DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Entwicklung mobiler Applikationen

WS2022/2023

Jan Brodhaecker | November 2022

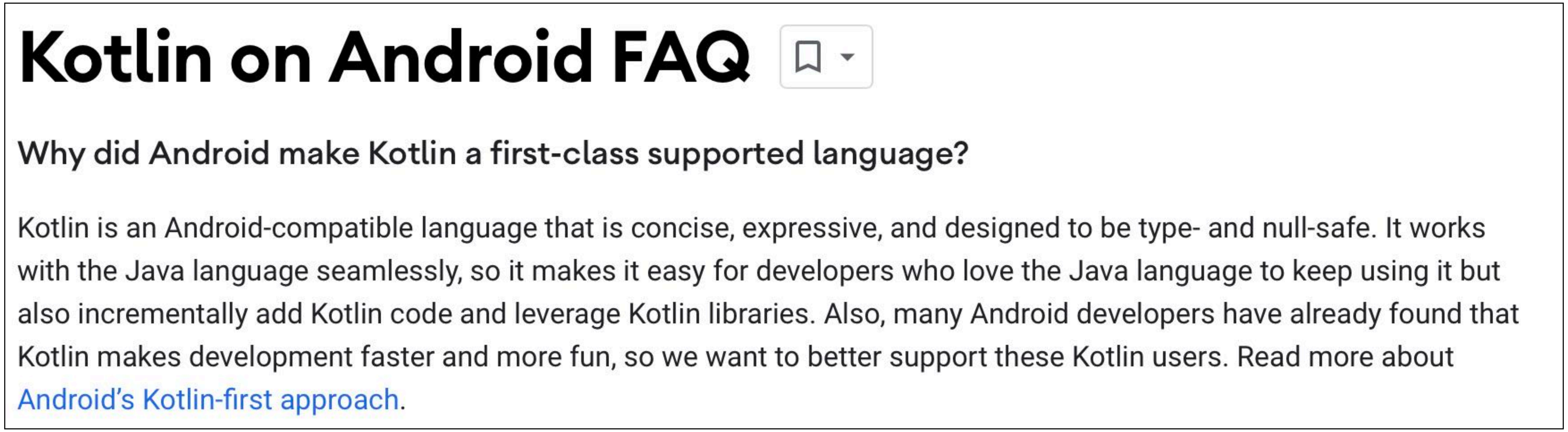
Agenda

https://developer.android.com/static/codelabs/basic-android-kotlin-compose-first-program/img/3bbebda874e6003b.png

Kotlin

- seit Google I/O 2017 neben Java offizielle Programmiersprache für Android
  - Motivation vermutlich Rechtsstreit seit 2010 zwischen Google und Oracle (https://www.droidwiki.org/wiki/Rechtsstreit_zwischen_Google_und_Oracle)
- von der Firma JetBrains (u.a IntelliJ IDE)
- Version 1.0 im Februar 2016 (entwickelt seit 2010, erstmals veröffentlicht seit 2011)
- aktuelle Version v1.7.20 (10/2022)
- open-source (https://github.com/jetbrains/kotlin)
- Programme wird in Java Virtual Machine ausgeführt

Kotlin



Kotlin on Android FAQ

Why did Android make Kotlin a first-class supported language?

Kotlin is an Android-compatible language that is concise, expressive, and designed to be type- and null-safe. It works with the Java language seamlessly, so it makes it easy for developers who love the Java language to keep using it but also incrementally add Kotlin code and leverage Kotlin libraries. Also, many Android developers have already found that Kotlin makes development faster and more fun, so we want to better support these Kotlin users. Read more about Android's Kotlin-first approach.

https://developer.android.com/kotlin/faq

Kotlin

- soll ~40% Quellcode einsparen
  - type-checks
  - null-safe
  - smart casts
  - lambdas
  - ….

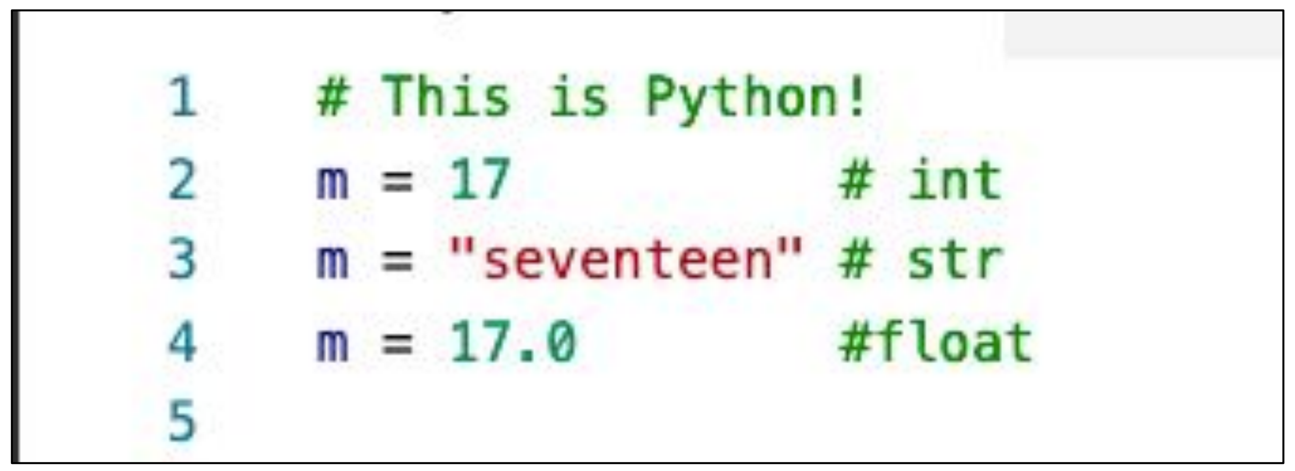
Kotlin

- … ist eine statisch typisierte Programmiersprache

https://knowyourmeme.com/memes/confused-nick-young

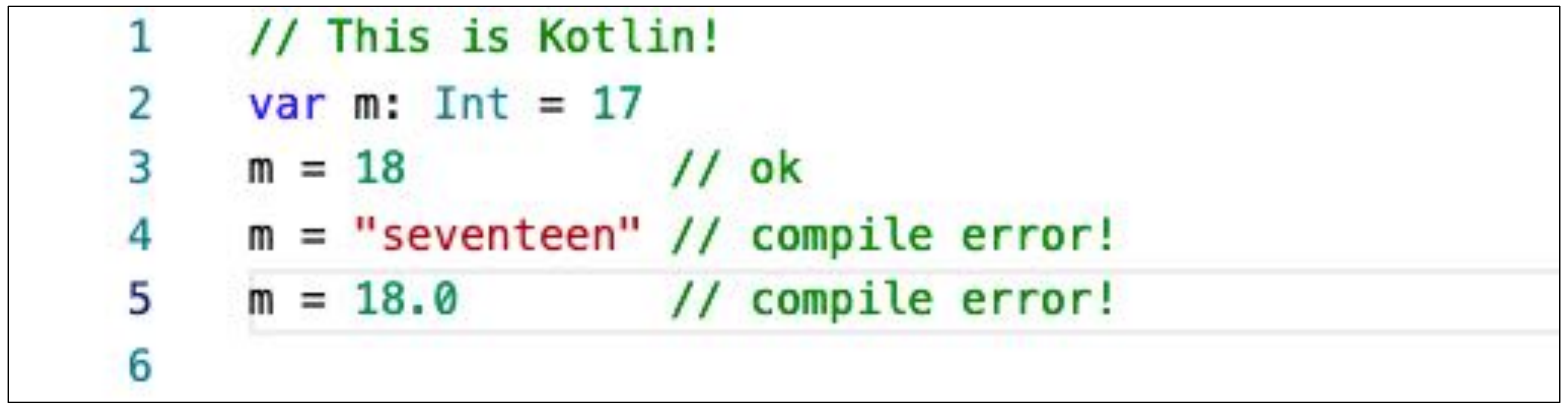
Kotlin

- das Gegenteil wäre eine dynamisch typisierte Programmiersprache
- Typisierung wird zur *compile time* (statisch) bzw. zur *runtime* (dynamisch) geprüft



```python
# This is Python!
m = 17          # int
m = "seventeen" # str
m = 17.0        #float

```

dynamisch



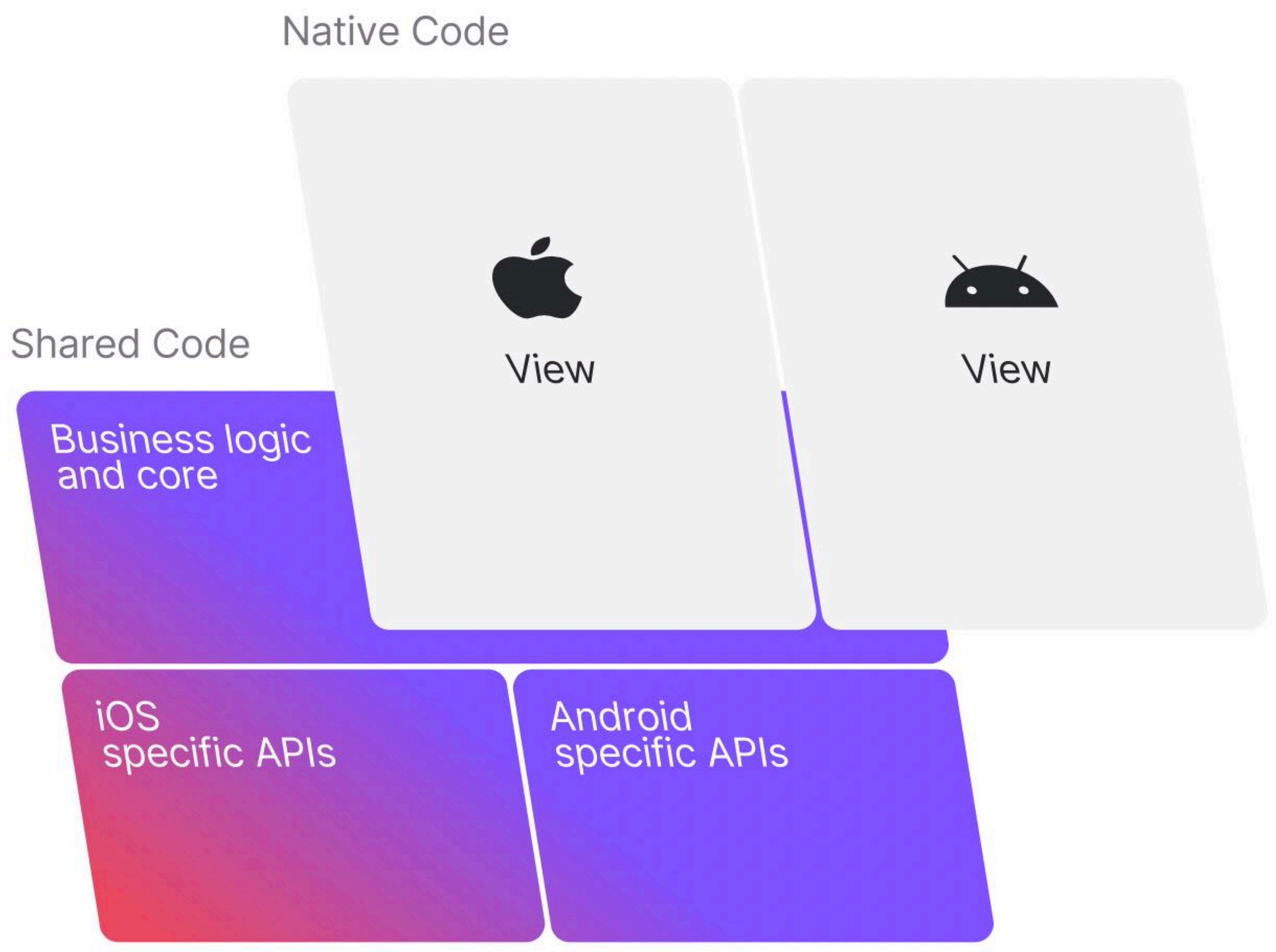
```kotlin
// This is Kotlin!
var m: Int = 17
m = 18            // ok
m = "seventeen" // compile error!
m = 18.0          // compile error!

