Sumário

[Palavras Reservadas 1](#_Toc77780312)

[Termos 2](#_Toc77780313)

[Variáveis 2](#_Toc77780314)

[Inferência de Tipos ou Dedução 3](#_Toc77780315)

[Matrizes 4](#_Toc77780316)

[A Função arrayOf() 4](#_Toc77780317)

[O Construtor Array() 4](#_Toc77780318)

[Comentários 5](#_Toc77780319)

[Operadores 5](#_Toc77780320)

[Operadores Aritmeticos 5](#_Toc77780321)

[Concatenação 6](#_Toc77780322)

[Operadores de atribuição 6](#_Toc77780323)

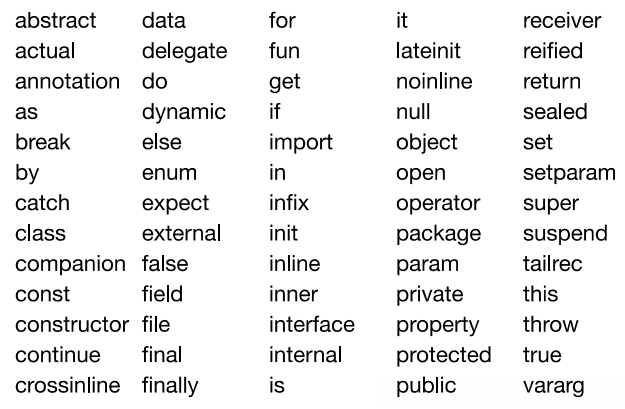
[Operadores Unarios e de Incremento 7](#_Toc77780324)

[Comparação e Igualdade 8](#_Toc77780325)

[Operadores Logicos 9](#_Toc77780326)

[Referencias 10](#_Toc77780327)

# Palavras Reservadas



# Termos



# Variáveis

Em Kotlin, use val para declarar uma constante ou a palavra-chave var para declarar uma variável. Você pode especificar um tipo como String ou Int após o nome da variável. No exemplo abaixo, nós declaramos uma constante firstName do tipo String com a palavra-chave val.

**val firstName: String = "Caique"**

Mas você logo vai perceber que em Kotlin, geralmente é possível omitir o tipo de declaração e o compilador não vai reclamar.

**val lastName = "Zaneti" // ainda compila**

No código acima, você vai observar que nós não declarar explicitamente o tipo String. O código acima ainda funcionará porque o compilador inferiu implicitamente o tipo usando a inferência de tipos.

A diferença entre as palavras-chave val e var é que a primeira é imutável ou somente leitura (seu valor não pode ser alterado), enquanto a última é mutável (seu valor pode ser alterado).

**val dateOfBirth = "29th March, 1709"**

**dateOfBirth = "25th December, 1600" // nao pode ser mudado**

**var car = "Toyota Matrix"**

**car = "Mercedes-Maybach" // pode mudar**

Observe que para uma variável declarada com a palavra-chave var que tem seu tipo inferido pelo compilador, atribuir um valor de um tipo diferente, não funcionará. Em outras palavras, o valor da variável pode mudar, mas não seu tipo! Por exemplo:

**var age = 12**

**age = "12 years old" // Error: type mismatch**

É altamente recomendável que você comece fazendo suas variáveis imutáveis, declarando-as com a palavra-chave val, para não manter muitos estados. Isso torna seu código mais seguro para multithreading, porque garante que suas variáveis não podem ser modificadas por outras threads inesperadamente.

Outra coisa que você deve saber sobre a palavra-chave val é que você pode declará-la com um tipo só e atribuir um valor mais tarde. Mas você ainda só pode atribuir um valor de cada vez.

**val carName: String**

**carName = "Toyota Matrix" // will compile**

Em Java, é possível declarar múltiplas variáveis do mesmo tipo em uma única linha, mas isso **não funciona no Kotlin**. Em Kotlin, todas as declarações de variável devem estar em suas próprias linhas.

**val carName = "BMW", streetName = "Oke street" // não compila**

**// isso compila**

**var carName = "BMW"**

**var streetName = "Oke street"**

# Inferência de Tipos ou Dedução

Kotlin é uma linguagem fortemente tipada que oferece suporte a inferência de tipos ou dedução. Este é o mecanismo utilizado pelo compilador para descobrir os tipos do contexto.

O Java não tem um mecanismo de inferência de tipos, o que significa que você deve declarar explicitamente o tipo de cada variável ou função. Inferência de tipos ajuda a reduzir o código clichê que você tem que escrever.

**val country = "Nigeria" // type is inferred by compiler**

**val code = 234**

O código acima compilaria mesmo que nós não indicássemos explicitamente o tipo para a variável país. O compilador é suficientemente inteligente para saber que o país é do tipo String, porque o valor, "Nigéria", é uma sequência de caracteres.

# Matrizes

Em Kotlin, há duas maneiras principais para criar uma matriz: usando a função auxiliar arrayOf() ou o construtor Array().

## A Função arrayOf()

Por exemplo, vamos criar uma matriz com alguns elementos usando arrayOf().

**val myArray = arrayOf(4, 5, 7, 3)**

Agora, para acessar qualquer um dos elemento, podemos usar seu índice: myArray[2]. Observe que podemos passar valores de tipos diferentes para o arrayOf() como argumentos e ainda funcionará — será uma matriz de tipo misto.

**val myArray = arrayOf(4, 5, 7, 3, "Chike", false)**

Para forçar que todos os valores da matriz tenham o mesmo tipo, por exemplo Int, declaramos o tipo chamando arrayOf<Int>() ou intArrayOf().

