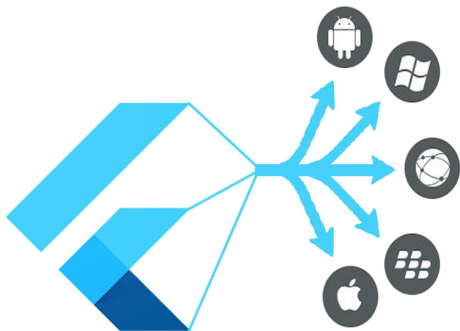


React



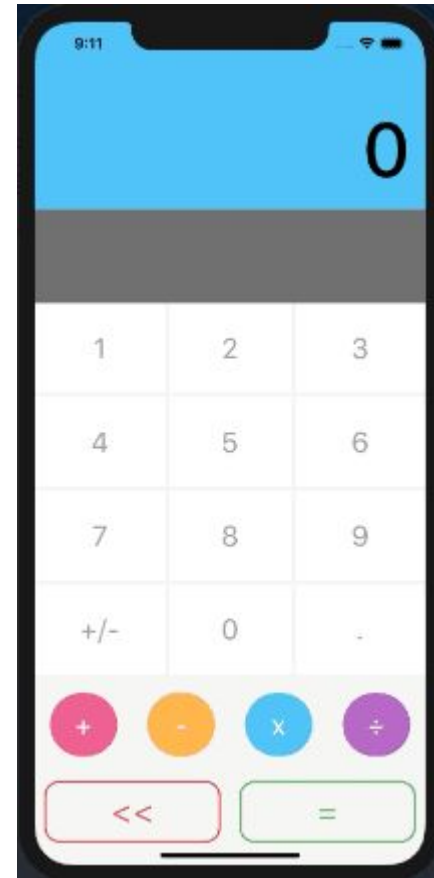


- Lançado em 2017
- Versão atual: 3.3



- Botões
- Textos
- Páginas
- Aplicação

Tudo é um Widget!



Dart

```
class _MyHomePageState extends State<MyHomePage> {  
  int _counter = 0;  
  
  void _incrementCounter() {  
    setState(() {  
      _counter++;  
    });  
  }  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text(widget.title),  
      ),  
      body: Center(  
        child: Column(  
          mainAxisAlignment: MainAxisAlignment.center,  
          children: [  
            const Text(  
              'You have pushed the button this many times:',  
            ),  
            Text(  
              '$_counter',  
              style: Theme.of(context).textTheme.headlineMedium,  
            ),  
          ],  
        ),  
      ),  
      floatingActionButton: FloatingActionButton(  
        onPressed: _incrementCounter,  
        tooltip: 'Increment',  
        child: const Icon(Icons.add),  
      ),  
    );  
  }  
}
```