```

statisch

Kotlin

Einsatzgebiete

- Multiplatform (beta)
  - erlaubt Code zu teilen bspw. zwischen bzw. für
    - iOS/Android
    - Fullstack Applikationen
    - multi-Plattform Libraries
    - mobile- and Web-Applikationen



https://kotlinlang.org/lp/mobile/

Kotlin
Einsatzgebiete

- Server-Side
  - Expressiveness
  - Scalability
  - Interoperability
  - Migration
  - Tooling
  - Learning Curve

https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSu6GSLwZWFN9kOMfG31qEoWKN4yyrCOjurAg&usqp=CAU

https://raw.githubusercontent.com/ktorio/ktor/main/.github/images/ktor-logo-for-light.svg

Kotlin

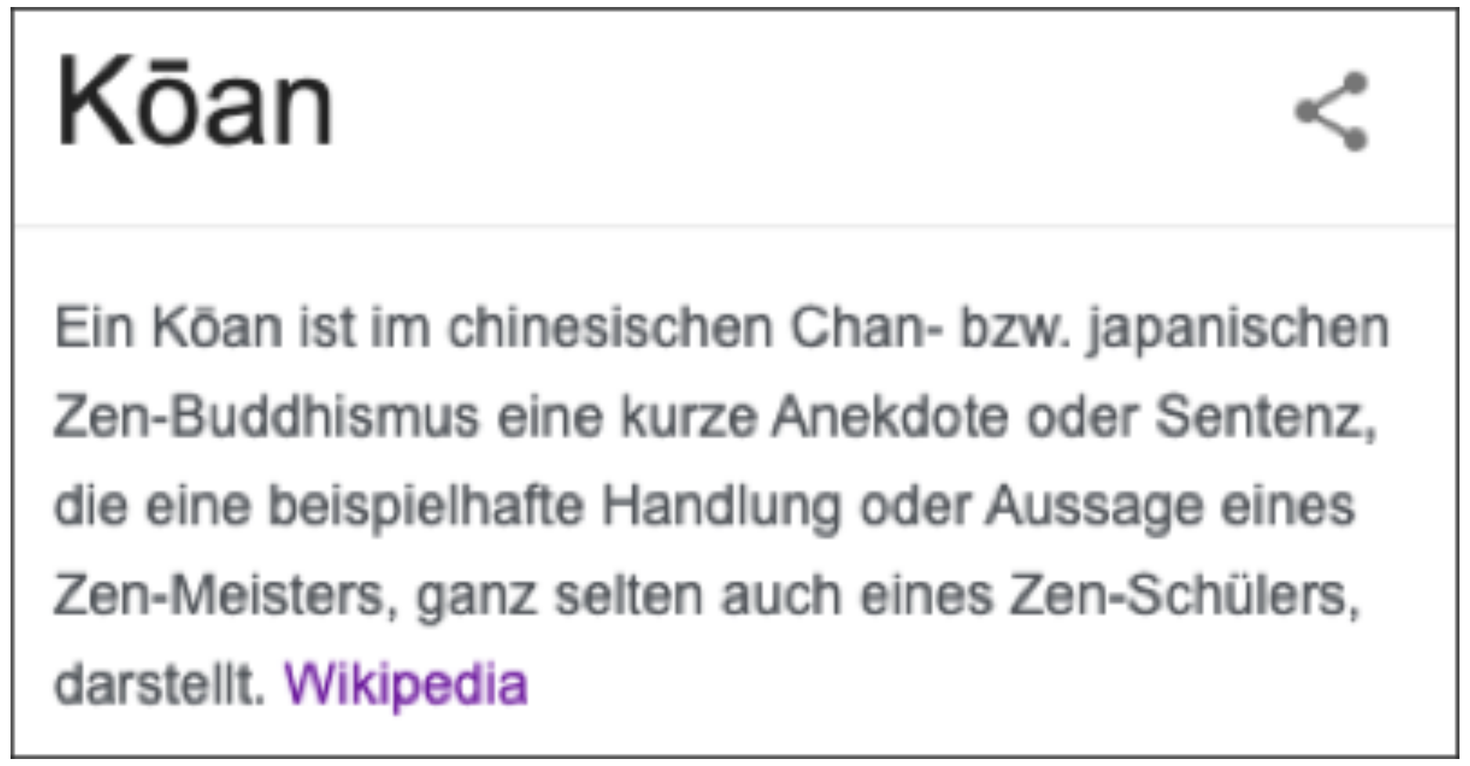
Einsatzgebiete

- Android
- für JavaScript
  - transpiles into JavaScript
- Kotlin Native
  - compiles into native Binaries (können ohne VM ausgeführt werden)
  - u.a für macOS, iOS, tvOS, watchOS, Linux, Windows, …
- Data Science (Jupyter, Datalore, Zeppelin, und viele Frameworks …)
- competitive programming

Kotlin

Ressourcen

- Kotlin im Vergleich zu Java: https://kotlinlang.org/docs/comparison-to-java.html
- Koans: https://play.kotlinlang.org/koans/overview
- kostenloser Cousera Kurs: https://www.coursera.org/learn/kotlin-for-java-developers?action=enroll&authMode=signup
- Playground: https://play.kotlinlang.org/


Kōan

Ein Kōan ist im chinesischen Chan- bzw. japanischen Zen-Buddhismus eine kurze Anekdote oder Sentenz, die eine beispielhafte Handlung oder Aussage eines Zen-Meisters, ganz selten auch eines Zen-Schülers, darstellt. Wikipedia

