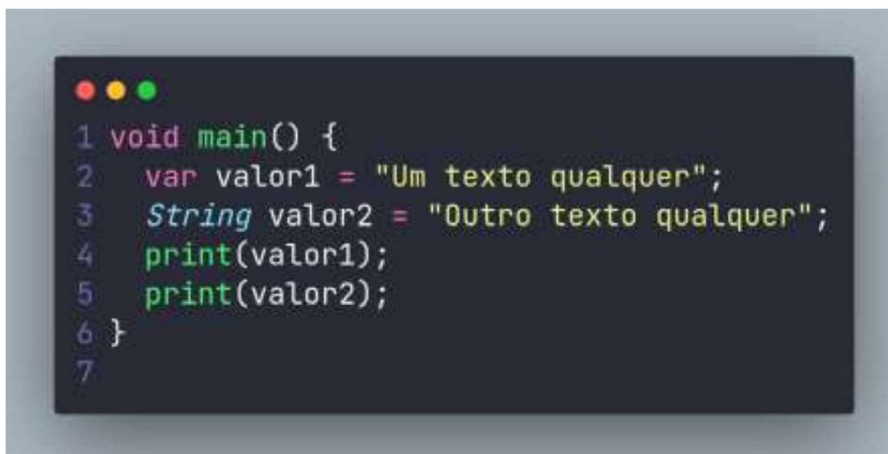


Declaração de variável

A declaração de uma variável, talvez seja o segundo passo em qualquer linguagem, logo atrás do nosso comando de impressão. E com o Dart não seria diferente. Ao declararmos uma variável no Dart, podemos utilizar duas estratégias:

- A primeira é declarar a variável utilizando a palavra reservada **“var”**. Dessa forma, a variável receberá o tipo de dado do mesmo tipo do valor informado. Ou seja, se eu criar uma chamada **valor1** e utilizar a palavra reservada **“var”**, poderei **colocar qualquer tipo de valor nela**, seja um número, um texto ou mesmo um valor booleano.
- A segunda estratégia seria inferir o tipo de dado da variável no momento da criação da mesma.



```
1 void main() {  
2   var valor1 = "Um texto qualquer";  
3   String valor2 = "Outro texto qualquer";  
4   print(valor1);  
5   print(valor2);  
6 }  
7
```

Declaração de constantes

A declaração de constante em Dart é tão simples quanto a criação de variáveis. Para declararmos uma constante em nosso código basta adicionarmos a palavra reservada **const** antes da declaração da tipagem da variável. Isso dirá ao Dart que aquela será uma variável imutável, ou seja, uma constante.

Ao declararmos uma constante no Dart, podemos utilizar duas estratégias:

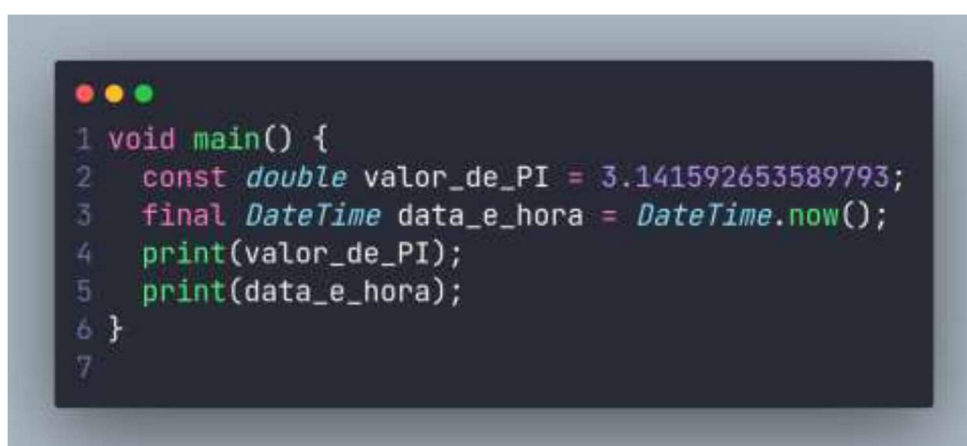
- A primeira é declará-la utilizando a palavra reservada **“const”**.
- A segunda estratégia seria utilizar a palavra reservada **“final”**.

