Kotlin
КОТЛИН

WIZELINE®

Contents

Introduction

- **Open Source (Apache 2)**

- **Started 2011; Version 1.0 2016; Currently 1.1.3-2**

- **Static; Inferred; Script; Functional; OOP**

- **Platform: JVM; JS; LLVM**

- **Concise; Safe; Interoperable; Tool-friendly**

- **Official Android Language**

How to start

- **Android Studio 2 Go to:**
  - Preferences
  - Plugins
  - Click on Install JetBrains plugin
  - Install Kotlin Language
  - Restart

- **Android Studio 3 ready out of the box.**

build.gradle

```
buildscript {
    repositories { jcenter() }
    dependencies {
        classpath "com.android.tools.build:gradle:2.3.3"
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:x.y.z"
    }
}
```

app/build.gradle

```gradle
apply plugin: "com.android.application"
apply plugin: "kotlin-android"

// If you're using annotation processor
apply plugin: "kotlin-kapt"

// A pro tip
apply plugin: "kotlin-android-extensions"
```
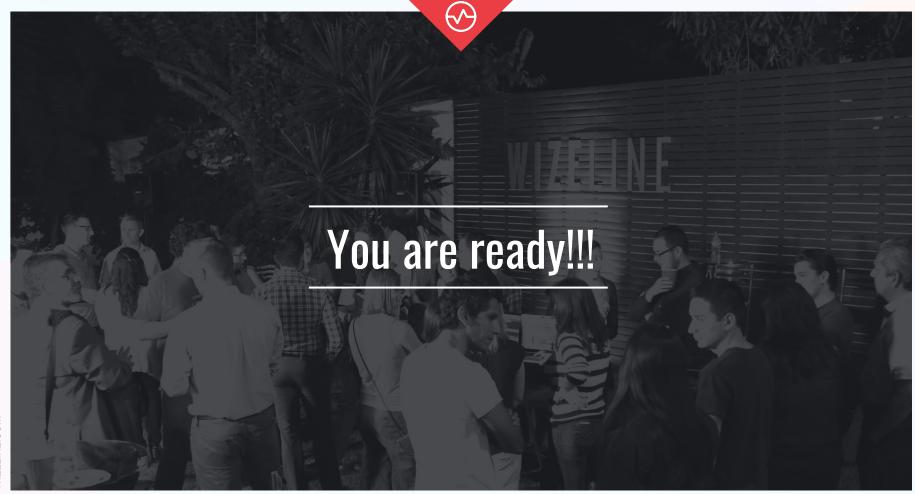
app/build.gradle

```
dependencies {

    // Change any annotationProcessor or apt to kapt

    compile "org.jetbrains.kotlin:kotlin-stdlib:x.y.z"

}
```

You are ready!!!

WIZELINE.COM

Basics

Package & Import

Sample.kt

```kotlin
package com.wizeline.academy

import java.io.File
import java.util.*
import android.view.View.GONE
import android.support.v4.content.ContextCompat.getColor
import android.support.v7.app.AppCompatActivity as Activity
```

Comments

Sample.kt

```kotlin
// This is an end-of-line comment

/* This is a block comment
   on multiple lines. */

/* This is /* a nested comment
   on multiple */ lines. */
```

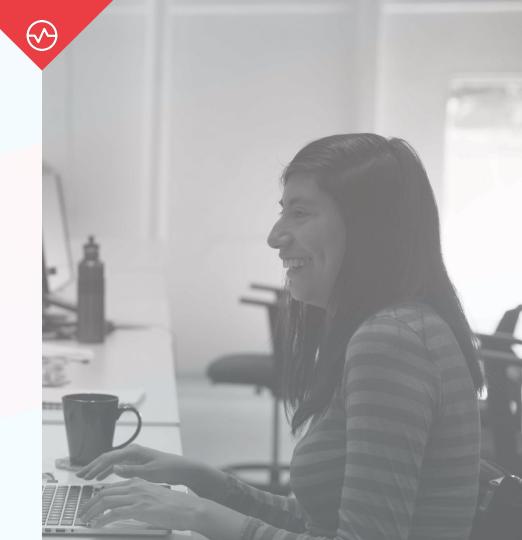# Variables

Val *value* :

immutable reference

Var *variable* :

mutable reference

Sample.kt

```kotlin
val message: String = "Hello World!"
```

Variables

Sample.kt

```kotlin
val message: String = "Hello World!"
message = "Bye World!"
```