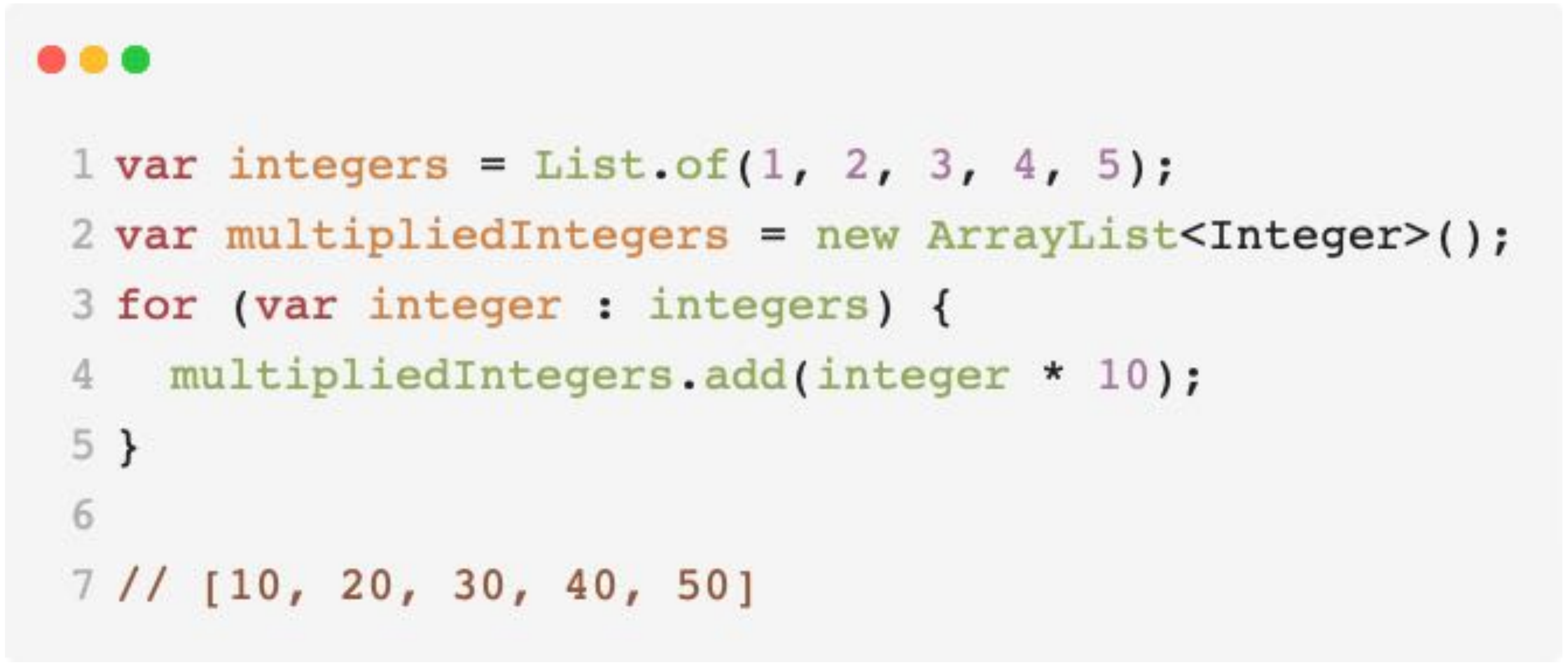
https://g.co/kgs/GfLGu1

Kotlin

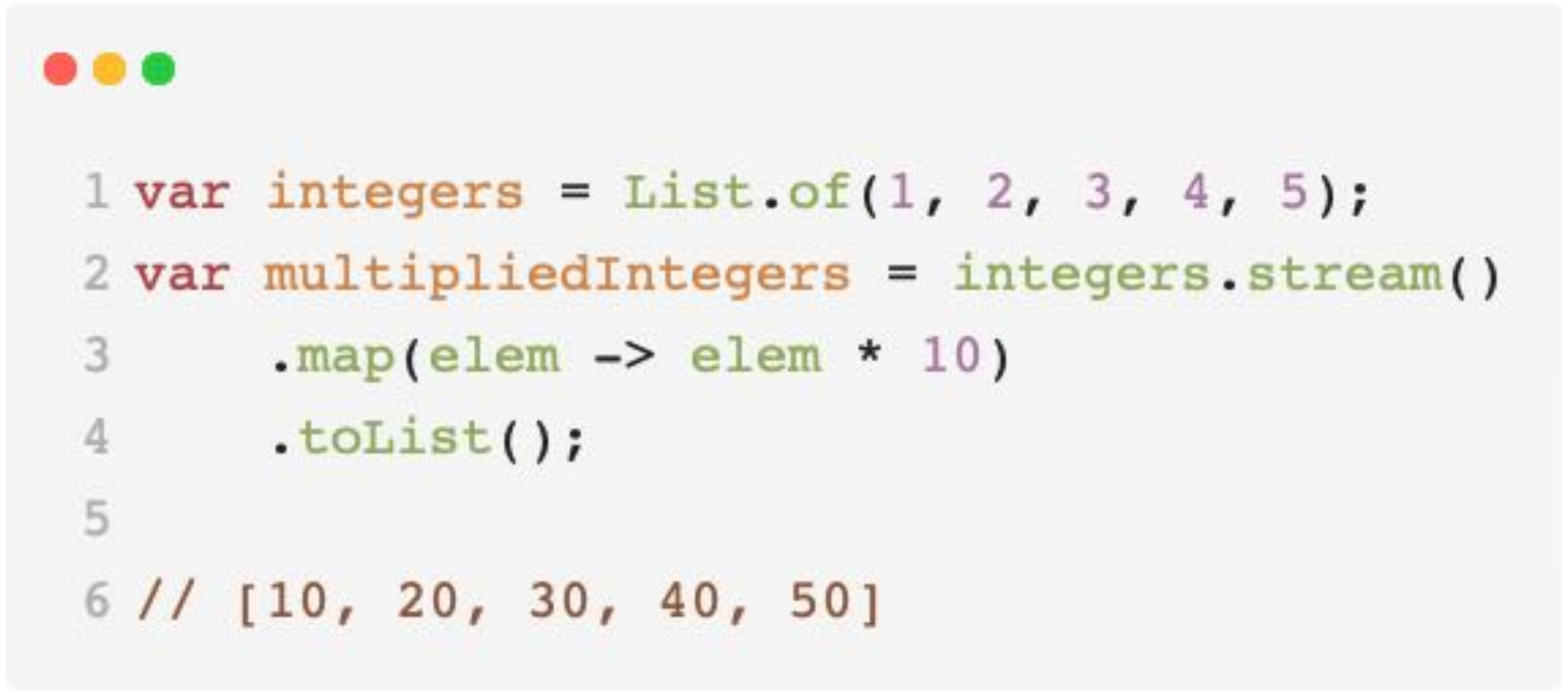
Exkurs zu Streams

A sequence of elements supporting sequential and parallel aggregate operations

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html



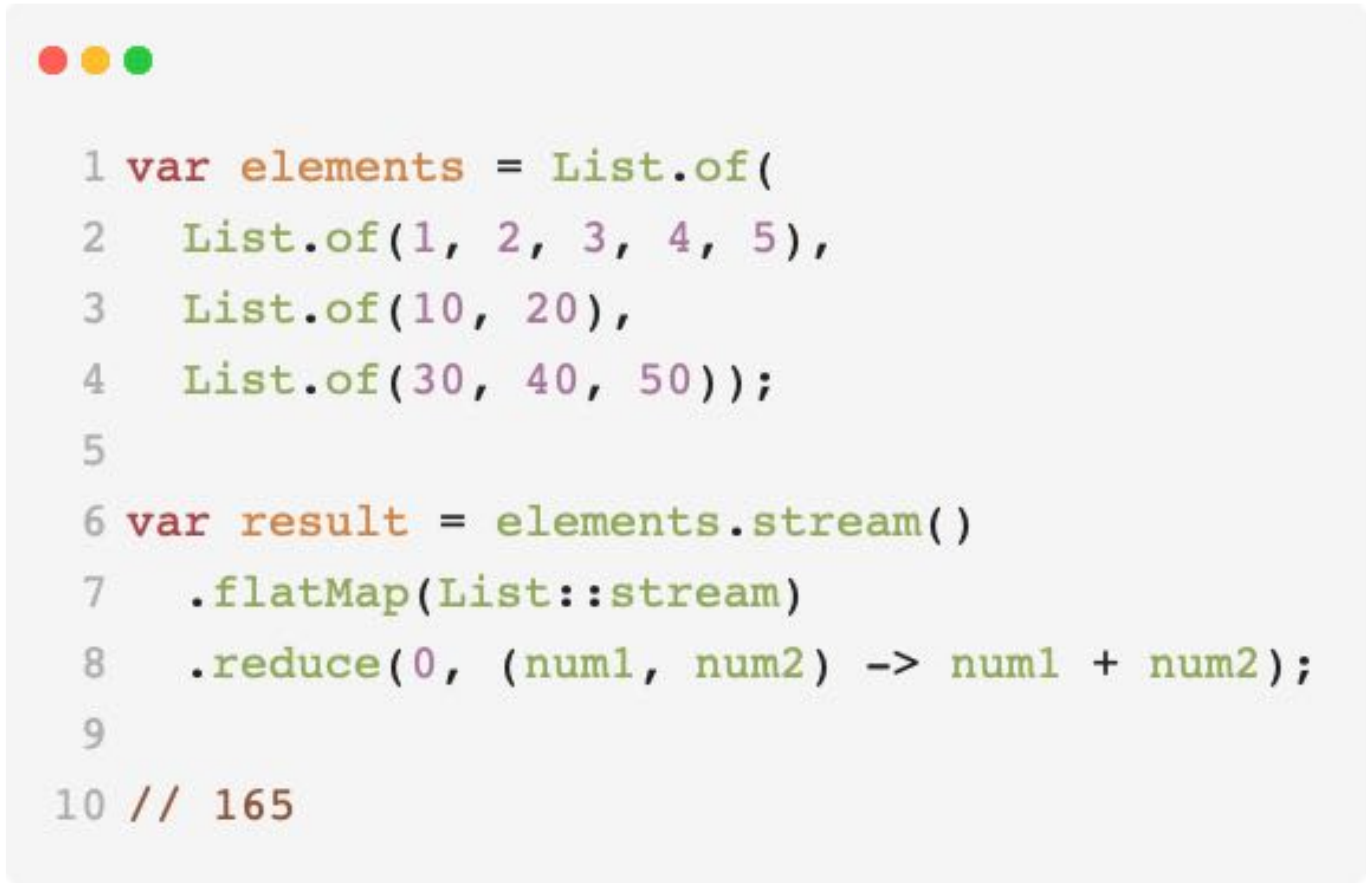
```java
var integers = List.of(1, 2, 3, 4, 5);
var multipliedIntegers = new ArrayList<Integer>();
for (var integer : integers) {
  multipliedIntegers.add(integer * 10);
}

// [10, 20, 30, 40, 50]
```



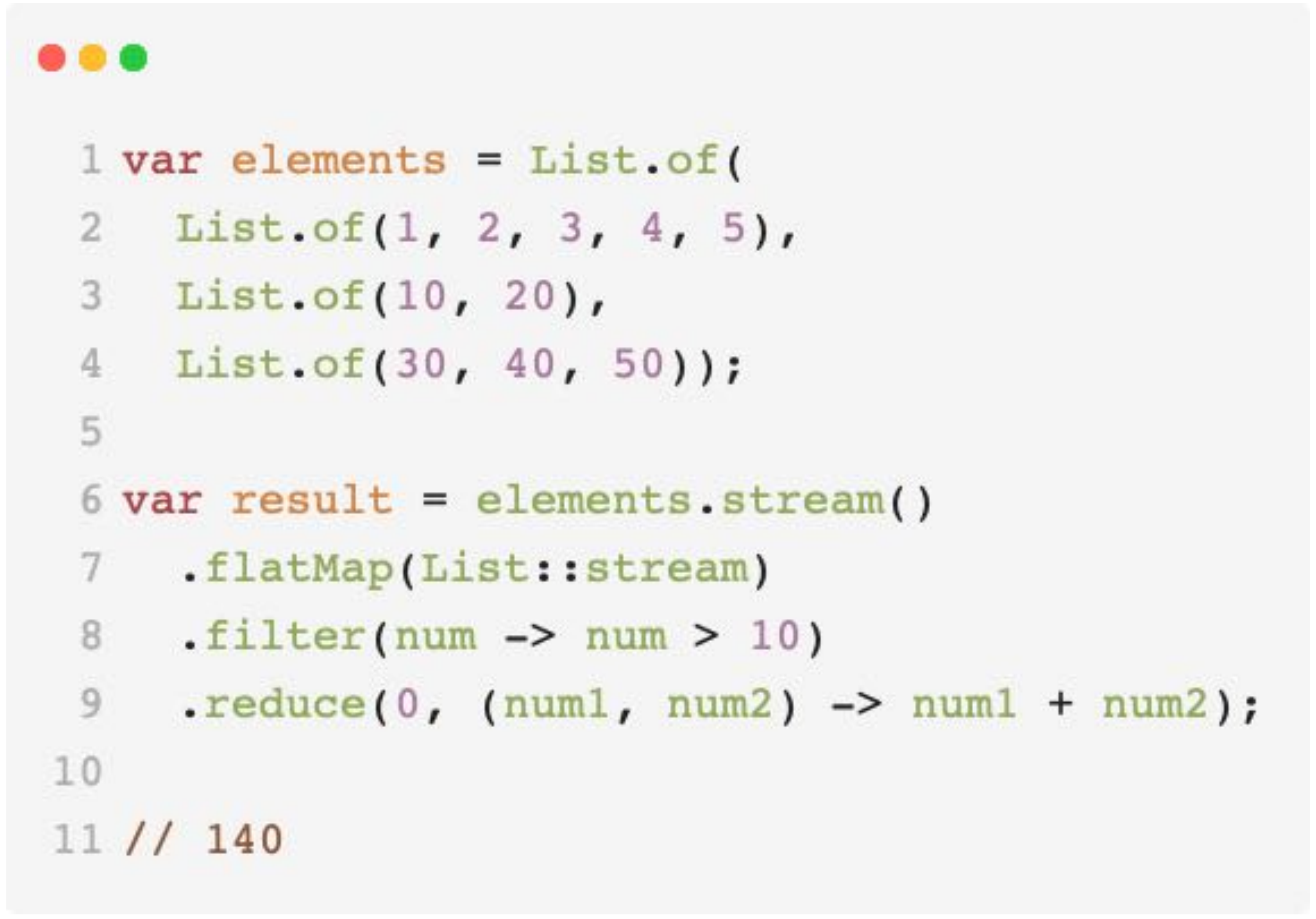
```java
var integers = List.of(1, 2, 3, 4, 5);
var multipliedIntegers = integers.stream()
     .map(elem -> elem * 10)
     .toList();

// [10, 20, 30, 40, 50]
```

Kotlin

Exkurs zu Streams

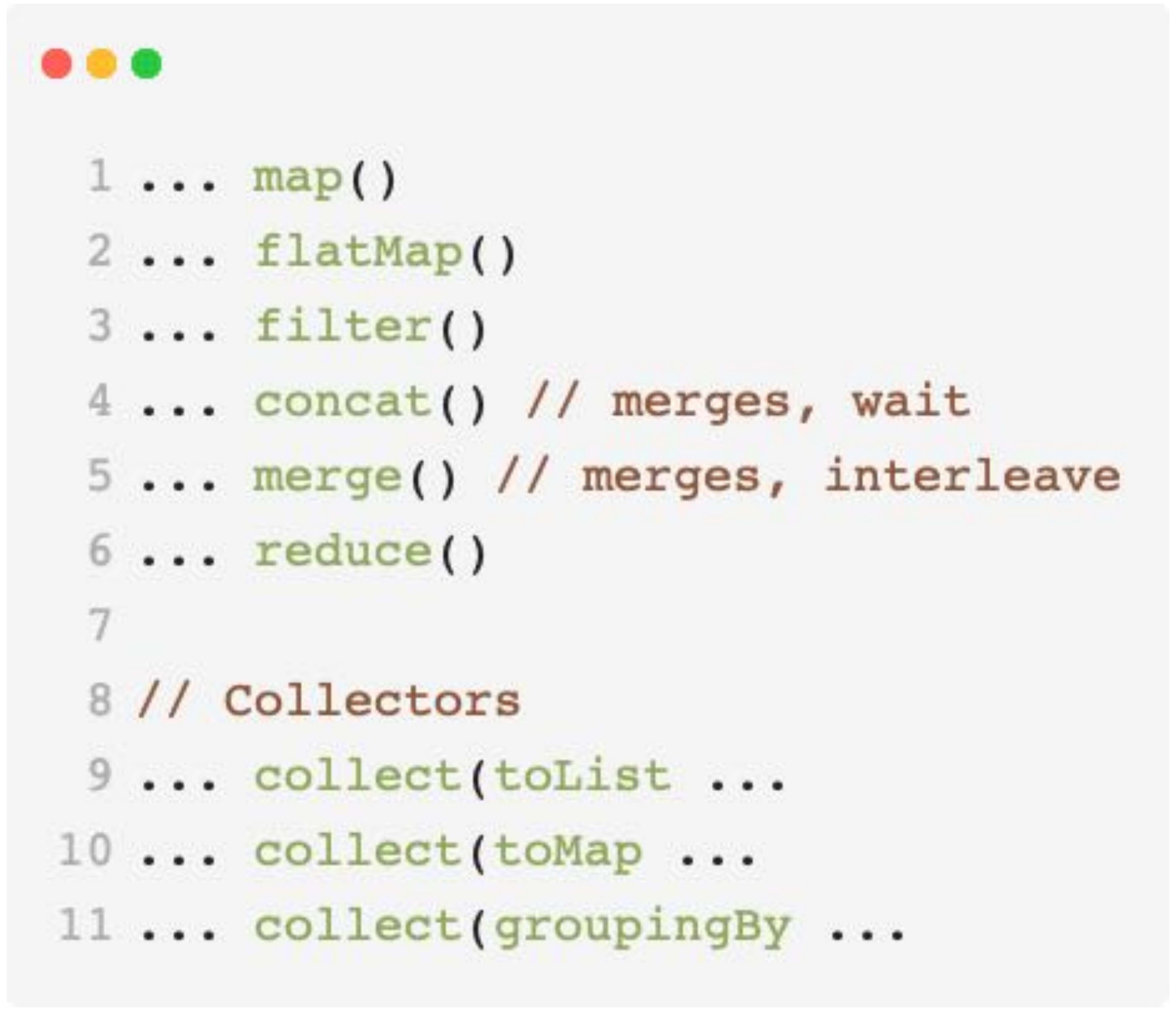


```
var elements = List.of(
  List.of(1, 2, 3, 4, 5),
  List.of(10, 20),
  List.of(30, 40, 50));

var result = elements.stream()
  .flatMap(List::stream)
  .reduce(0, (num1, num2) -> num1 + num2);

// 165
```



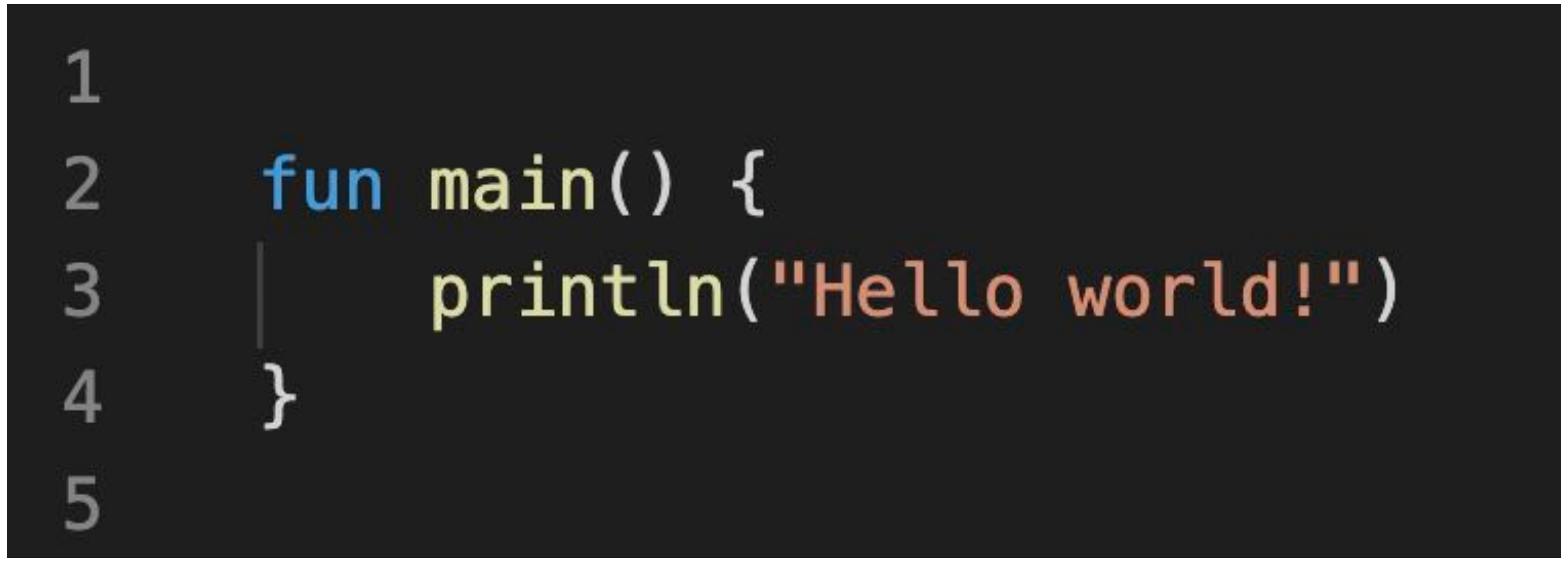
```
var elements = List.of(
  List.of(1, 2, 3, 4, 5),
  List.of(10, 20),
  List.of(30, 40, 50));

var result = elements.stream()
  .flatMap(List::stream)
  .filter(num -> num > 10)
  .reduce(0, (num1, num2) -> num1 + num2);

// 140
```

