Kotlin Cheatsheet

devhints.io

December 20, 2022

Abstract

Research paper on quantum gravity; one potential theory of everything

Contents

Kotlin cheatsheet

Kotlin is a statically typed programming language for modern multiplatform applications.

Mutability

```
var mutableString: String = "Adam"
val immutableString: String = "Adam"
val inferredString = "Adam"
```

Strings

```
val name = "Adam"
val greeting = "Hello, " + name
val greetingTemplate = "Hello, $name"
val interpolated = "Hello, ${name.toUpperCase()}"
```

Numbers

```
val intNum = 10
val doubleNum = 10.0
val longNum = 10L
val floatNum = 10.0F
```

Booleans

```
val trueBoolean = true
val falseBoolean = false
val andCondition = trueBoolean && falseBoolean
val orCondition = trueBoolean || falseBoolean
```

Static Fields

```
class Person {
    companion object {
        val NAME_KEY = "name_key"
    }
}

val key = Person.NAME_KEY
```

Null Safety

Nullable properties

```
val cannotBeNull: String = null // Invalid
val canBeNull: String? = null // Valid

val cannotBeNull: Int = null // Invalid
val canBeNull: Int? = null // Valid
```

```
val name: String? = "Adam"

if (name != null && name.length > 0) {
    print("String length is ${name.length}")
} else {
    print("String is empty.")
}
```

Checking for null

Safe Operator

```
val nullableStringLength: Int? = nullableString?.length
val nullableDepartmentHead: String? = person?.department?.head?.name
```

Elvis Operator

```
val nonNullStringLength: Int = nullableString?.length ?: 0
val nonNullDepartmentHead: String = person?.department?.head?.name ?: ""
val nonNullDepartmentHead: String = person?.department?.head?.name.orEmpty()
```

Safe Casts

```
// Will not throw ClassCastException
val nullableCar: Car? = (input as? Car)
```

Collections

Creation

```
val numArray = arrayOf(1, 2, 3)
val numList = listOf(1, 2, 3)
val mutableNumList = mutableListOf(1, 2, 3)
```

Accessing

```
val firstItem = numList[0]
val firstItem = numList.first()
val firstItem = numList.firstOrNull()
```

Maps

```
val faceCards = mutableMapOf("Jack" to 11, "Queen" to 12, "King" to 13)
val jackValue = faceCards["Jack"] // 11
faceCards["Ace"] = 1
```

Mutability

```
val immutableList = listOf(1, 2, 3)
val mutableList = immutableList.toMutableList()
```

```
val immutableMap = mapOf("Jack" to 11, "Queen" to 12, "King" to 13)
val mutableMap = immutableMap.toMutableMap()
```

Iterating

```
for (item in myList) {
    print(item)
}

myList.forEach {
    print(it)
}

myList.forEachIndexed { index, item ->
    print("Item at $index is: $item")
}
```

Filtering & Searching

```
val evenNumbers = numList.filter { it % 2 == 0 }
val containsEven = numList.any { it % 2 == 0 }
val containsNoEvens = numList.none { it % 2 == 0 }
val containsNoEvens = numList.all { it % 2 == 1 }
val firstEvenNumber: Int = numList.first { it % 2 == 0 }
val firstEvenOrNull: Int? = numList.firstOrNull { it % 2 == 0 }
val fullMenu = objList.map { "${it.name} - $${it.detail}" }
```

Note: it is the implicit name for a single parameter. ## Functions ### Parameters & Return Types

```
fun printName() {
    print("Adam")
}

fun printName(person: Person) {
    print(person.name)
}

fun getGreeting(person: Person): String {
    return "Hello, ${person.name}"
}

fun getGreeting(person: Person): String = "Hello, ${person.name}"
fun getGreeting(person: Person) = "Hello, ${person.name}"
```

Higher Order Functions

```
fun callbackIfTrue(condition: Boolean, callback: () -> Unit) {
    if (condition) {
        callback()
    }
}
```

```
callbackIfTrue(someBoolean) {
    print("Condition was true")
}
```

Extension Functions

```
fun Int.timesTwo(): Int {
    return this * 2
}

val four = 2.timesTwo()
```

Default Parameters

```
fun getGreeting(person: Person, intro: String = "Hello,") {
    return "$intro ${person.name}"
}

// Returns "Hello, Adam"
val hello = getGreeting(Person("Adam"))

// Returns "Welcome, Adam"
val welcome = getGreeting(Person("Adam"), "Welcome,")
```

Named Parameters

```
class Person(val name: String = "", age: Int = 0)

// All valid
val person = Person()
val person = Person("Adam", 100)
val person = Person(name = "Adam", age = 100)
val person = Person(age = 100)
val person = Person(age = 100, name = "Adam")
```

Static Functions

```
class Fragment(val args: Bundle) {
    companion object {
        fun newInstance(args: Bundle): Fragment {
            return Fragment(args)
        }
    }
}

val fragment = Fragment.newInstance(args)

    Companion Objects
```

Classes

Primary Constructor

```
class Person(val name: String, val age: Int)
val adam = Person("Adam", 100)
```

Secondary Constructors

```
class Person(val name: String) {
    private var age: Int? = null

    constructor(name: String, age: Int) : this(name) {
        this.age = age
    }
}


// Above can be replaced with default params
class Person(val name: String, val age: Int? = null)
```

Inheritance & Implementation

```
open class Vehicle
class Car : Vehicle()

interface Runner {
    fun run()
}

class Machine : Runner {
    override fun run() {
        // ...
    }
}
```

Control Flow

If Statements

```
if (someBoolean) {
    doThing()
} else {
    doOtherThing()
}
```

For Loops

```
for (i in 0..10) { } // 1 - 10
for (i in 0 until 10) // 1 - 9
(0..10).forEach { }
for (i in 0 until 10 step 2) // 0, 2, 4, 6, 8
```

When Statements

```
when (direction) {
    NORTH -> {
        print("North")
    }
    SOUTH -> print("South")
    EAST, WEST -> print("East or West")
    "N/A" -> print("Unavailable")
    else -> print("Invalid Direction")
}
```

While Loops

```
while (x > 0) {
    x--
}

do {
    x--
} while (x > 0)
```

Destructuring Declarations

Objects & Lists

```
val person = Person("Adam", 100)
val (name, age) = person

val pair = Pair(1, 2)
val (first, second) = pair

val coordinates = arrayOf(1, 2, 3)
val (x, y, z) = coordinates
```

ComponentN Functions

```
class Person(val name: String, val age: Int) {
    operator fun component1(): String {
        return name
    }

    operator fun component2(): Int {
        return age
    }
}
```