# Kotlin cheat sheet

Romain Rochegude

# Classes

From Java

```
public class JavaCode
```

to Kotlin

```
class JavaCode
```

with primary constructor

```
class Person(val name: String, var age: Int)
                // name is readonly, age is mutable
```

# Inheritence

```
open class Person(val name: String) {
    open fun sayHello() = "Hello $name"
}

class FrenchPerson(name: String) : Person(name) {
    override fun sayHello() = "Bonjour $name"
}
```

# Properties with accessors

```
class Person(val name: String, var age: Int) {
    var birthYear: Int
        get() = LocalDate.now().year - age
        set(value) {
            age = LocalDate.now().year - value
        }
}
```

# Lazy properties

```
val str: String by lazy {
      // Compute the string
}
```

# Data class

## Autogenerated `equals()`, `hashCode()`, `toString()`, `copy()`

```
data class Person(val name: String, var age: Int)

val olderPerson = person.copy(age = person.age + 1)
```

# Methods

From Java

```
public String toJSON(Collection<Integer> collection)
```

to Kotlin

```
fun toJSON(Collection:Collection<Int>):String
```

```
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

or simply

```
fun sum(a: Int, b: Int) = a + b
```

# Variables

From Java

```
StringBuilder sb = new StringBuilder();
```

to Kotlin

```
val sb: StringBuilder = StringBuilder()

// or

val sb = StringBuilder() // type is inferred
```

# Null-safety

```
val x: Int // Error - variable must be initialized

val x: Int = null // Error - null cannot be a value for non-null type Int

val x: Int? = null // OK

val y: String = null
// Error - null cannot be a value for non-null type String

val y: String? = null // OK
```

```
var a : String = "abc"
a = null // compilation error

var b: String? = "abc"
b = null // ok

b?.length
// returns b.length if b is not null, and null otherwise.
// The type of this expression is Int?
```

```
val name: String? = null

println(name.length)
// Error - Only safe (?.) or non-null asserted (!!.)
// calls are allowed on a nullable receiver of type String?

println(name?.length) // prints "null"

name = "Treehouse"
println(name!!.length) // prints "9"
```

# The Elvis operator

```
val l: Int = if (b != null) b.length else -1
```

becomes

```
val l = b?.length ?: -1
```

# String formats

```
val releaseDate = "July 2011"
val releaseString = "Kotlin was released in $releaseDate"
// releaseString = "Kotlin was released in July 2011"
```

# Numbers

```
val num1 = 42 // Int
val num2 = 3.14 // Double
val num3 = 42L // Long
val num4 = 3.14f // Float
val num5 = 42.toFloat() // Float
val num6 = num1.toDouble() // Double
```

# Collections

```
val cardNames = arrayOf("Jack", "Queen", "King")

val cards = mutableListOf("Jack", "Queen", "King")
for ((index, card) in cards.withIndex()) {
    // String interpolation: $index, $card
    println("Card at $index is $card")
}

val cards = mapOf("Jack" to 11, "Queen" to 12, "King" to 13)
for ((name, value) in cards) {
    println("$name, $value")
}
```

# The `when` keyword

```
fun whenDemo(x: Number) = when(x) {
    0 -> "Zero"                     // Equality check
    in 1..4 -> "Four or less"       // Range check
    5, 6, 7 -> "Five to seven"      // Multiple values
    is Byte -> "Byte"               // Type check
    else -> "Some number"
}
```

# The `let` keyword

```
val listWithNulls: List<String?> = listOf("A", null)
for (item in listWithNulls) {
    if (item != null) {
        println(item!!)
    }
}
```

becomes

```
val listWithNulls: List<String?> = listOf("A", null)
for (item in listWithNulls) {
    item?.let { println(it) } // prints A and ignores null
}
```

# The `it` keyword

```
val positives = list.filter { x -> x > 0 }
```

becomes

```
val positives = list.filter { it > 0 }
```

# Gradle configuration

```
buildscript {
    // ...
    ext.kotlin_version = '<version to use>'

    dependencies {
        classpath "org.jetbrains.kotlin" +
            "kotlin-gradle-plugin:$kotlin_version"
    }
}

apply plugin: 'kotlin'

dependencies {
    compile "org.jetbrains.kotlin:kotlin-stdlib-jre8"
}
```

# Other concepts

- class extensions
- function extensions
- infix notations
- operator overload
- property delegation
- sealed class
- `with` keyword

# Testing

- http://spekframework.org/docs/latest/
- https://markusamshove.github.io/Kluent/

# Bibliography


- https://www.infoworld.com/article/3224868/java/what-is-kotlin-the-java-alternative-explained.amp.html
- http://blog.teamtreehouse.com/absolute-beginners-guide-kotlin
- https://jaxenter.com/kotlin-cheat-sheet-tips-tricks-136716.html
- https://medium.com/default-to-open/kotlin-tips-singleton-utility-functions-group-object-initialization-and-more-27cdd6f63a41

# Resources

- https://blog.jetbrains.com/kotlin/
- https://antonioleiva.com/kotlin/
- https://antonioleiva.com/kotlin-android-developers-book/
- https://medium.com/tag/kotlin
- https://github.com/mcxiaoke/awesome-kotlin
- https://kotlinlang.org/docs/resources.html
- https://github.com/KotlinBy/awesome-kotlin
- https://kotlin.link/