Kotlin
Exkurs zu Streams



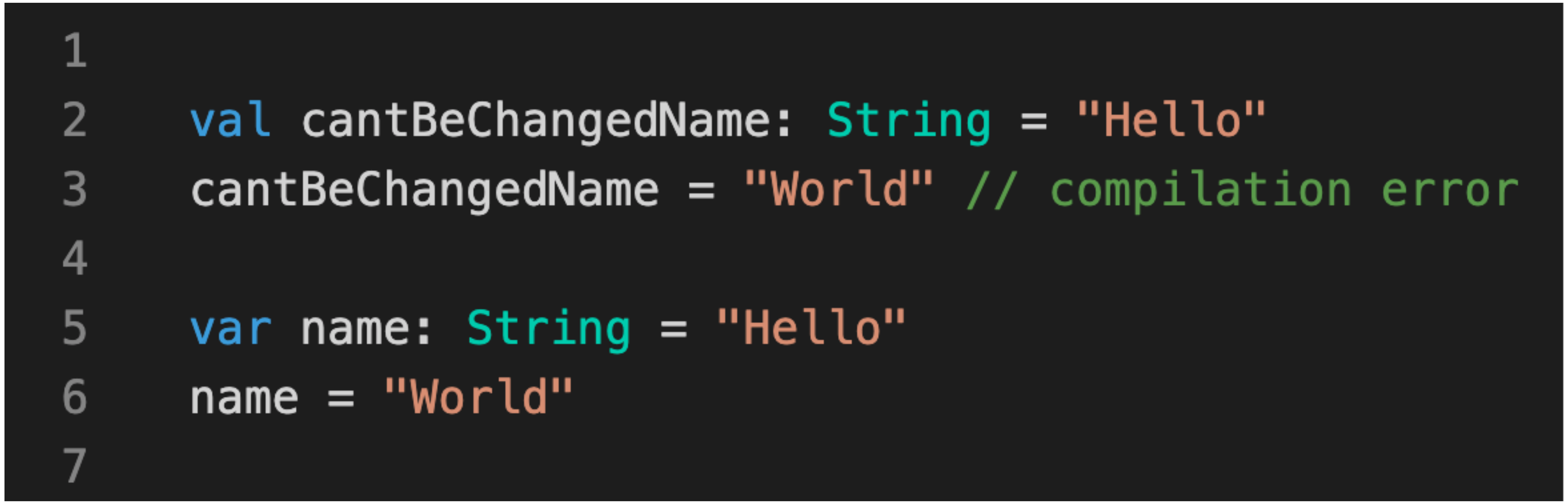
```
... map()
... flatMap()
... filter()
... concat() // merges, wait
... merge() // merges, interleave
... reduce()

// Collectors
... collect(toList ...
... collect(toMap ...
... collect(groupingBy ...
```

Kotlin



```kotlin

fun main() {
    println("Hello world!")
}

```

Kotlin



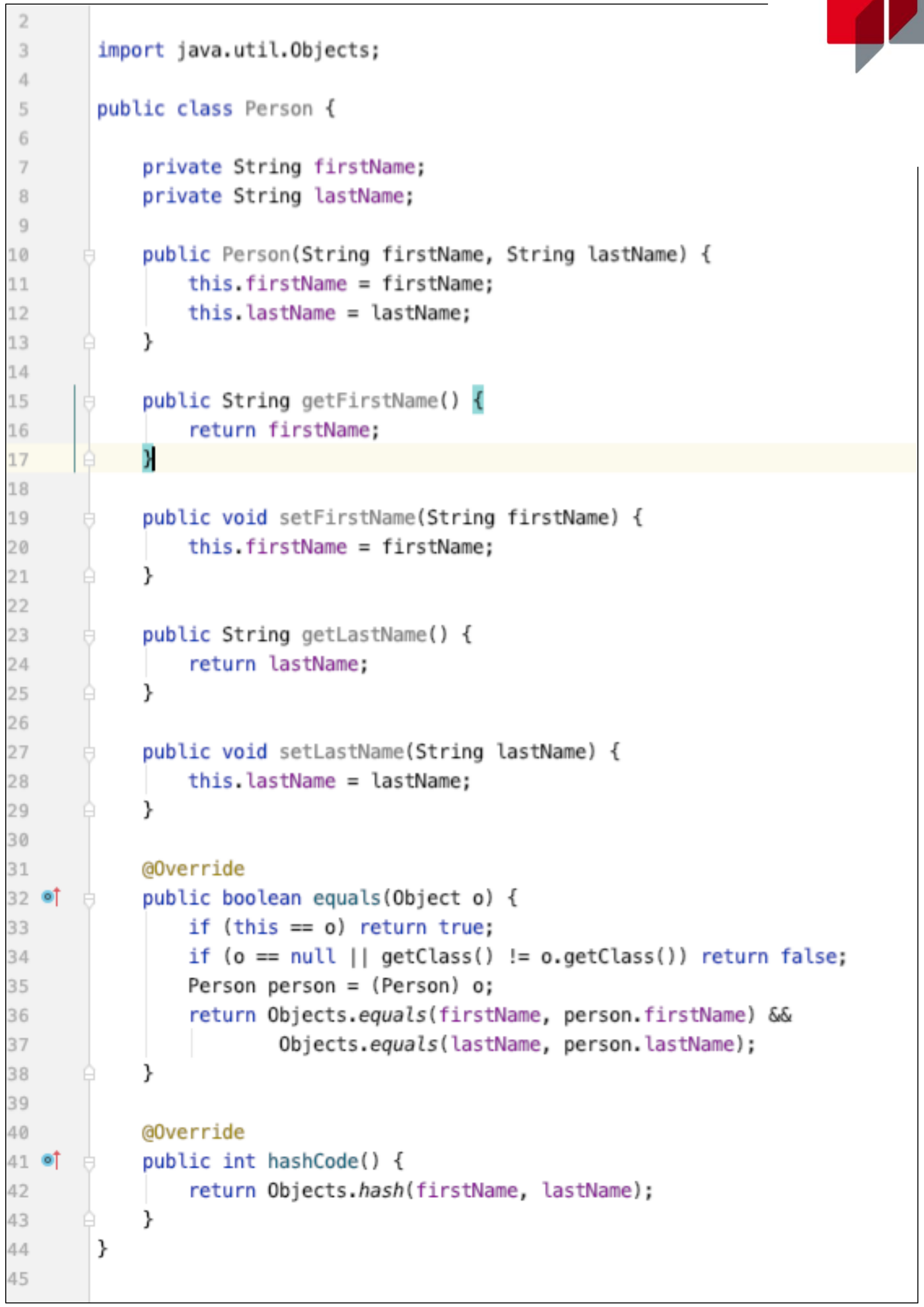
```kotlin

val cantBeChangedName: String = "Hello"
cantBeChangedName = "World" // compilation error

var name: String = "Hello"
name = "World"

```

Kotlin

```kotlin
data class Person(var firstName: String, var lastName: String)
```

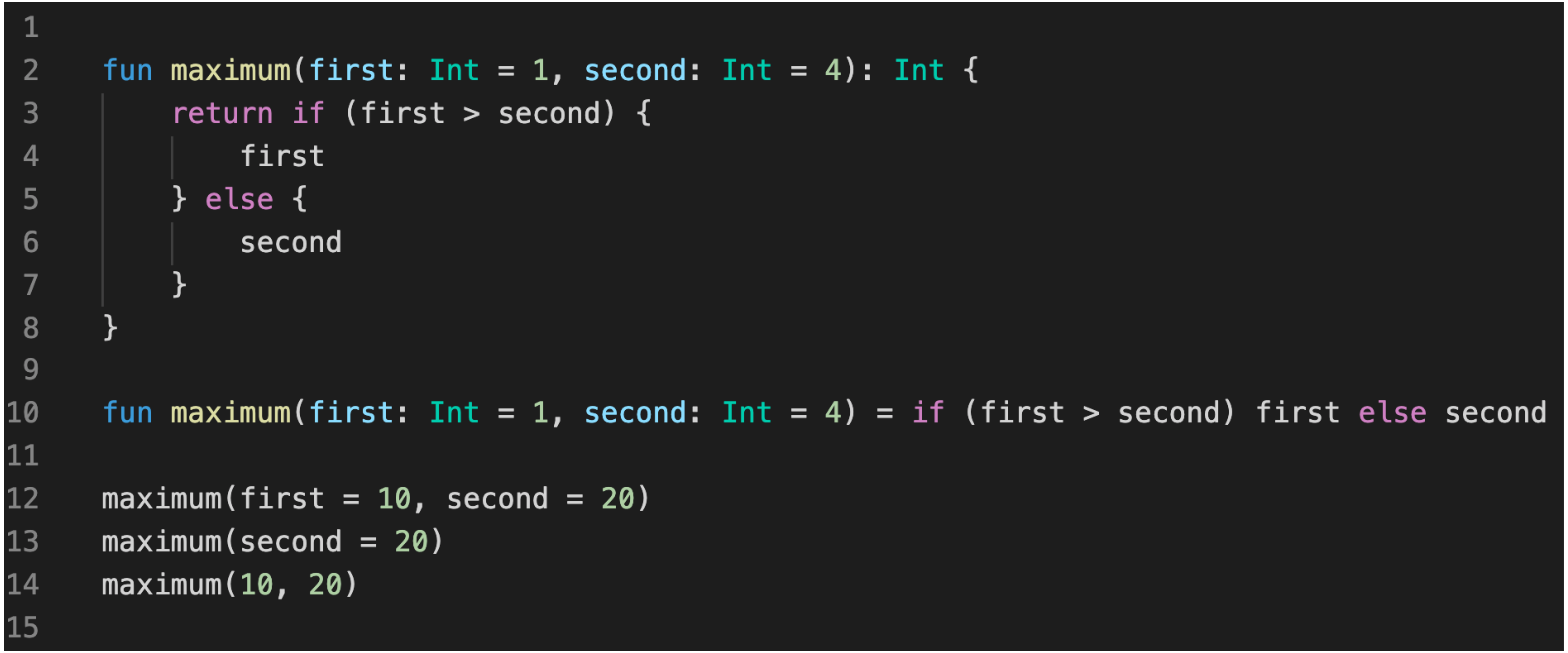
```kotlin
fun main() {

    val person = Person("Jan", "Brodhaecker")
    val otherPerson = Person("John", "Doe")
    val samePerson = Person("John", "Doe")

    println(person)
    println(person.equals(otherPerson))
    println(otherPerson.equals(samePerson))
}
```

```kotlin
data class Person(var firstName: String, var lastName: String) {

    override fun toString(): String {
        return "Hello $firstName, $lastName!"
    }

}
```



