

2 A LINGUAGEM DART

Em 2011, o Google lançou o Dart como uma linguagem de scripts com o objetivo de se tornar a principal linguagem para desenvolvimento de scripts em navegadores, substituindo o JavaScript. Dart foi introduzido com o intuito de oferecer uma alternativa poderosa e flexível ao JavaScript, atendendo às demandas crescentes de desenvolvimento web.

Dart recebeu reconhecimento significativo, especialmente devido ao *framework* Flutter, que possibilita a programação de aplicativos nativos para Android e iOS. Além disso, a linguagem também oferece suporte para a criação de aplicações web e desktop.

Assim como C, C++, Java, C# e seguindo o paradigma de orientação a objetos, Dart é uma linguagem fortemente tipada, porém, não é necessário especificar os tipos, uma vez que o Dart consegue inferi-los automaticamente. Além disso, a utilização do caractere underline () antes de um atributo ou método o torna privado.

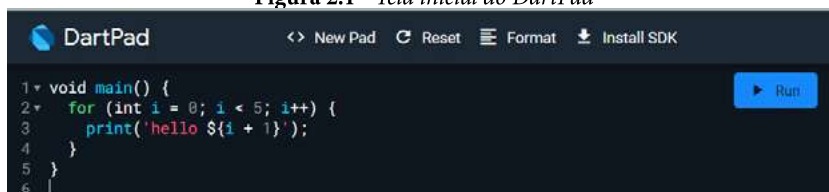
Códigos feitos em Dart podem ser compilados de duas formas: *ahead-of-time* (AOT) e *just-in-time* (JIT). No primeiro caso, o código é compilado para uma linguagem nativa (ARM nativo, por exemplo), proporcionando um desempenho semelhante ao de aplicativos nativos. No segundo caso, a compilação ocorre diretamente no dispositivo em que a aplicação está sendo executada, como em um smartphone, permitindo o recurso de *hot reload*.

2.1 DartPad

Para programar em Dart de forma mais fácil e sem a necessidade de um ambiente Flutter configurado é possível usar o site DartPad⁸. O DartPad possui código aberto e pode ser executado em qualquer navegador atual (GOOGLE, 2023c).

A Figura 2.1 apresenta a tela inicial do DartPad. É nesse ambiente que todos os exemplos deste capítulo serão executados.

Figura 2.1 - Tela inicial do DartPad



Fonte: Elaborado pelos autores.

2.2 Conceitos iniciais

Em Dart, a programação é estaticamente tipada, o que significa que, em geral, uma vez que atribuímos um valor a uma variável com um tipo específico, não é possível armazenar valores de outros tipos nessa mesma variável (exceto para o tipo *dynamic*). Por exemplo, se uma variável é declarada como uma String, ela não será automaticamente convertida em um número para fins de soma.

Neste tópico serão discutidas classes que trabalham com números e que em outras linguagens comumente são tratadas como tipos primitivos. Na sequência será tratada a classe String e a classe bool, também será tratada a classe dynamic e as classes Function, List e Map. Por fim serão tratados os argumentos de uma Function e possibilidades relacionadas a esse conceito.

2.2.1 Classes que trabalham com números

8 <https://dartpad.dev/>

Tipos primitivos, como existem em linguagens como C ou Java não existem em Dart. Em Dart “Tudo é objeto”. A Figura 2.2 ilustra uma variável do tipo **int** chamando o método **abs()**.

Figura 2.2 - Tipos de dados



Fonte: Elaborado pelos autores.

