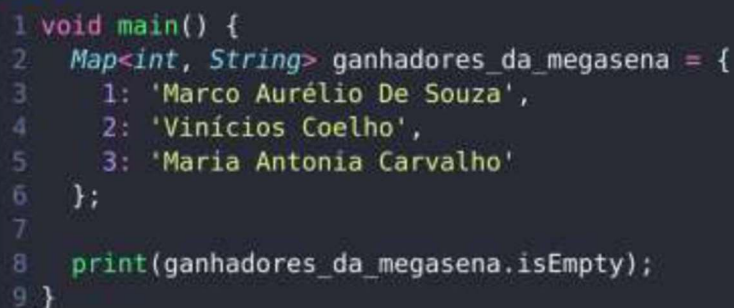


Propriedades

Como bem sabemos, todos os tipos de dados do Dart são objetos. Dessa forma possuem propriedades, e com o mapa não seria diferente. Neste tópico falaremos das principais propriedades do objeto *Map*.

Propriedade *isEmpty*

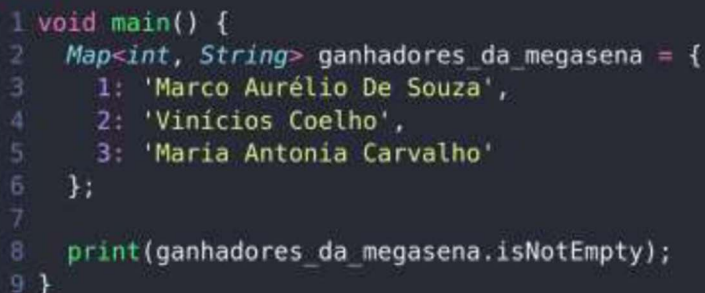
A propriedade *isEmpty*, como o próprio nome já sugere, verifica se o mapa está vazio, ou seja, se ele não possui nenhum elemento. Caso o mapa esteja vazio, retornará *true*, caso contrário retornará *false*.

A screenshot of a code editor with a dark background and light-colored text. The code is in Dart and defines a main function. Inside the function, a Map<int, String> named 'ganhadores_da_megasena' is initialized with three entries: 1: 'Marco Aurélio De Souza', 2: 'Vinícios Coelho', and 3: 'Maria Antonia Carvalho'. The code then prints the result of 'ganhadores_da_megasena.isEmpty'.

```
1 void main() {  
2   Map<int, String> ganhadores_da_megasena = {  
3     1: 'Marco Aurélio De Souza',  
4     2: 'Vinícios Coelho',  
5     3: 'Maria Antonia Carvalho'  
6   };  
7  
8   print(ganhadores_da_megasena.isEmpty);  
9 }
```

Propriedade *isNotEmpty*

A propriedade *isNotEmpty*, como o próprio nome já sugere, verifica se o mapa **não está vazio**, ou seja, se ele possui nenhum elemento. Caso o mapa não esteja vazio, retornará *true*, caso contrário retornará *false*.

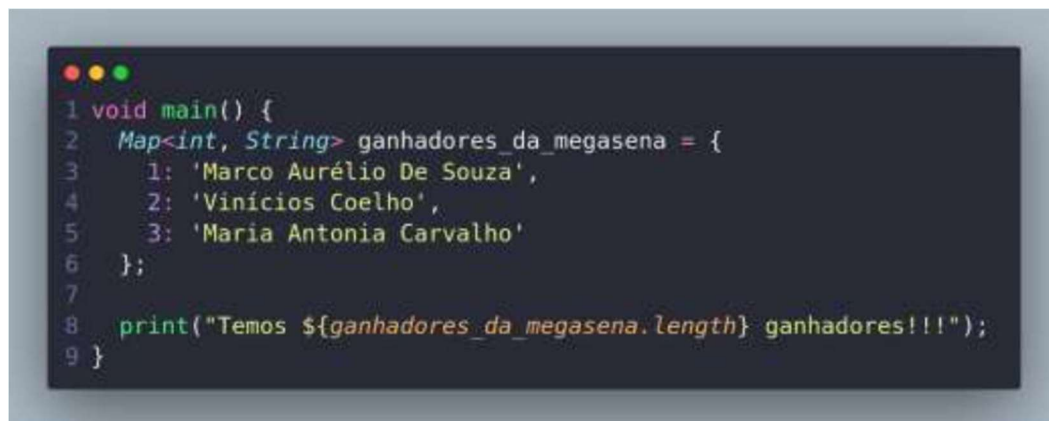
A screenshot of a code editor with a dark background and light-colored text. The code is in Dart and defines a main function. Inside the function, a Map<int, String> named 'ganhadores_da_megasena' is initialized with three entries: 1: 'Marco Aurélio De Souza', 2: 'Vinícios Coelho', and 3: 'Maria Antonia Carvalho'. The code then prints the result of 'ganhadores_da_megasena.isNotEmpty'.

```
1 void main() {  
2   Map<int, String> ganhadores_da_megasena = {  
3     1: 'Marco Aurélio De Souza',  
4     2: 'Vinícios Coelho',  
5     3: 'Maria Antonia Carvalho'  
6   };  
7  
8   print(ganhadores_da_megasena.isNotEmpty);  
9 }
```

