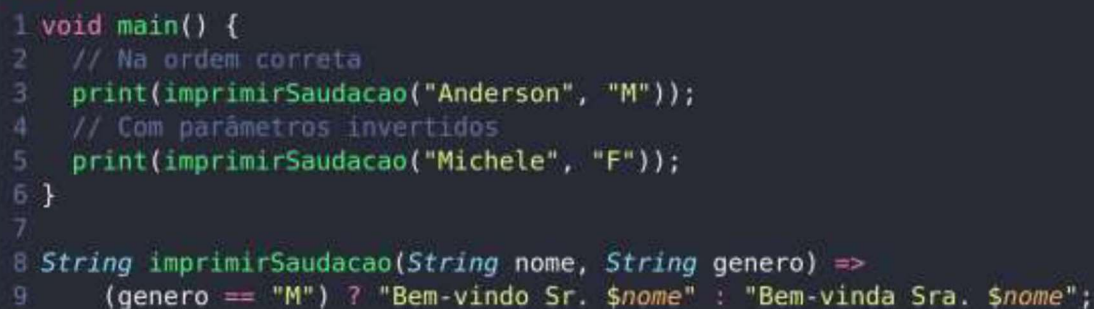


Arrow Function

Outro aspecto importante a ser ressaltado no Dart é o fato do suporte à “Arrow function”, também conhecida como “lambda function” ou função anônima.

Mas a critério de resumo, esse tipo de declaração nos permite criar funções com uma sintaxe mais simples e limpa, o que deixa nosso código menor e consequentemente mais fácil de entender e de dar manutenção.

Para nosso exemplo, vamos recriar nosso exemplo anterior, desta vez utilizando arrow function e operador ternário. Dessa forma poderemos reduzir as linhas de nossa função, tornando-a mais simples e fácil de compreender.



```
1 void main() {  
2   // Na ordem correta  
3   print(imprimirSaudacao("Anderson", "M"));  
4   // Com parâmetros invertidos  
5   print(imprimirSaudacao("Michele", "F"));  
6 }  
7  
8 String imprimirSaudacao(String nome, String genero) =>  
9   (genero == "M") ? "Bem-vindo Sr. $nome" : "Bem-vinda Sra. $nome";
```

Método construtor

Um construtor é um método especial, invocado de forma automática, que tem como objetivo preparar e inicializar um objeto, definindo seu comportamento inicial, ou seja, definir quais propriedades serão inicializadas e/ou quais métodos serão chamados no momento da criação do objeto.

A declaração de um método construtor em Dart não é nada diferente das aplicadas em outras linguagens de programação, onde devemos informar os parâmetros de inicialização, se existirem, já que não são obrigatórios, é propriamente seu comportamento no momento de sua inicialização.

Vejamos um exemplo prático onde definimos que nossos médicos devem possuir um nome, um CRM, e especialidade definida no momento de sua inicialização.

```

1 class Medico {
2   int? _CRM;
3   String? _nome;
4   List<String>? _especialidades;
5
6   Medico(int CRM, String nome, List<String> especialidades) {
7     this._CRM = CRM;
8     this._nome = nome;
9     this._especialidades = especialidades;
10  }
11 }
12

```

No exemplo acima podemos ver a forma tradicional de declaração do método construtor da classe. Mas o Dart nos oferece uma forma ainda mais simples e limpa para que possamos declarar nosso método construtor. Vejamos como podemos fazê-lo.

```

1 class Medico {
2   int? _CRM;
3   String? _nome;
4   List<String>? _especialidades;
5
6   Medico(this._CRM, this._nome, this._especialidades);
7 }
8

```

Muito mais simples e eficiente. Não concorda? Mas ainda não acabou, ainda podemos usar parâmetros nomeados em nosso construtor, assim podemos declarar que apenas o nome e o CRM são obrigatórios.

```

1 class Medico {
2     int CRM;
3     String nome;
4     List<String>? especialidades;
5     Medico(this.CRM, this.nome, {this.especialidades});
6 }
7

```

Método toString

O método `toString` é um método herdado da classe objeto. Ele é utilizado para converter todas as propriedades de um objeto em uma string e retorná-la. Esse tipo de método é muito utilizado como uma forma de permitir a impressão de objeto em um formato de string.

```

1 class Medico {
2     int? CRM;
3     String? nome;
4     List<String>? especialidades;
5
6     Medico({this.CRM, this.nome, this.especialidades});
7
8     @override
9     String toString() {
10         return "Dr(a). ${this.nome} tem o CRM ${this.CRM} e tem as seguintes especialidades ${this.especialidades}";
11     }
12 }
13

```