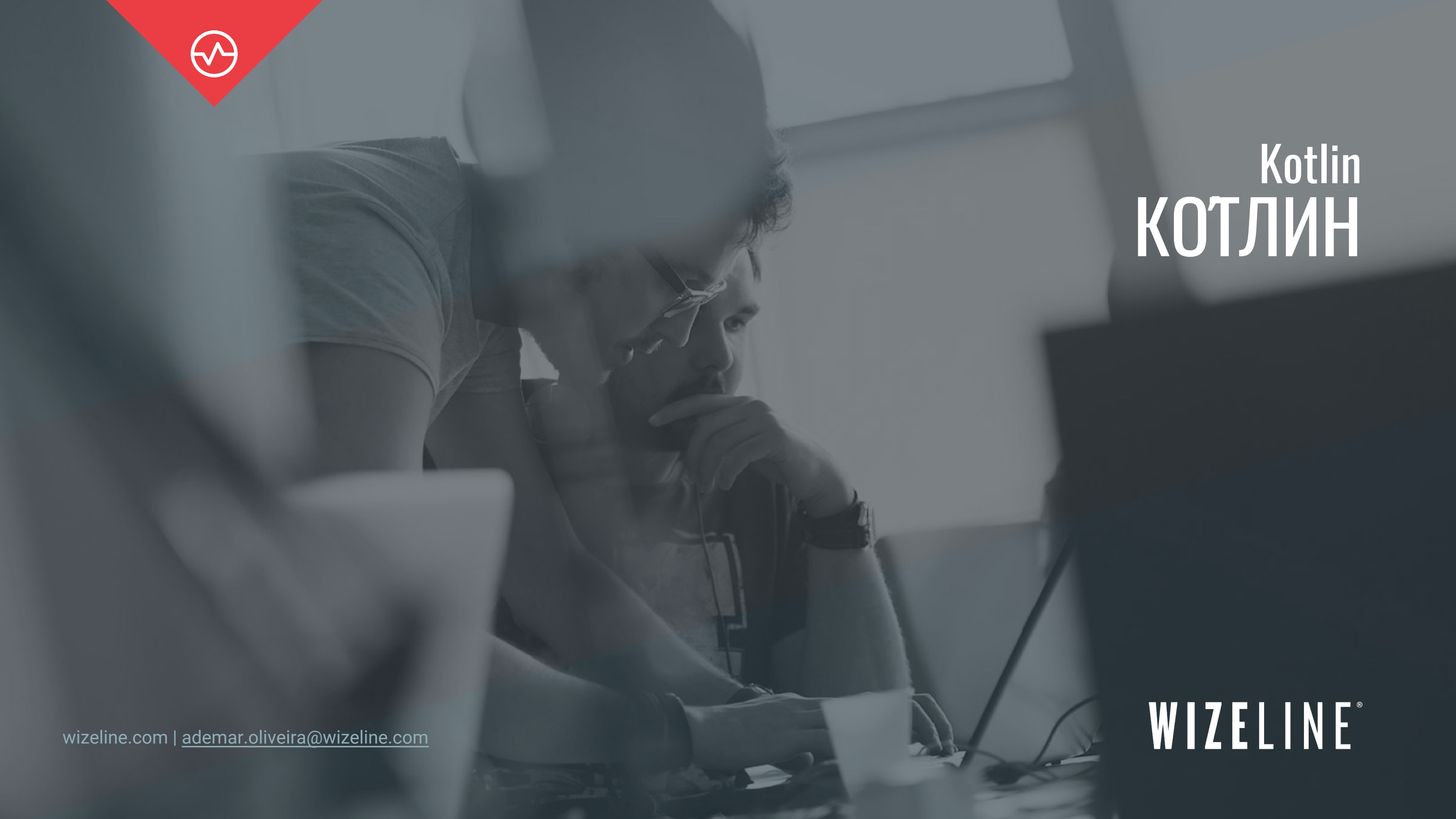


Kotlin

КО́ТЛИН

WIZELINE®

wizeline.com | ademar.oliveira@wizeline.com

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

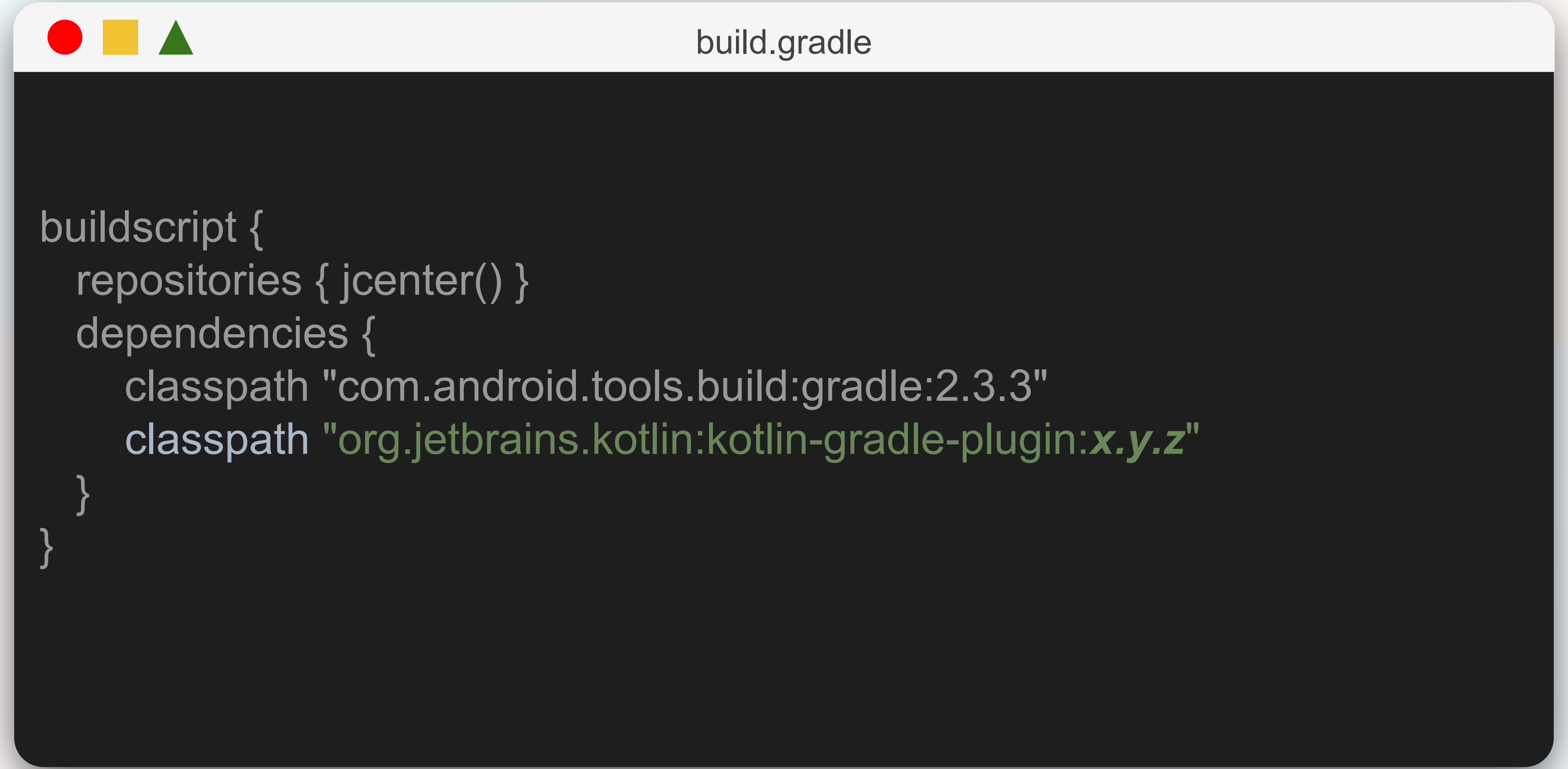
- **Android Studio 2 Go to:**
  - Preferences
  - Plugins
  - Click on Install JetBrains plugin
  - Install Kotlin Language
  - Restart
- **Android Studio 3 ready out of the box.**

How to start

build.gradle

```
buildscript {
   repositories { jcenter() }
   dependencies {
       classpath "com.android.tools.build:gradle:2.3.3"
   }
}
```

How to start



build.gradle

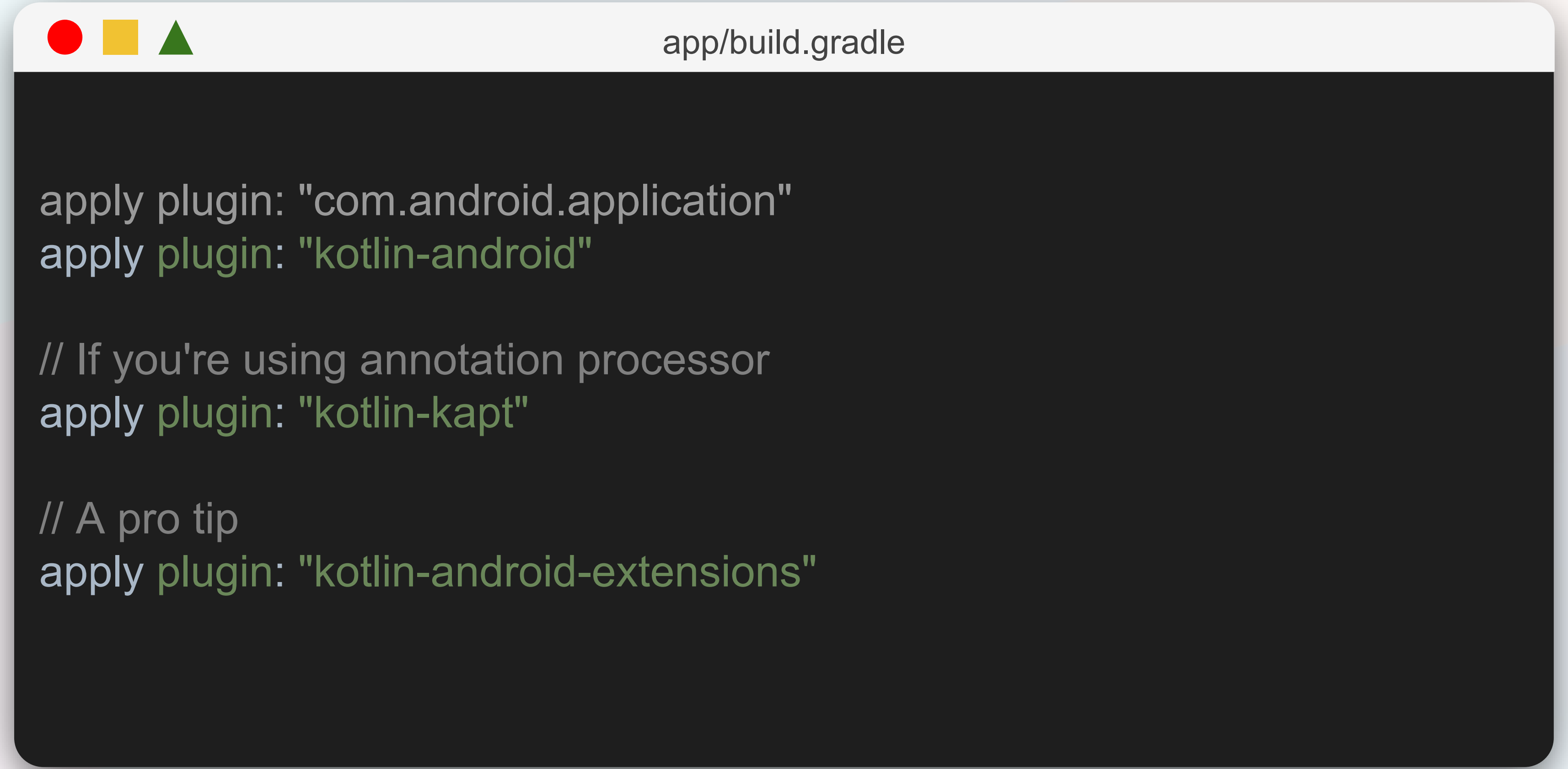
```
buildscript {
   repositories { jcenter() }
   dependencies {
       classpath "com.android.tools.build:gradle:2.3.3"
       classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:x.y.z"
   }
}
```

How to start

app/build.gradle

```
apply plugin: "com.android.application"
```

How to start



app/build.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"
```

How to start

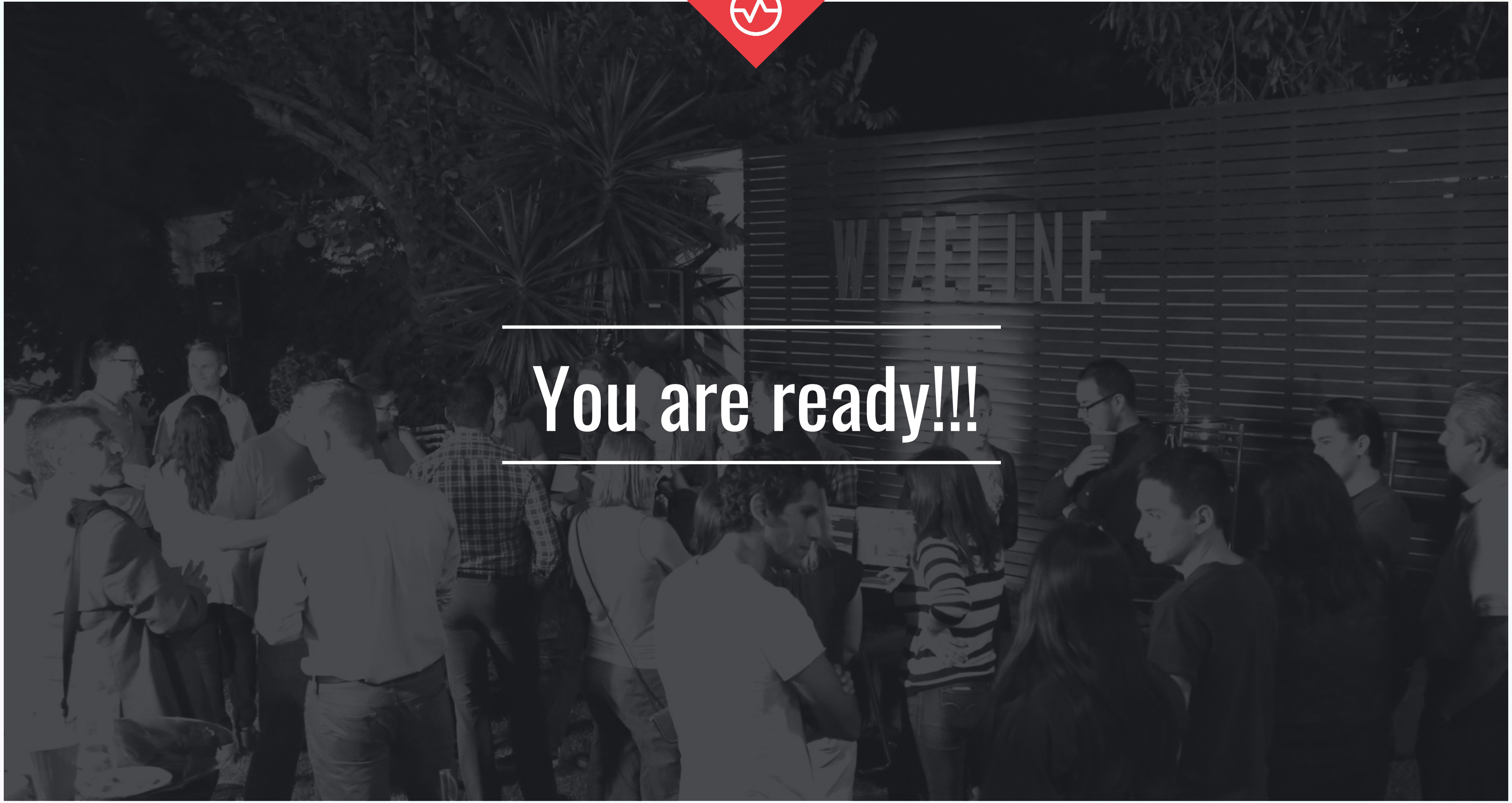
app/build.gradle

```
dependencies {
   annotationProcessor "com.google.dagger:dagger-compiler:2.11"

   compile "com.android.support:appcompat-v7:25.3.1"
   compile "com.android.support:design:25.3.1"
   compile "com.google.dagger:dagger-android:2.11"

   testCompile "junit:junit:4.12"
}
```

How to start

app/build.gradle

```
dependencies {
   kapt "com.google.dagger:dagger-compiler:2.11"

   compile "com.android.support:appcompat-v7:25.3.1"
   compile "com.android.support:design:25.3.1"
   compile "com.google.dagger:dagger-android:2.11"
   compile "org.jetbrains.kotlin:kotlin-stdlib:x.y.z"

   testCompile "junit:junit:4.12"
}
```



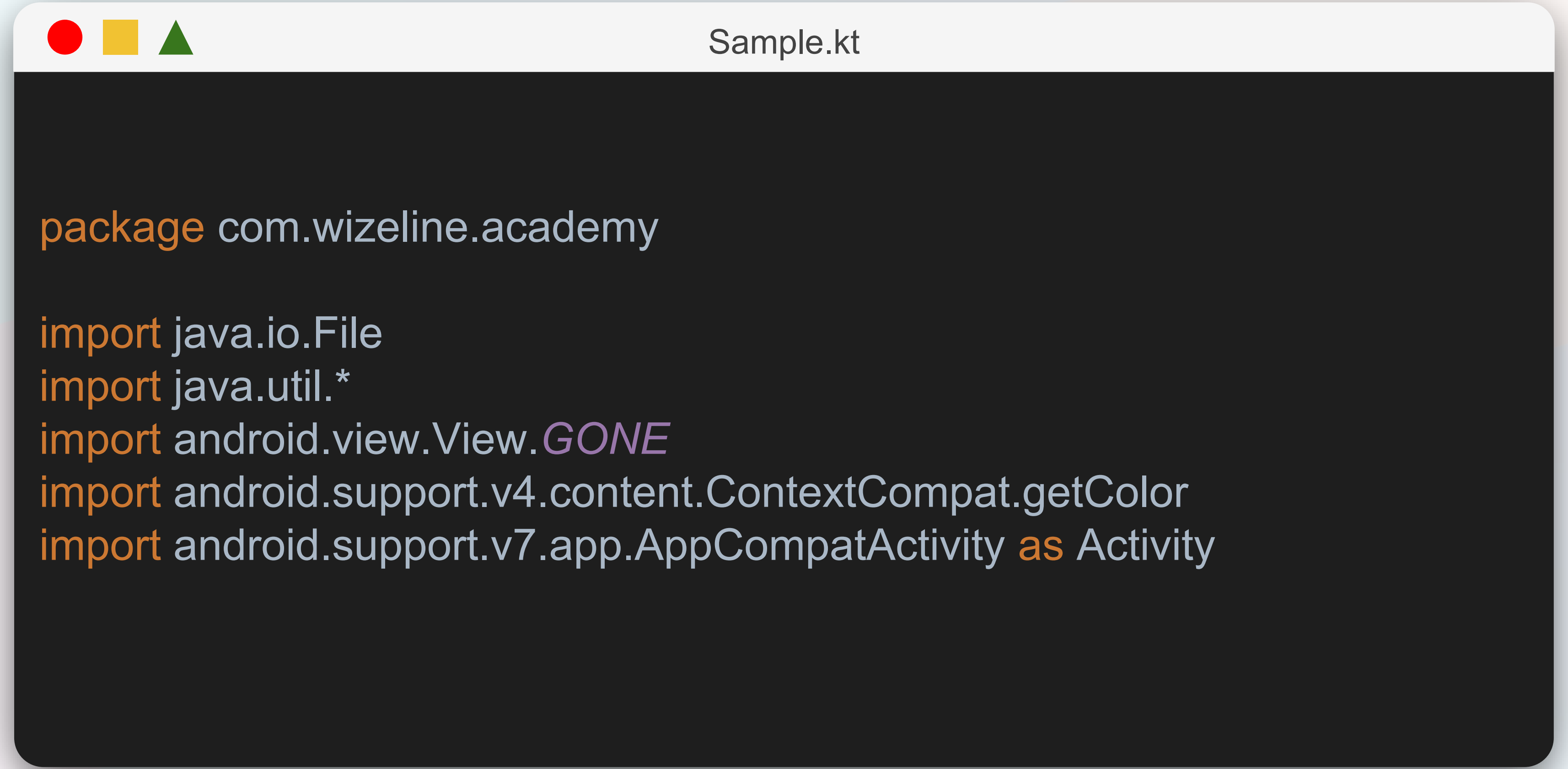
You are ready!!!

Basics

Basics

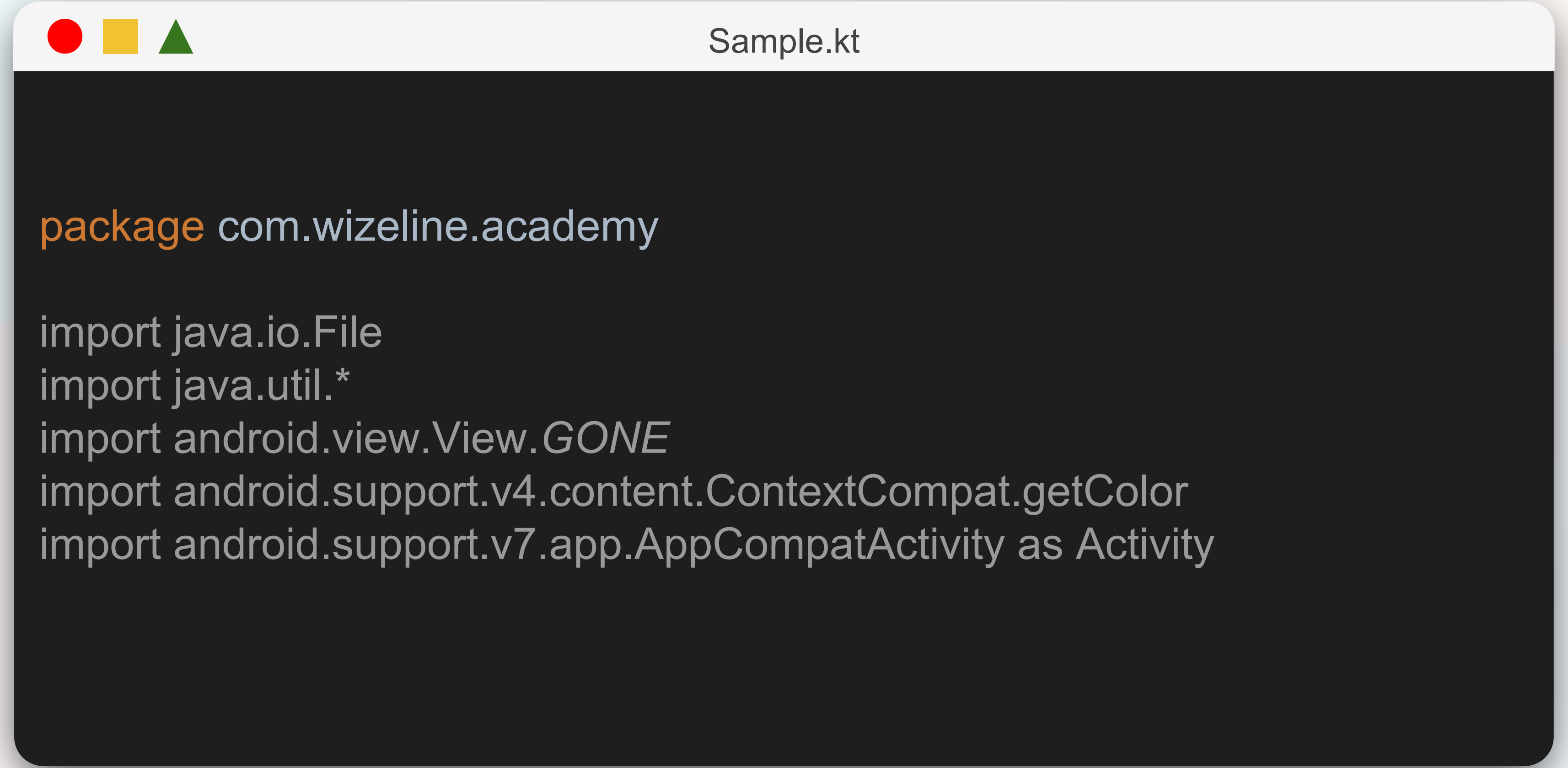
Package & Import

Package & Import



```
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
```

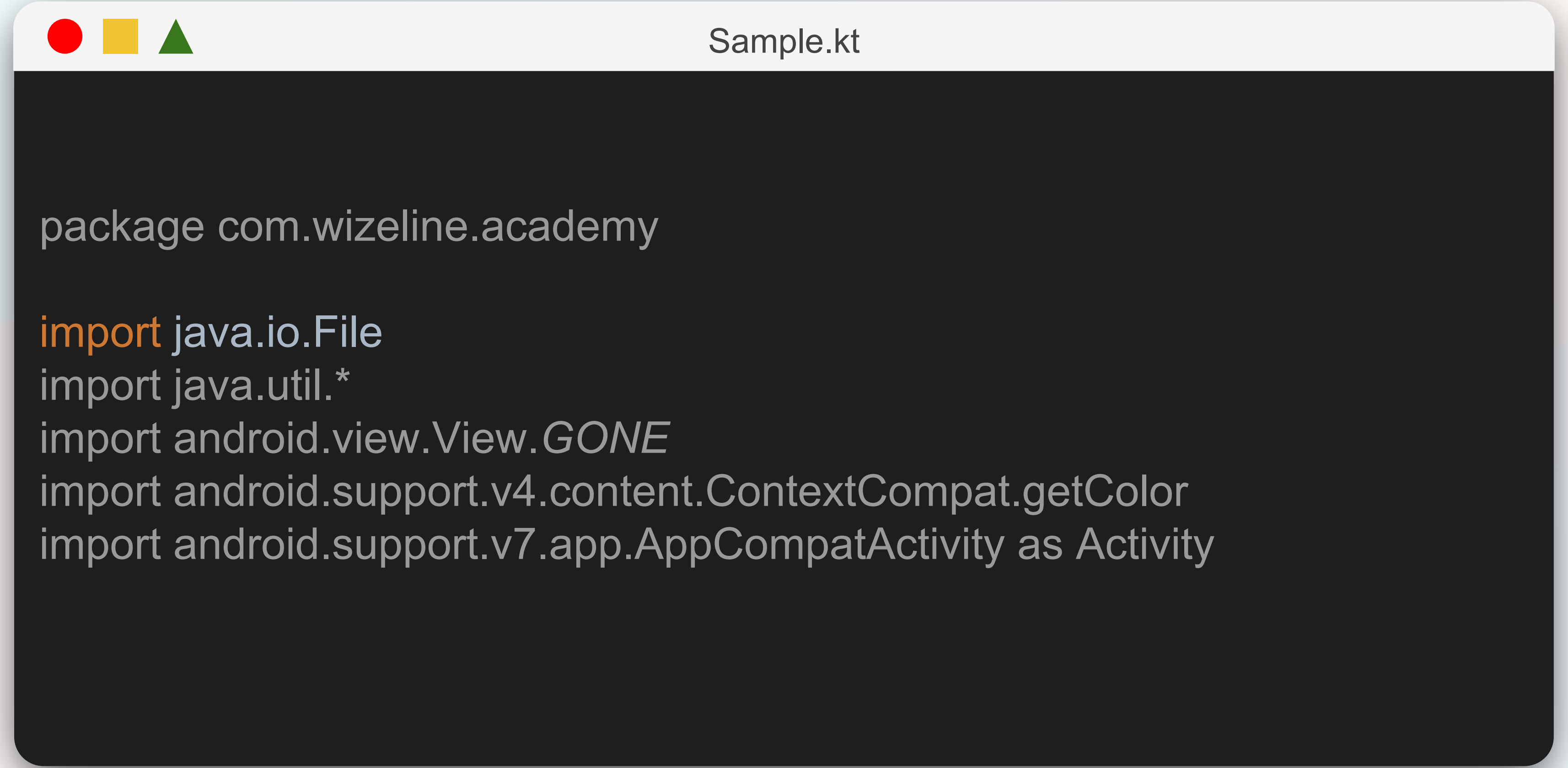
Package & Import



Sample.kt

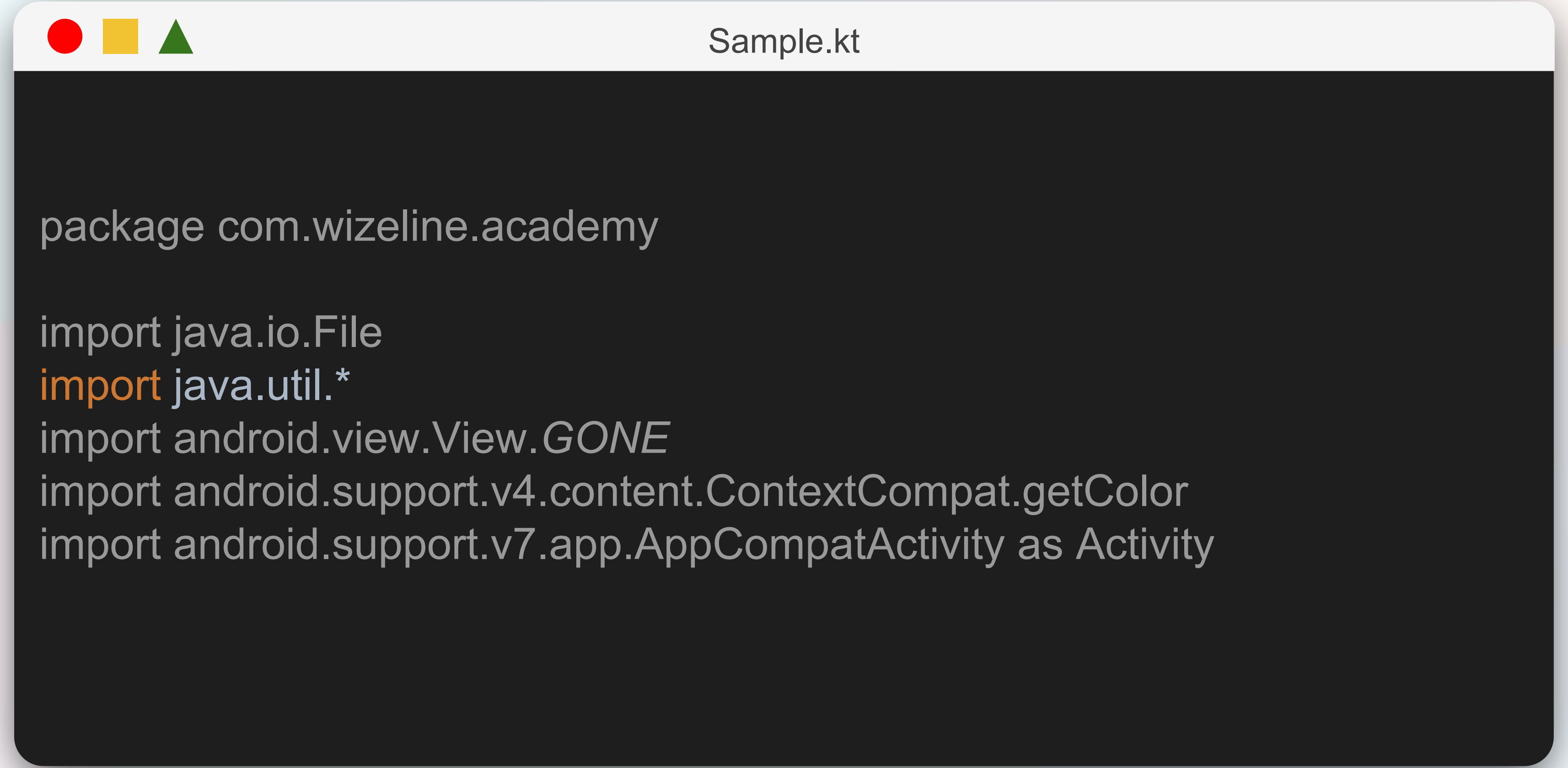
```
package com.wizeline.academy

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
```

Package & Import



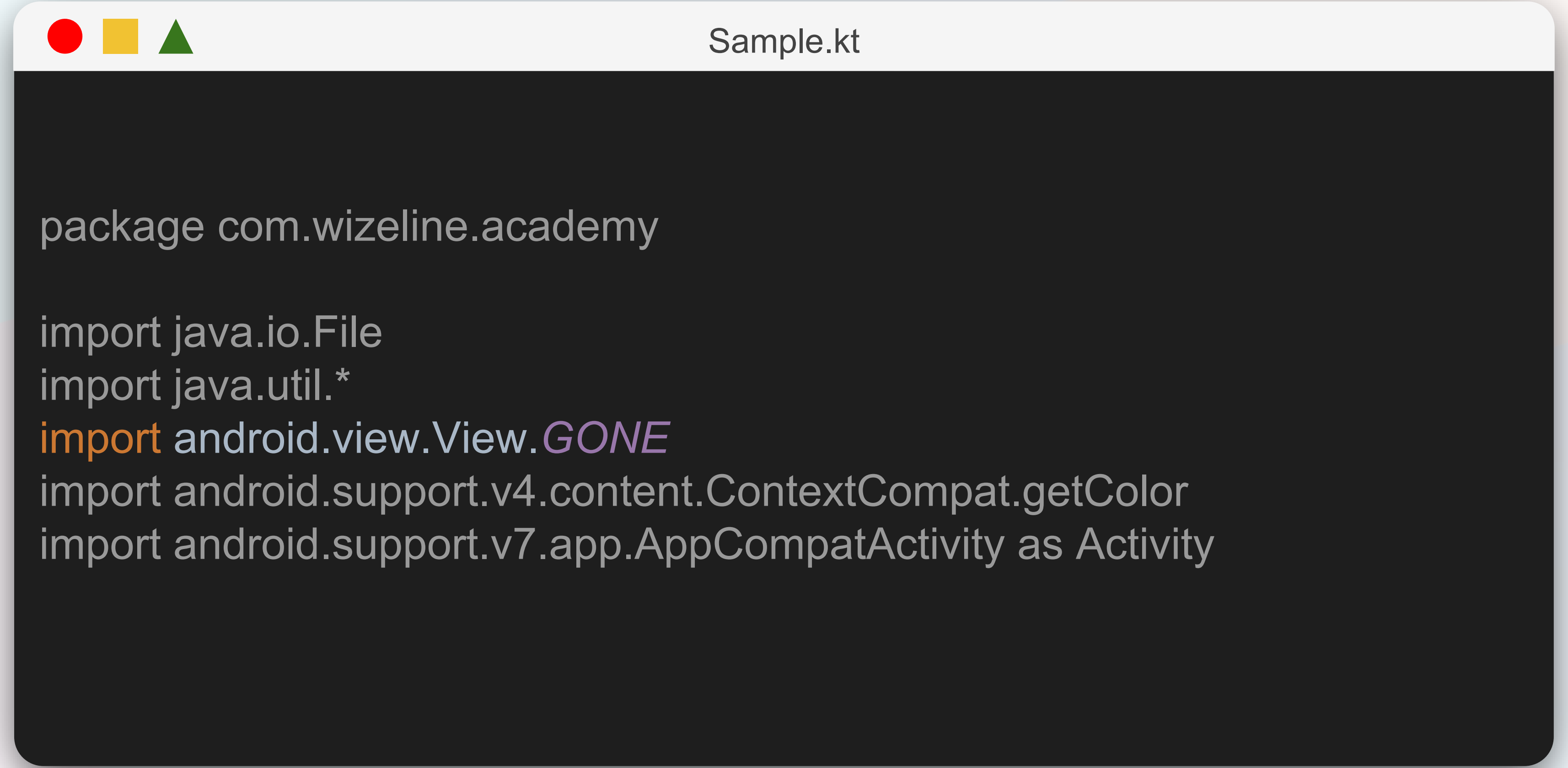
```kotlin
package com.wizeline.academy

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
```

Package & Import


Sample.kt

```
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
```

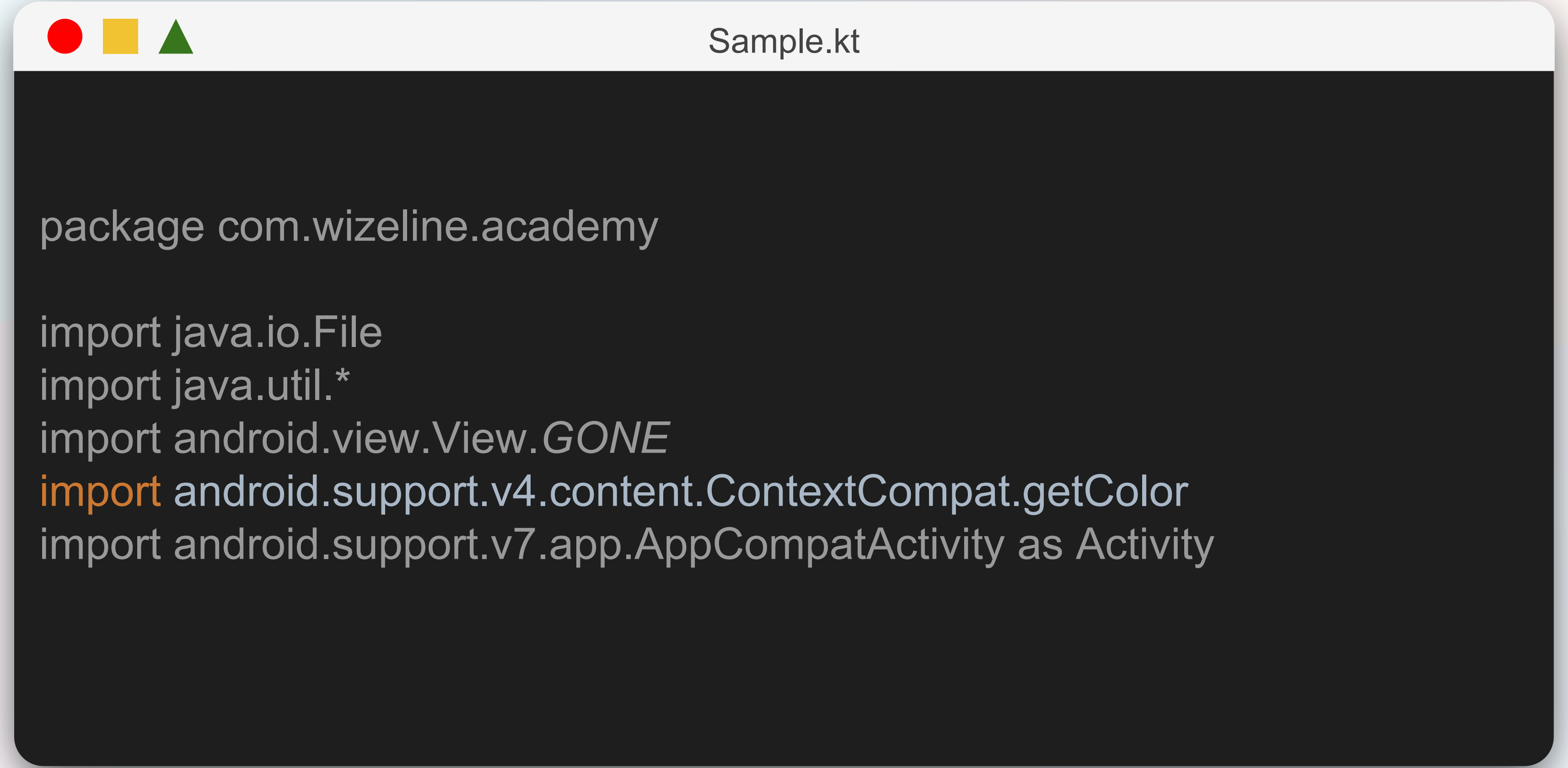
Package & Import



Sample.kt

```
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
```

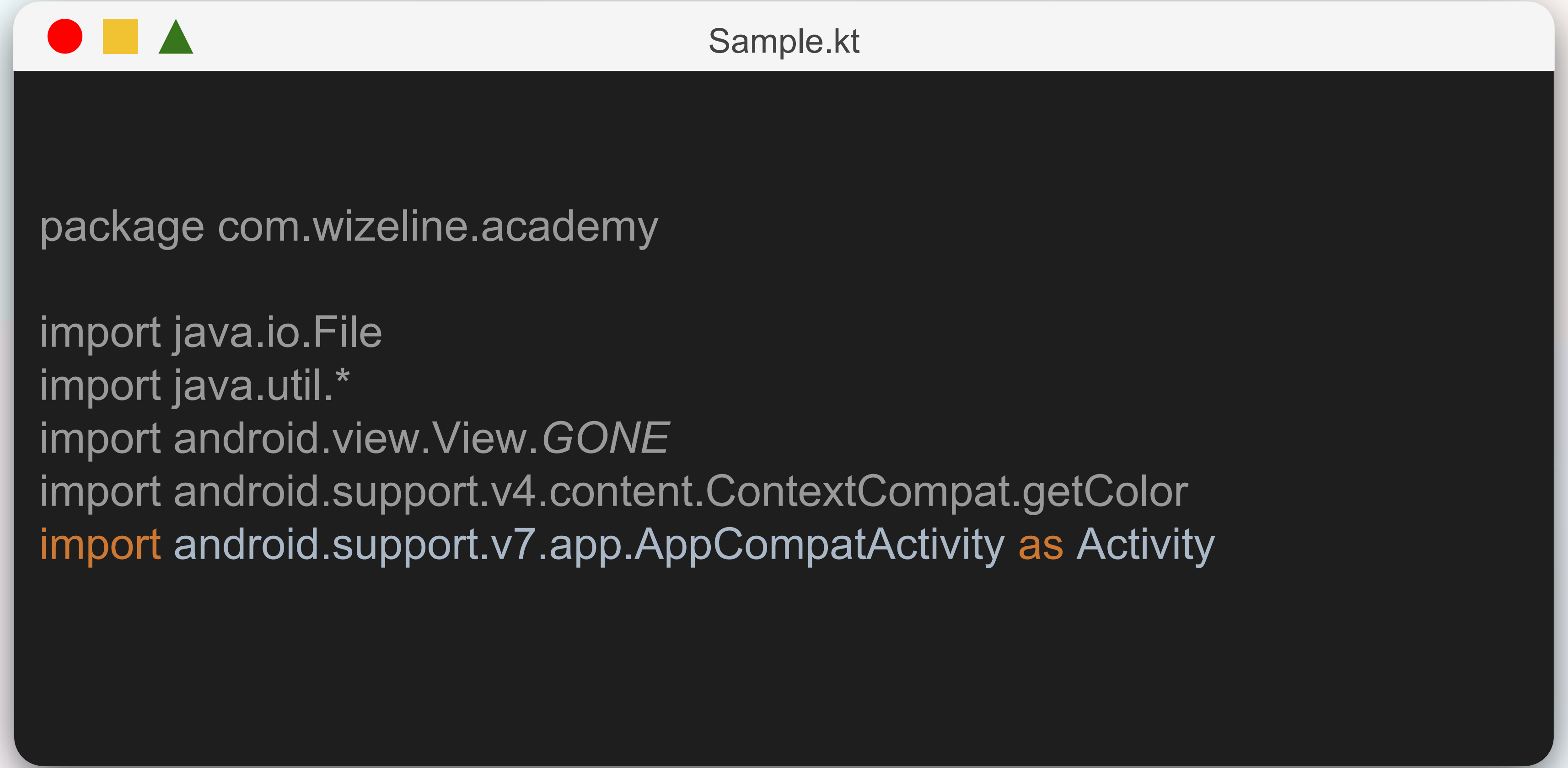
Package & Import



Sample.kt

```
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
```

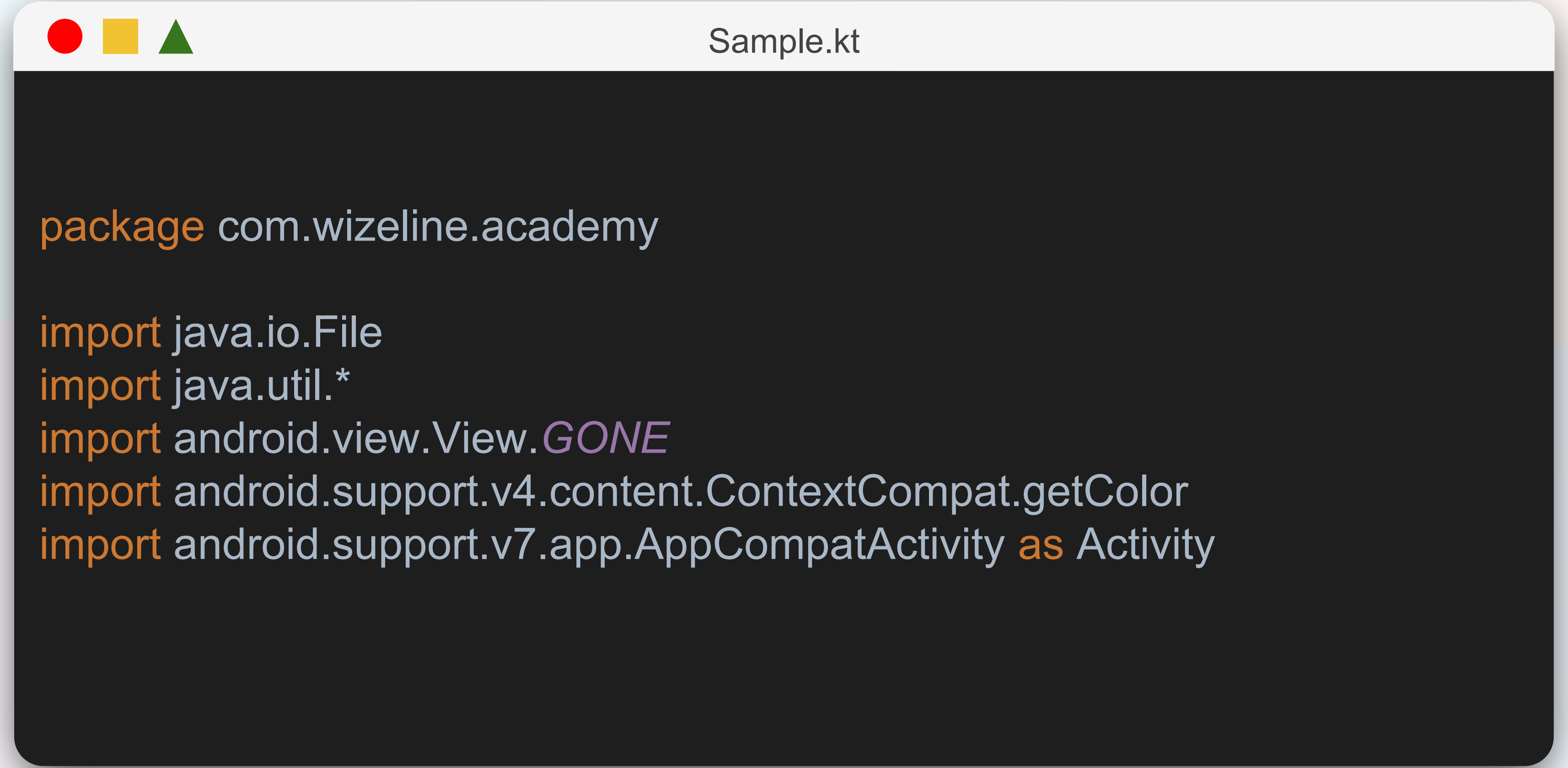
Package & Import



Sample.kt

```
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
```

Package & Import



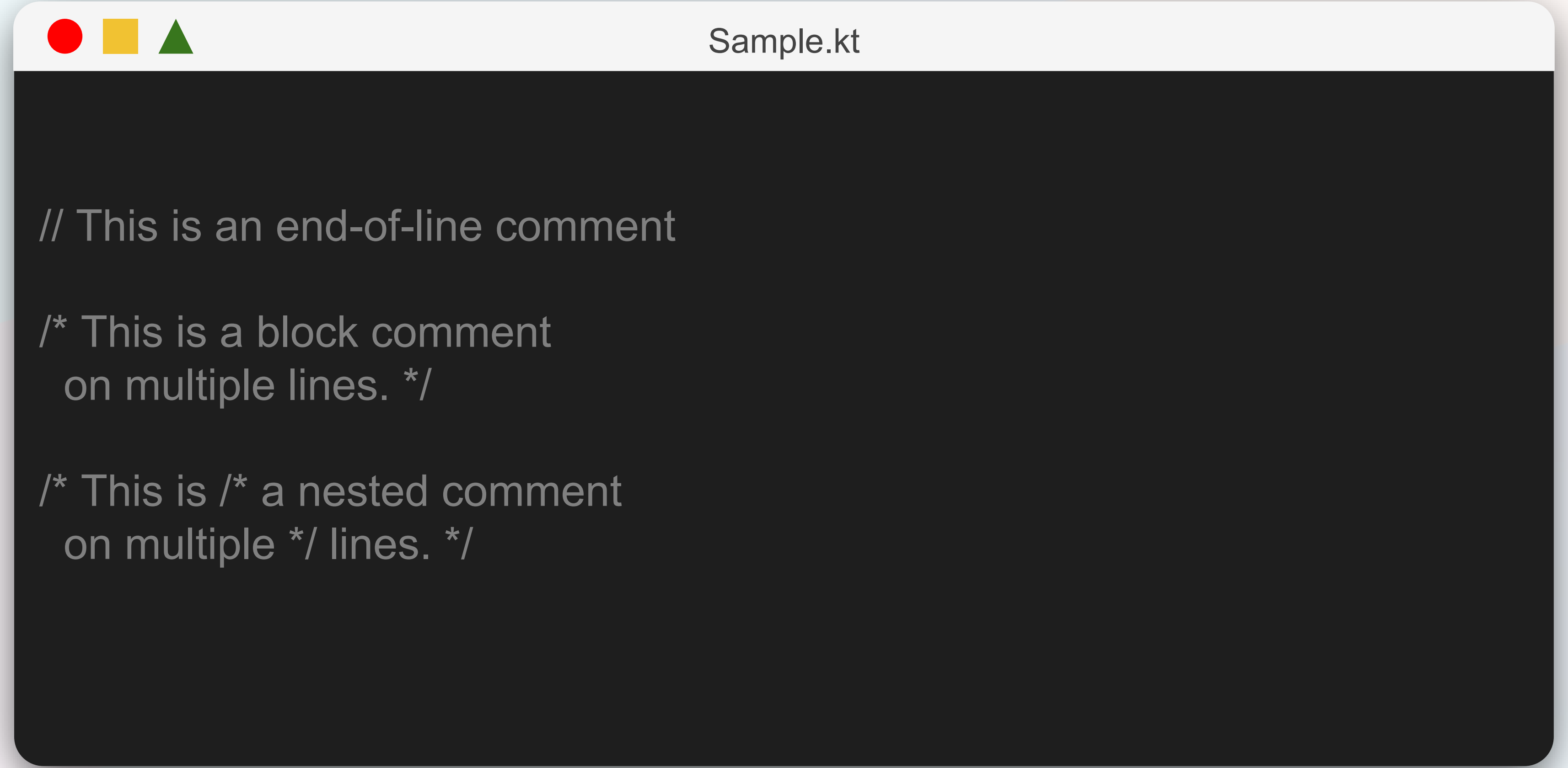
Sample.kt