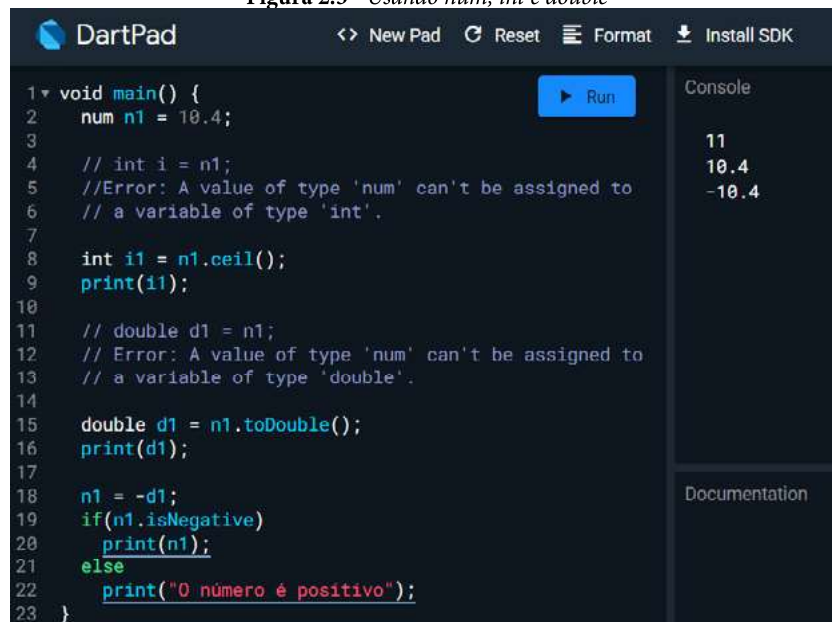
Para trabalhar com números a linguagem Dart disponibiliza o tipo **num** (Number) tendo como subtipos **int** e **double**. O tipo **num** trabalha números de forma mais genérica já os tipos **int** e **double** são os subtipos mais específicos. É importante notar que **num**, **int** e **double** são classes e portanto possuem seus métodos.

A Figura 2.3 apresenta o uso dos tipos **num**, **int** e **double**. Como **int** e **double** herdam de **num** não é possível fazer variáveis do tipo **int** ou **double** receberem um **num**, mas o contrário pode ser feito, ou seja, variáveis do tipo **num** podem receber um **int** ou **double** normalmente.

Para maiores informações sobre classes que trabalham com números recomenda-se a página de referência da Google sobre essa temática⁹.

9 <https://dart.dev/guides/language/numbers>

Figura 2.3 - Usando num, int e double



The screenshot shows the DartPad interface with a code editor on the left and a console on the right. The code defines a `main` function that initializes a `num` variable `n1` with the value `10.4`. It then attempts to assign `n1` to an `int` variable `i`, which results in a runtime error: "Error: A value of type 'num' can't be assigned to a variable of type 'int'". The code then uses `n1.ceil()` to get the integer part `11` and prints it. Next, it attempts to assign `n1` to a `double` variable `d1`, which also results in a runtime error: "Error: A value of type 'num' can't be assigned to a variable of type 'double'". The code then uses `n1.toDouble()` to get the double value `10.4` and prints it. Finally, it checks if `n1` is negative (it is not) and prints a message: "0 número é positivo".

```

1 void main() {
2   num n1 = 10.4;
3
4   // int i = n1;
5   //Error: A value of type 'num' can't be assigned to
6   // a variable of type 'int'.
7
8   int i1 = n1.ceil();
9   print(i1);
10
11  // double d1 = n1;
12  // Error: A value of type 'num' can't be assigned to
13  // a variable of type 'double'.
14
15  double d1 = n1.toDouble();
16  print(d1);
17
18  n1 = -d1;
19  if(n1.isNegative)
20    print(n1);
21  else
22    print("0 número é positivo");
23 }

```

The console output shows the results of the execution: `11`, `10.4`, and `-10.4`.

Fonte: Elaborado pelos autores.

2.2.2 Classe String

Em Dart, Strings podem ser representadas por aspas simples ou aspas duplas. Isso facilita a colocação de caracteres aspas simples e aspas duplas no meio de String, ou seja, passa a não ser necessário o uso de um caractere de escape.

Na Figura 2.4 é possível visualizarmos alguns dos principais métodos da classe String:

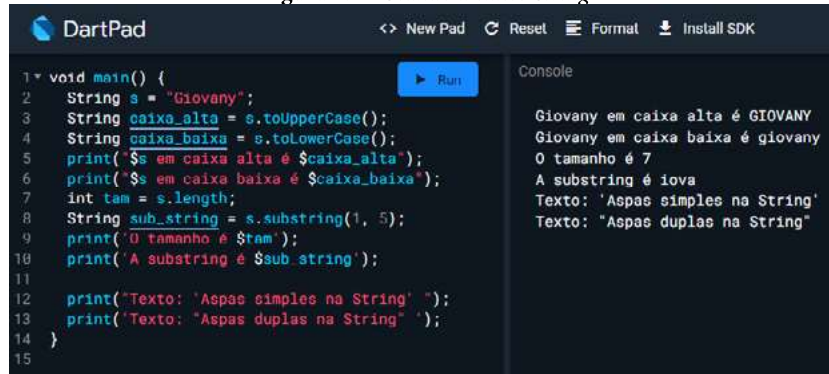
toUpperCase: retorna a String em “caixa alta”, ou seja, toda em maiúscula;

toLowerCase: retorna a String em “caixa baixa”, ou seja, toda em minúscula;

length: possui o tamanho (quantidade de caracteres) da String;

substring: retorna a substring que começa no caractere com a posição informada no primeiro parâmetro e termina na posição imediatamente anterior a informada no segundo parâmetro.