Variables

```kotlin
val message: String = "Hello World!"
message = "Bye World!" // Compile time error, value cannot be assigned
```

Sample.kt

```kotlin
var message: String = "Hello World!"
```

Sample.kt

```kotlin
var message: String = "Hello World!"
message = "Bye World!"
```

Variables

Sample.kt

```kotlin
var message: String = "Hello World!"
message = "Bye World!" // It is ok
```

Sample.kt

```kotlin
val message: String = "Hello World!"
```

Sample.kt

```kotlin
val message = "Hello World!"
```

Sample.kt

```kotlin
var message: String = "Hello World!"
```

Sample.kt

```kotlin
var message: String = "Hello World!"
message = null
```

Sample.kt

```kotlin
var message: String = "Hello World!"
message = null // Error: null can not be a value of a non-null type
```

Sample.kt

```kotlin
var message: String? = "Hello World!"
```

Sample.kt

```kotlin
var message: String? = "Hello World!"
message = null
```

Sample.kt

```kotlin
var message: String? = "Hello World!"
message = null // It is ok
```

Sample.kt

```kotlin
var message: String? = null
```

Sample.kt

```kotlin
var message: String? = null
message.length
```

Sample.kt

```kotlin
var message: String? = null
message.length // Error: Only safe or non-null asserted calls are
               // allowed on a nullable receiver of type String?
```

Sample.kt

```kotlin
var message: String? = null
message?.length
```

Sample.kt

```kotlin
var message: String? = null
message?.length // It is ok
```

Sample.kt

```kotlin
var message: String? = null
message!!.length
```

Sample.kt

```kotlin
var message: String? = null
message!!.length // Possible but don't do it unless you know what you're doing
```

Sample.kt

```kotlin
var message: String? = null
if (message != null) {
    message.length // It is ok. Spoiler: if syntax
}
```

Strings

Sample.kt

```kotlin
val message = "Hello World!"
```
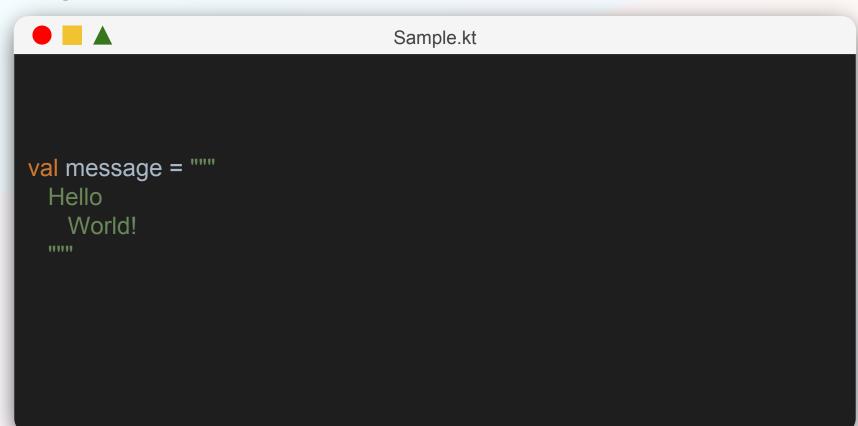
Sample.kt

```kotlin
val message = """
    Hello
        World!
    """
```

Sample.kt

```kotlin
val message = """
   |Hello
   |   World!
   """.trimMargin()
```

Sample.kt

```kotlin
val message = "Hello World!"
println("$message length is: ${message.length}")
```

Control Flow

IF - ELSE

Sample.kt

```kotlin
val a = random()
val b = random()
if (a > b) {
    println("A is greater than b.")
} else if (a == b) {
    println("A is equal to b.")
} else {
    println("A is less than b.")
}
```

For

Sample.kt

```kotlin
val message = "Hello World!"
for (character in message) {
    println(character)
}
// Prints each letter
```

Sample.kt

```kotlin
for (number in 0..10) {
    println(number)
}
// Prints 0, 1, ..., 10
```

Sample.kt

```kotlin
for (number in 0 until 10) {
    println(number)
}
// Prints 0, 1, ..., 9
```

Sample.kt

```kotlin
for (number in 0..10 step 3) {
    println(number)
}
// Prints 0, 3, 6, 9
```

Sample.kt

```kotlin
for (number in 10 downTo 0) {
    println(number)
}
// Prints 10, 9, ..., 0
```