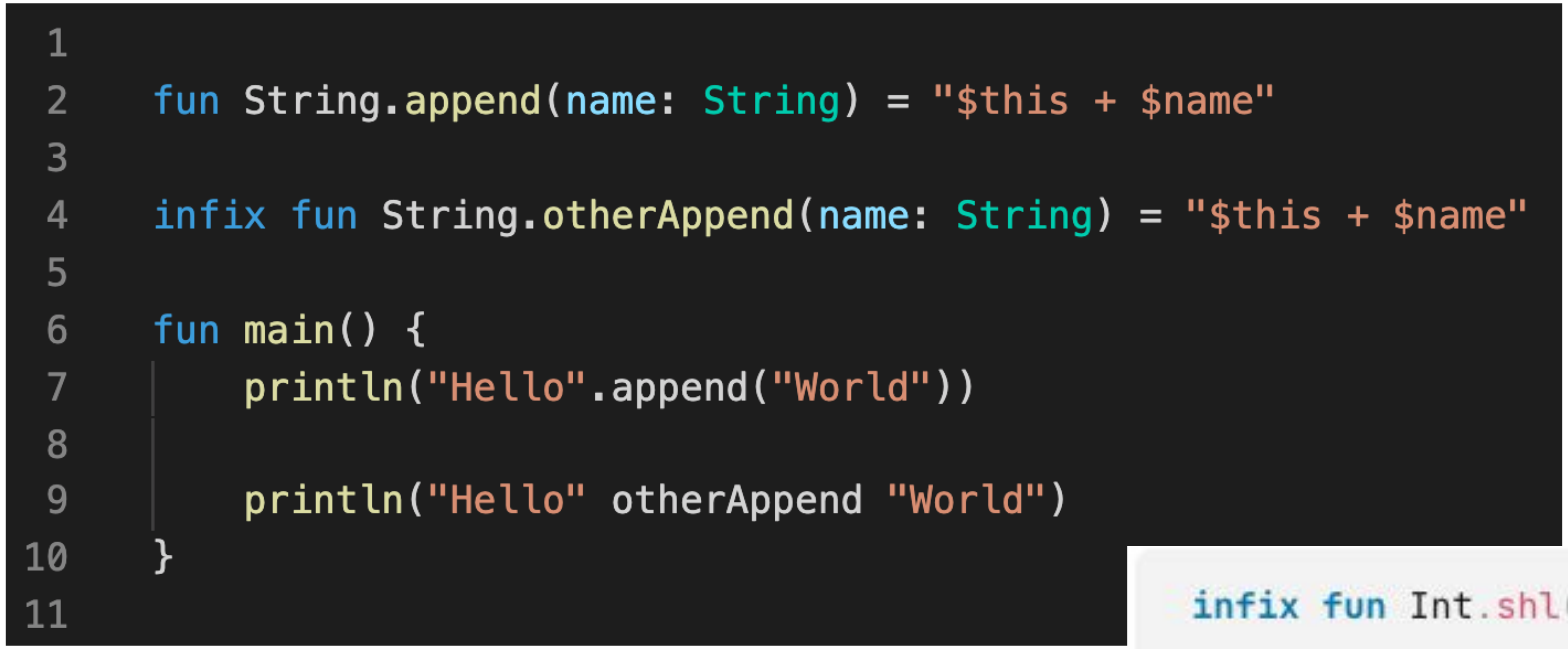
```java

import java.util.Objects;

public class Person {

    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Person person = (Person) o;
        return Objects.equals(firstName, person.firstName) &&
                Objects.equals(lastName, person.lastName);
    }

    @Override
    public int hashCode() {
        return Objects.hash(firstName, lastName);
    }
}

```

Kotlin



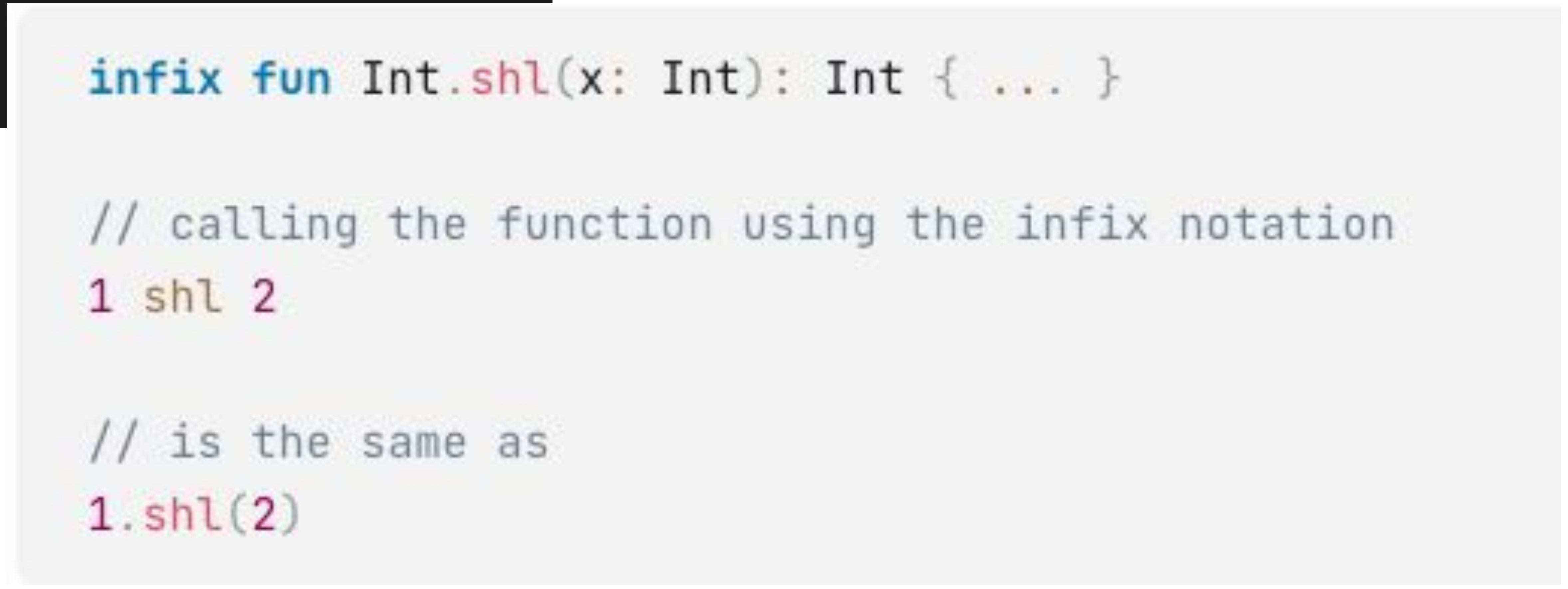
```kotlin

fun maximum(first: Int = 1, second: Int = 4): Int {
    return if (first > second) {
        first
    } else {
        second
    }
}

fun maximum(first: Int = 1, second: Int = 4) = if (first > second) first else second

maximum(first = 10, second = 20)
maximum(second = 20)
maximum(10, 20)

```

Kotlin



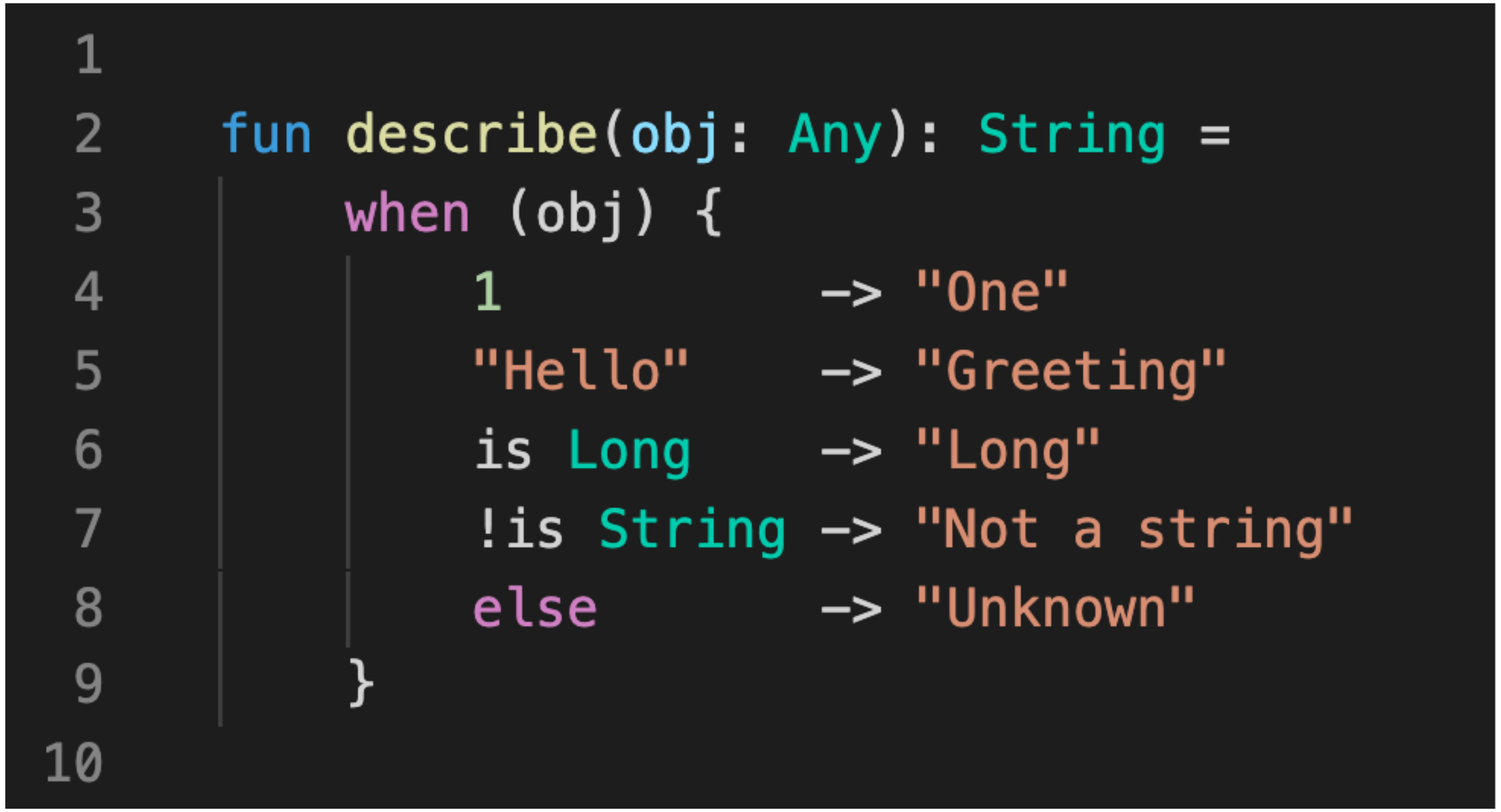
```kotlin

fun String.append(name: String) = "$this + $name"

infix fun String.otherAppend(name: String) = "$this + $name"

fun main() {
    println("Hello".append("World"))

    println("Hello" otherAppend "World")
}

```

https://kotlinlang.org/docs/functions.html#infix-notation



```kotlin
infix fun Int.shl(x: Int): Int { ... }

// calling the function using the infix notation
1 shl 2

// is the same as
1.shl(2)
```

Kotlin



```kotlin

fun describe(obj: Any): String =
    when (obj) {
        1          -> "One"
        "Hello"    -> "Greeting"
        is Long    -> "Long"
        !is String -> "Not a string"
        else       -> "Unknown"
    }

```

Kotlin

```kotlin
var b: String? = "abc" // can be set to null
b = null // ok
print(b)
```

Safe calls

```kotlin
val a = "Kotlin"
val b: String? = null
println(b?.length)
println(a?.length) // Unnecessary safe call
```

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

```kotlin
bob?.department?.head?.name
```

```kotlin
// If either `person` or `person.department` is null, the function is not
person?.department?.head = managersPool.getManager()
```

Safe casts

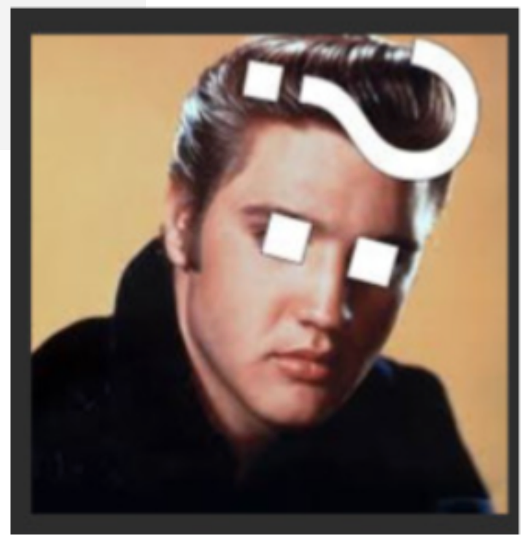
```kotlin
val aInt: Int? = a as? Int
```

Elvis Operator https://speakerdeck.com/alexgherschon/introduction-to-kotlin?slide=35