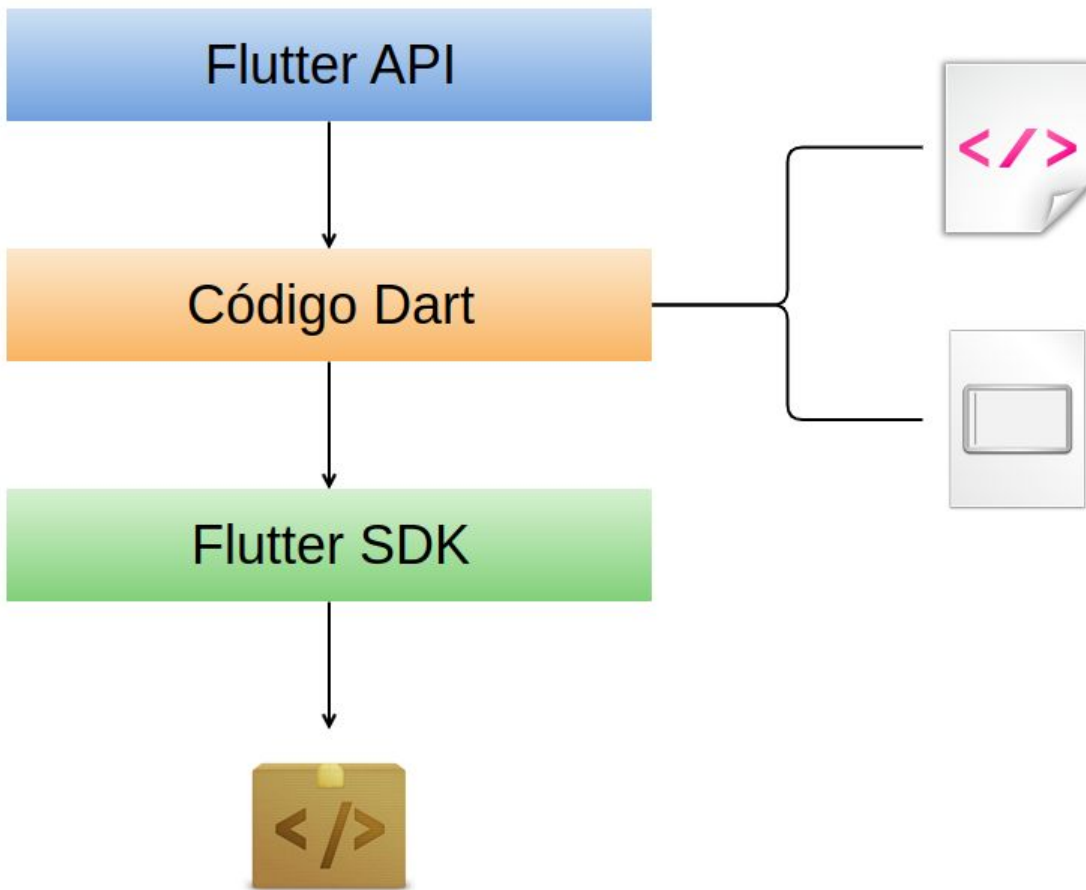
Flutter Demo Home Page

You have pushed the button this many times:

3



Como o Flutter funciona?





- O que é *null*?
- O que é uma linguagem *null safety*?

```
String nome = "Pedro";

String nomeCompleto(String sobrenome) {
    return nome + " " + sobrenome;
}

nomeCompleto(null);
```




- Compilada
- Fortemente tipada
- Lançada em 2011
- Semelhante a C
- *Case sensitive*

```
void main() {  
    // Seu código aqui  
}  
  
void main(List<String> arguments) {  
    // Seu código aqui  
}
```



Object

- Numbers - int & double
- String
- Bool
- List
- Set
- Maps

null

```
// String para inteiro  
int numero = int.parse('1')
```

```
// Inteiro para string  
String numeroStr = 1.toString()
```



- Tipo mais simples array
- Index começa por 0

```
var listNotas = [9, 7.5, 8];  
  
listNotas[1] = 9;  
  
for(var i = 0; i < listNotas.length; i++) {  
    somaMedia += listNotas[i];  
}
```



- Coleção desordenada de itens únicos

```
var tecnologiasMobile = {'Flutter', 'React Native', 'Kotlin', 'Swift'};

tecnologiasMobile.add("Xamarin");

for (String tecnologia in tecnologiasMobile)
    print(tecnologia);
```



- Coleção de itens com chave e valor
- Chave e valor podem ser de qualquer tipo de objeto

```
var mapaNotas = Map<String, double>();  
  
mapaNotas["Prova 1"] = 9;  
mapaNotas["Prova 2"] = 7.5;  
mapaNotas["Prova 3"] = 8.5;  
  
mapaNotas.forEach((chave, valor) => print("$chave - $valor"));
```



- Nome deve começar com letra ou _
- Fortemente tipada
- Tipo nomeVariavel

```
int idade = 21;

var nome = "Paulo";

String sobrenome = 'Silva';

var nomeCompleto = nome + " " + sobrenome;

print("O nome completo do usuário é
$nomeCompleto");
```



- ? - Variável pode ser nula
- ! - Variável não pode ser nula

```
int? numeroLinha
```

```
int idade = 25;
```



- late: Receberá o valor mais tarde
- final: Valor setado uma única vez
- const: Valor setado em compilação


```
const double pi = 3.14;  
  
final usuarioId = "42141241221512";  
  
late nomeUsuario;
```




- Matemáticos: `+` `-` `*` `/`
- Lógicos: `&&` `||` `!`
- Igualdade: `==` `!=`
- Cascata: `..` `?..`
- Relacional e teste de tipo: `>=` `>` `<=` `<` `as` `is` `is!`
- Operadores condicionais: `condição ? resultado1 : resultado2` `expressao1 ?? expressao2;`



