Em **Kotlin** decisões como **if, else if** e **else** se mantém igual ao Java. Mas existe uma alternativa própria dela para encurtar essa estrutura, o que a torna mais elegante, essa alternativa se chama **when**:

```
if(saldo.compareTo(BigDecimal( val: 0.0)) == 1) {
    println("Ta positivo na conta")
}else if(saldo.compareTo(BigDecimal( val: 0.0)) == -1){
    println("Ta negativado na conta")
}else{
    println("Não deve, nem tem, ta zerado")
}
```

```
when{
    saldo.compareTo(BigDecimal( val: 0.0)) == 1 -> println("Ta positivo na conta")
    saldo.compareTo(BigDecimal( val: 0.0)) == -1 -> println("Ta negativado na conta")
    else -> println("Não deve, nem tem, ta zerado")
}
```

A grande vantagem do **when** está bem atrelado a ***diminuir boilerplate***, também é possível abrir chaves no escopo das condições e adicionar mais coisas a serem feitas além de um print.

Iterações em **Kotlin** são um pouco mais diferentes do que o usual em algumas linguagens, são chamados de **For Loops**. Um jeito de se ***iterar*** em **Kotlin** é usando **"ranges"**, esse range fala pra gente onde a iteração **começa e onde ela termina**:

```
for ( i in 1 ≤ .. ≤ 3){
    println(i)
}
```

```
1
2
3
```

Kotlin também tem uma **sintaxe pra fazer um loop de forma decrescente**, em outras linguagens a gente muda o valor das variáveis de manipulação do looping, mas **kotlin** faz isso de **forma sintática** onde eu posso **dizer explicitamente**: a *partir desse número*, faça de *forma decrescente* de *tanto em tanto:*

```
for ( i in 6 ≥ downTo ≥ 1 step 2){
    println(i)
}
```

```
6
4
2
```

Break e **continue** funcionam igual a outras linguagens, break para o laço quando atinge determinada condição e continue pula a iteração atual para a próxima iteração.

While também funciona como em outras linguagens.