```
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
```

Basics

Comments

Package & Import



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

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

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

Basics

Variables

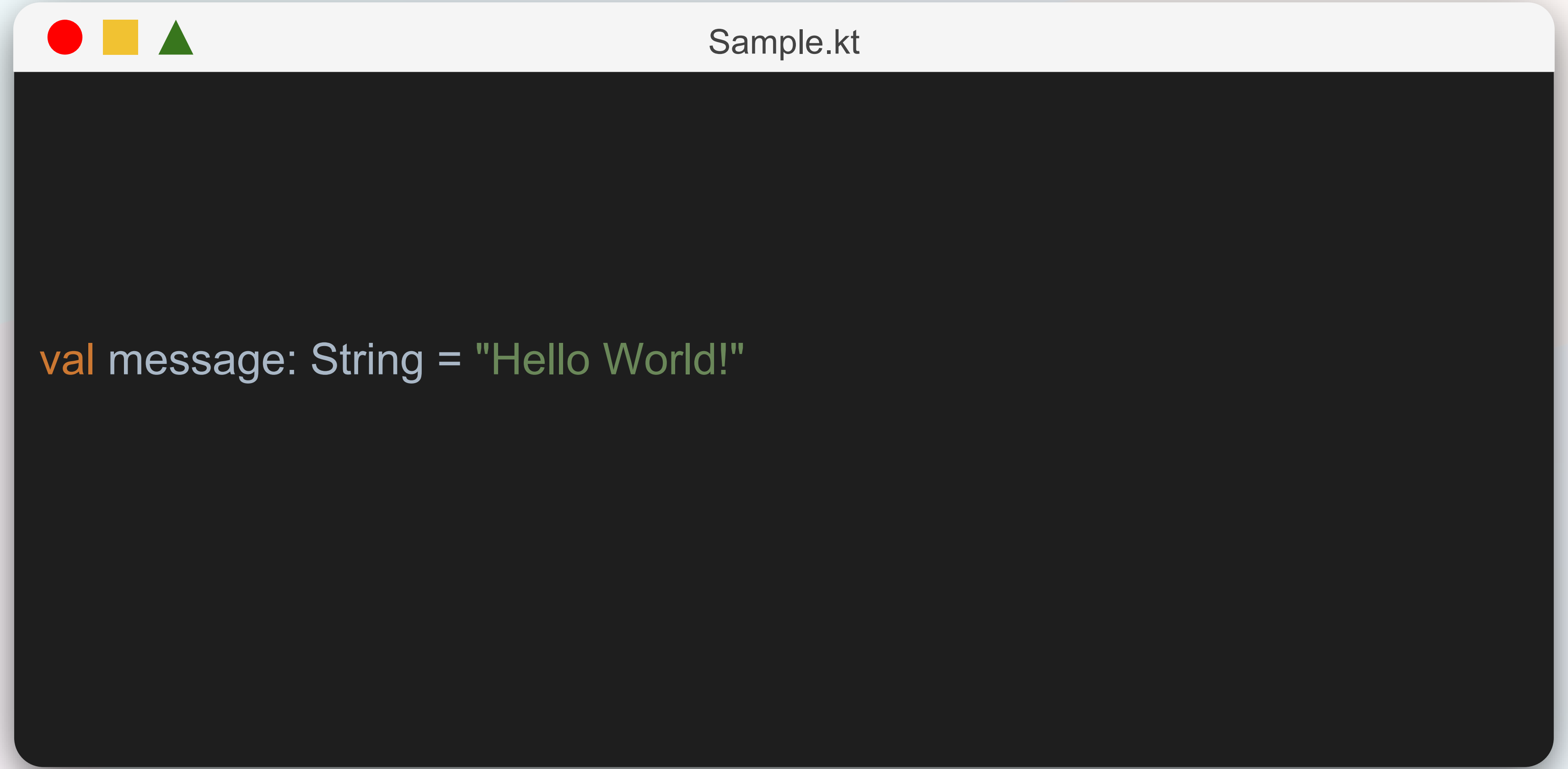
Val *value* :

immutable reference

Var *variable* :

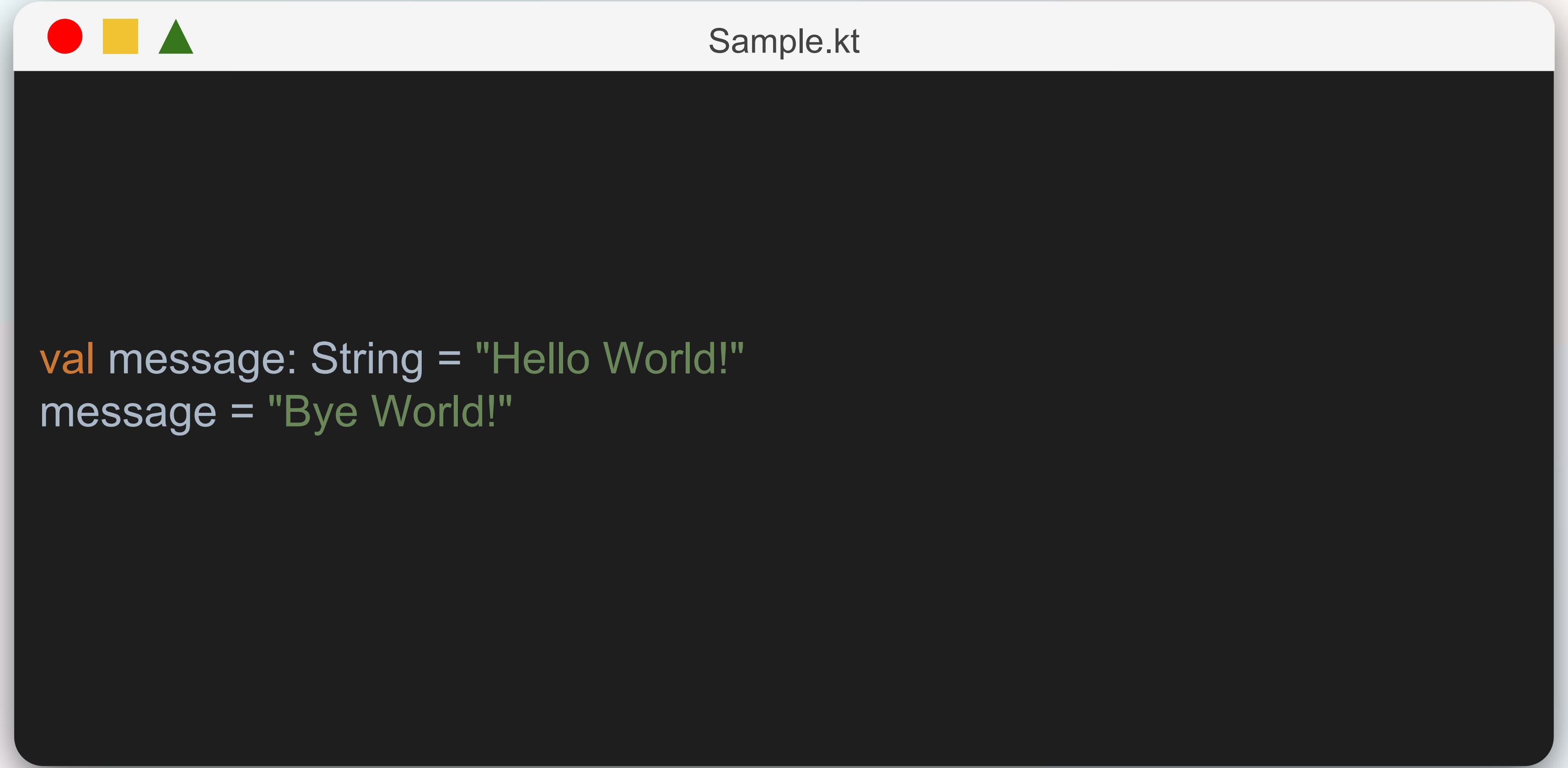
mutable reference

Variables



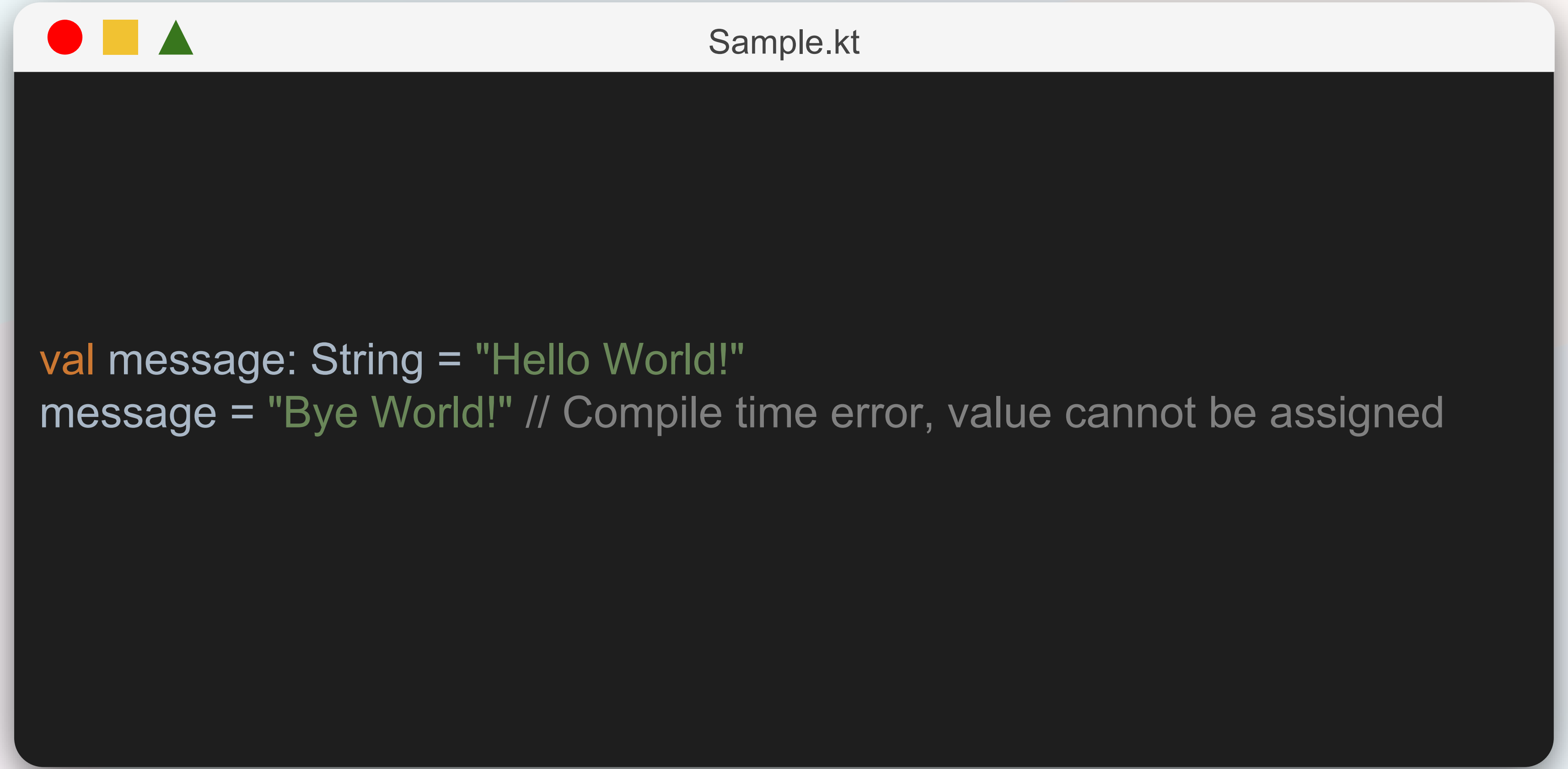
```kotlin
val message: String = "Hello World!"
```

Variables


Sample.kt

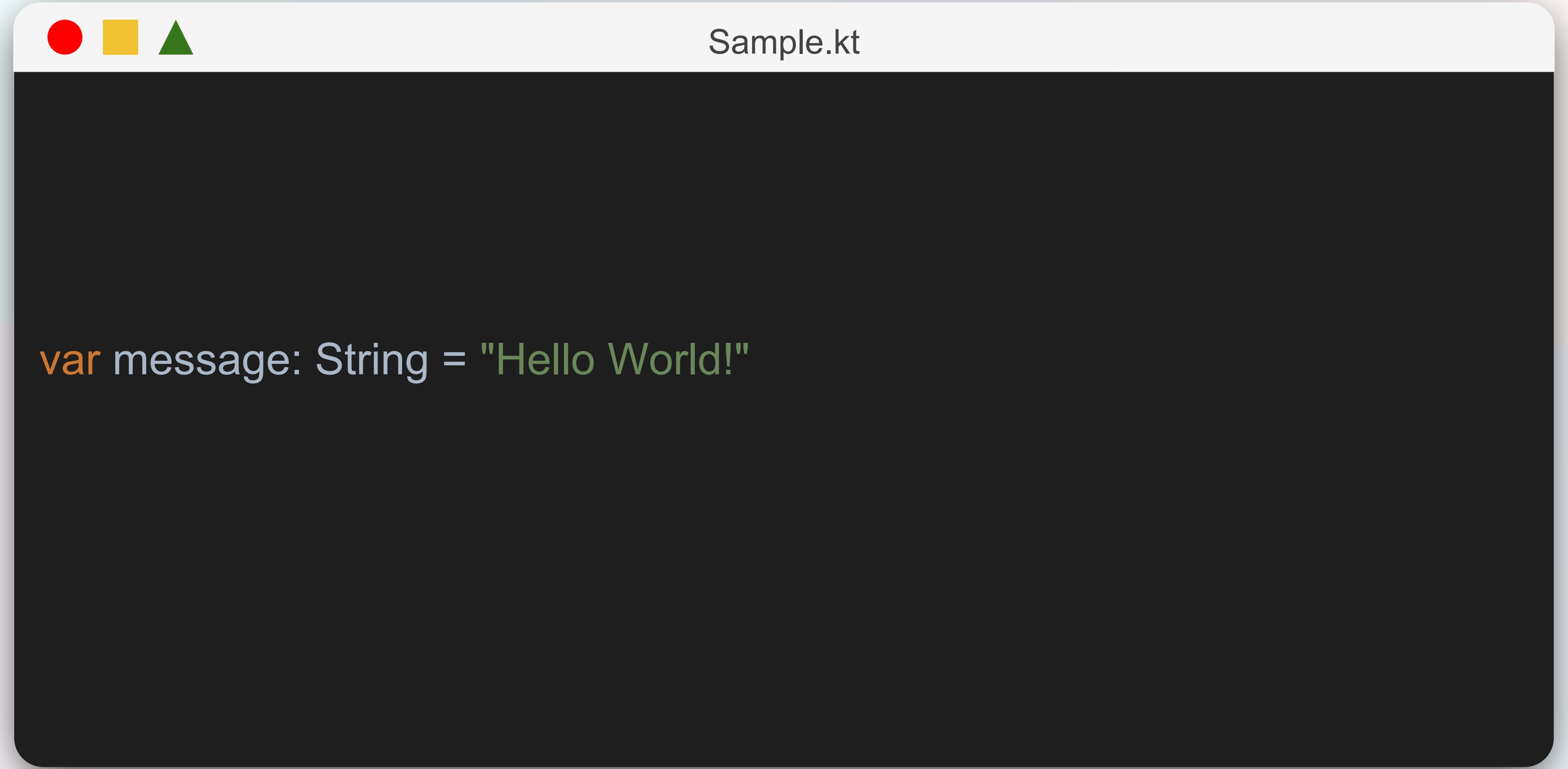
```kotlin
val message: String = "Hello World!"
message = "Bye World!"
```

Variables



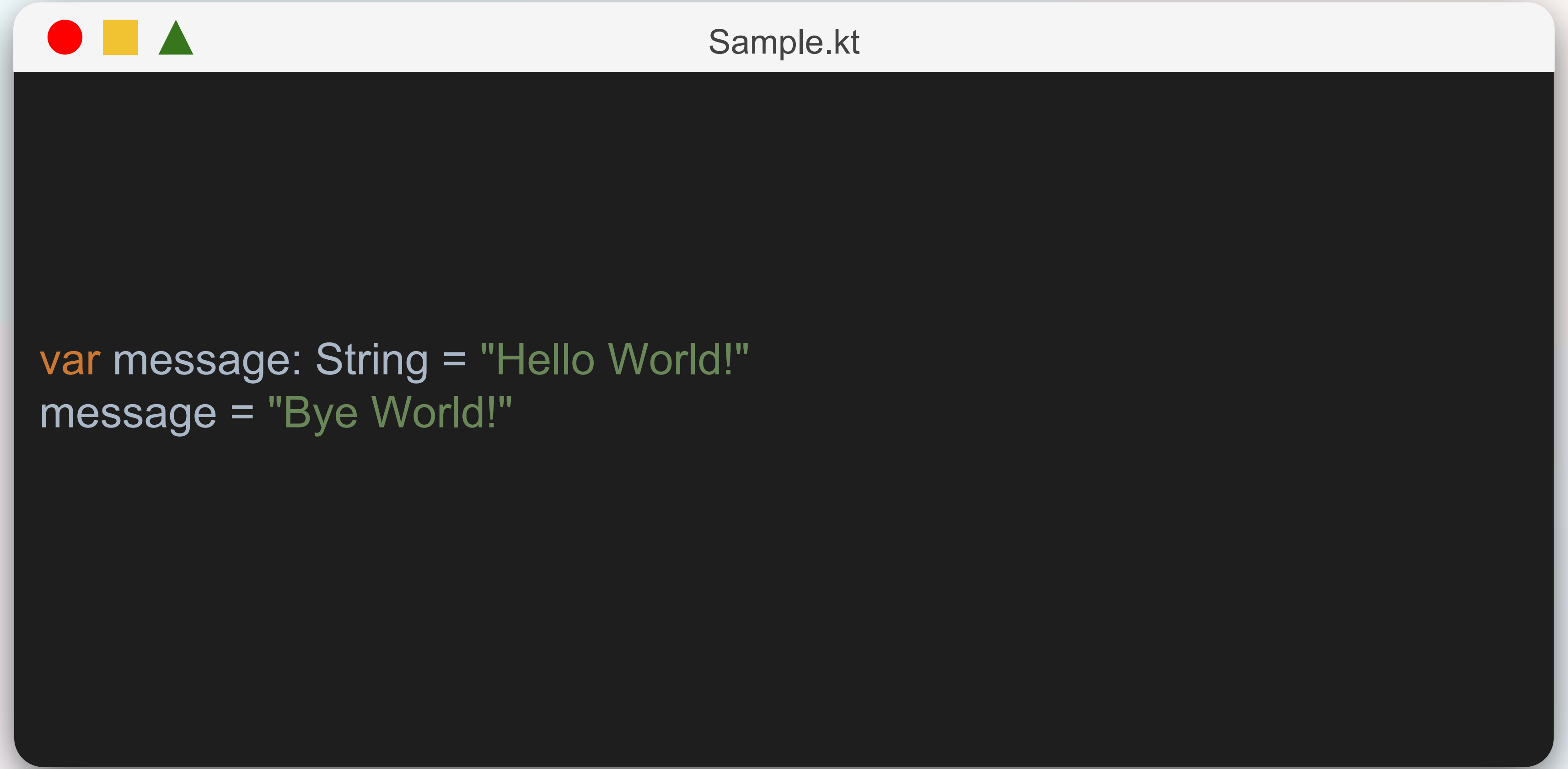
```kotlin
val message: String = "Hello World!"
message = "Bye World!" // Compile time error, value cannot be assigned
```

Variables


Sample.kt

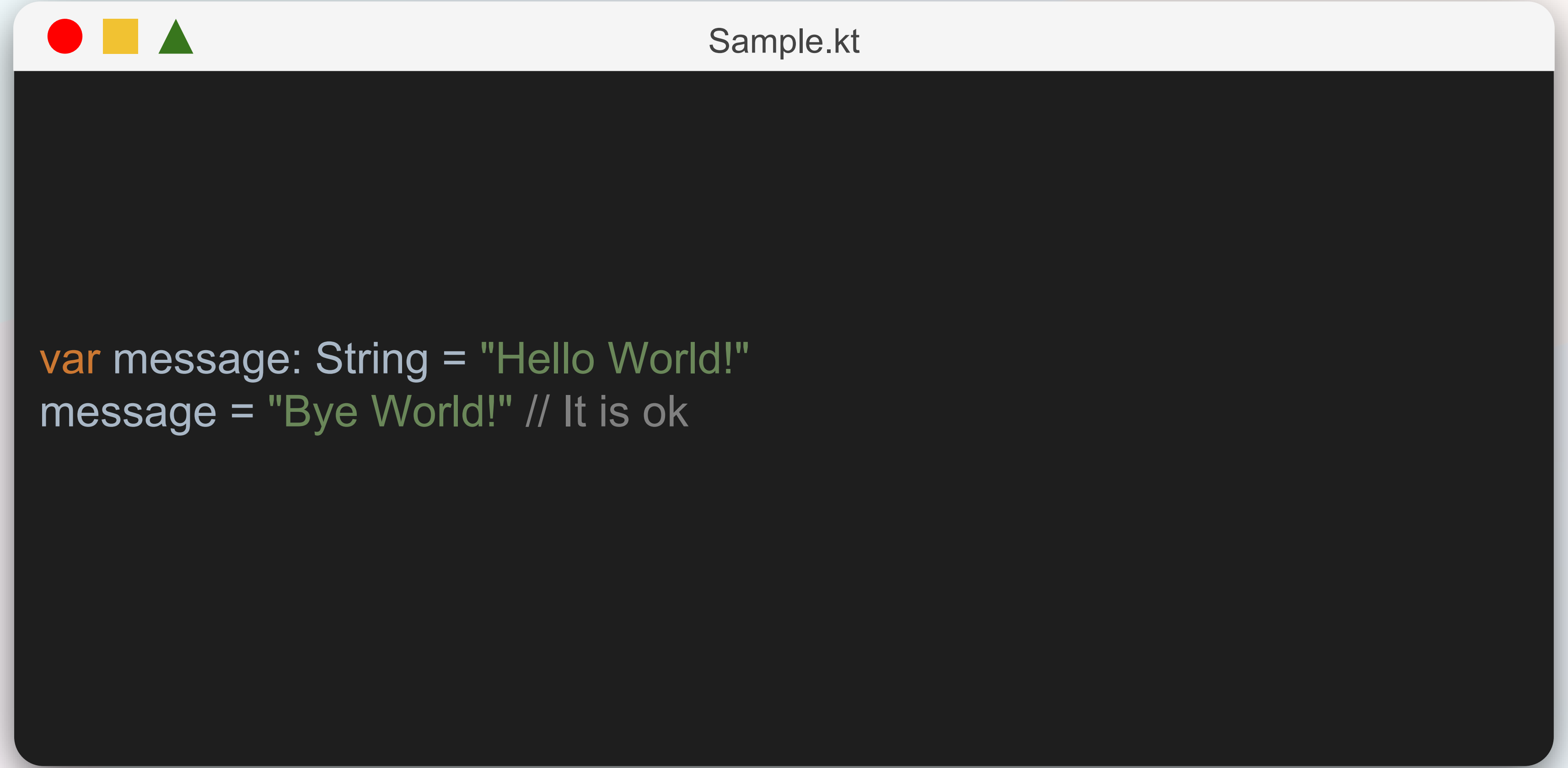
```kotlin
var message: String = "Hello World!"
```

Variables



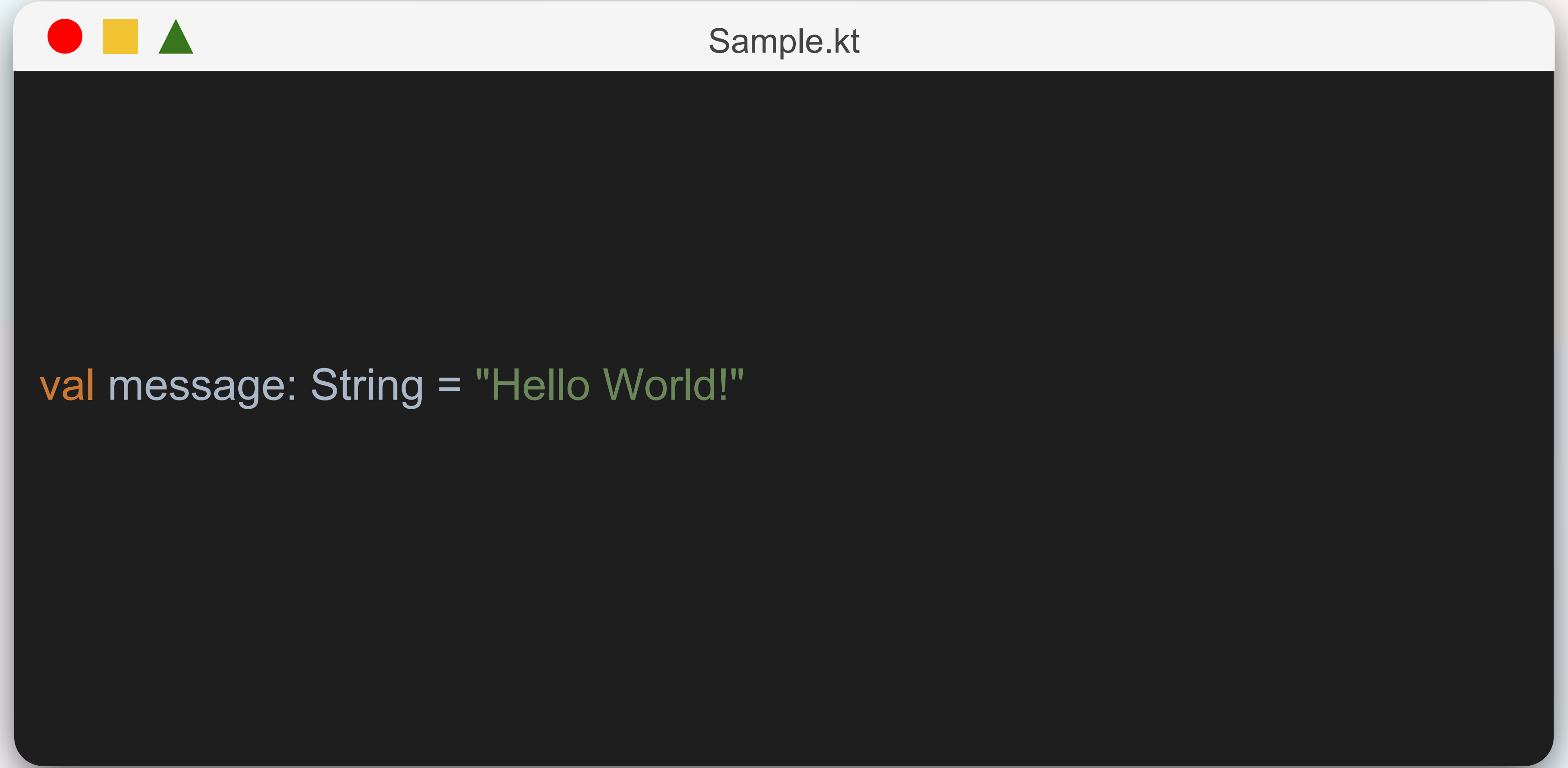
```kotlin
var message: String = "Hello World!"
message = "Bye World!"
```

Variables


Sample.kt

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

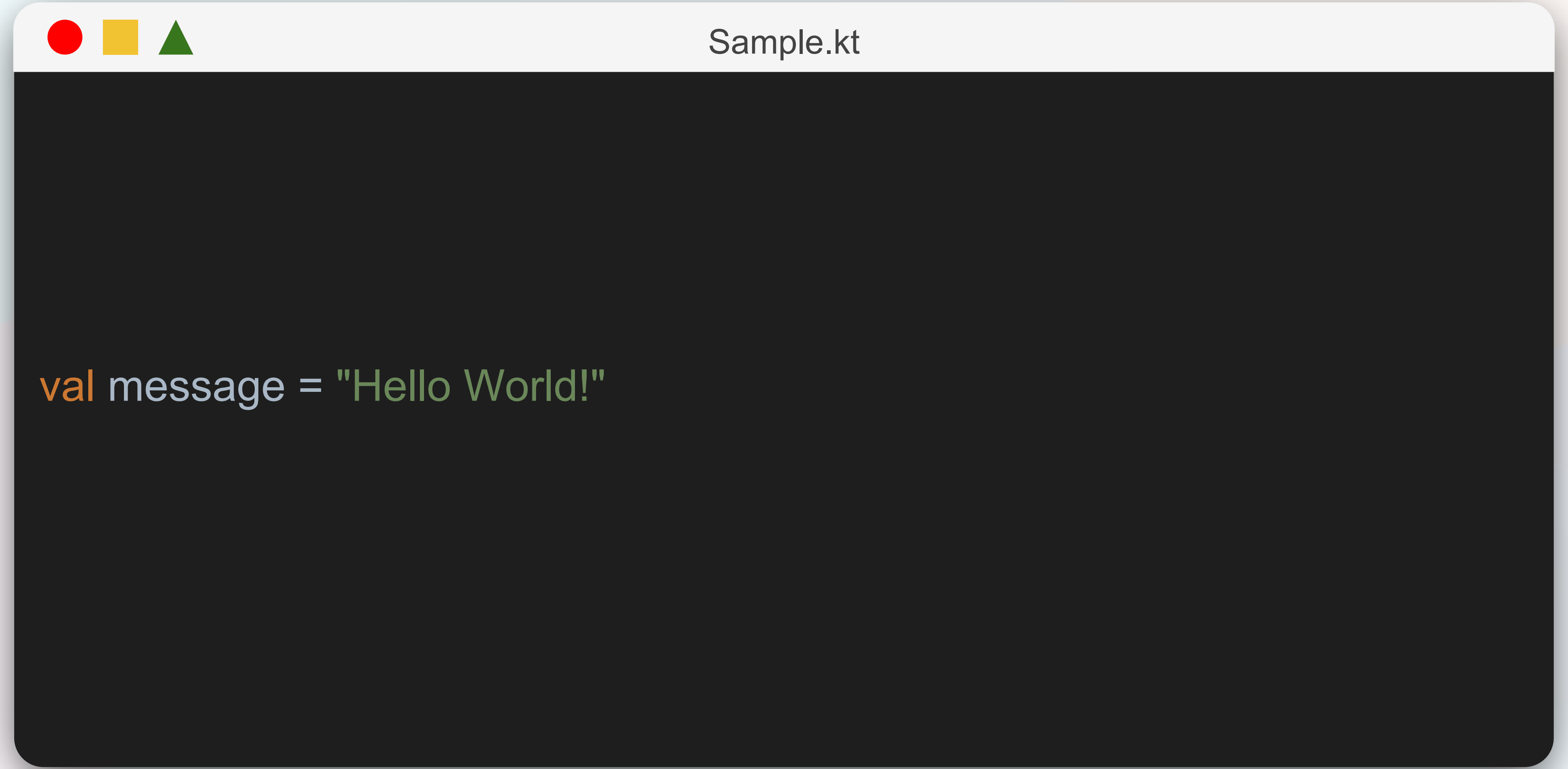
Variables



Sample.kt

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

Variables

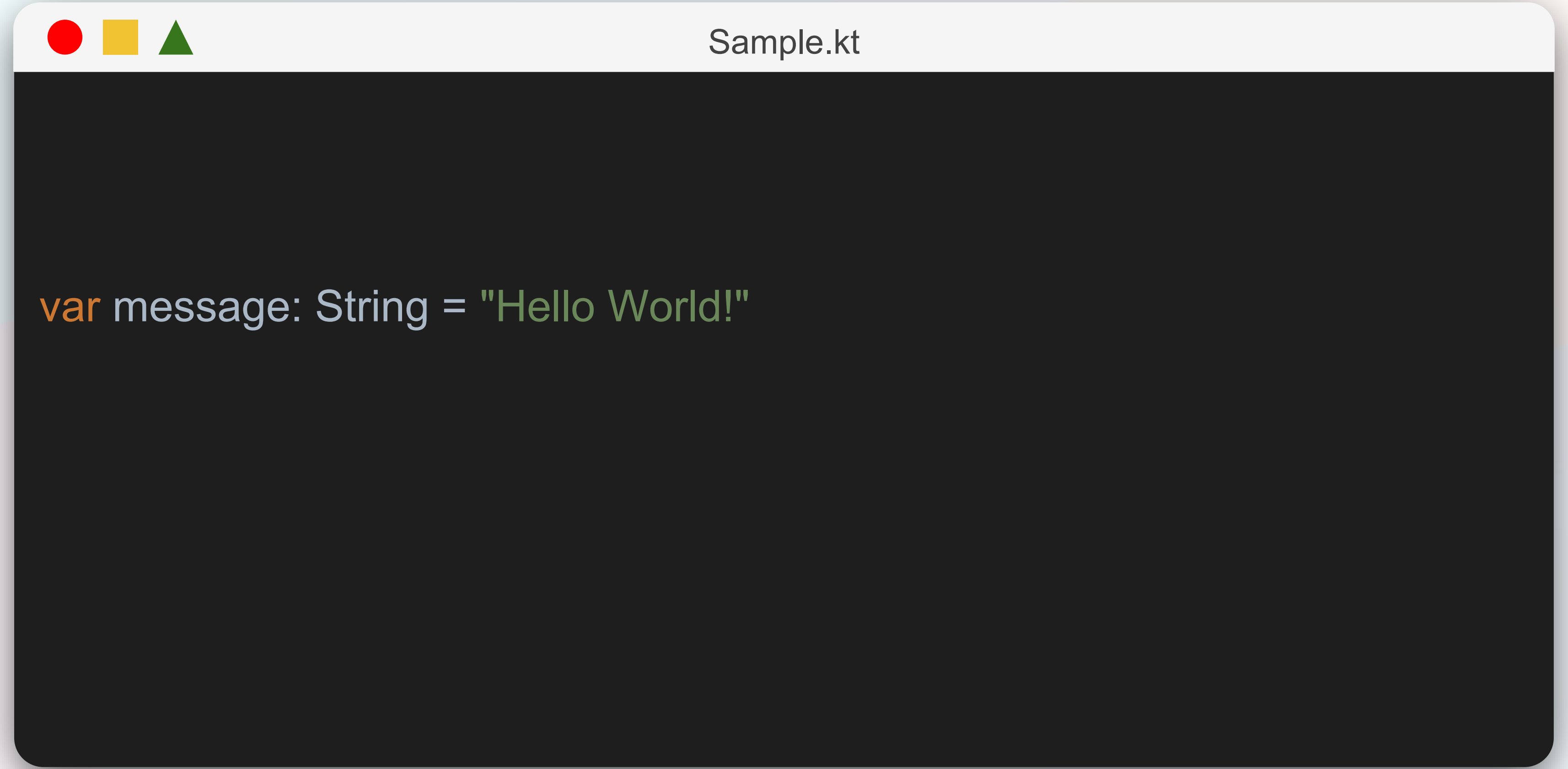

Sample.kt

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

Basics

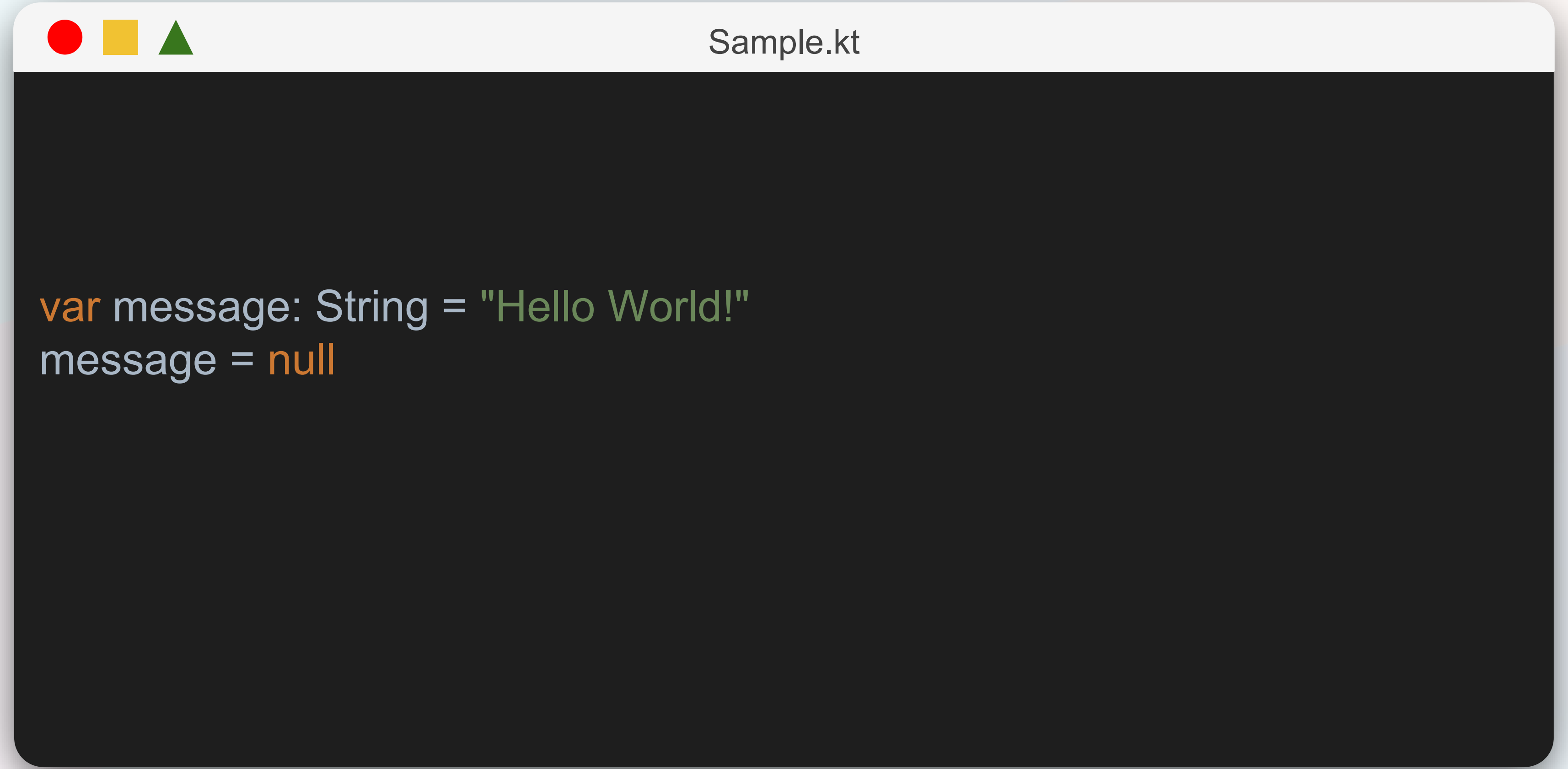
Null Safety

Null Safety


Sample.kt

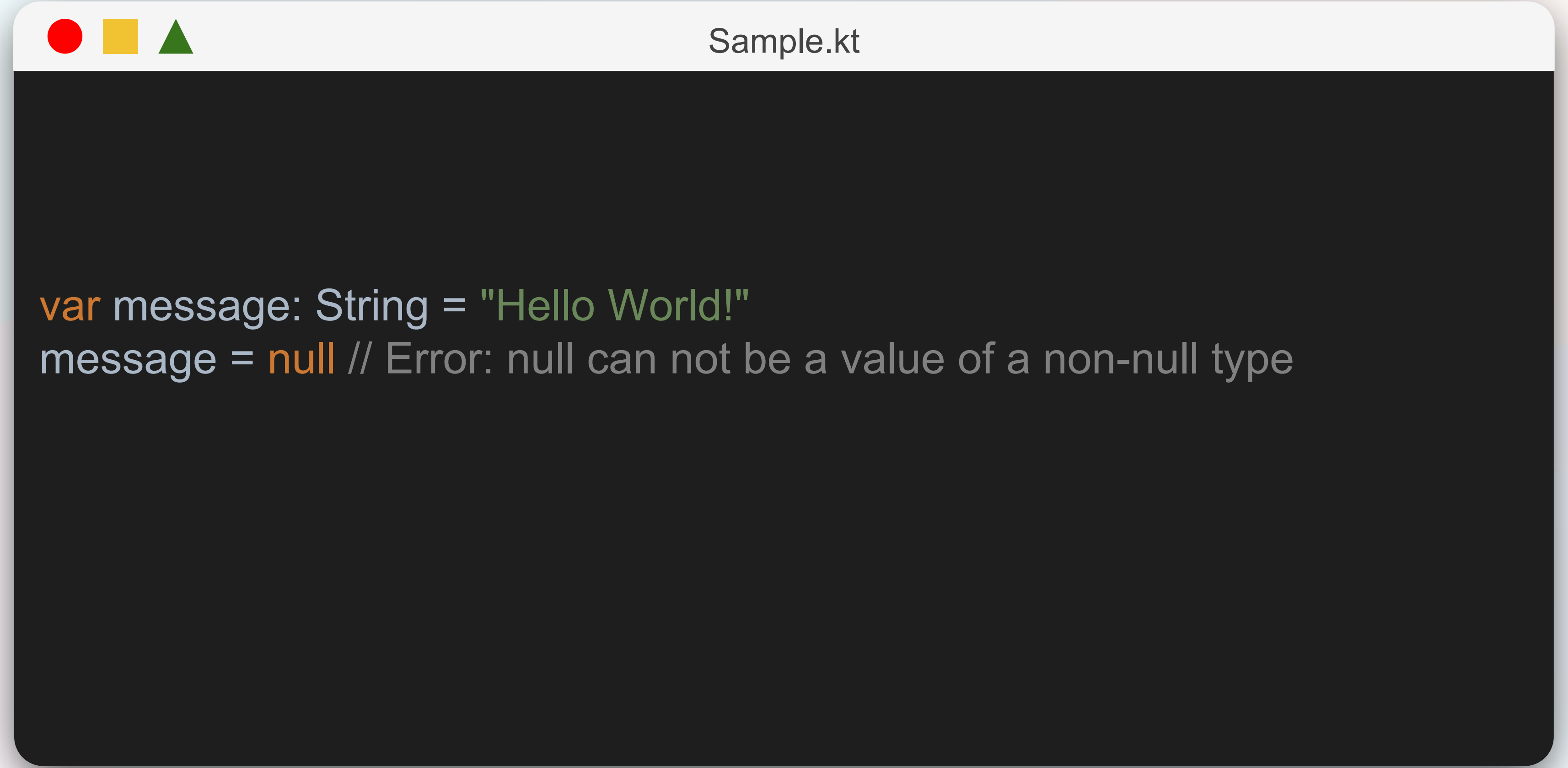
```kotlin
var message: String = "Hello World!"
```

Null Safety


Sample.kt

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

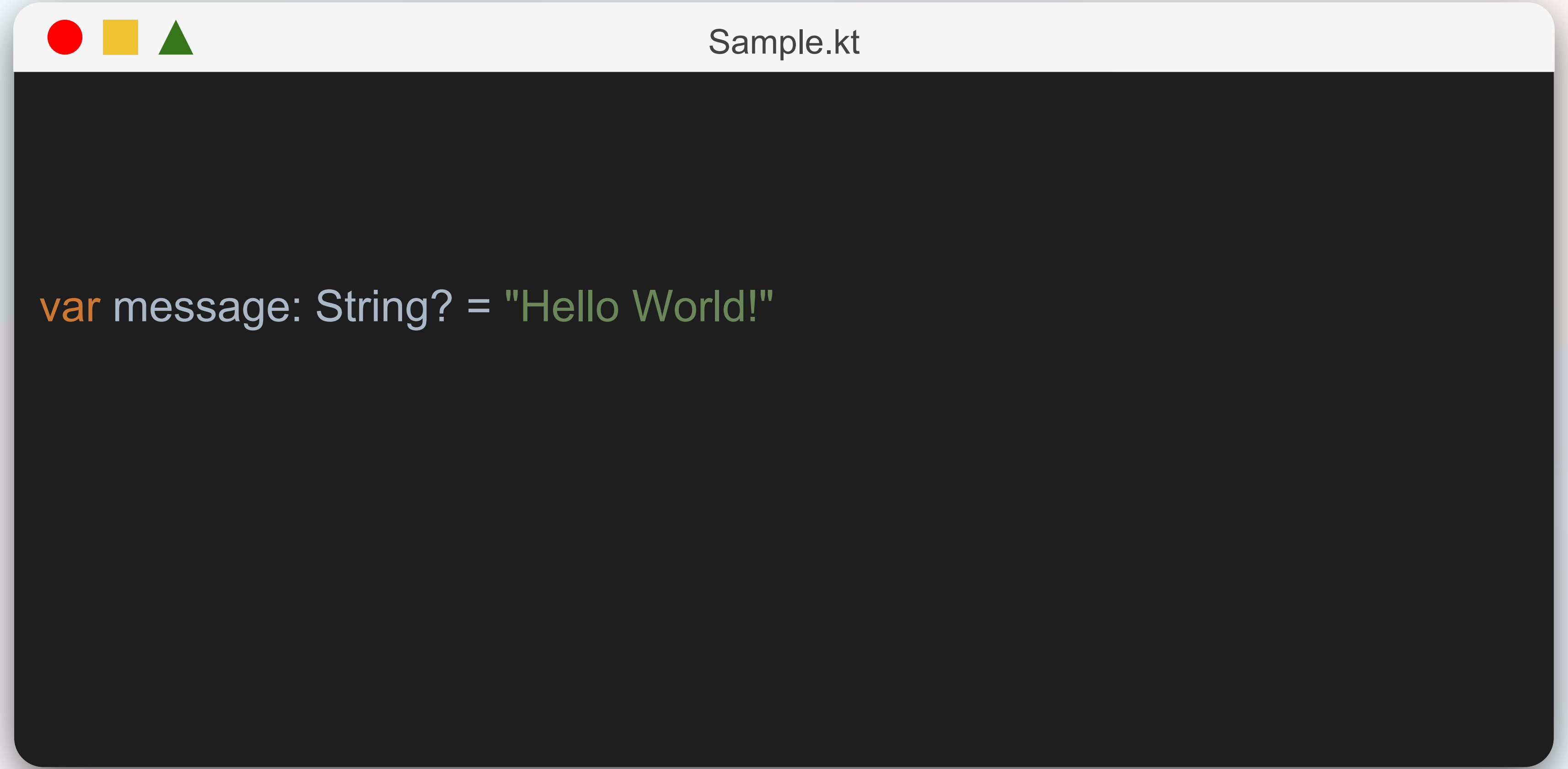
Null Safety



Sample.kt

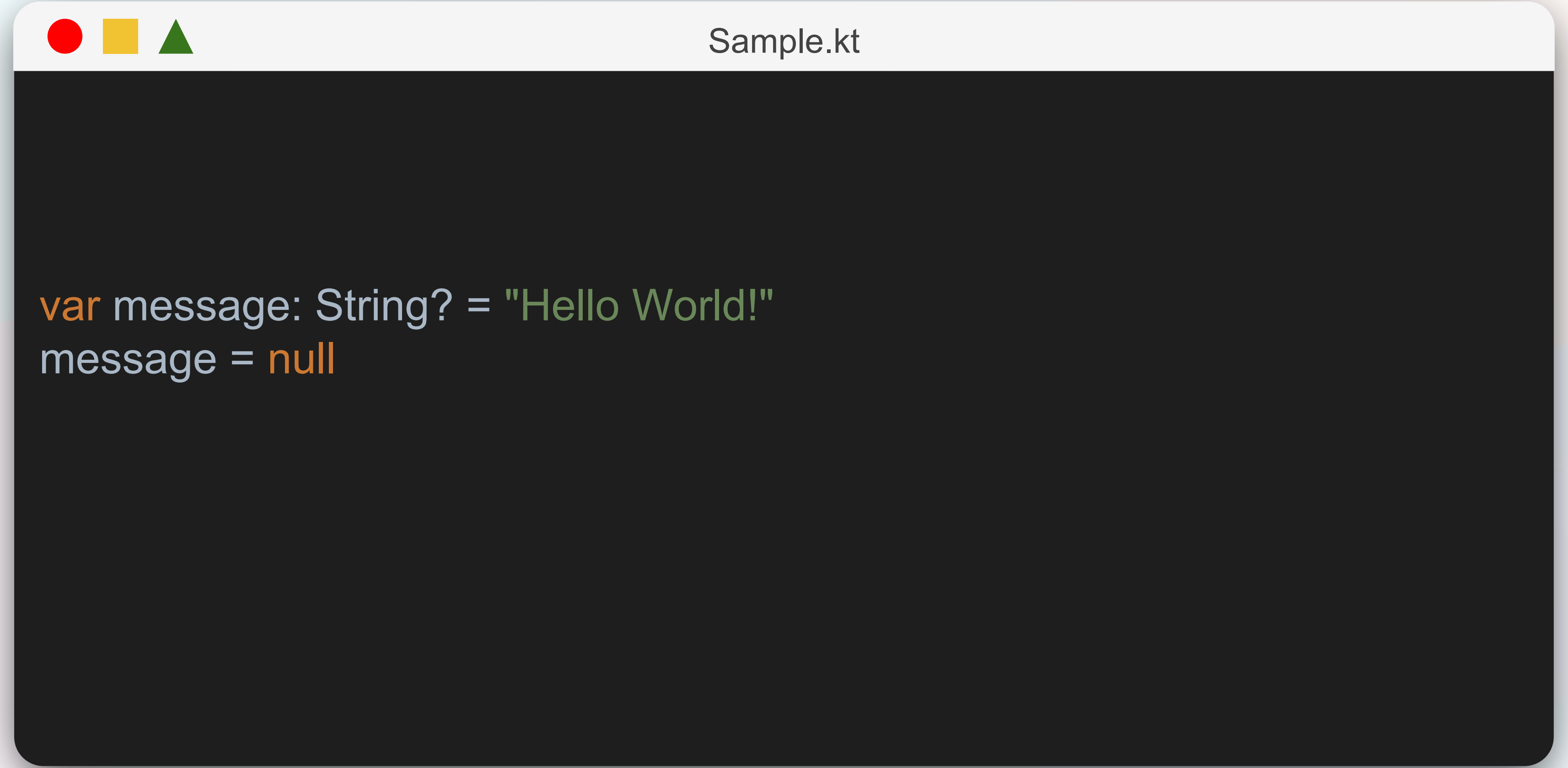
```kotlin
var message: String = "Hello World!"
message = null // Error: null can not be a value of a non-null type
```

Null Safety


Sample.kt

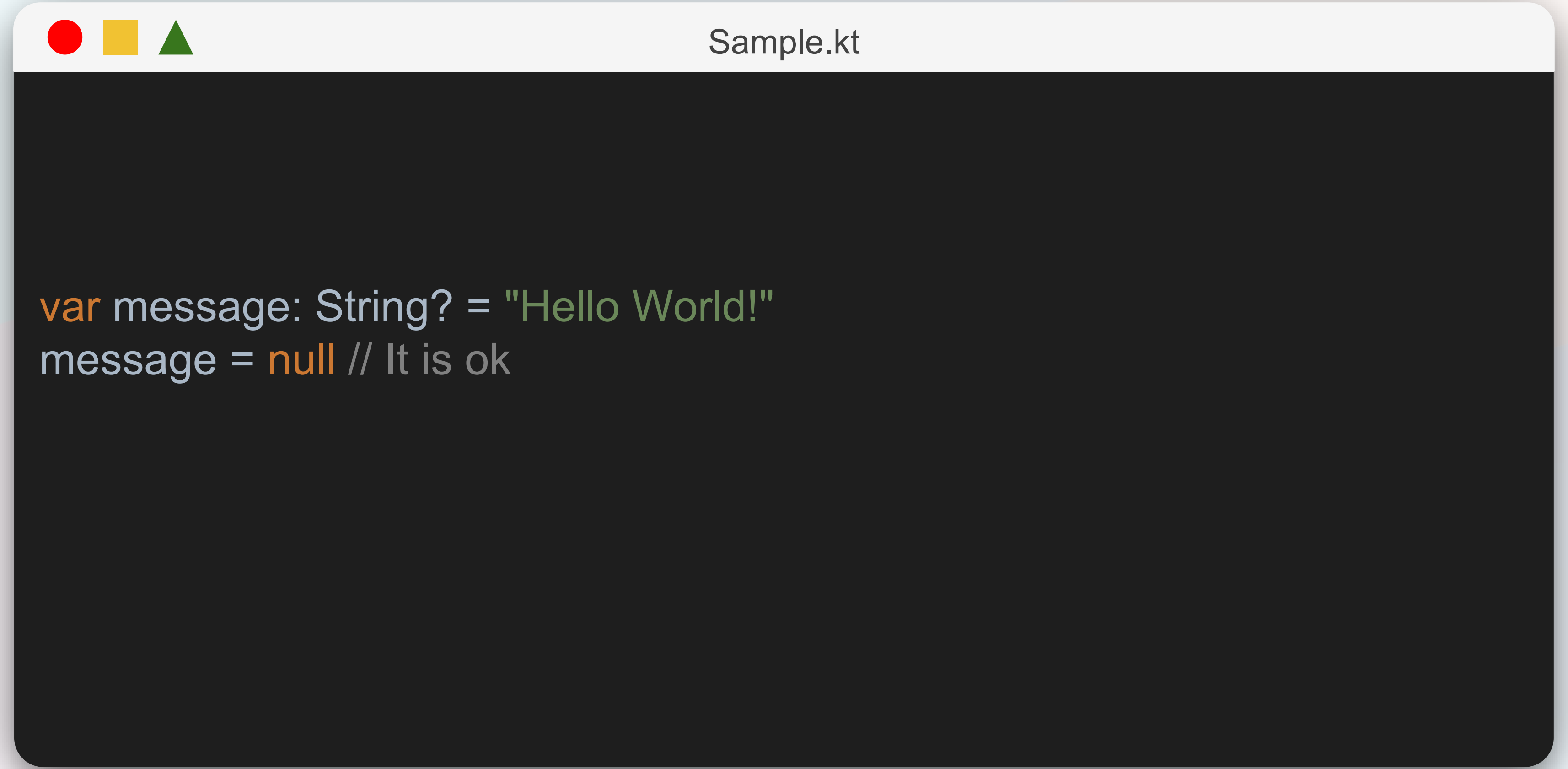
```kotlin
var message: String? = "Hello World!"
```

Null Safety



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

Null Safety



Sample.kt

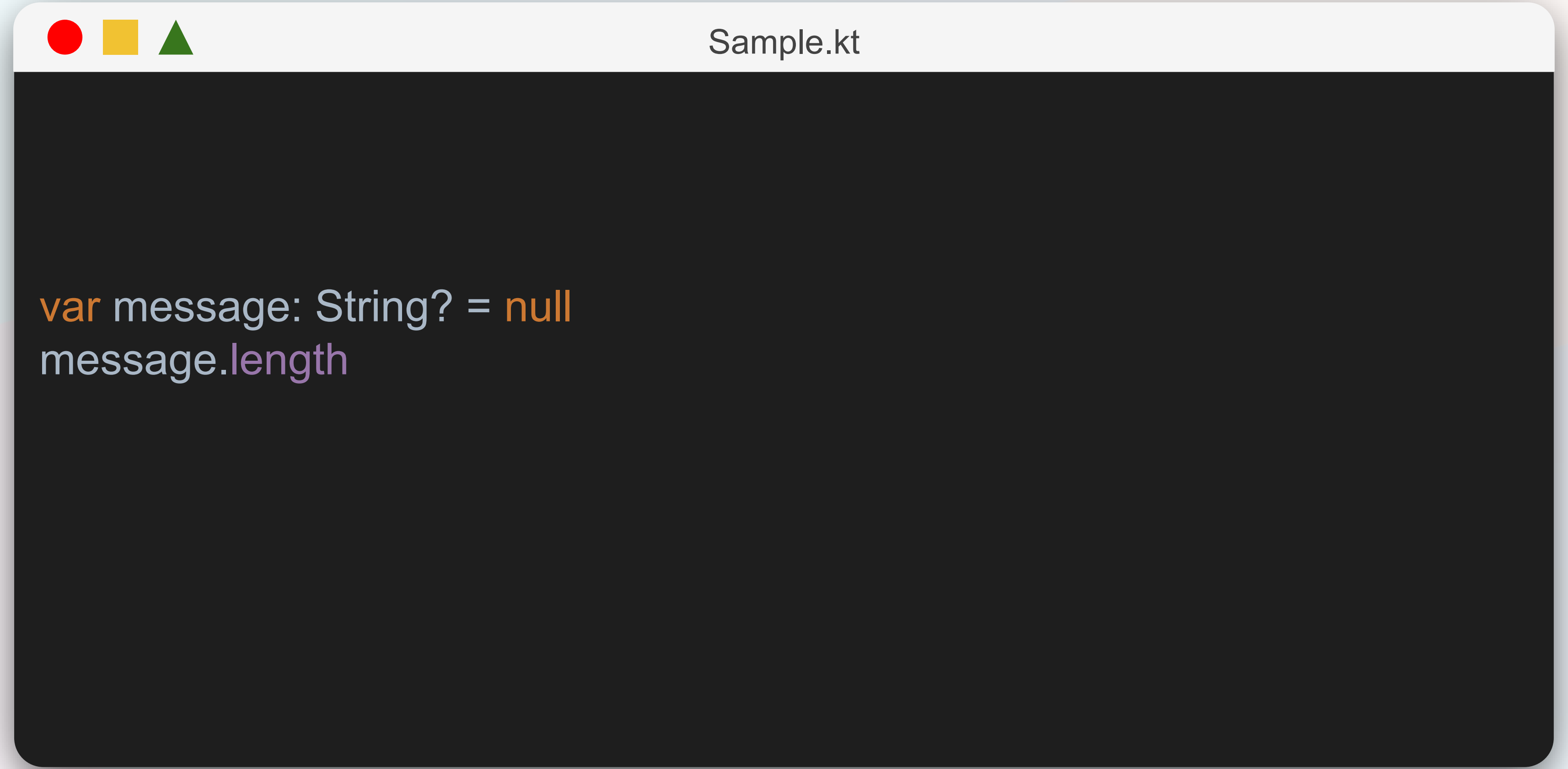
```kotlin
var message: String? = "Hello World!"
message = null // It is ok
```

Null Safety

Sample.kt

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

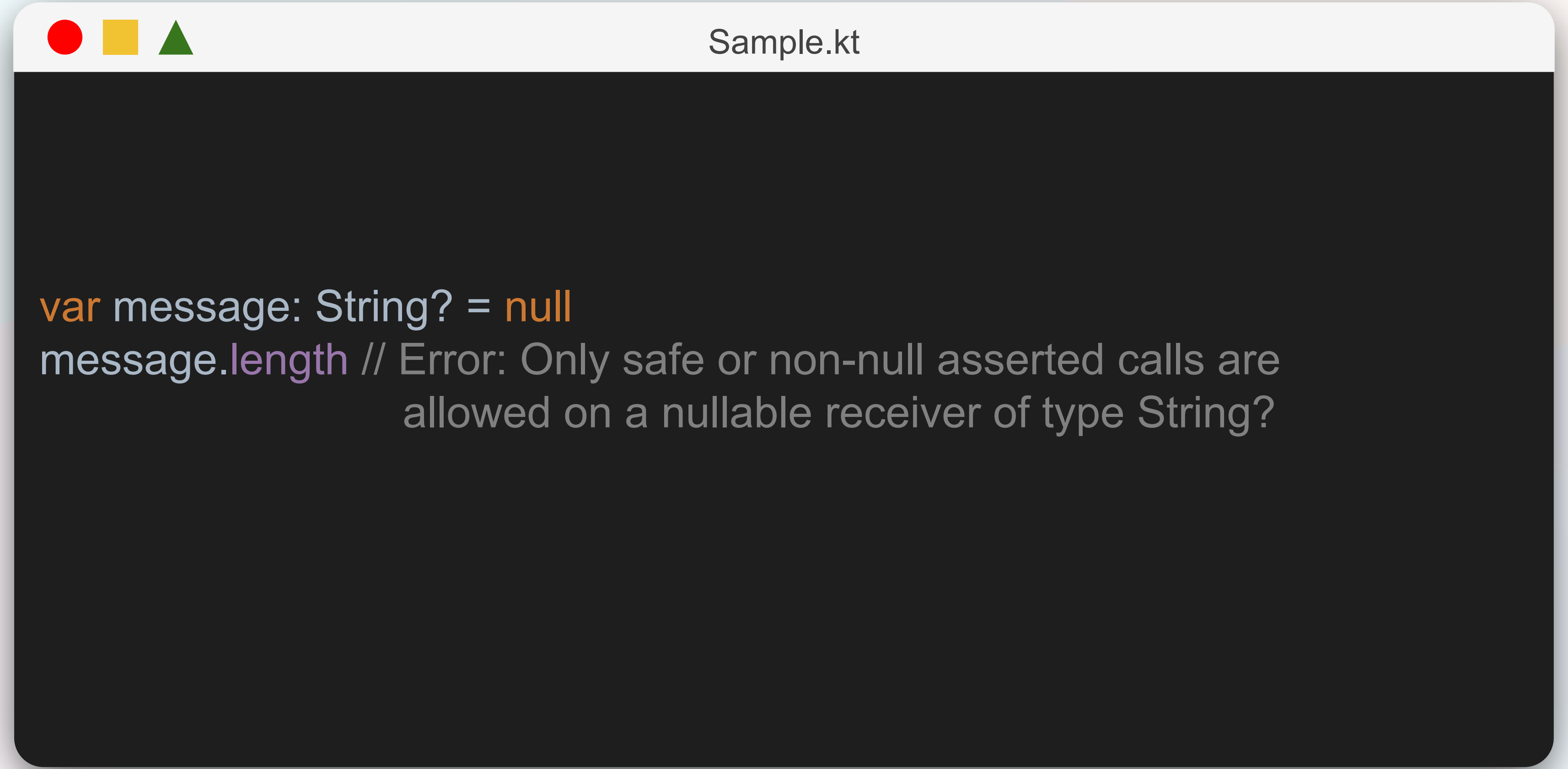
Null Safety



Sample.kt

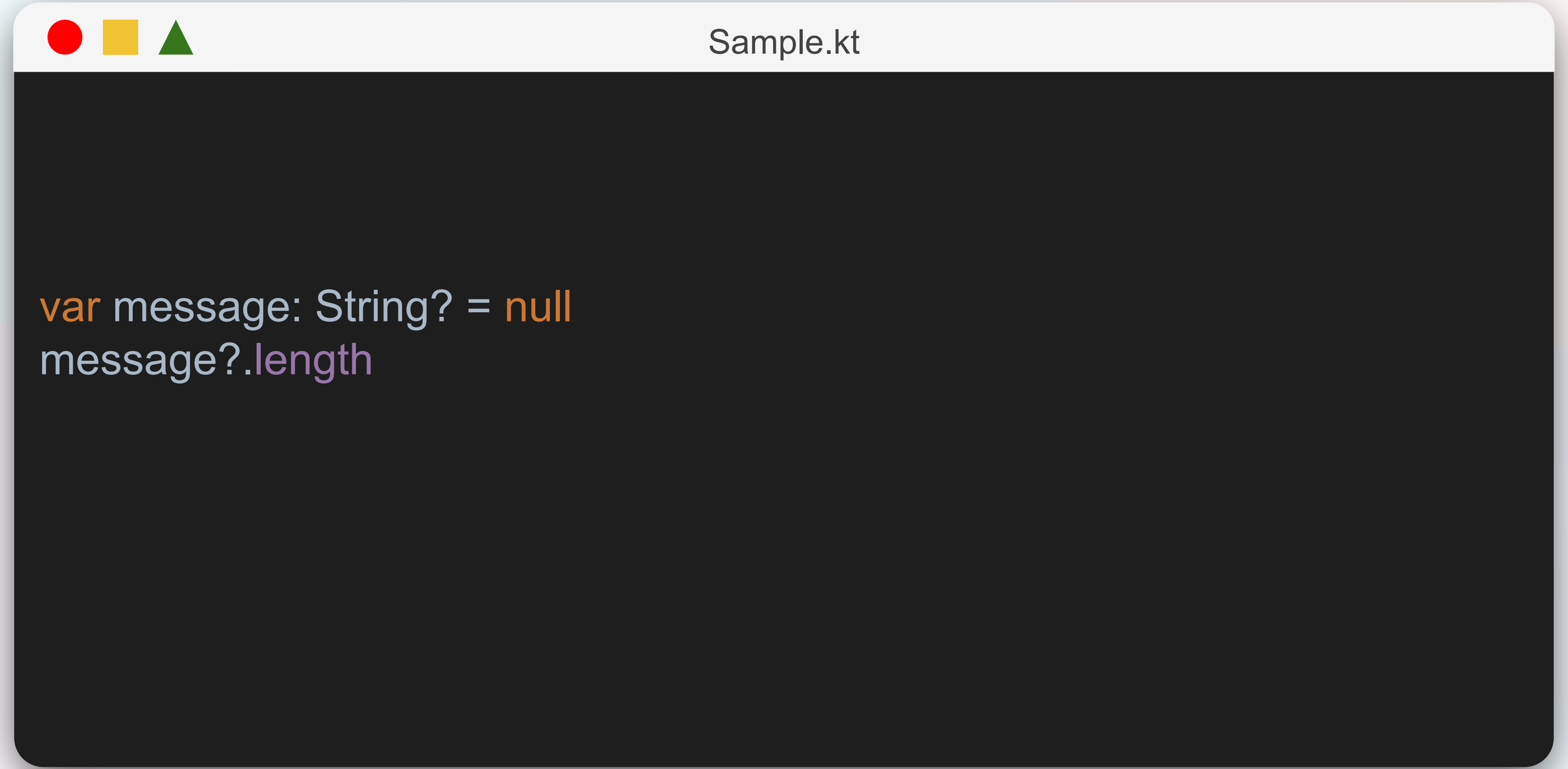
```kotlin
var message: String? = null
message.length
```

Null Safety


Sample.kt

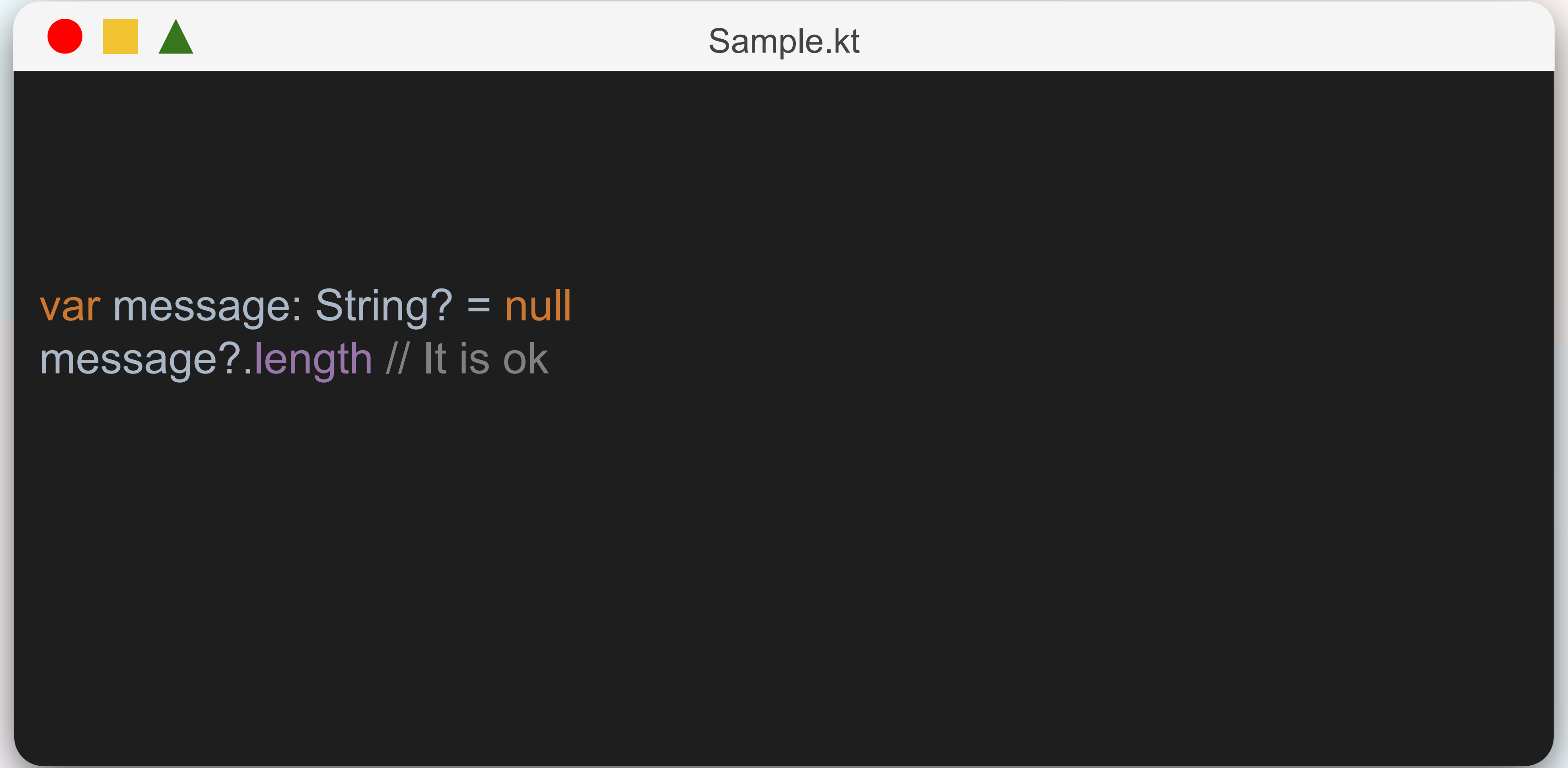
```kotlin
var message: String? = null
message.length // Error: Only safe or non-null asserted calls are
                          allowed on a nullable receiver of type String?
```

Null Safety


Sample.kt

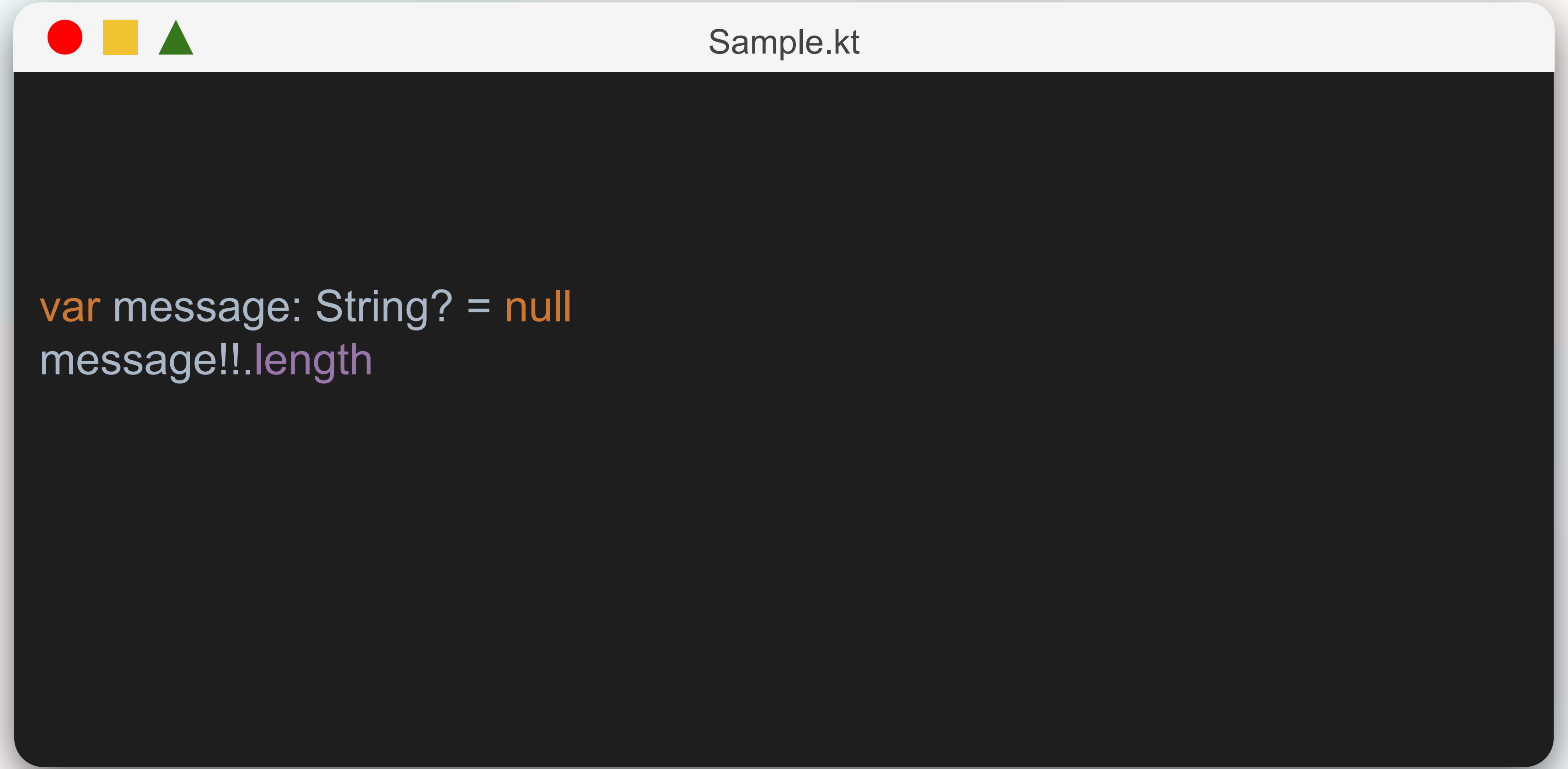
```kotlin
var message: String? = null
message?.length
```

Null Safety



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

Null Safety


Sample.kt

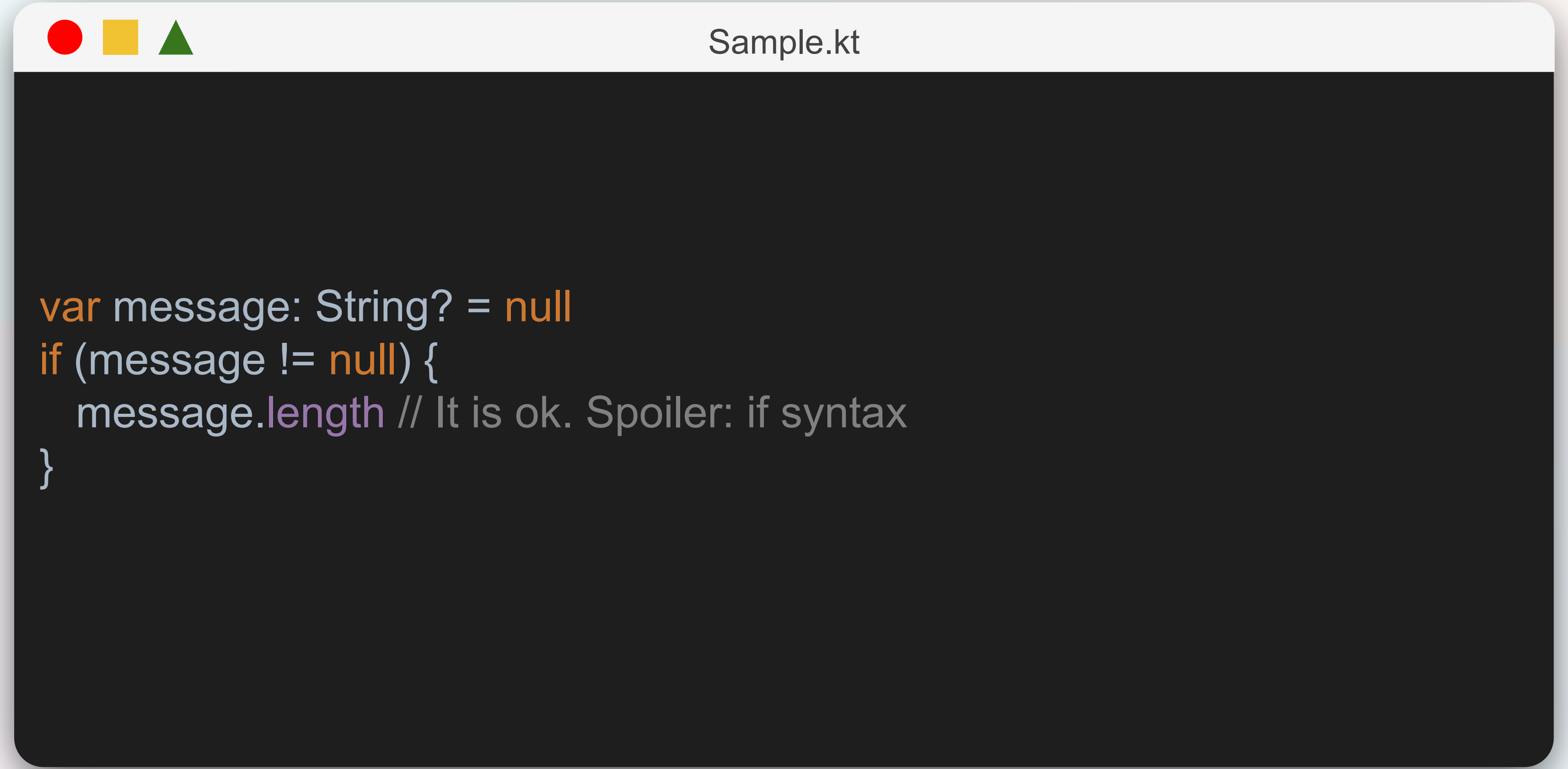
```kotlin
var message: String? = null
message!!.length
```

Null Safety

Sample.kt

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

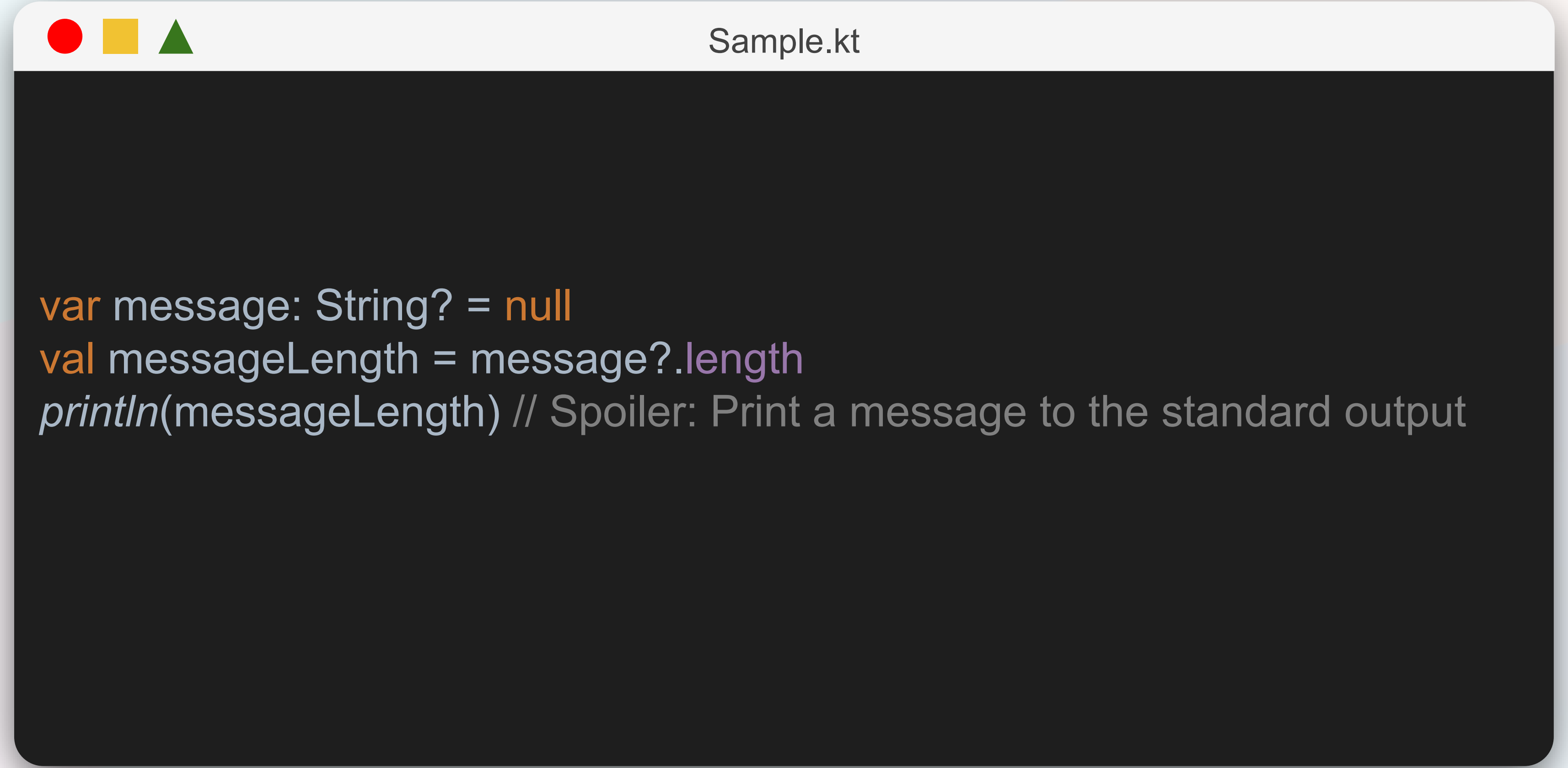
Null Safety



Sample.kt

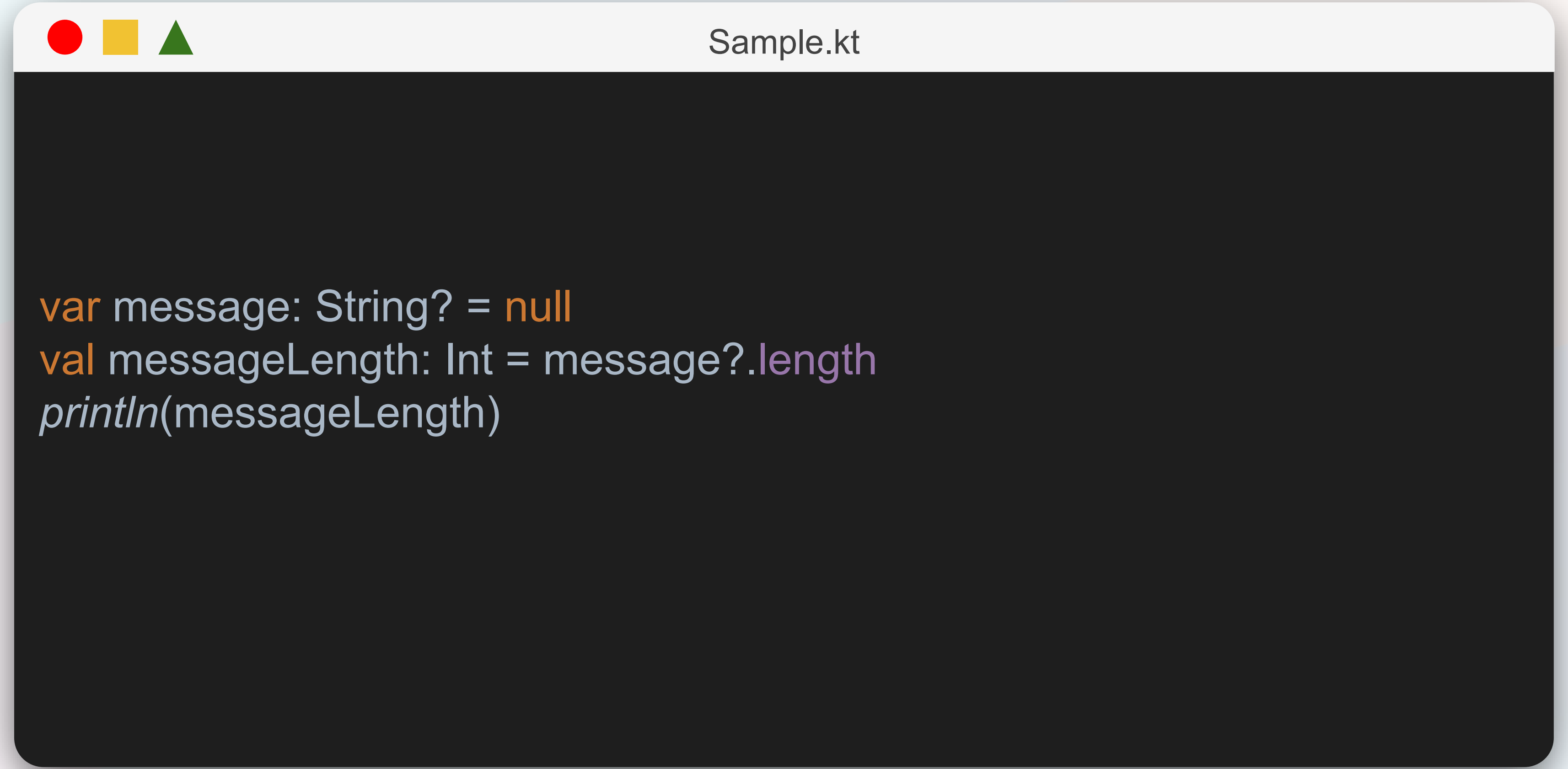
```
var message: String? = null
if (message != null) {
   message.length // It is ok. Spoiler: if syntax
}
```

Null Safety


Sample.kt

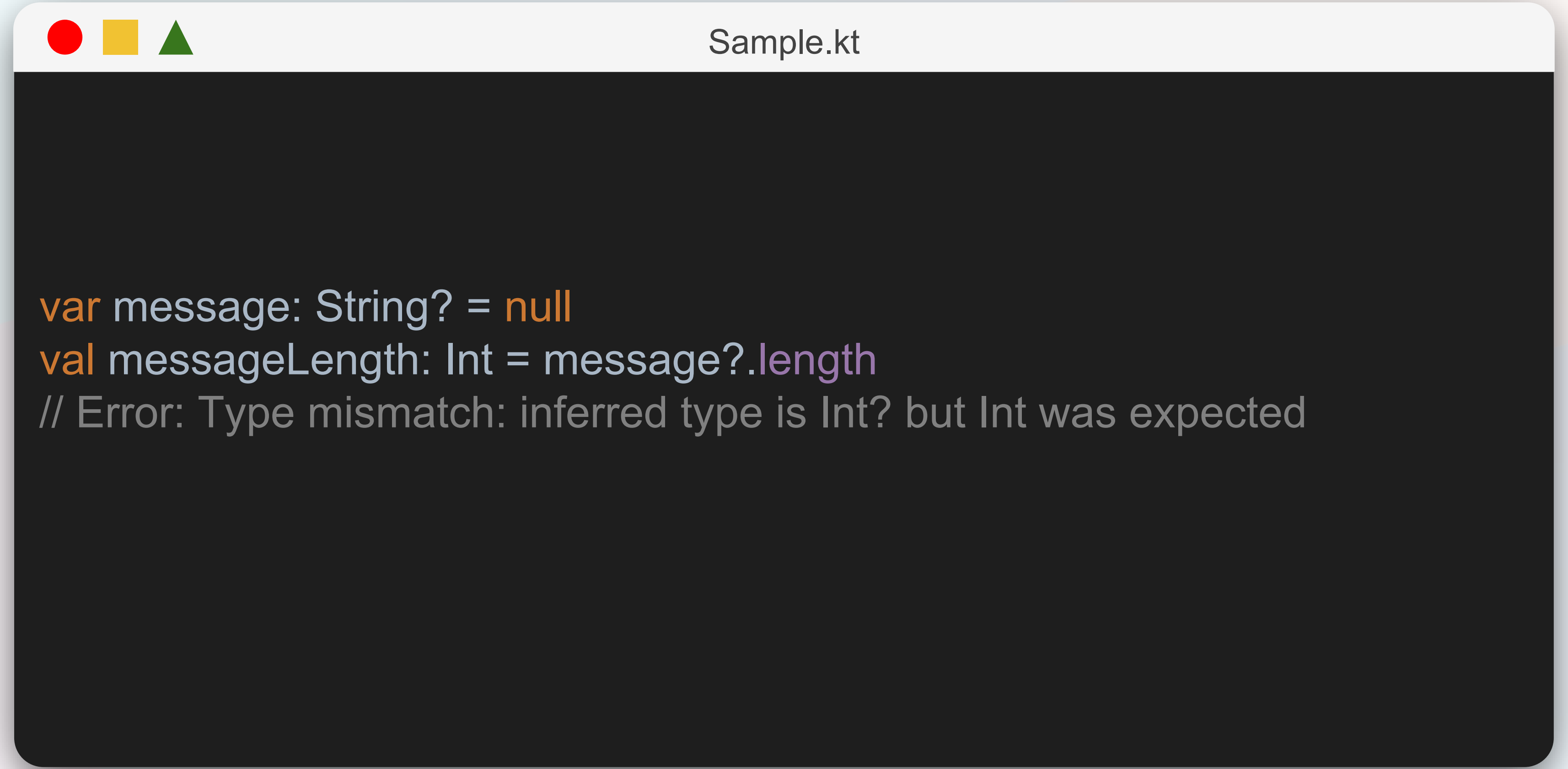
```kotlin
var message: String? = null
val messageLength = message?.length
println(messageLength) // Spoiler: Print a message to the standard output
```

Null Safety



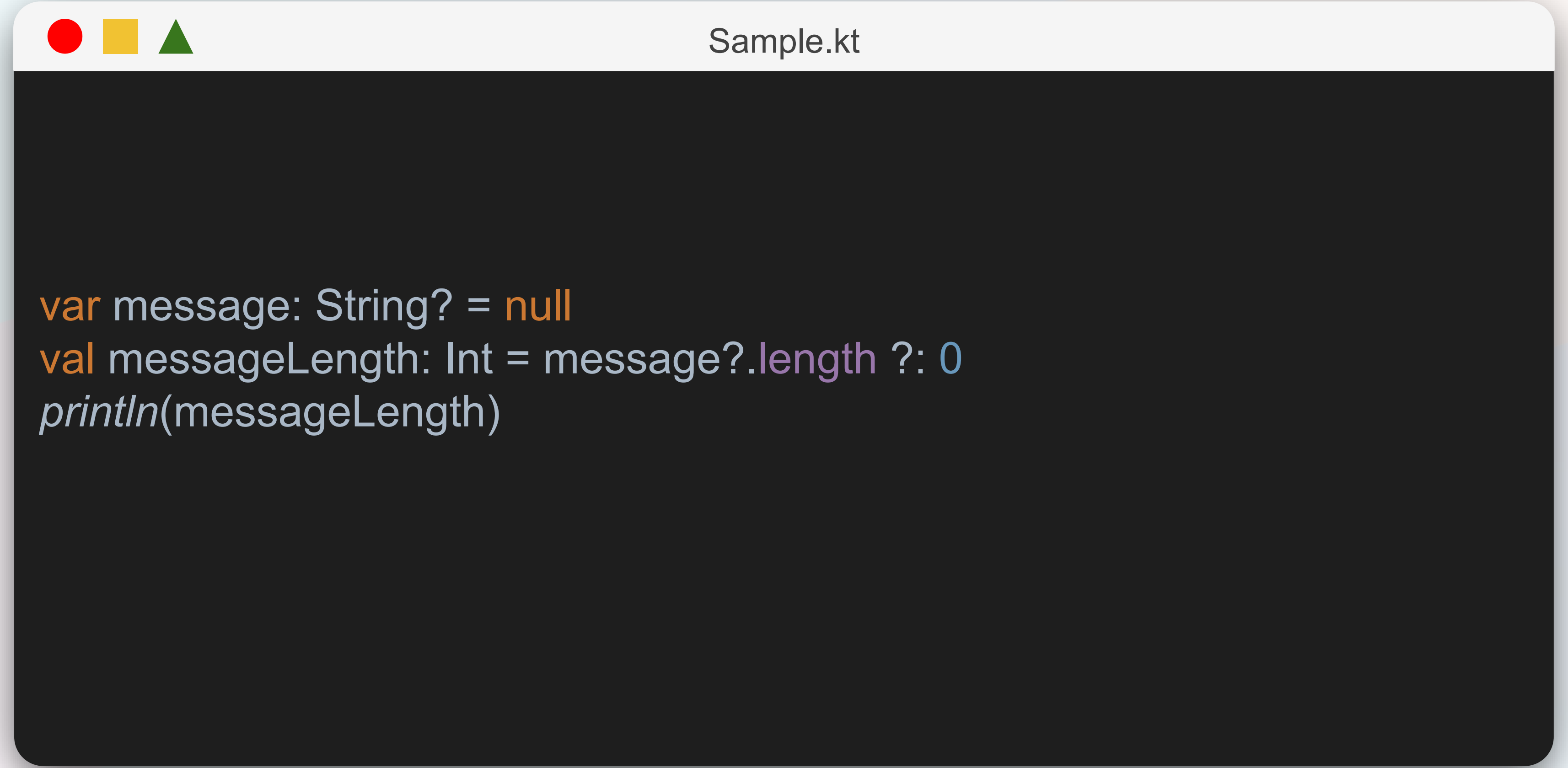
```kotlin
var message: String? = null
val messageLength: Int = message?.length
println(messageLength)
```

Null Safety


Sample.kt

```kotlin
var message: String? = null
val messageLength: Int = message?.length
// Error: Type mismatch: inferred type is Int? but Int was expected
```

Null Safety



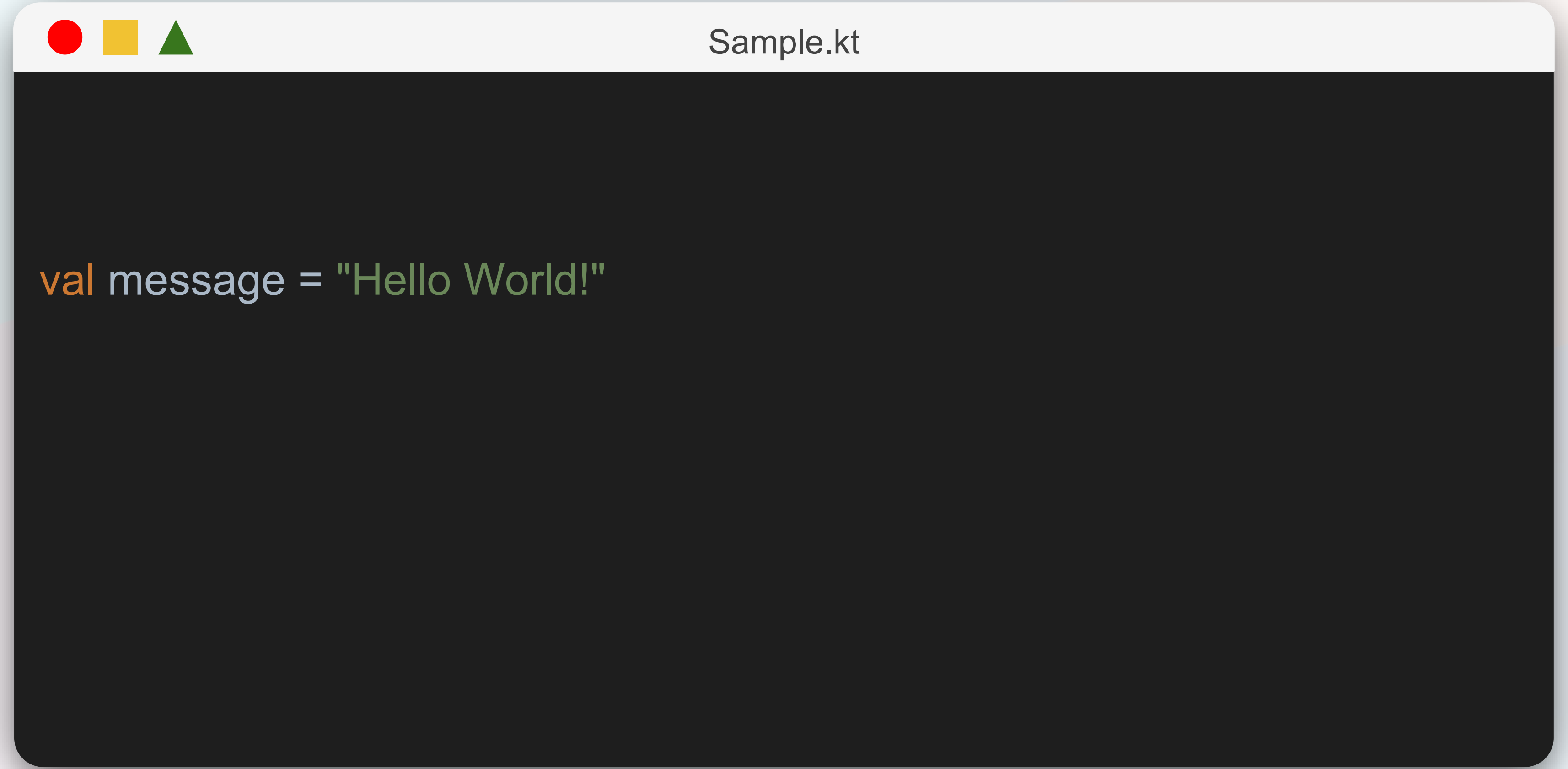
Sample.kt

```kotlin
var message: String? = null
val messageLength: Int = message?.length ?: 0
println(messageLength)
```

Basics

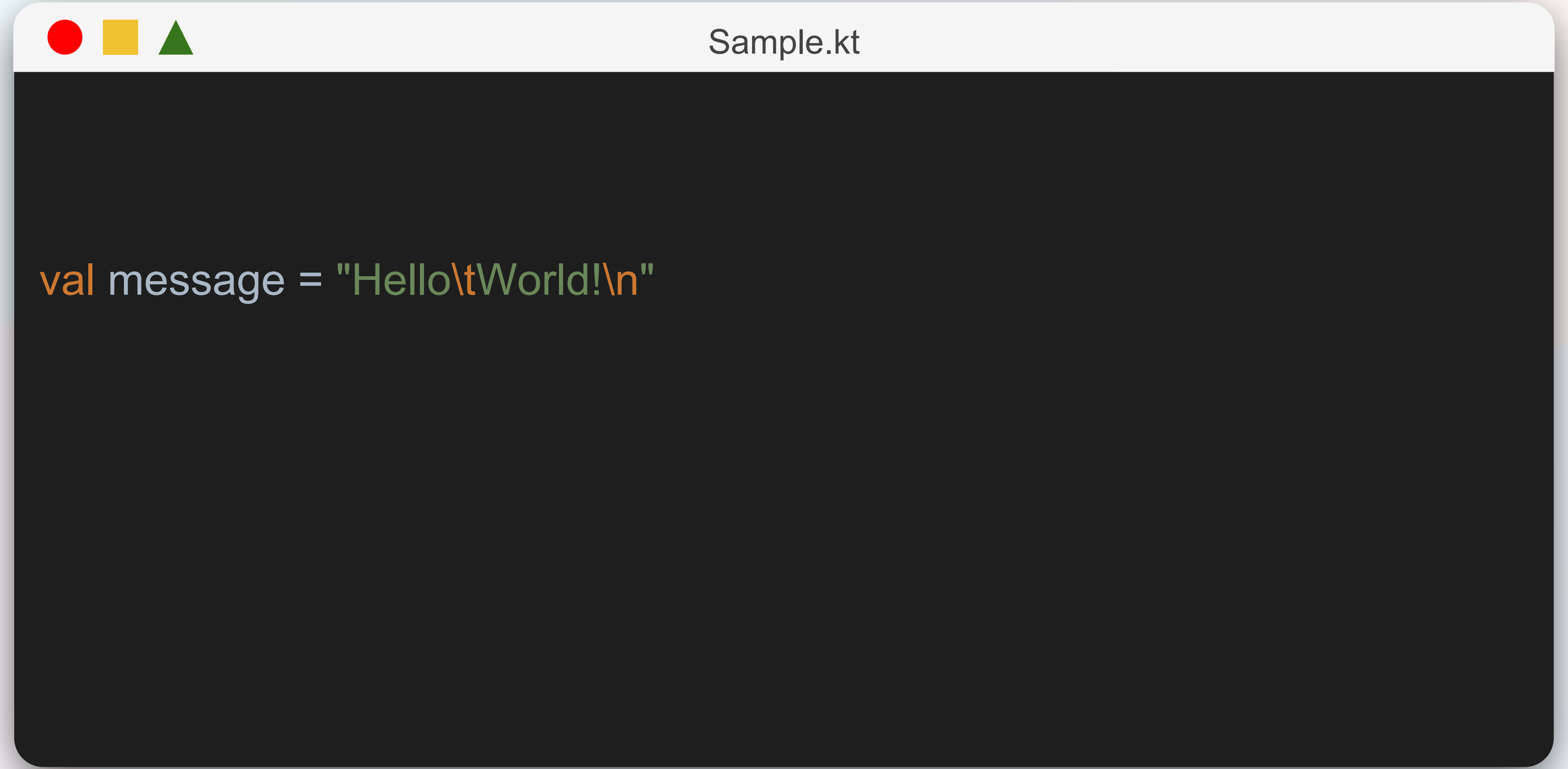
Strings

Strings


Sample.kt

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

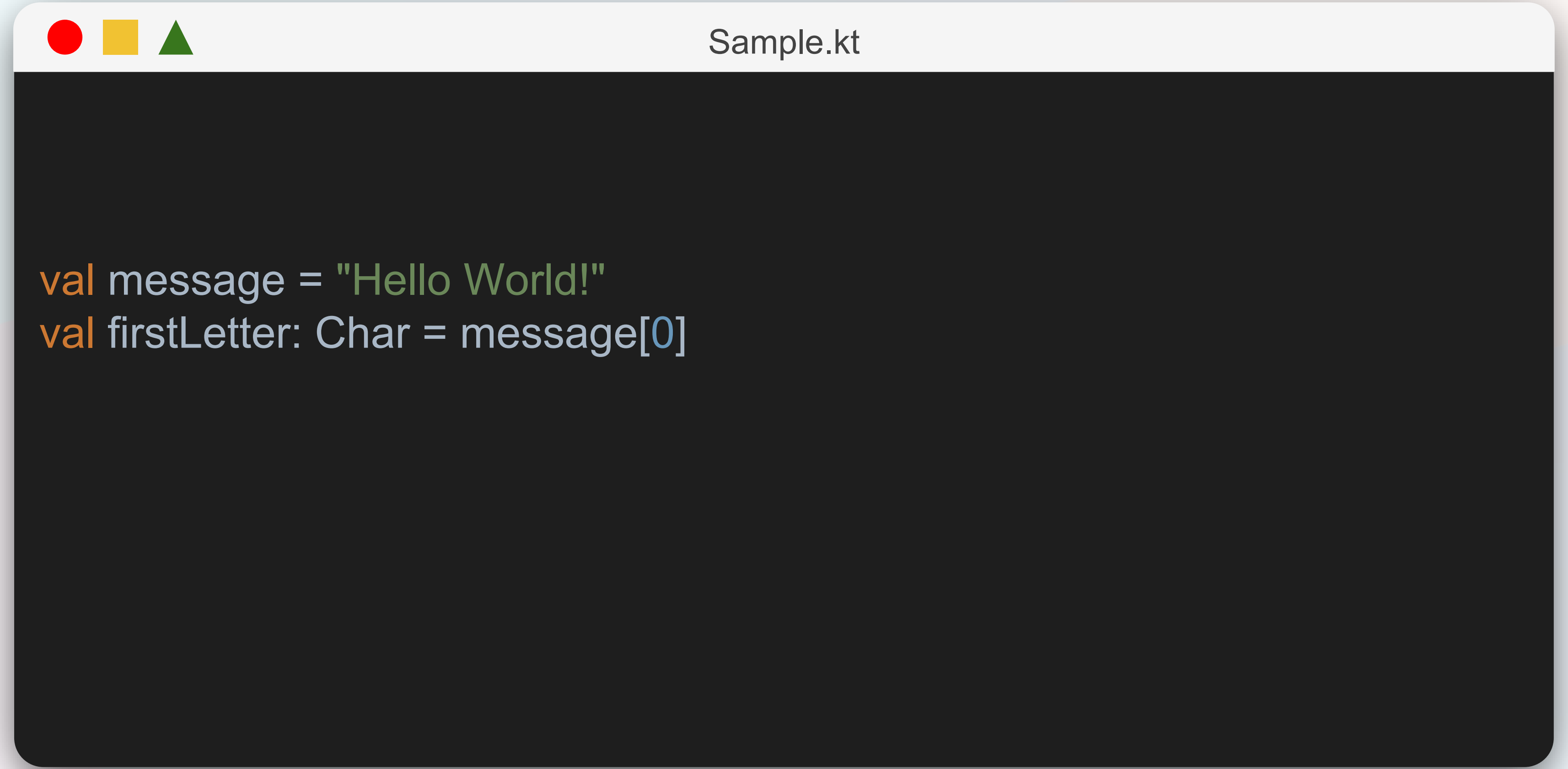
Strings



Sample.kt