Sample.kt

```kotlin
val array: Array<String> = arrayOf<String>("Hello", "World!")
// Spoiler: How to create an array
```

Sample.kt

```kotlin
val array = arrayOf("Hello", "World!")
// Spoiler: How to create an array
```

Sample.kt

```kotlin
val array = arrayOf("Hello", "World!")
for (item in array) {
    println(item)
}
// Prints Hello and World!
```

Sample.kt

```kotlin
val array = arrayOf("Hello", "World!")
for ((index, item) in array.withIndex()) {
  println("$index $item")
}
// Prints 0 Hello and 1 World!
```

Sample.kt

```kotlin
val clumpsOfSugar = 0
when (clumpsOfSugar) {
  0 -> println("The right way to drink coffee")
  1 -> println("It is a sin, but I can ignore")
  else -> {
    println("Wait, $clumpsOfSugar clumps of sugar!?")
    println("It is an unforgivable sin!!!")
  }
}
```

Sample.kt

```kotlin
val clumpsOfSugar = 0
when (clumpsOfSugar) {
    0 -> println("The right way to drink coffee")
    1, 2 -> println("It is a sin, but I can ignore")
    else -> {
        println("Wait, $clumpsOfSugar clumps of sugar!?")
        println("It is an unforgivable sin!!!")
    }
}
```

Sample.kt

```kotlin
val clumpsOfSugar = 0
when (clumpsOfSugar) {
    0 -> println("The right way to drink coffee")
    1, 2 -> println("It is a sin, but I can ignore")
    in 3..5 -> println("Is it a kind of candy???")
    else -> {
        println("Wait, $clumpsOfSugar clumps of sugar!?")
        println("It is an unforgivable sin!!!")
    }
}
```

Sample.kt

```kotlin
val clumpsOfSugar = 0
when (clumpsOfSugar) {
    0 -> println("The right way to drink coffee")
    1, 2 -> println("It is a sin, but I can ignore")
    !in 3..5 -> println("It is not 3, 4 or 5 :)")
    else -> {
        println("Wait, $clumpsOfSugar clumps of sugar!?")
        println("It is an unforgivable sin!!!")
    }
}
```

Sample.kt

```kotlin
val clumpsOfSugar = 0
val candyNumbers = arrayOf(3, 4, 5)
when (clumpsOfSugar) {
    0 -> println("The right way to drink coffee")
    1, 2 -> println("It is a sin, but I can ignore")
    in candyNumbers -> println("Is it a kind of candy???")
    else -> {
        println("Wait, $clumpsOfSugar clumps of sugar!?")
        println("It is an unforgivable sin!!!")
    }
}
```

Sample.kt

```kotlin
val clumpsOfSugar = 0
when (clumpsOfSugar) {
    0 -> println("The right way to drink coffee")
    1, 2 -> println("It is a sin, but I can ignore")
    randomInt() -> println("Are we still talking about coffee?")
    else -> {
        println("Wait, $clumpsOfSugar clumps of sugar!?")
        println("It is an unforgivable sin!!!")
    }
}
```

Sample.kt

```kotlin
val clumpsOfSugar = 0
when {
    clumpsOfSugar % 2 == 1 -> println("An odd amount of sugar clumps")
    clumpsOfSugar % 2 == 0 -> println("An even amount of sugar clumps")
    else -> println("We have a really strange amount of sugar clumps")
}
```

Sample.kt

```kotlin
val clumpsOfSugar = 0
when {
    clumpsOfSugar.isOdd() -> println("An odd amount of sugar clumps")
    clumpsOfSugar.isEven() -> println("An even amount of sugar clumps")
    else -> println("We have a really strange amount of sugar clumps")
}
```