Figura 2.4 - Usando a classe String



```

1 void main() {
2   String s = "Giovany";
3   String caixa_alta = s.toUpperCase();
4   String caixa_baixa = s.toLowerCase();
5   print("Ss em caixa alta é $caixa_alta");
6   print("Ss em caixa baixa é $caixa_baixa");
7   int tam = s.length;
8   String sub_string = s.substring(1, 5);
9   print("O tamanho é $tam");
10  print("A substring é $sub_string");
11
12  print("Texto: 'Aspas simples na String' ");
13  print("Texto: \"Aspas duplas na String\" ");
14 }
15

```

Console

```

Giovany em caixa alta é GIOVANY
Giovany em caixa baixa é giovany
O tamanho é 7
A substring é iova
Texto: 'Aspas simples na String'
Texto: "Aspas duplas na String"

```

Fonte: Elaborado pelos autores.

A classe String da linguagem Dart possui vários outros métodos como os encontrados em linguagens como Java (FURGERI, 2018a). Para maiores informações sobre métodos e funcionamento da classe String recomenda-se a página de referência da Google sobre essa temática¹⁰.

2.2.3 Classe bool

Essa classe permite armazenar apenas dois valores: **true** (verdadeiro) e **false** (falso). A Figura 2.5 ilustra o uso da classe bool.

Figura 2.5 - Usando a classe bool



```

1 void main() {
2   bool eh_legal = false;
3   if(!eh_legal)
4     eh_legal = true;
5   print("Agora ficou legal. Certo? $eh_legal");
6 }
7

```

Console

```

Agora ficou legal. Certo? true

```

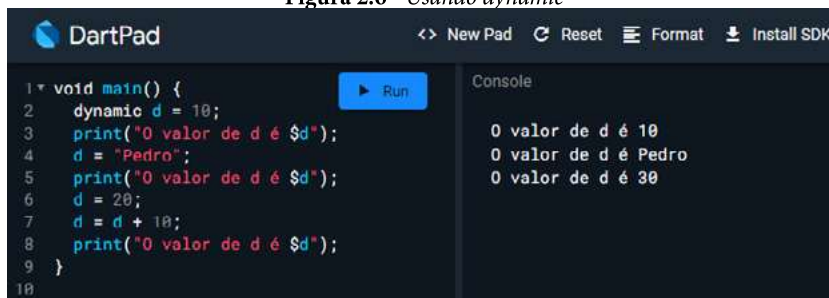
Fonte: Elaborado pelos autores.

¹⁰ <https://api.dart.dev/stable/3.0.4/dart-core/String-class.html>

2.2.4 Classe *dynamic*

Variáveis do tipo *dynamic* podem receber valores de todos os outros tipos e é possível mudar esses valores em tempo de execução. A Figura 2.6 mostra a variável ‘d’ recebendo um valor do tipo **int** (10), depois recebendo uma **String** (“Pedro”) e novamente recebendo um valor do tipo **int** (20):

Figura 2.6 - Usando *dynamic*



```
1 void main() {  
2   dynamic d = 10;  
3   print("O valor de d é $d");  
4   d = "Pedro";  
5   print("O valor de d é $d");  
6   d = 20;  
7   d = d + 10;  
8   print("O valor de d é $d");  
9 }  
10
```

Console

```
O valor de d é 10  
O valor de d é Pedro  
O valor de d é 30
```

Fonte: Elaborado pelos autores.

2.2.5 Classe Function

Em Dart é possível criar variáveis e parâmetros do tipo Function (função). Isso significa que comportamentos podem ser parametrizados e/ou armazenados em variáveis, na prática isso possibilita a implementação de comportamentos mais dinâmicos o que facilita e potencializa o processo de programação. Na Figura 2.7 tem-se o uso de classes do tipo Function como variáveis e parâmetros de funções.