```
val message = "Hello\tWorld!\n"
```

Strings


Sample.kt

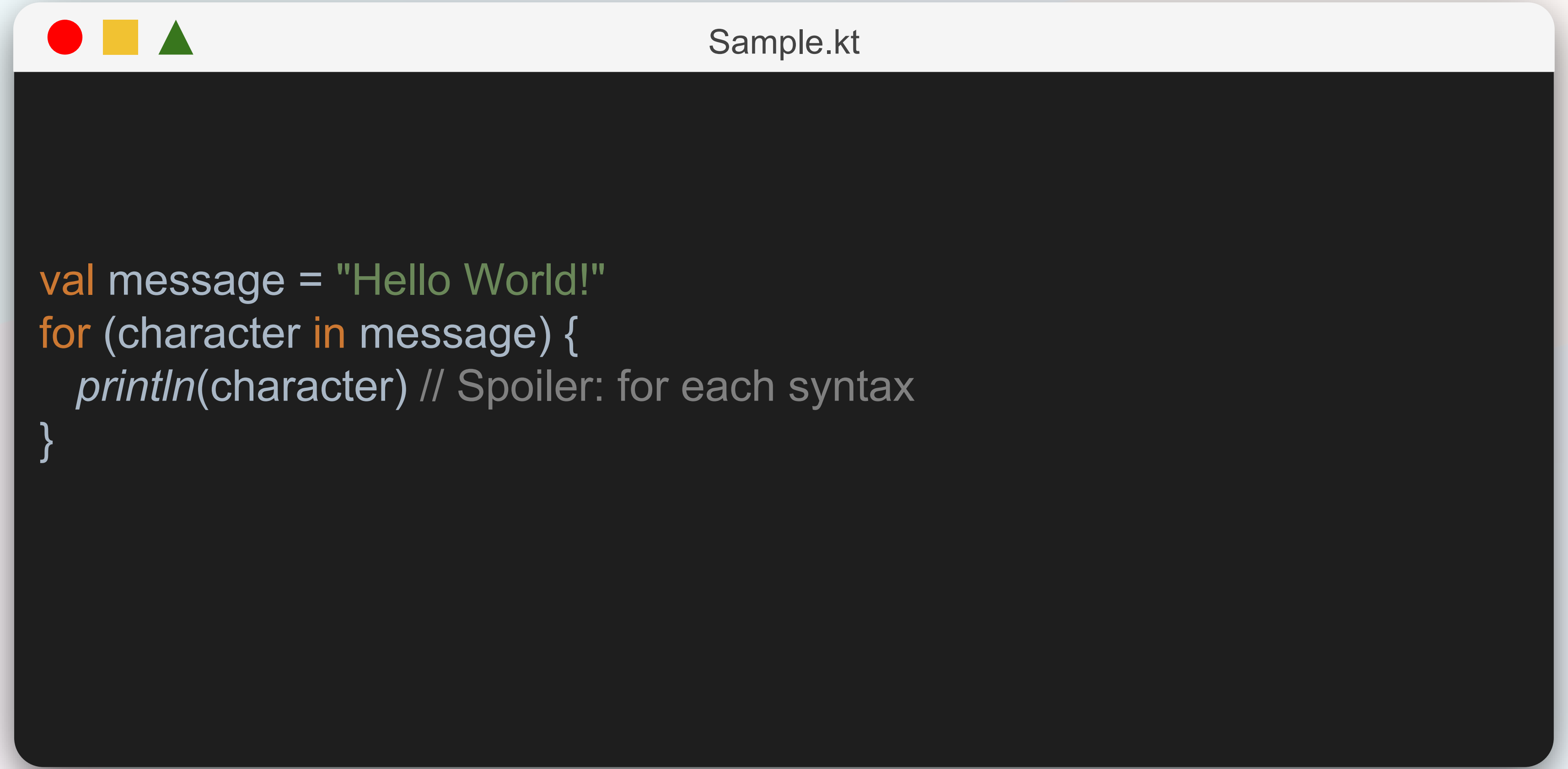
```
val message = "Hello World!"
val firstLetter: Char = message[0]
```

Strings

Sample.kt

```kotlin
val message = "Hello World!"
val firstLetter = message[0]
```

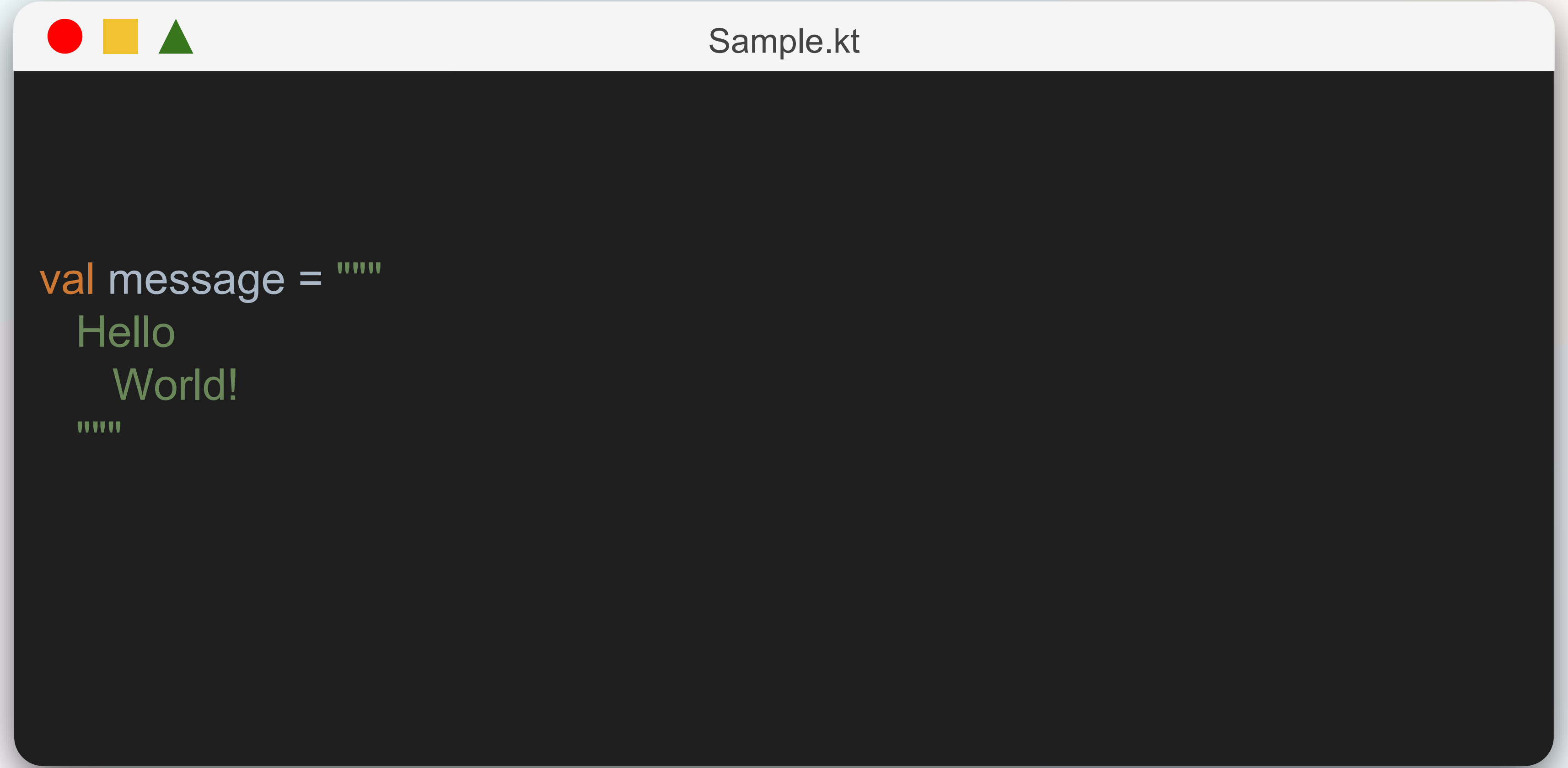
Strings



Sample.kt

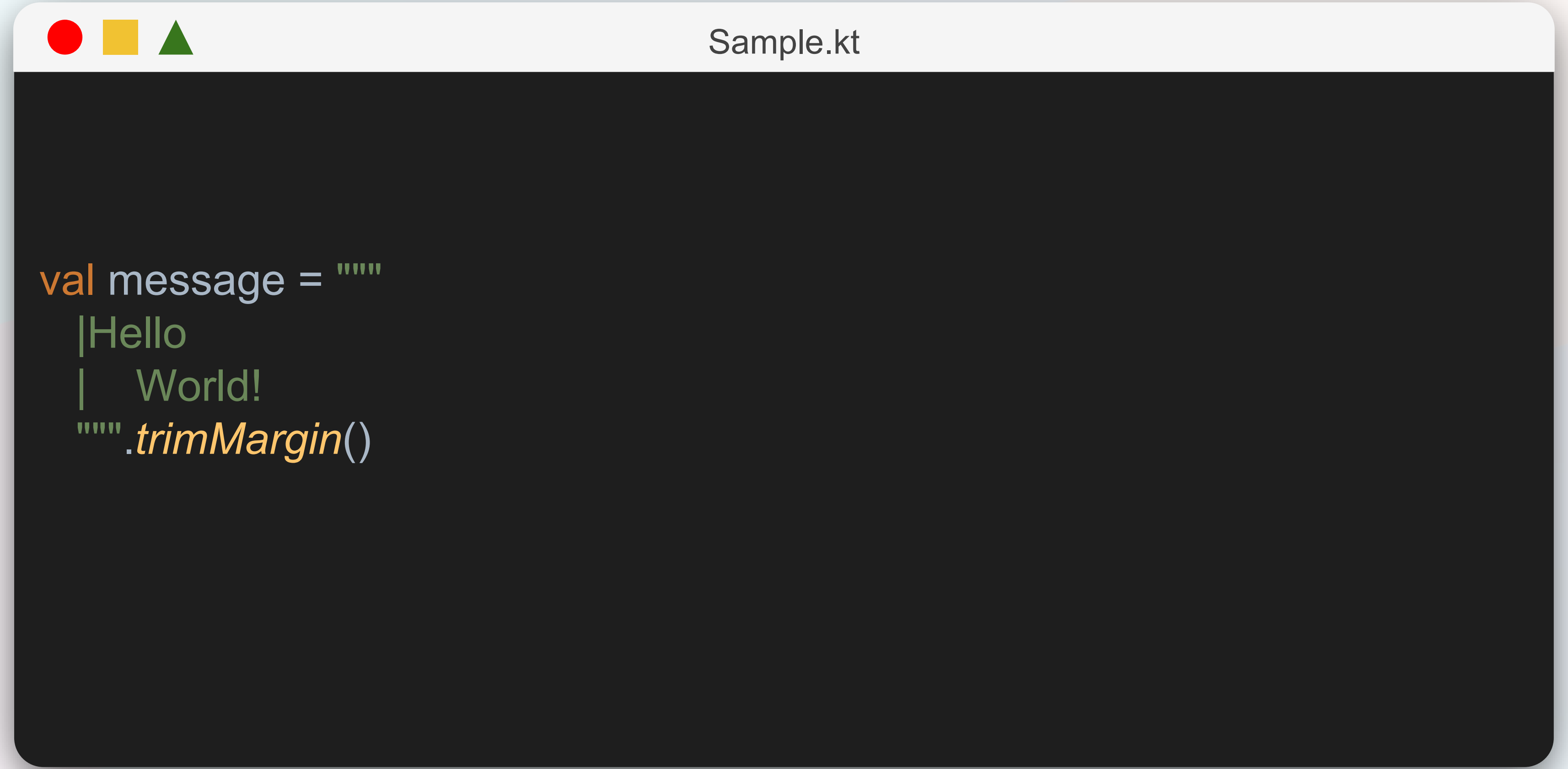
```kotlin
val message = "Hello World!"
for (character in message) {
    println(character) // Spoiler: for each syntax
}
```

Strings


Sample.kt

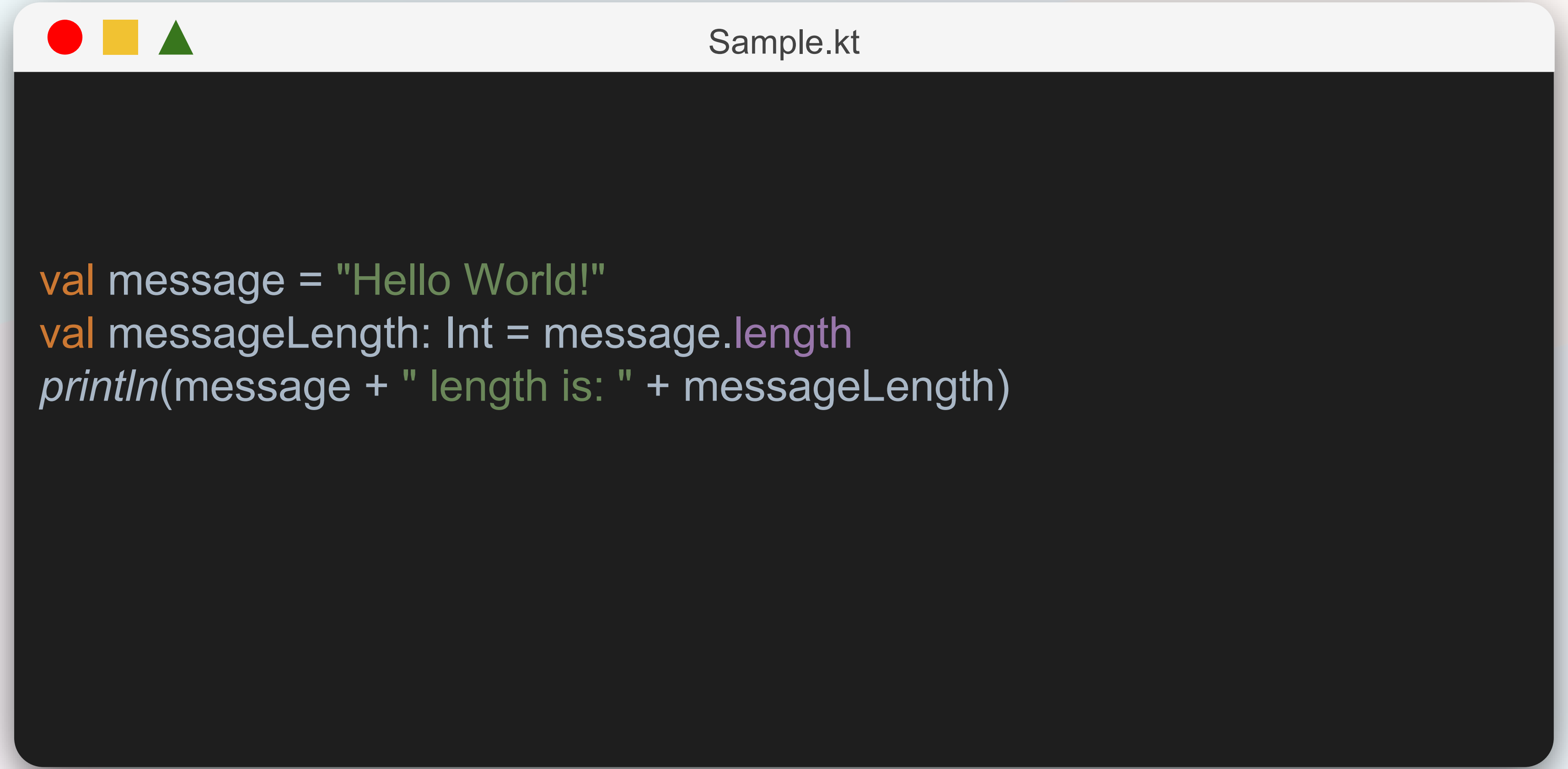
```
val message = """
   Hello
      World!
   """
```

Strings



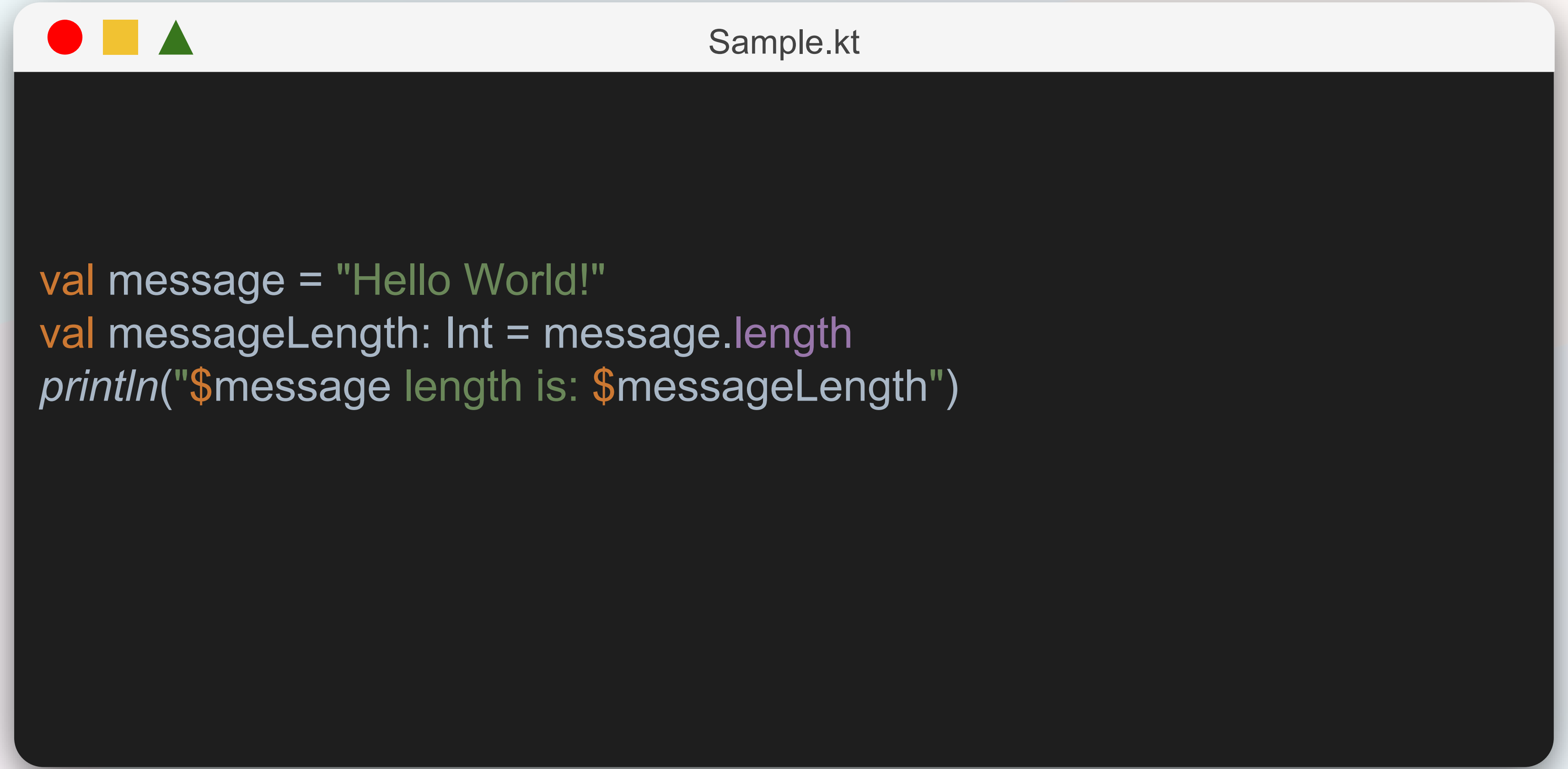
```kotlin
val message = """
    |Hello
    |    World!
    """.trimMargin()
```

Strings



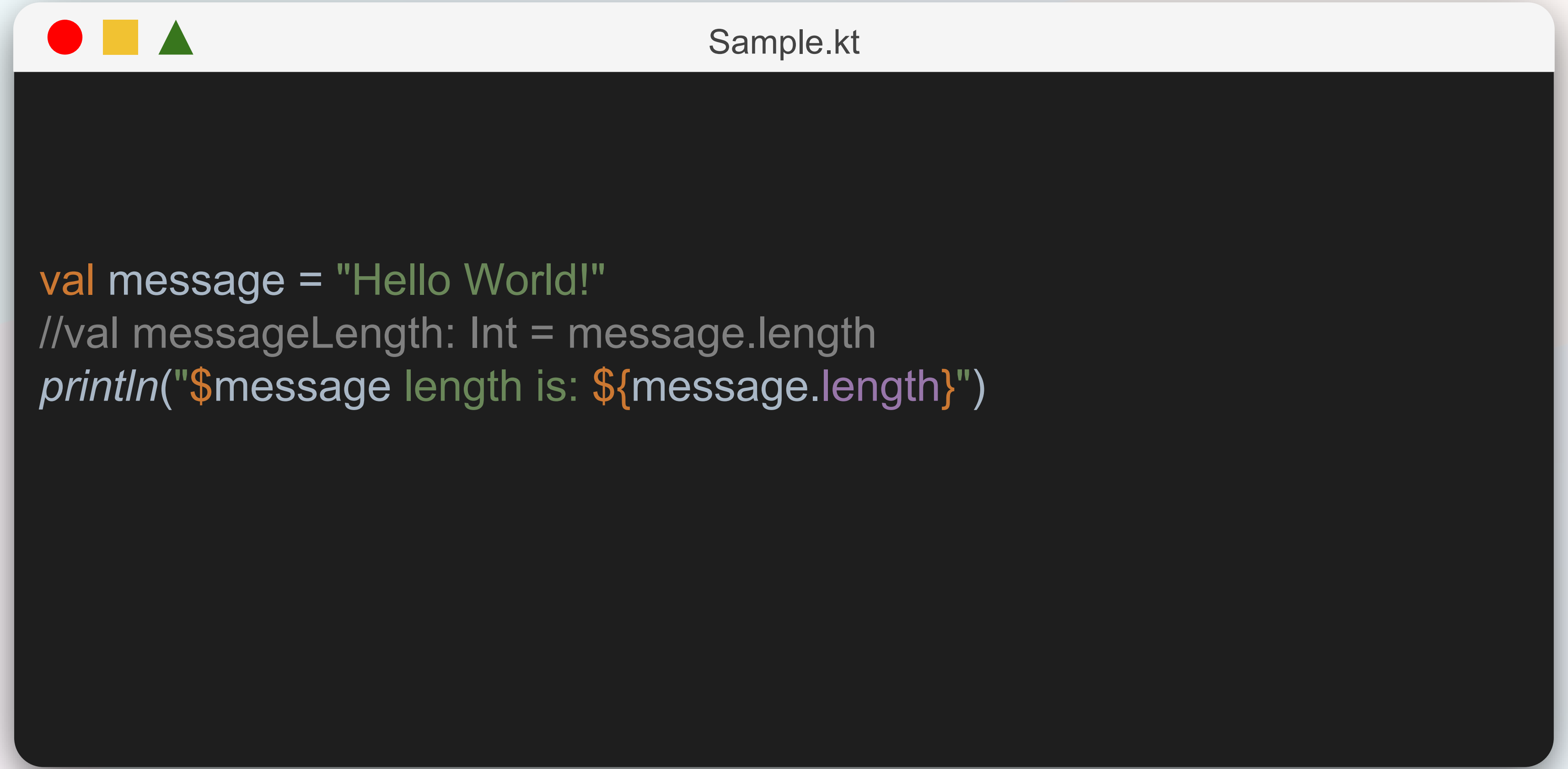
```kotlin
val message = "Hello World!"
val messageLength: Int = message.length
println(message + " length is: " + messageLength)
```

Strings



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

Strings



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

Basics

Control Flow

Basics

Control Flow

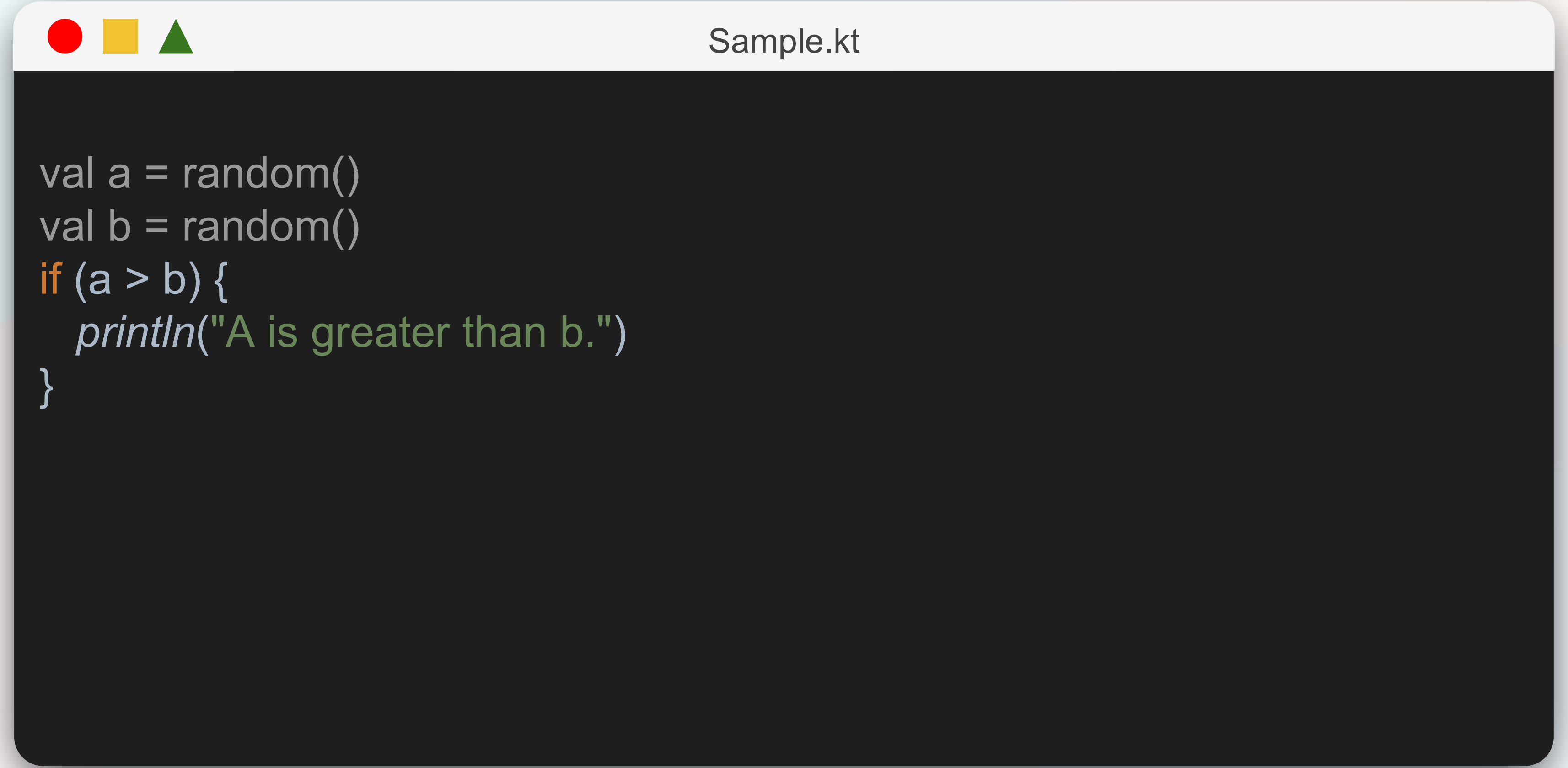
IF - ELSE

Control Flow - If else

Sample.kt

```kotlin
val a = random()
val b = random()
```

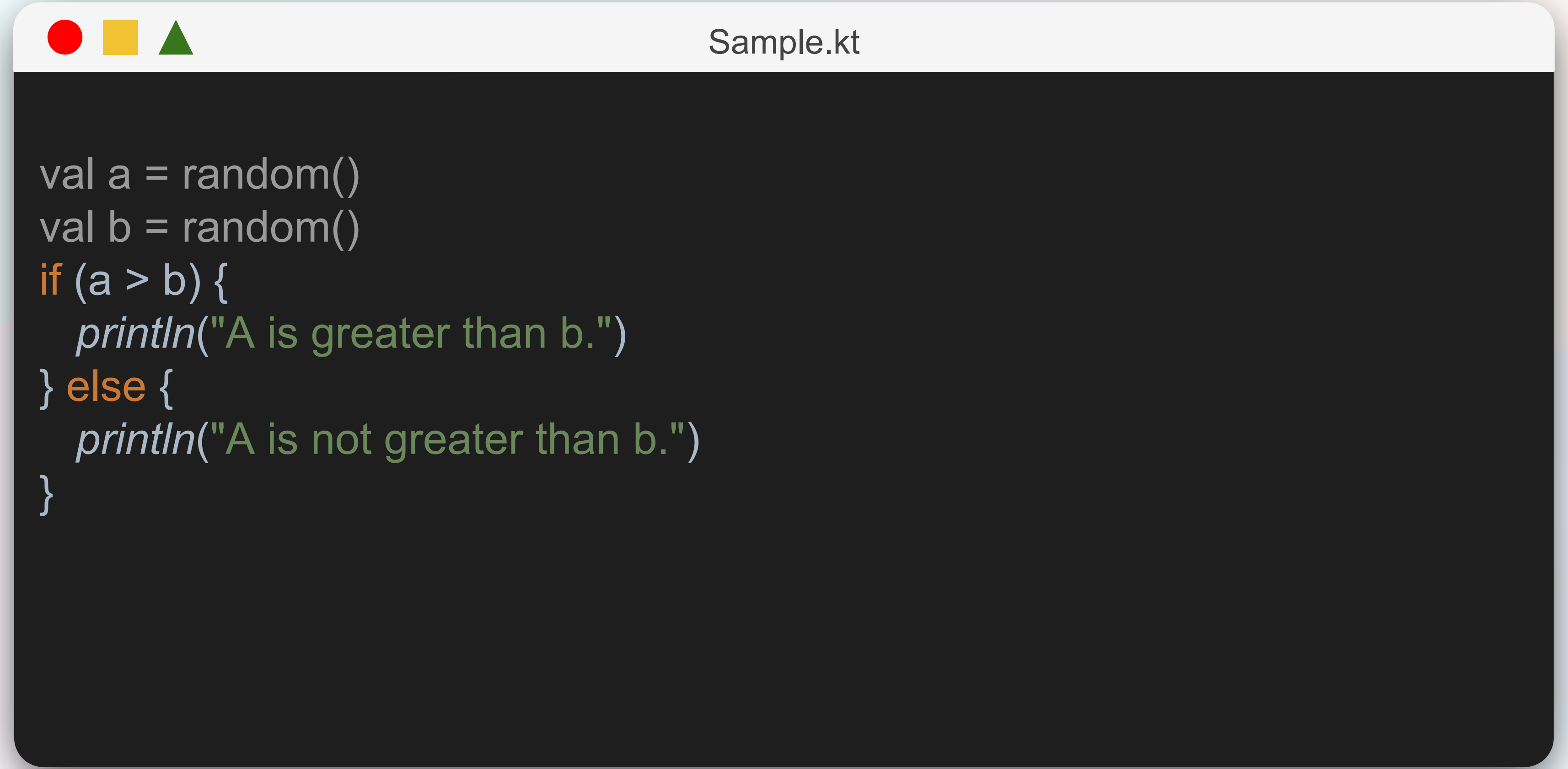
Control Flow - If else



Sample.kt

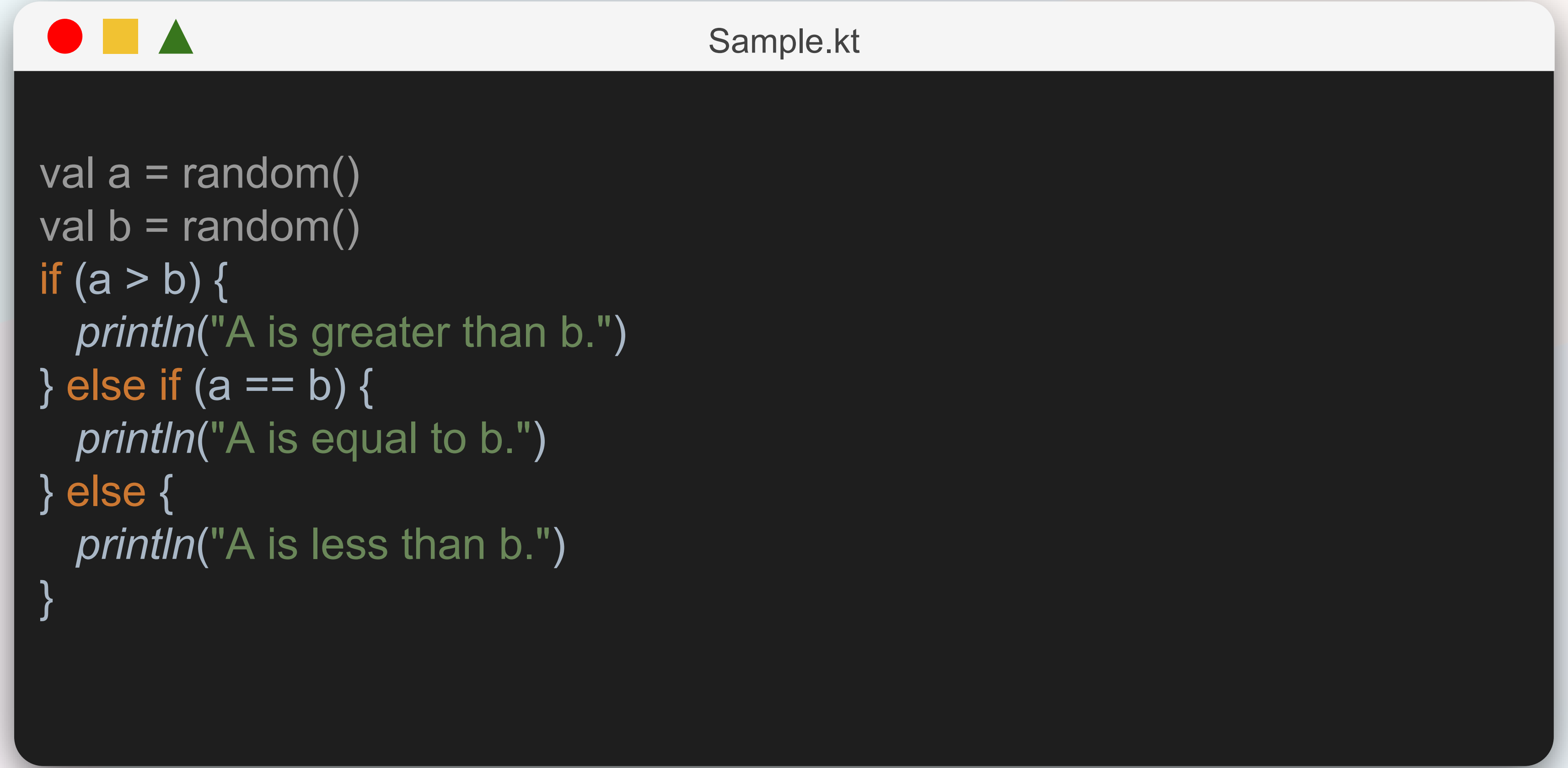
```kotlin
val a = random()
val b = random()
if (a > b) {
   println("A is greater than b.")
}
```

Control Flow - If else


Sample.kt

```
val a = random()
val b = random()
if (a > b) {
   println("A is greater than b.")
} else {
   println("A is not greater than b.")
}
```

Control Flow - If else


Sample.kt

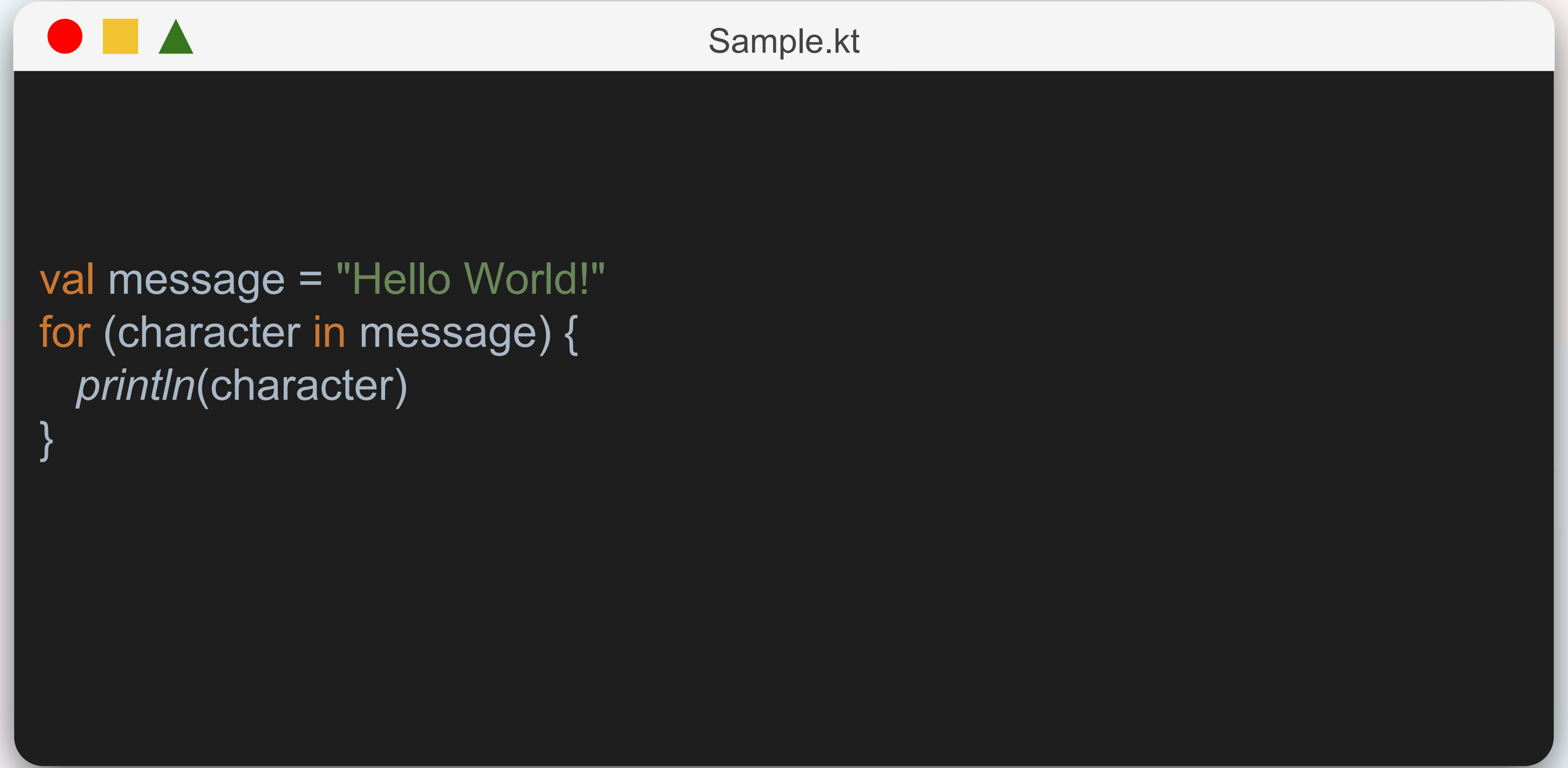
```
val a = random()
val b = random()
if (a > b) {
   println("A is greater than b.")
} else if (a == b) {
   println("A is equal to b.")
} else {
   println("A is less than b.")
}
```

Basics

Control Flow

For

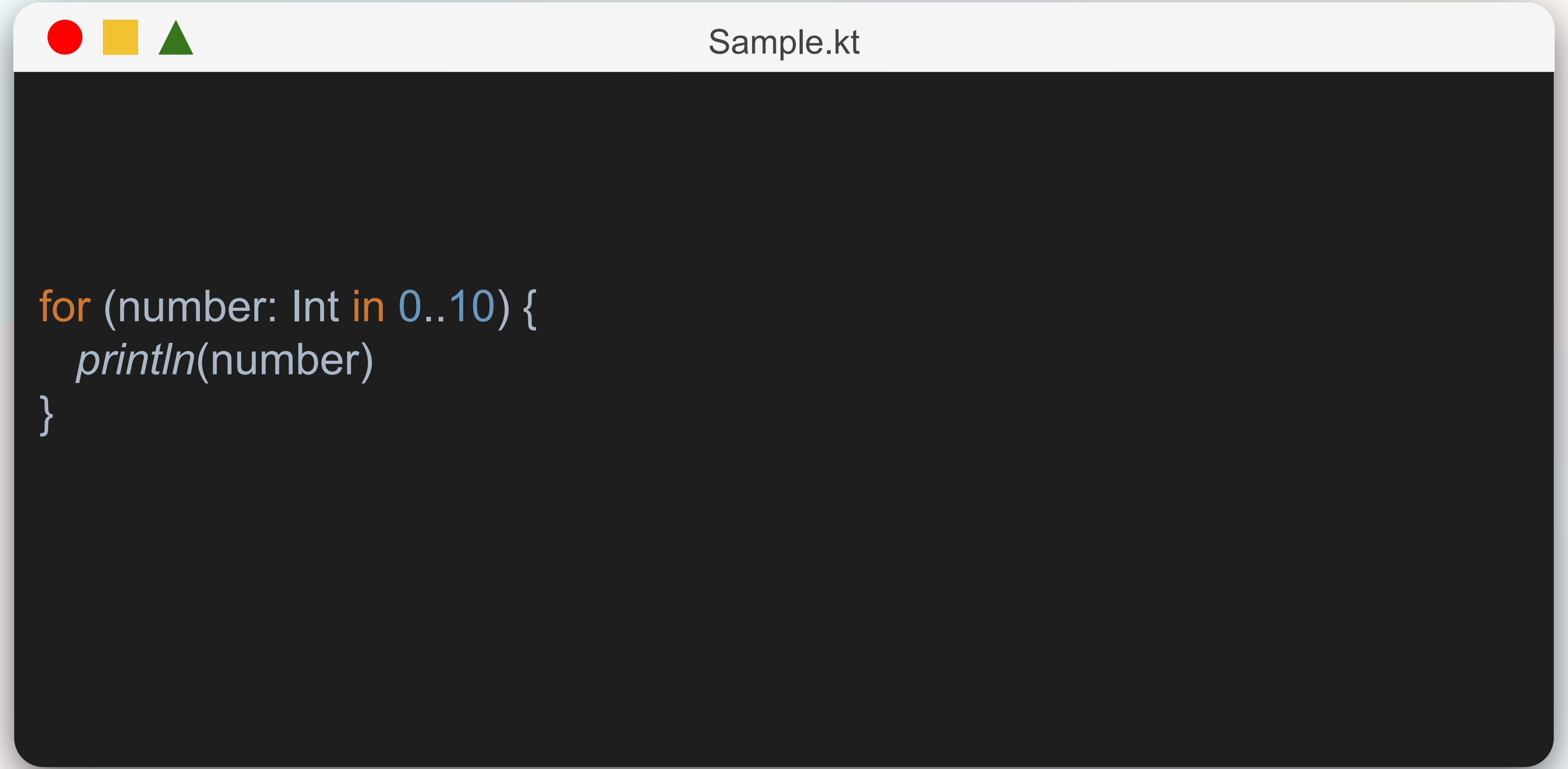
Control Flow - For



Sample.kt

```
val message = "Hello World!"
for (character in message) {
    println(character)
}
```

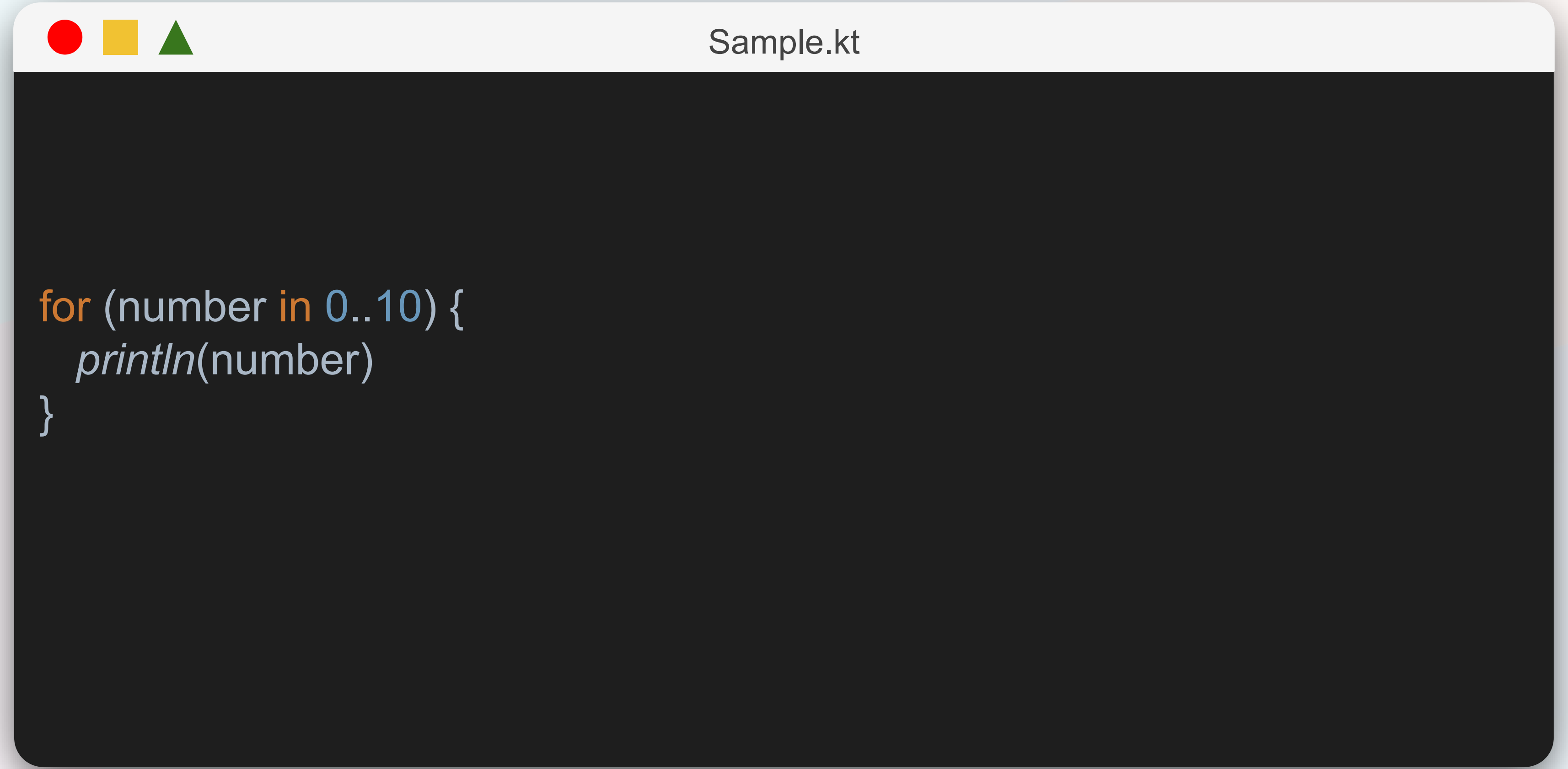
Control Flow - For



Sample.kt

```kotlin
for (number: Int in 0..10) {
   println(number)
}
```

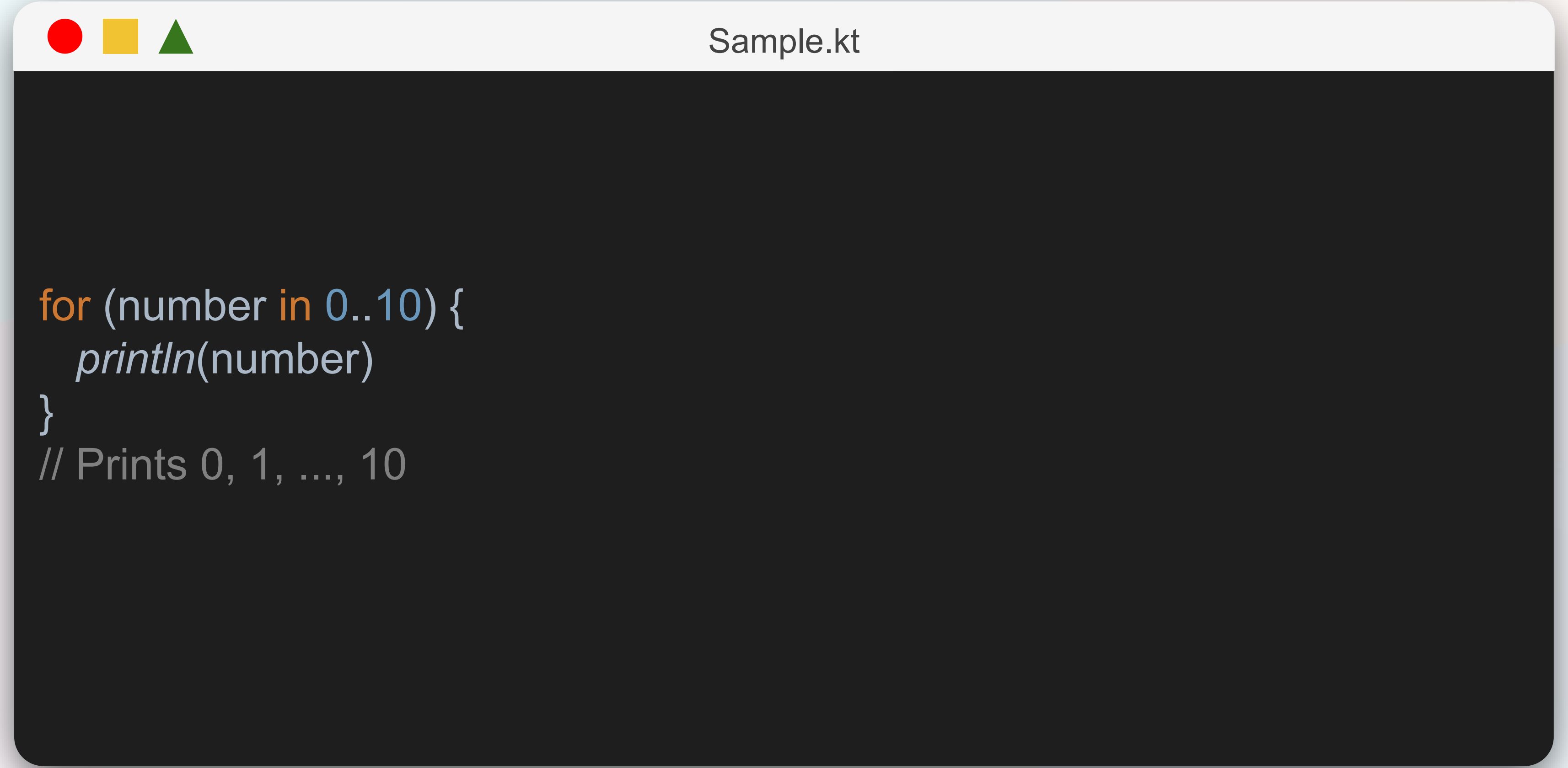
Control Flow - For



Sample.kt

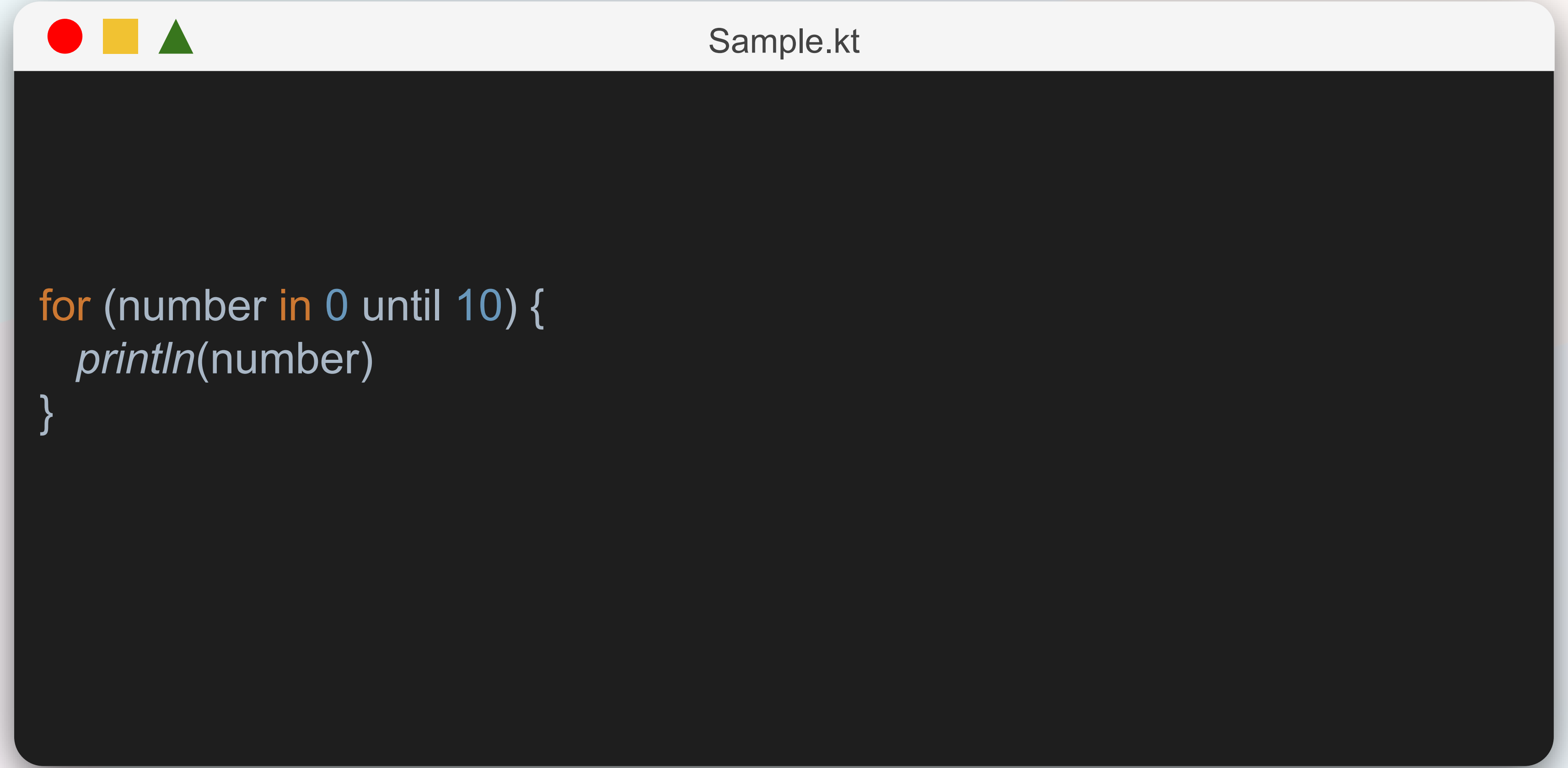
```kotlin
for (number in 0..10) {
   println(number)
}
```

Control Flow - For


Sample.kt

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

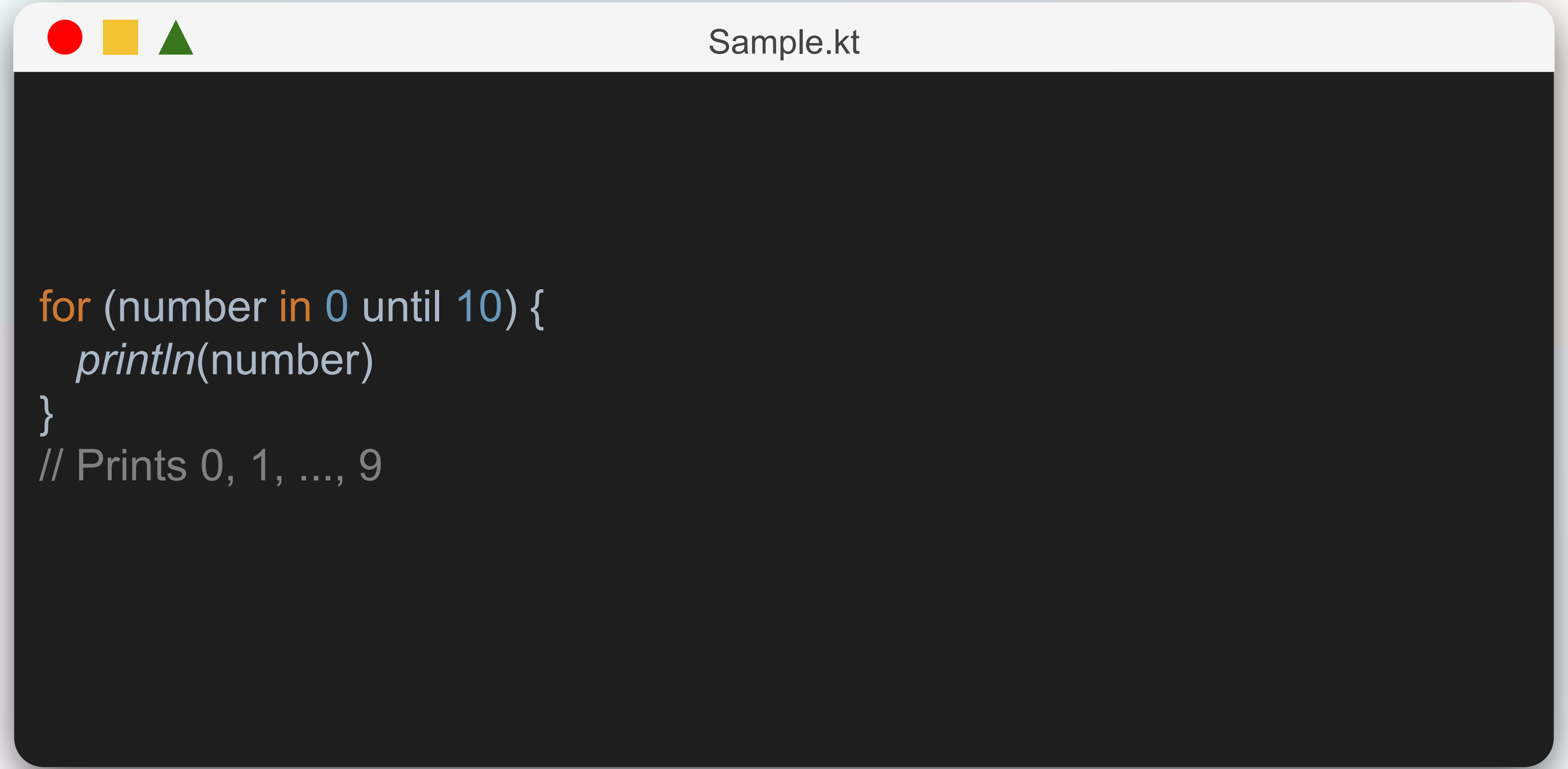
Control Flow - For



Sample.kt

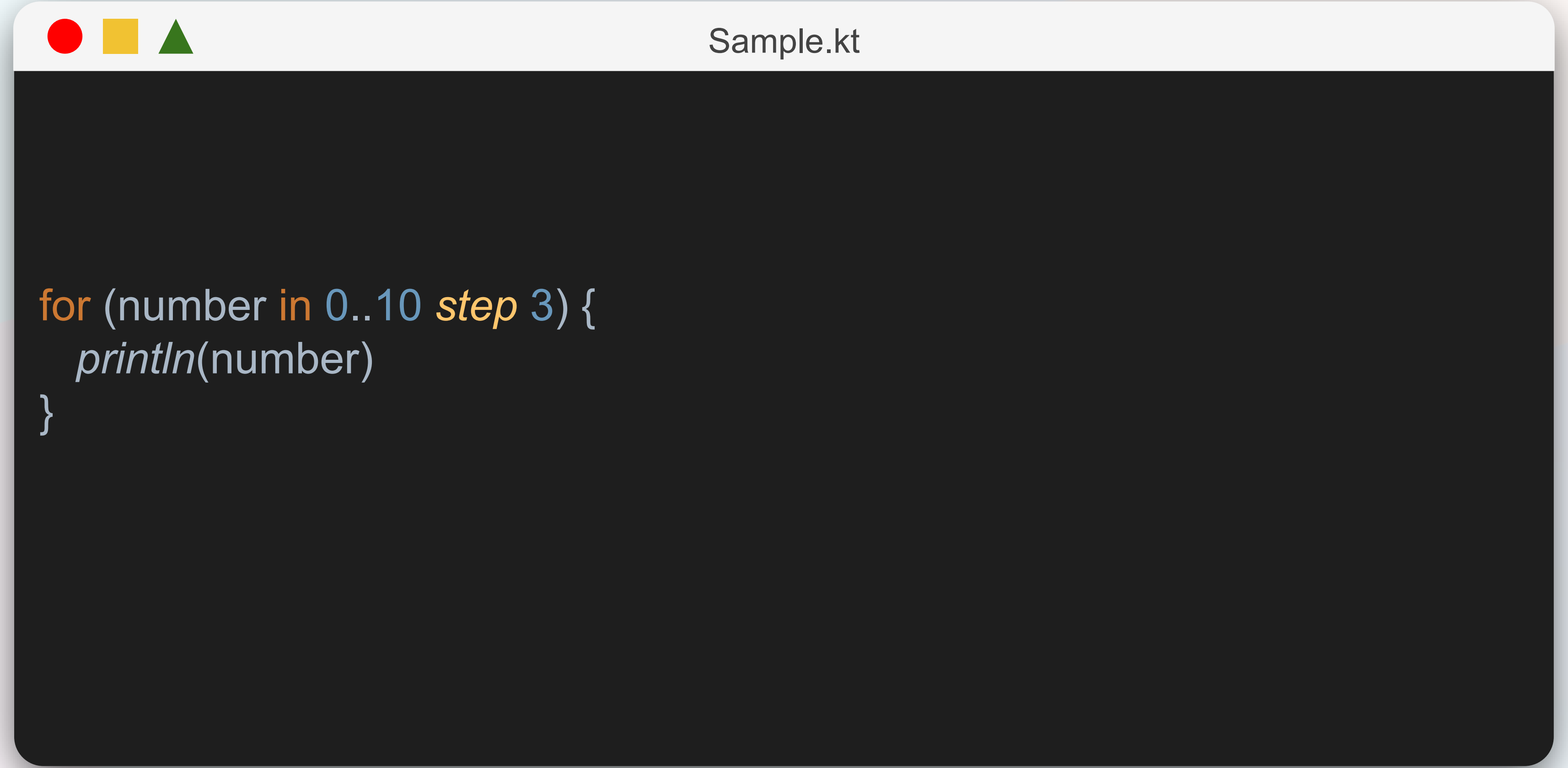
```kotlin
for (number in 0 until 10) {
    println(number)
}
```

Control Flow - For



Sample.kt

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



Sample.kt

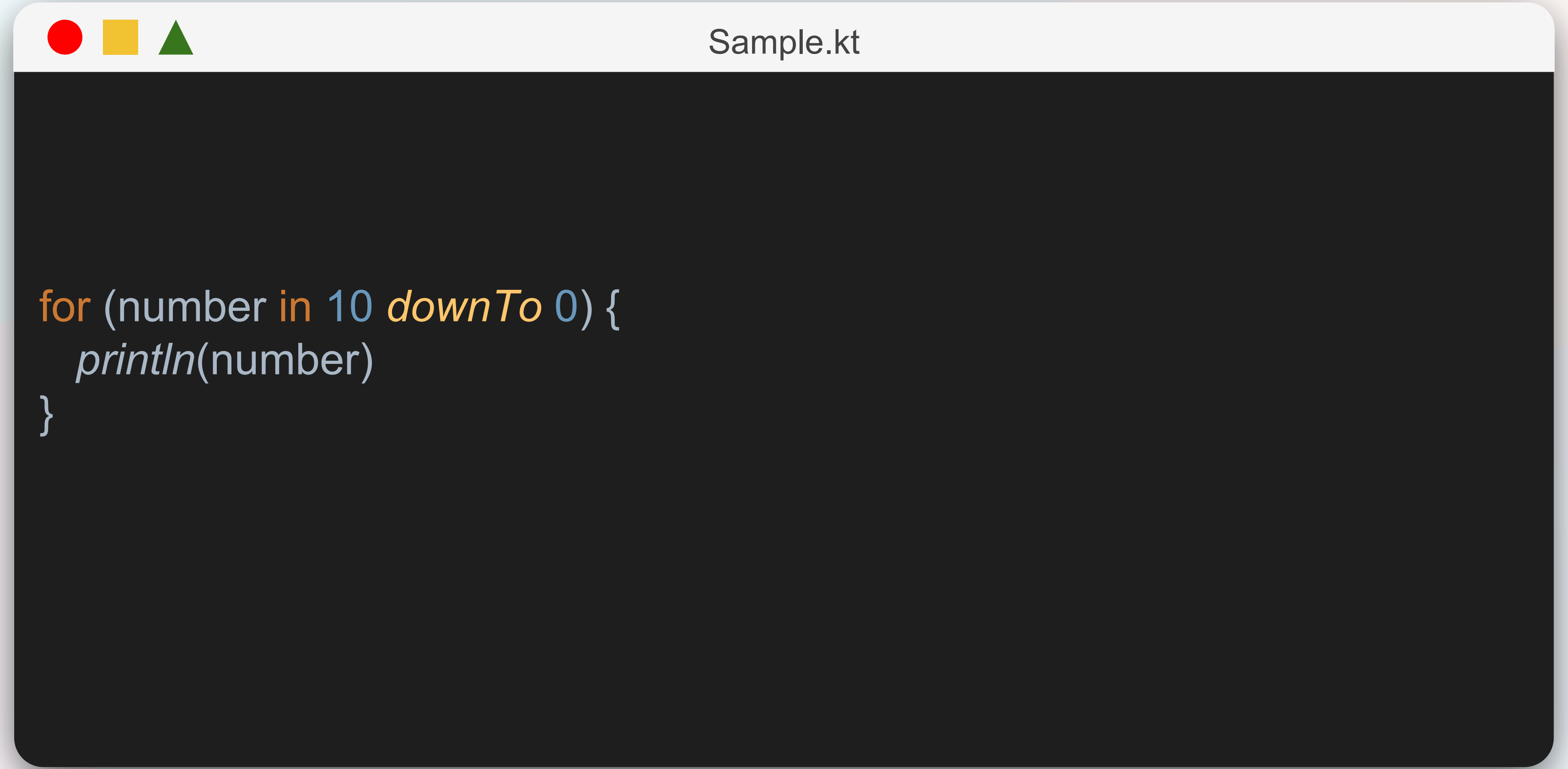
```
for (number in 0..10 step 3) {
   println(number)
}
```

Control Flow - For

Sample.kt

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

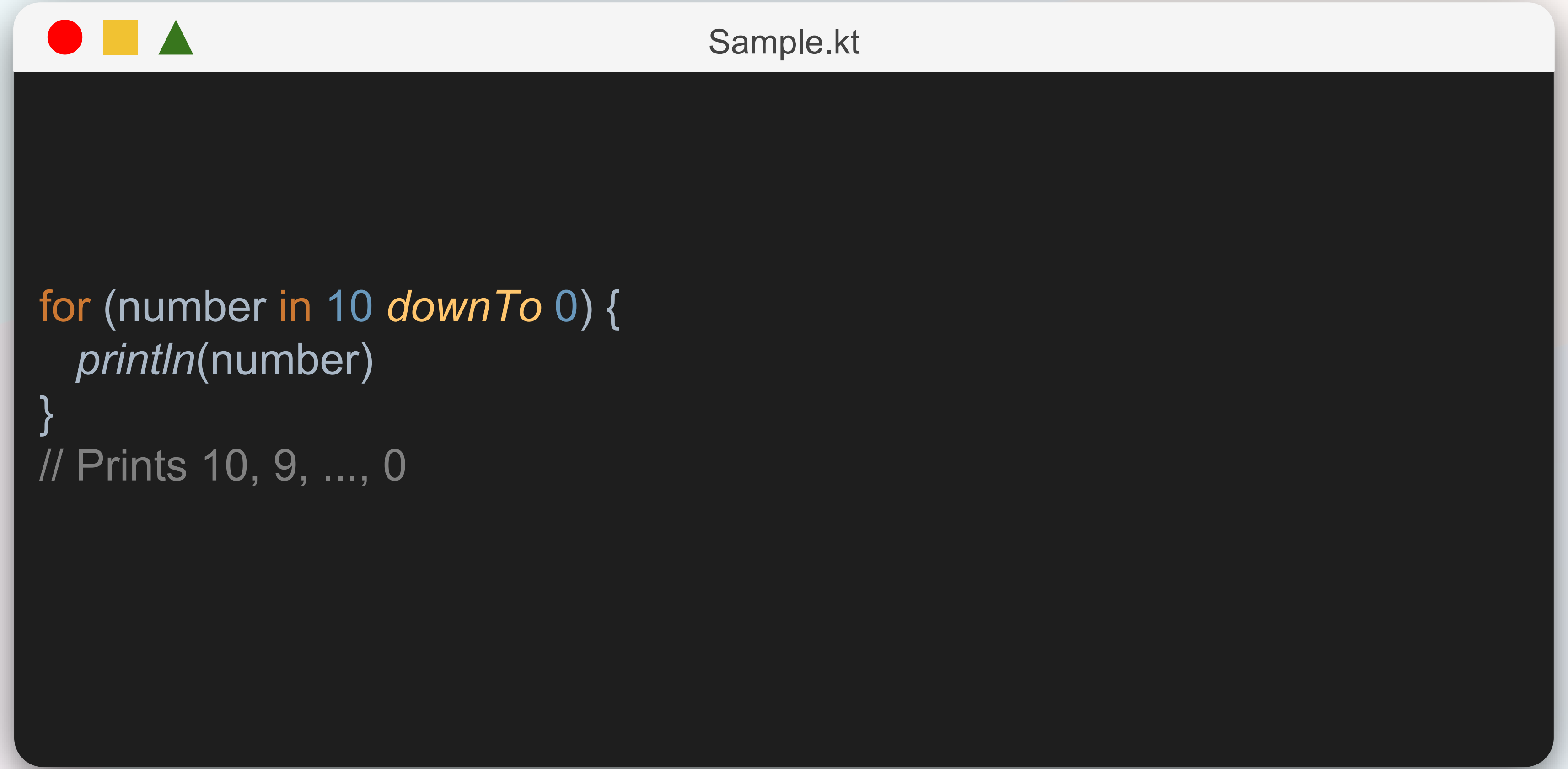
Control Flow - For



Sample.kt

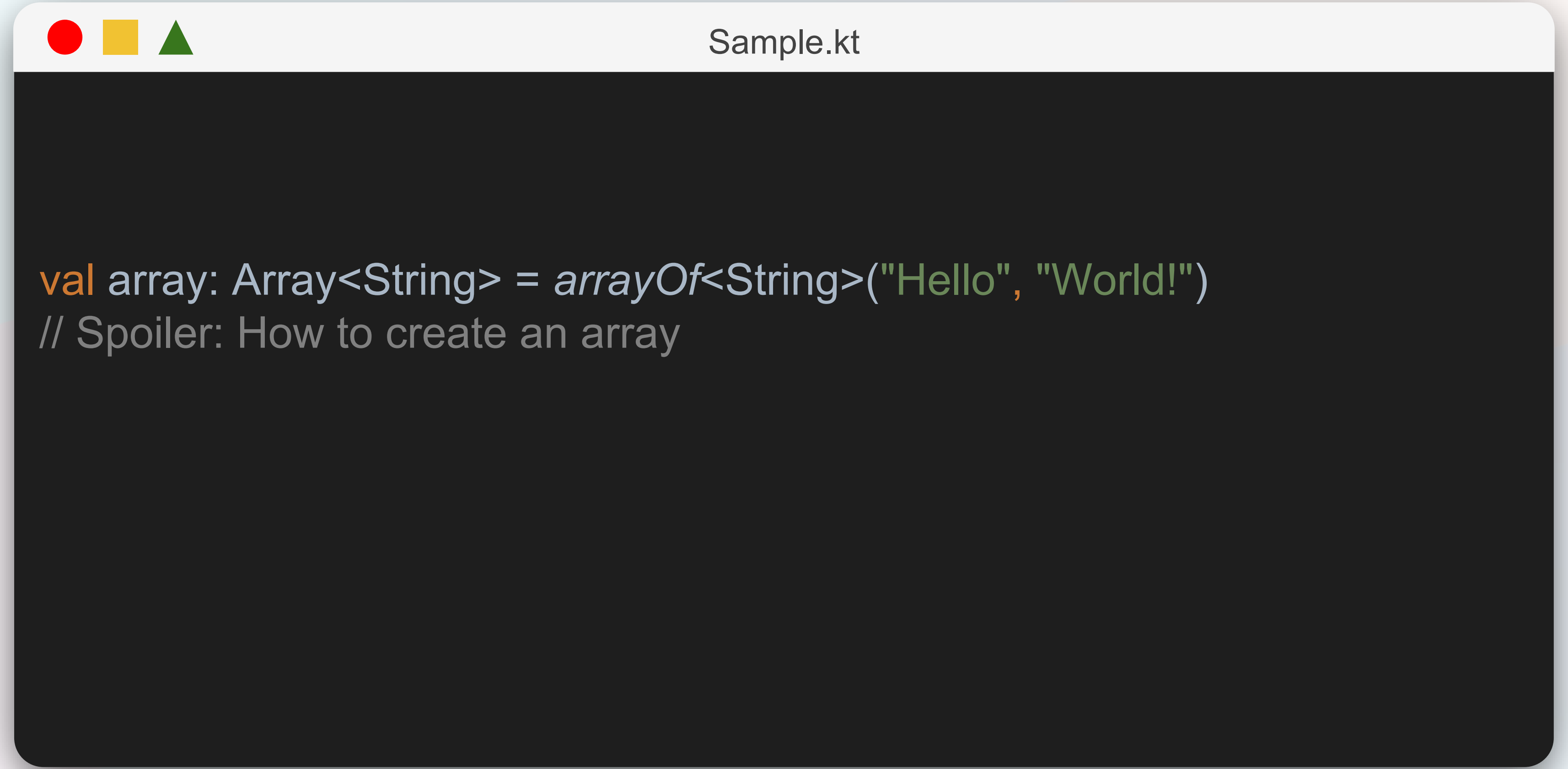
```
for (number in 10 downTo 0) {
    println(number)
}
```

Control Flow - For



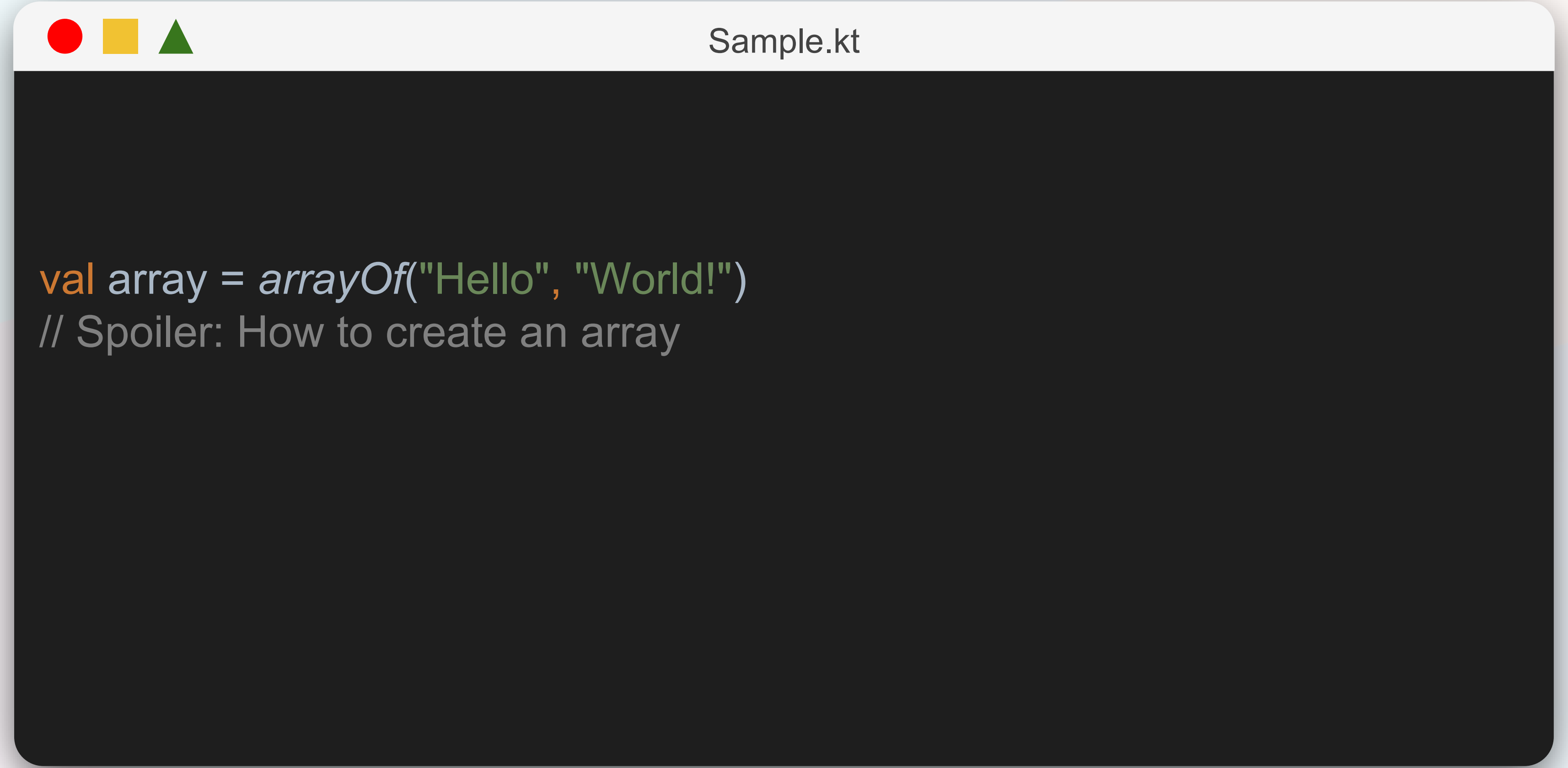
```kotlin
for (number in 10 downTo 0) {
   println(number)
}
// Prints 10, 9, ..., 0
```

Control Flow - For


Sample.kt

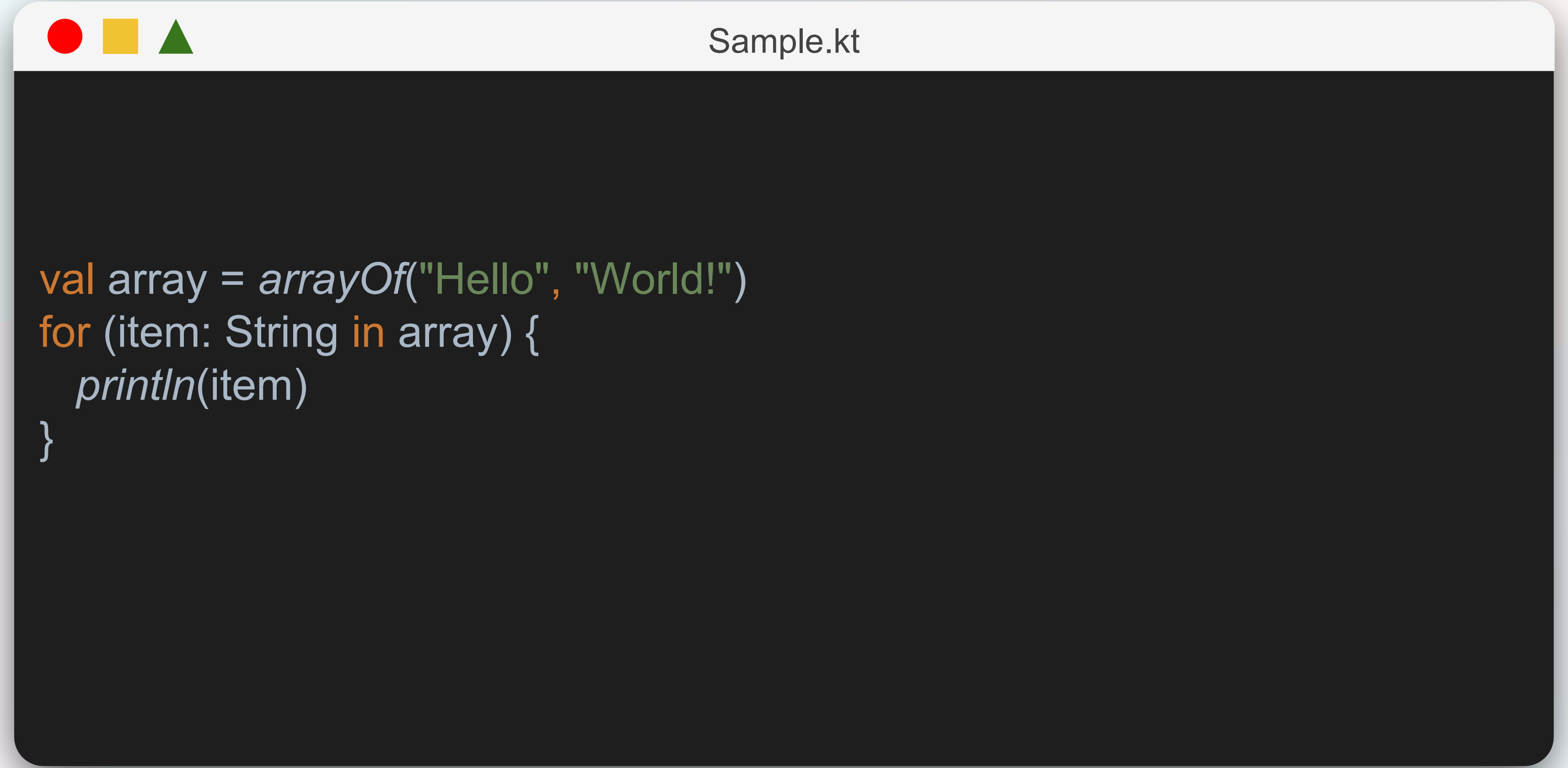
```
val array: Array<String> = arrayOf<String>("Hello", "World!")
// Spoiler: How to create an array
```

Control Flow - For


Sample.kt

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

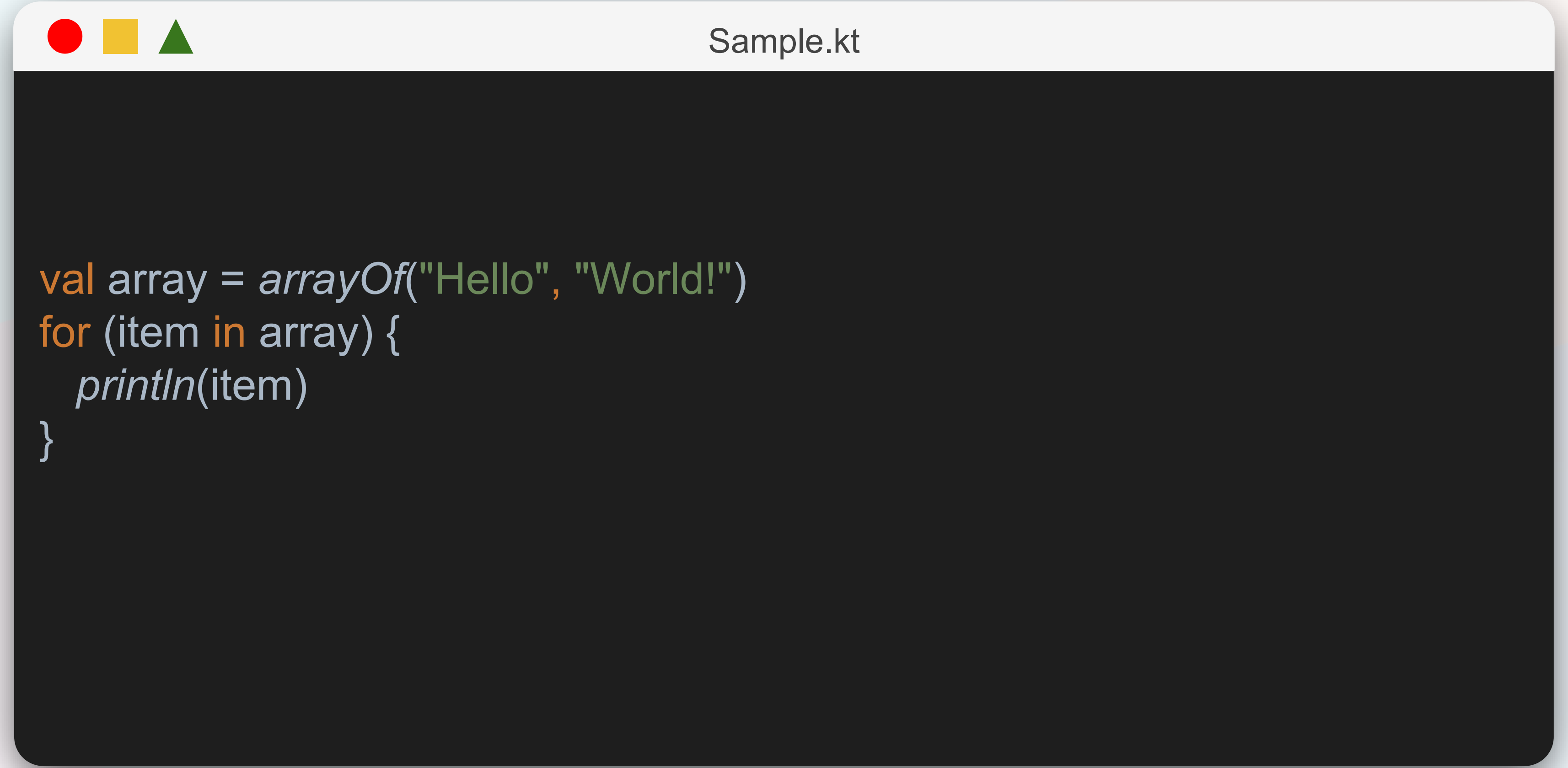
Control Flow - For



Sample.kt