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

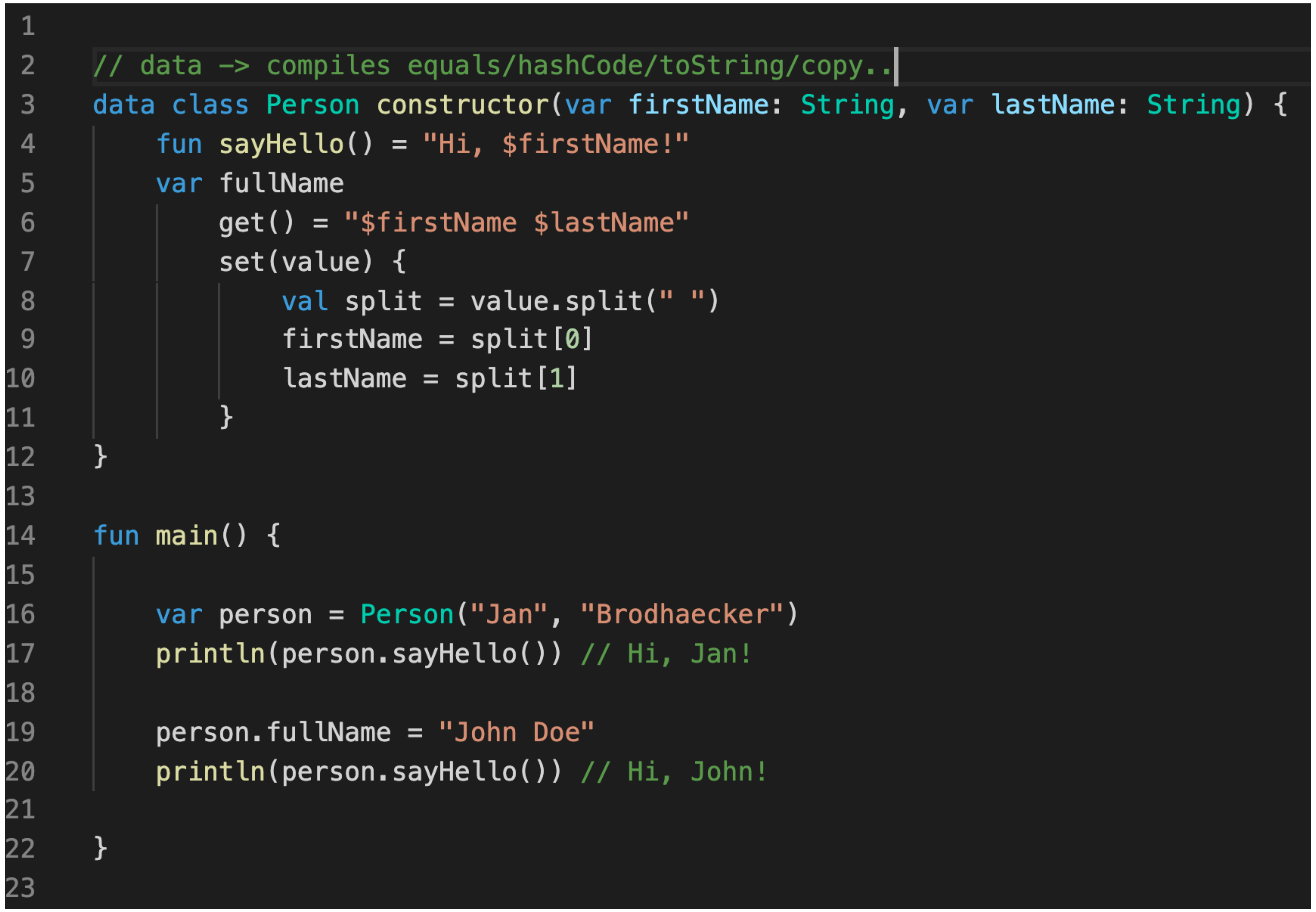
$\searrow$

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



https://kotlinlang.org/docs/null-safety.html#nullable-types-and-non-null-types

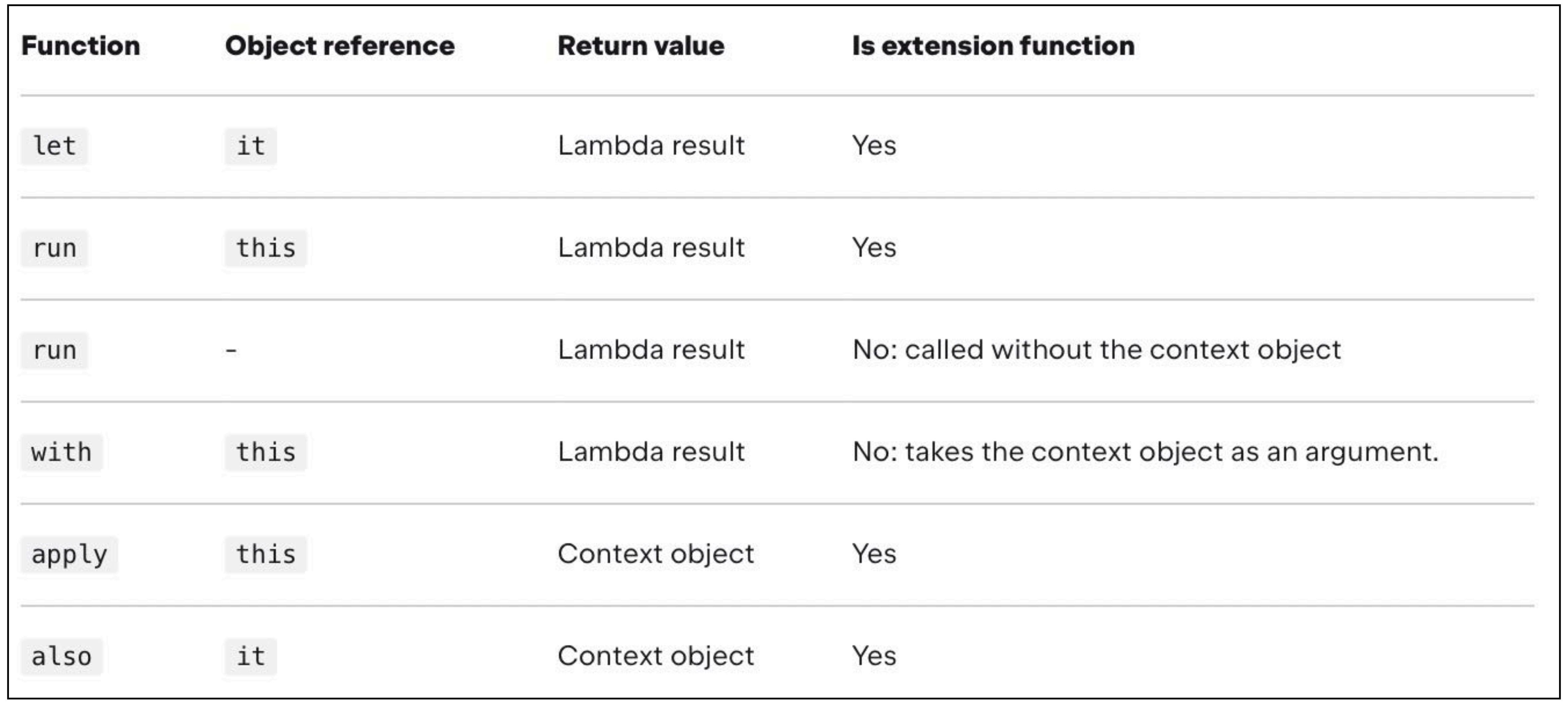
Kotlin



```kotlin

// data -> compiles equals/hashCode/toString/copy..
data class Person constructor(var firstName: String, var lastName: String) {
    fun sayHello() = "Hi, $firstName!"
    var fullName
        get() = "$firstName $lastName"
        set(value) {
            val split = value.split(" ")
            firstName = split[0]
            lastName = split[1]
        }
}

fun main() {

    var person = Person("Jan", "Brodhaecker")
    println(person.sayHello()) // Hi, Jan!

    person.fullName = "John Doe"
    println(person.sayHello()) // Hi, John!

}

```

Kotlin
Scope Functions

- oft austauschbar, Empfehlungen bzw. nur meist genutzte Verwendung
  - let
  - with
  - run
  - apply
  - also



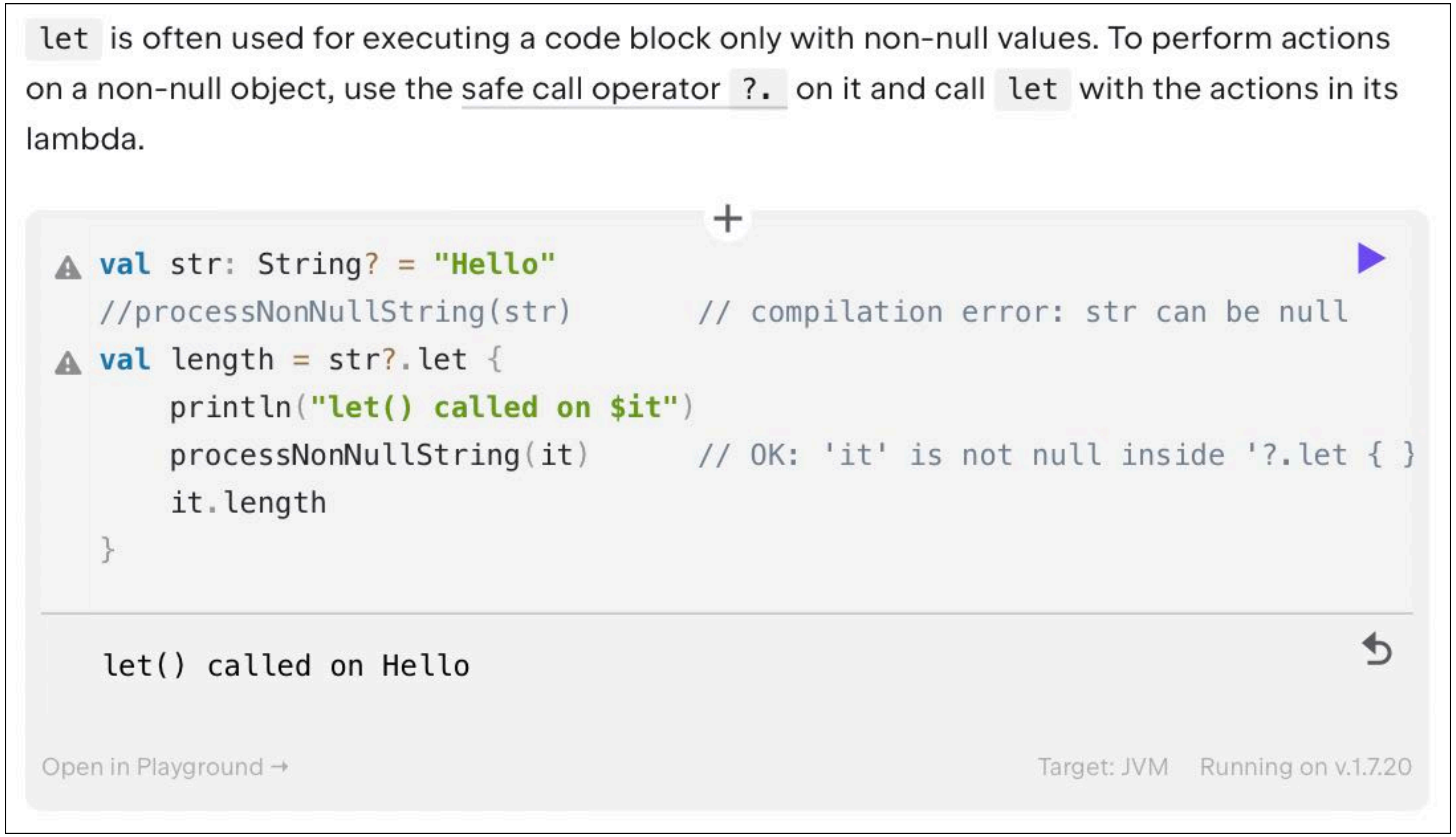
Function	Object reference	Return value	Is extension function
`let`	`it`	Lambda result	Yes
`run`	`this`	Lambda result	Yes
`run`	-	Lambda result	No: called without the context object
`with`	`this`	Lambda result	No: takes the context object as an argument.
`apply`	`this`	Context object	Yes
`also`	`it`	Context object	Yes

https://kotlinlang.org/docs/scope-functions.html#functions

Kotlin

Scope Functions - let

https://kotlinlang.org/docs/scope-functions.html#let



`let` is often used for executing a code block only with non-null values. To perform actions on a non-null object, use the safe call operator `?.` on it and call `let` with the actions in its lambda.