**val myArray3 = arrayOf<Int>(4, 5, 7, 3, "Chike", false) // will not compile**

**val myArray4 = intArrayOf(4, 5, 7, 3, "Chike", false) // will not compile**

Temos também outras funções utilitárias para criar matrizes de outros tipos, como charArrayOf(), booleanArrayOf(), longArrayOf(), shortArrayOf(), byteArrayOf() e assim por diante. Nos bastidores, usar essas funções irá criar uma matriz de seus respectivos tipos primitivos Java. Em outras palavras, intArrayOf() irá compilar para o tipo primitivo normal de Java int[], byteArrayOf() será byte[], longArrayOf() será long[], e assim por diante.

## O Construtor Array()

Agora vamos ver como criar uma matriz com Array(). O construtor dessa classe requer um tamanho e uma função lambda. Vamos aprender mais sobre funções lambda mais tarde nesta série, mas por enquanto, apenas entenda que é uma maneira simples de declarar uma função anônima inline. Neste caso, o trabalho da função lambda é inicializar a matriz com os elementos.

**val numbersArray = Array(5, { i -> i \* 2 })**

No código acima, nós passamos 5 como o tamanho da matriz no primeiro argumento. O segundo argumento leva uma função lambda, que recebe o índice do elemento da matriz e retorna o valor a ser inserido no índice na matriz. Assim, no exemplo acima, criamos uma matriz com elementos 0, 2, 4, 6 e 8.

# Comentários

Isso é fácil. Em Kotlin, os comentários são os mesmos que em Java. Podemos usar comentários de bloco ou linha:

**/\***

**bloco**

**\*/**

**// linha**

# Operadores

## Operadores Aritmeticos

|  |  |
| --- | --- |
| **Operador** | **Significado** |
| + | Adição e Concatenação |
| - | Subtração |
| \* | Multiplicação |
| / | Divisão |
| % | Resto |

fun main(args: Array<String>) {

val number1 = 12.5

val number2 = 3.5

var result: Double

result = number1 + number2

println("number1 + number2 = $result")

result = number1 - number2

println("number1 - number2 = $result")

result = number1 \* number2

println("number1 \* number2 = $result")

result = number1 / number2

println("number1 / number2 = $result")

result = number1 % number2

println("number1 % number2 = $result")

}

saida

number1 + number2 = 16.0

number1 - number2 = 9.0

number1 \* number2 = 43.75

number1 / number2 = 3.5714285714285716

number1 % number2 = 2.0

## Concatenação

fun main(args: Array<String>) {

val start = "Talk is cheap. "

val middle = "Show me the code. "

val end = "- Linus Torvalds"

val result = start + middle + end

println(result)

}

saida

Talk is cheap. Show me the code. - Linus Torvalds

# Operadores de atribuição

Operadores de atribuição são usados ​​para atribuir valor a uma variável.

val age = 5

| expressão | equivalência | tradução |
| --- | --- | --- |
| a +=b | a = a + b | a.plusAssign(b) |
| a -= b | a = a - b | a.minusAssign(b) |
| a \*= b | a = a \* b | a.timesAssign(b) |
| a /= b | a = a / b | a.divAssign(b) |
| a %= b | a = a % b | a.modAssign(b) |

fun main(args: Array<String>) {

var number = 12

number \*= 5 // number = number\*5

println("number = $number")

}

saida

number = 60

# Operadores Unarios e de Incremento

| Operador | Significado | Expression | Translates to |
| --- | --- | --- | --- |
| + | Unary plus | +a | a.unaryPlus() |
| - | Unary minus (inverts sign) | -a | a.unaryMinus() |
| ! | not (inverts value) | !a | a.not() |
| ++ | Increment: increases value by1 | ++a | a.inc() |
| -- | Decrement: decreases value by 1 | --a | a.dec() |

fun main(args: Array<String>) {

val a = 1

val b = true

var c = 1

var result: Int

var booleanResult: Boolean

result = -a

println("-a = $result")

booleanResult = !b

println("!b = $booleanResult")

--c

println("--c = $c")

}

saida

-a = -1

!b = false

--c = 0

# Comparação e Igualdade

| Operator | Meaning | Expression | Translates to |
| --- | --- | --- | --- |
| > | greater than | a > b | a.compareTo(b) > 0 |
| < | less than | a < b | a.compareTo(b) < 0 |
| >= | greater than or equals to | a >= b | a.compareTo(b) >= 0 |
| <= | less than or equals to | a < = b | a.compareTo(b) <= 0 |
| == | is equal to | a == b | a?.equals(b) ?: (b === null) |
| != | not equal to | a != b | !(a?.equals(b) ?: (b === null)) |

fun main(args: Array<String>) {

val a = -12

val b = 12

// use of greater than operator

val max = if (a > b) {

println("a is larger than b.")

a

} else {

println("b is larger than a.")

b

}

println("max = $max")

}

saida

b is larger than a.

max = 12

# Operadores Logicos

| Operator | Description | Expression | Corresponding Function |
| --- | --- | --- | --- |
| || | true if either of the Boolean expression is true | (a>b)||(a<c) | (a>b)or(a<c) |
| && | true if all Boolean expressions are true | (a>b)&&(a<c) | (a>b)and(a<c) |

fun main(args: Array<String>) {

val a = 10

val b = 9

val c = -1

val result: Boolean

// result is true is a is largest

result = (a>b) && (a>c) // result = (a>b) and (a>c)

println(result)

}

saida

true

# Referencias

<https://code.tutsplus.com/pt/tutorials/kotlin-from-scratch-variables-basic-types-arrays-type-inference-and-comments--cms-29328>