```kotlin
val array = arrayOf("Hello", "World!")
for (item: String in array) {
   println(item)
}
```

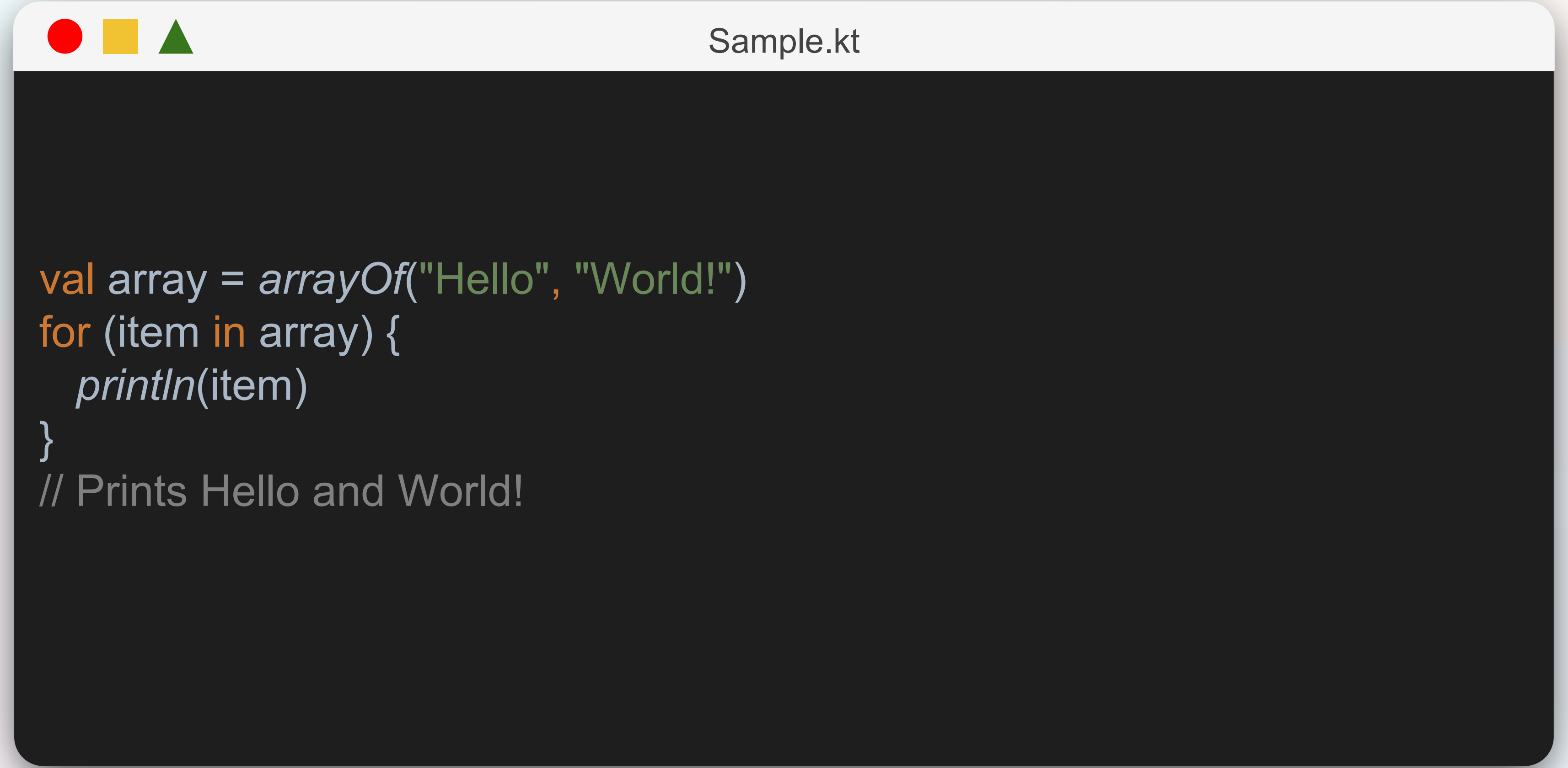
Control Flow - For



Sample.kt

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

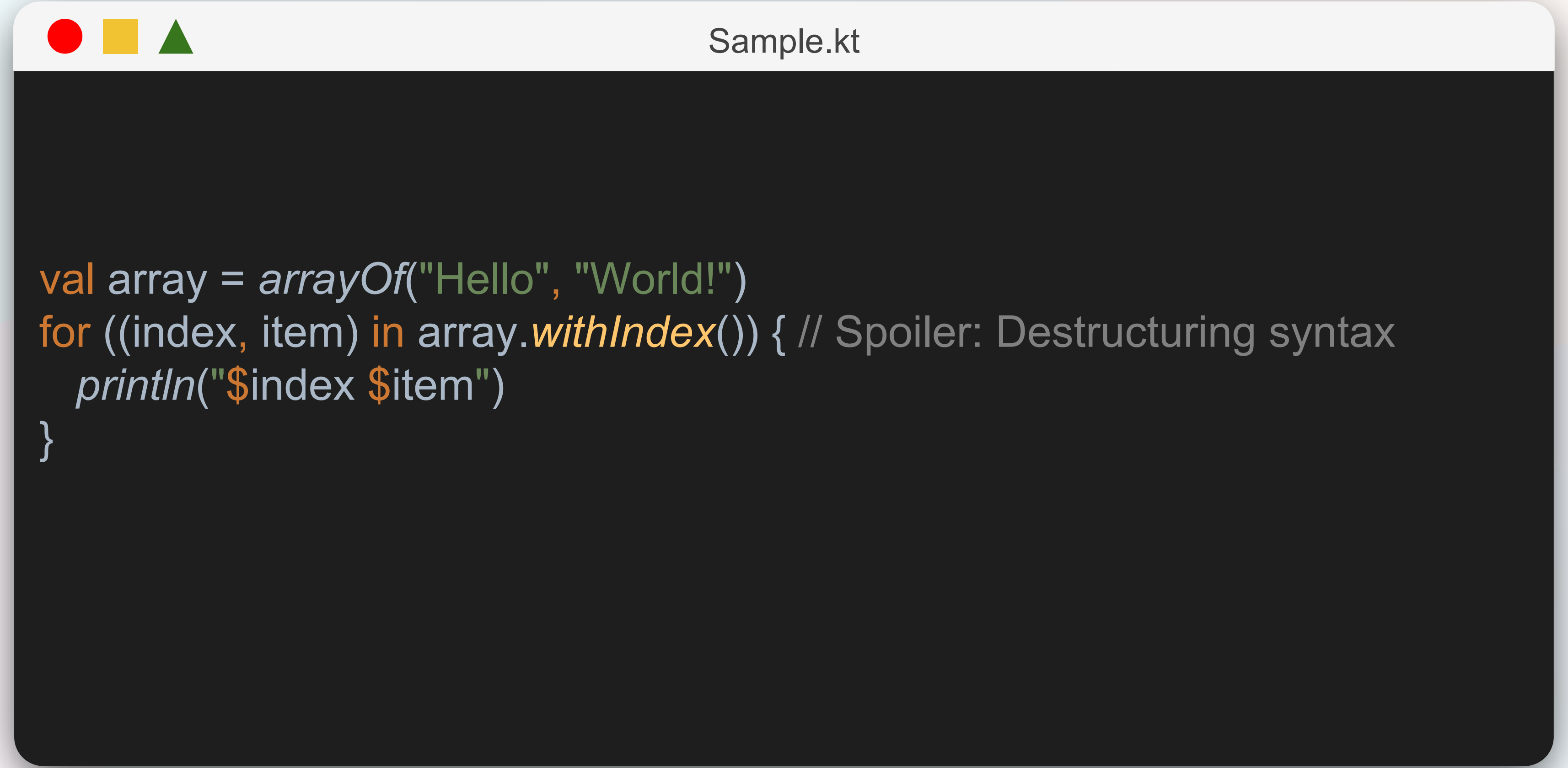
Control Flow - For



Sample.kt

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

Control Flow - For


Sample.kt

```
val array = arrayOf("Hello", "World!")
for ((index, item) in array.withIndex()) { // Spoiler: Destructuring syntax
    println("$index $item")
}
```

Control Flow - For

Sample.kt

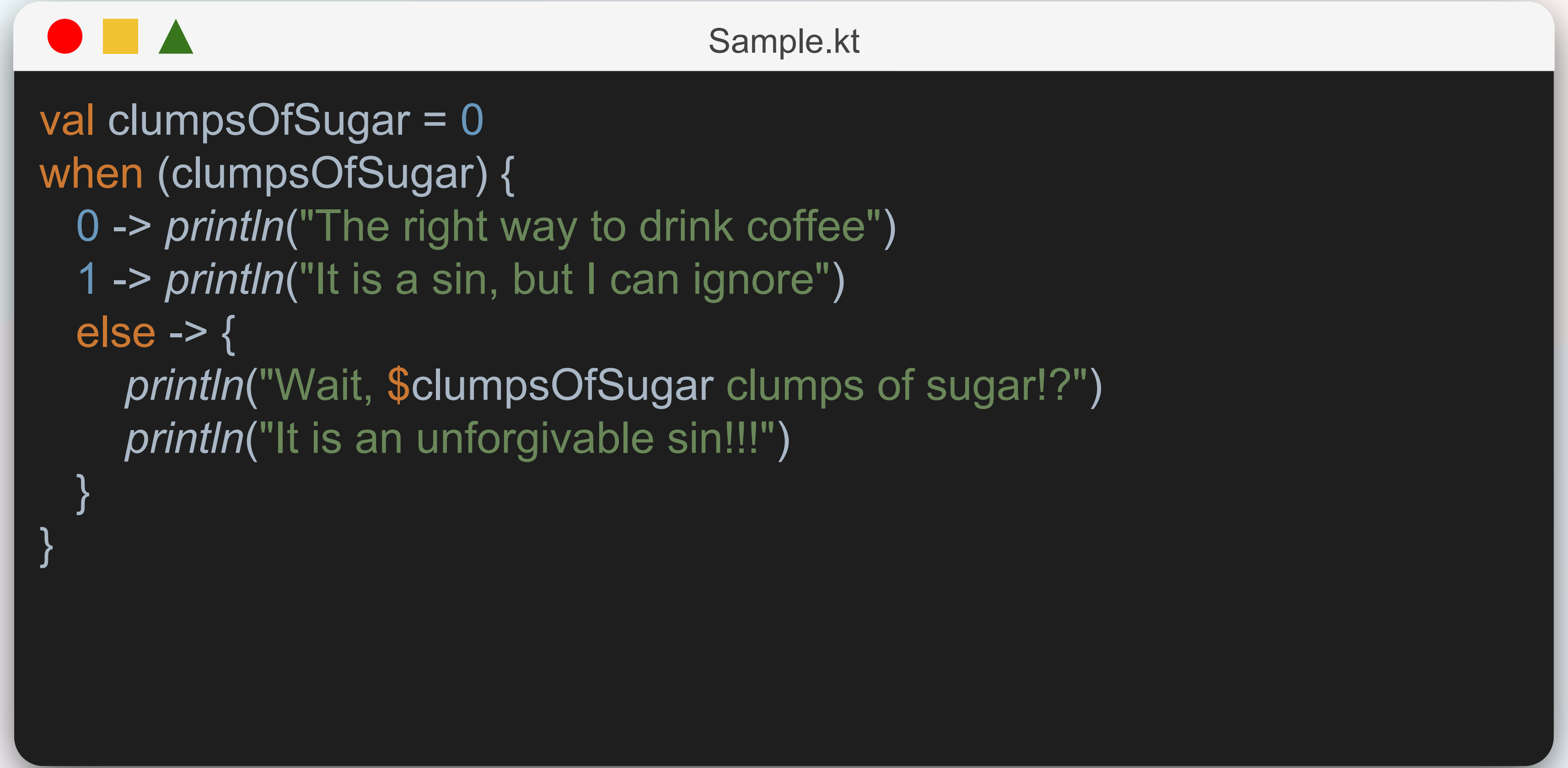
```kotlin
val array = arrayOf("Hello", "World!")
for ((index, item) in array.withIndex()) {
   println("$index $item")
}
// Prints 0 Hello and 1 World!
```

Basics

Control Flow

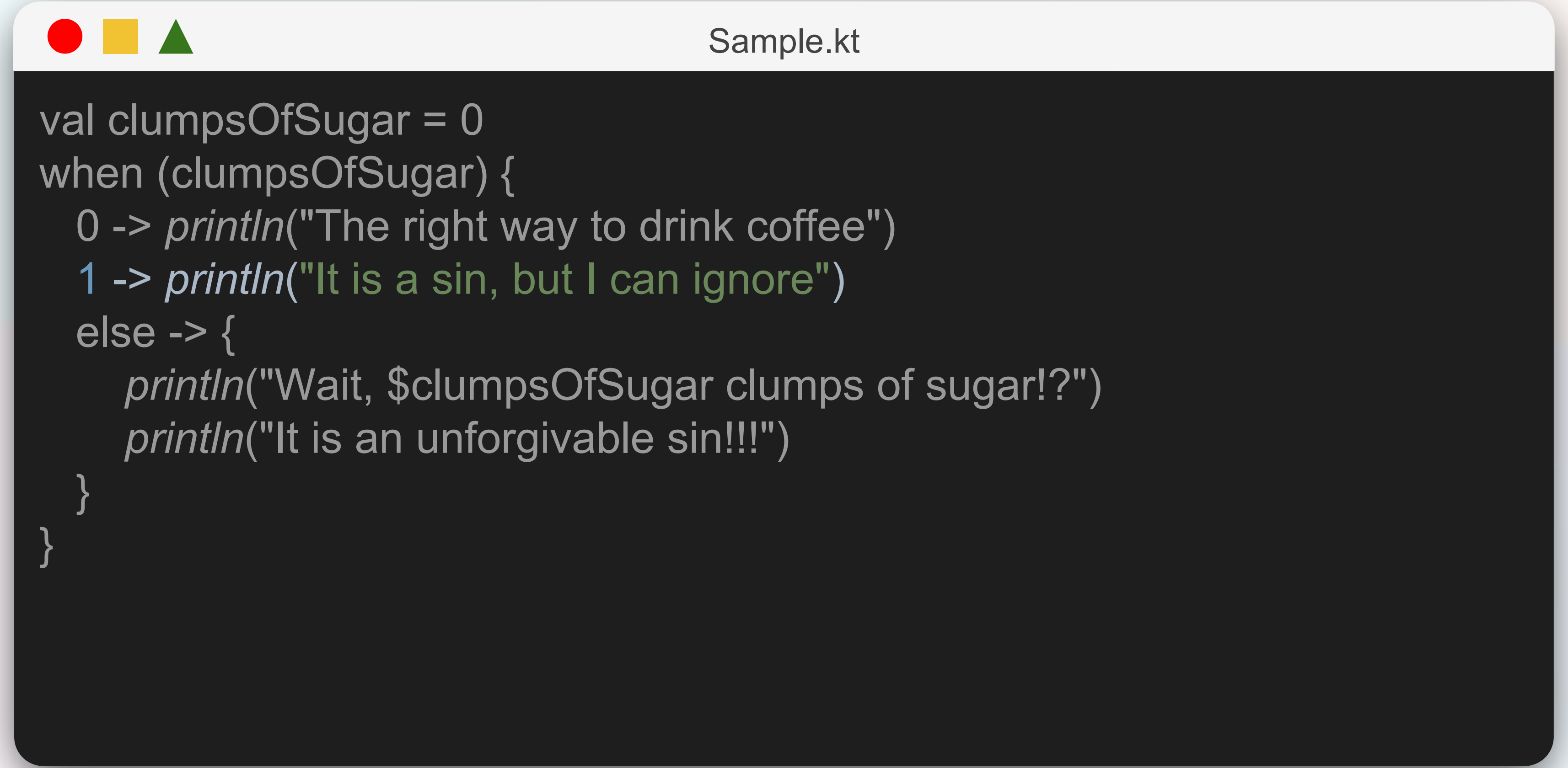
When

Control Flow - When


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!!!")
   }
}
```

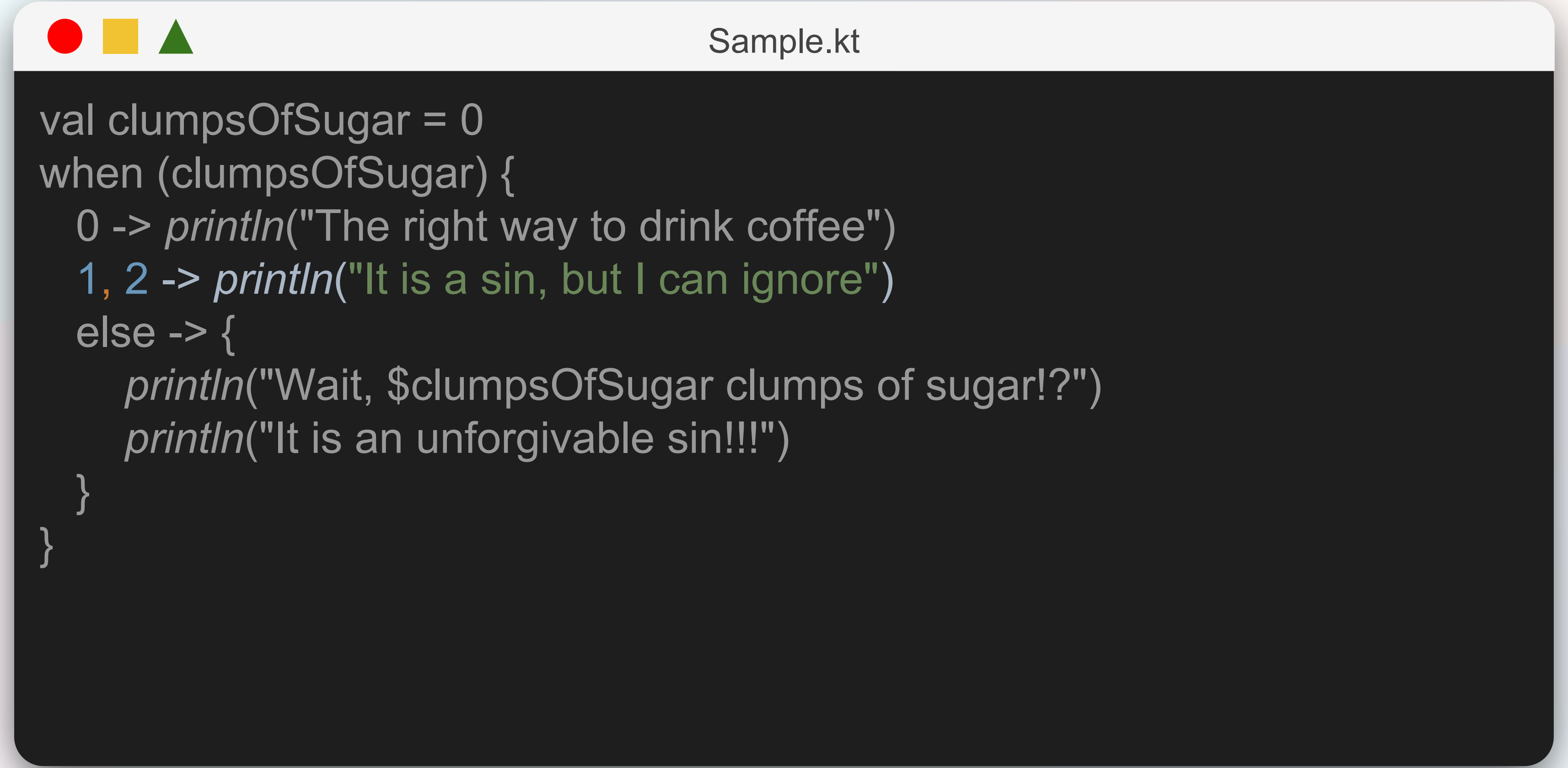
Control Flow - When



Sample.kt

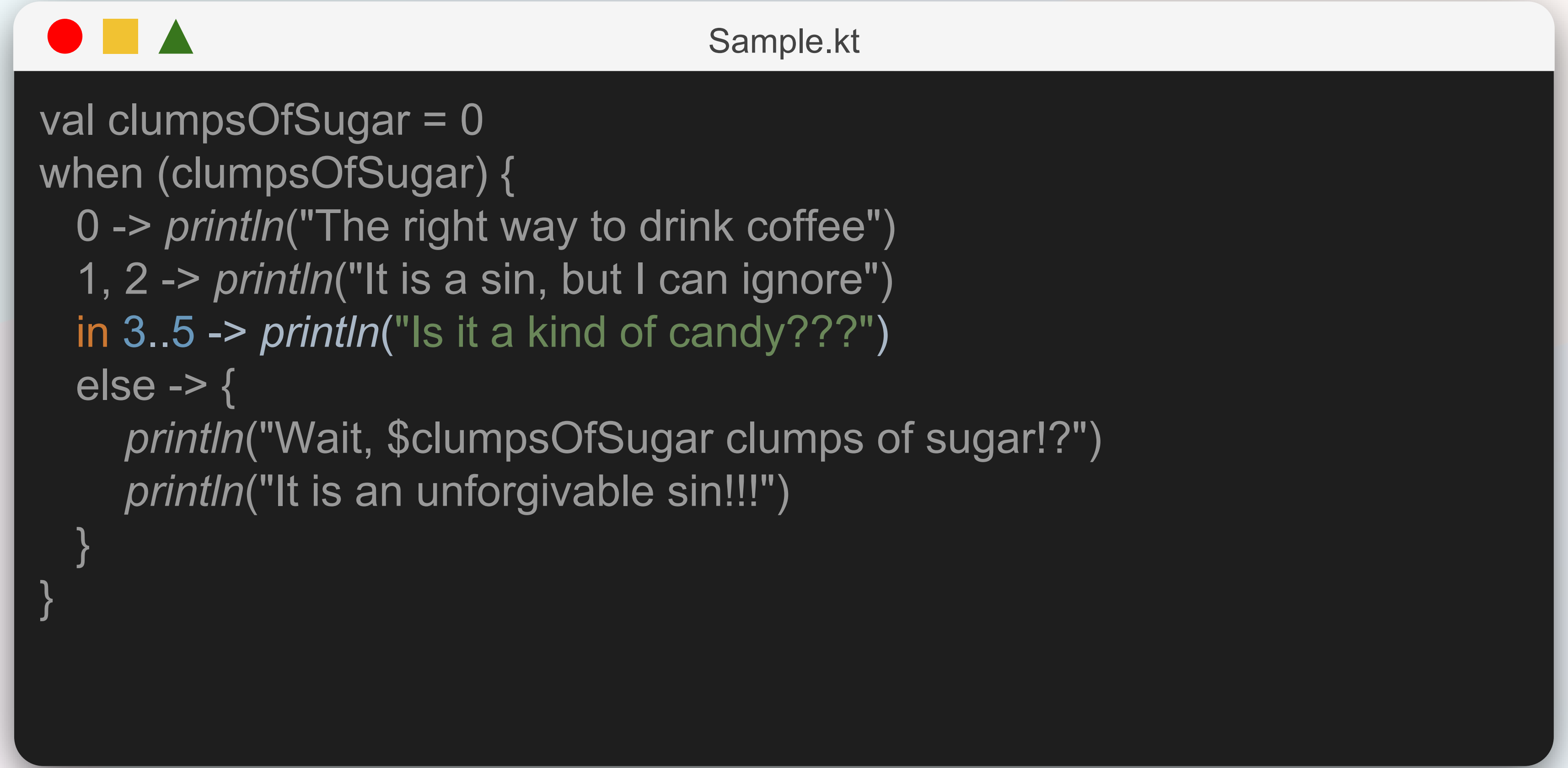
```
val clumpsOfSugar = 0
when (clumpsOfSugar) {
   0 -> println("The right way to drink coffee")
   1 -> println("It is a sin, but I can ignore")
   else -> {
       println("Wait, $clumpsOfSugar clumps of sugar!?")
       println("It is an unforgivable sin!!!")
   }
}
```

Control Flow - When



```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!!!")
   }
}
```

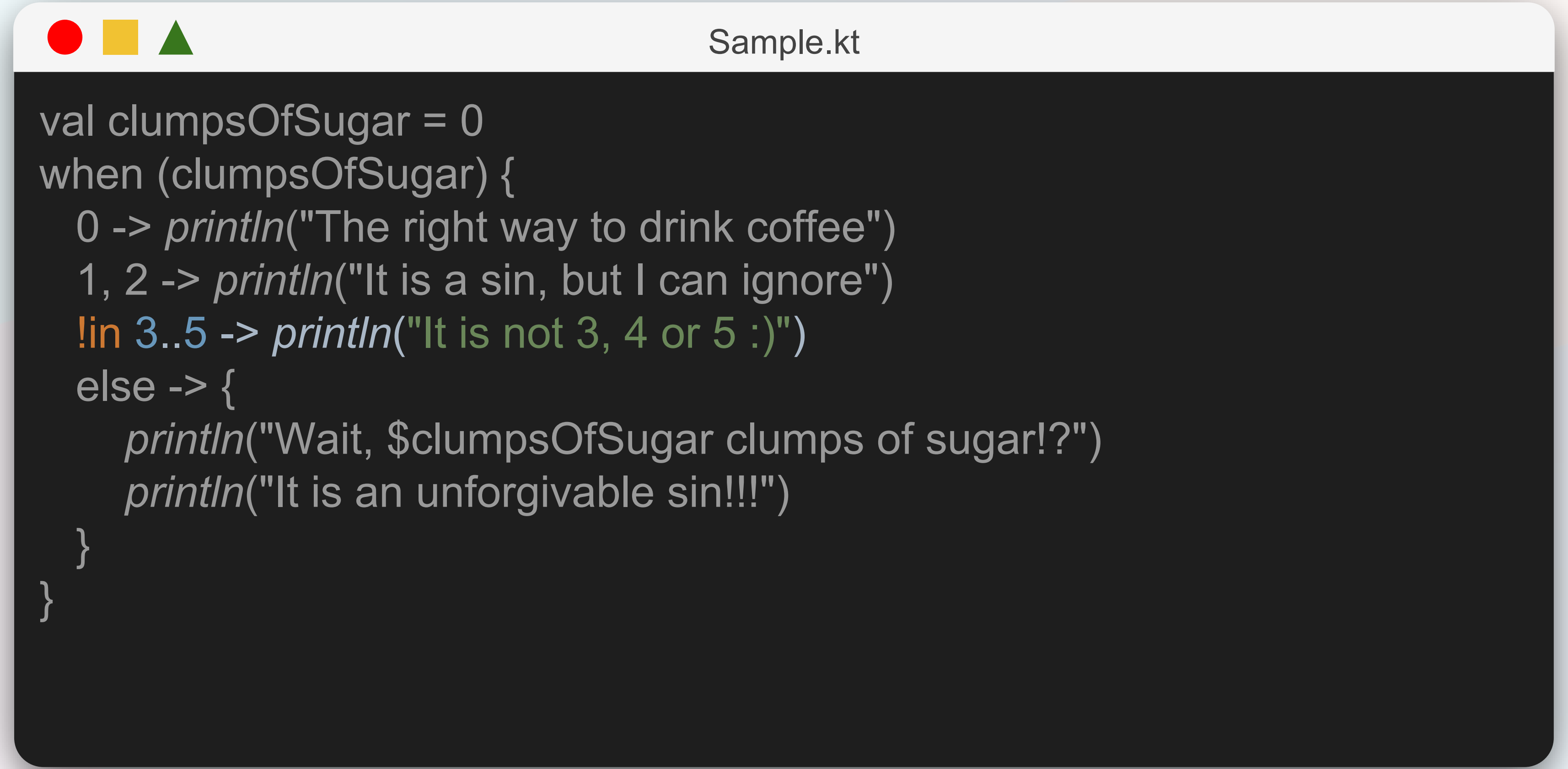
Control Flow - When



Sample.kt

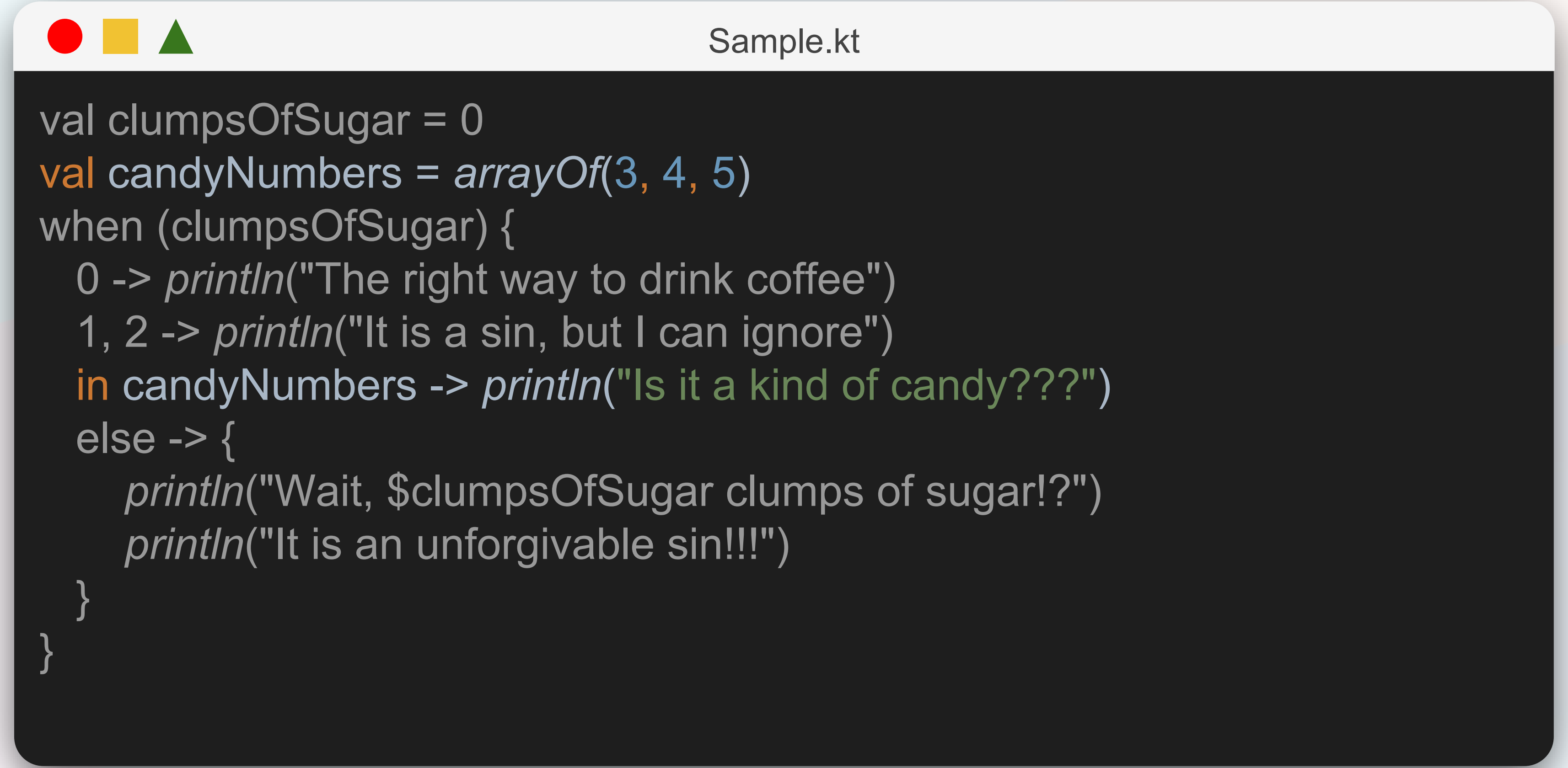
```kotlin
val clumpsOfSugar = 0
when (clumpsOfSugar) {
   0 -> println("The right way to drink coffee")
   1, 2 -> println("It is a sin, but I can ignore")
   in 3..5 -> println("Is it a kind of candy???")
   else -> {
      println("Wait, $clumpsOfSugar clumps of sugar!?")
      println("It is an unforgivable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

Control Flow - When


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!!!")
   }
}
```

Control Flow - When


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!!!")
   }
}
```

Control Flow - When

Sample.kt

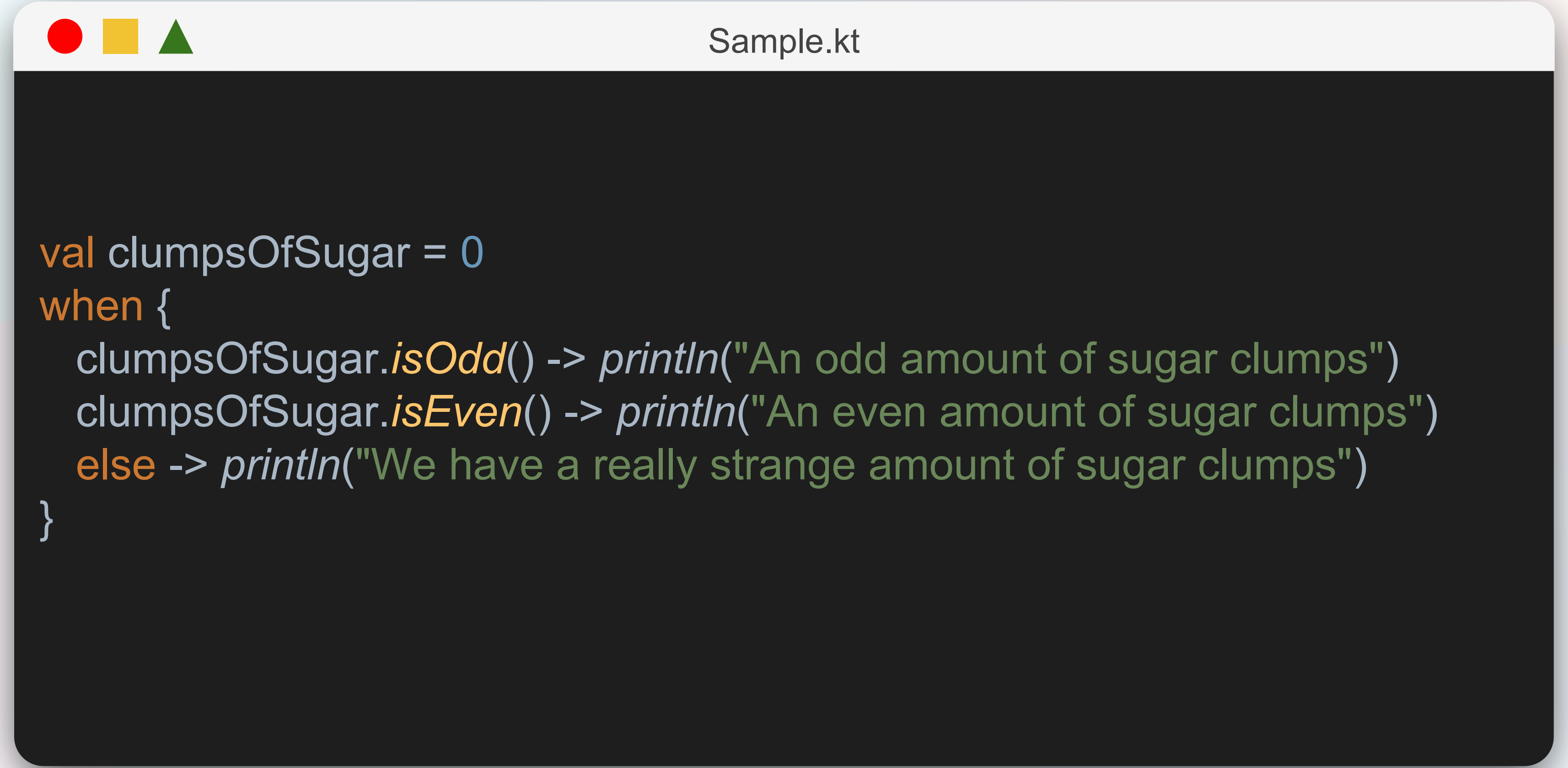
```
val clumpsOfSugar = 0
when (clumpsOfSugar) {
   0 -> println("The right way to drink coffee")
   1, 2 -> println("It is a sin, but I can ignore")
   randomInt() -> println("Are we still talking about coffee?")
   else -> {
      println("Wait, $clumpsOfSugar clumps of sugar!?")
      println("It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

Control Flow - When

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")
}
```

Control Flow - When


Sample.kt

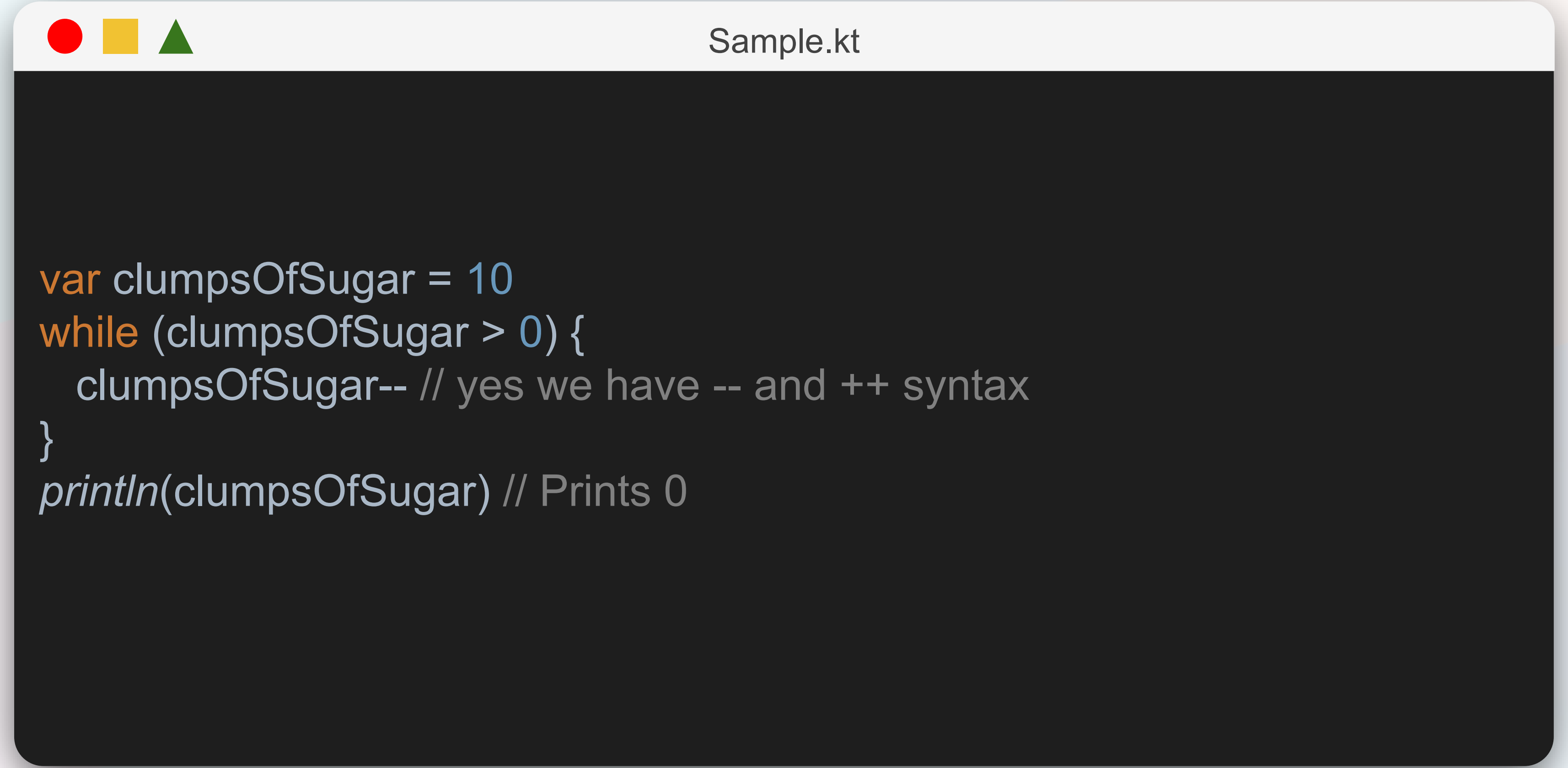
```kotlin
val clumpsOfSugar = 0
when {
   clumpsOfSugar.isOdd() -> println("An odd amount of sugar clumps")
   clumpsOfSugar.isEven() -> println("An even amount of sugar clumps")
   else -> println("We have a really strange amount of sugar clumps")
}
```

Basics

Control Flow

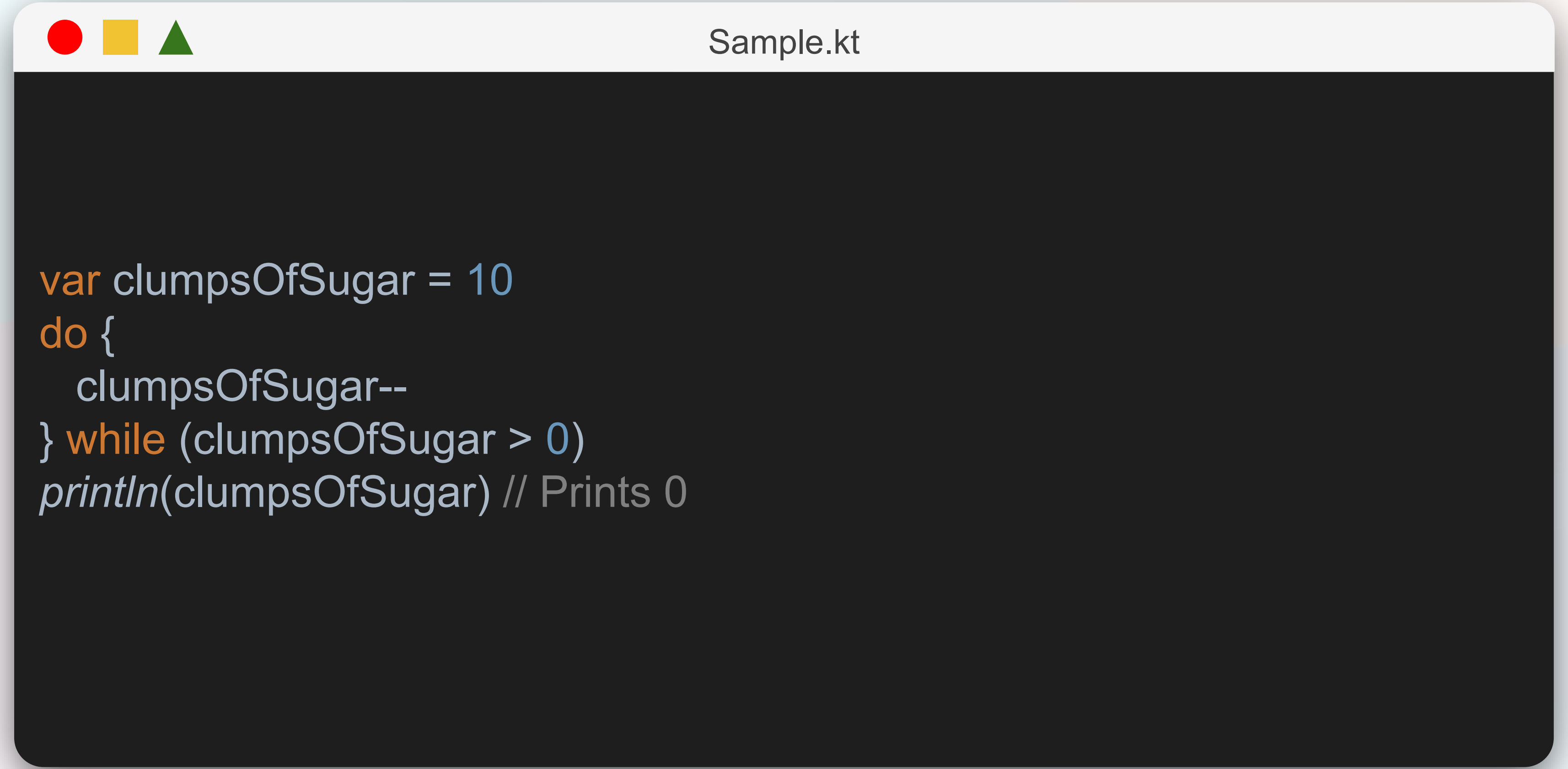
While

Control Flow - While


Sample.kt

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

Control Flow - While

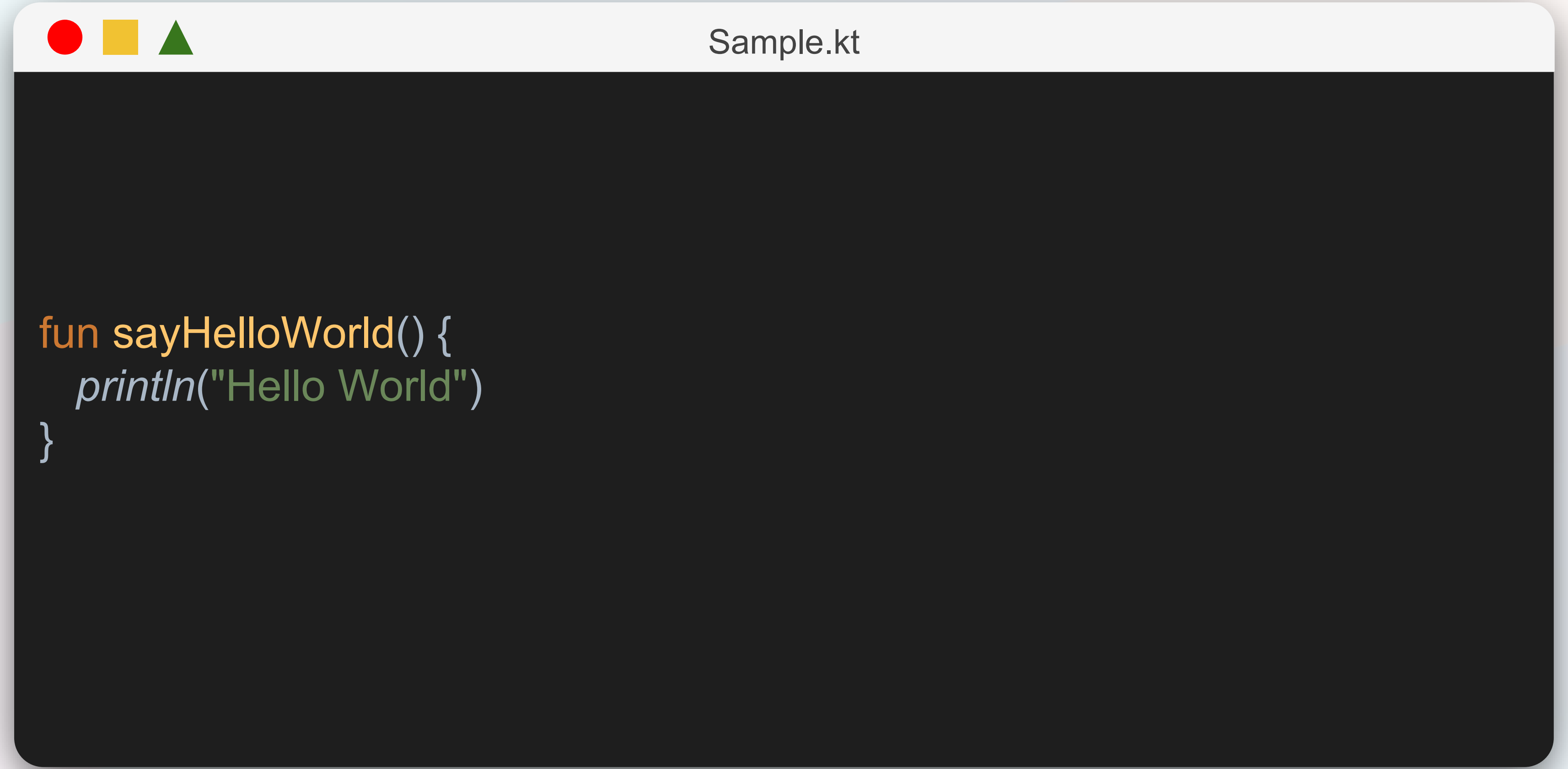

Sample.kt

```
var clumpsOfSugar = 10
do {
   clumpsOfSugar--
} while (clumpsOfSugar > 0)
println(clumpsOfSugar) // Prints 0
```

Basics

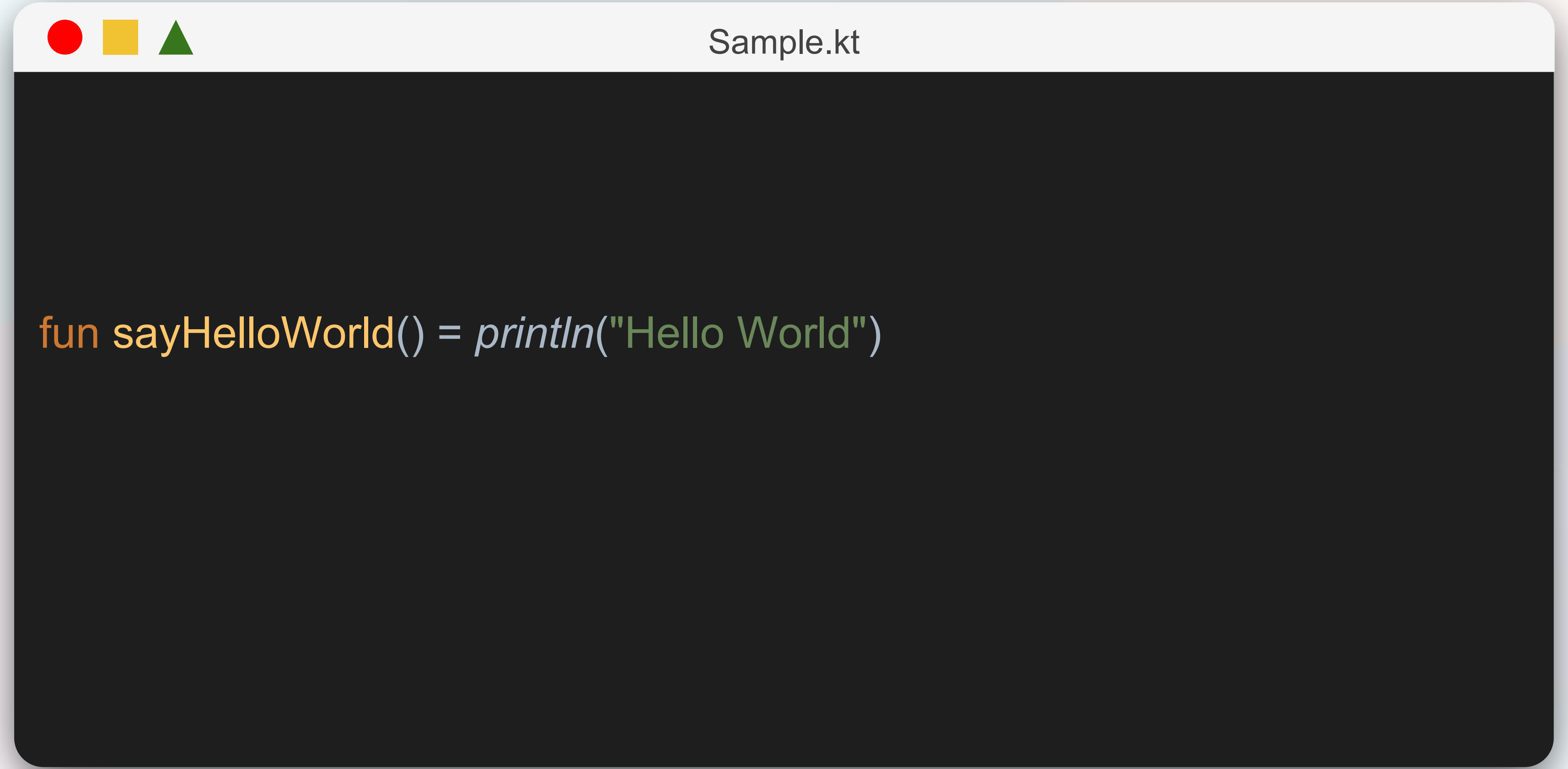
Function

Function


Sample.kt

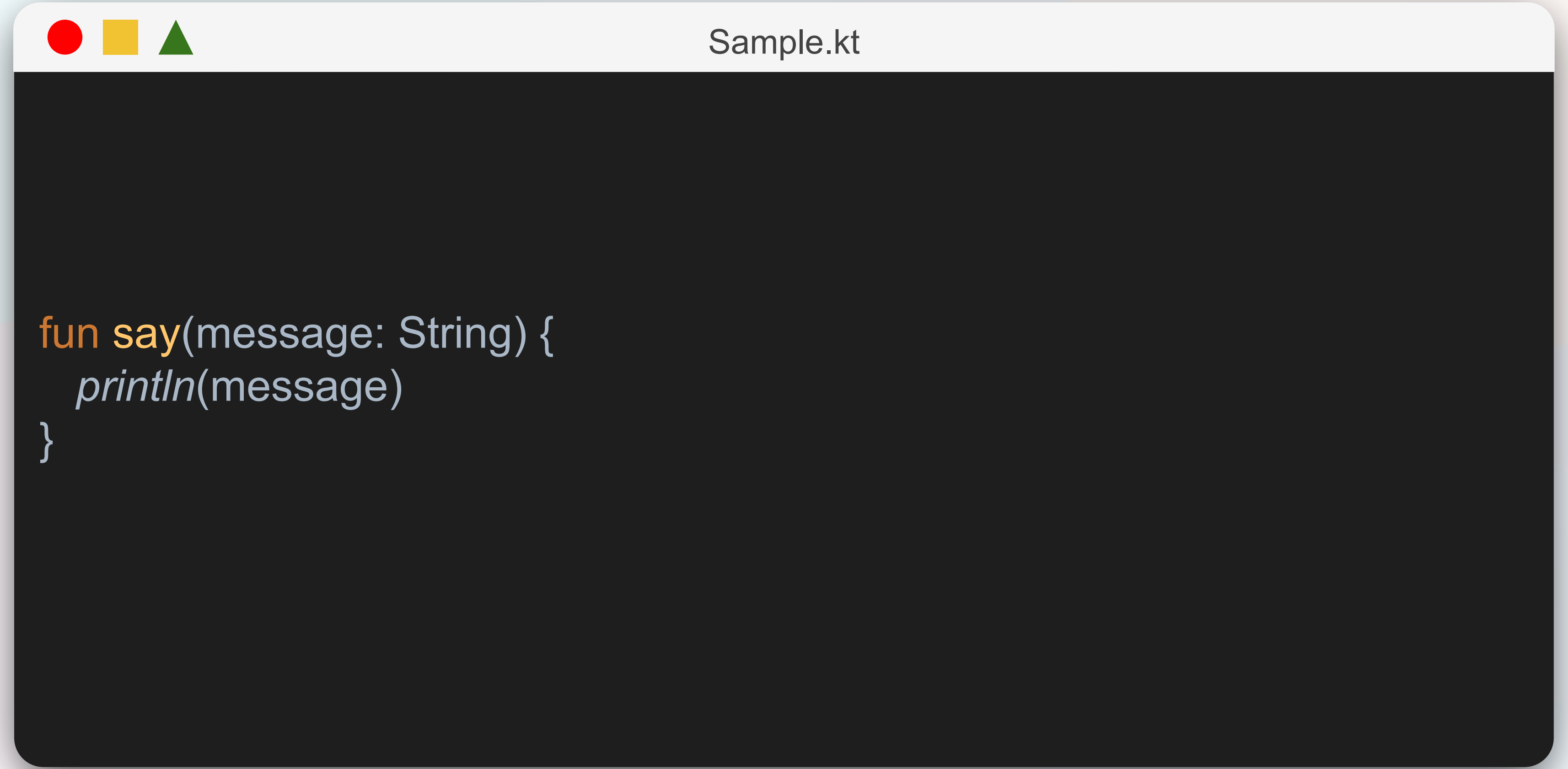
```kotlin
fun sayHelloWorld() {
   println("Hello World")
}
```

Function


Sample.kt

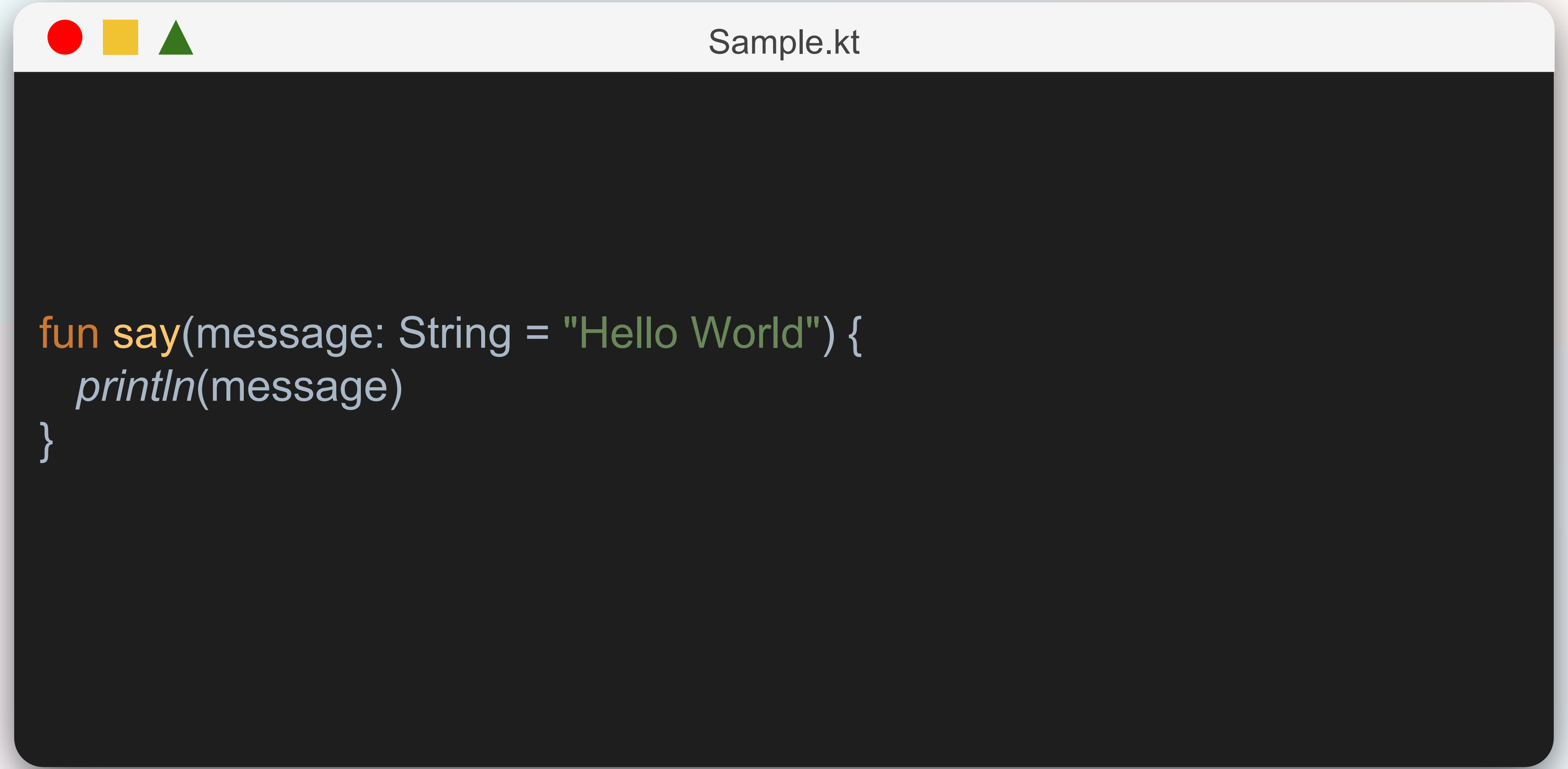
```
fun sayHelloWorld() = println("Hello World")
```

Function


Sample.kt

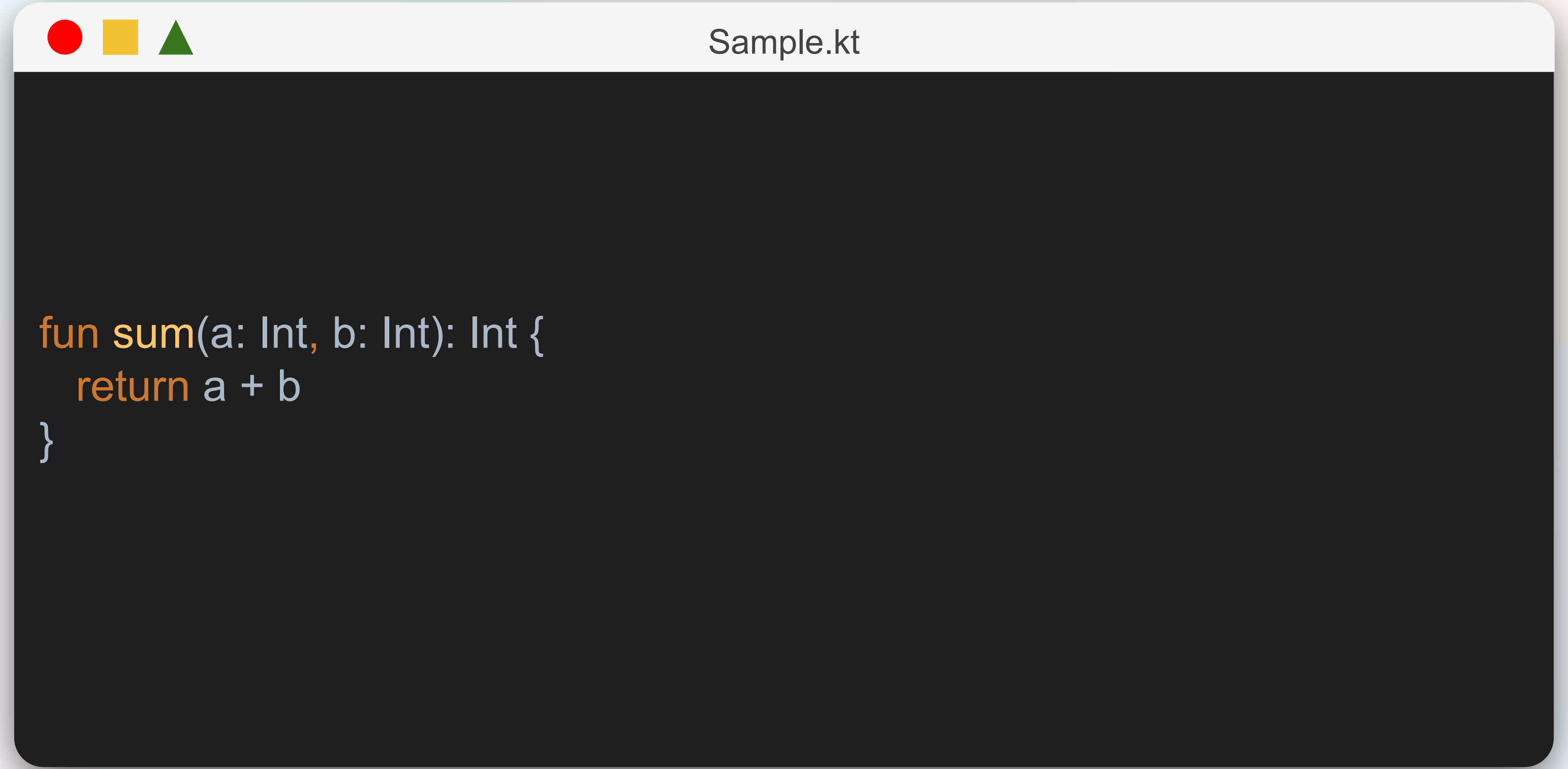
```kotlin
fun say(message: String) {
    println(message)
}
```

Function



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

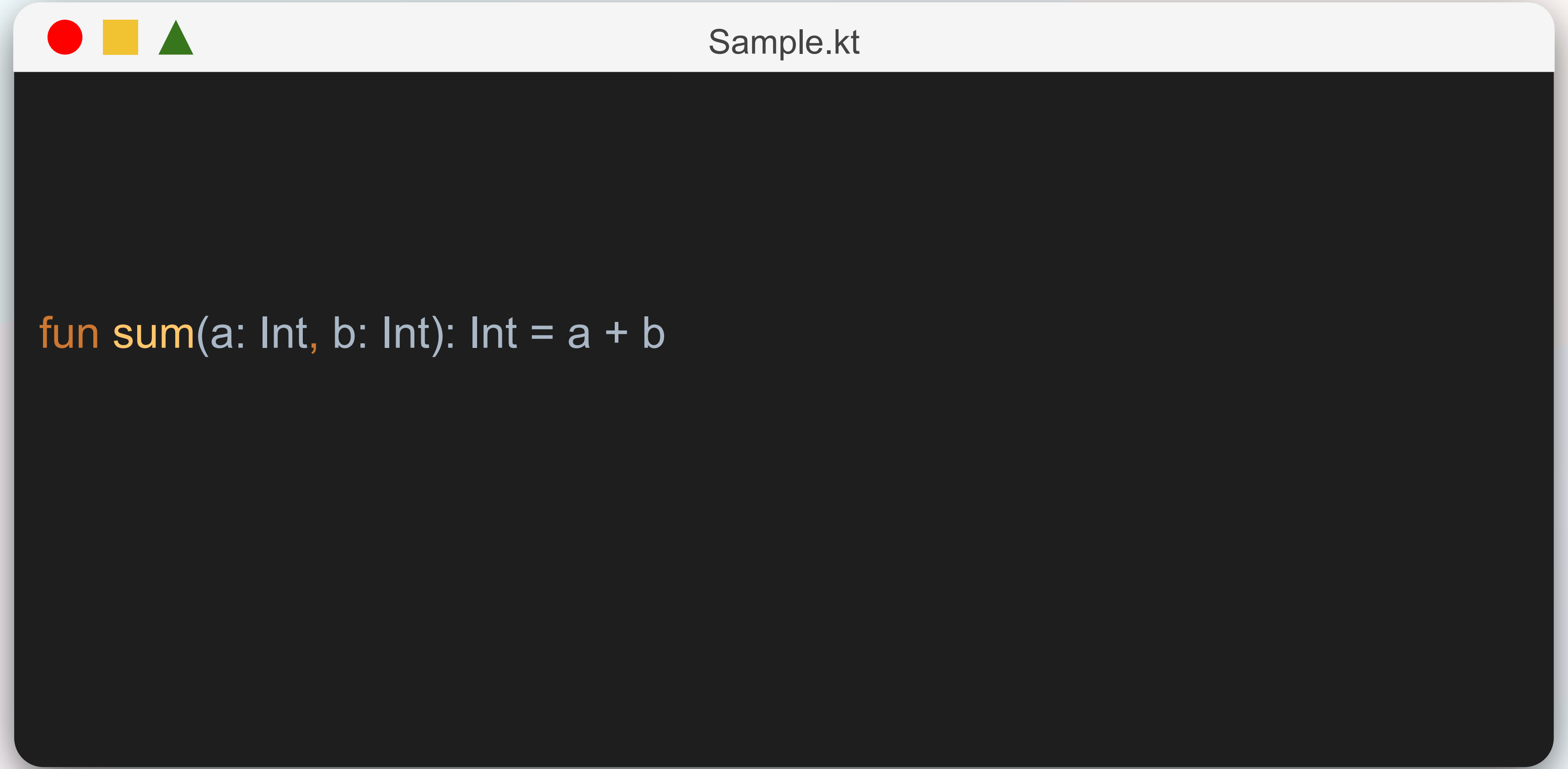
Function



Sample.kt

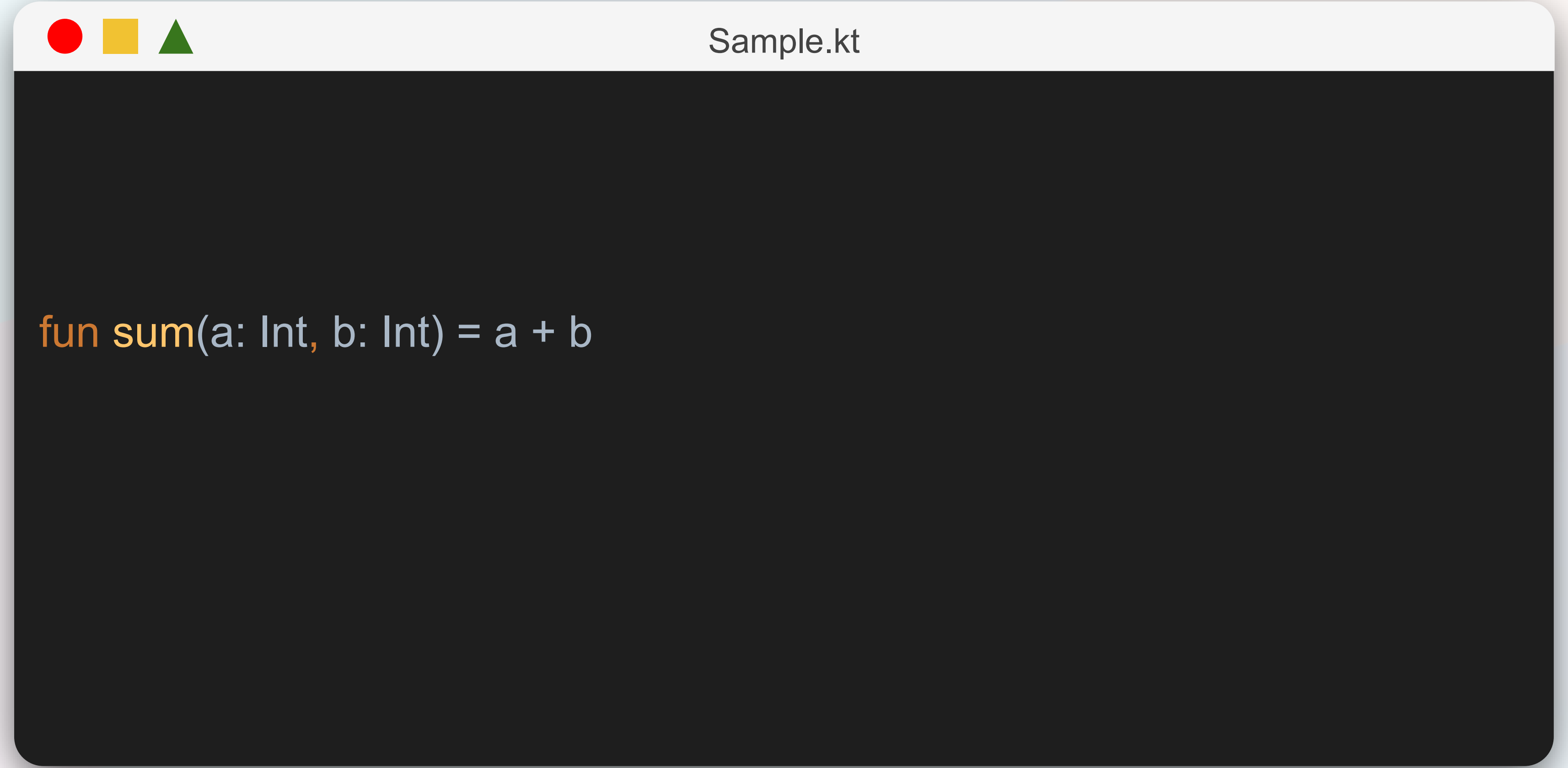
```kotlin
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Function