Sample.kt

```kotlin
var clumpsOfSugar = 10
while (clumpsOfSugar > 0) {
   clumpsOfSugar-- // yes we have -- and ++ syntax
}
println(clumpsOfSugar) // Prints 0
```
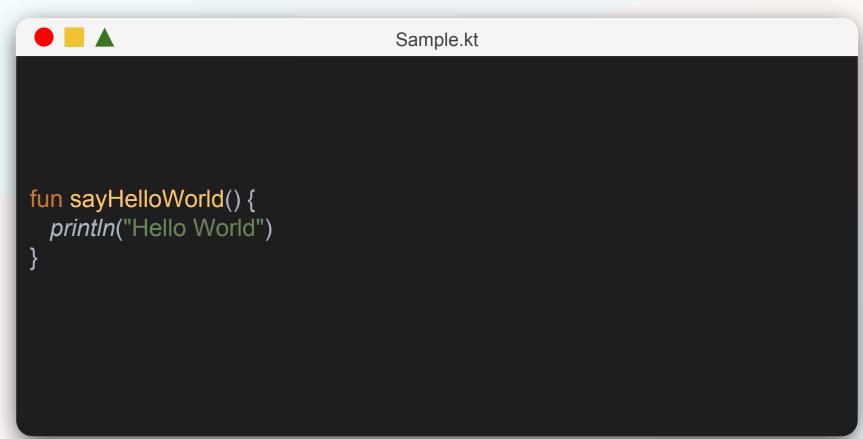
Sample.kt
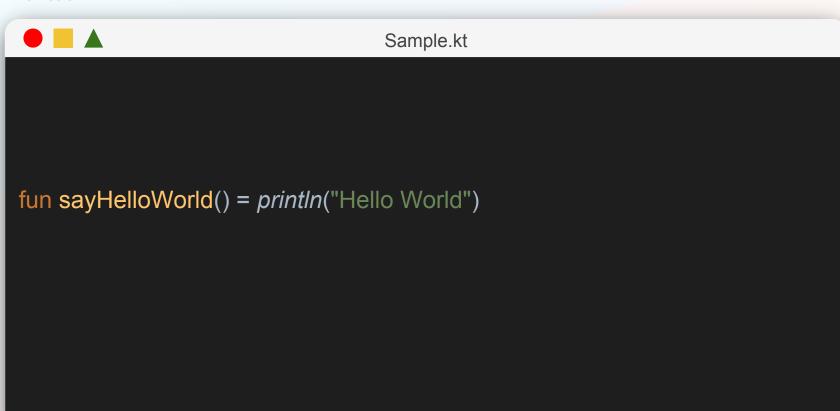
```kotlin
var clumpsOfSugar = 10
do {
    clumpsOfSugar--
} while (clumpsOfSugar > 0)
println(clumpsOfSugar) // Prints 0
```
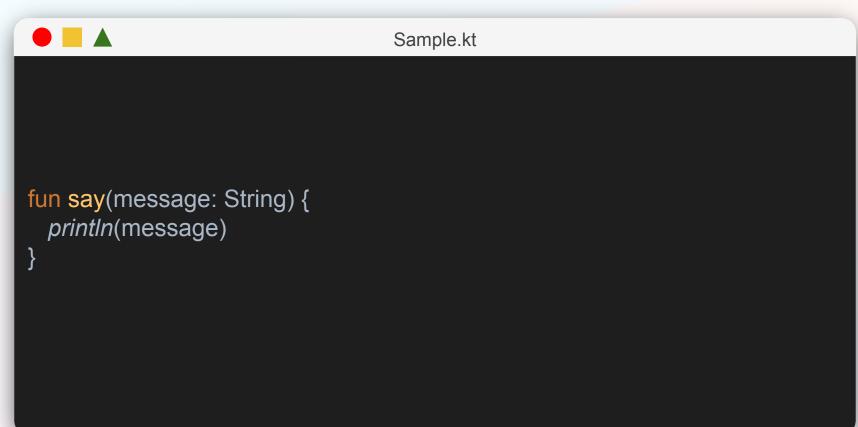
Function
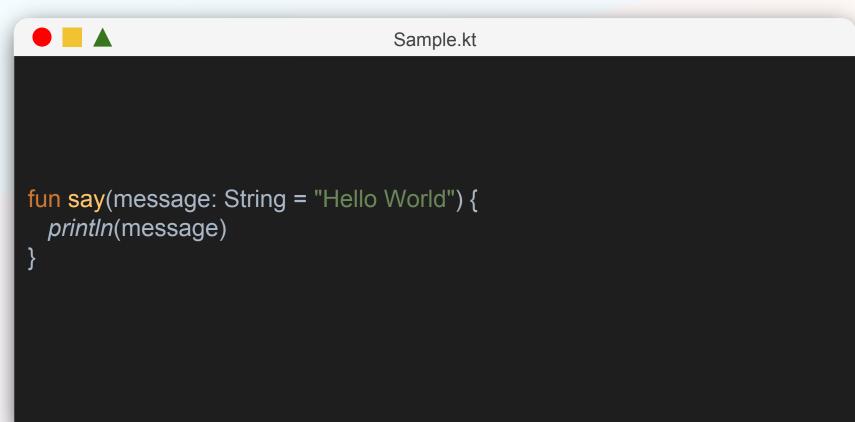
Function

