

Bootcamp - Android

Roshka - 2022



Operadores lógicos Python vs Kotlin

| Python | Kotlin |
|----------------|-----------------------|
| a and b | a && b |
| a or b | a b |
| not a | ! a |

Condicionales: if

El condicional **if** se define de la siguiente manera y puede definirse también como expresión:

```
var max = a
if (a < b) max = b

// con un else
if (a > b) {
  max = a
} else {
  max = b
}

// como expresion
max = if (a > b) a else b
```

Palabra clave

La condición a evaluar

Bloque a ejecutar

Condicionales: when

El condicional **when** es parecida al switch de otros lenguajes, se define de la siguiente manera:

Palabra clave

when

(x)

{

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

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

else -> {

print("x is neither 1 nor 2")

}

}

Expresión a evaluar

Valores
posibles

Arrays

Los arreglos en Kotlin están representados por la **clase Array**. Tiene funciones `get` y `set` que se convierten en `[]` por convenciones de sobrecarga de operadores y la propiedad de tamaño, junto con otros métodos útiles.




```
class Array<T> private constructor() {  
    val size: Int  
    operator fun get(index: Int): T  
    operator fun set(index: Int, value: T): Unit  
    operator fun iterator(): Iterator<T>  
}
```

Arrays

Para crear una array, use la función `arrayOf()` y se pasan los valores de los elementos, de modo que `arrayOf(1, 2, 3)` cree una matriz `[1, 2, 3]`.


Otra opción es usar el constructor `Array` que toma el tamaño de la matriz y la función que devuelve los valores de los elementos de la matriz dado su índice.



```
//[1,2,3]
var a = arrayOf(1,2,3)
// ["0", "1", "4", "9", "16"]
val asc = Array(5) { i -> (i * i).toString()
}
```

Arrays primitivos


Kotlin también tiene clases que representan matrices de tipos primitivos sin sobrecarga: `ByteArray`, `ShortArray`, `IntArray`, etc. Cada uno de ellos también tiene una función de fábrica correspondiente:



```
val x: IntArray = intArrayOf(1, 2, 3)
val arr = IntArray(3) // [0,0,0]
print(x.contentToString())
```

Ciclos: for

El ciclo **for** itera a través de cualquier cosa que proporcione un **iterator** y también puede usarse como un for en bloque. La sintaxis de for es la siguiente:




```
for (item in collection) print(item)

for (item: Int in ints) {
    // ...
}
```


Ciclos: for


También podemos utilizar el for para ciclar en un rango de valores.



```
for (i in 1..3) {  
    println(i)  
}  
for (i in 6 downTo 0 step 2) {  
    println(i)  
}
```

Ciclos: for

Si desea iterar a través de una matriz o una lista con un índice, puede hacerlo de esta manera:



```
for (i in array.indices) {  
    println(array[i])  
}
```

Ciclos: while

Los bucles **while** y **do-while** ejecutan su cuerpo continuamente mientras se cumple su condición.



```
while (x > 0) {  
    x--  
}  
  
do {  
    val y = retrieveData()  
} while (y != null)
```