Sample.kt

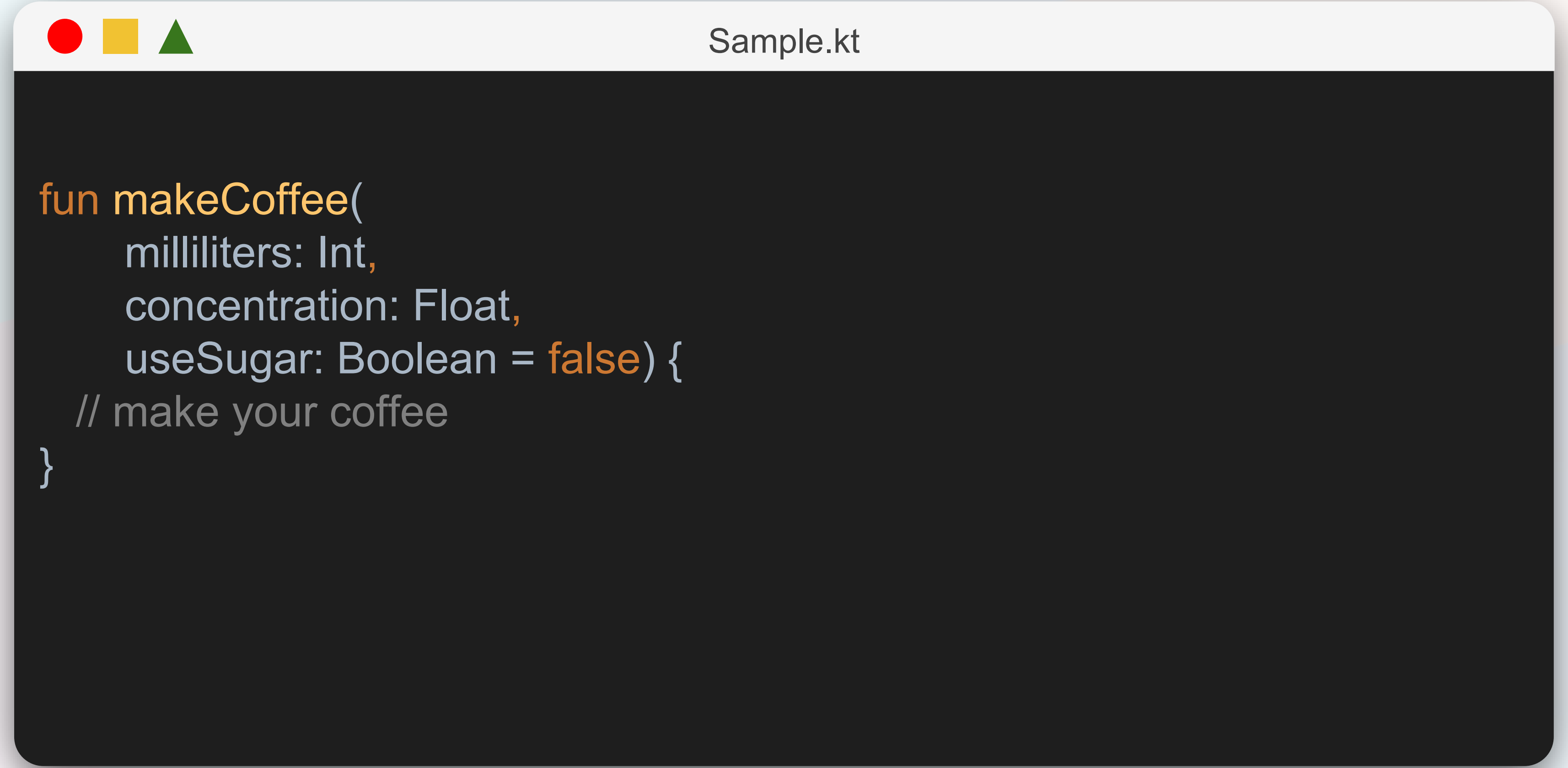
```kotlin
fun sum(a: Int, b: Int): Int = a + b
```

Function



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

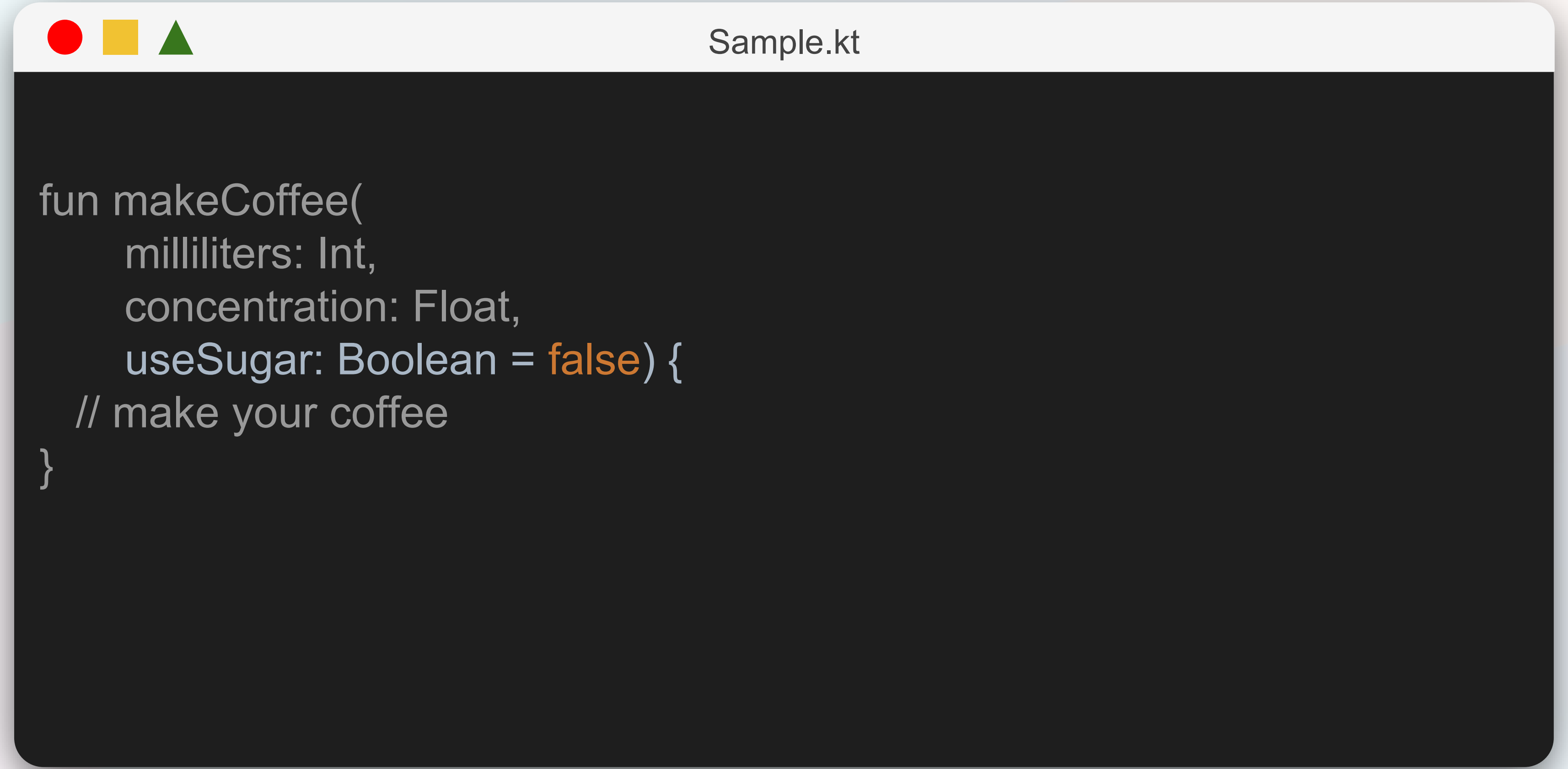
Function



Sample.kt

```
fun makeCoffee(
        milliliters: Int,
        concentration: Float,
        useSugar: Boolean = false) {
    // make your coffee
}
```

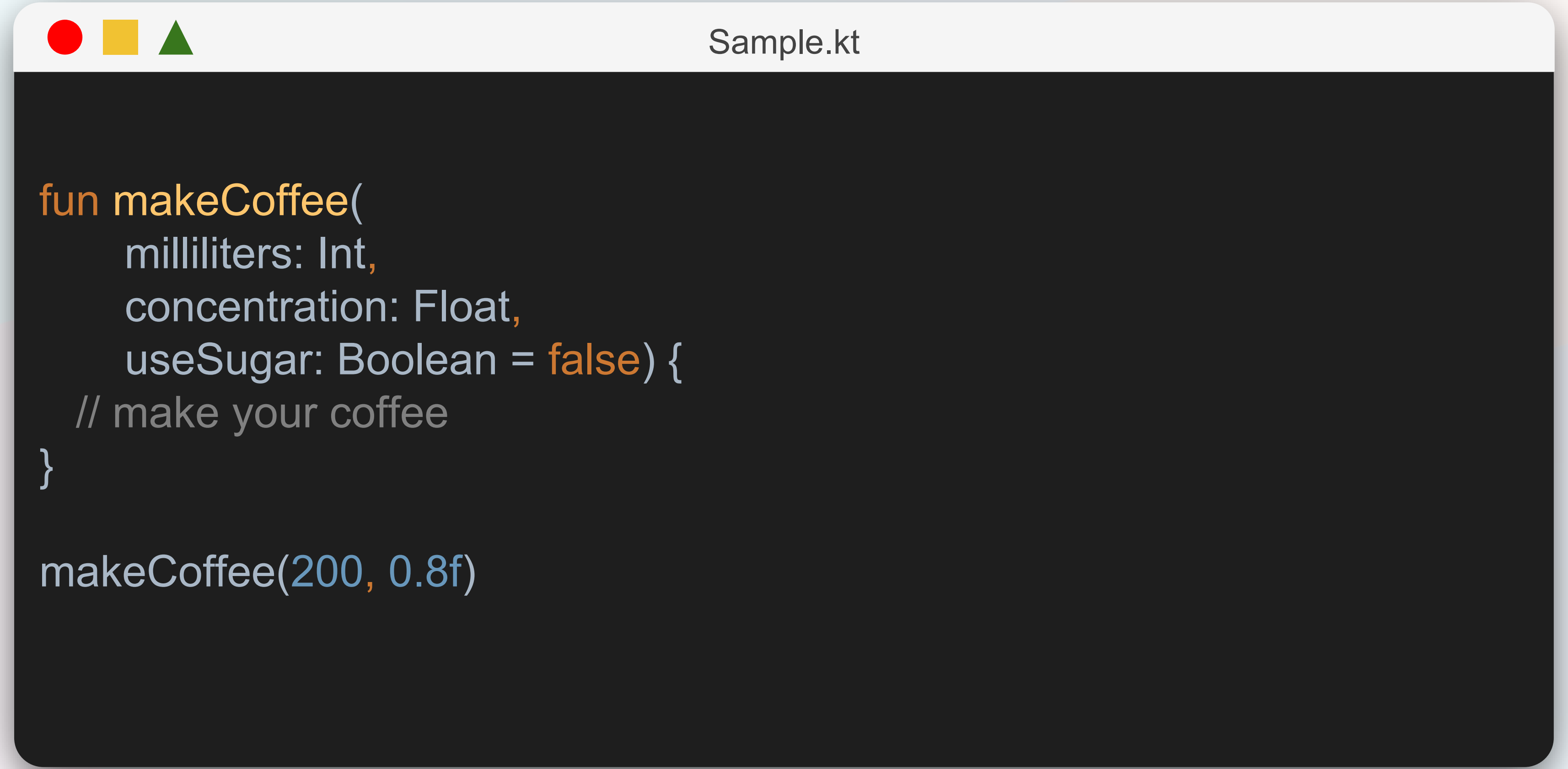
Function



Sample.kt

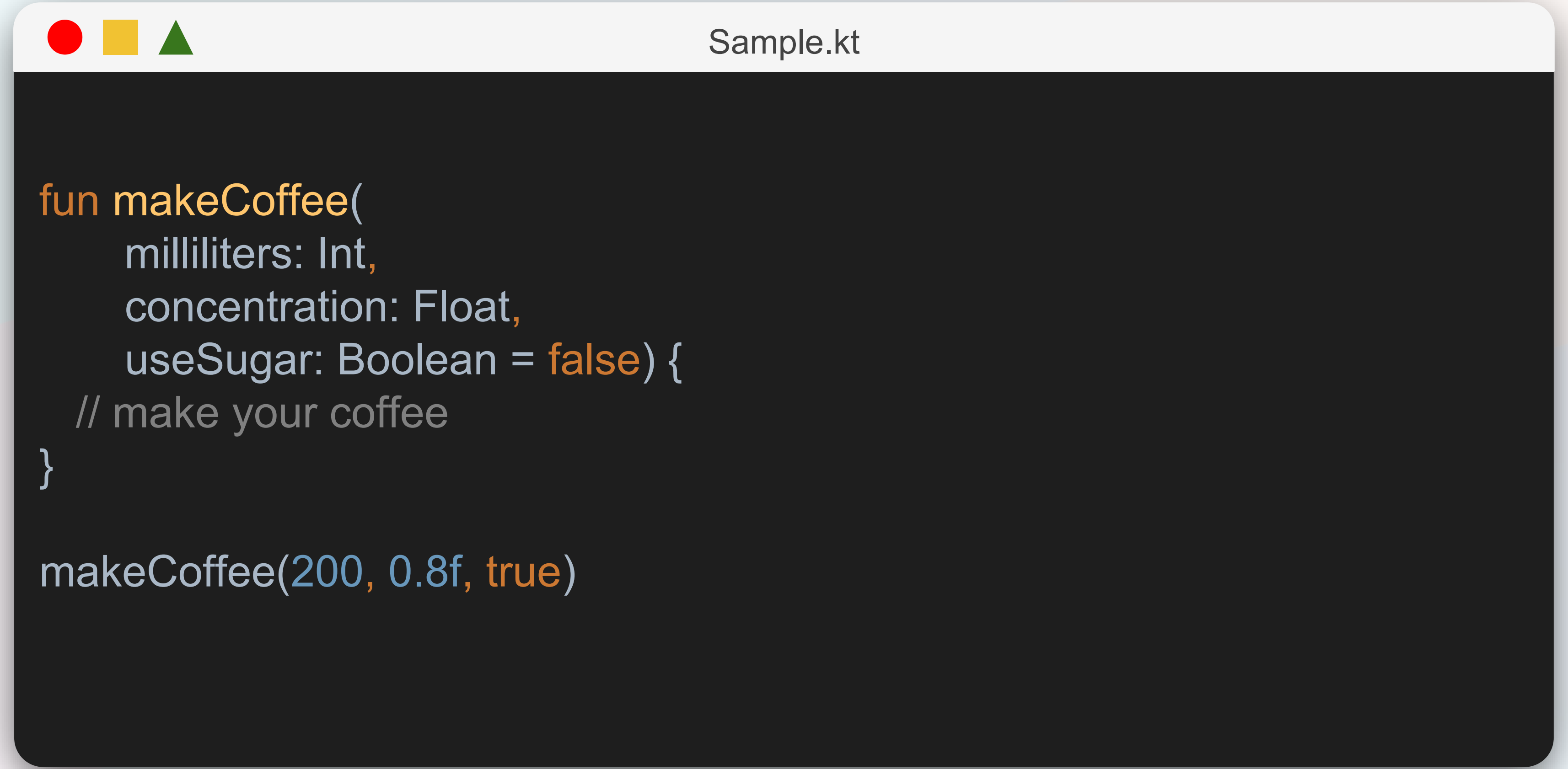
```
fun makeCoffee(
        milliliters: Int,
        concentration: Float,
        useSugar: Boolean = false) {
    // make your coffee
}
```

Function


Sample.kt

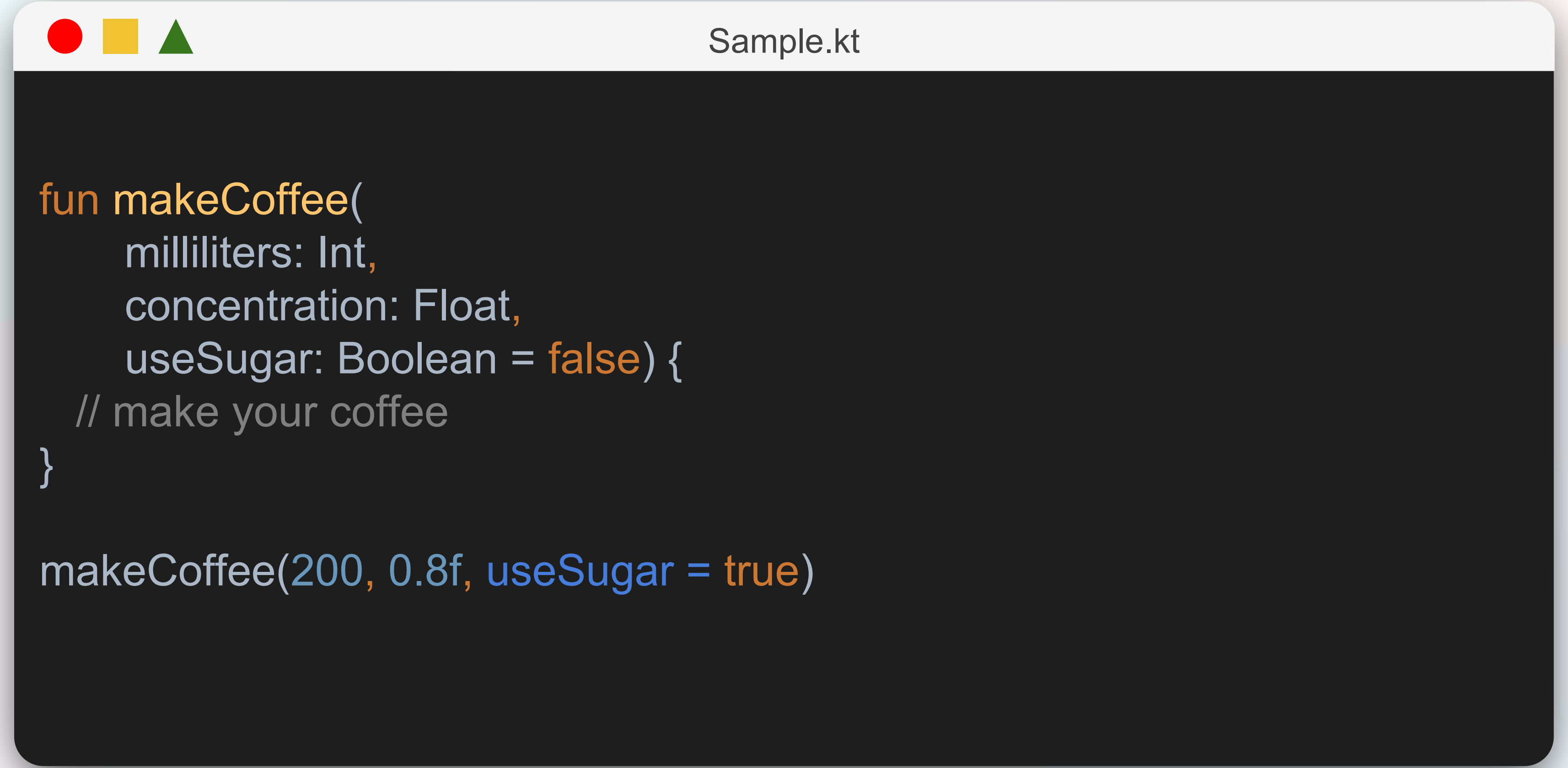
```
fun makeCoffee(
        milliliters: Int,
        concentration: Float,
        useSugar: Boolean = false) {
    // make your coffee
}

makeCoffee(200, 0.8f)
```

Function


Sample.kt

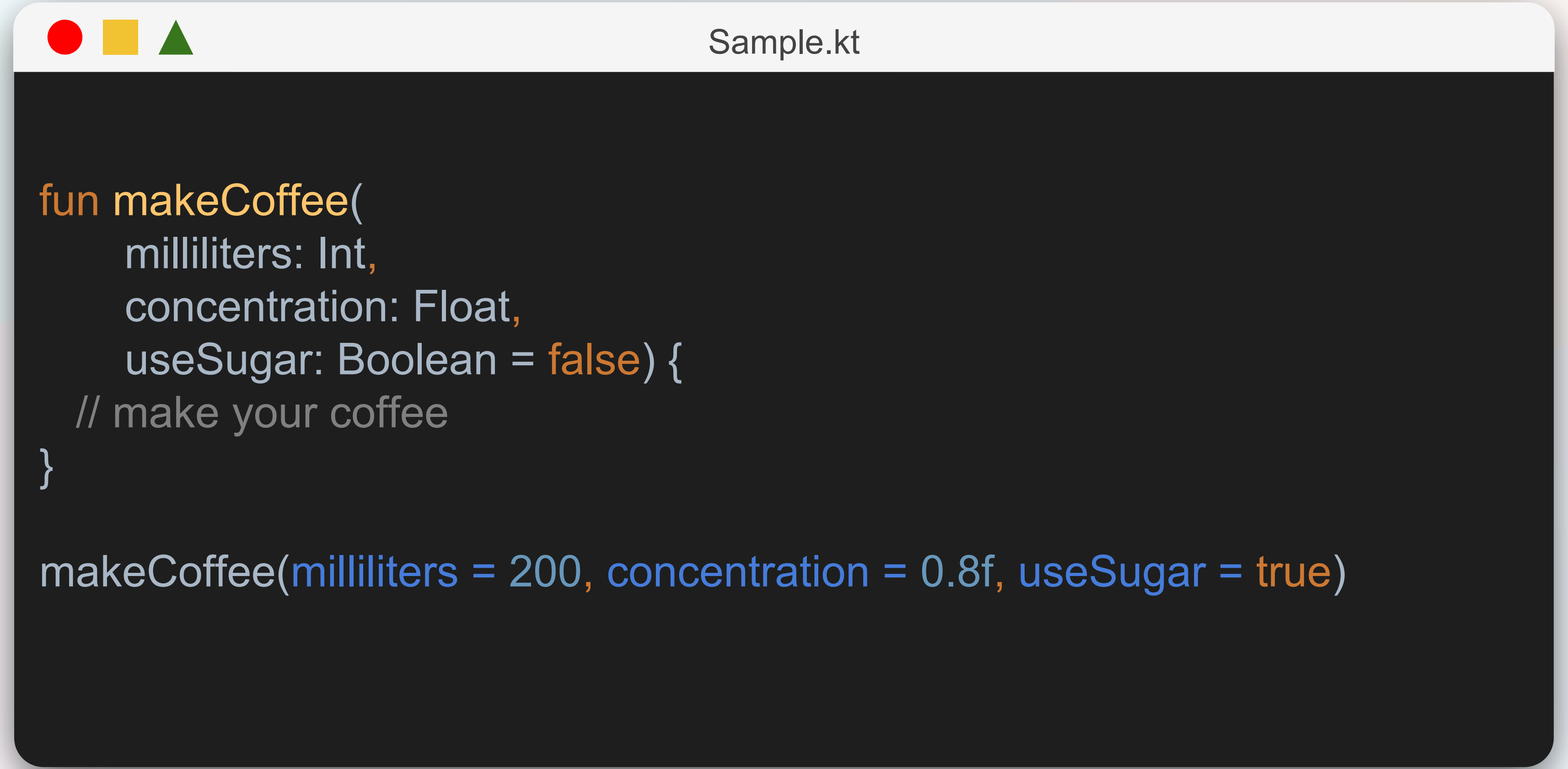
```kotlin
fun makeCoffee(
        milliliters: Int,
        concentration: Float,
        useSugar: Boolean = false) {
    // make your coffee
}

makeCoffee(200, 0.8f, true)
```

Function



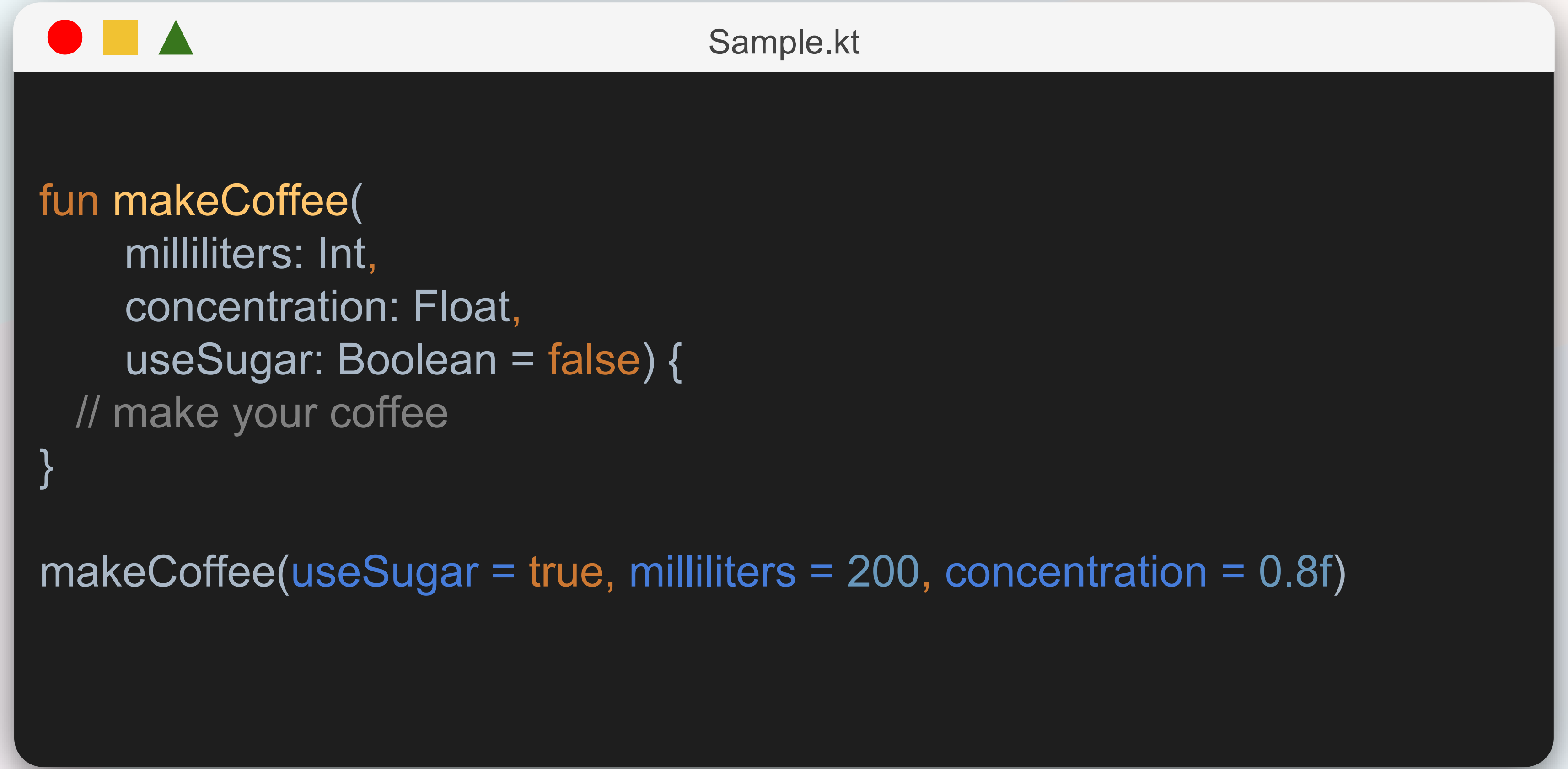
```kotlin
fun makeCoffee(
        milliliters: Int,
        concentration: Float,
        useSugar: Boolean = false) {
   // make your coffee
}

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

Function


Sample.kt

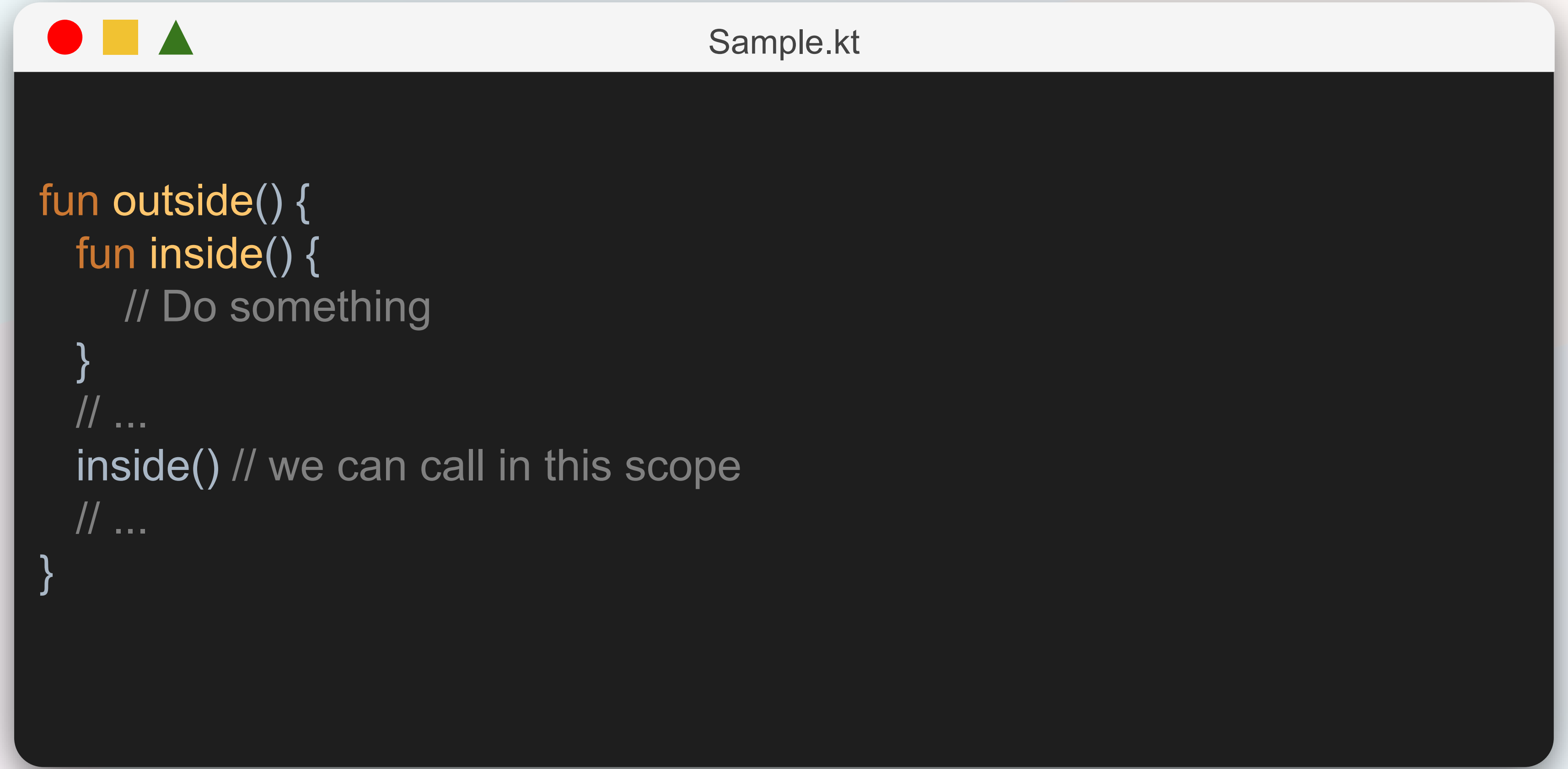
```kotlin
fun makeCoffee(
        milliliters: Int,
        concentration: Float,
        useSugar: Boolean = false) {
   // make your coffee
}

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

Function


Sample.kt

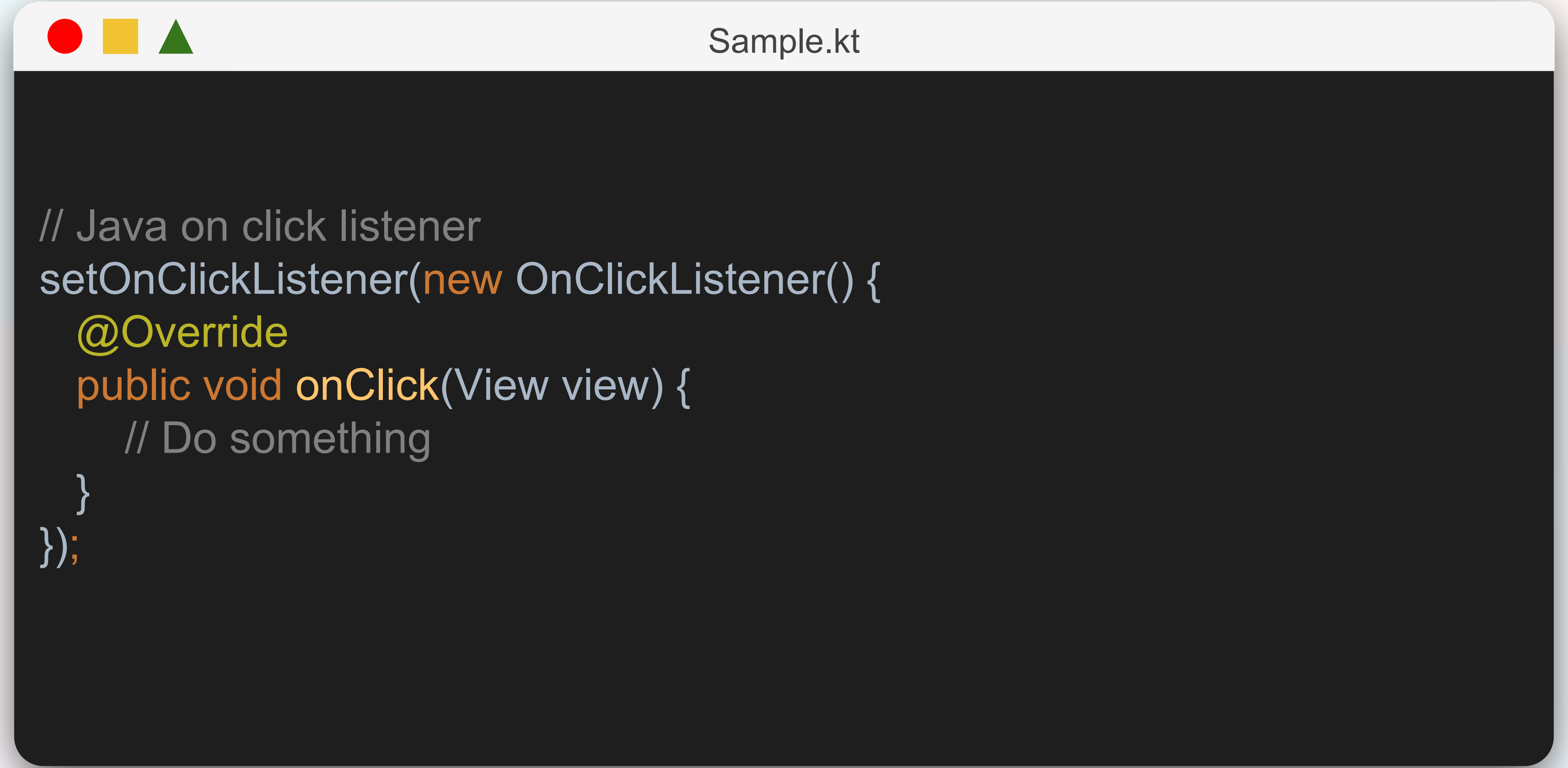
```
fun makeCoffee(
        milliliters: Int,
        concentration: Float,
        useSugar: Boolean = false) {
    // make your coffee
}

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

Function


Sample.kt

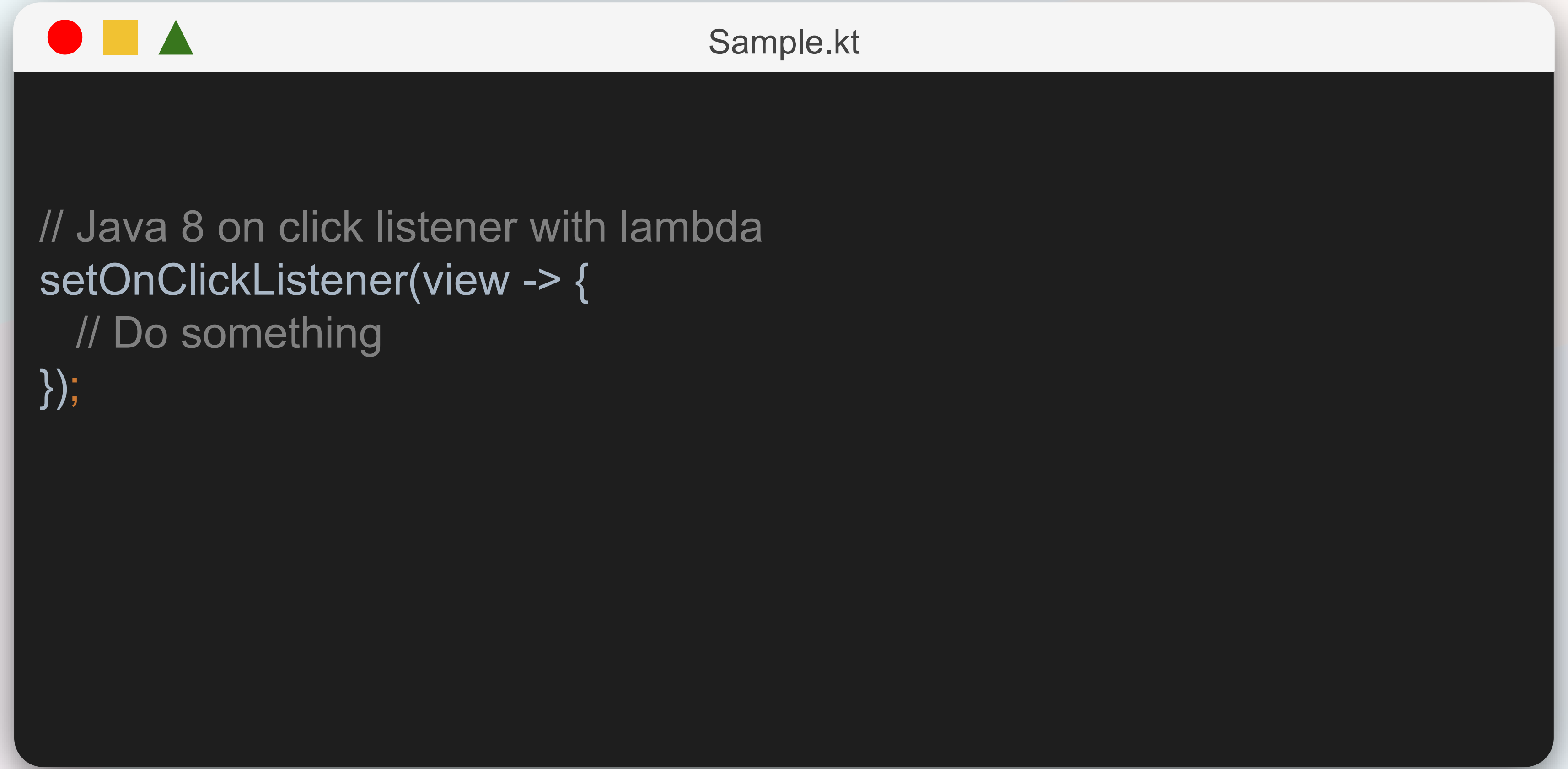
```
fun outside() {
   fun inside() {
      // Do something
   }
   // ...
   inside() // we can call in this scope
   // ...
}
```

Higher-Order Functions & Lambda



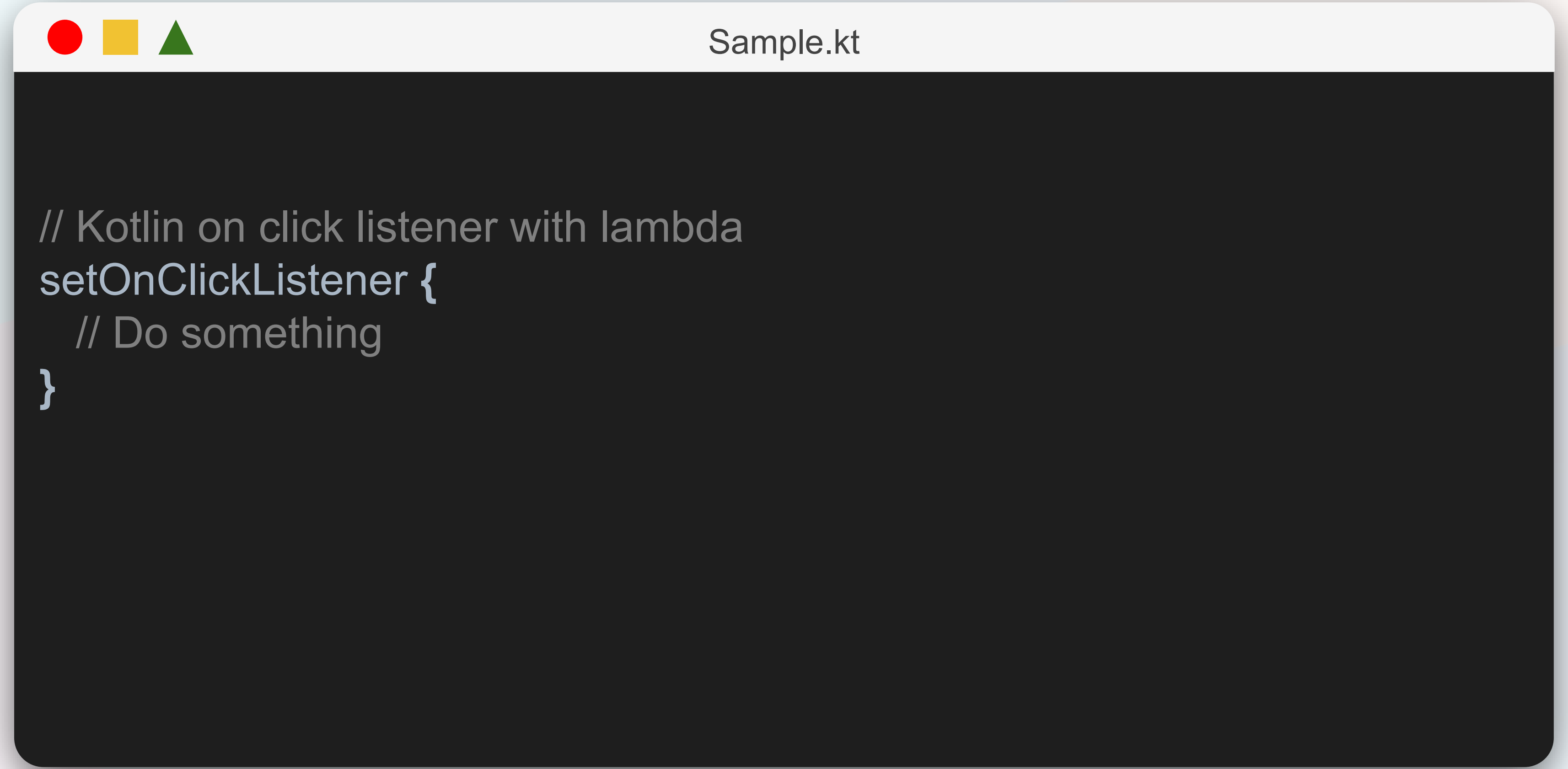
Sample.kt

```
// Java on click listener
setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View view) {
      // Do something
   }
});
```



Sample.kt

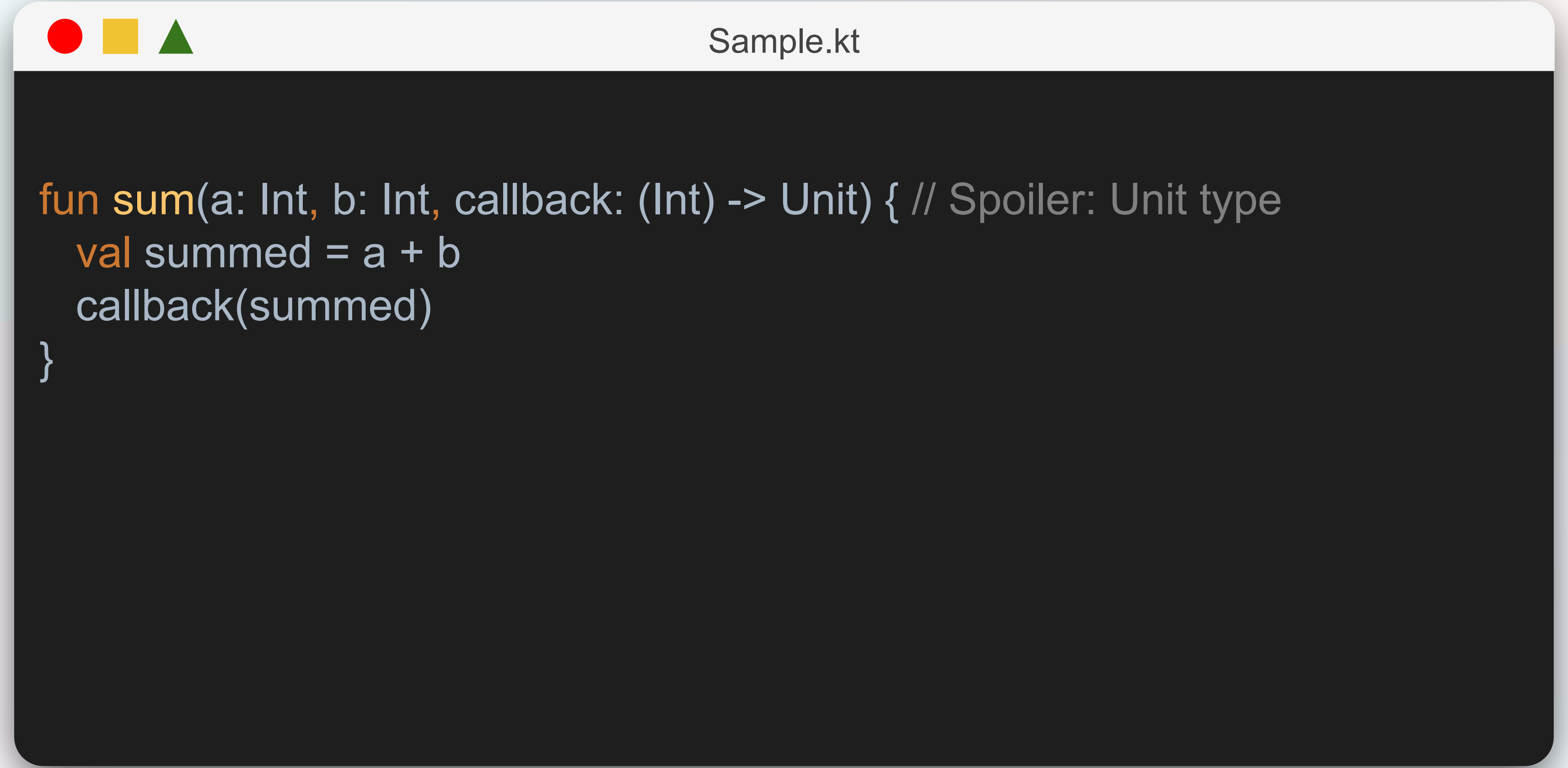
```
// Java 8 on click listener with lambda
setOnClickListener(view -> {
   // Do something
});
```



Sample.kt

```
// Kotlin on click listener with lambda
setOnClickListener {
  // Do something
}
```

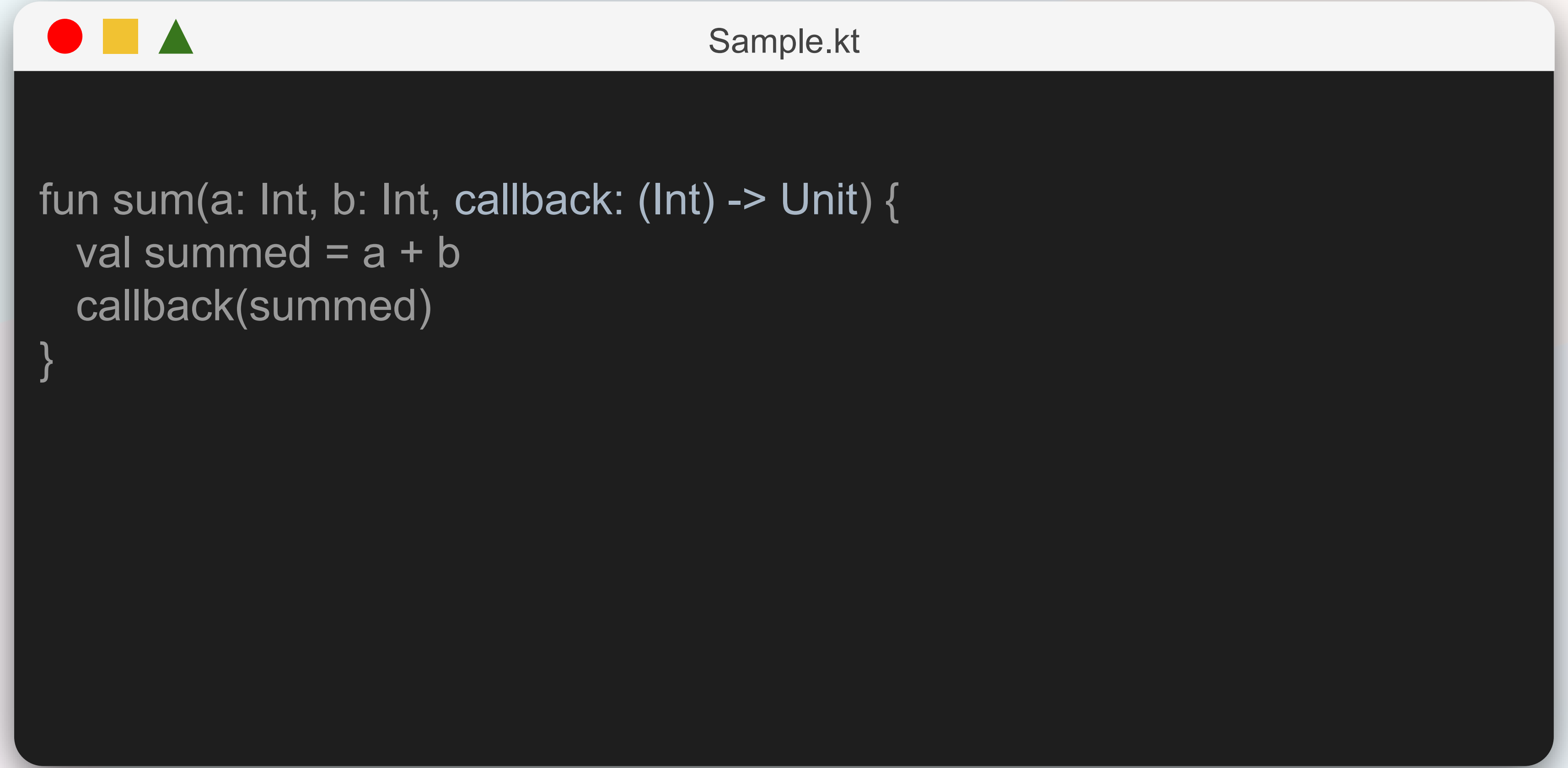
Higher-Order Functions & Lambda



Sample.kt

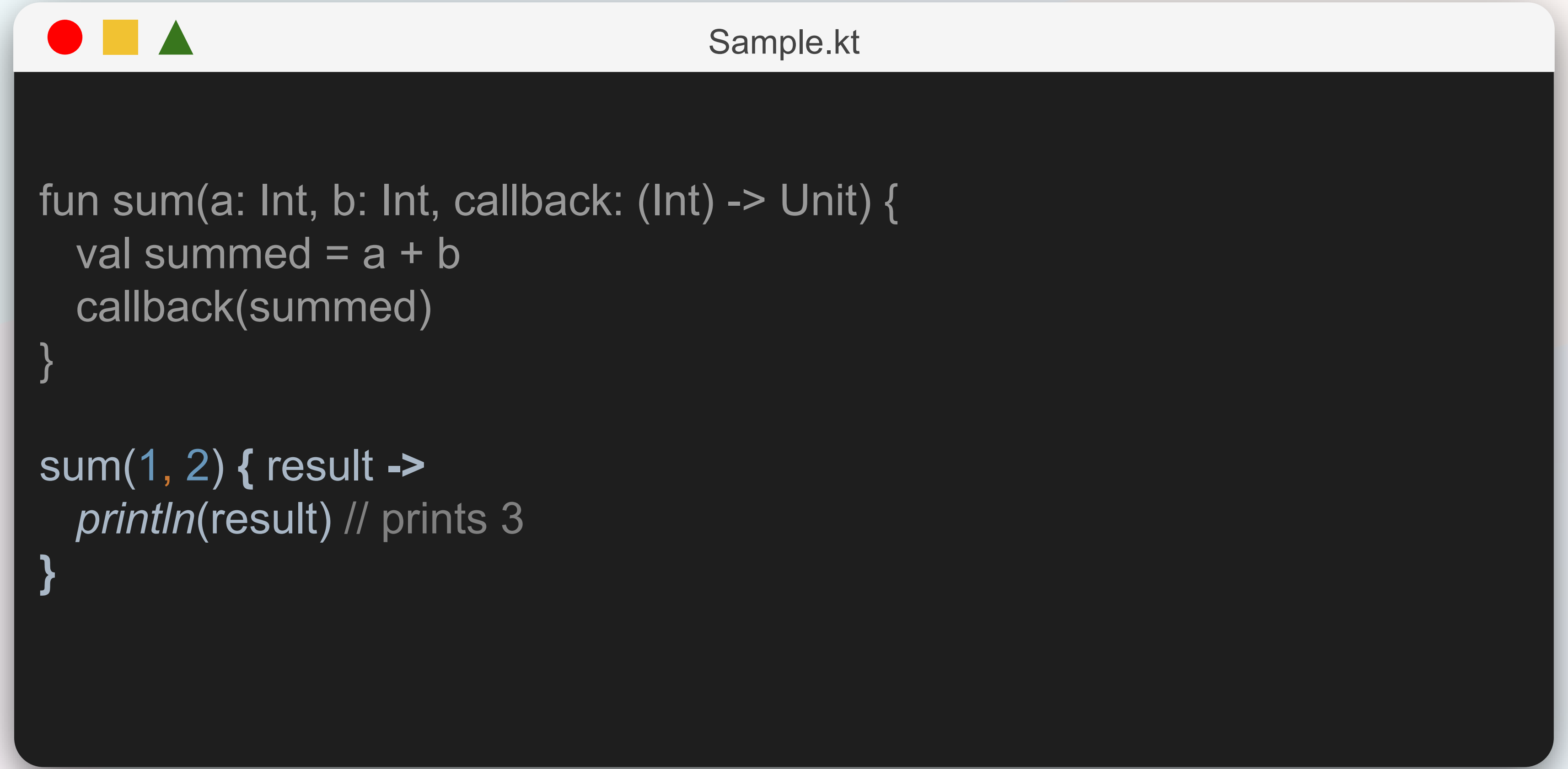
```kotlin
fun sum(a: Int, b: Int, callback: (Int) -> Unit) { // Spoiler: Unit type
    val summed = a + b
    callback(summed)
}
```

Higher-Order Functions & Lambda

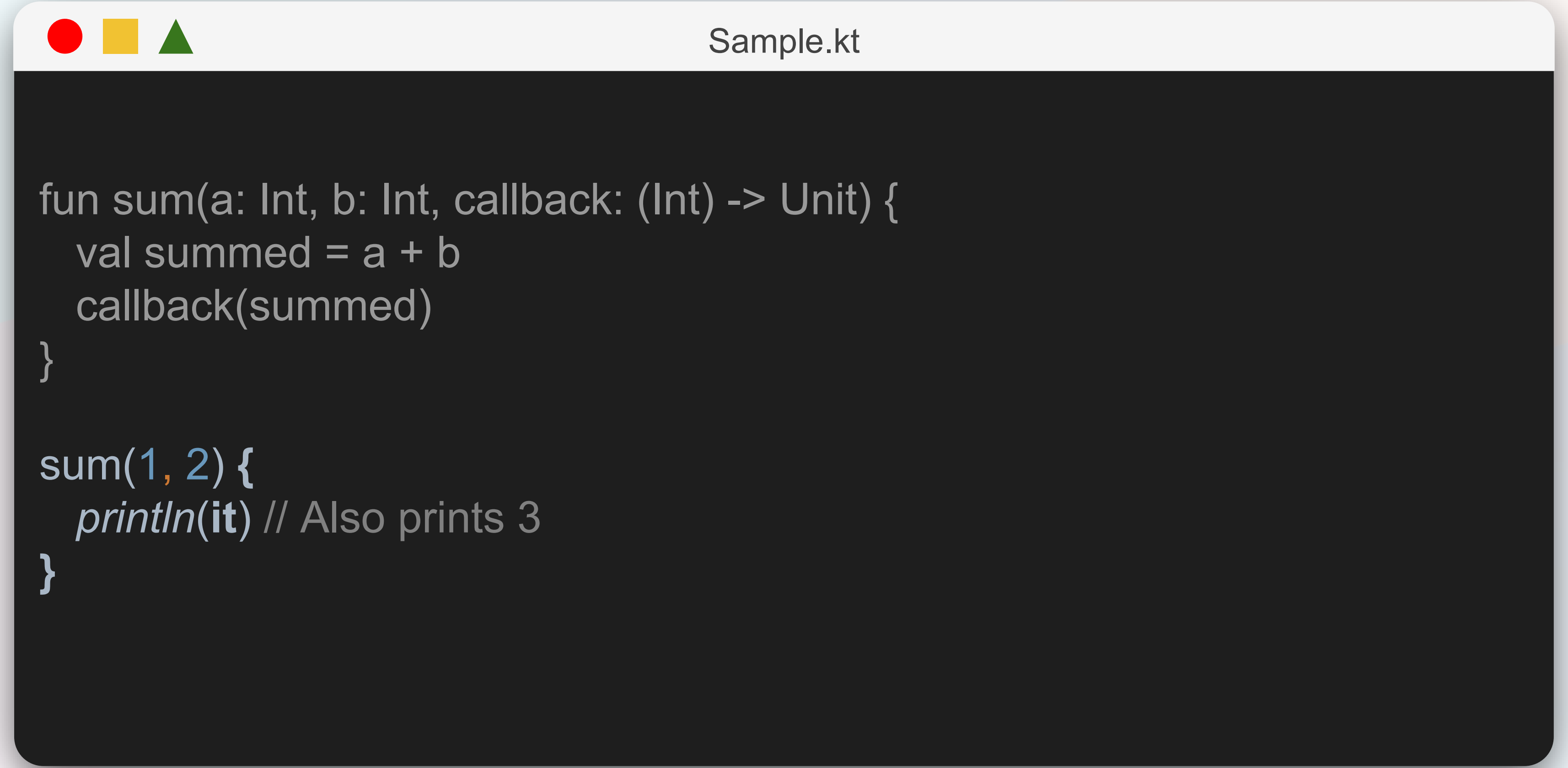

Sample.kt

```
fun sum(a: Int, b: Int, callback: (Int) -> Unit) {
    val summed = a + b
    callback(summed)
}
```

Higher-Order Functions & Lambda

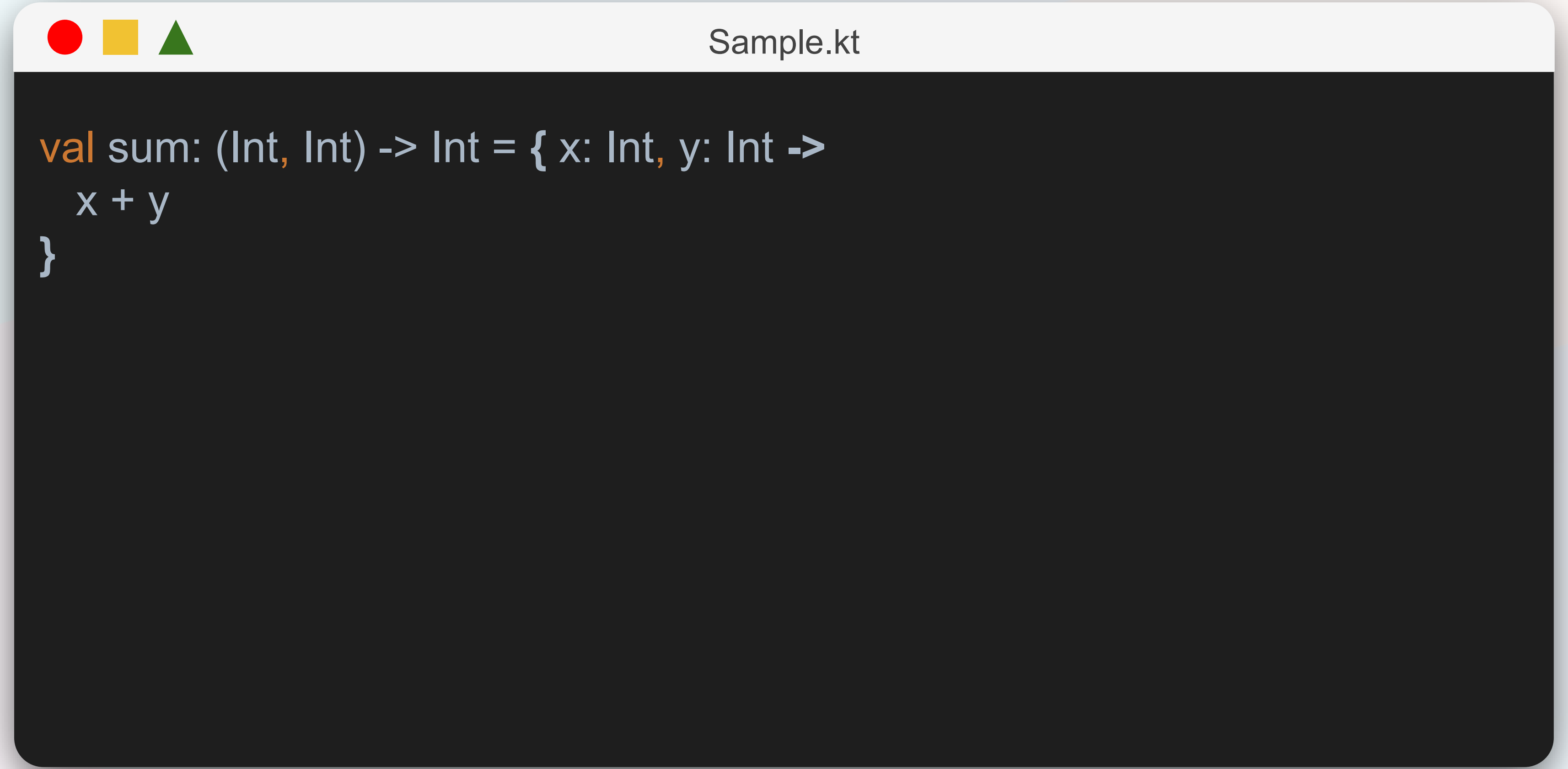

Sample.kt

```kotlin
fun sum(a: Int, b: Int, callback: (Int) -> Unit) {
    val summed = a + b
    callback(summed)
}

sum(1, 2) { result ->
    println(result) // prints 3
}
```

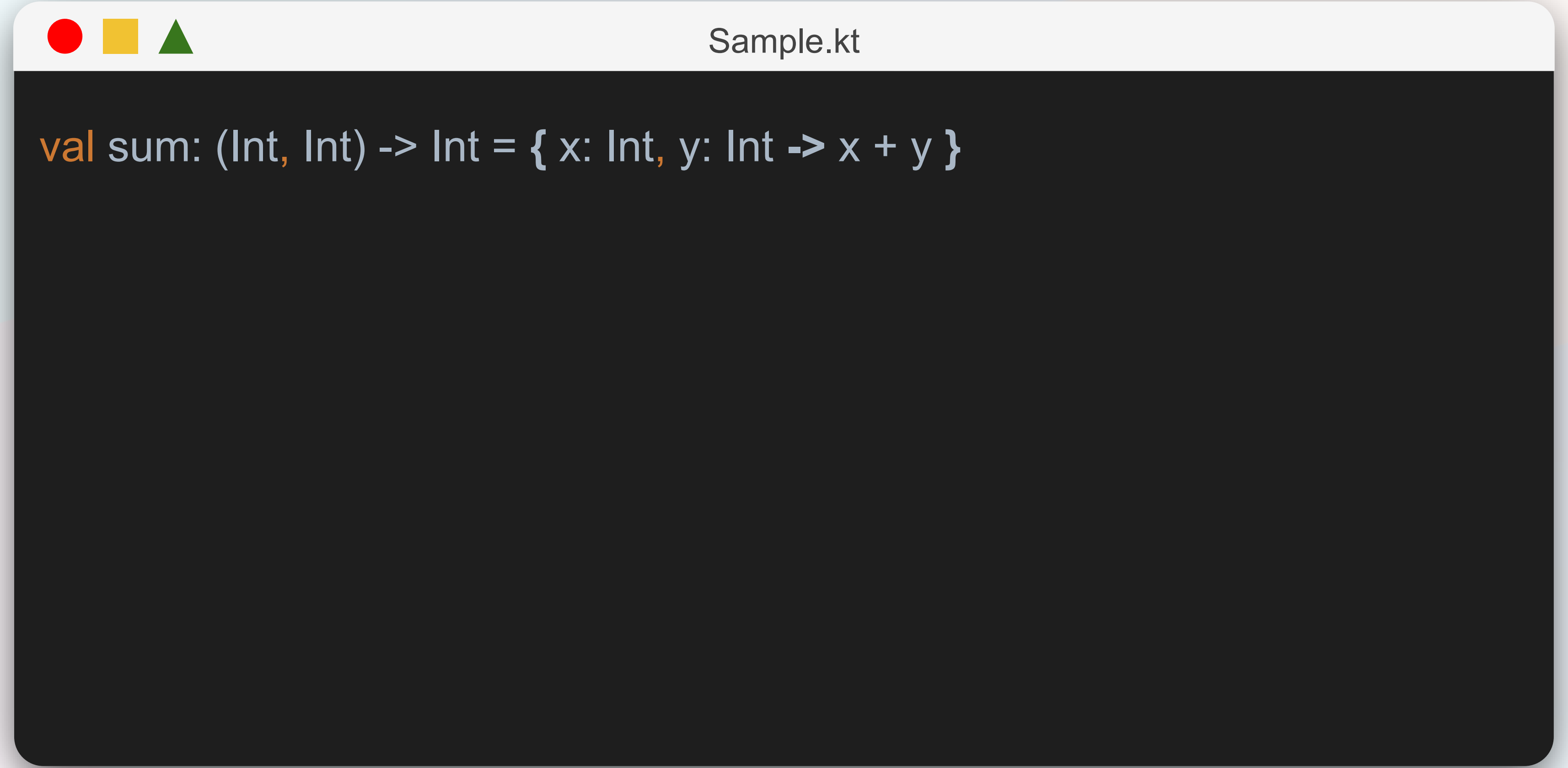


Sample.kt

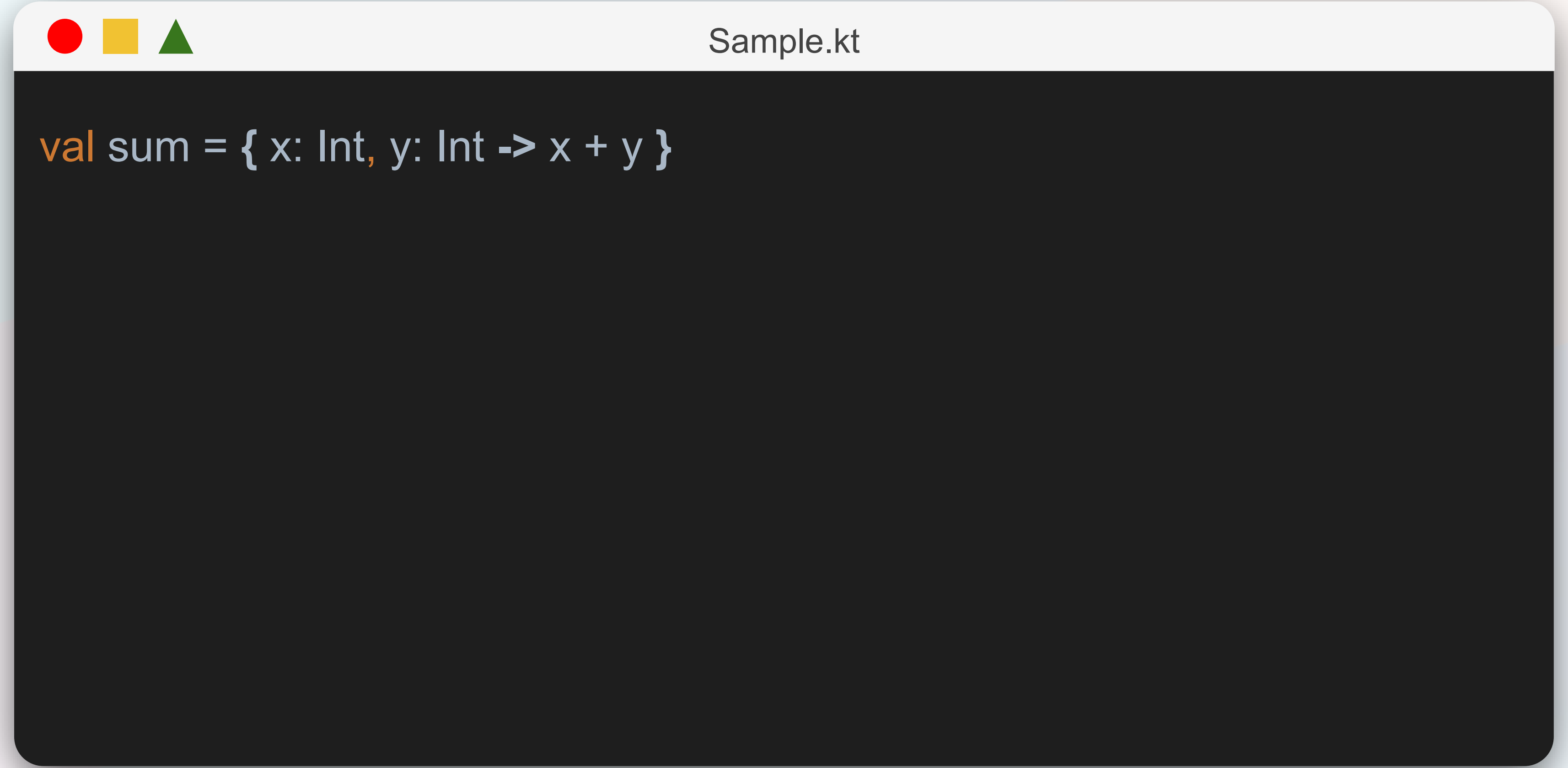
```kotlin
fun sum(a: Int, b: Int, callback: (Int) -> Unit) {
   val summed = a + b
   callback(summed)
}

sum(1, 2) {
   println(it) // Also prints 3
}
```


Sample.kt

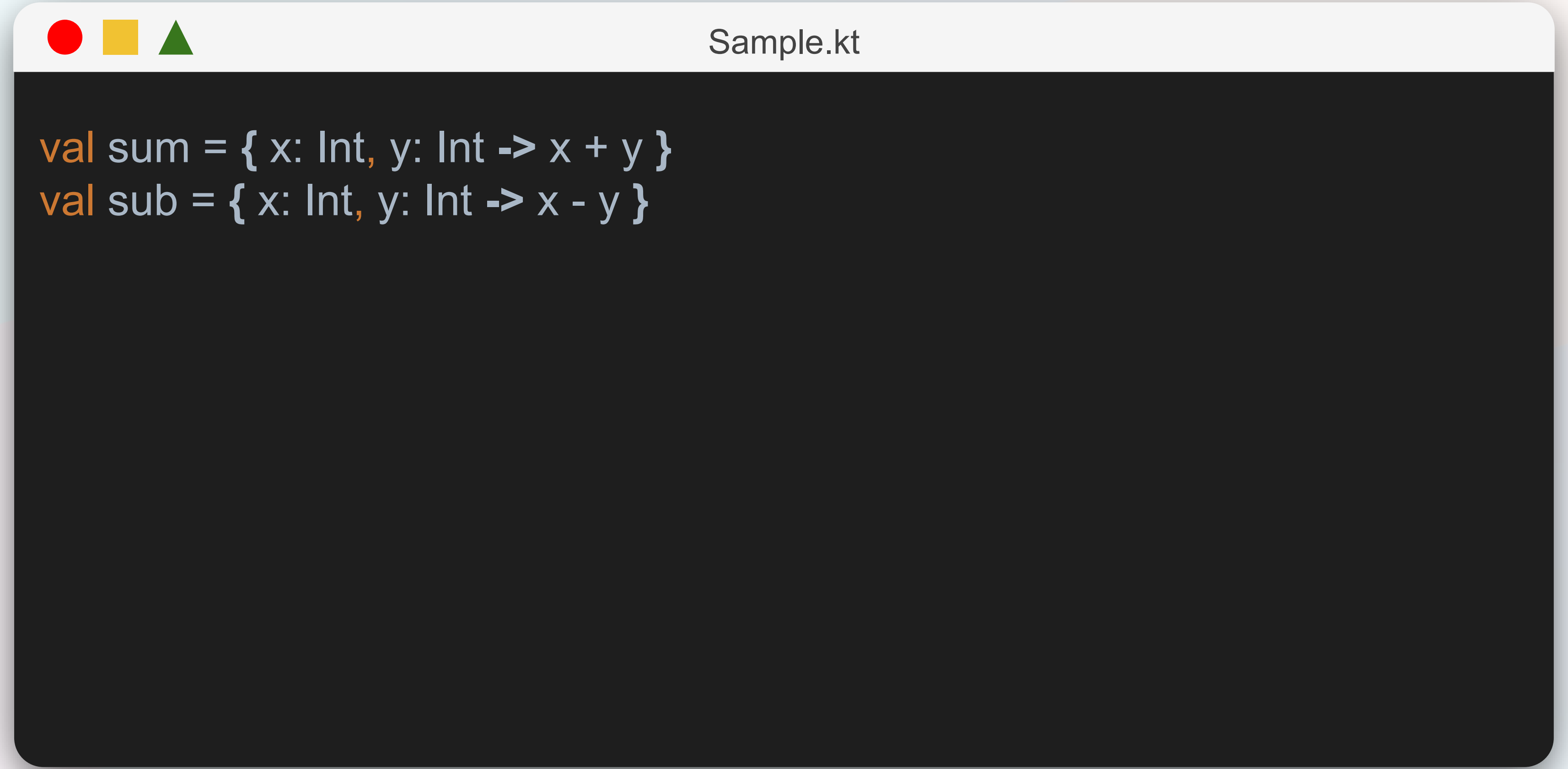
```
val sum: (Int, Int) -> Int = { x: Int, y: Int ->
  x + y
}
```