Essas implementações têm uma finalidade em comum, a declaração de uma

constante. Porém existe uma diferença fundamental entre elas:

Uma declaração *const* é uma declaração em tempo de compilação, e a declaração *final* é uma declaração em tempo de execução. Isso quer dizer que uma declaração em tempo de compilação exige que o valor da variável esteja disponível no momento em que a aplicação é compilada, não podendo depender da execução de algum método.

Já a implementação em tempo de execução permite que o valor da constante seja inferido após a execução de um determinado método, ou seja, após a compilação do nosso código.

A screenshot of a code editor with a dark background and light-colored text. The code is written in Dart and demonstrates the use of 'const' and 'final' keywords. It includes a 'main' function that declares a constant double value for PI and a final DateTime value for the current date and time, both followed by print statements.

```
1 void main() {  
2   const double valor_de_PI = 3.141592653589793;  
3   final DateTime data_e_hora = DateTime.now();  
4   print(valor_de_PI);  
5   print(data_e_hora);  
6 }  
7
```

Tipos de dados

Um aspecto muito importante sobre qualquer linguagem são os tipos de dados suportados. E no Dart podemos dizer que estamos vivenciando um sonho, pois o Dart possui uma variedade de tipos de dados e como já foi comentado, todos eles são objetos, já que o Dart não possui tipos primitivos, como acontece com linguagens como o Java e o PHP.

Dentre os principais tipo de dados que o Dart possui, podemos ressaltar os seguintes:

- ***String***: Representa uma sequência de caracteres alfanuméricos, podendo conter letras, números e caracteres especiais (!,@,#,\$,%,&, e etc.);
- ***double***: Representa um valor com fração de até 64 bits;
- ***bool***: Representa um valor booleano true ou false;
- ***int***: Representa um valor inteiro de até 64 bits;

- **List:** Representa uma coleção de objetos indexados por um valor inteiro;
- **Map:** Representa uma coleção de objetos indexados por um conjunto de chave valor/ valor;
- **num:** Representa um número inteiro ou com precisão. Esse objeto é implementado pelas classes Int e Double;
- **Object:** É a classe base para todos os objetos Dart. Como Object é a raiz da hierarquia de classes Dart, todas as outras classes Dart são subclasses de Object;
- **Future:** Um Future é usado para representar um valor potencial, ou erro, que estará disponível em algum momento no futuro. Os receptores de um futuro podem registrar retornos de chamada que tratam do valor ou erro, uma vez que ele está disponível.
- **dynamic:** Como o próprio nome já sugere, representa um valor dinâmico. Ou seja, um tipo de dado que pode ser de qualquer tipo e pode receber objetos de qualquer tipo. Enfim, dinâmico.
- **var:** Utilizamos a palavra reservada var para criar variáveis sem ter tipo definido. Temos que ter atenção ao utilizar var, pois após a inferência de tipo de dado, o mesmo não pode ser modificado, ou seja, se criarmos uma variável e ela receber uma String, não poderemos modificar para double posteriormente.
- **var x dynamic:** A diferença entre var e dynamic é que var após receber a primeira inferência de tipo de dados não poderá ser alterada, no entanto dynamic pode receber diversos tipos de dados no decorrer da execução do código.

```

1 void main() {
2   var nome = "Thiago";
3   print(nome);
4
5   nome = "José"; //Funciona. podemos substituir uma String por outra.
6   print(nome);
7
8   //nome = 10; //Erro. o tipo de dados foi inferido como String.
9   //print(nome);
10
11  dynamic variavelCoringa = "Thiago";
12  print(variavelCoringa);
13  //Funciona. com dynamic podemos inferir
14  //novos tipos de dados múltiplas vezes.
15  variavelCoringa = 10;
16  print(variavelCoringa);
17}
18

```

Amarrações - Palavras reservadas

Algumas palavras são reservadas para a tecnologia, impedindo o uso para finalidade diferente da determinada na linguagem.

abstract	continue	external	implements	operator	this
as	covariant	factory	import	part	throw
assert	default	false	in	rethrow	true
async	deferred	final	interface	return	try
await	do	finally	is	set	typedef
break	dynamic	for	library	show	var
case	else	Function	mixin	static	void
catch	enum	get	new	super	while
class	export	hide	null	switch	with
const	extends	if	on	sync	yield