Propriedade Length

A propriedade *length*, como o próprio nome já sugere, retorna o número de elementos armazenados no mapa.

Esse valor é muito utilizado para verificações relacionadas a impressão de elementos na tela, sendo utilizada em conjuntos com estruturas de repetição, como o *while* e o *for*.



```
1 void main() {
2     Map<int, String> ganhadores_da_megasena = {
3         1: 'Marco Aurélio De Souza',
4         2: 'Vinícios Coelho',
5         3: 'Maria Antonia Carvalho'
6     };
7
8     print("Temos ${ganhadores_da_megasena.length} ganhadores!!!");
9 }
```

Métodos

A classe *Map* possui alguns métodos, os quais podemos utilizar para manipular o valor do objeto. Vejamos quais são eles e como podemos utilizá-los.

Método remove

O método *remove*, como o nome sugere, remove um elemento do mapa. Caso seja possível remover, um valor *true* será retornado, caso contrário será retornado um valor *false*.

Parâmetro	Descrição
elemento	Elemento a ser removido do mapa

```

1 void main() {
2     Map<int, String> ganhadores_da_megasena = {
3         1: 'Marco Aurélio De Souza',
4         2: 'Vinícios Coelho',
5         3: 'Maria Antonia Carvalho'
6     };
7     // Imprimimos o mapa
8     print(ganhadores_da_megasena);
9
10    // Removendo um elemento do mapa
11    ganhadores_da_megasena.remove(2); // Chave do elemento
12
13    // Imprimimos o mapa
14    print(ganhadores_da_megasena);
15 }
16

```

Método clear

O método *clear*, como o nome sugere, limpa o mapa. Ou seja, remove todos os elementos presentes no mesmo.

```

1 void main() {
2     Map<int, String> ganhadores_da_megasena = {
3         1: 'Marco Aurélio De Souza',
4         2: 'Vinícios Coelho',
5         3: 'Maria Antonia Carvalho'
6     };
7     // Imprimimos o mapa
8     print(ganhadores_da_megasena);
9
10    // Limpando o mapa
11    ganhadores_da_megasena.clear();
12
13    // Imprimimos o mapa
14    print(ganhadores_da_megasena);
15 }
16

```

Método update

O método *update*, como o próprio nome já sugere, atualiza um elemento do mapa. Mas não é só isso, ele também nos permite adicionar o elemento caso o mesmo não esteja presente na lista. Para tal é necessário informar o parâmetro opcional “*ifAbsent*”

Para exemplificar seu uso criamos uma lista de super carros, a qual contém um Porsche 911 Carrera, uma Ferrari 812 Superfast e uma BMW M8. Depois tentamos atualizar o elemento Volkswagen Fusca, o qual não está na lista. Mas utilizamos o parâmetro opcional “*ifAbsent*” para adicionar este elemento na lista.

Parâmetro	Descrição
chave	Chave a qual estamos procurando
função de atualização	A função que utilizaremos para atualizar o elemento
[função de adição]	A função que utilizaremos para adicionar o elemento caso ele não exista

```
1 void main() {
2     Map<String, int> carros = {
3         // Nome: Potência em CV
4         "Porsche 911 Carrera": 385,
5         "Ferrari 812 Superfast": 800,
6         "BMW M8": 600,
7     };
8
9     // Imprimindo a lista
10    print(carros);
11
12    // Adicionando um Fusquinha 🚗
13    carros.update("Volkswagen Fusca", (valor_atual) => 38, ifAbsent: () => 38);
14
15    // Atualizando a BMW M8
16    carros.update("BMW M8", (valor_atual) => 625);
17
18    // Imprimindo a lista atualizada
19    print(carros);
20 }
21
```

Método map

O método *map*, como o próprio nome já sugere, mapeia todos os elementos de um mapa, passando cada um desses elementos por uma função que irá retornar o elemento modificado.