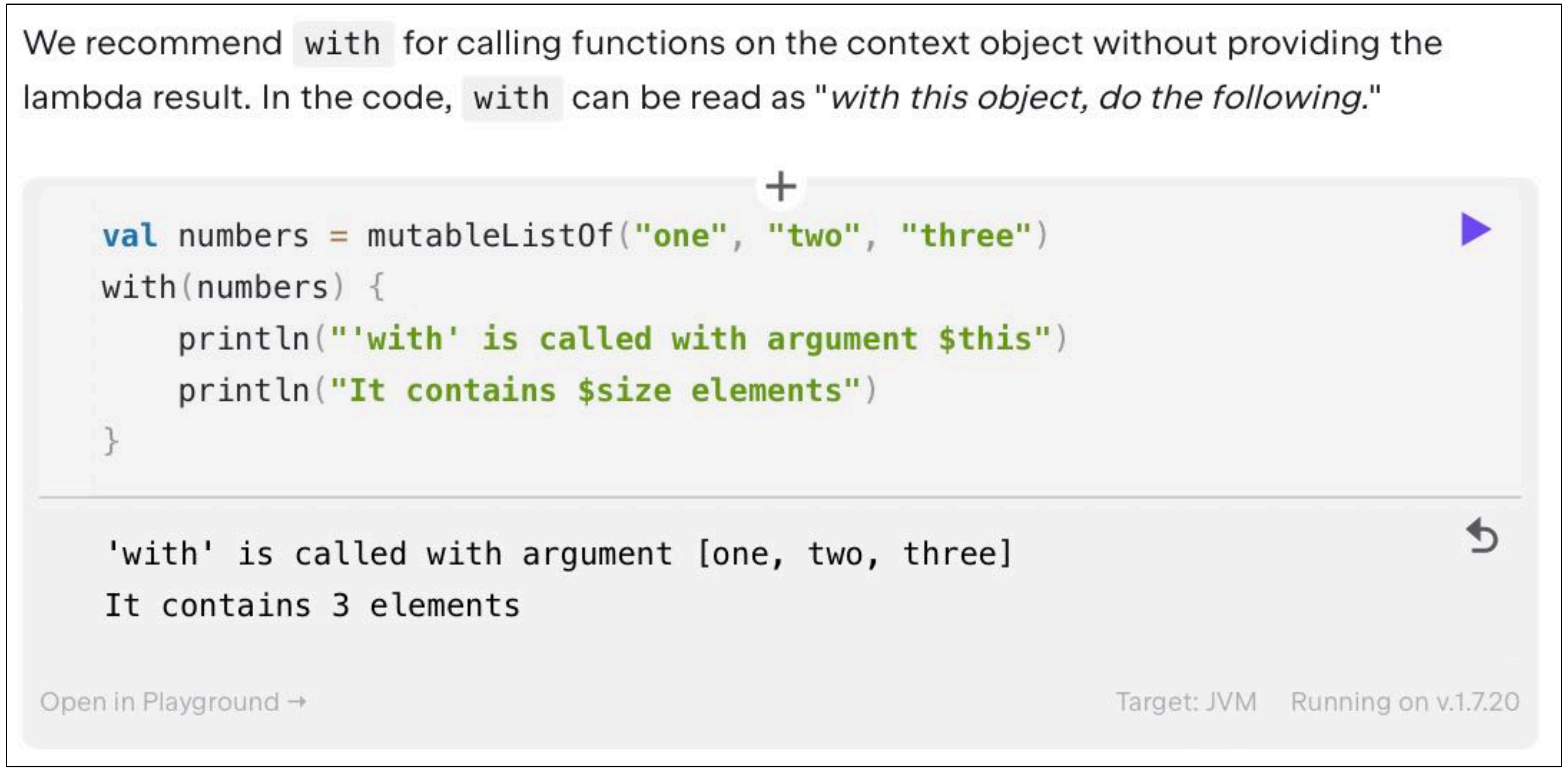
```kotlin
val str: String? = "Hello"
//processNonNullString(str)       // compilation error: str can be null
val length = str?.let {
    println("let() called on $it")
    processNonNullString(it)      // OK: 'it' is not null inside '?.let { }'
    it.length
}
```

```
let() called on Hello
```

Kotlin

Scope Functions - with

https://kotlinlang.org/docs/scope-functions.html#with



We recommend `with` for calling functions on the context object without providing the lambda result. In the code, `with` can be read as "*with this object, do the following.*"

```kotlin
val numbers = mutableListOf("one", "two", "three")
with(numbers) {
    println("'with' is called with argument $this")
    println("It contains $size elements")
}
```

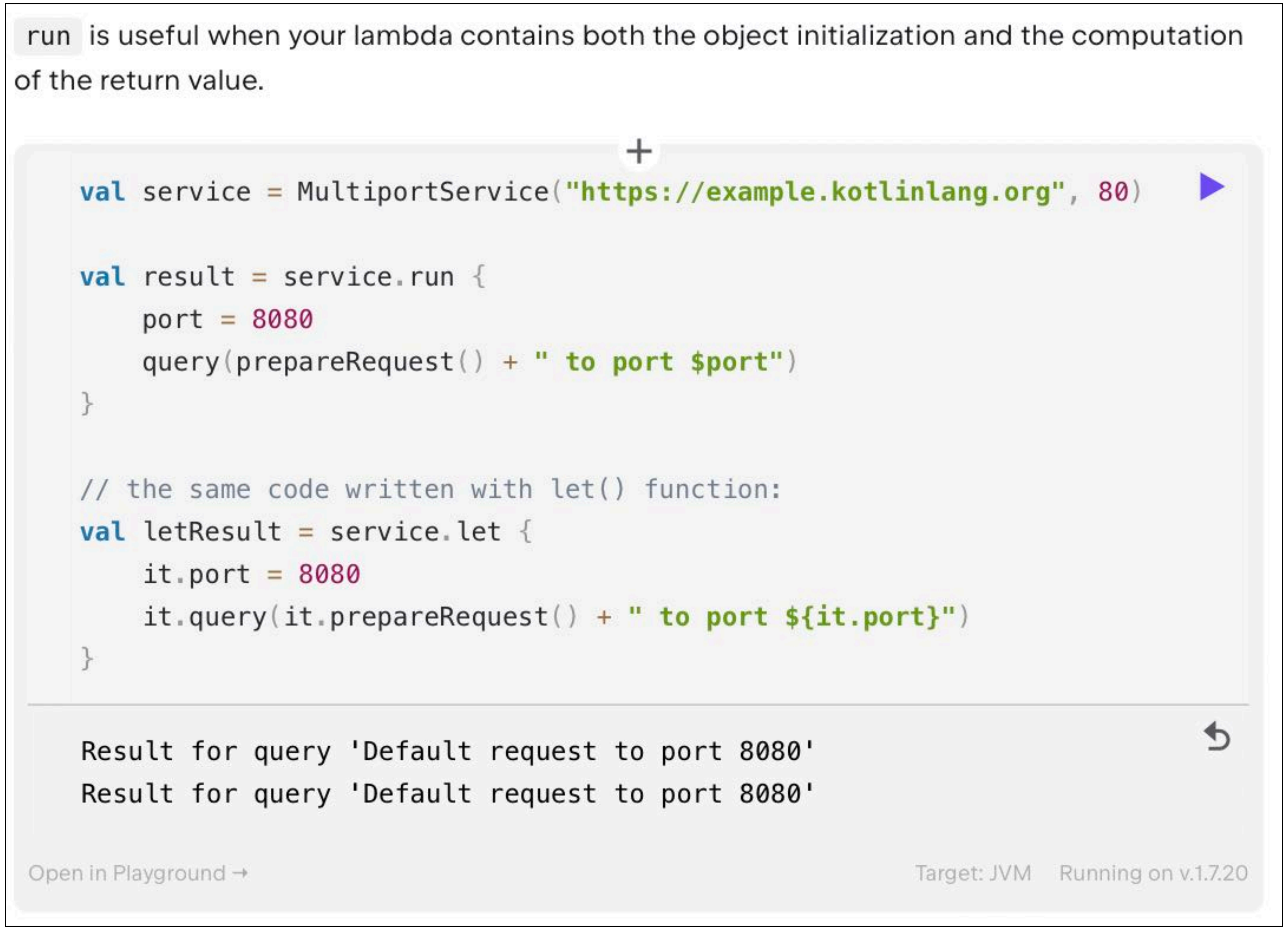
```
'with' is called with argument [one, two, three]
It contains 3 elements
```

Open in Playground → Target: JVM Running on v.1.7.20

Kotlin

Scope Functions - run

https://kotlinlang.org/docs/scope-functions.html#run



run is useful when your lambda contains both the object initialization and the computation of the return value.

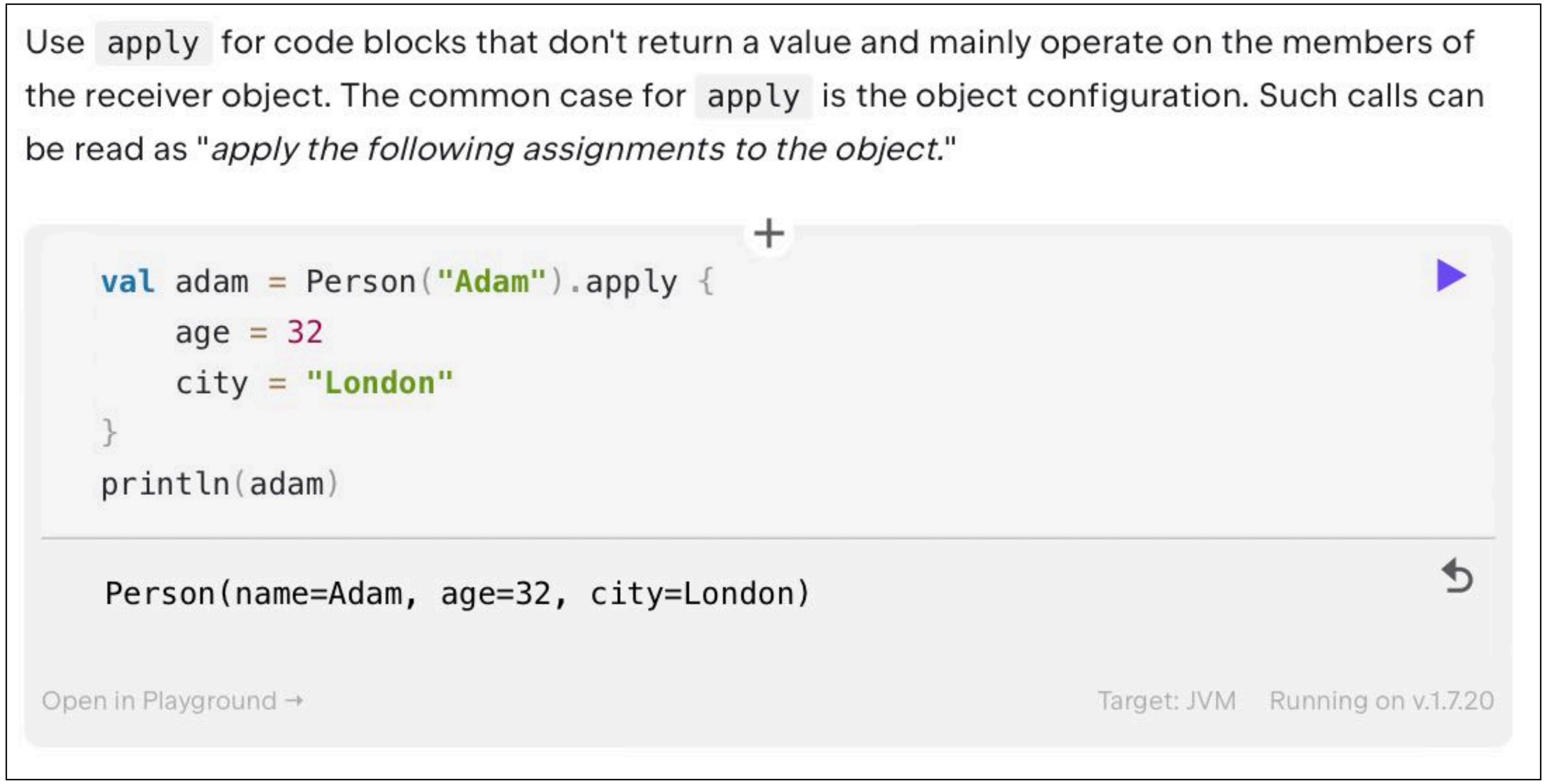
```
val service = MultiportService("https://example.kotlinlang.org", 80)

val result = service.run {
    port = 8080
    query(prepareRequest() + " to port $port")
}

// the same code written with let() function:
val letResult = service.let {
    it.port = 8080
    it.query(it.prepareRequest() + " to port ${it.port}")
}
```

```
Result for query 'Default request to port 8080'
Result for query 'Default request to port 8080'
```

Kotlin

Scope Functions - apply

https://kotlinlang.org/docs/scope-functions.html#apply



Use `apply` for code blocks that don't return a value and mainly operate on the members of the receiver object. The common case for `apply` is the object configuration. Such calls can be read as "*apply the following assignments to the object.*"