```kotlin
val sum: (Int, Int) -> Int = { x: Int, y: Int -> x + y }
```


Sample.kt

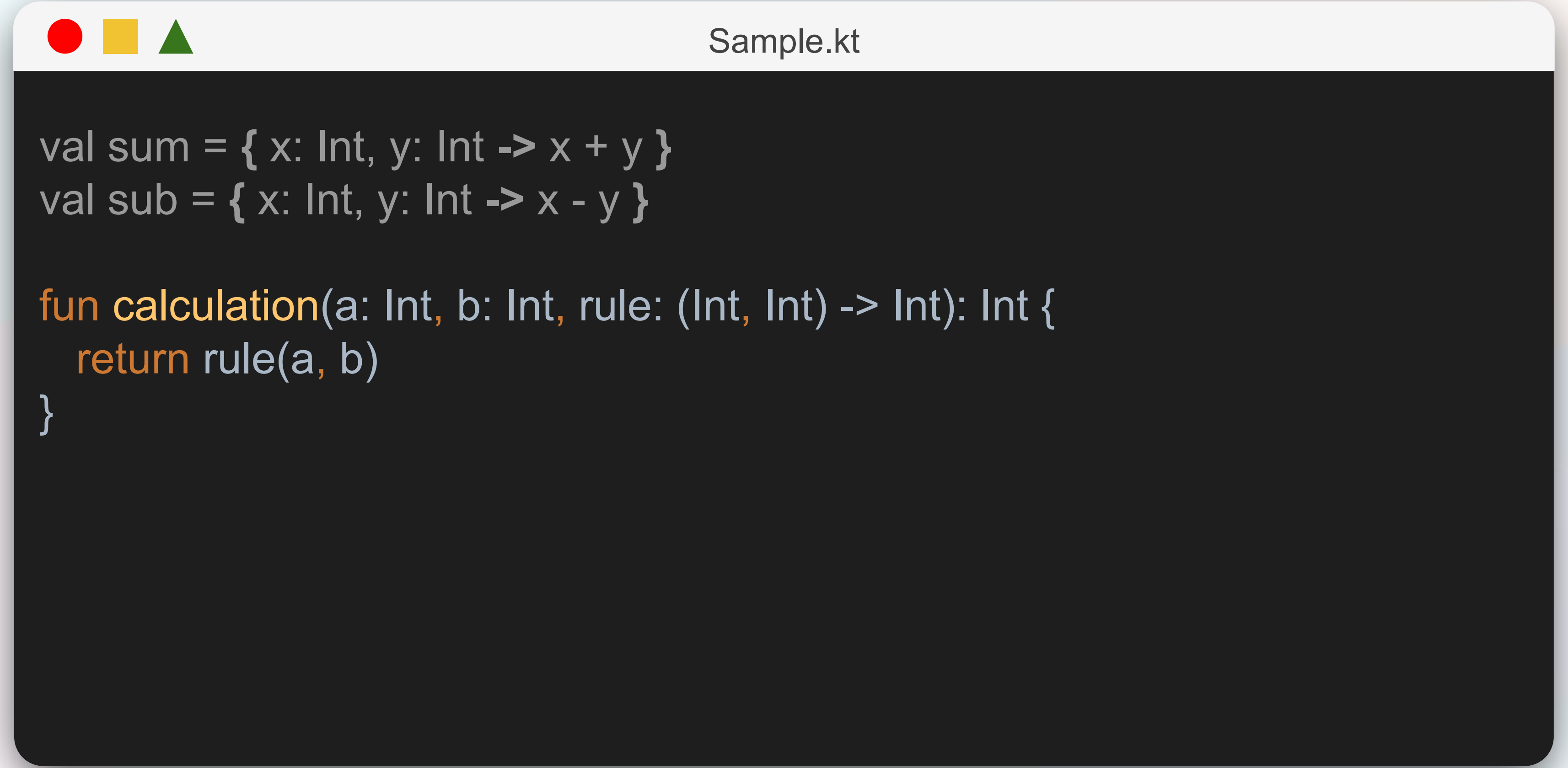
```
val sum = { x: Int, y: Int -> x + y }
```



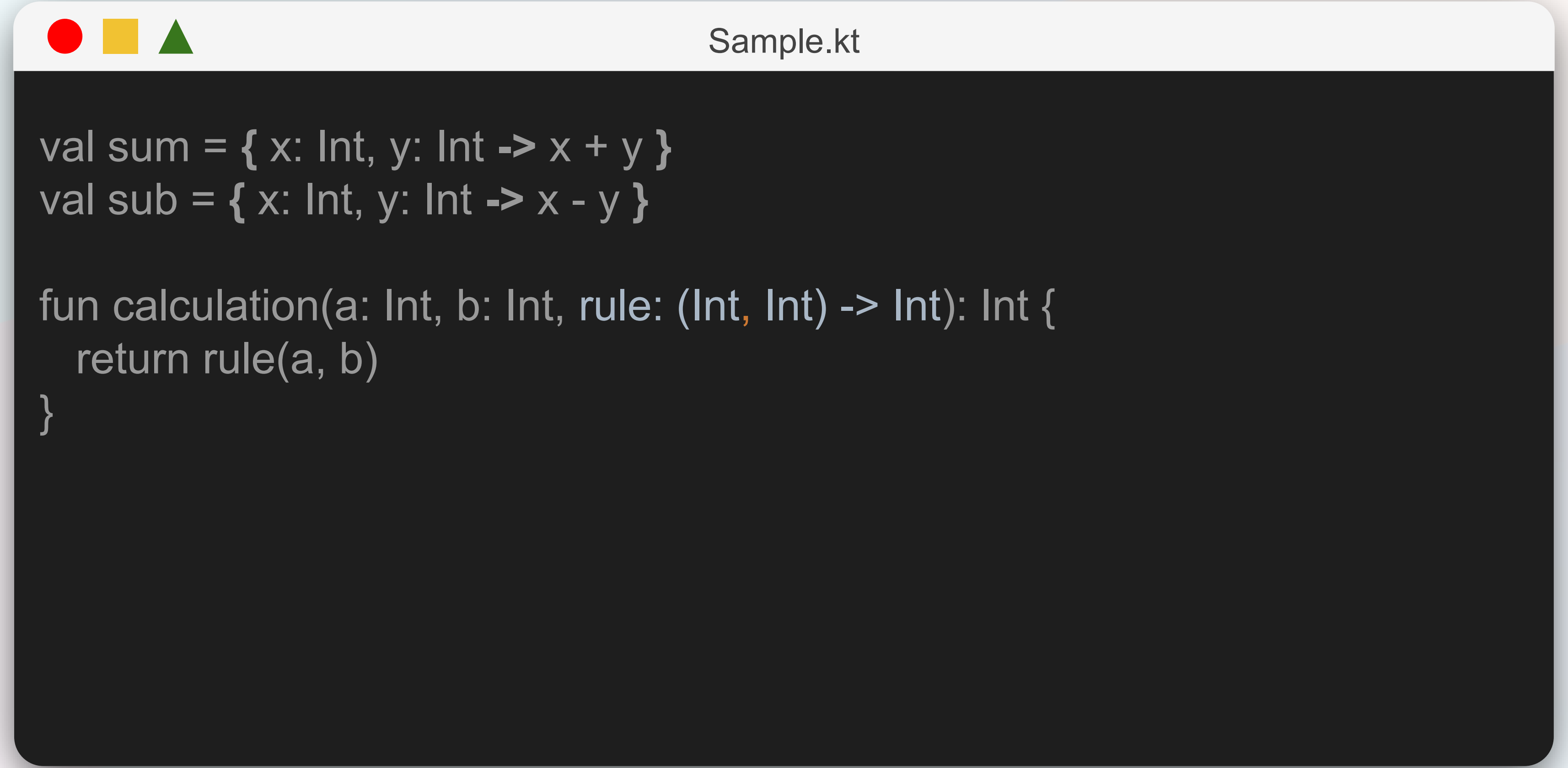
Sample.kt

```kotlin
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }
```

Higher-Order Functions & Lambda

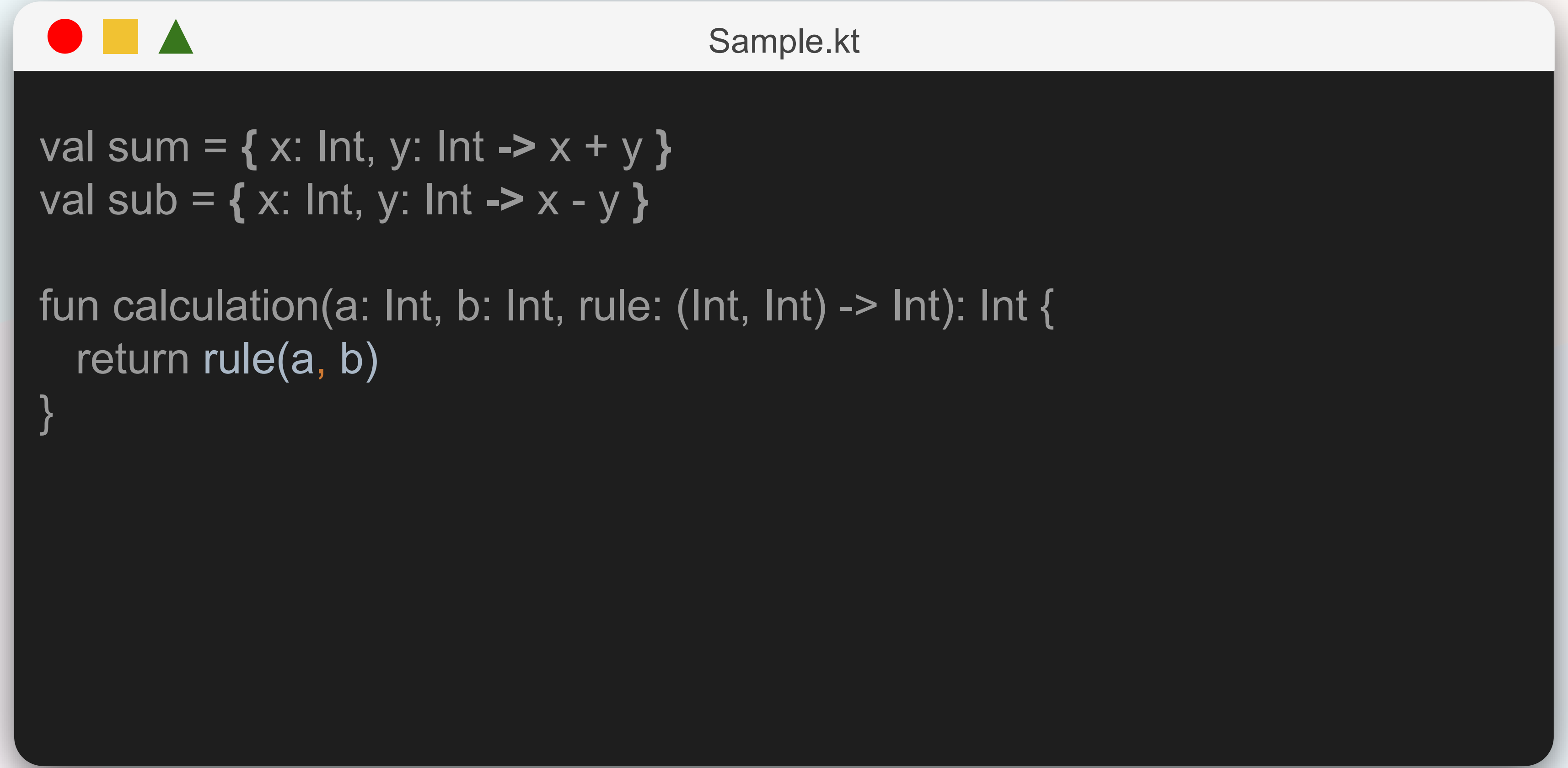

Sample.kt

```
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
   return rule(a, b)
}
```



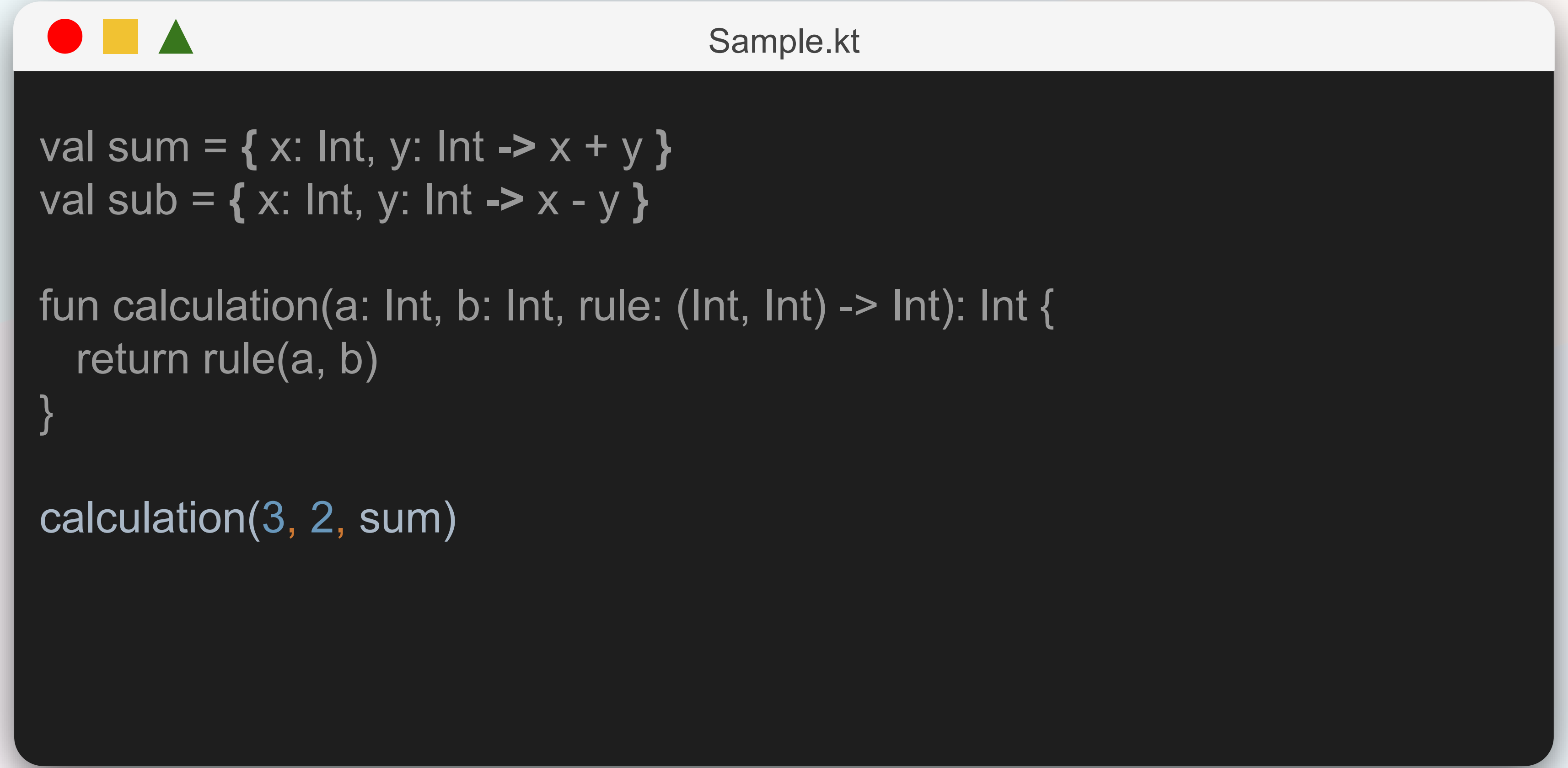
Sample.kt

```
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
   return rule(a, b)
}
```



Sample.kt

```
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
    return rule(a, b)
}
```


Sample.kt

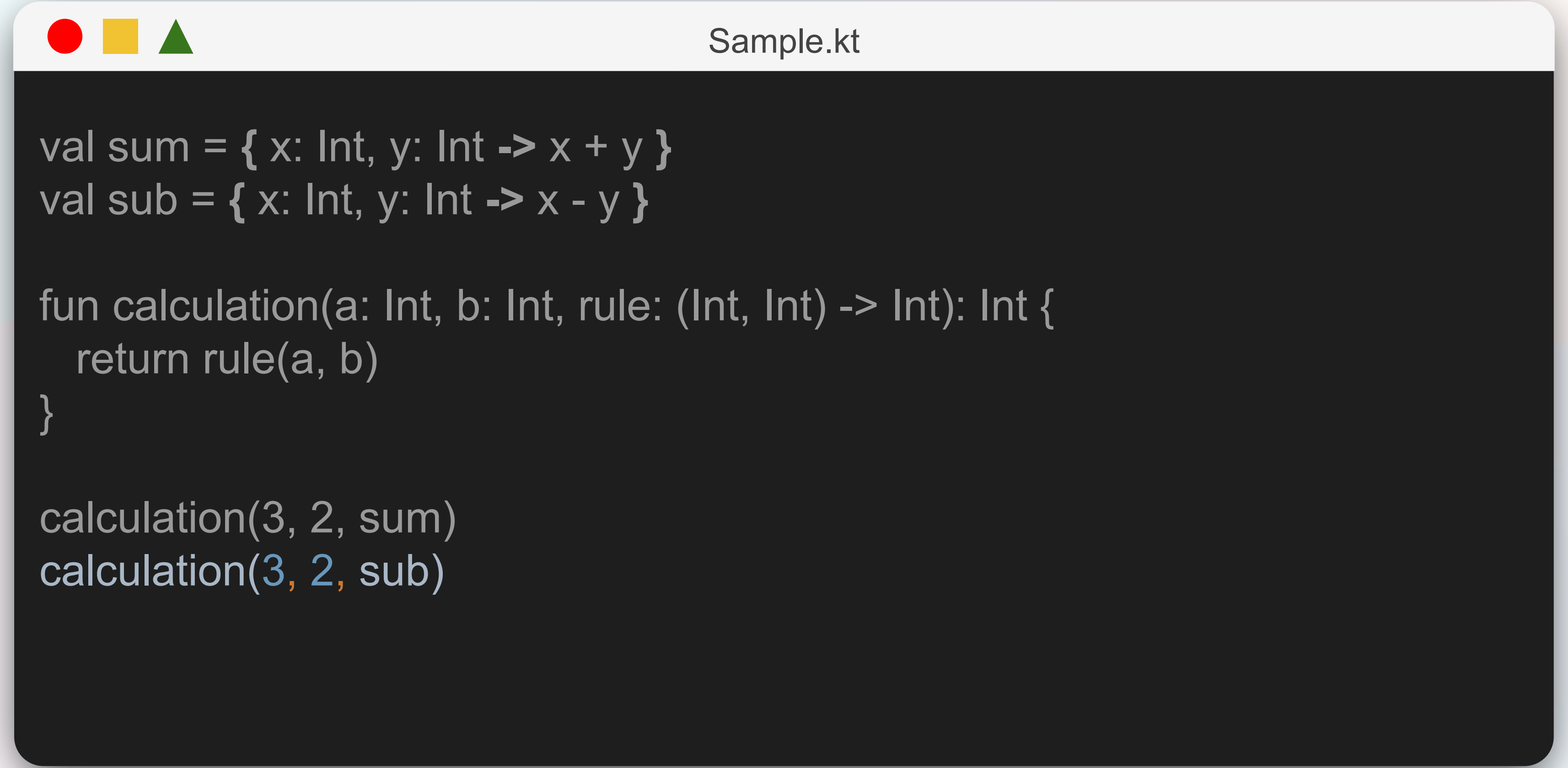
```kotlin
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
   return rule(a, b)
}

calculation(3, 2, sum)
```

Higher-Order Functions & Lambda

Sample.kt

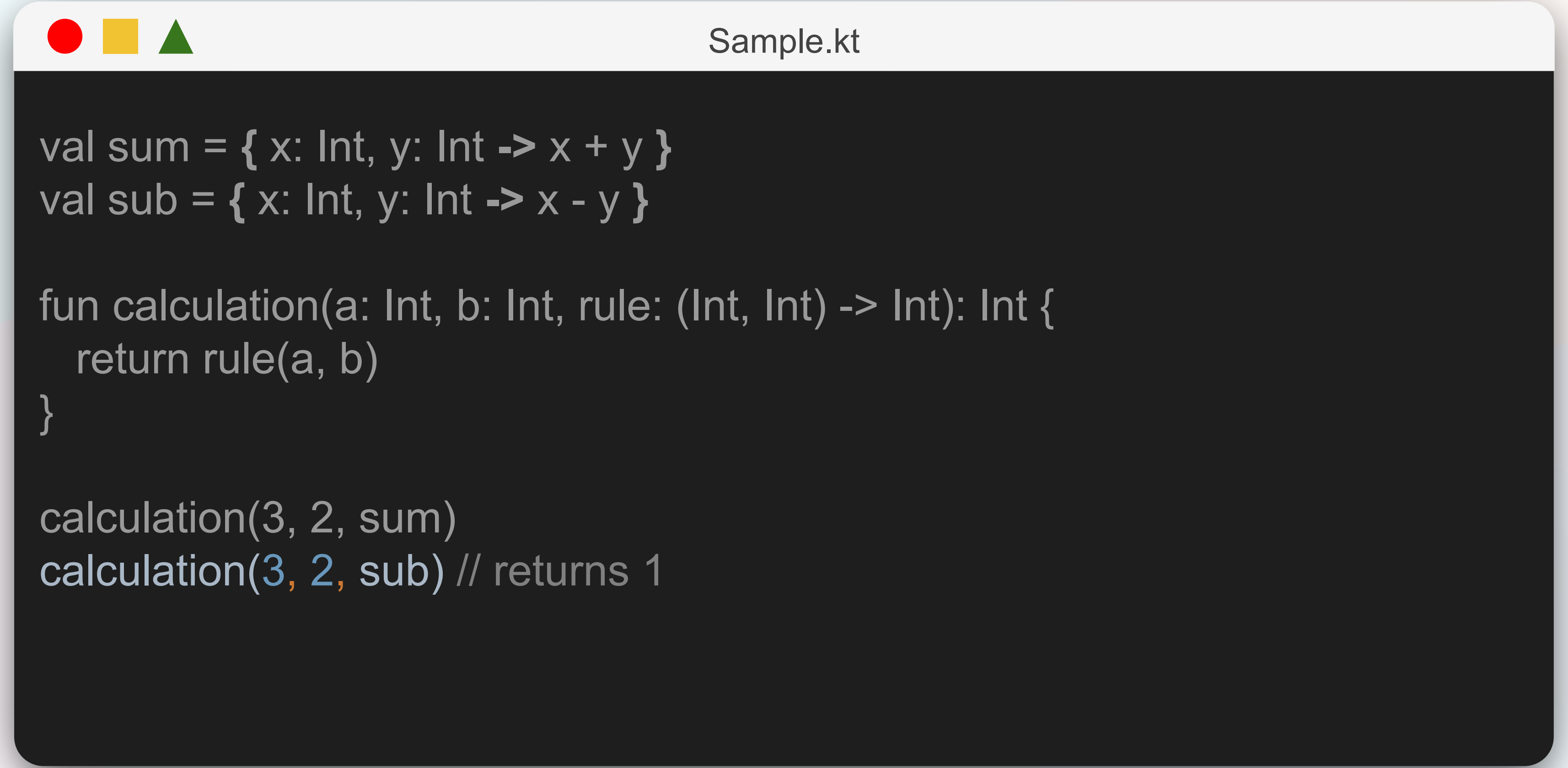
```kotlin
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
   return rule(a, b)
}

calculation(3, 2, sum) // returns 5
```

Higher-Order Functions & Lambda

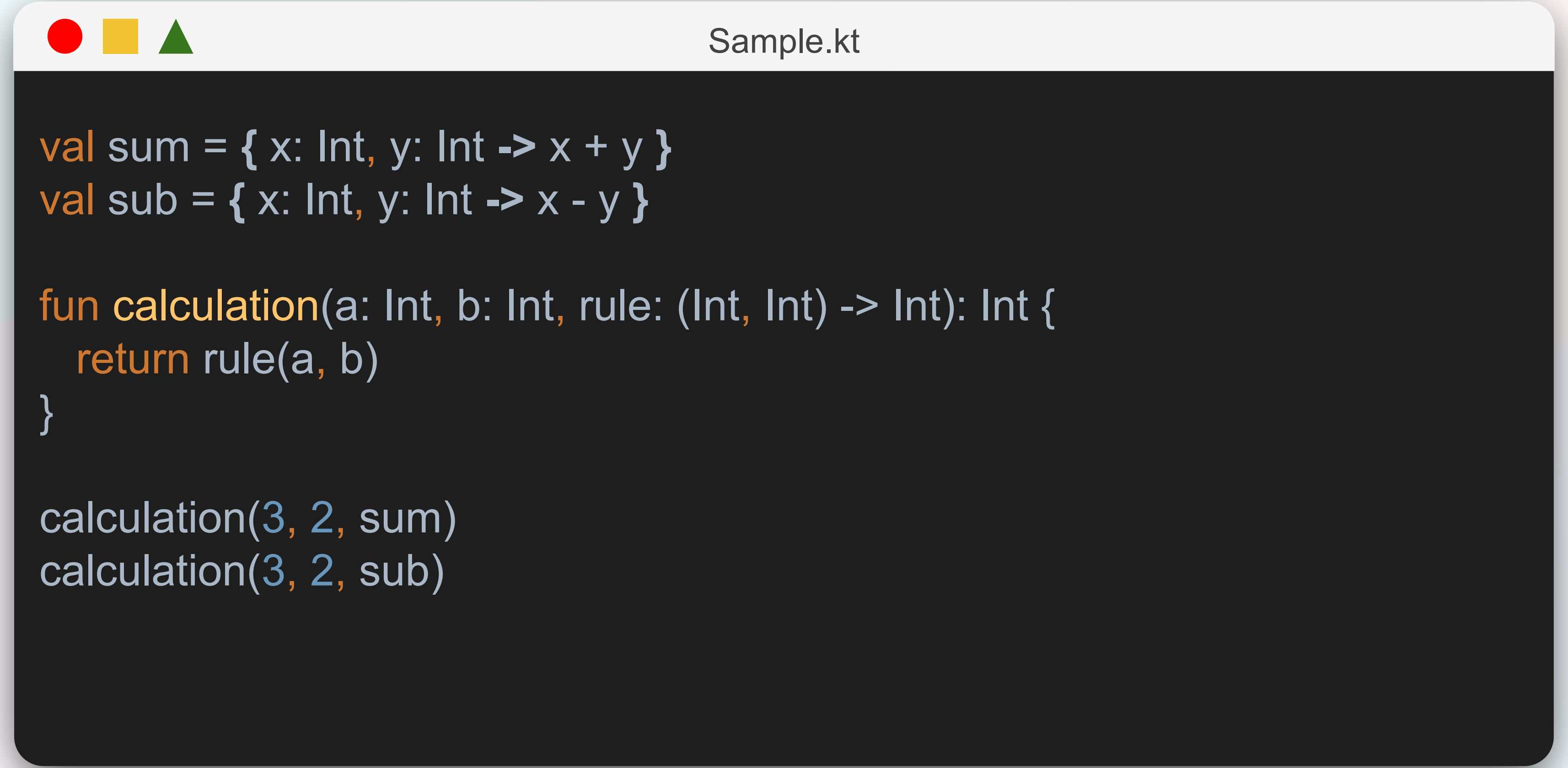

Sample.kt

```kotlin
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
    return rule(a, b)
}

calculation(3, 2, sum)
calculation(3, 2, sub)
```

Higher-Order Functions & Lambda

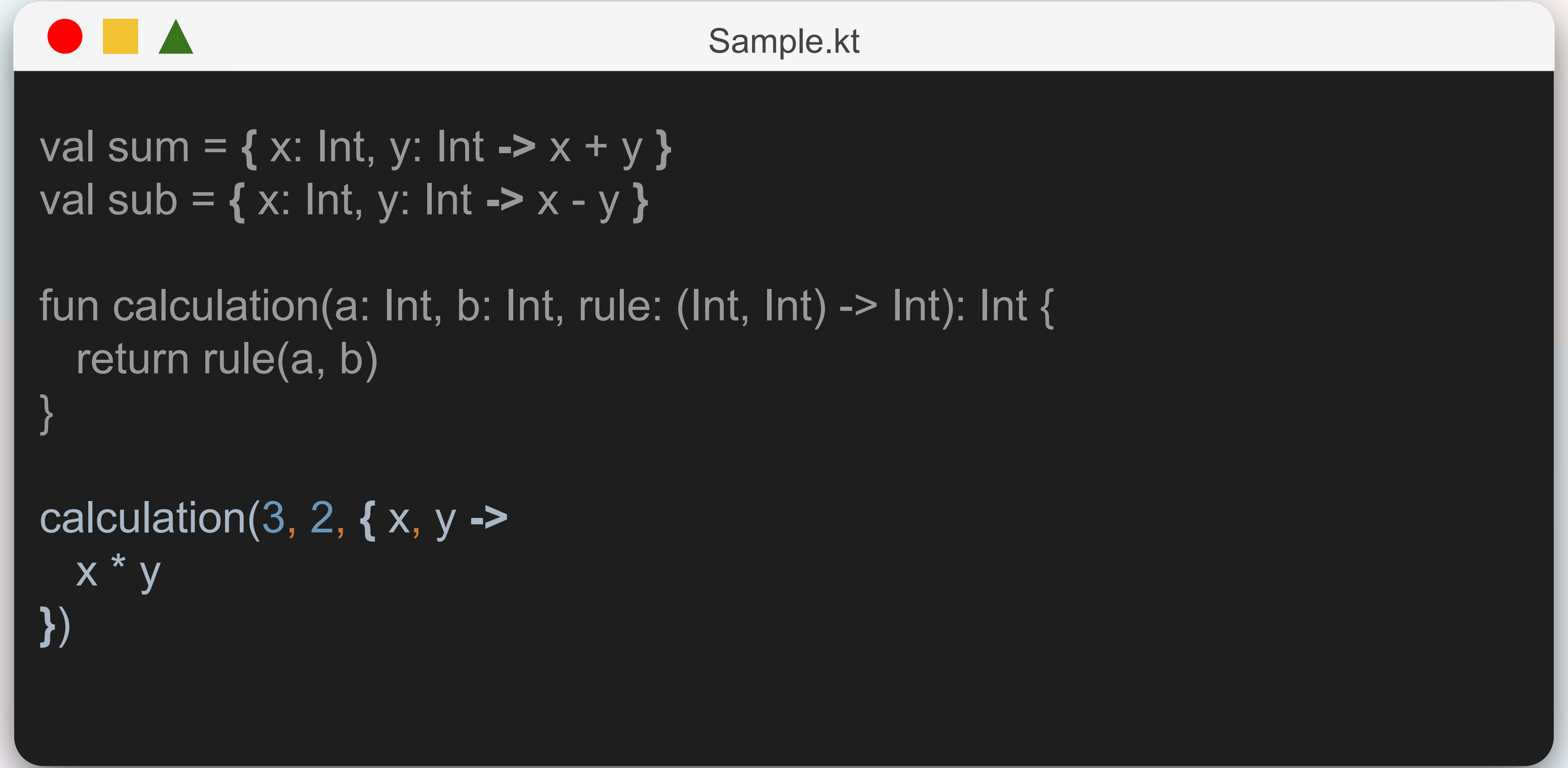

Sample.kt

```
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
    return rule(a, b)
}

calculation(3, 2, sum)
calculation(3, 2, sub) // returns 1
```



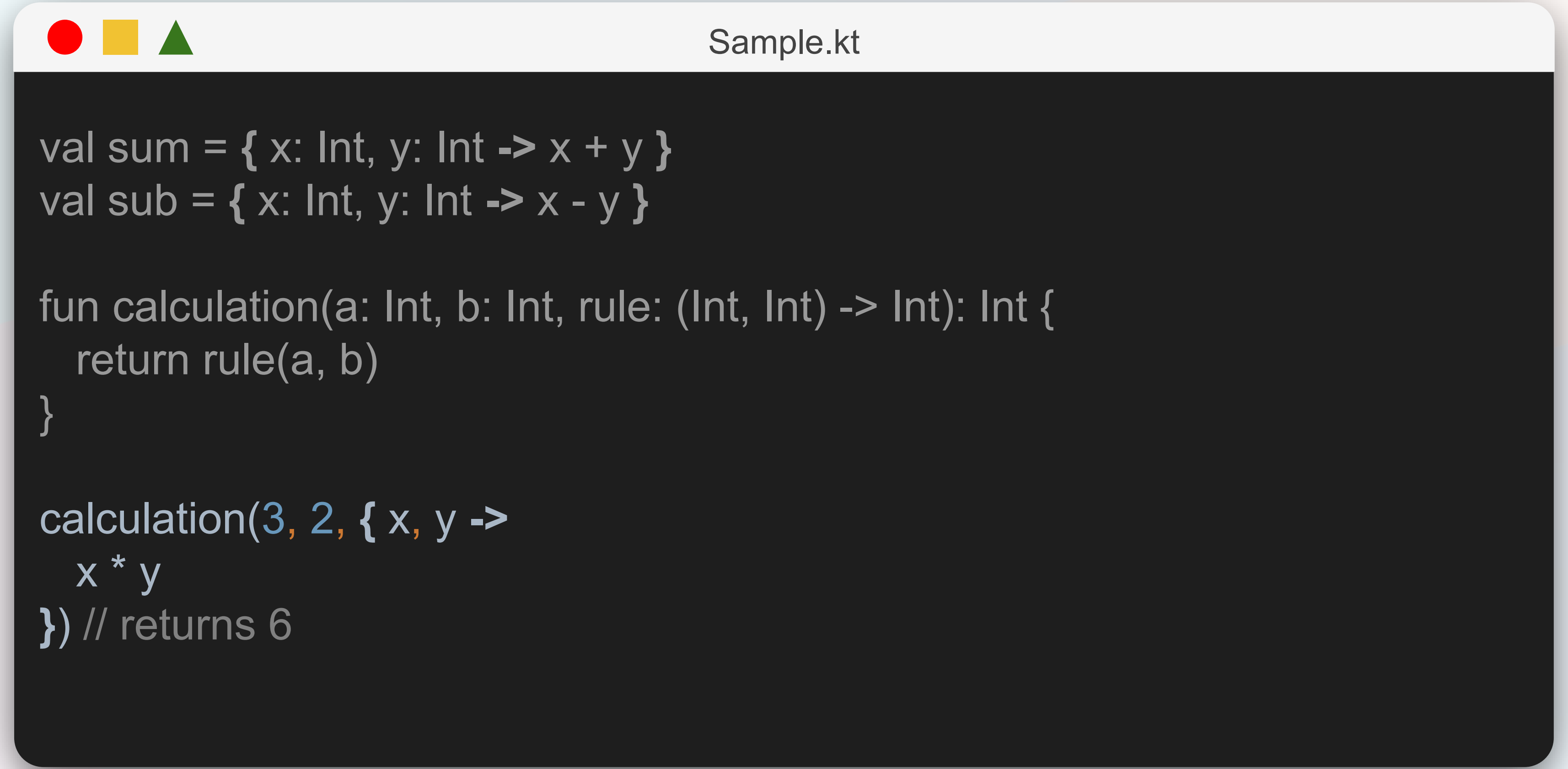
Sample.kt

```kotlin
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
   return rule(a, b)
}

calculation(3, 2, sum)
calculation(3, 2, sub)
```


Sample.kt

```
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
   return rule(a, b)
}

calculation(3, 2, { x, y ->
   x * y
})
```

Higher-Order Functions & Lambda

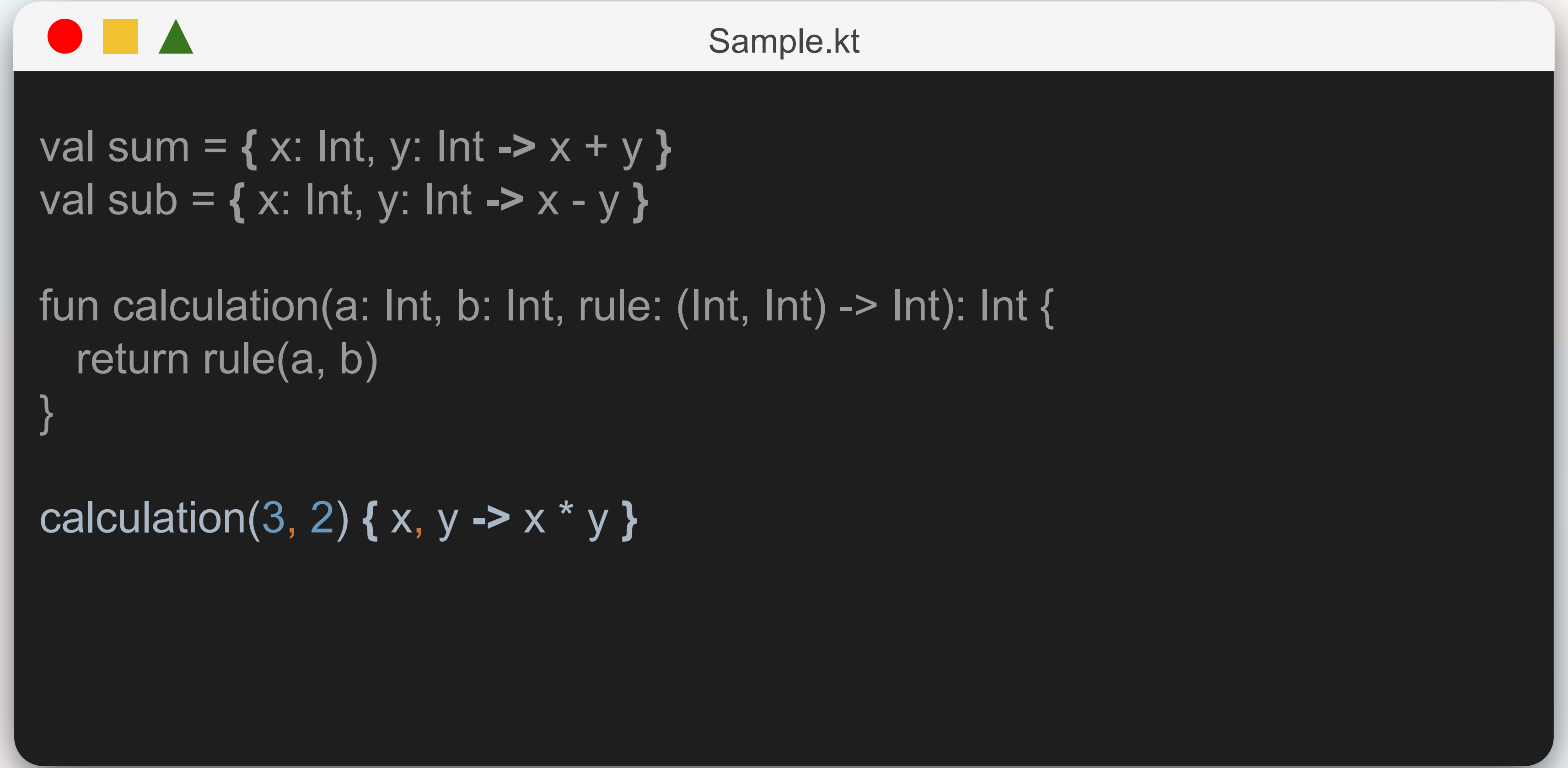

Sample.kt

```kotlin
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
    return rule(a, b)
}

calculation(3, 2, { x, y ->
    x * y
}) // returns 6
```

Higher-Order Functions & Lambda

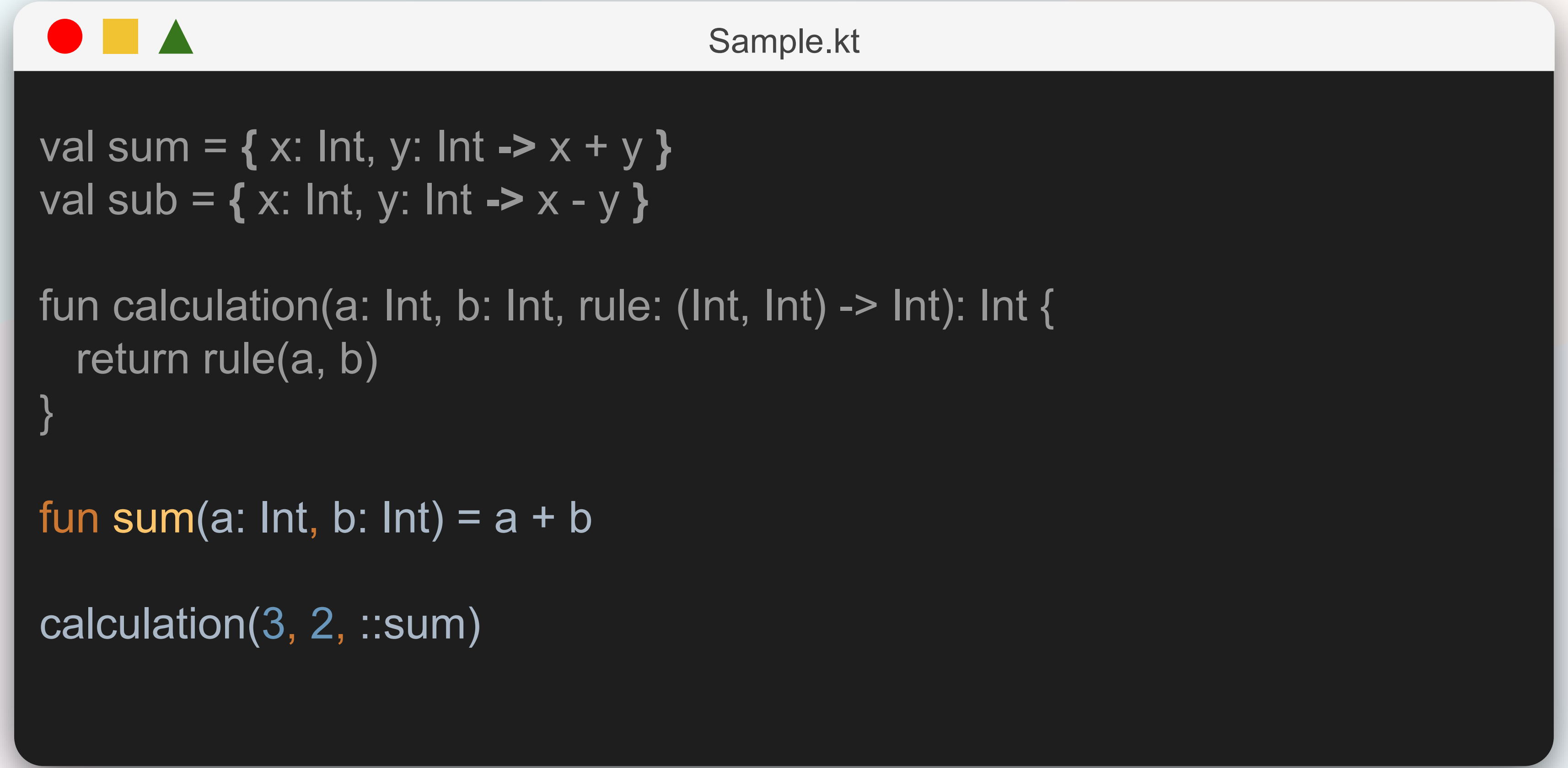
Sample.kt

```kotlin
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
    return rule(a, b)
}

calculation(3, 2) { x, y ->
    x * y
}
```



Sample.kt

```
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
    return rule(a, b)
}

calculation(3, 2) { x, y -> x * y }
```



Sample.kt

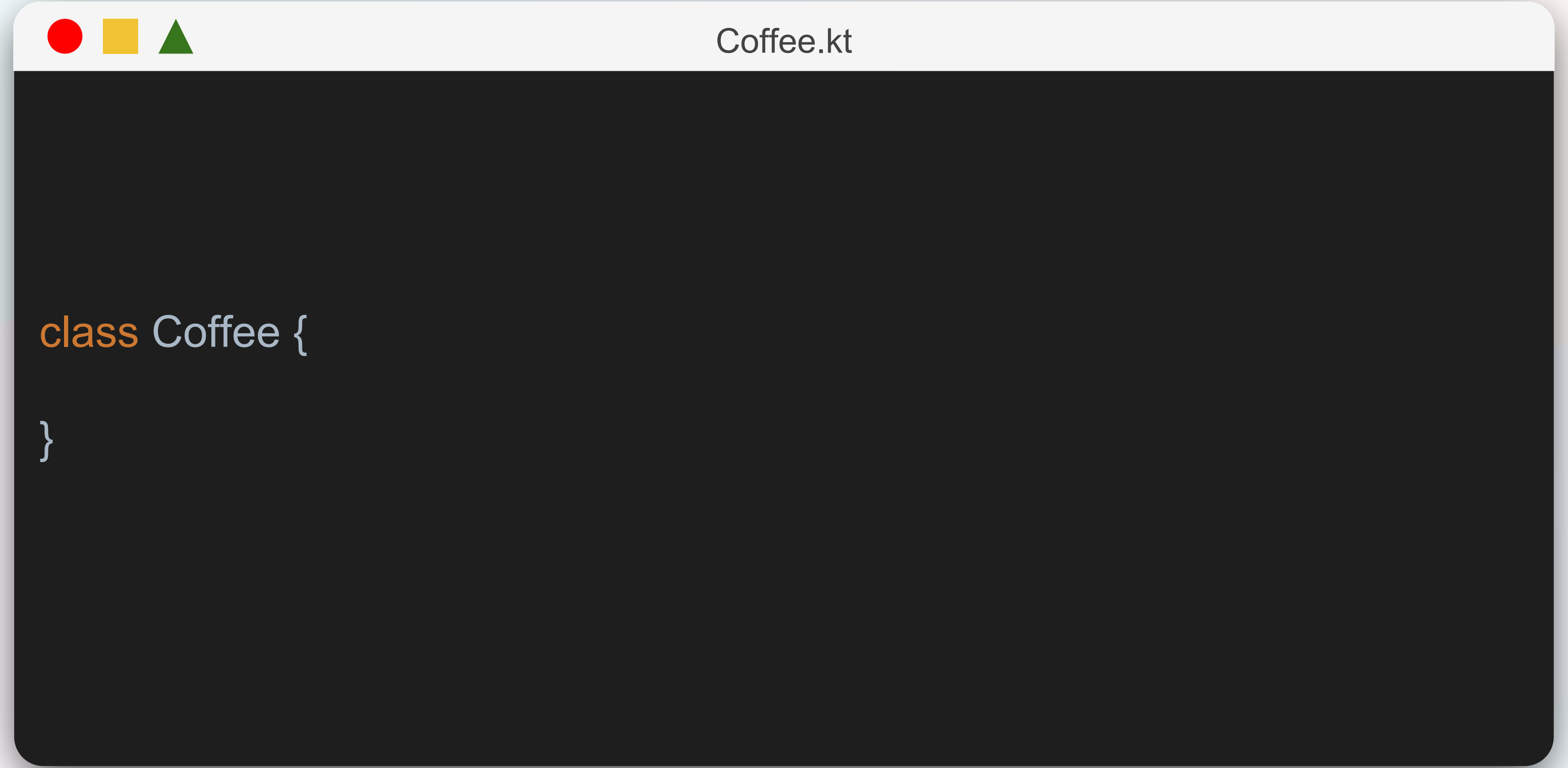
```kotlin
val sum = { x: Int, y: Int -> x + y }
val sub = { x: Int, y: Int -> x - y }

fun calculation(a: Int, b: Int, rule: (Int, Int) -> Int): Int {
   return rule(a, b)
}

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

calculation(3, 2, ::sum)
```

OOP

OOP

Classes

Classes



Coffee.kt

```kotlin
class Coffee {

}
```

Classes

Coffee.kt

```kotlin
class Coffee
```

Classes

Coffee.kt

```kotlin
class Coffee

val coffee = Coffee()
```

OOP

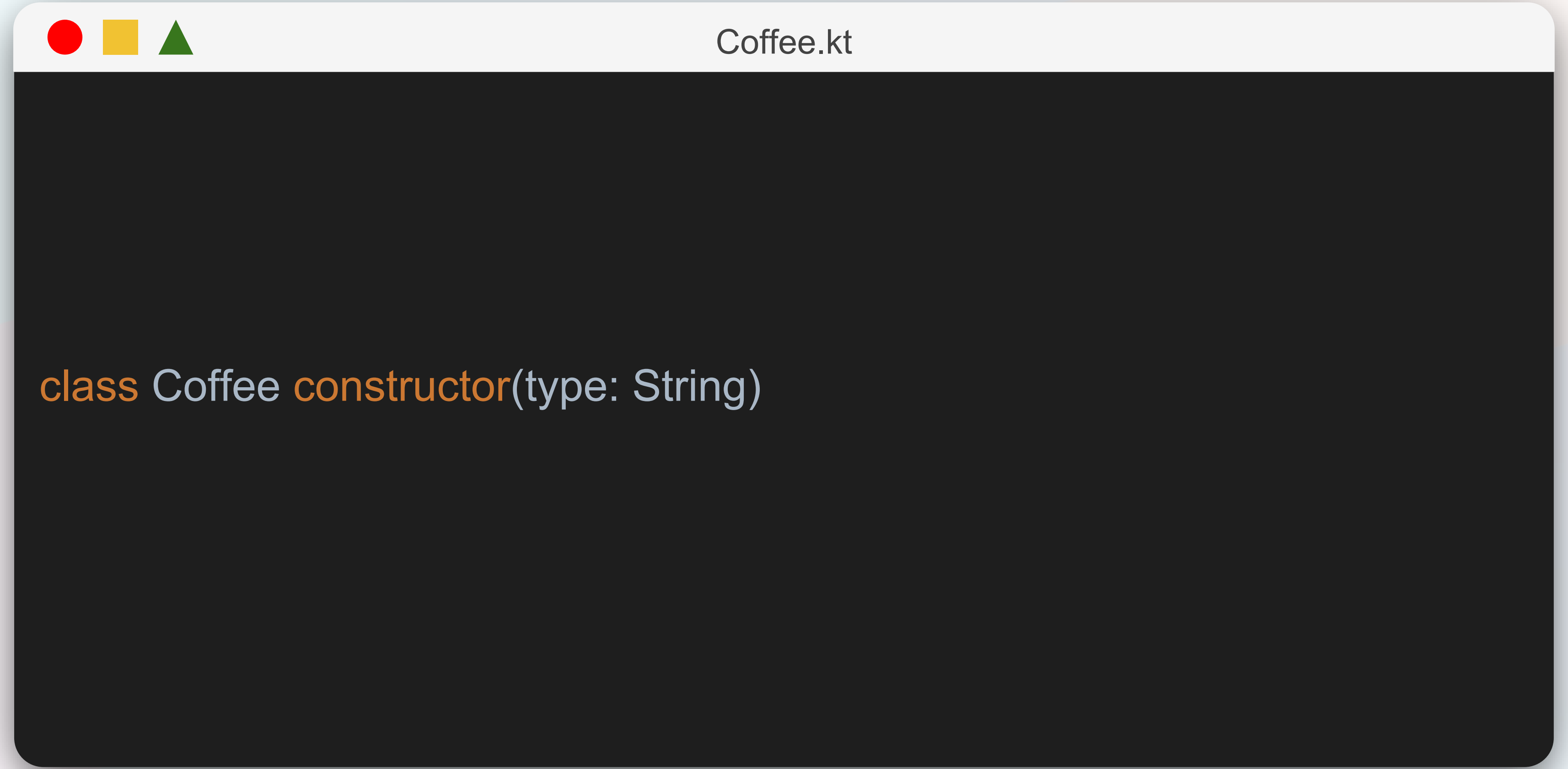
Constructor

Constructor

Coffee.kt

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

Constructor


Coffee.kt

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

Constructor

Coffee.kt

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

Constructor

CustomView.kt

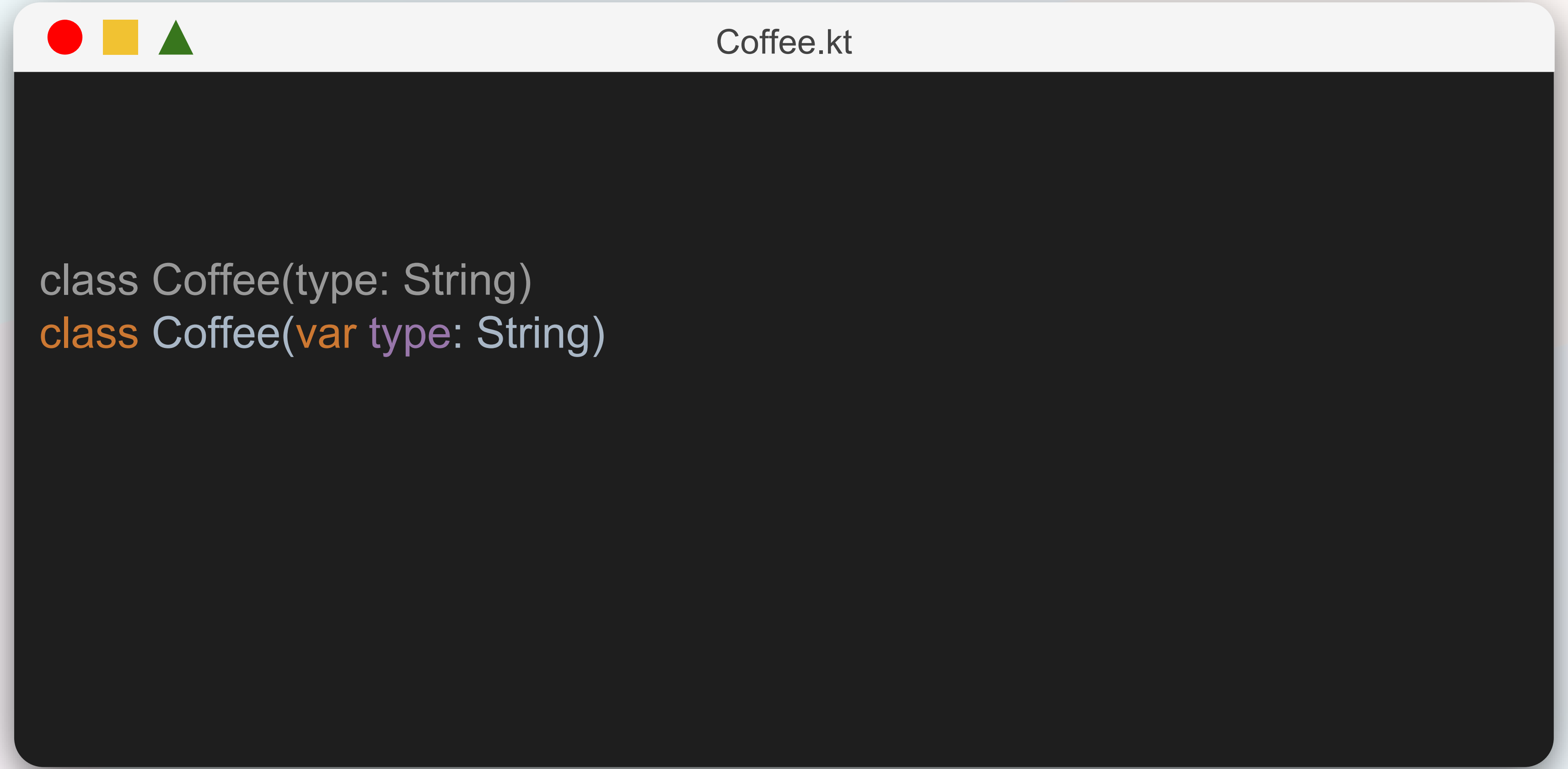
```kotlin
class CustomView : View { // Spoiler: Inheritance

    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) {}

}
```

Constructor

Coffee.kt

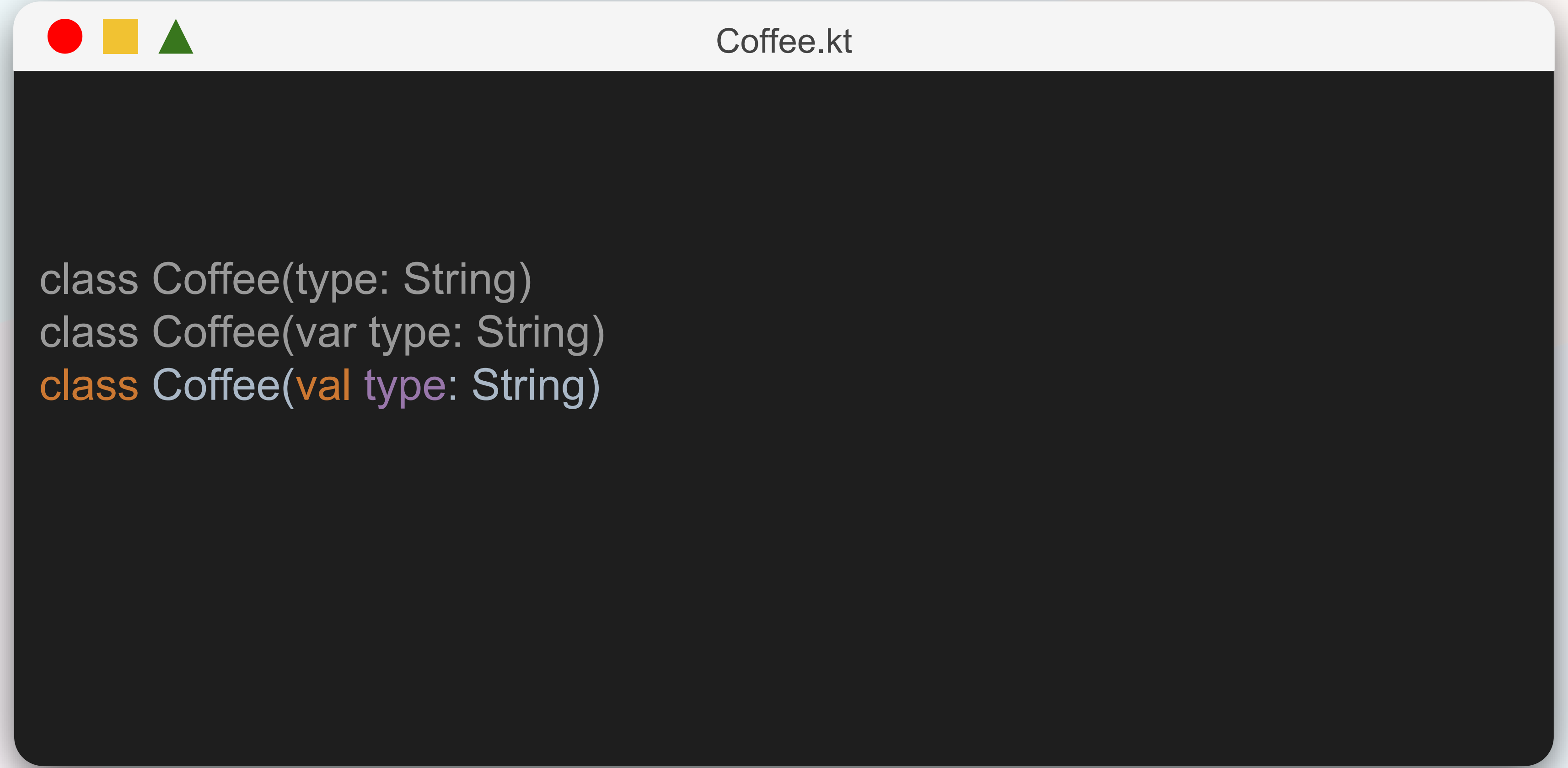
```kotlin
class Coffee(type: String)
```

Constructor


Coffee.kt

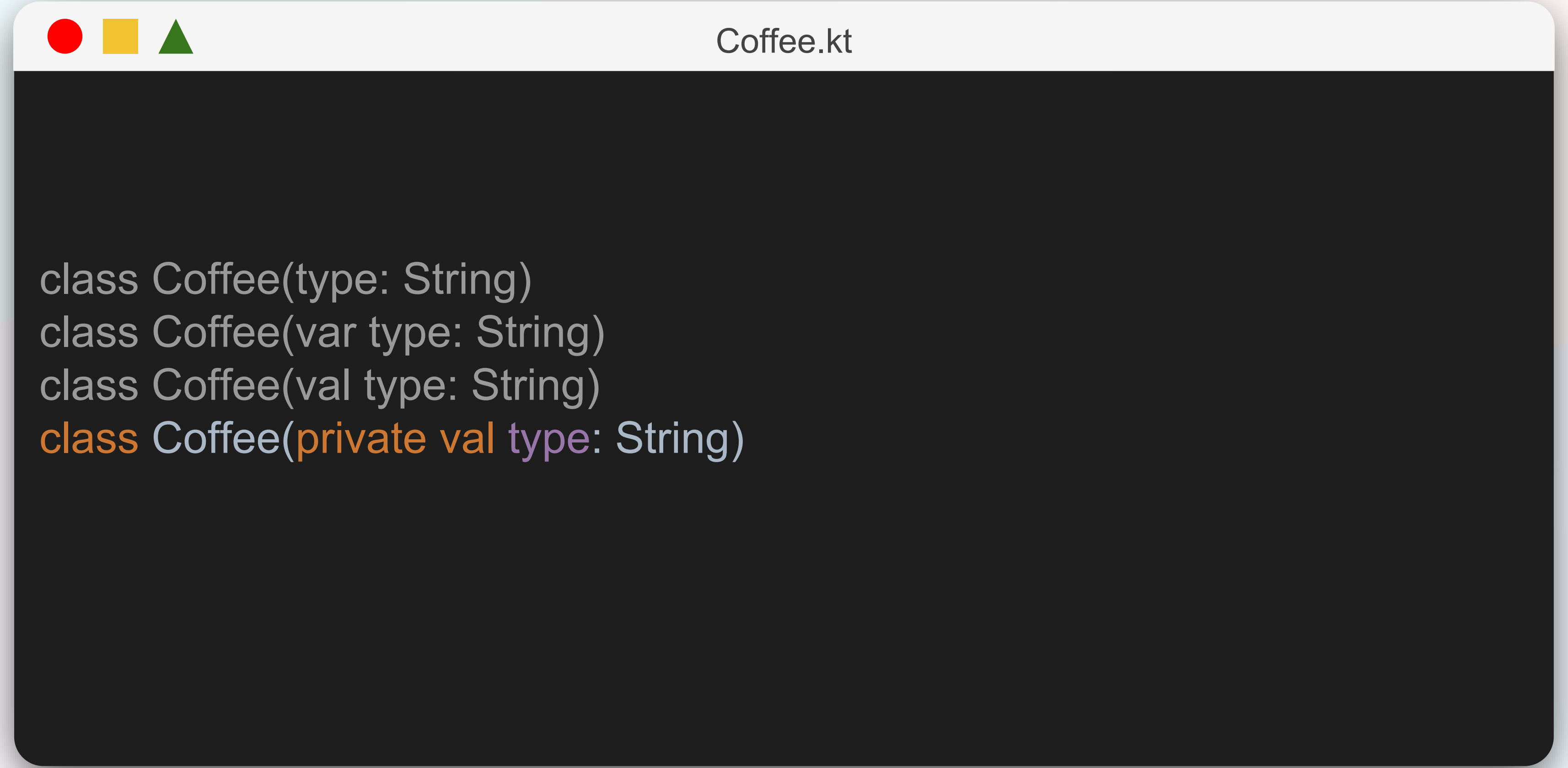
```
class Coffee(type: String)
class Coffee(var type: String)
```

Constructor


Coffee.kt

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

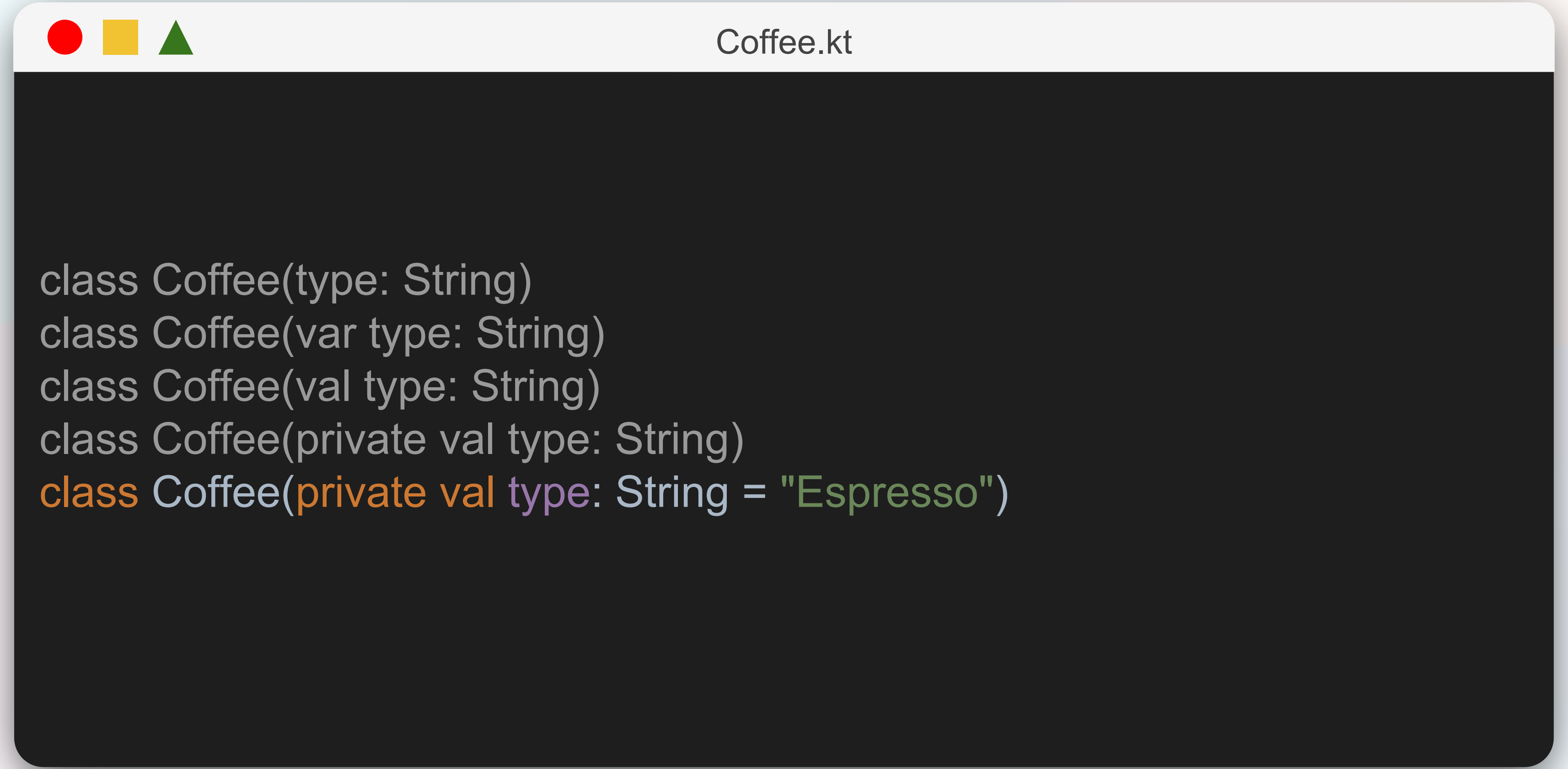
Constructor



Coffee.kt

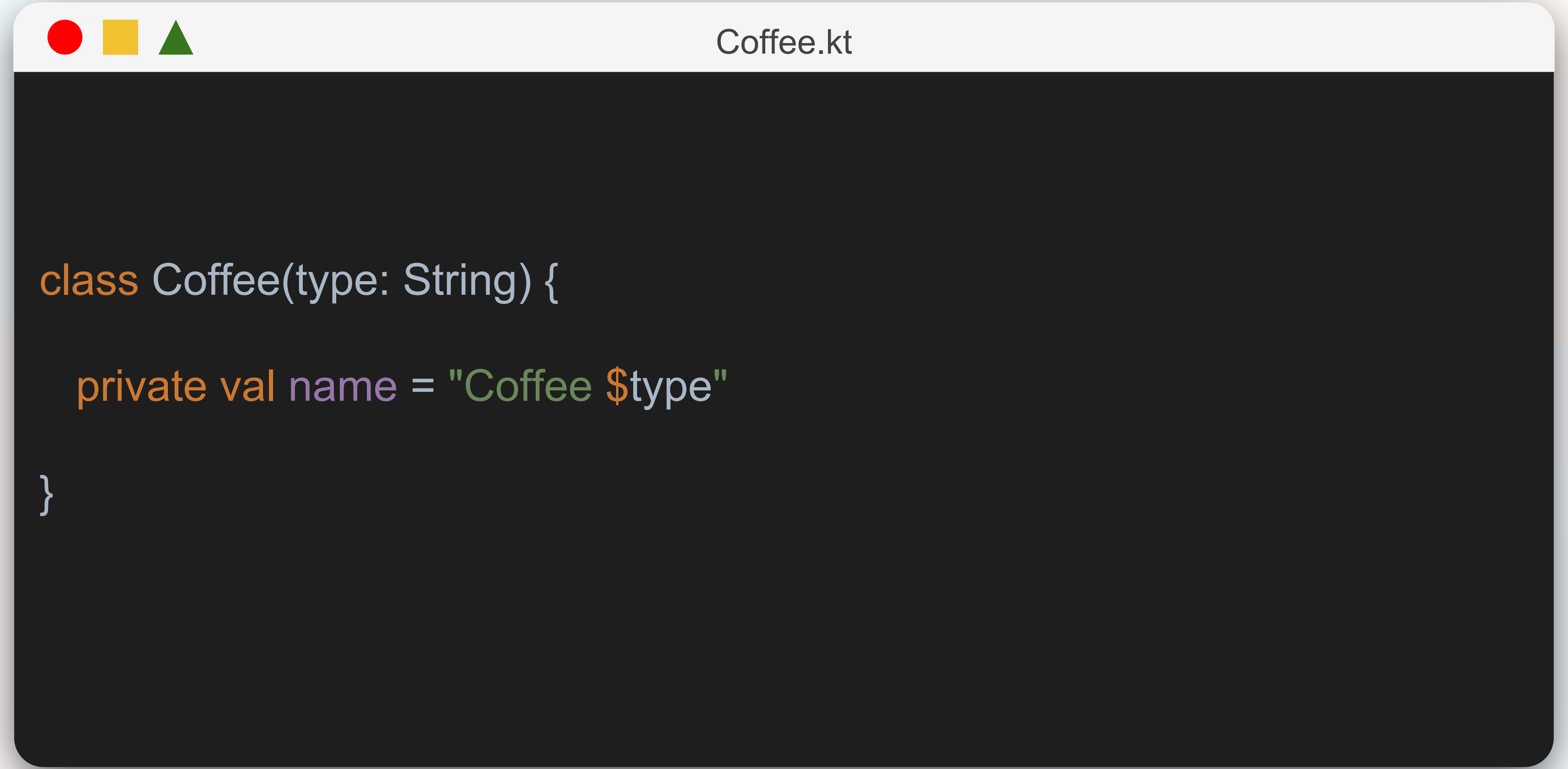
```
class Coffee(type: String)
class Coffee(var type: String)
class Coffee(val type: String)
class Coffee(private val type: String)
```

Constructor



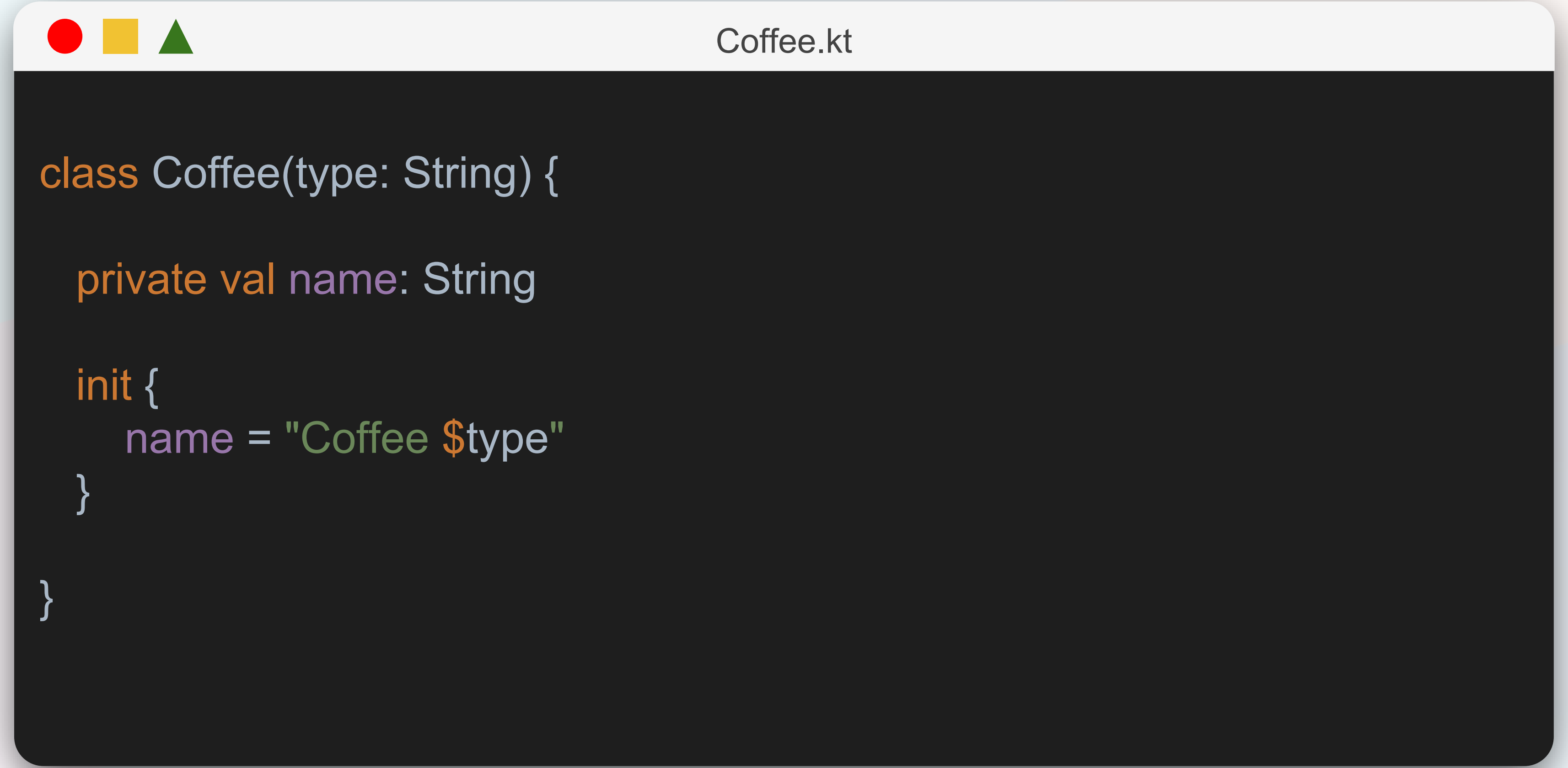
```kotlin
class Coffee(type: String)
class Coffee(var type: String)
class Coffee(val type: String)
class Coffee(private val type: String)
class Coffee(private val type: String = "Espresso")
```

Constructor


Coffee.kt

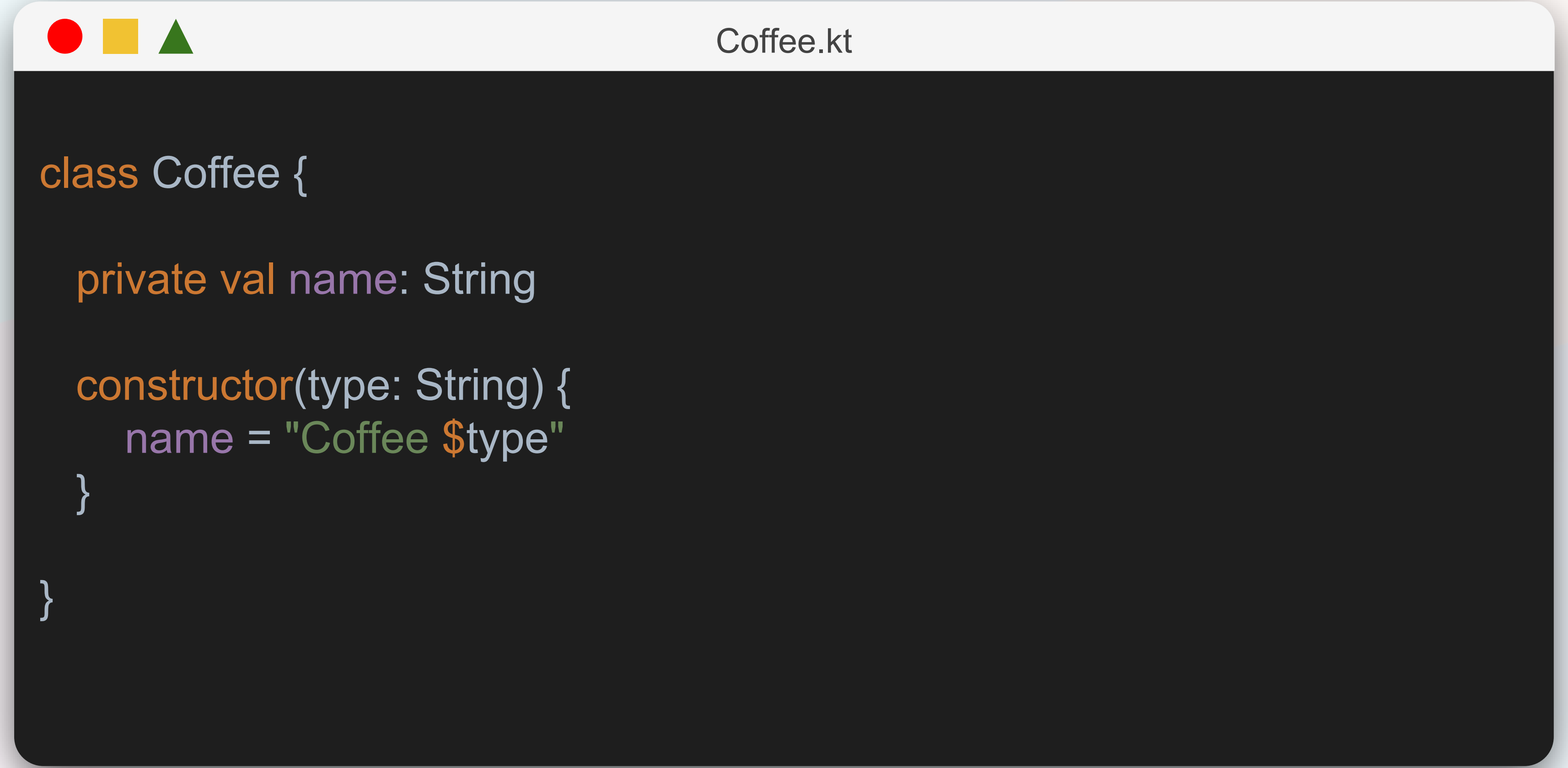
```
class Coffee(type: String) {

  private val name = "Coffee $type"

}
```

Constructor


Coffee.kt

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

    private val name: String

    init {
        name = "Coffee $type"
    }

}
```

