Sumário

[Condições e Loops 1](#_Toc78210682)

[Relembrando o if 1](#_Toc78210683)

[When 2](#_Toc78210684)

[For loops﻿ 3](#_Toc78210685)

[forEach e iterator 4](#_Toc78210686)

[Iterando por meio de uma matriz 4](#_Toc78210687)

[Iterando pela String 4](#_Toc78210688)

[While loops﻿ 5](#_Toc78210689)

# Condições e Loops

## Relembrando o if

**Apenas if**

fun main() {

val a = 1

val b = 2

var max = a

if (a < b) max = b

println(max)

}

**If-else**

fun main() {

val a = 1

val b = 2

var max: Int

if (a > b) {

max = a

} else {

max = b

}

println(max)

}

**Como expressão**

fun main() {

val a = 1

val b = 2

val max = if (a > b) a else b

println(max)

}

# When

When define uma expressão condicional com várias ramificações. É semelhante ao switch, instrução encontrada em linguagens semelhantes a C. Sua forma simples é parecida com esta.

fun main() {

val x = 3

when (x) {

1 -> print("x == 1")

2 -> print("x == 2")

else -> {

print("x não é nem 1 e nem 2")

}

}

}

Para definir um comportamento comum para vários casos, combine suas condições em uma única linha com uma vírgula:

fun main() {

var x = 1

when (x) {

0, 1 -> print("x == 0 or x == 1")

else -> print("outra coisa")

}

}

Você também pode verificar um valor por estar ou não dentro de um intervalo ou coleção:

fun main() {

var x = 17

var numerosvalidos = 11..15

when (x) {

in 1..10 -> print("x esta no intervalo") // entre 1 e 10

in numerosvalidos -> print("x é valido") // nos numeros validos

!in 16..20 -> print("x esta fora do intervalo") //não estiver entre 16 e 20

else -> print("nenhuma das anteriores") // nenhum dos anteriores

}

}

# For loops﻿

O loop for no Kotlin itera por meio de qualquer coisa que forneça um iterador. Neste artigo, você aprenderá a criar um loop for (com a ajuda de exemplos).

Não existe um loop for tradicional no Kotlin, ao contrário do Java e de outras linguagens.

No Kotlin, o loop for é usado para iterar por meio de intervalos, matrizes, mapas e assim por diante (qualquer coisa que forneça um iterador).

A sintaxe do loop for no Kotlin é:

for (item in collection) {

// body of loop

}

Para iterar em um intervalo de números, use uma expressão de intervalo:

fun main(){

for (i in 1..3) {

println(i)

}

}

Aqui, o loop itera através do intervalo e imprime itens individuais.

Se o corpo do loop contém apenas uma instrução (como o exemplo acima), não é necessário usar chaves {}.

fun main() {

for (i in 1..5) println(i)

}

Para iterar na ordem reversa os números, existe a função downTo ao invés de .., como mostra o exemplo abaixo:

for (i in 1..4) print(i) //1,2,3,4

for (i in 4 downTo 1) print(i) //4,3,2,1

É possível definir também o salto (o padrão é um) para uma iteração de um range, usando a sintaxe step.

for (i in 1..8 step 2) print(i)

println()

for (i in 8 downTo 1 step 2) print(i)

Como podemos observar no exemplo acima, os intervalos de um range são inclusivos. Caso queira excluir o último elemento deve ser usado a função until, como no exemplo abaixo:

for (i in 1 until 10) { // i in [1, 10), 10 is excluded

print(i) }

Exemplo: diferentes maneiras de iterar em um intervalo

fun main() {

print("for (i in 1..5) print(i) = ")

for (i in 1..5) print(i)

println()

print("for (i in 5..1) print(i) = ")

for (i in 5..1) print(i) // não printa nada

println()

print("for (i in 5 downTo 1) print(i) = ")

for (i in 5 downTo 1) print(i)

println()

print("for (i in 1..5 step 2) print(i) = ")

for (i in 1..5 step 2) print(i)

println()

print("for (i in 5 downTo 1 step 2) print(i) = ")

for (i in 5 downTo 1 step 2) print(i)

}

**Saida**

for (i in 1..5) print(i) = 12345

for (i in 5..1) print(i) =

for (i in 5 downTo 1) print(i) = 54321

for (i in 1..5 step 2) print(i) = 135

for (i in 5 downTo 1 step 2) print(i) = 531

## forEach e iterator

Para iterar um range também existe a possibilidade de utilizar um forEach

fun main() {

(1..5).forEach(::println)

(1..5).reversed().forEach { e -> print("$e ") }

}

## Iterando por meio de uma matriz

Aqui está um exemplo para iterar por meio de uma matriz String.

fun main() {

var language = arrayOf("Ruby", "Kotlin", "Python", "Java")

for (item in language)

println(item)

}

## Iterando pela String

fun main() {

var text= "Kotlin"

for (letter in text) {

println(letter)

}

}

**saida**

# while

A segunda e última estrutura de repetição disponibilizada pela linguagem é while. Ela possui duas variações, sendo a primeira while e a segunda do..while. A sintaxe dessa estrutura podemos ver, onde expressão-lógica deve ser avaliada como verdadeira para que a repetição continue.

while([expressão-lógica]) {

[código]

}

do {

[código]

} while([expressão-lógica])

A principal diferença entre while ou do..while está na forma como eles verificam se a repetição deve acontecer.

Enquanto while primeiro avalia a expressão lógica e depois executa a repetição, do..while faz o oposto, primeiro executando a repetição e depois avaliando a expressão lógica.

Dessa forma, é certo que do..while repetirá pelo menos uma vez, enquanto while pode nunca se repetir.

Por exemplo, temos a mesma expressão lógica sendo aplicada a while e do..while.

while (false) {

println("Olá mundo!")

}

do {

println("Olá mundo!")

} while (false)

Como resultado, no caso de while a mensagem Olá mundo! nunca será impressa, uma vez que a expressão lógica logo é avaliada como falsa. Em do..while a mensagem Olá mundo! será impressa uma vez, então a expressão lógica será avaliada como falsa e as repetições se encerrarão.

A principal diferença entre while e for é que no caso de while o controle de quando as repetições se encerrarão fica com o programador. Com while devemos nós mesmos nos certificar de que, em algum momento, as repetições se encerrarão, o que normalmente é feito utilizando variáveis de controle. Sendo assim, um código para percorrer uma série de número utilizando while poderia ser este:

var nums = 0

val serie = 1..9

while (nums < serie.last) {

println(serie.elementAt(nums))

nums++

}

Aqui utilizamos a variável de controle nums, que vai sendo incrementada a cada repetição. Ela é utilizada também na expressão lógica nums < serie.last para determinar que as repetições devem se encerrar assim que nums chegar ao valor 9.

Além disso, note que agora precisamos acessar o valor que corresponde a repetição atual na série a partir de um índice.

A função elementAt() é utilizada para acessar um número na série a partir da sua posição. Aqui e em muitas outras coleções de valores a primeira posição é 0. Sendo assim, na primeira repetição nums terá o valor 0 e o valor acessado na série estará no índice 0, que contém o número 1. Na próxima repetição, nums terá o valor 1 e acessará o valor no índice 1 da série, que é 2. Isso continuará acontecendo até que nums assuma o valor 8, índice que contém o valor 9 na série.