Sample.kt

```kotlin
fun sayHelloWorld() {
    println("Hello World")
}
```

Sample.kt

```kotlin
fun sayHelloWorld() = println("Hello World")
```

Function

```kotlin
fun say(message: String) {
    println(message)
}
```

Sample.kt

```kotlin
fun say(message: String = "Hello World") {
    println(message)
}
```

Function

Sample.kt

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

Sample.kt

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

Function

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

Sample.kt

```kotlin
fun makeCoffee(ml: Int, concentration: Float, useSugar: Boolean = false)

makeCoffee(200, 0.8f)
makeCoffee(200, 0.8f, true)
makeCoffee(200, 0.8f, useSugar = true)
makeCoffee(ml = 200, concentration = 0.8f, useSugar = true)
makeCoffee(useSugar = true, ml = 200, concentration = 0.8f)
```

Function
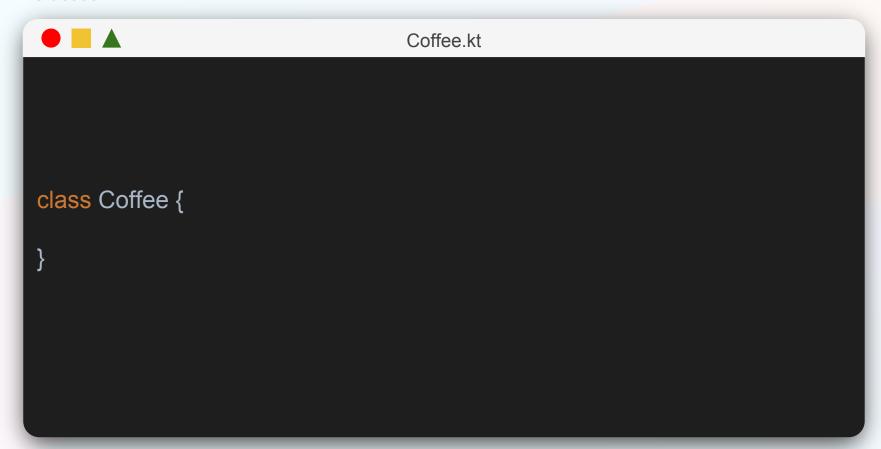
Sample.kt

```kotlin
fun outside() {
    fun inside() {
        // Do something
    }
    // ...
    inside() // we can call in this scope
    // ...
}
```