- Comentário de somente uma linha: `//`
- Comentário de várias linhas: `/**/`
- Documentação: `///` `/**`
- Link para documentação: `[]`



```
void teste(int prParam) {  
    return 4;  
}
```

```
bool teste(int prParam) => prParam == 4;
```

```
list.map((item) => item.toUpperCase());
```



if...else

```
if (diaAula()) {  
    voce.arrumarMaterial();  
} else if (diaTrabalho()) {  
    voce.irTrabalhar();  
} else {  
    voce.abrirGelada();  
}
```



for

```
// for
for (var i = 0; i < 5; i++) {
    print("Partiu Flutter x$i");
}

// foreach
for (final candidato in listaCandidatos) {
    candidato.entrevistar();
}
```



while...do

```
while (!estaGanhando()) {  
    meninoNey.fazerGol();  
}
```

do...while

```
do {  
    escreverLinha();  
} while (!terminouDocumento())
```



- break
- continue

switch

```
string comando = "run";

switch(comando) {
    case "open": executarOpen();
                  break;

    case "run":  executarRun();
                  break;

    default: enviarMensagem();
}
```



Vamos criar um sistema de cálculo de notas? A aplicação deve armazenar uma quantidade N de notas e calcular a média a qual o aluno chegou. A nota para passar de ano é 7.




Desenvolva uma função calculadora, que receberá três parâmetros:

`calculadora(double valor1, double valor2, String operacao)`

`operacao` pode ser igual a: “SOMA”, “SUBTRACAO”, “DIVISAO” e “MULTIPLICACAO”


Obs: Os valores `valor1` e `valor2` podem ser nulos. Nesse caso, considere seu valor igual a zero (0).

A função deve retornar o valor calculado.



```
// Lançando uma exceção  
throw FormatedException("Teste")
```


```
// Lançando uma exceção  
throw "Teste só de string";
```



```
try {
    rodarProcesso();
} catch (e) {
    print("Erro ao executar o processo. Erro: $e");
}
```

```
try {
    rodarProcesso();
} catch (e) {
    gravarLog(e);

    rethrow;
}
```




```
try {  
    rodarProcesso();  
} finally {  
    // mesmo que tenha ocorrido um erro, vai vir para cá  
    fecharPortas();  
}
```




Desenvolva uma função que retorne o fatorial de um número. Caso o número passado por parâmetro for maior que 99, você deverá gerar uma exceção, avisando que o número passado por parâmetro não é aceito, por ser grande demais.

Ao chamar esse método, você deve tratar qualquer exceção gerada, exibindo a seguinte mensagem no console: “Ocorreu um erro ao executar o fatorial: <mensagem de erro>” e exibi-la no console.



```
class Teste {  
    String atributo1 = "";  
    String _atributoPrivado = "";  
  
    // Método construtor  
    Teste(String atributo1) {  
        this.atributo1 = atributo1;  
    }  
}
```



```
class Teste {
    String atributo1 = "";
    String _atributoPrivado = "";

    // getter
    String get atributoPrivado {
        return this._atributoPrivado;
    }

    // setter
    void set atributoPrivado(String novoAtributo) {
        this._atributoPrivado = novoAtributo;
    }
}
```



Crie uma classe chamada “Pessoa”. Esta classe deve possuir os seguintes atributos:

- ID: só pode ser preenchido uma vez, no construtor;
- Nome: String, Atributo público;
- Apelido: Atributo privado, podendo ser nulo;

Você deverá criar também um método público, que retorna o nome completo, junto com o seu apelido.


```
class SmarTV extends TV {  
  
    @Override  
    void trocarCanal() {  
        // ...  
    }  
  
    @Override  
    void ligar() {  
        super.ligar();  
  
        this._conectarInternet();  
    }  
  
    SmarTV(String modelo) : super(modelo) {  
        this.ligar();  
    }  
}
```



Crie uma classe chamada “Aluno”, herdando a classe Pessoa, criada no exercício anterior. Esta classe deve possuir os seguintes atributos:

- Sala: String, privado. Deve ser criado um getter para ler seu conteúdo;
- Curso: String, privado. Deve ser criado um getter para ler seu conteúdo;

Ambos devem ser preenchidos no método construtor, enviando também o ID da pessoa;