

## OPERADORES RELACIONAIS

Outro aspecto muito comum e primordial em qualquer linguagem de programação é a capacidade de relacionar e comparar valores. E com o Dart não é diferente, pois com ele é possível comparar valores de forma simples e extremamente prática.

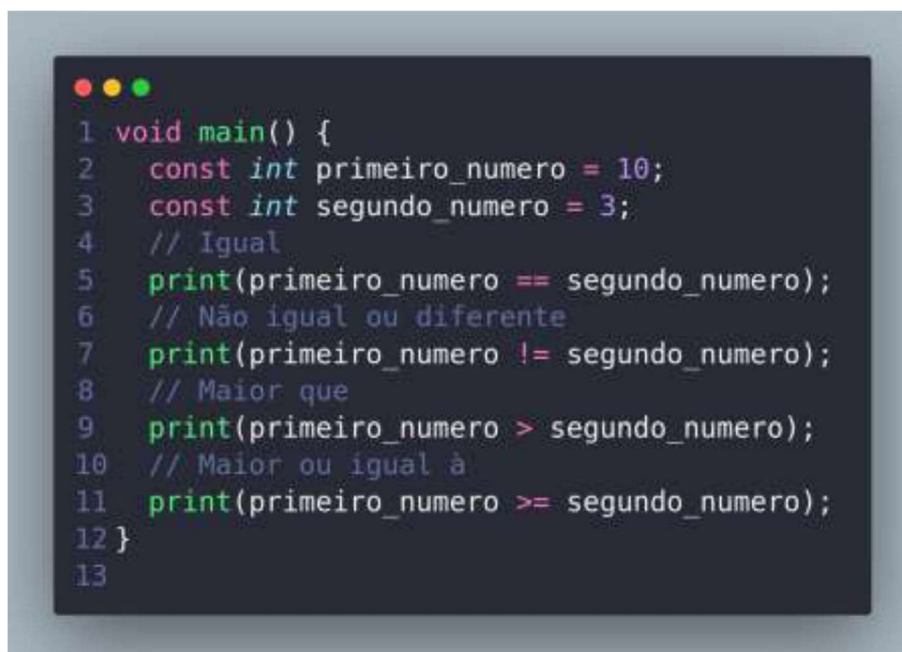
Na tabela podemos ver esses operadores, assim como uma breve explicação sobre seu significado na linguagem.

**Tabela - Operadores relacionais no Dart**

| Operador | Significado            | Objetivo prático   |
|----------|------------------------|--|
| ==       | Igual                  | Verificar se o valor à esquerda do operador é <b>igual</b> ao que está à direita do mesmo.       |
| !=       | Não igual ou diferente | Verificar se o valor à esquerda do operador é <b>diferente</b> do que está à direita do mesmo.   |
| >        | Maior que              | Verificar se o valor à esquerda do operador é <b>maior</b> que está à direita do mesmo.          |
| <        | Menor que              | Verificar se o valor à esquerda do operador é <b>menor</b> que está à direita do mesmo.          |
| >=       | Maior ou igual à       | Verificar se o valor à esquerda do operador é <b>maior ou igual</b> que está à direita do mesmo. |
| <=       | Menor ou igual à       | Verificar se o valor à esquerda do operador é <b>menor ou igual</b> que está à direita do mesmo. |

Agora que já entendemos o que são os operadores relacionais, vamos a um exemplo prático de seu uso. Em nosso exemplo vamos declarar duas variáveis do tipo inteiro com os seguintes valores 10 e 3, respectivamente.

Feito isso, vamos compará-las a fim de descobrir o resultado de tais comparações. O resultado final pode ser visto abaixo.



```
1 void main() {
2     const int primeiro_numero = 10;
3     const int segundo_numero = 3;
4     // Igual
5     print(primeiro_numero == segundo_numero);
6     // Não igual ou diferente
7     print(primeiro_numero != segundo_numero);
8     // Maior que
9     print(primeiro_numero > segundo_numero);
10    // Maior ou igual à
11    print(primeiro_numero >= segundo_numero);
12 }
13
```

Resultado:

false

true

true

true

---

## OPERADORES LÓGICOS

Outro grupo de operadores muito importante para qualquer linguagem de programação são os operadores lógicos. Esses operadores são capazes de comparar conjuntos de operadores relacionais, resultando em um valor booleano.

Na tabela podemos ver esses operadores, assim como uma breve explicação sobre seu significado na linguagem.

Tabela - Operadores lógicos no Dart

| Operador   | Significado  |
|------------|--|
| !expressão | <b>Inverte o resultado lógico</b> (Verdadeiro se torna falso e falso se torna verdadeiro).                             |
|            | Retorna <b>verdadeiro</b> caso ao menos uma das comparações tiverem um valor verdadeiro. Caso contrário retorna falso. |
| &&         | Retorna <b>verdadeiro</b> caso todas as comparações tiverem um valor verdadeiro. Caso contrário retorna falso.         |

Agora que já entendemos o que são os operadores lógicos, vamos a um exemplo prático de seu uso. Em nosso exemplo vamos declarar três variáveis do tipo inteiro com os seguintes valores 10, 3 e 14, respectivamente.

