06

Faça como eu fiz

Nesta lição, nós aprendemos que podemos utilizar ranges para simplificar condições que visam identificar intervalos:

```
val base = 1500.0
val topo = 4000.0
var contador = 0
for (salario in salarios) {
   if(salario in base..topo) {
       contador++
   }
}
```

Utilizando funções agregadoras em conjunto com ranges, podemos simplificar esse código para: `val contador = salarios.count { it in base..topo }` Aprendemos, então, a utilizar outras funções agregadoras e de processamento de coleções, como uma forma mais concisa de encontrar o maior valor dentro de um array:

```
val maiorIdade: Int = idades.max()
```

E também o menor e a média:

```
val menorIdade: Int = idades.min()
val media: Double = idades.average()
```

Para identificar se **todos** ou **qualquer um** dos elementos de uma lista se adequam a determinada condição, utilizamos respectivamente as funções `all` e `any`:

```
val todosMaiores: Boolean = idades.all { it >= 18 }
val existeMenor: Boolean = idades.any { it < 18 }
```

Vimos como filtrar valores utilizando a função `filter`:

```
val maiores: List<Int> = idades.filter { it >= 18 }
val menores: List<Int> = idades.filter { it < 18 }
```

Essa função, no entanto, retorna uma lista. Caso queira continuar trabalhando com `IntArray`, utilize a função `toIntArray()`:

```
val menores: IntArray = idades.filter { it < 18 }.toIntArray()
```

Para verificar se um valor específico está contido no array, ou fazer uma busca por esse valor, vimos como usar, respectivamente, as funções `contains` e `find`:

```
val contemDez: Boolean = idades.contains(10)
val idade: Int? = idades.find { it == 10 }
```

Para fazer o cálculo dos salários do tipo BigDecimal, primeiro criamos uma função para criar o array de BigDecimal:

```
fun bigDecimalArrayOf(vararg valores: String): Array<BigDecimal> {
    return Array<BigDecimal>(valores.size) { i ->
        valores[i].toBigDecimal()
    }
}
```

Com essa função pronta, nós fomos capazes de criar nosso array de salários:

```
val salarios = bigDecimalArrayOf("1500.55", "2000.00", "5000.00", "10000.00")
```

Depois, vimos como calcular um aumento relativo a cada salário, em que salários maiores que R$ 5000,00 recebem um aumento de 10% e salários menores recebem um aumento fixo de R$ 500,00:

```
private fun calculaAumentoRelativo(salario: BigDecimal, aumento: BigDecimal)
        : BigDecimal {
    return if (salario < "5000".toBigDecimal()) {
        salario + "500".toBigDecimal()
    } else {
        (salario * aumento).setScale(2, RoundingMode.UP)
    }
}
```

Utilizamos, então, a função `map` do Kotlin para realizar uma transformação do nosso array de salários atual para um novo array de salários com aumento:

```
val aumento = "1.1".toBigDecimal()
val salariosComAumento: Array<BigDecimal> = salarios
        .map { salario -> calculaAumentoRelativo(salario, aumento) }
        .toTypedArray()
```