OOP

Classes

Coffee.kt

```kotlin
class Coffee {

}
```

Coffee.kt

```kotlin
class Coffee
```

Coffee.kt

```kotlin
class Coffee

val coffee = Coffee()
```

Constructor

Coffee.kt

```kotlin
class Coffee(type: String)
```

Coffee.kt

```kotlin
class Coffee @Inject constructor(type: String)
```
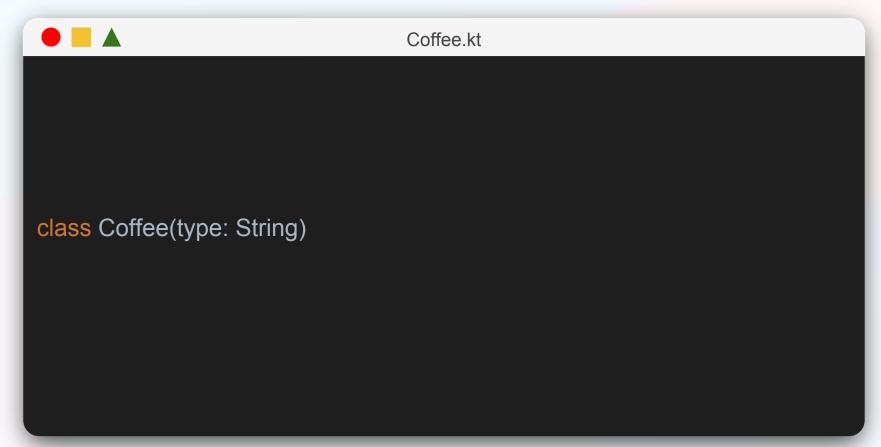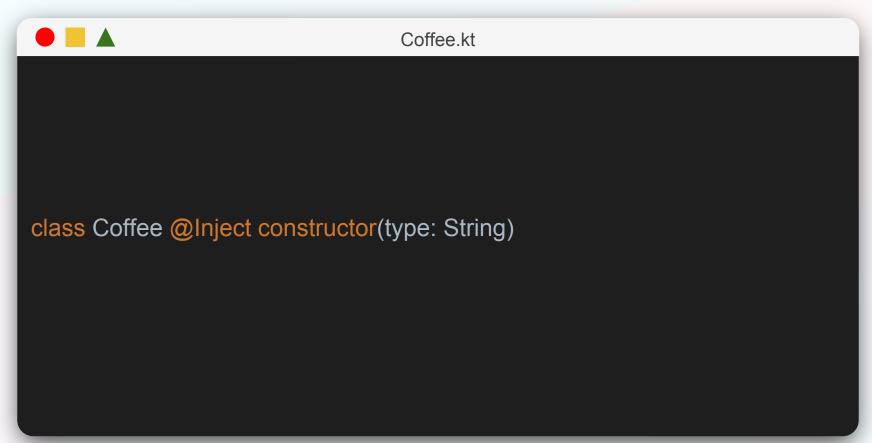
CustomView.kt

```kotlin
class CustomView : View { // Spoiler: Inheritance

    constructor(c: Context) : super(c) {}
    constructor(c: Context, a: AttributeSet?) : super(c, a) {}
    constructor(c: Context, a: AttributeSet?, d: Int) : super(c, a, d) {}
    constructor(c: Context, a: AttributeSet?, d: Int, r: Int) : super(c, a, d, r) {}

}
```
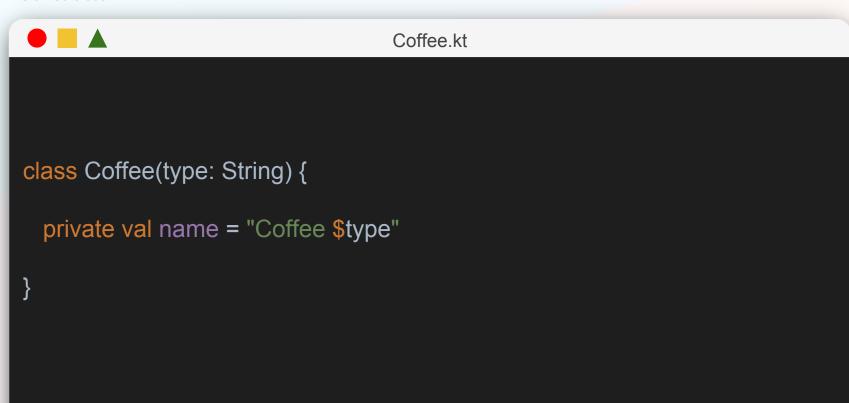
Coffee.kt

```kotlin
class Coffee(type: String)
class Coffee(var type: String)
class Coffee(val type: String)
class Coffee(private val type: String)
class Coffee(private val type: String = "Espresso")
```

Coffee.kt

```kotlin
class Coffee(type: String) {

    private val name = "Coffee $type"

}
```

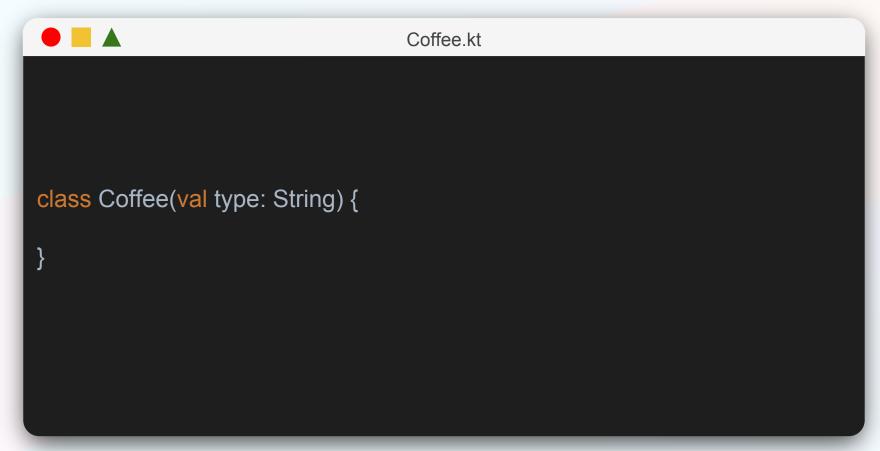Coffee.kt

```kotlin
class Coffee(type: String) {

    private val name: String

    init {
        name = "Coffee $type"
    }

}
```