Feito isso, vamos compará-las a fim de descobrir o resultado de tais comparações. O resultado final pode ser visto abaixo.

```

1 void main() {
2   const int primeiro_numero = 10;
3   const int segundo_numero = 3;
4   const int terceiro_numero = 14;
5   // O primeiro número é maior que os demais
6   print(primeiro_numero > segundo_numero && primeiro_numero > terceiro_numero);
7   // O segundo número é maior que o primeiro OU menor que o segundo
8   print(segundo_numero > primeiro_numero || segundo_numero < terceiro_numero);
9   // O primeiro número NÃO é maior que os demais
10  print(!(primeiro_numero > segundo_numero && primeiro_numero > terceiro_numero));
11 }
12

```

Resultado:

false

true

true

## OPERADORES DE TESTE

---

Os operadores de teste são muito úteis quando necessitamos realizar verificações e associações em tempo de execução.

Esses operadores também podem ser utilizados para realizar uma conversão explícita de dados, o que pode ser muito útil em determinados casos e operações do cotidiano de nossas aplicações.

Na tabela podemos ver esses operadores, assim como uma breve explicação sobre seu significado na linguagem.

**Tabela - Operadores de teste no Dart**

| Operador | Significado   |
|----------|---|
| as       | Conversão de tipos (Também utilizado para criação de prefixos em importações) |
| is       | Retorna True caso um objeto seja de um determinado tipo                       |
| is!      | Retorna False caso um objeto seja de um determinado tipo                      |

```
1 void main() {
2   const num primeiro_numero = 10.15;
3   const num segundo_numero = 3;
4   final double terceiro_numero = (primeiro_numero as double) * 2;
5   print(primeiro_numero is double);
6   print(segundo_numero is double);
7   print(terceiro_numero is double);
8 }
9
```

Resultado:

true

false

true

## ESTRUTURAS DE DECISÃO

---

Muitas são as vezes em que necessitamos comparar valores e objetos em nossos sistemas. E como vimos, o Dart está mais que preparado para tal. Mas até o momento só foi possível comparar tais valores, não podemos tomar decisões com base nesses resultados. É aí que entram as estruturas de decisão. Também conhecidas como estruturas condicionais ou estruturas de fluxo.

Uma estrutura de decisão, como o próprio nome já sugere, nos permite tomar uma decisão com base em um resultado booleano, seja ele vindo de uma comparação ou mesmo do valor de uma variável ou constante do sistema.

### ***Estrutura IF***

---

A estrutura IF é a mais comum e usual estrutura de decisão, e está presente em praticamente todas as linguagens de programação. Conhecer seu funcionamento é fundamental para qualquer programador, seja ele um iniciante ou experiente no universo do desenvolvimento.

Uma estrutura IF é composta por duas palavras chave, são eles: *if* e *else*. O uso destas duas palavras chave, assim como suas "sub combinações", nos permitem decidir como nosso sistema irá reagir a uma determinada situação. Por exemplo, com uso desta estrutura podemos decidir se um determinado bloco de comandos ou cálculo será executado ou não.

A criação de uma estrutura *IF* é bastante simples, podemos fazer uso da palavra-chave *if*, onde passamos a condição para sua execução, e caso essa condição seja verdadeira, o bloco de comandos referentes a essa estrutura será executado.

No exemplo abaixo podemos ver um exemplo básico do uso de uma estrutura condicional *IF*.

```
1 void main() {
2     const int primeiro_valor = 22;
3     const int segundo_valor = 12;
4     const bool e_maior = primeiro_valor > segundo_valor;
5
6     if (e_maior) {
7         print("O primeiro valor é maior que o segundo");
8     }else{
9         print("O segundo valor é maior que o primeiro");
10    }
11}
12
```

Outro aspecto que vale ressaltar é que podemos combinar as duas palavras chaves que compõem a estrutura IF para formar mais cenários possíveis.

```
1 void main() {
2     const int primeiro_valor = 22;
3     const int segundo_valor = 22;
4     const bool e_maior = primeiro_valor > segundo_valor;
5
6     if (e_maior) {
7         print("O primeiro valor é maior que o segundo");
8     }else{
9         print("O segundo valor é maior que o primeiro");
10    }
11}
12
```

Logo temos um pequeno problema, pois o segundo valor não é maior que o primeiro, pois ambos são iguais. Mas o que podemos fazer para resolver isso?

E a resposta é bem simples, vamos verificar a possibilidade de ambos serem iguais. Para isso vamos adicionar mais uma condição à estrutura, essa condição será a palavra chave *"else if"*

```
1 void main() {
2     const int primeiro_valor = 22;
3     const int segundo_valor = 22;
4     const bool e_maior = primeiro_valor > segundo_valor;
5     const bool sao_iguais = primeiro_valor == segundo_valor;
6
7     if (e_maior) {
8         print("O primeiro valor é maior que o segundo");
9     } else if (sao_iguais) {
10        print("Os números são iguais");
11    } else {
12        print("O segundo valor é maior que o primeiro");
13    }
14}
15
```