Parâmetro	Descrição
função	Função que desejamos executar a cada volta do ciclo. Essa função deverá retornar um objeto <i>MapEntry</i> .

```
1 void main() {
2     Map<String, double> medias = {
3         // Aluno: Média
4         "João Paulo": 7.8,
5         "Maria Júlia": 6.9,
6         "Roberta": 8.5,
7     };
8
9     Map<String, double> medias_finais =
10         medias.map((chave, valor) => MapEntry(chave, (valor * 2) / 1.8));
11
12     print(medias_finais);
13 }
```

Método forEach

O método *forEach* navega por cada elemento da lista, realizando uma operação a cada ciclo de interação. Podemos utilizar esse método para modificar cada elemento da lista de acordo com nosso desejo ou necessidade.

Parâmetro	Descrição
-----------	-----------

função	Função que desejamos executar a cada volta do ciclo. Essa NÃO deve retornar nenhum valor
--------	--

```

1 void main() {
2     Map<String, double> medias = {
3         // Aluno: Média
4         "Marília Gabriela": 9.2,
5         "Mario": 4.5,
6         "Joana": 6.0,
7         "Clóvis": 3.2,
8         "Vinícius": 5.2,
9     };
10
11     // Criaremos três variáveis inteiras
12     int alunos_aprovados=0, alunos_em_recuperacao=0, alunos_reprovador=0;
13
14     medias.forEach((chave, nota) {
15         if (nota >= 6) {
16             alunos_aprovados++;
17         } else if (nota >= 4) {
18             alunos_em_recuperacao++;
19         } else {
20             alunos_reprovador++;
21         }
22     });
23
24     print("$alunos_aprovados ${alunos_aprovados>1}?alunos foram aprovado
s!!!:'aluno foi aprovado!!!'");
25     print("$alunos_em_recuperacao ${alunos_em_recuperacao>1}?alunos estão em
recuperação!!!:'aluno está em reprovação!!!'");
26     print("$alunos_reprovador ${alunos_reprovador>1}?alunos foram reprovado
s!!!:'aluno foi reprovado!!!'");
27 }

```

Método toString

O método toString é utilizado para converter um objeto Map em um objeto String.

```

1 void main() {
2     Map<String, double> medias = {
3         // Aluno: Média
4         "Marília Gabriela": 9.2,
5         "Mario": 4.5,
6         "Joana": 6.0,
7         "Clóvis": 3.2,
8         "Vinícius": 5.2,
9     };
10
11     print(medias.toString());
12 }

```

Objeto Future

Um objeto **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.

Palavra-chave await

A palavra-chave *await* é utilizada para definirmos que desejamos aguardar a resposta que será fornecida por uma função que retorna um objeto **Future**. Caso não utilizemos tal palavra-chave, o código continuará sendo executado, não havendo uma parada para aguardar o recebimento da resposta da mesma.

Palavra-chave async

Sempre que fazemos uso de operações demoradas como acesso a um banco de dados ou a um serviço Web, necessitamos informar ao Dart que desejamos executar tal operação de forma assíncrona, ou seja, de forma independente do fluxo principal da aplicação. Dessa forma podemos dizer que nosso programa poderá continuar rodando sem a necessidade de aguardar a conclusão de tal operação.

No Dart, quando desejamos informar que uma função irá receber um objeto em um momento futuro, devemos informar que tal função é assíncrona. Para tal devemos utilizar a palavra chave **async** logo após a declaração dos parâmetros da mesma.

Requisição assíncrona

Agora que já entendemos um pouco sobre o uso do objeto **Future** e sobre as palavras-chave **async** e **await**, vamos a um exemplo prático para entender seu uso. Em nosso exemplo, vamos criar uma função chamada **numeroAleatorio**, a mesma irá retornar um número inteiro aleatório após o intervalo de 1 segundo. Sendo assim nossa função *main* deverá aguardar a conclusão da mesma, para somente após exibir o valor no console.



```
1 import 'dart:math' as mat;
2
3 void main() async {
4   print(await numeroAleatorio());
5 }
6
7 Future<int> numeroAleatorio() {
8   return Future.delayed(Duration(seconds: 1)).then(
9     (numero) => mat.Random().nextInt(10),
10  );
11 }
12
```