Coffee.kt

```kotlin
class Coffee {

    private val name: String

    constructor(type: String) {
        name = "Coffee $type"
    }

}
```

Coffee.kt

```kotlin
class Coffee(val type: String) {

}
```

Properties

Coffee.kt

```kotlin
class Coffee {

    val type = "Espresso"
    var drinked = false
    private val gourmet = true

}
```

Properties

Coffee.kt

```kotlin
class Coffee {

    var type = "Espresso"
        get() {
            return field
        }
        set(value) {
            field = value
        }

}
```
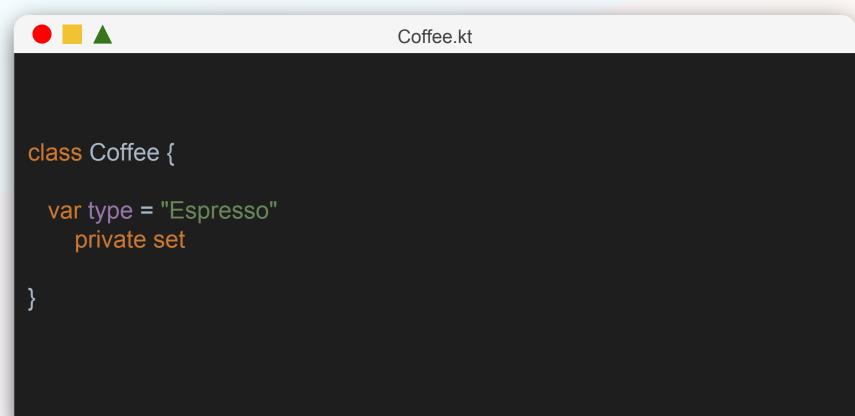
Coffee.kt

```kotlin
class Coffee(context: Context) {

    private val preferences = context.getSharedPreferences("Coffee", MODE_PRIVATE)

    var type
        get() = preferences.getString("TypeKey", "Default")
        set(value) {
            preferences.edit().putString("TypeKey", value).apply()
        }

}
```
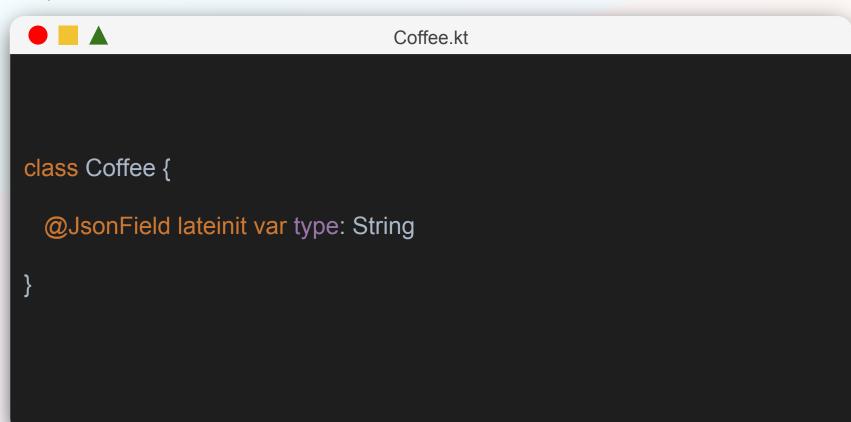
Coffee.kt

```kotlin
class Coffee {

    var type = "Espresso"
        private set

}
```

Coffee.kt

```kotlin
class Coffee {

    @JsonField lateinit var type: String

}
```

Coffee.kt

```kotlin
class Coffee {

    val type: String by lazy {
        println("You are running this code!")
        "Hello"
    }


    val x = type // It prints "You are running this code!" 1 time
    val y = type // It don't prints anything


}
```

Methods

Kotlin methods are functions, and you already know it.