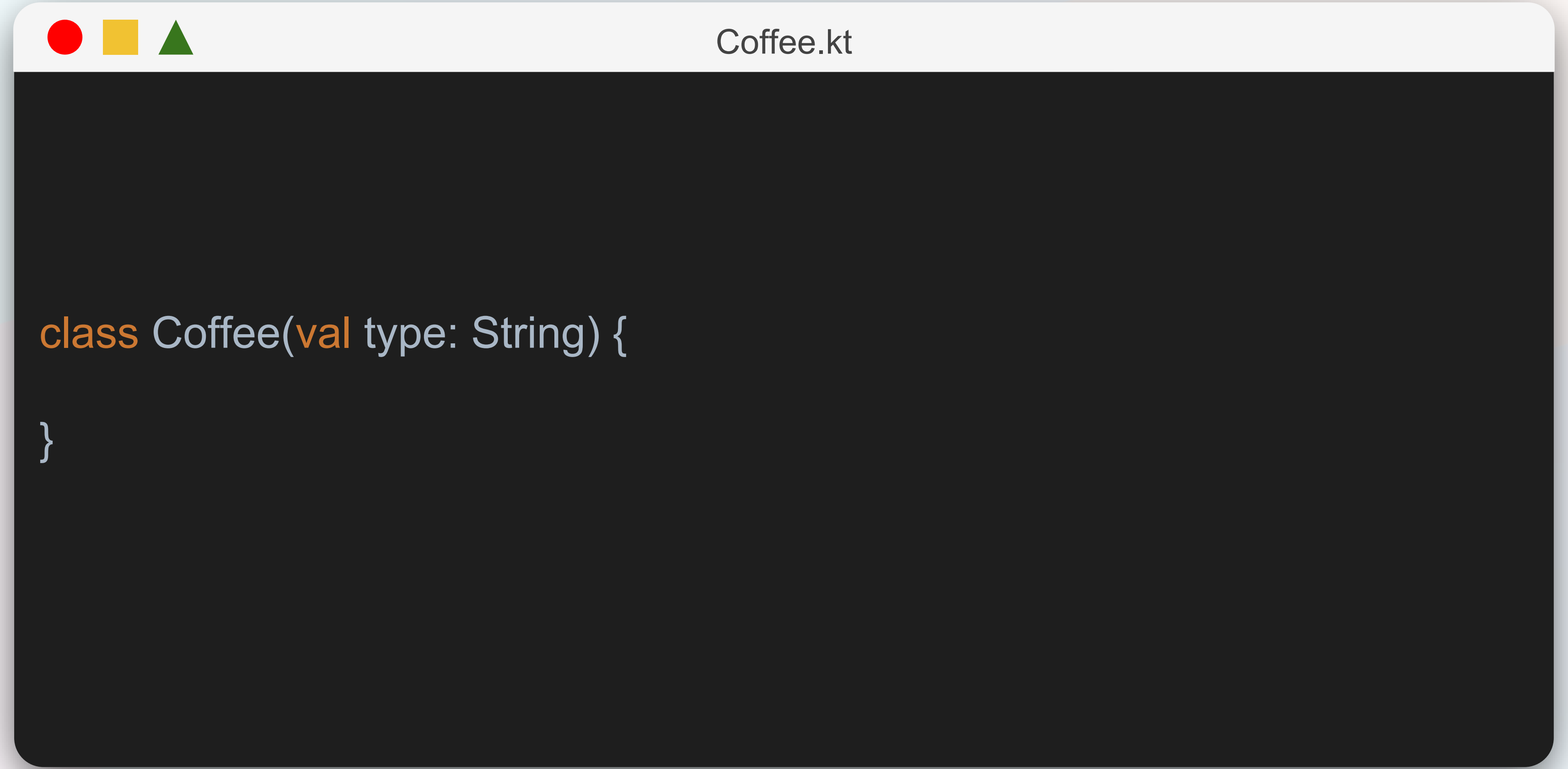
Constructor



Coffee.kt

```kotlin
class Coffee {

    private val name: String

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

}
```

Constructor



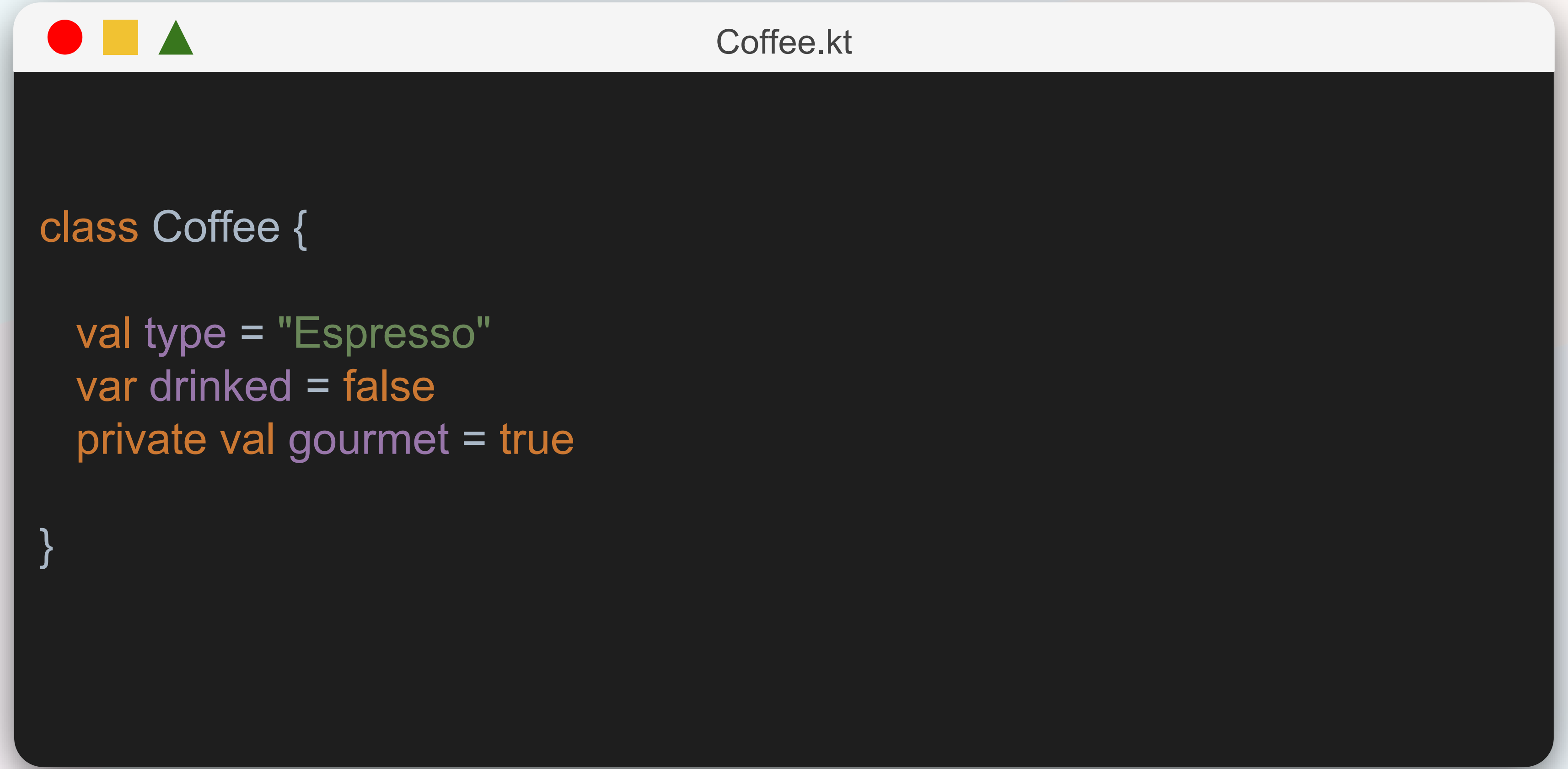
Coffee.kt

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

}
```

OOP

Properties

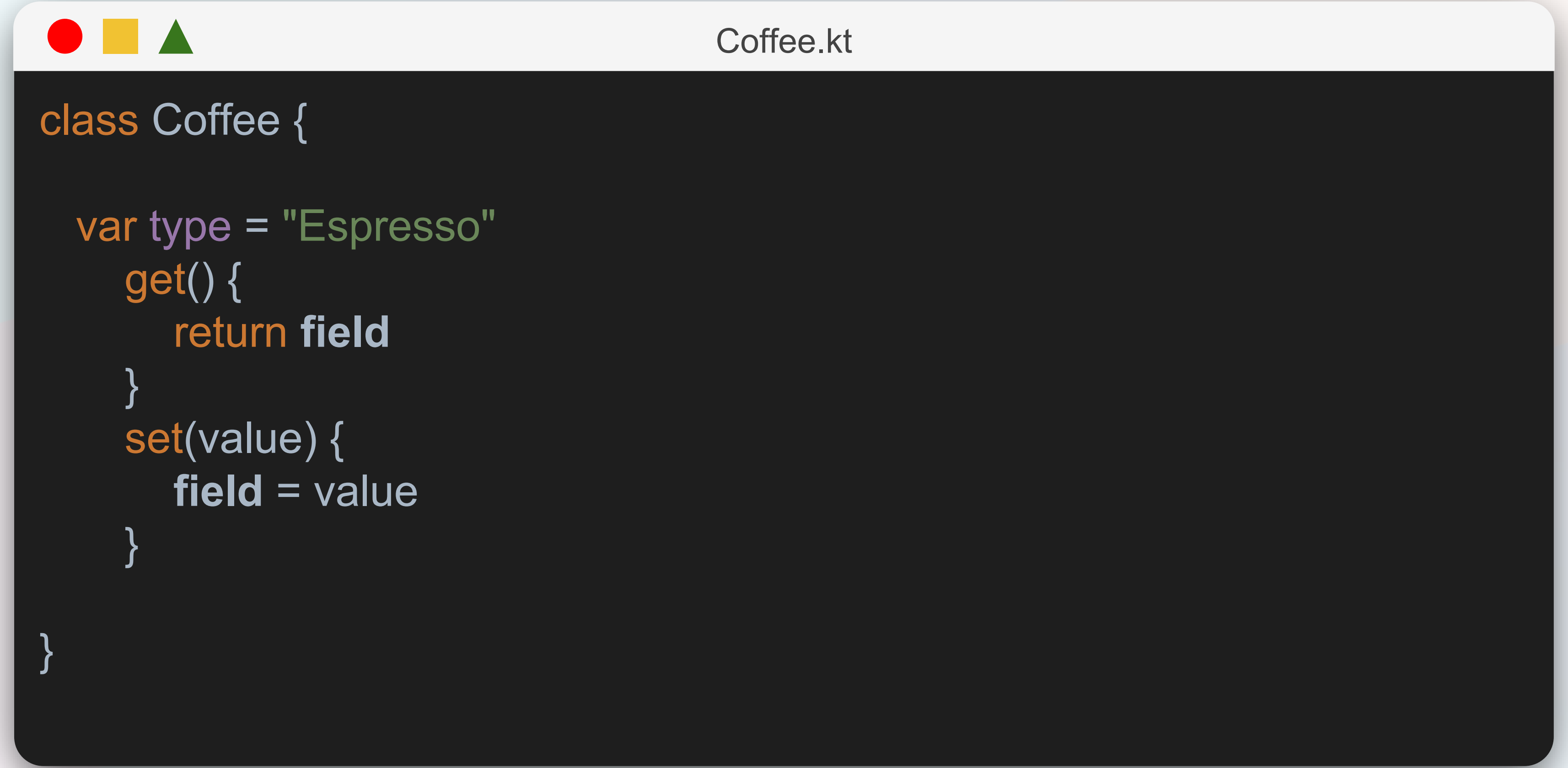
Properties



Coffee.kt

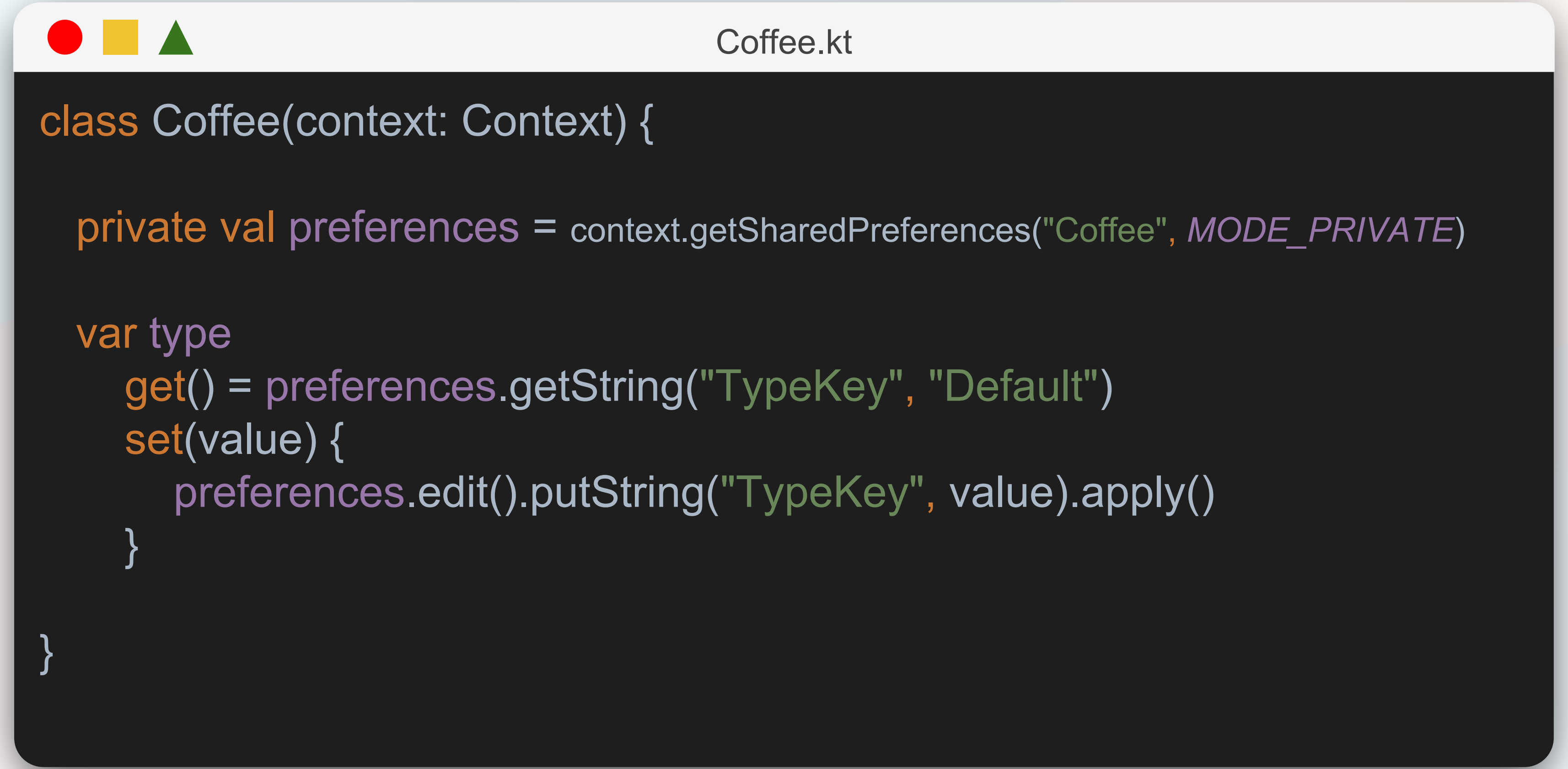
```kotlin
class Coffee {

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

}
```

Properties


Coffee.kt

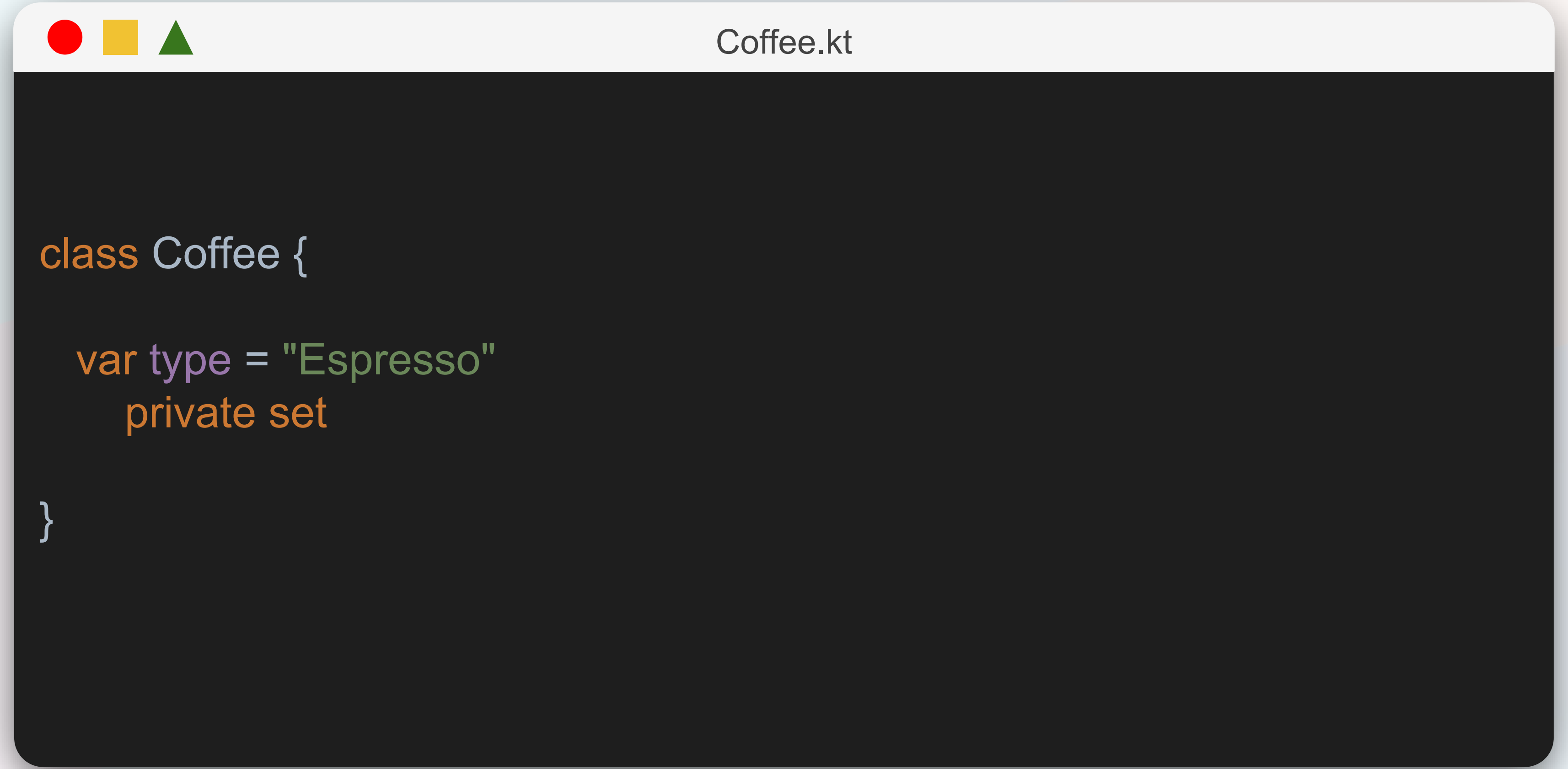
```kotlin
class Coffee {

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

}
```

Properties


Coffee.kt

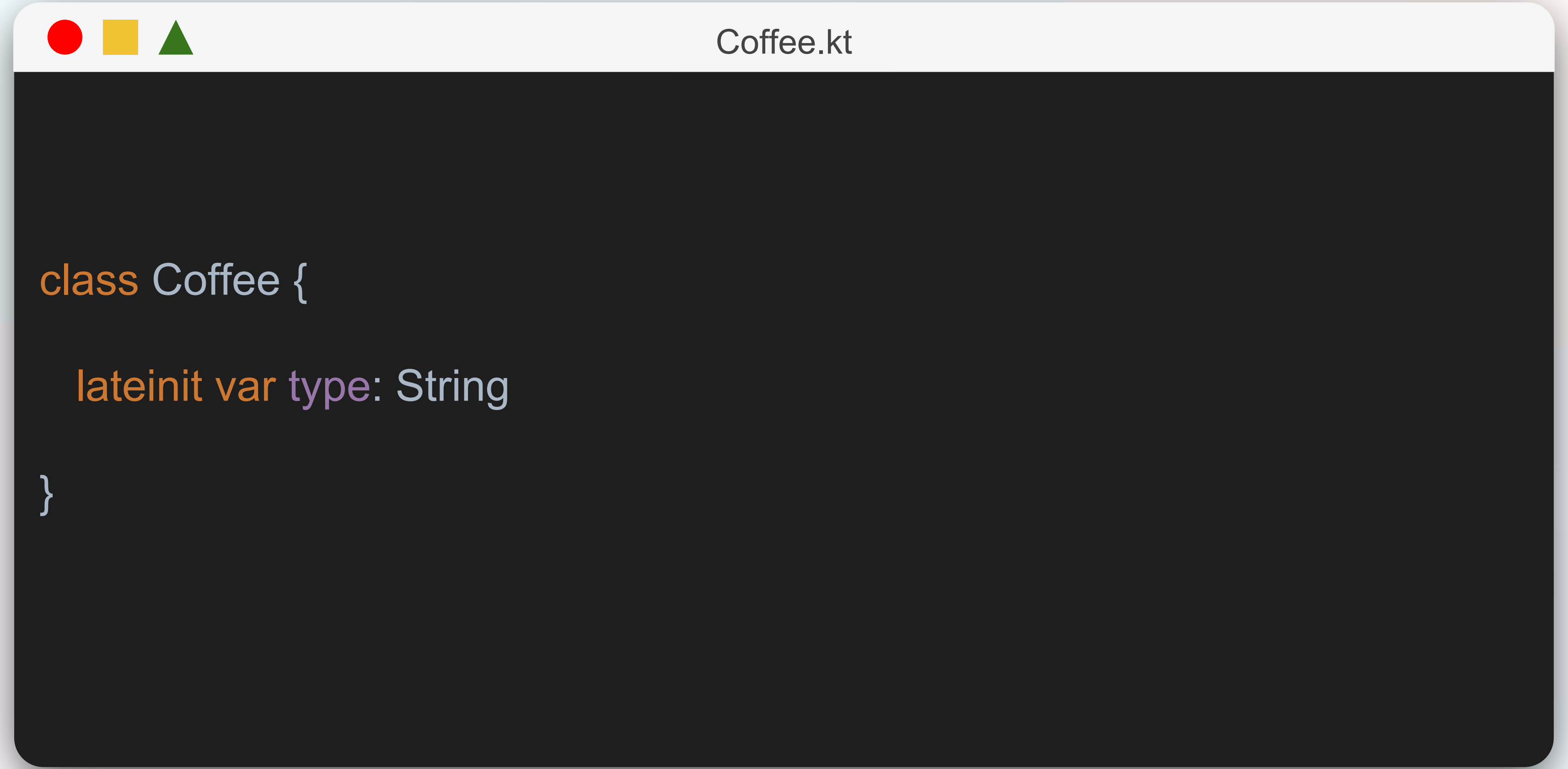
```kotlin
class Coffee(context: Context) {

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

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

}
```

Properties



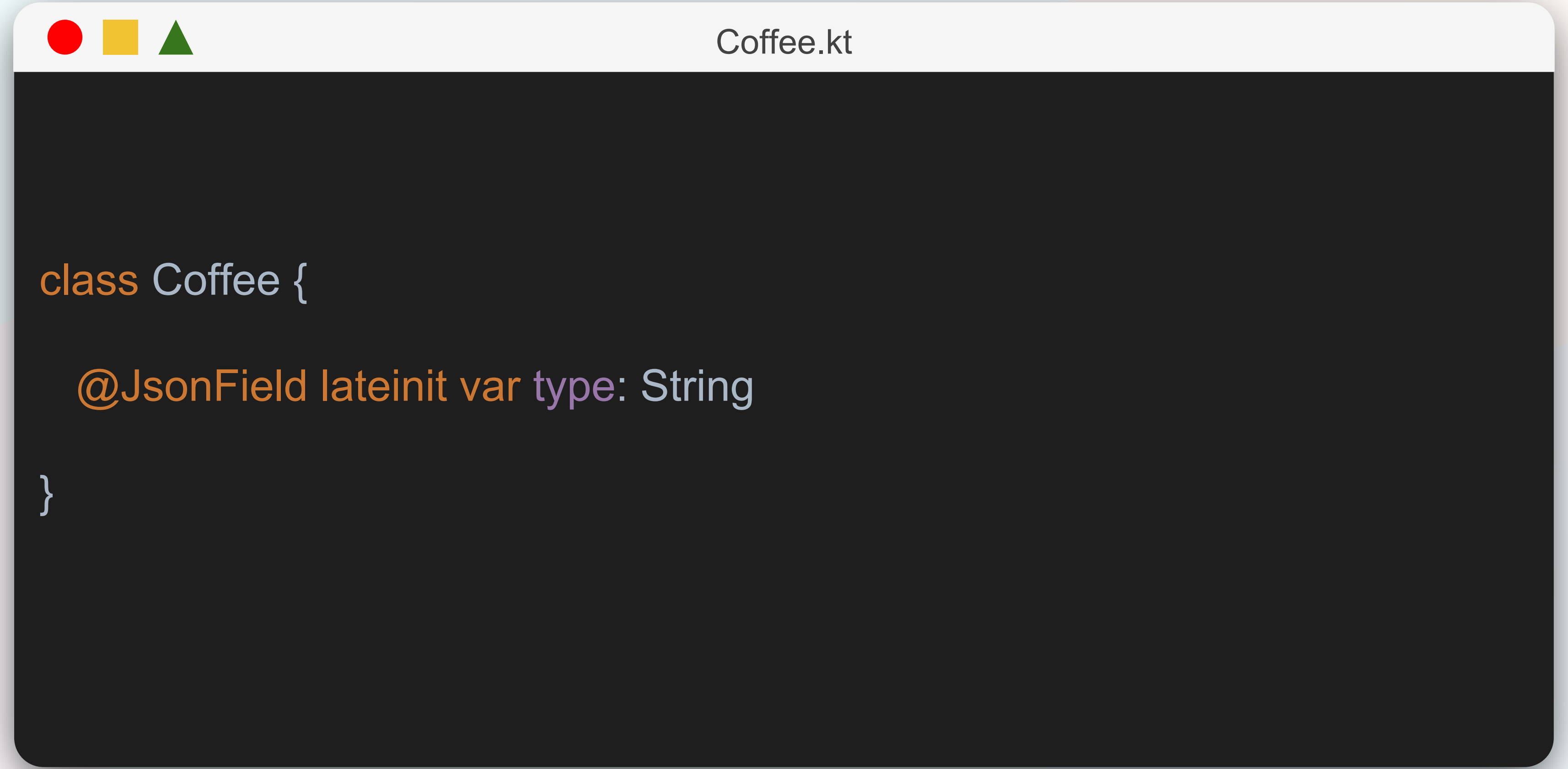
```
class Coffee {

  var type = "Espresso"
      private set

}
```

Properties


Coffee.kt

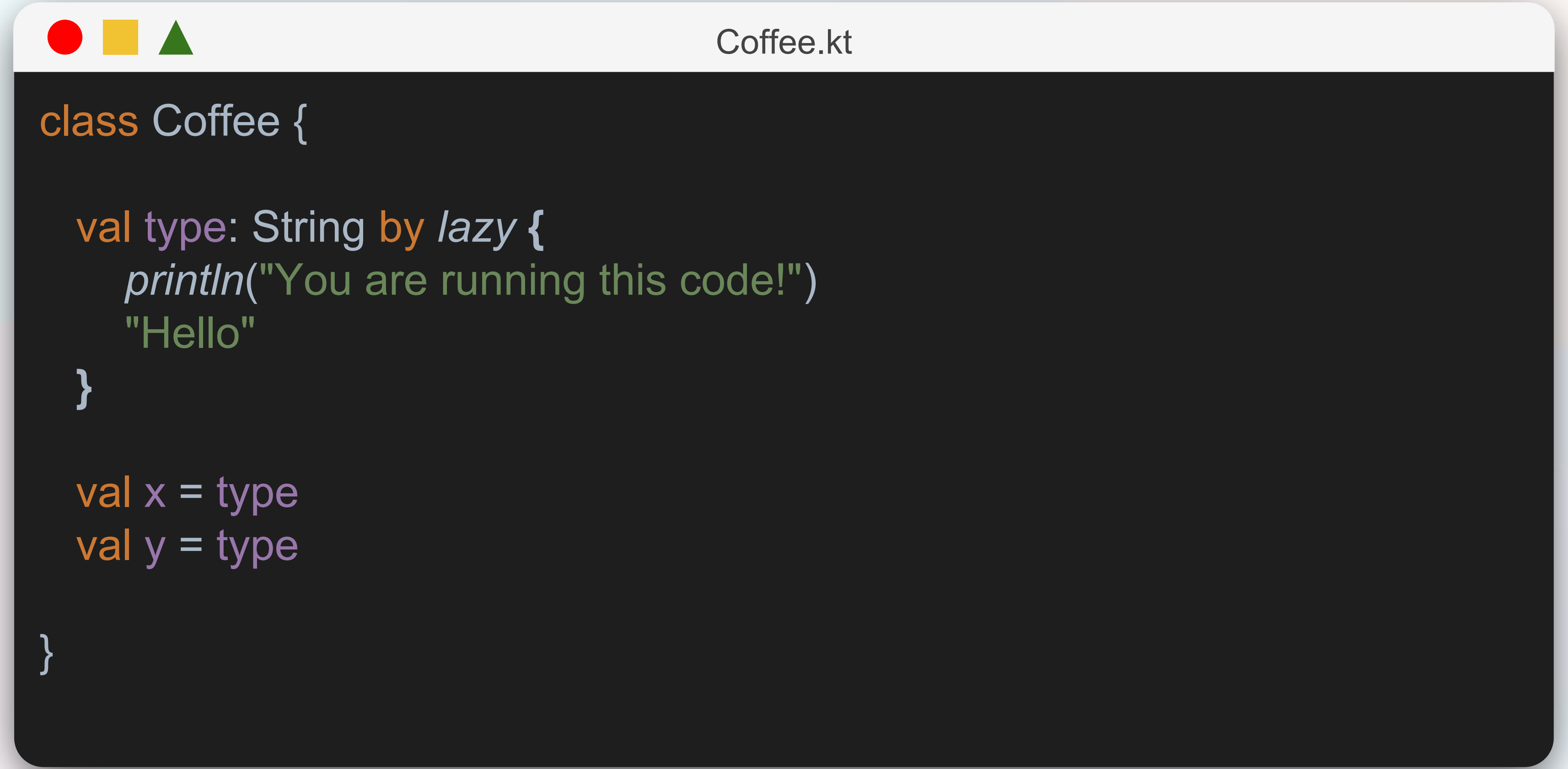
```
class Coffee {

  lateinit var type: String

}
```

Properties



```kotlin
class Coffee {

  @JsonField lateinit var type: String

}
```

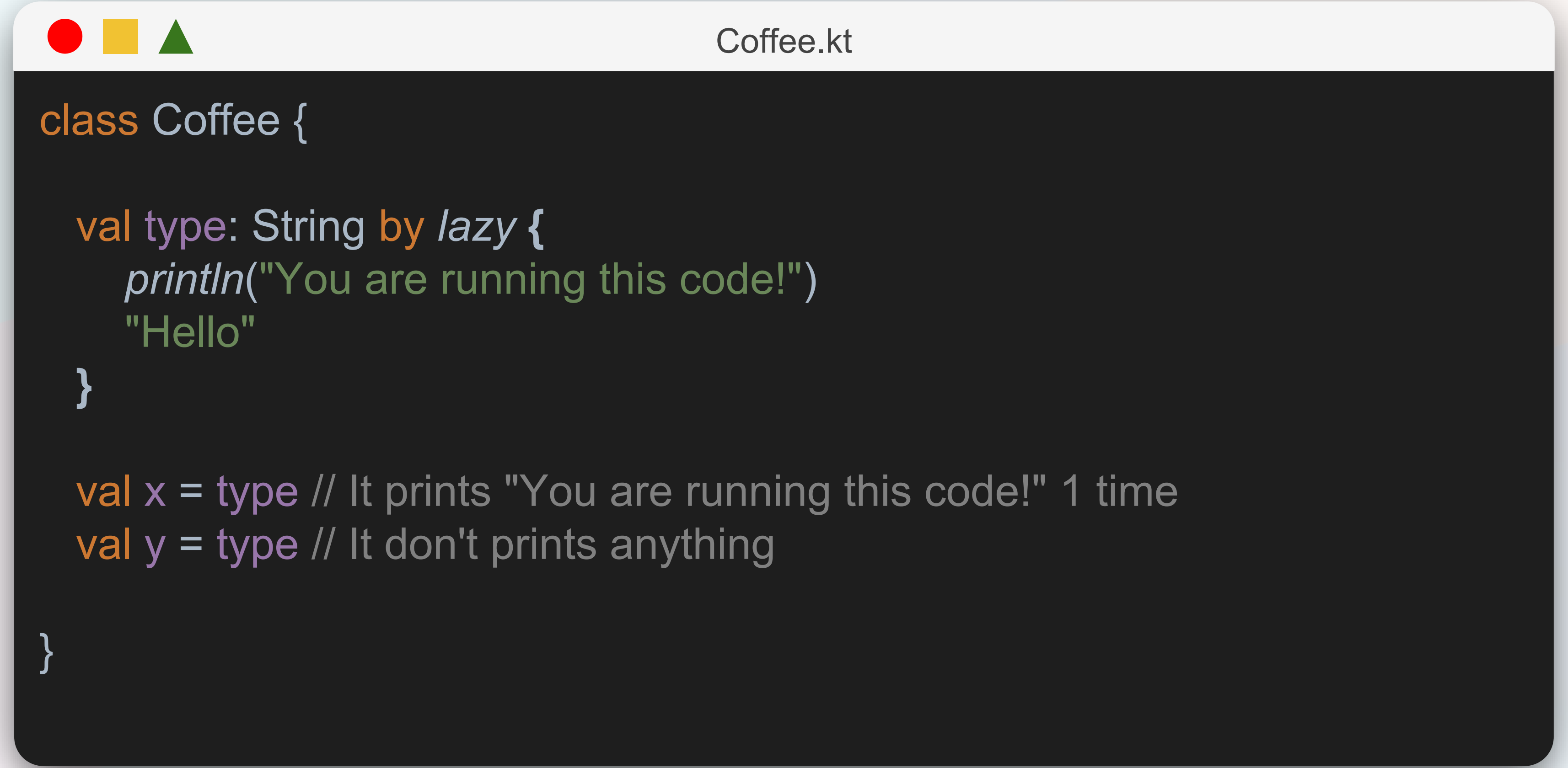
Properties



Coffee.kt

```
class Coffee {

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

    val x = type
    val y = type

}
```

Properties



```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

}
```

OOP

Methods

Kotlin methods are functions,
and you already know it.

OOP

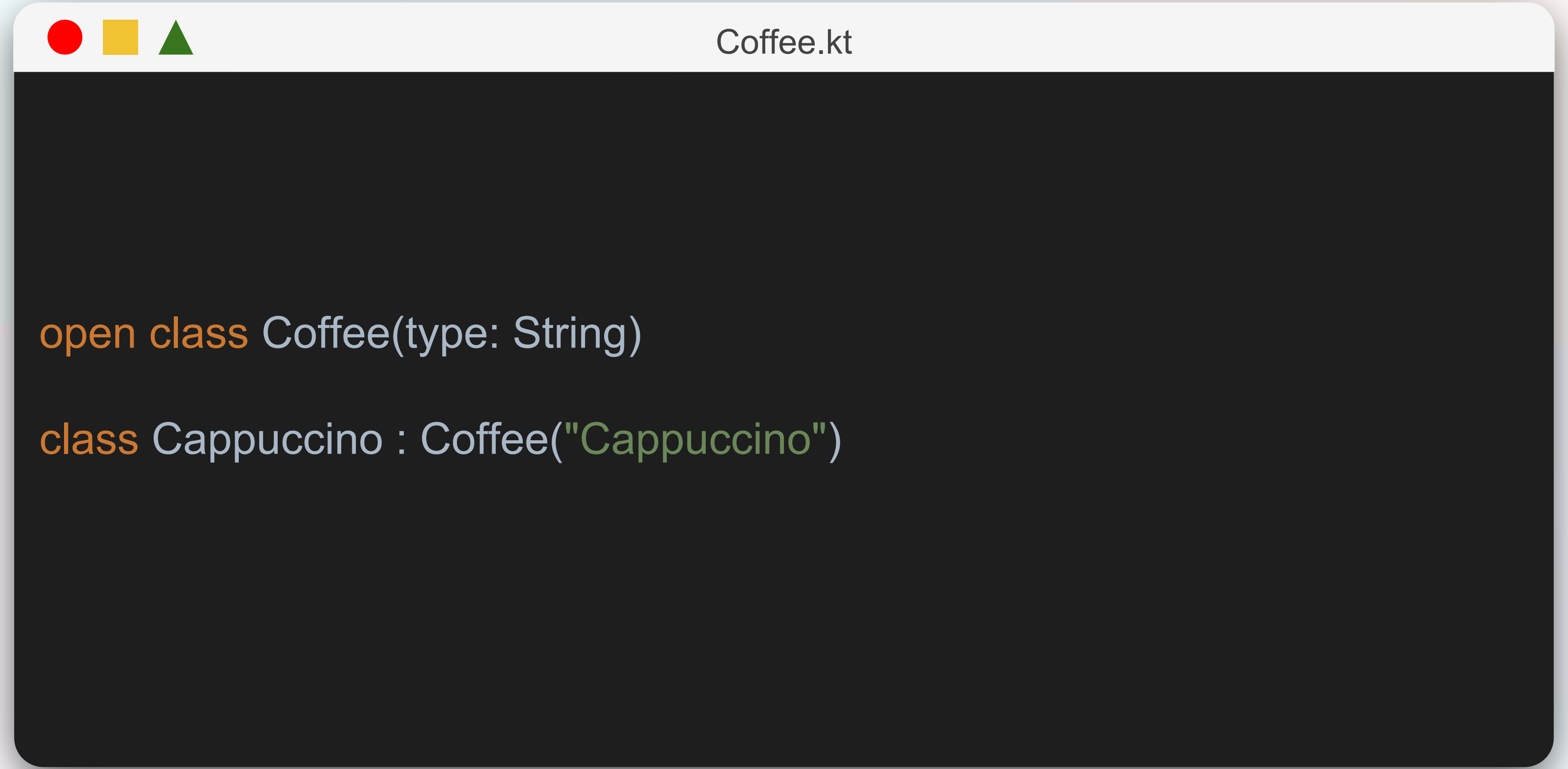
Inheritance

Inheritance

Coffee.kt

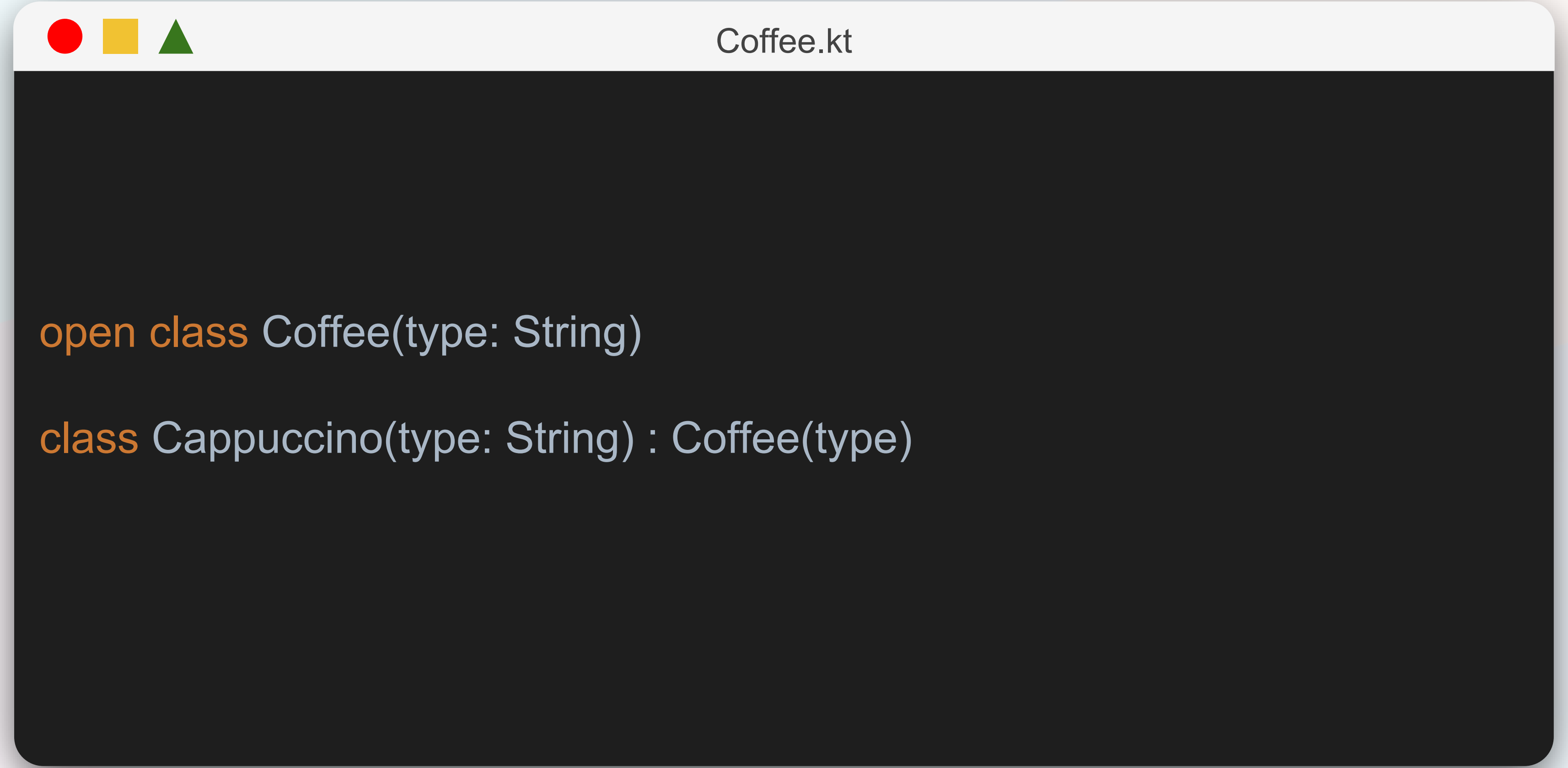
```kotlin
open class Coffee

class Cappuccino : Coffee()
```

Inheritance


Coffee.kt

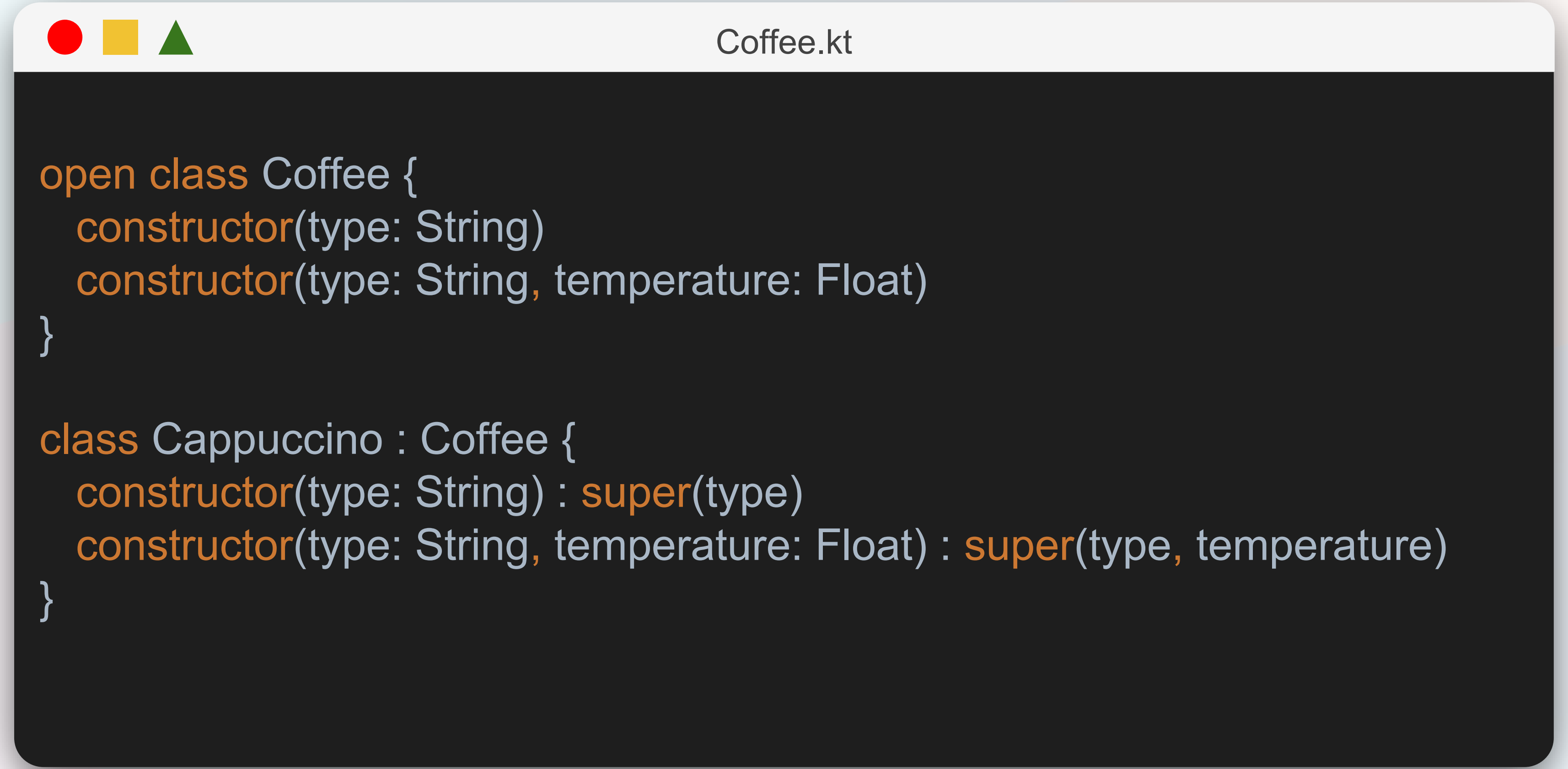
```kotlin
open class Coffee(type: String)

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

Inheritance



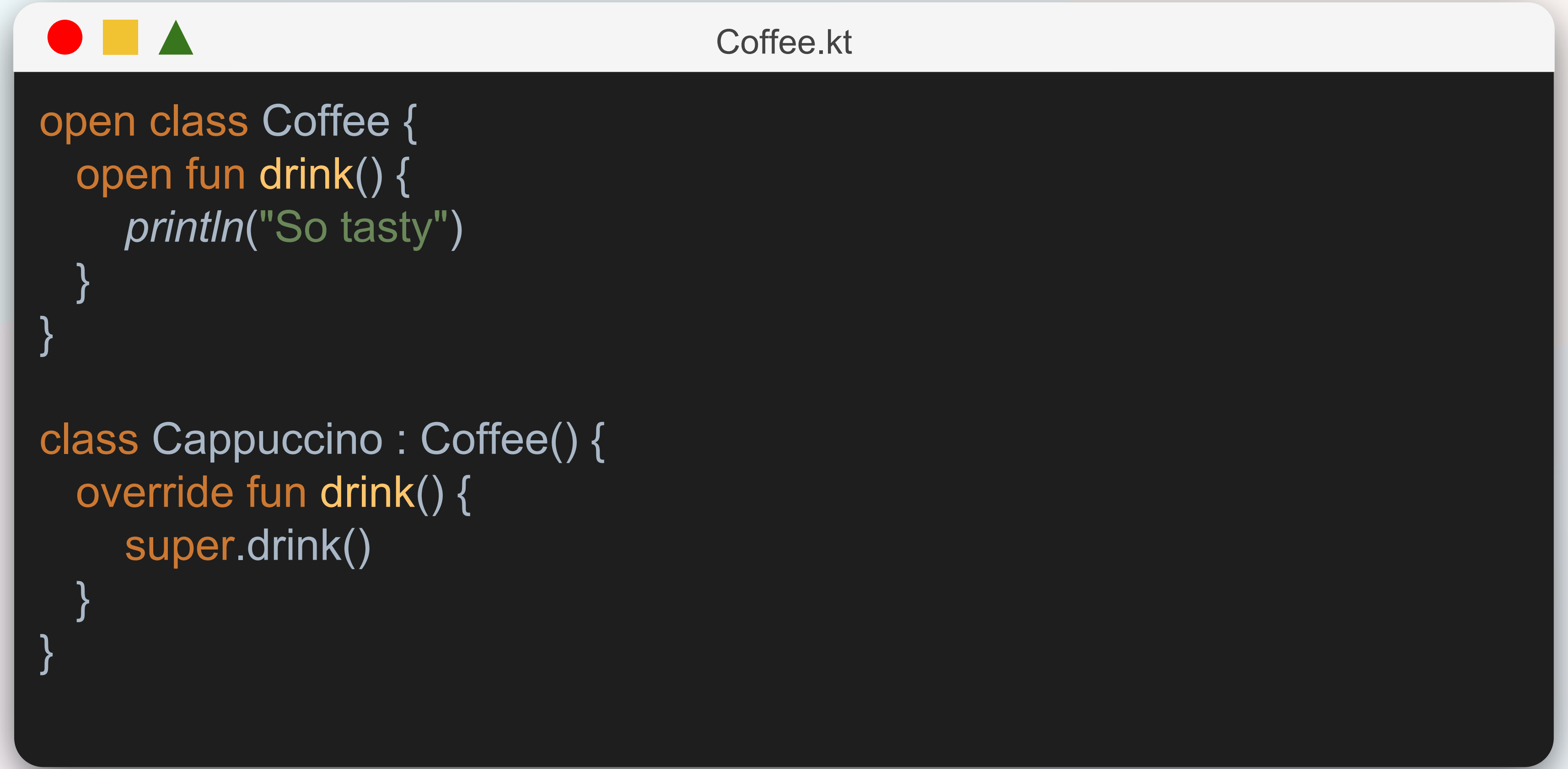
```
open class Coffee(type: String)

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

Inheritance



```
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)
}
```

Inheritance

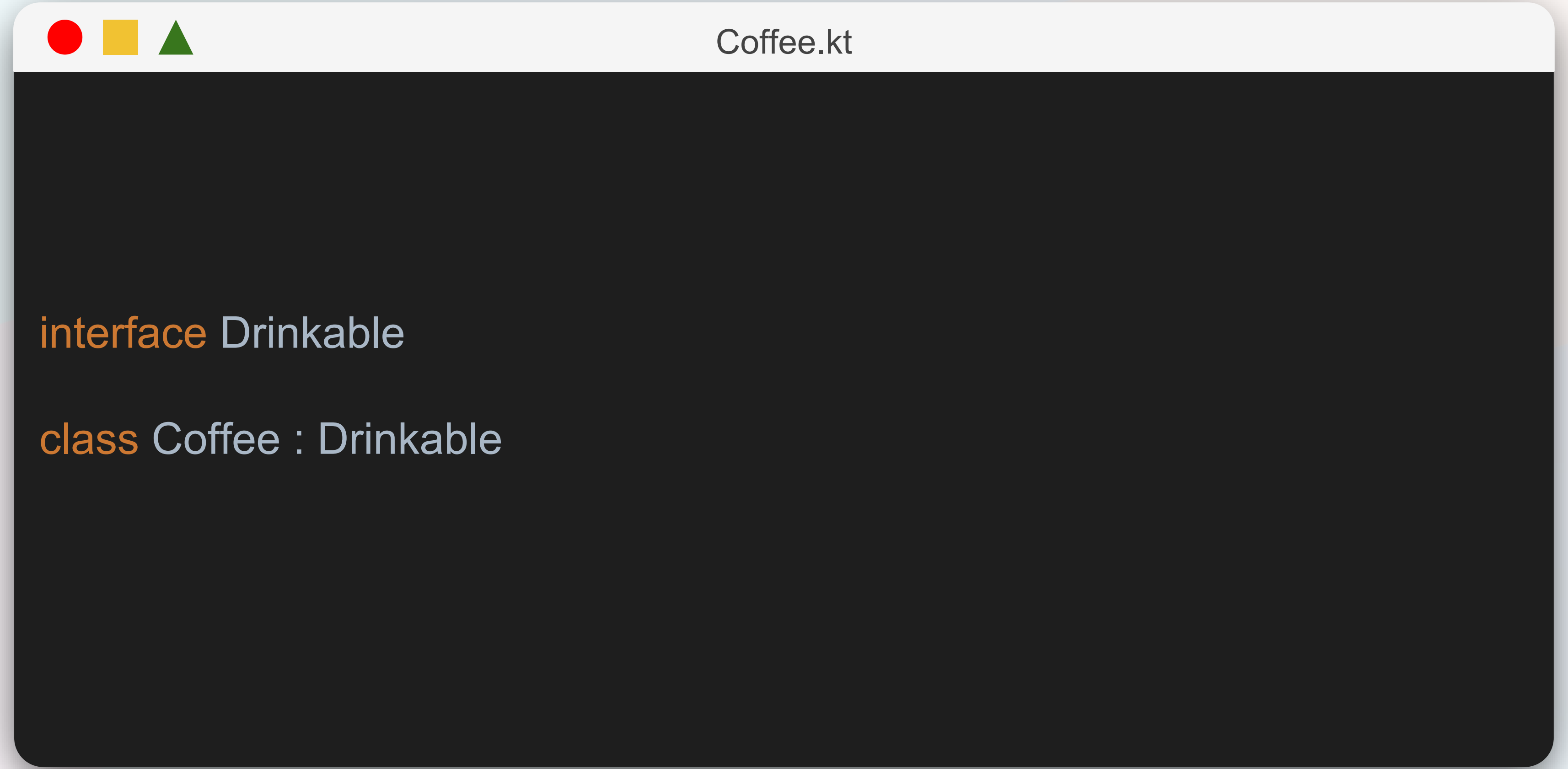

Coffee.kt

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

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

OOP

Interfaces

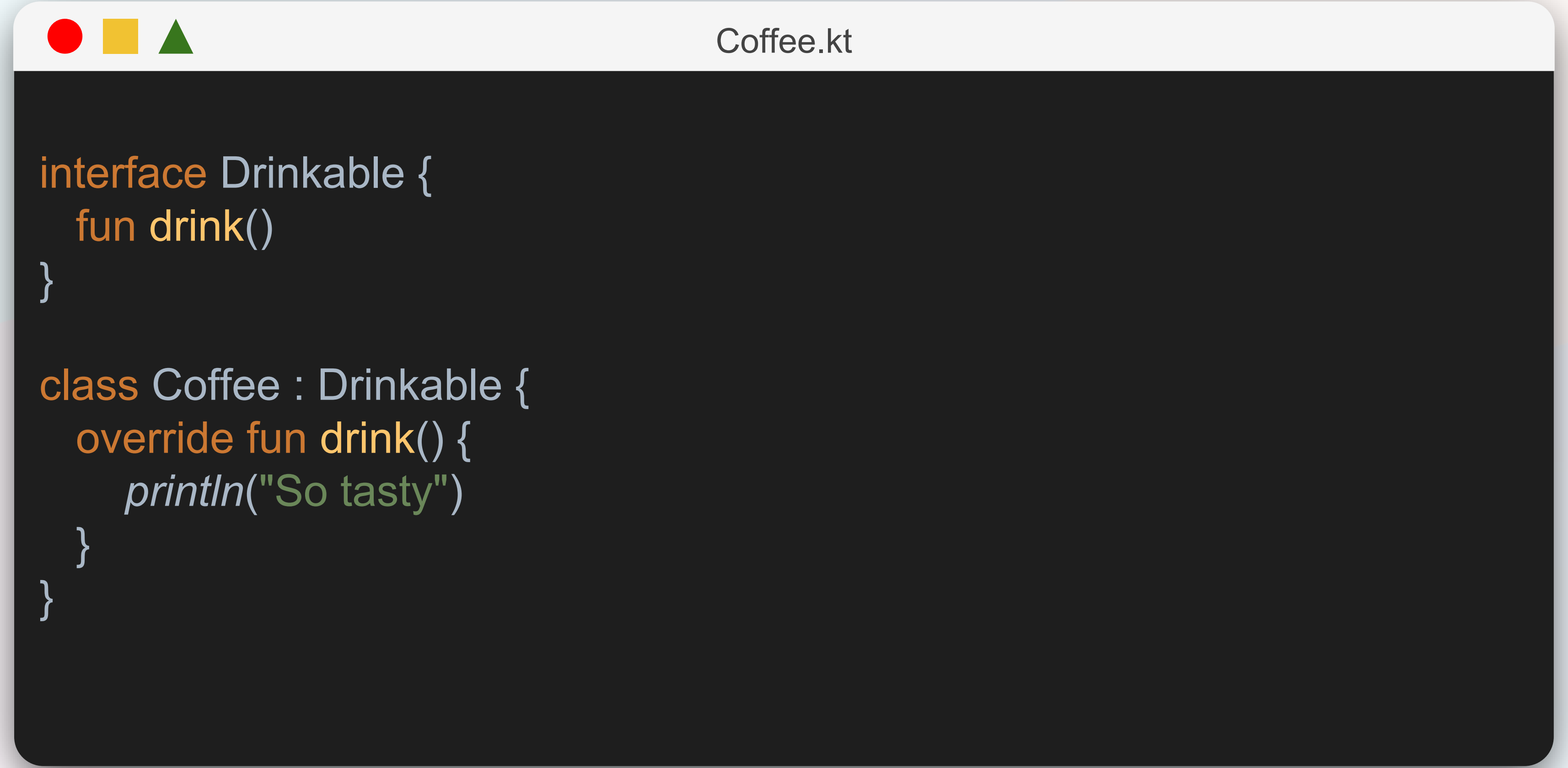
Interfaces



Coffee.kt

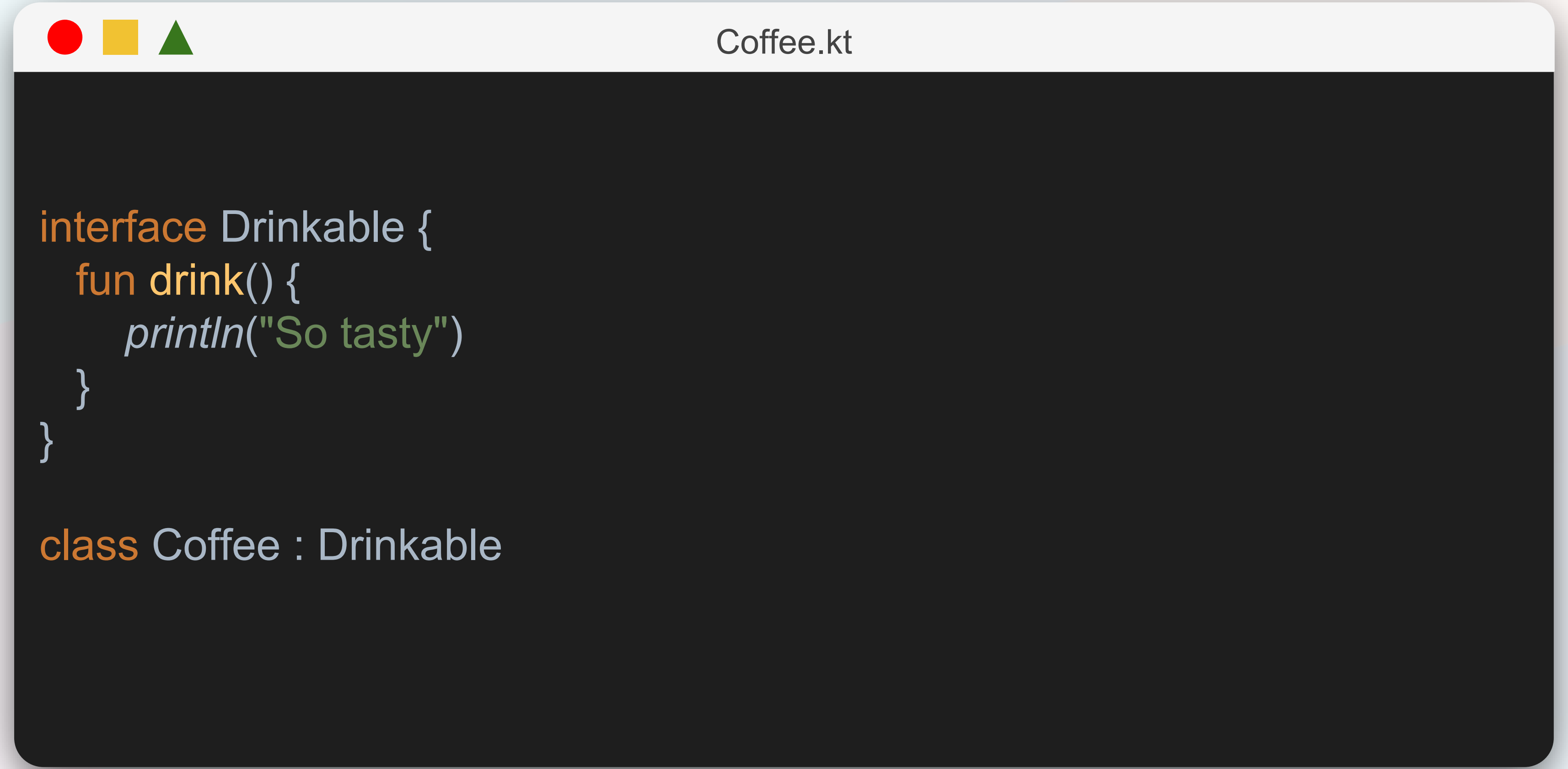
```kotlin
interface Drinkable

class Coffee : Drinkable
```

Interfaces


Coffee.kt

```
interface Drinkable {
    fun drink()
}

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

Interfaces


Coffee.kt

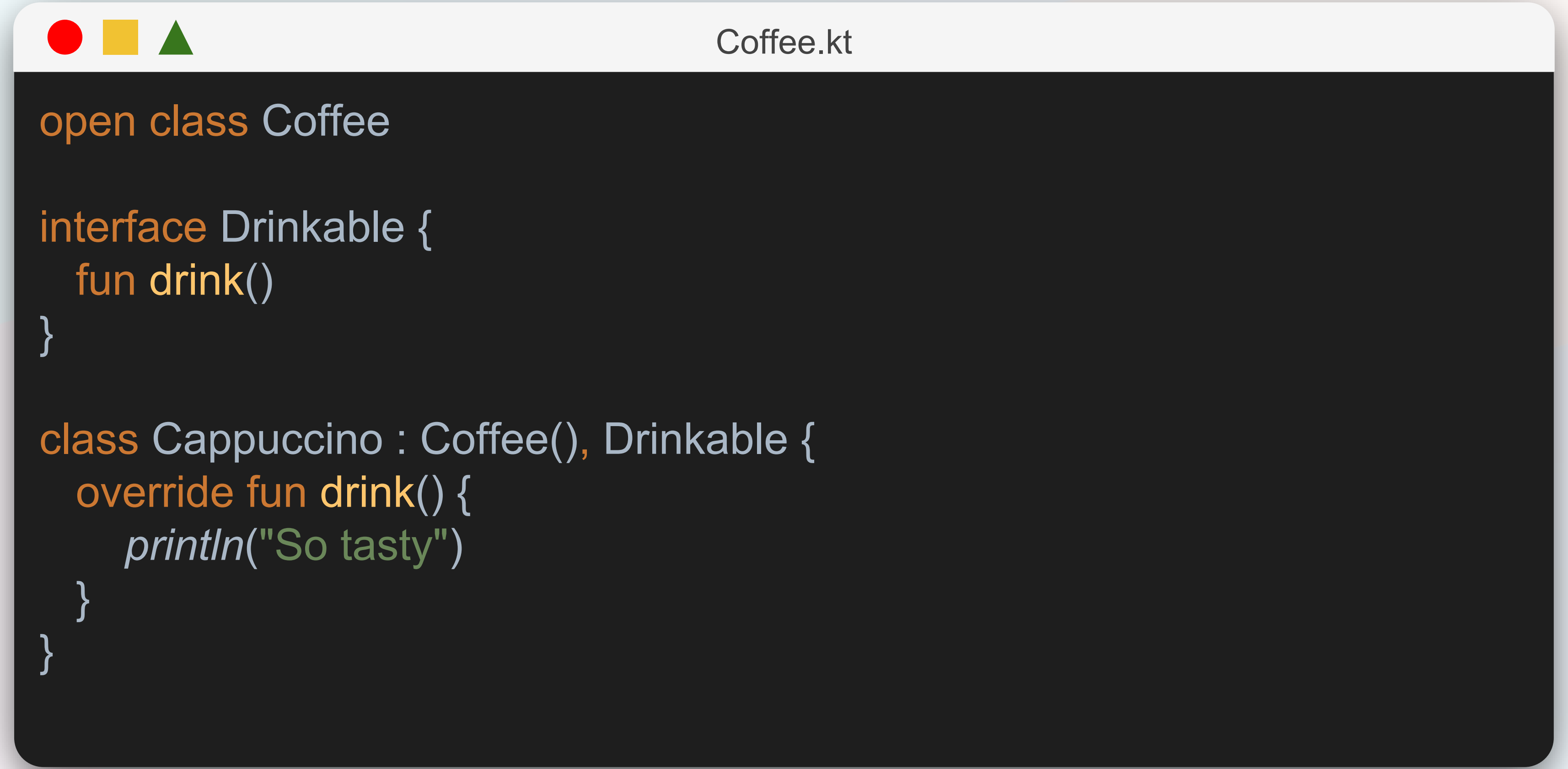
```
interface Drinkable {
   fun drink() {
       println("So tasty")
   }
}

class Coffee : Drinkable
```

Interfaces

Coffee.kt

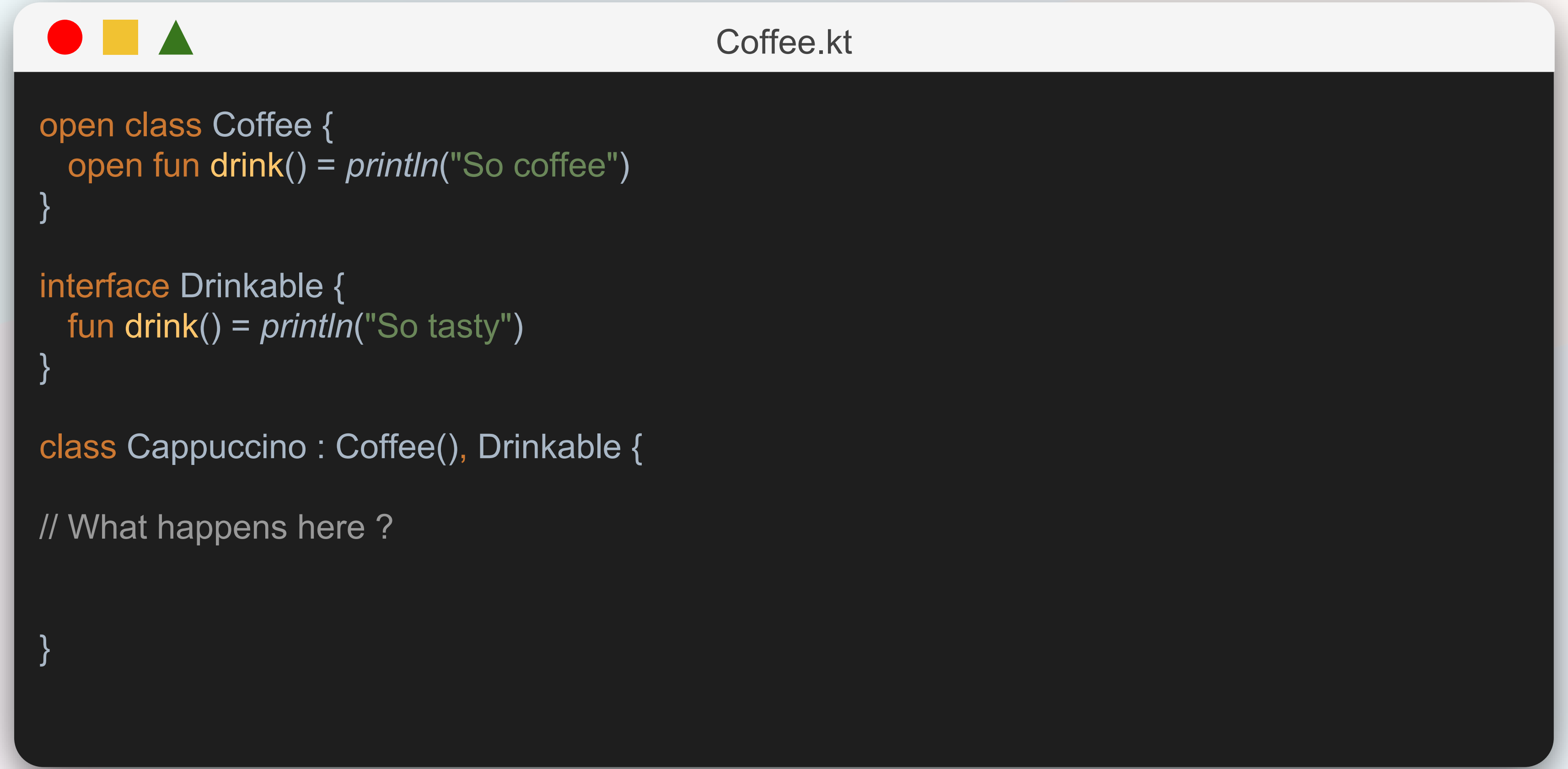
```kotlin
interface Drinkable {
   val type: String
}

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

Interfaces


Coffee.kt

```
open class Coffee

interface Drinkable {
   fun drink()
}

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

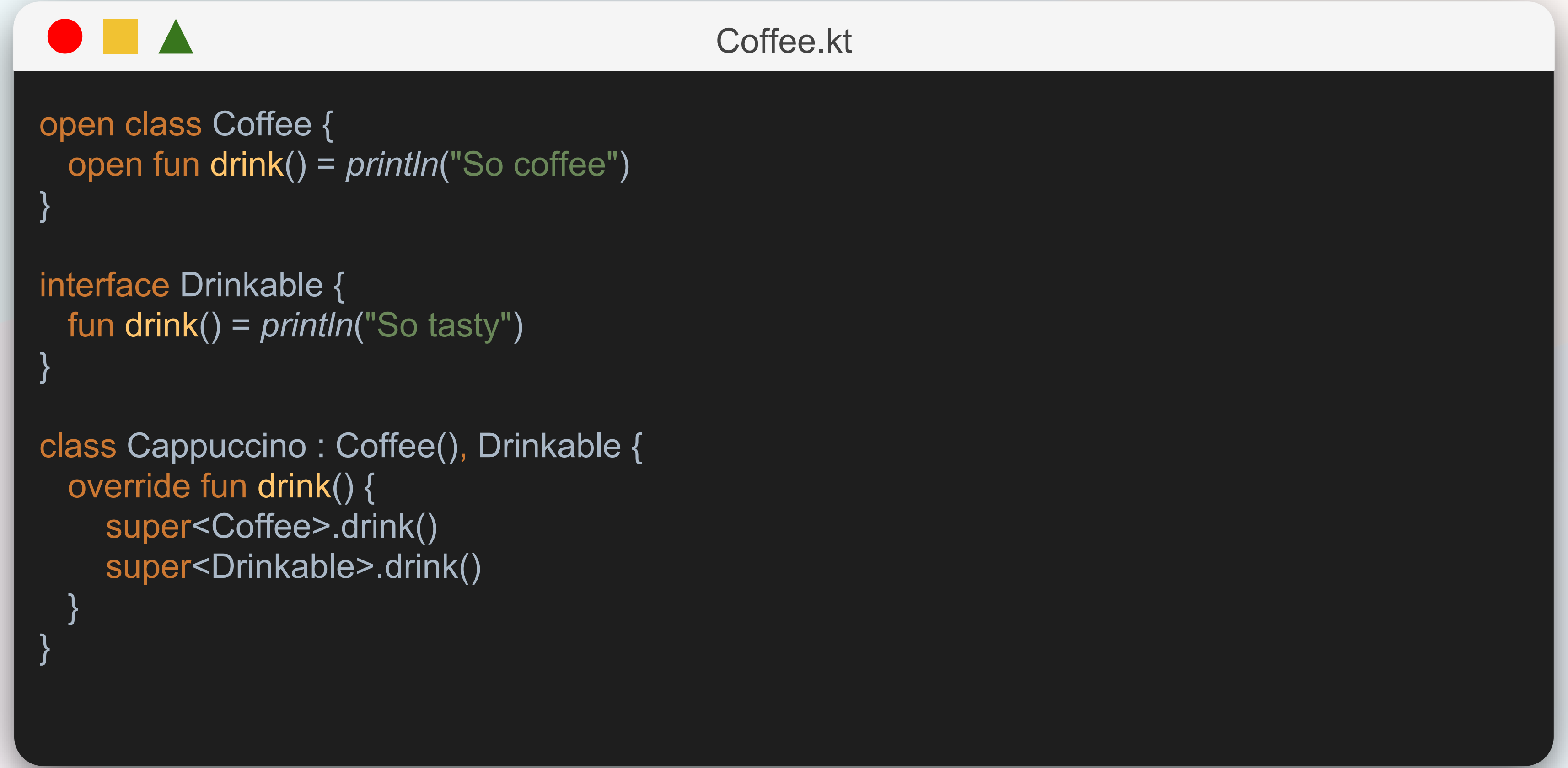
Interfaces



Coffee.kt

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

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

class Cappuccino : Coffee(), Drinkable {

// What happens here ?