Inheritance

Coffee.kt

```kotlin
open class Coffee

class Cappuccino : Coffee()
```

Coffee.kt

```kotlin
open class Coffee(type: String)

class Cappuccino : Coffee("Cappuccino")
```

Coffee.kt

```kotlin
open class Coffee(type: String)

class Cappuccino(type: String) : Coffee(type)
```

Coffee.kt

```kotlin
open class Coffee {
    constructor(type: String)
    constructor(type: String, temperature: Float)
}

class Cappuccino : Coffee {
    constructor(type: String) : super(type)
    constructor(type: String, temperature: Float) : super(type, temperature)
}
```

Coffee.kt

```kotlin
open class Coffee {
  open fun drink() {
    println("So tasty")
  }
}


class Cappuccino : Coffee() {
  override fun drink() {
    super.drink()
  }
}
```

Interfaces

Coffee.kt

```kotlin
interface Drinkable

class Coffee : Drinkable
```

Coffee.kt

```kotlin
interface Drinkable {
    fun drink()
}

class Coffee : Drinkable {
    override fun drink() {
        println("So tasty")
    }
}
```

Coffee.kt

```kotlin
interface Drinkable {
  fun drink() {
    println("So tasty")
  }
}


class Coffee : Drinkable
```

Coffee.kt

```kotlin
interface Drinkable {
    val type: String
}

class Coffee : Drinkable {
    override val type = "Coffee"
}
```

Coffee.kt

```kotlin
open class Coffee

interface Drinkable {
    fun drink()
}

class Cappuccino : Coffee(), Drinkable {
    override fun drink() {
        println("So tasty")
    }
}
```

Coffee.kt

```kotlin
open class Coffee {
    open fun drink() = println("So coffee")
}

interface Drinkable {
    fun drink() = println("So tasty")
}

class Cappuccino : Coffee(), Drinkable {

// What happens here ?


}
```
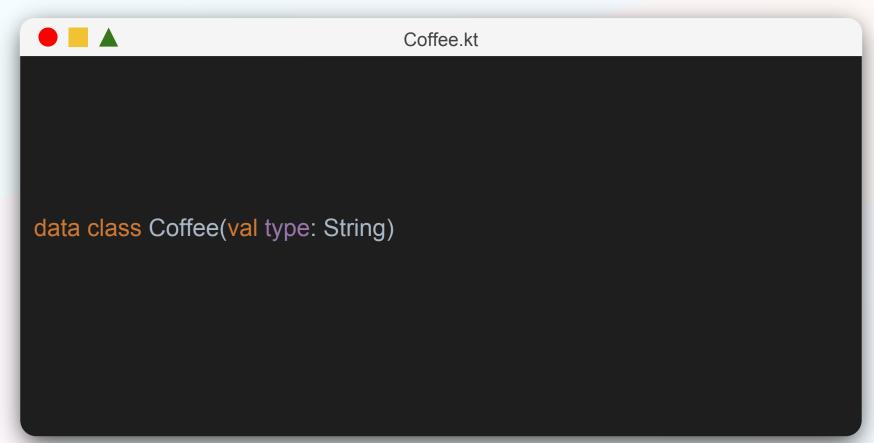
Coffee.kt

```kotlin
open class Coffee {
    open fun drink() = println("So coffee")
}

interface Drinkable {
    fun drink() = println("So tasty")
}

class Cappuccino : Coffee(), Drinkable {
    override fun drink() {
        super<Coffee>.drink()
        super<Drinkable>.drink()
    }
}
```

Data Classes

Coffee.kt

```kotlin
data class Coffee(val type: String)
```

Coffee.kt

```kotlin
data class Coffee(val type: String)

val a = Coffee("Cappuccino")
val b = Coffee("Cappuccino")

println(a == b) // Prints true
```

Coffee.kt

```kotlin
data class Coffee(val type: String)

val a = Coffee("Cappuccino")
val b = Coffee("Ristretto")

println(a == b) // Prints false
```

Coffee.kt

```kotlin
data class Coffee(val type: String)

val a = Coffee("Cappuccino")

println(a.toString()) // Prints: Coffee(type=Cappuccino)
```

Coffee.kt

```kotlin
data class Coffee(val type: String)

val a = Coffee("Cappuccino")
val b = a.copy()
```

Coffee.kt

```kotlin
data class Coffee(val type: String)

val a = Coffee("Cappuccino")
val b = a.copy()
val c = a.copy(type = "Ristretto")
```

Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

THANK YOU

WIZELINE®

ademar.oliveira@wizeline.com