}
```

Interfaces



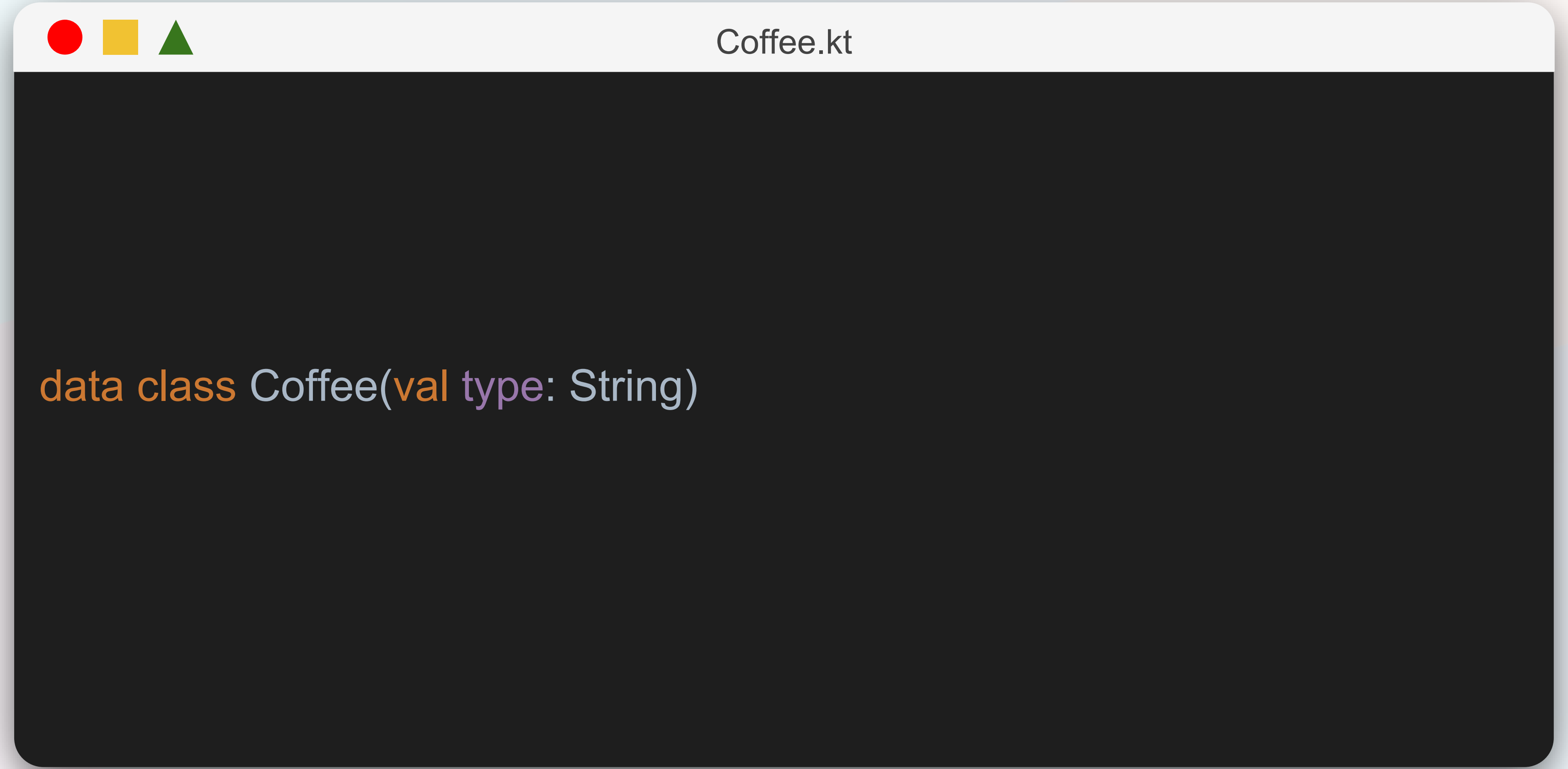
Coffee.kt

```
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()
   }
}
```

OOP

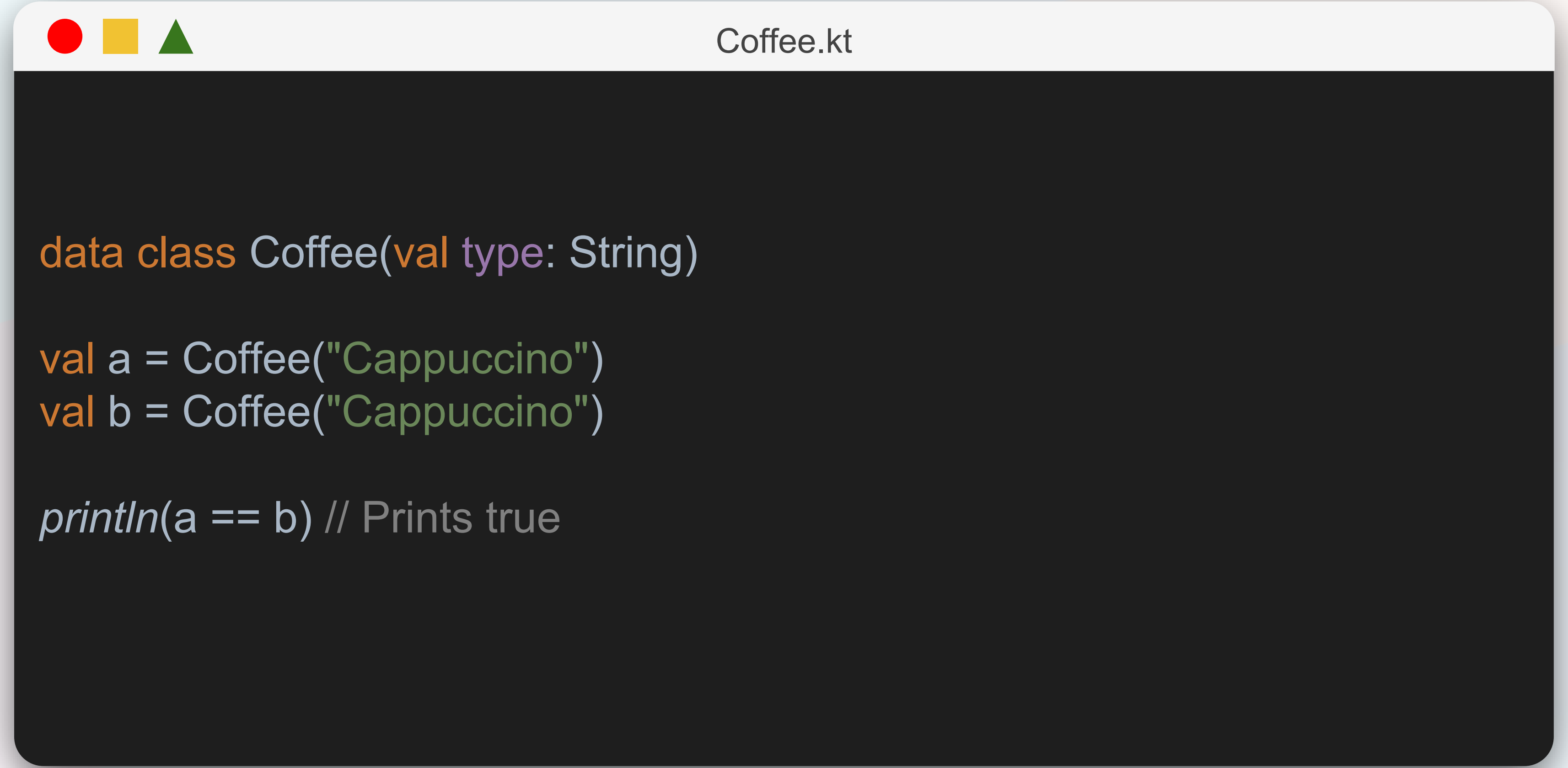
Data Classes

Data Classes



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

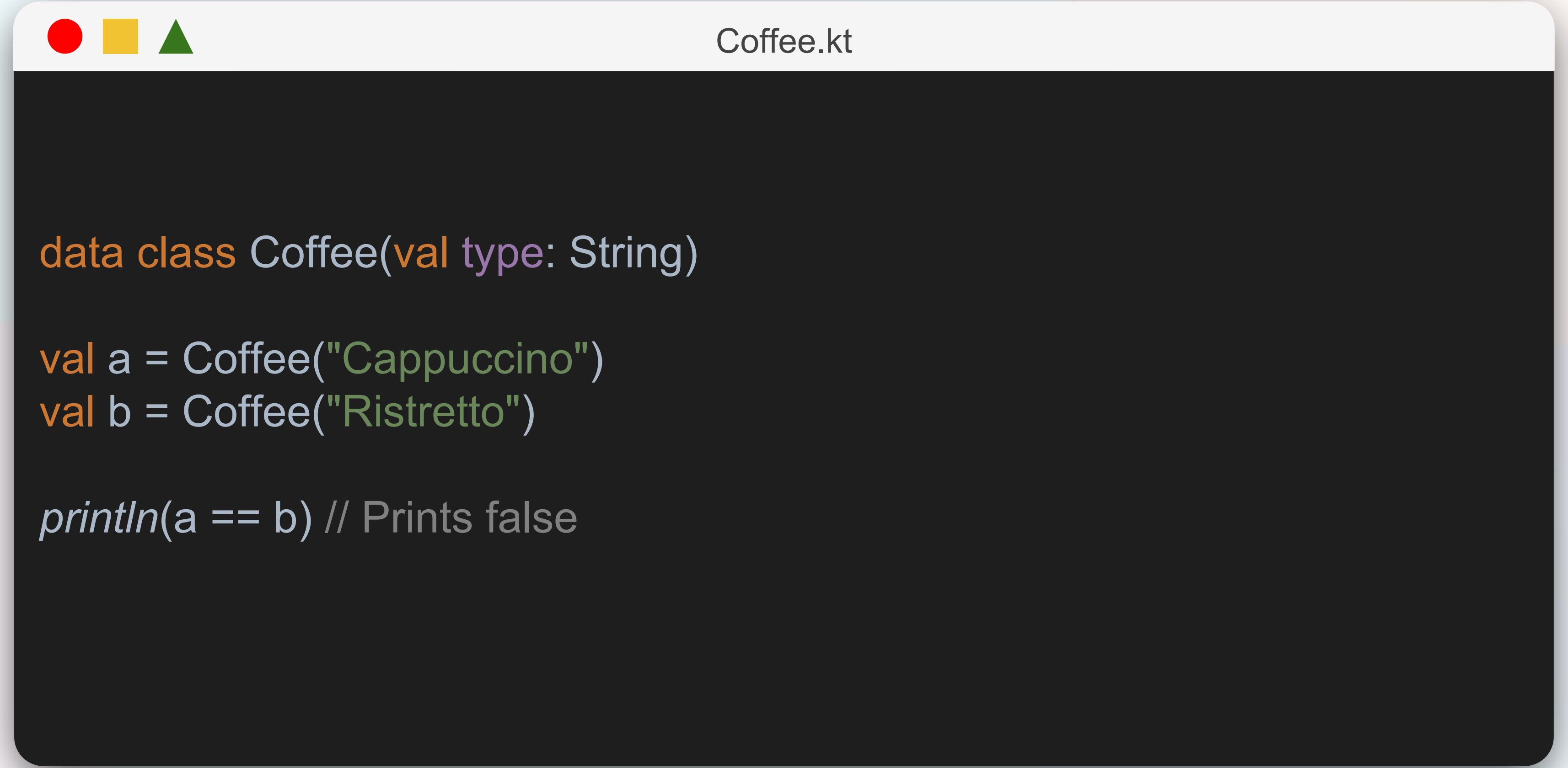
Data Classes - Equals & HashCode



Coffee.kt

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

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

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

Data Classes - Equals & HashCode



Coffee.kt

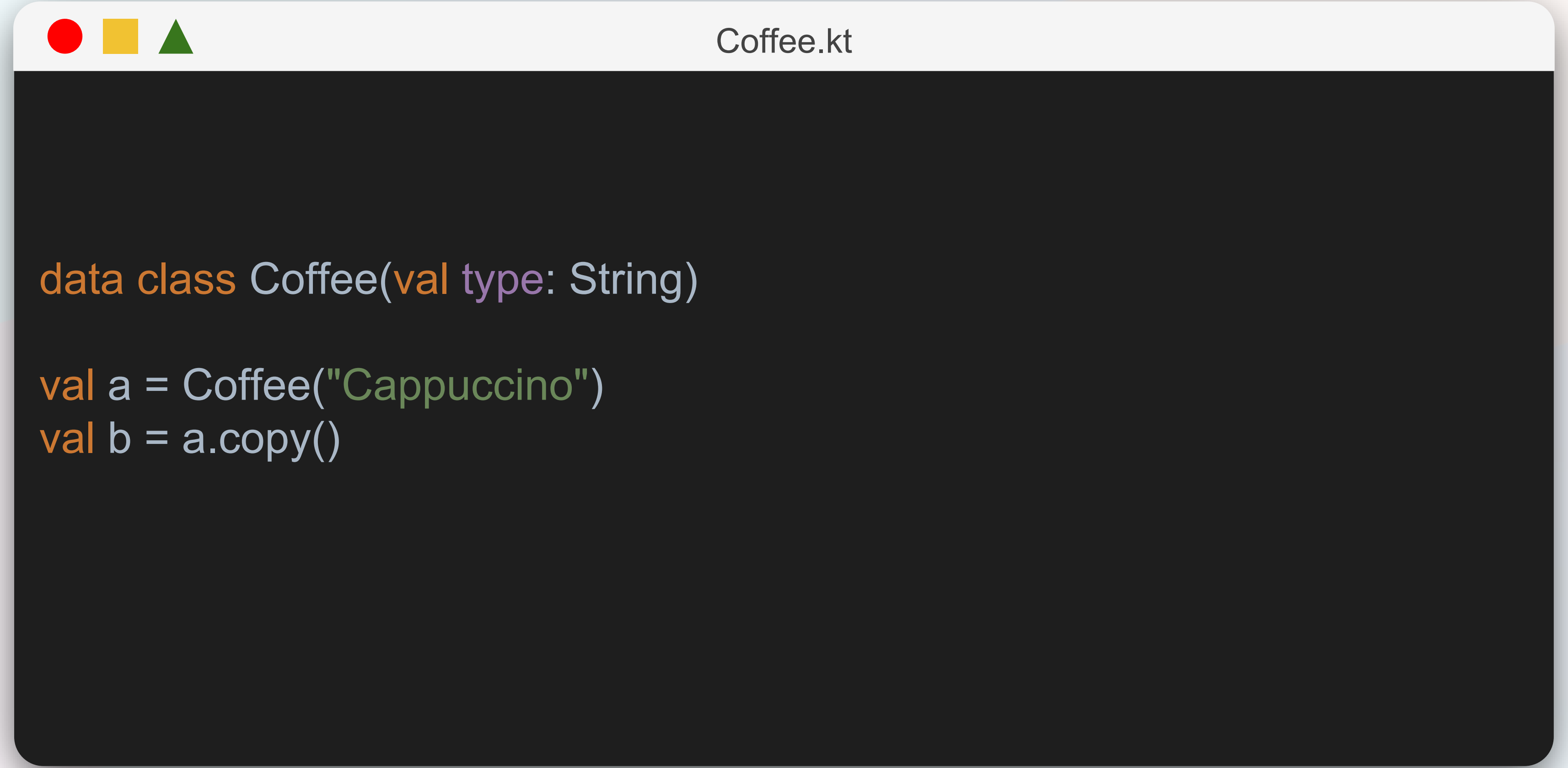
```kotlin
data class Coffee(val type: String)

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

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

Data Classes - toString()

Coffee.kt

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

val a = Coffee("Cappuccino")

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

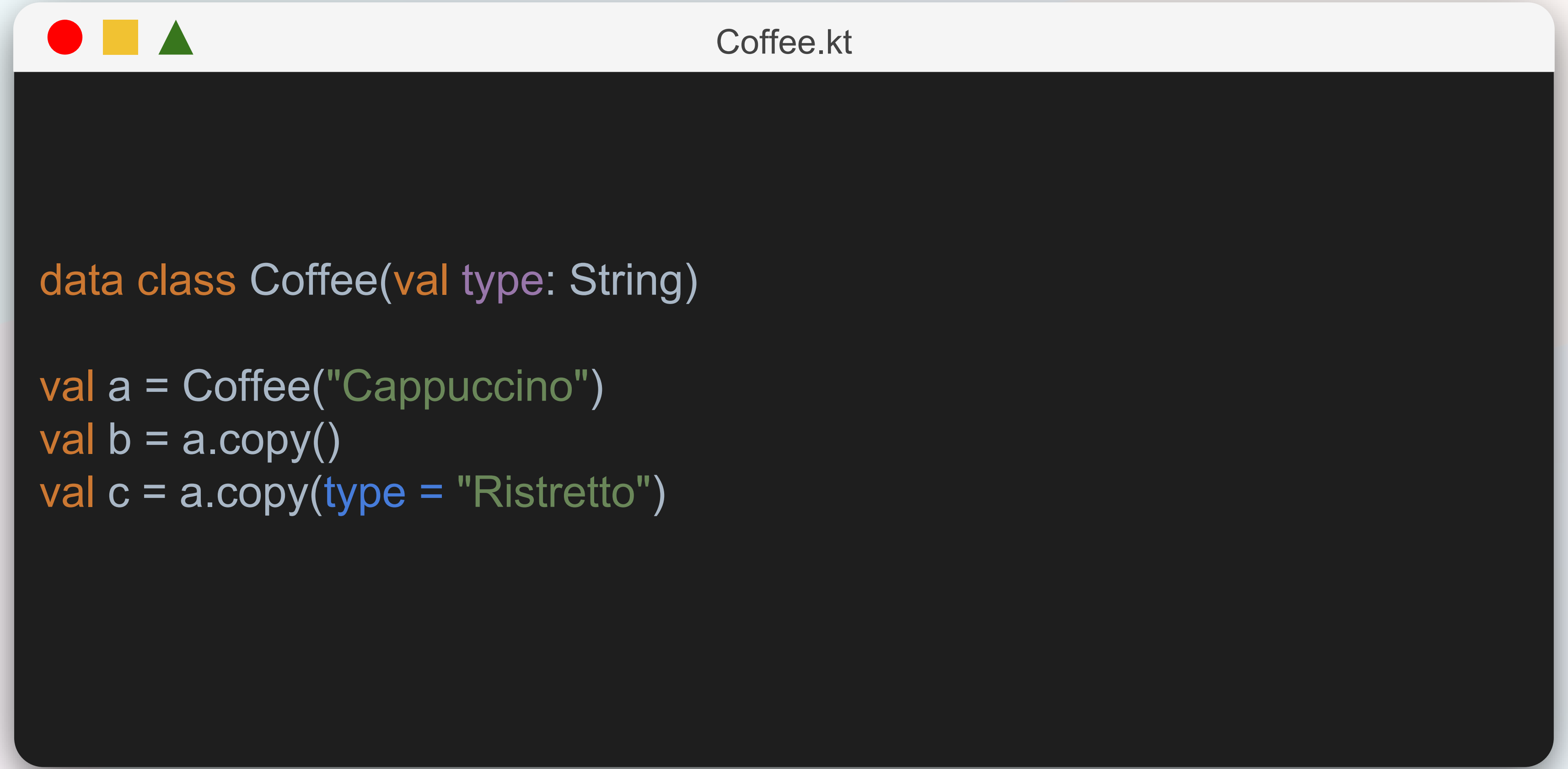
Data Classes - copy



Coffee.kt

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

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

Data Classes - copy


Coffee.kt

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

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

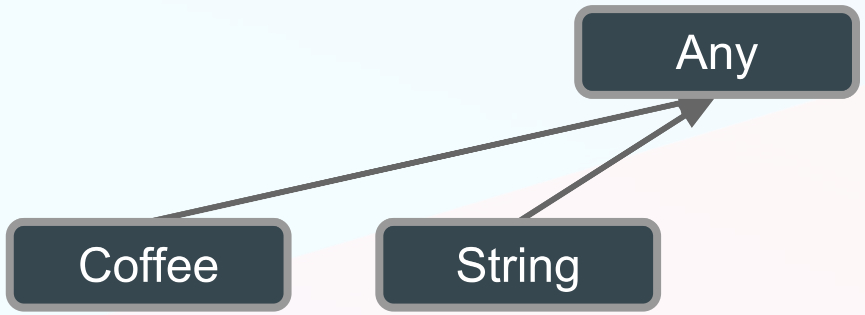
Ad Libitum

SKIP

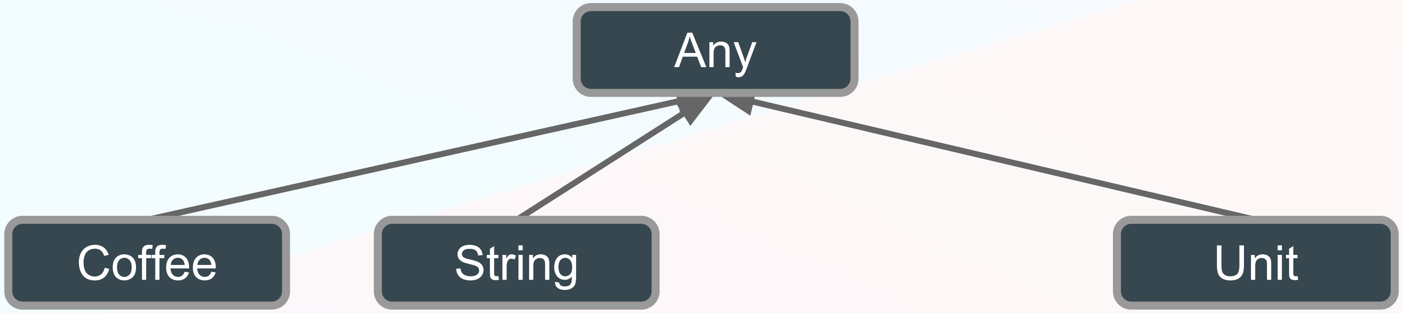
Ad Libitum

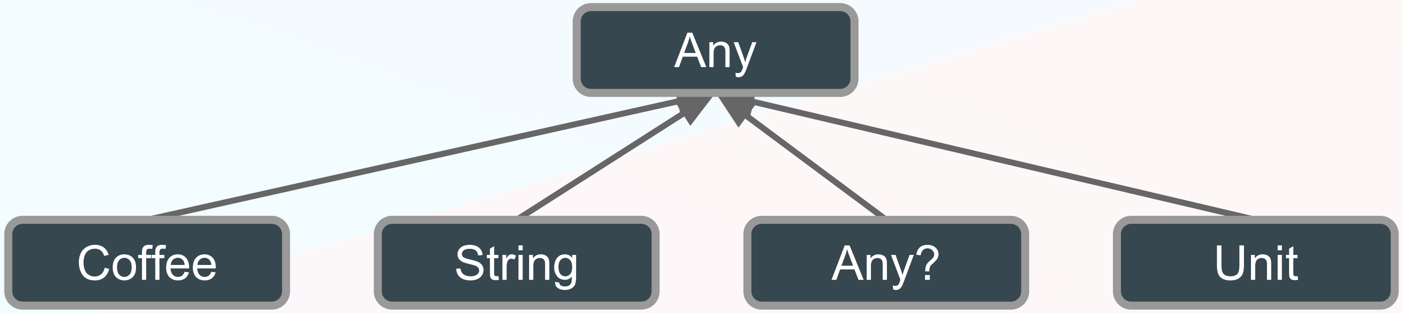
Any - Unit - Nothing

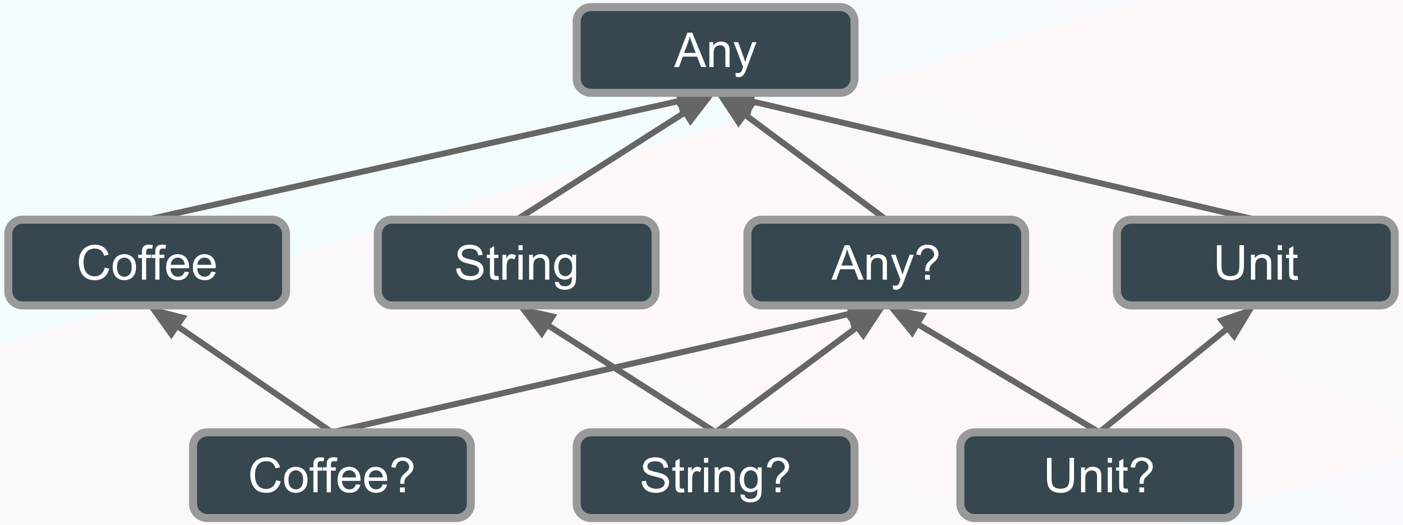
Any

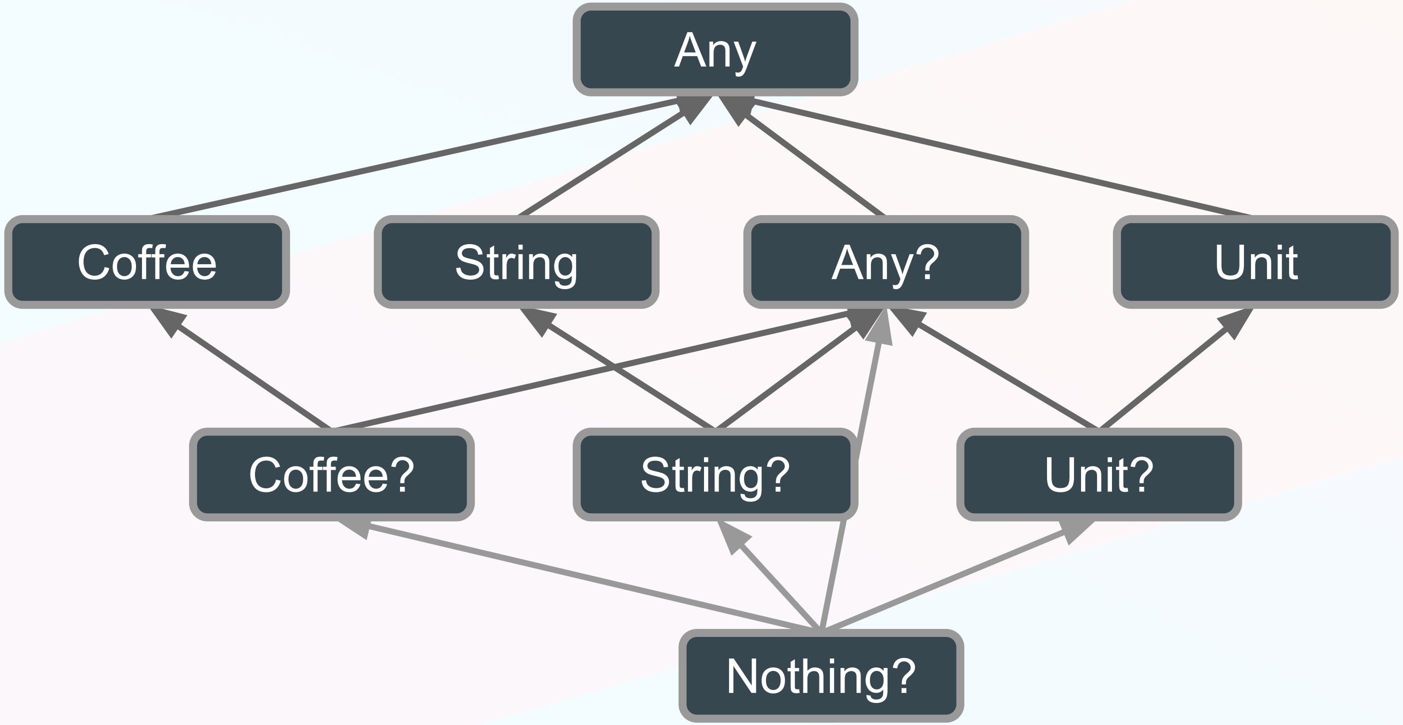

Any
Coffee
String

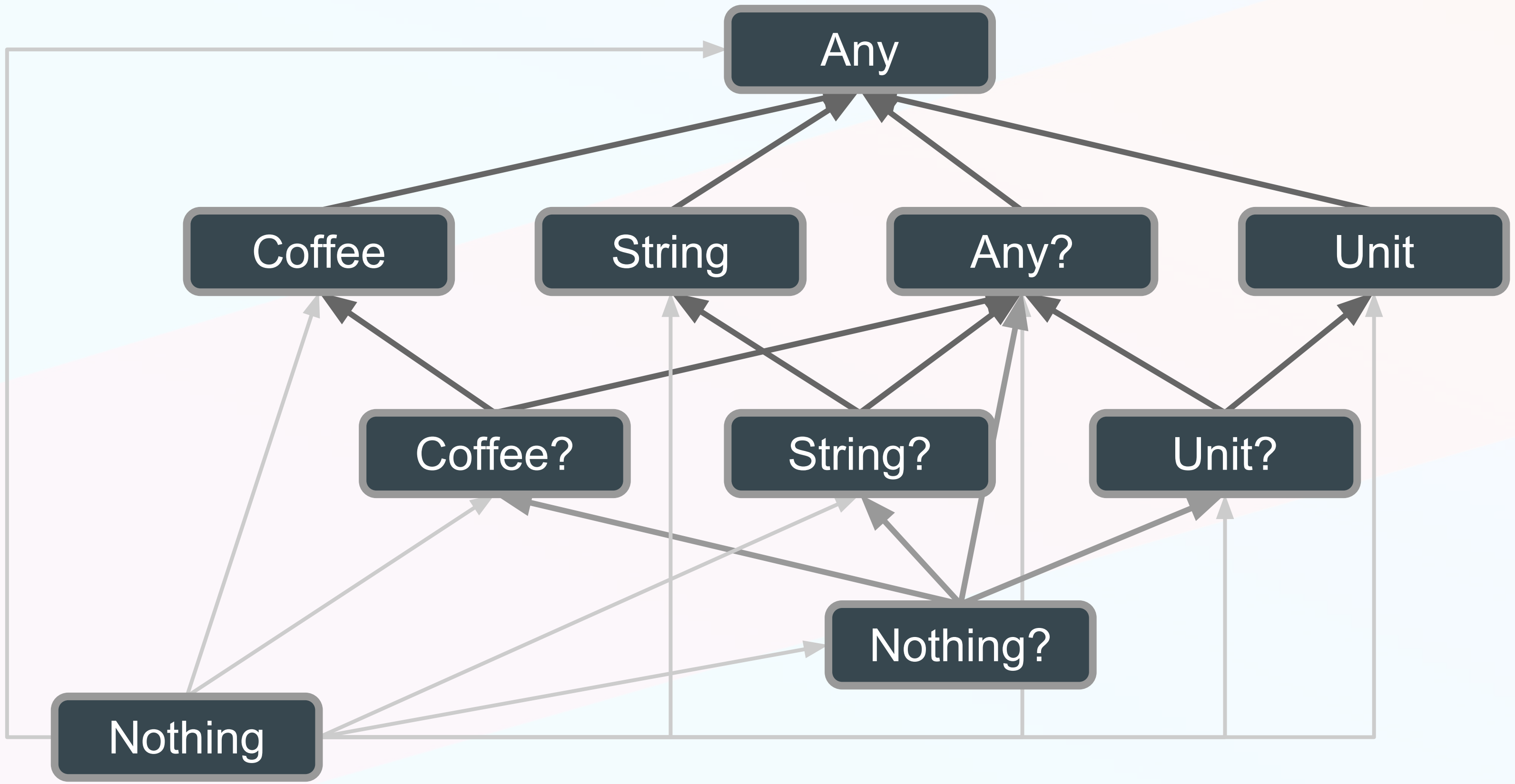
$$\begin{tikzcd}
 & \text{Any} & \\
\text{Coffee} \arrow[ur] & \text{String} \arrow[u] & \text{Unit} \arrow[ul]
\end{tikzcd}$$




Any
Coffee
String
Any?
Unit


Any
Coffee
String
Any?
Unit
Coffee?
String?
Unit?

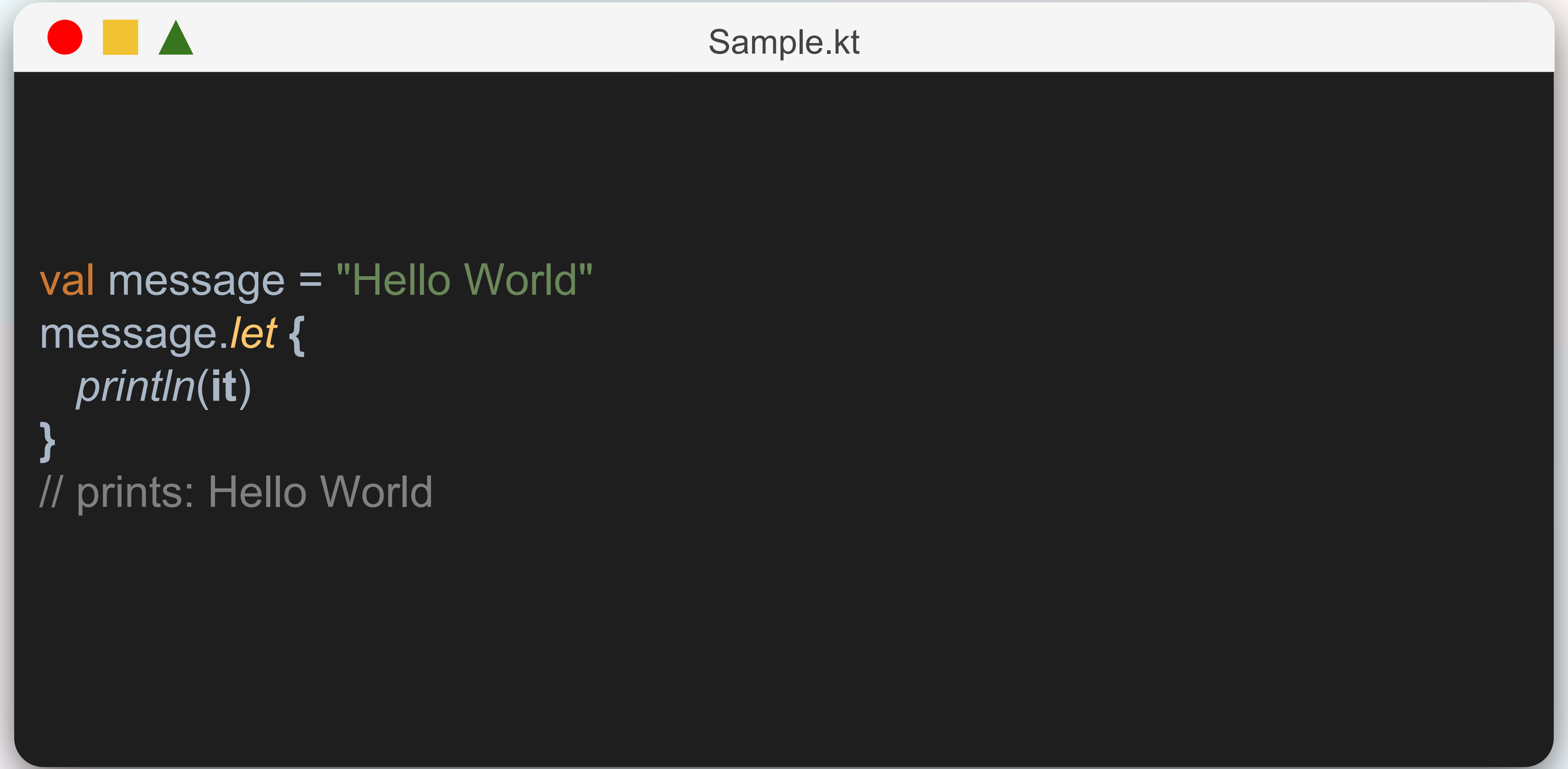

Any
Coffee
String
Any?
Unit
Coffee?
String?
Unit?
Nothing?


Any
Coffee
String
Any?
Unit
Coffee?
String?
Unit?
Nothing?
Nothing

Ad Libitum

Let

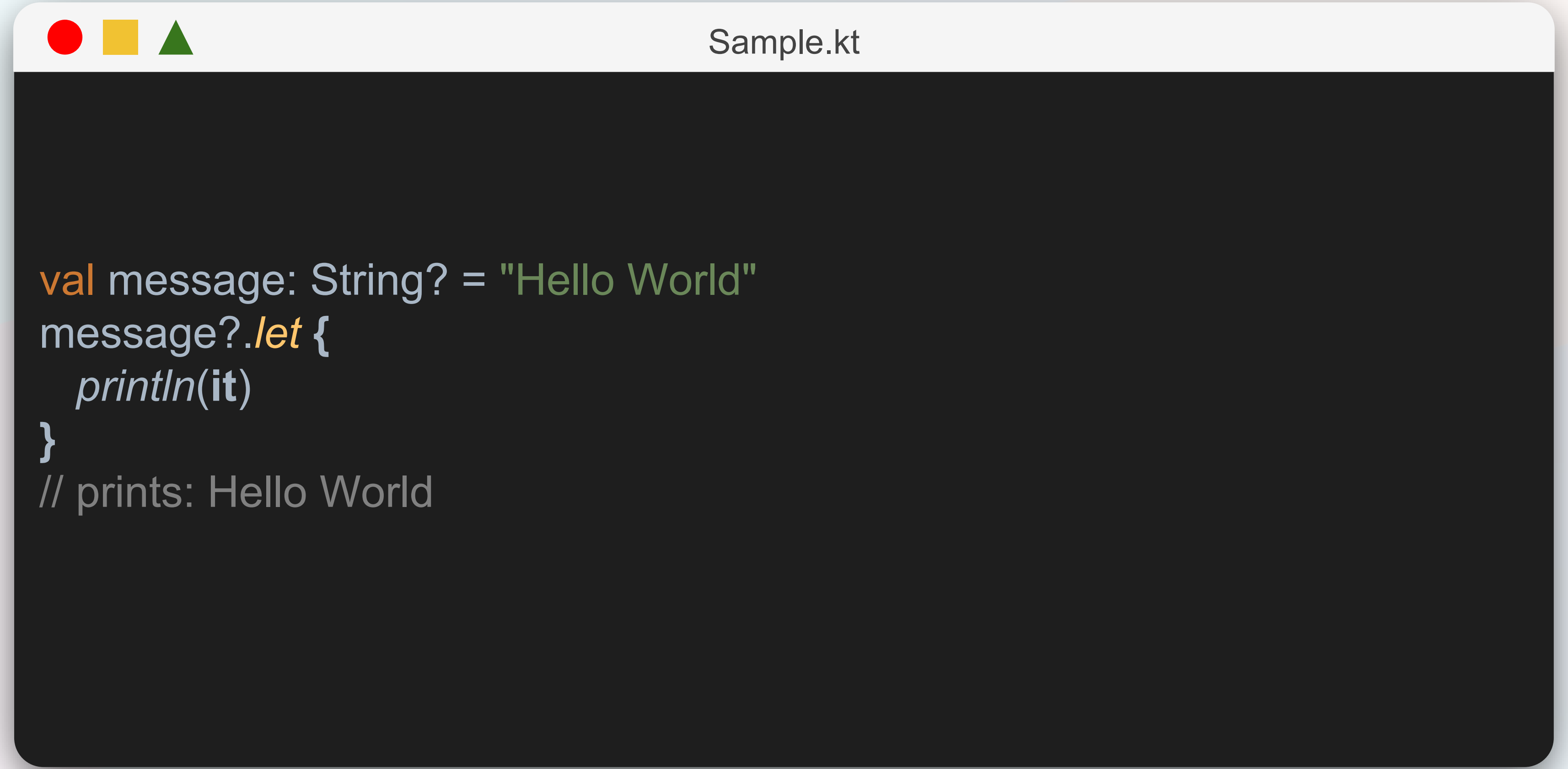
Let



Sample.kt

```
val message = "Hello World"
message.let {
   println(it)
}
// prints: Hello World
```

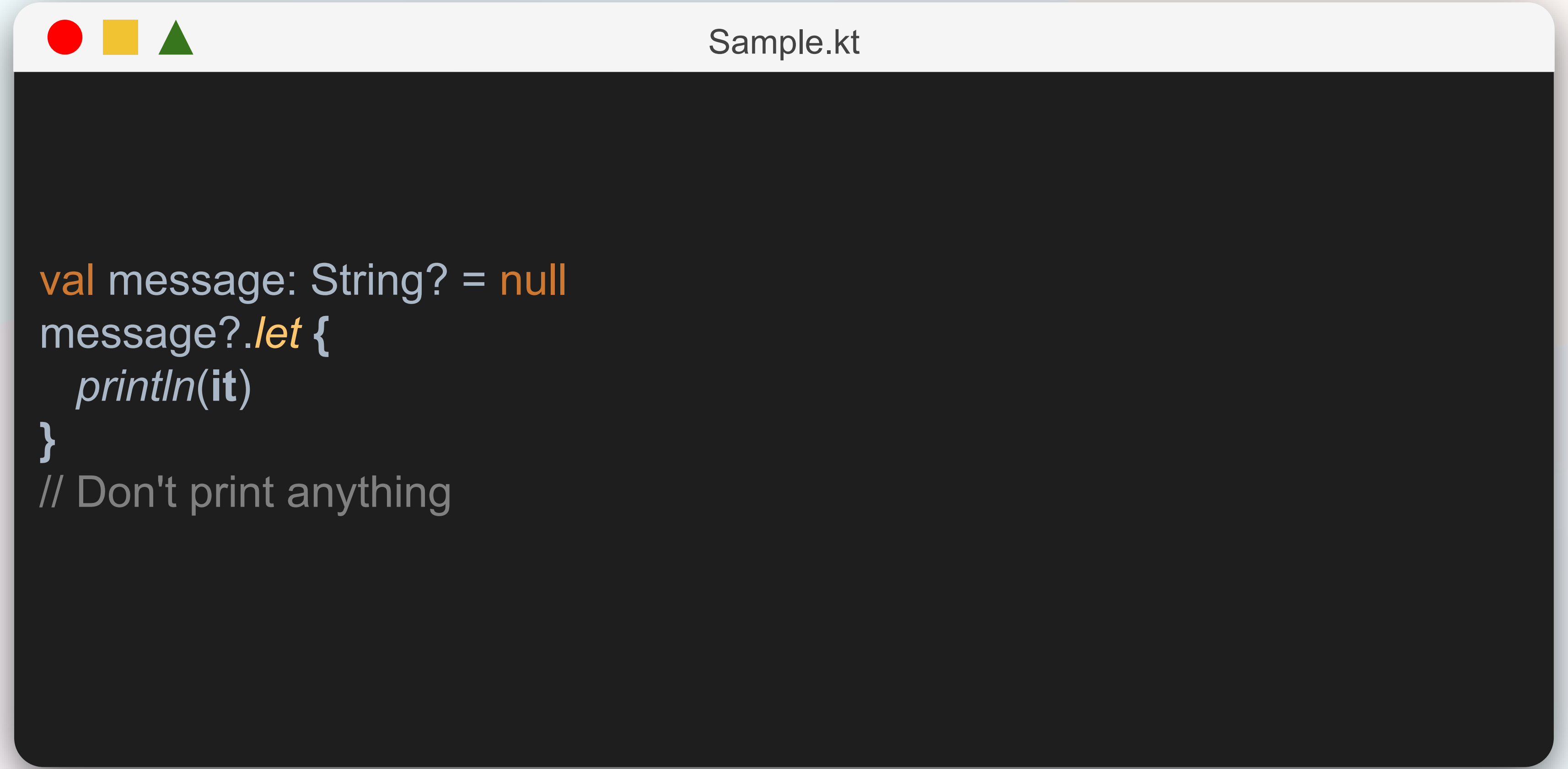
Let



Sample.kt

```kotlin
val message: String? = "Hello World"
message?.let {
    println(it)
}
// prints: Hello World
```

Let



Sample.kt

```kotlin
val message: String? = null
message?.let {
    println(it)
}
// Don't print anything
```

Ad Libitum

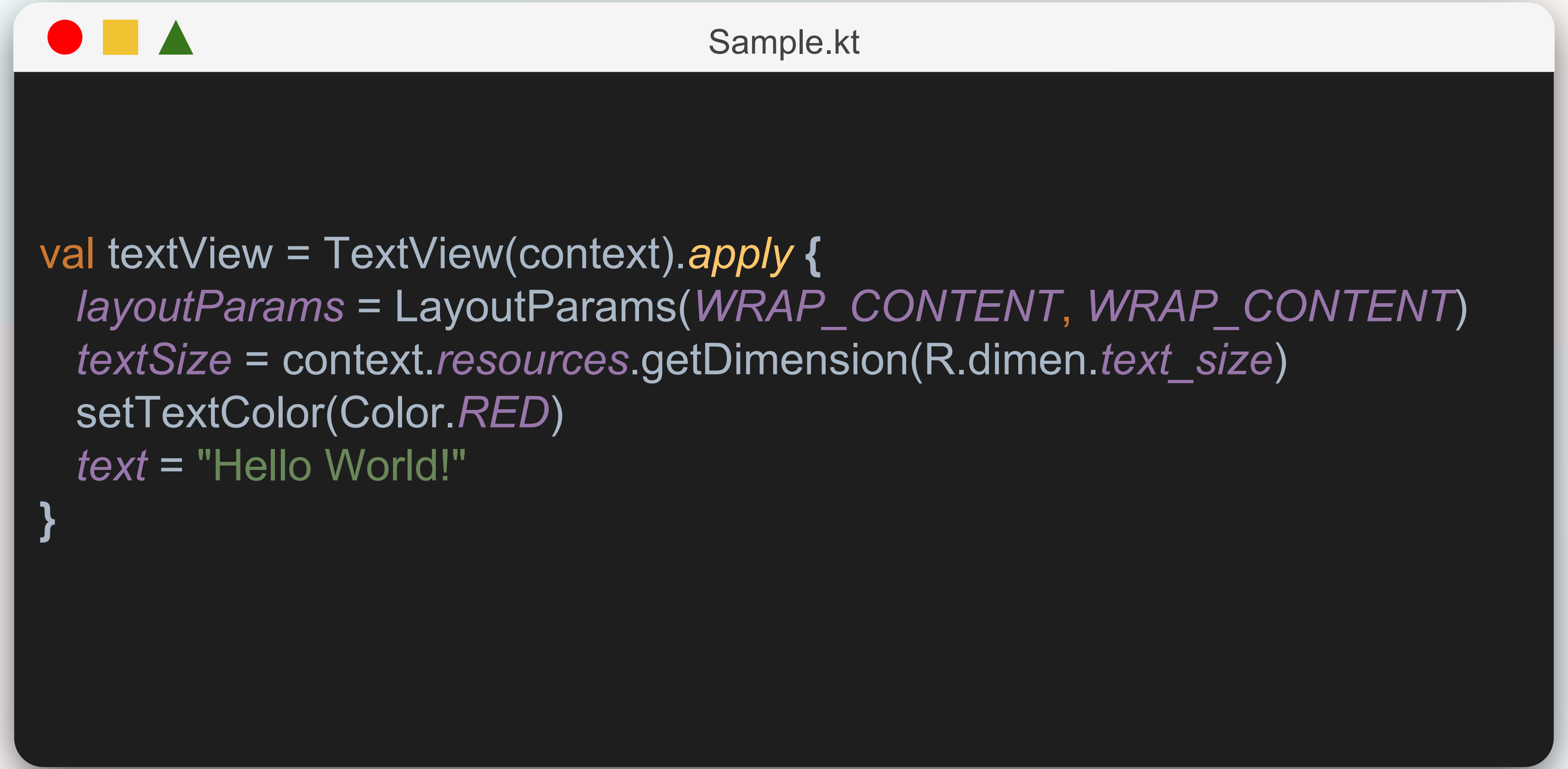
Apply

Apply

Sample.kt

```kotlin
val textView = TextView(context)
textView.layoutParams = LayoutParams(WRAP_CONTENT, WRAP_CONTENT)
textView.textSize = context.resources.getDimension(R.dimen.text_size)
textView.setTextColor(Color.RED)
textView.text = "Hello World!"
```

Apply



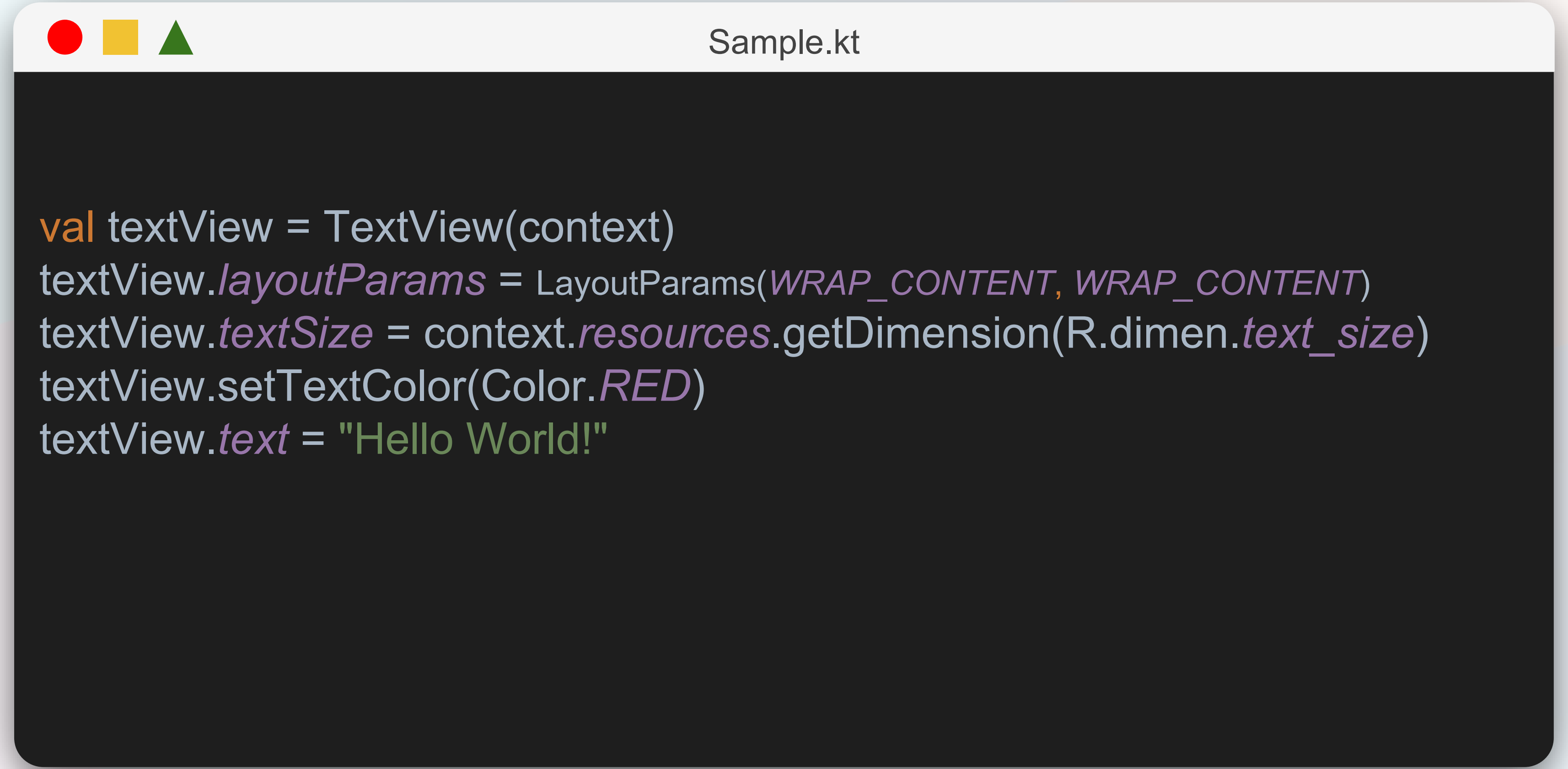
Sample.kt

```kotlin
val textView = TextView(context).apply {
   layoutParams = LayoutParams(WRAP_CONTENT, WRAP_CONTENT)
   textSize = context.resources.getDimension(R.dimen.text_size)
   setTextColor(Color.RED)
   text = "Hello World!"
}
```

Ad Libitum

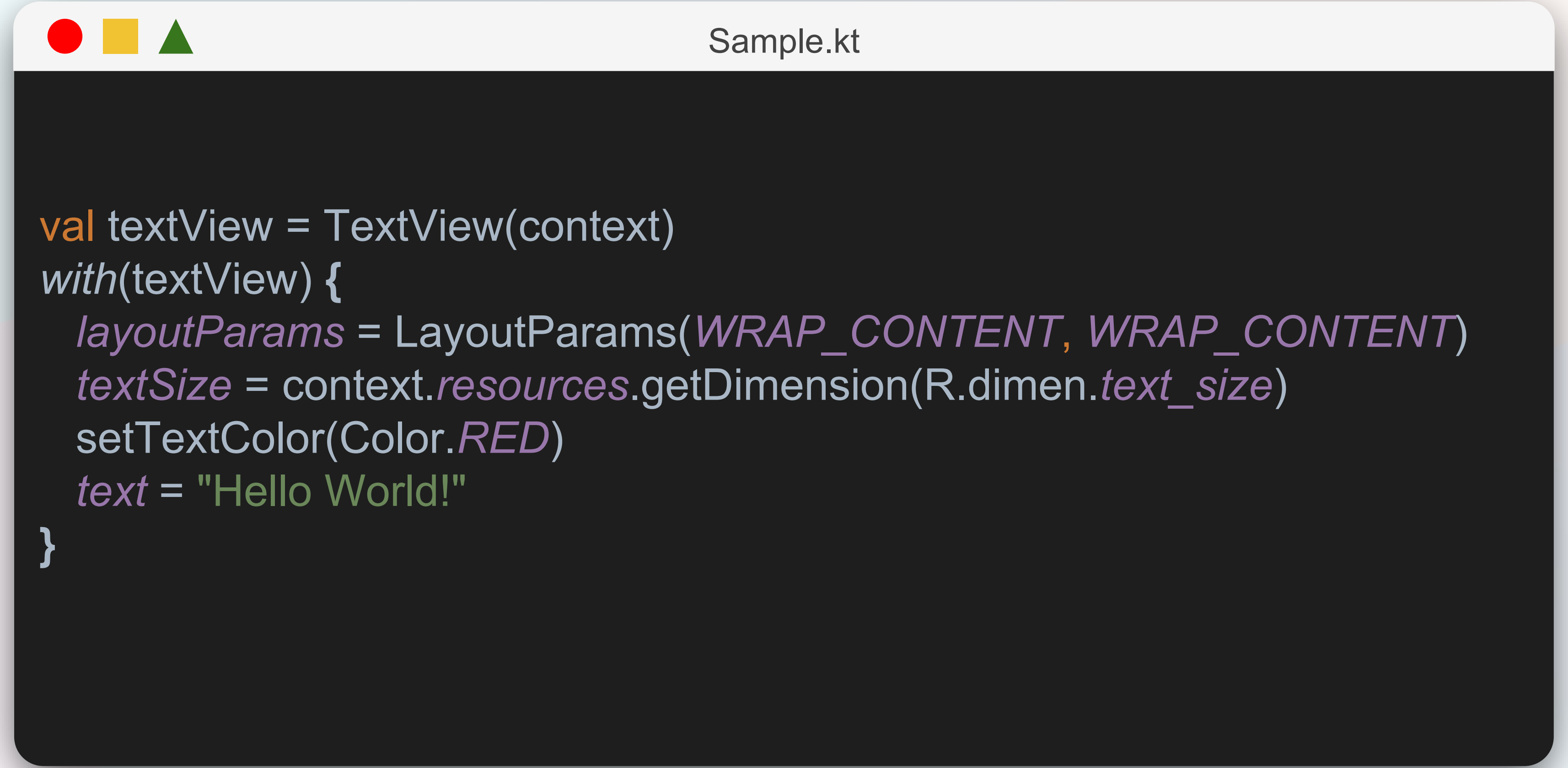
With

With


Sample.kt

```kotlin
val textView = TextView(context)
textView.layoutParams = LayoutParams(WRAP_CONTENT, WRAP_CONTENT)
textView.textSize = context.resources.getDimension(R.dimen.text_size)
textView.setTextColor(Color.RED)
textView.text = "Hello World!"
```

With



Sample.kt

```kotlin
val textView = TextView(context)
with(textView) {
   layoutParams = LayoutParams(WRAP_CONTENT, WRAP_CONTENT)
   textSize = context.resources.getDimension(R.dimen.text_size)
   setTextColor(Color.RED)
   text = "Hello World!"
}
```

Ad Libitum

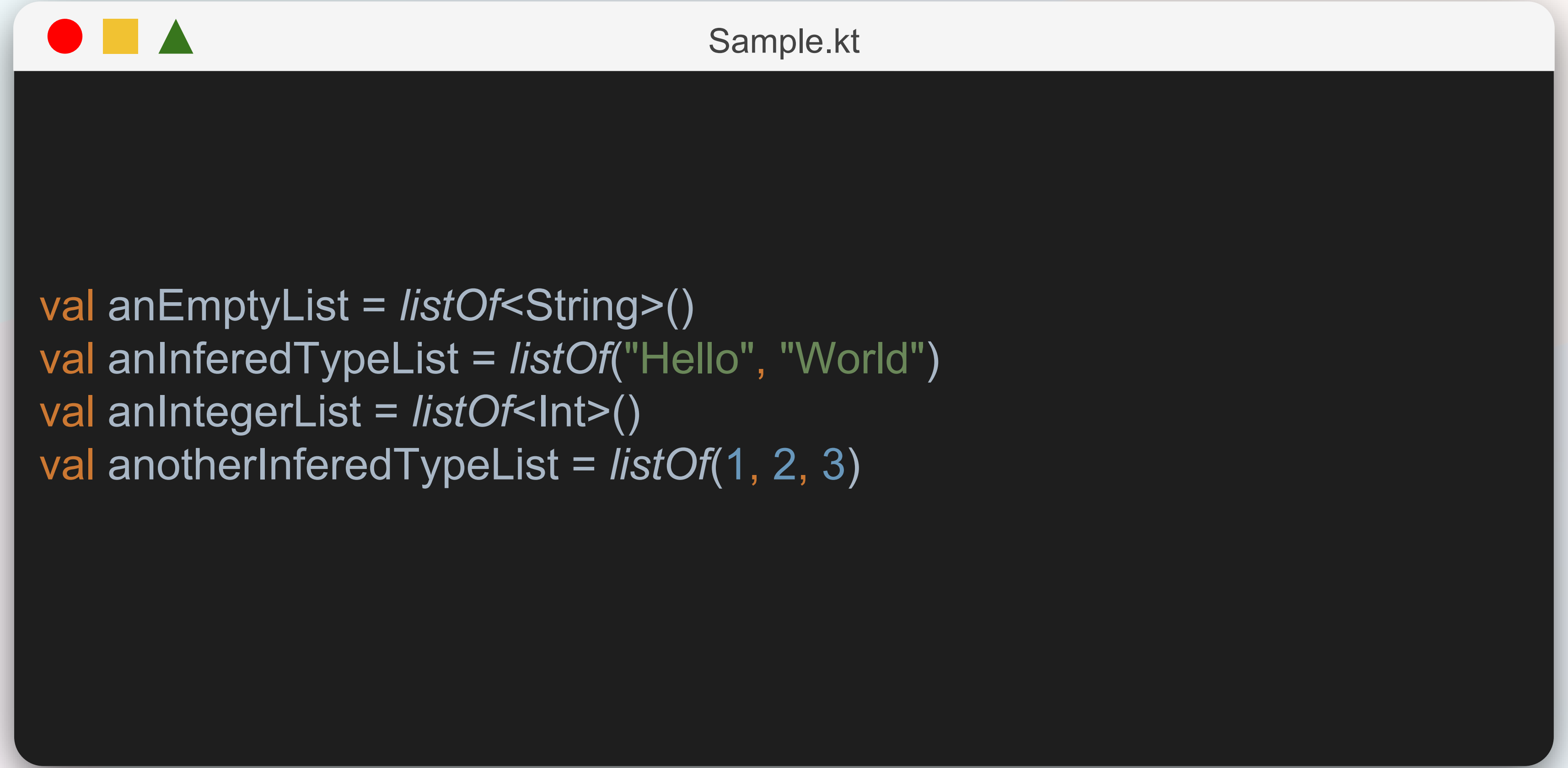
Collections

Ad Libitum

Collections

listOf - mapOf - <type>ArrayOf

Collections - listOf



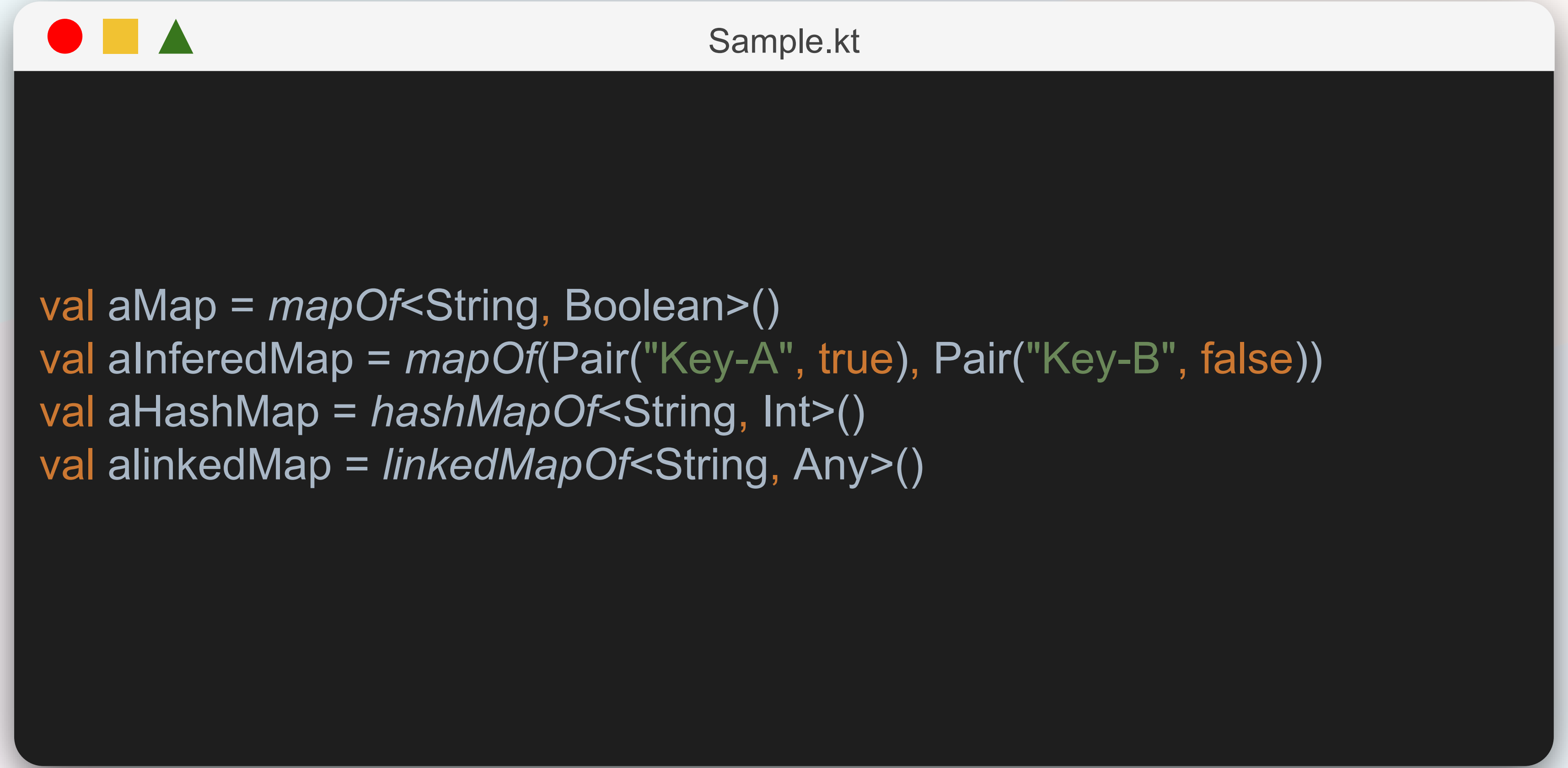
```kotlin
val anEmptyList = listOf<String>()
val anInferedTypeList = listOf("Hello", "World")
val anIntegerList = listOf<Int>()
val anotherInferedTypeList = listOf(1, 2, 3)
```

Collections - <type>ArrayOf

Sample.kt

```kotlin
val aPrimitiveIntArray = intArrayOf(1, 2, 3)
val aPrimitiveByteArray = byteArrayOf()
val aPrimitiveBooleanArray = booleanArrayOf()
```

Collections - mapOf


Sample.kt

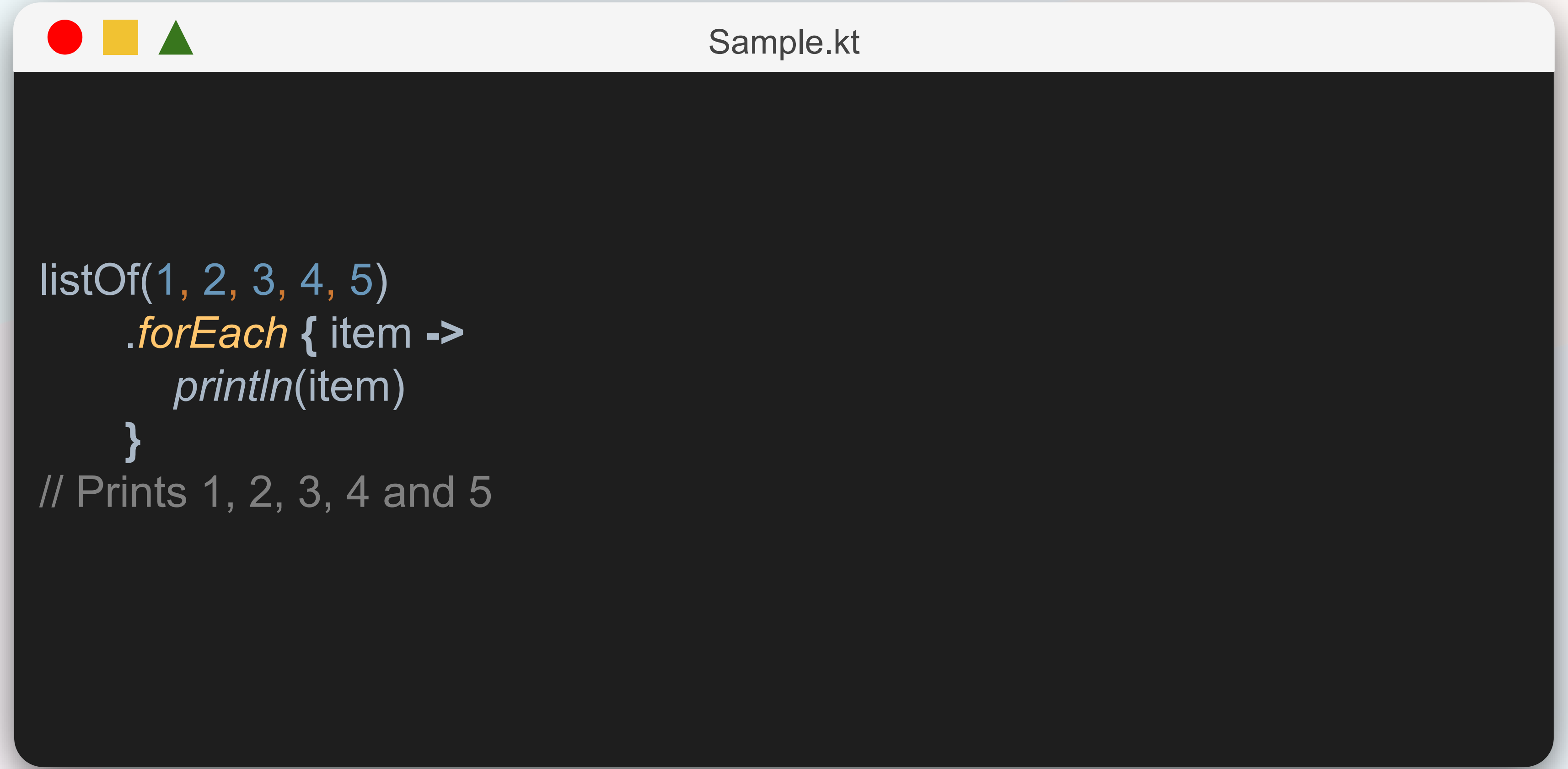
```
val aMap = mapOf<String, Boolean>()
val aInferedMap = mapOf(Pair("Key-A", true), Pair("Key-B", false))
val aHashMap = hashMapOf<String, Int>()
val alinkedMap = linkedMapOf<String, Any>()
```

Ad Libitum

Collections

Operators

Collections - Operators - Foreach

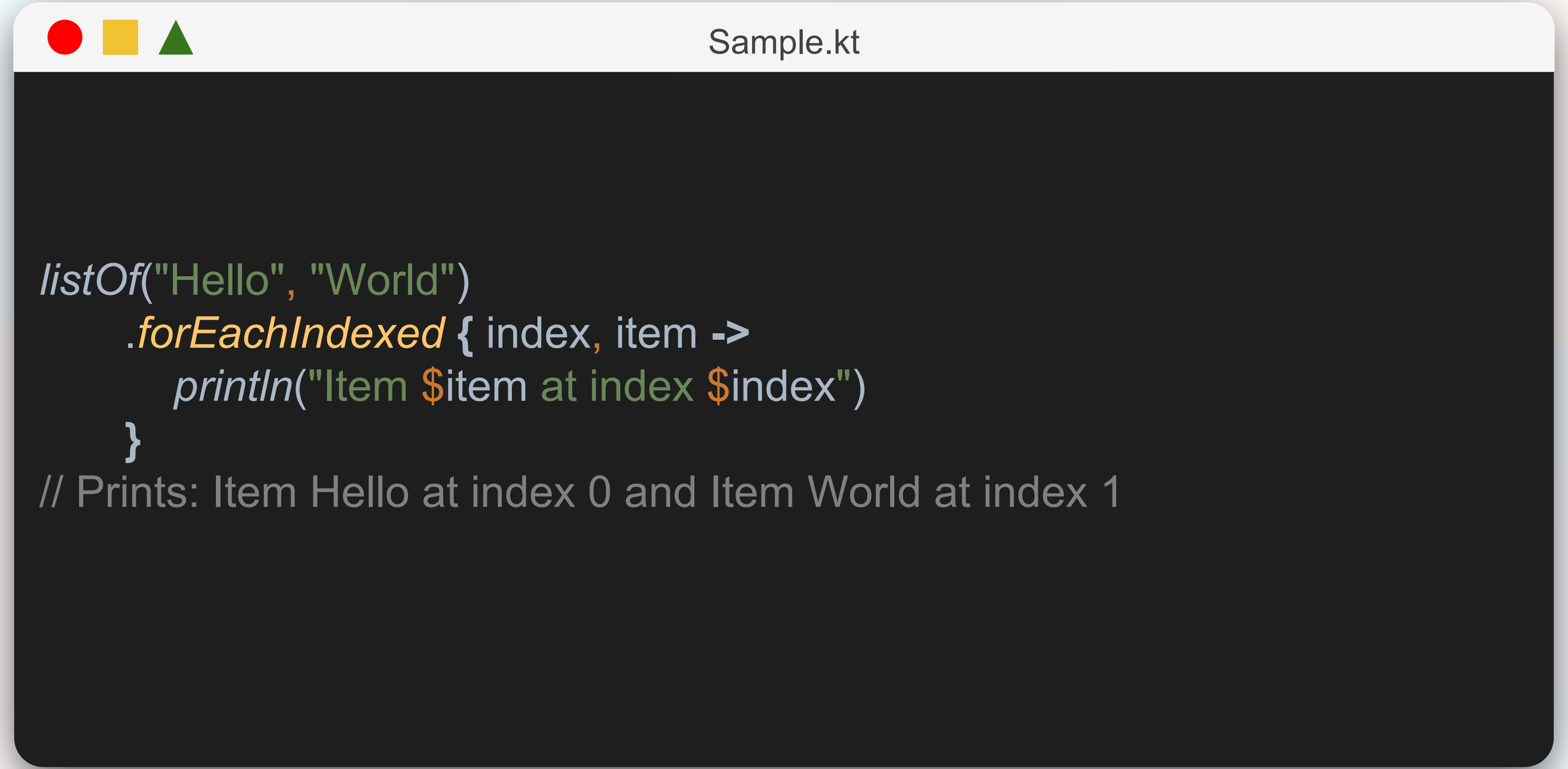

Sample.kt

```
listOf(1, 2, 3, 4, 5)
      .forEach { item ->
          println(item)
      }
// Prints 1, 2, 3, 4 and 5
```

Collections - Operators - Foreach

Sample.kt

```kotlin
listOf(1, 2, 3, 4, 5)
        .forEach {
            println(it)
        }
// Prints 1, 2, 3, 4 and 5
```


Sample.kt

```kotlin
listOf("Hello", "World")
        .forEachIndexed { index, item ->
            println("Item $item at index $index")
        }
// Prints: Item Hello at index 0 and Item World at index 1
```

Collections - Operators - Map

Sample.kt

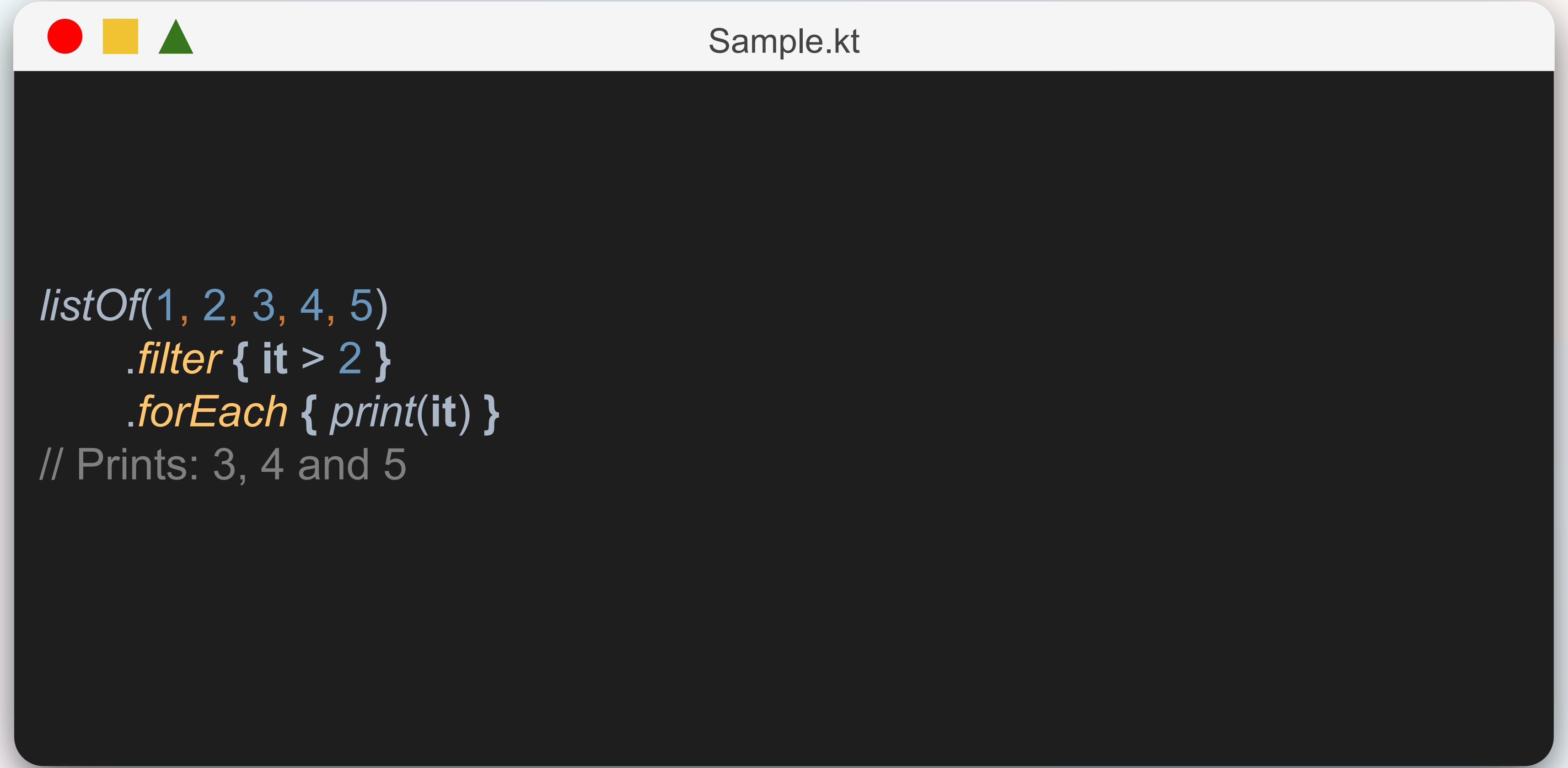
```kotlin
listOf(1, 2, 3, 4, 5)
        .map { it * 2 }
        .forEach { println(it) }
// Prints 2, 4, 6, 8 and 10
```

Collections - Operators - Sort

Sample.kt

```kotlin
listOf(5, 3, 1, 4, 2)
        .sortedBy { it }
        .forEach { println(it) }
// Prints 1, 2, 3, 4 and 5
```

Collections - Operators - Filter

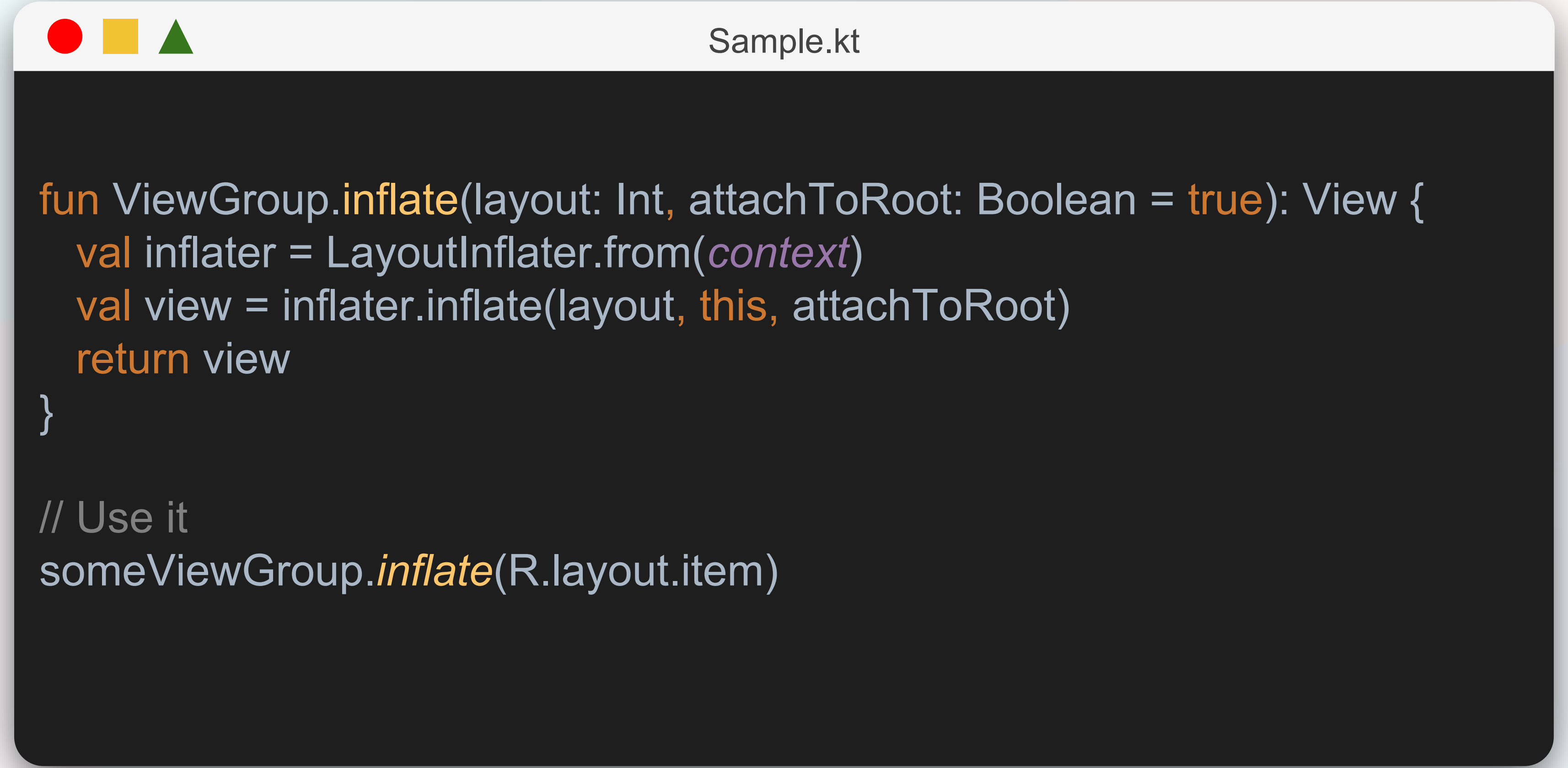

Sample.kt

```kotlin
listOf(1, 2, 3, 4, 5)
        .filter { it > 2 }
        .forEach { print(it) }
// Prints: 3, 4 and 5
```

Ad Libitum

Extensions

Extensions


Sample.kt

```kotlin
fun ViewGroup.inflate(layout: Int, attachToRoot: Boolean = true): View {
   val inflater = LayoutInflater.from(context)
   val view = inflater.inflate(layout, this, attachToRoot)
   return view
}

// Use it
someViewGroup.inflate(R.layout.item)
```

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