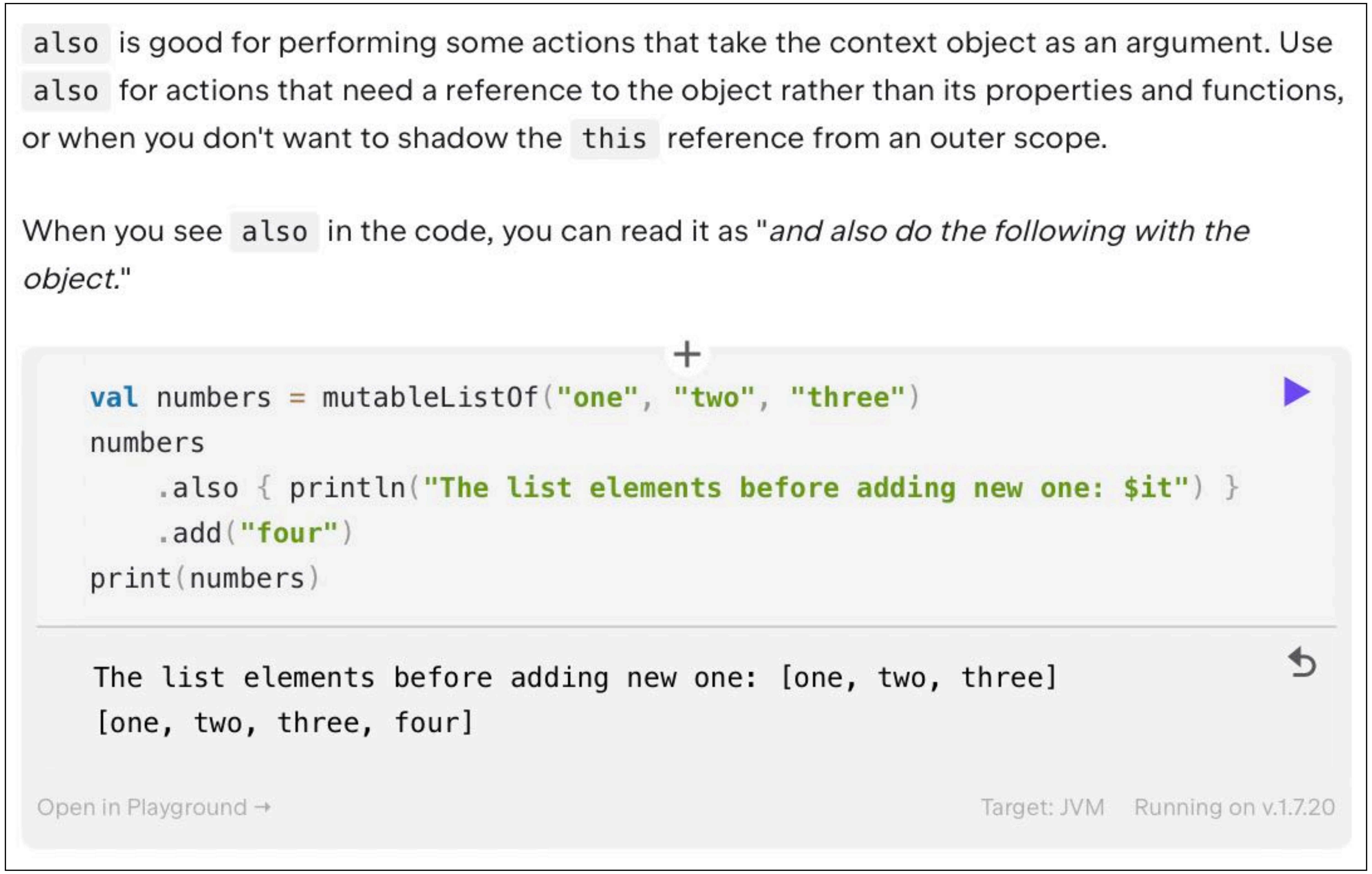
```
val adam = Person("Adam").apply {
    age = 32
    city = "London"
}
println(adam)
```

```
Person(name=Adam, age=32, city=London)
```

Kotlin

Scope Functions - also

https://kotlinlang.org/docs/scope-functions.html#also



`also` is good for performing some actions that take the context object as an argument. Use `also` for actions that need a reference to the object rather than its properties and functions, or when you don't want to shadow the `this` reference from an outer scope.

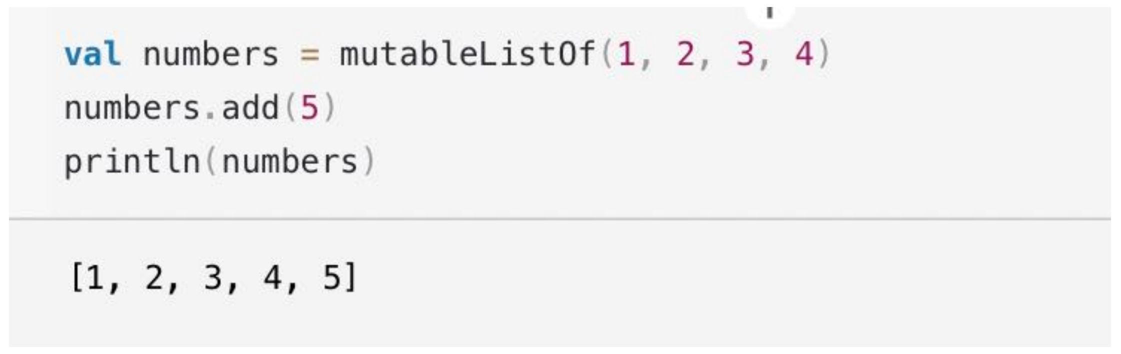
When you see `also` in the code, you can read it as "*and also do the following with the object.*"

```kotlin
val numbers = mutableListOf("one", "two", "three")
numbers
    .also { println("The list elements before adding new one: $it") }
    .add("four")
print(numbers)
```

```
The list elements before adding new one: [one, two, three]
[one, two, three, four]
```

Kotlin

- Collections
  - Lists, Sets, Map (... oder Dictionary)
  - können immutable (read-only) oder mutable sein

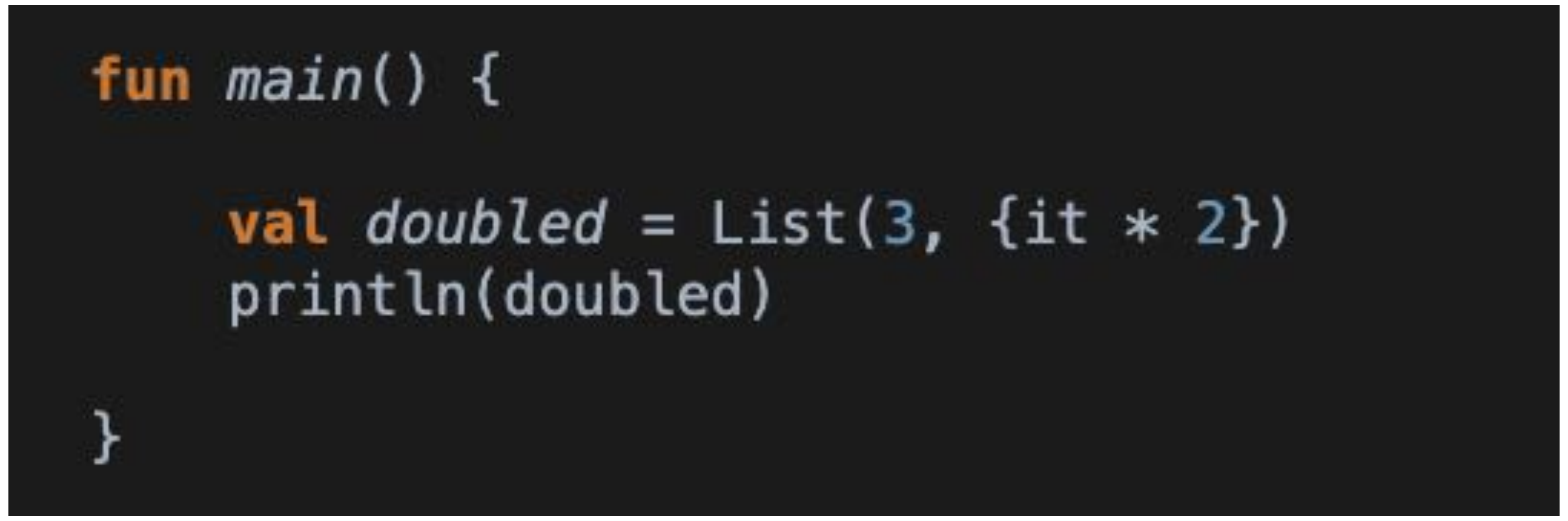


```
val numbers = mutableListOf(1, 2, 3, 4)
numbers.add(5)
println(numbers)
```

```
[1, 2, 3, 4, 5]
```

  - Erzeugen von Collections
    - mit Hilfe von Elementen
      - [set | list | map]Of<>()
      - mutable[Set | List | Map]Of<>()
      - empty[Set | List | Map]<>()
    - copy

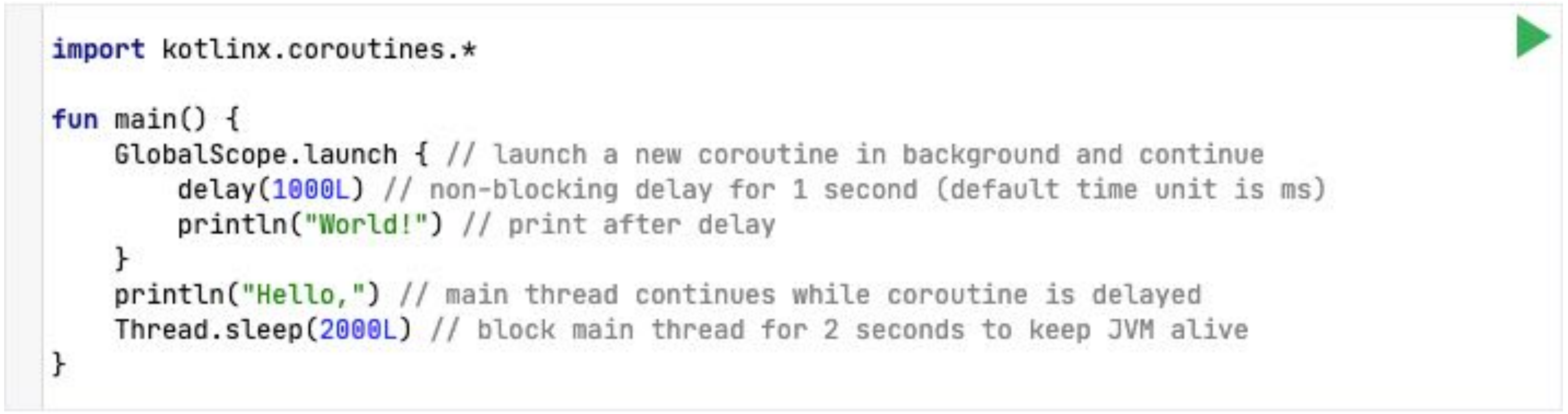
Initialiser Funktionen



```
fun main() {

    val doubled = List(3, {it * 2})
    println(doubled)

}
```

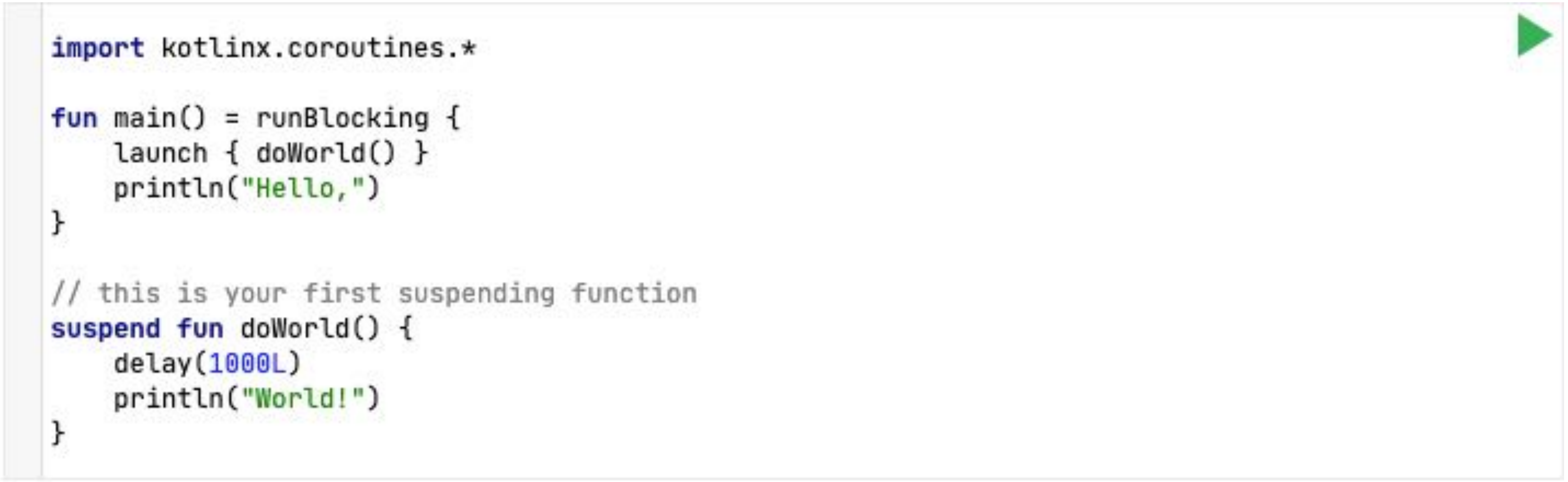
Kotlin

- Coroutines
  - leichtgewichtige Threads
  - ermöglichen Parallelität



```
import kotlinx.coroutines.*

fun main() {
    GlobalScope.launch { // launch a new coroutine in background and continue
        delay(1000L) // non-blocking delay for 1 second (default time unit is ms)
        println("World!") // print after delay
    }
    println("Hello,") // main thread continues while coroutine is delayed
    Thread.sleep(2000L) // block main thread for 2 seconds to keep JVM alive
}
```

Kotlin

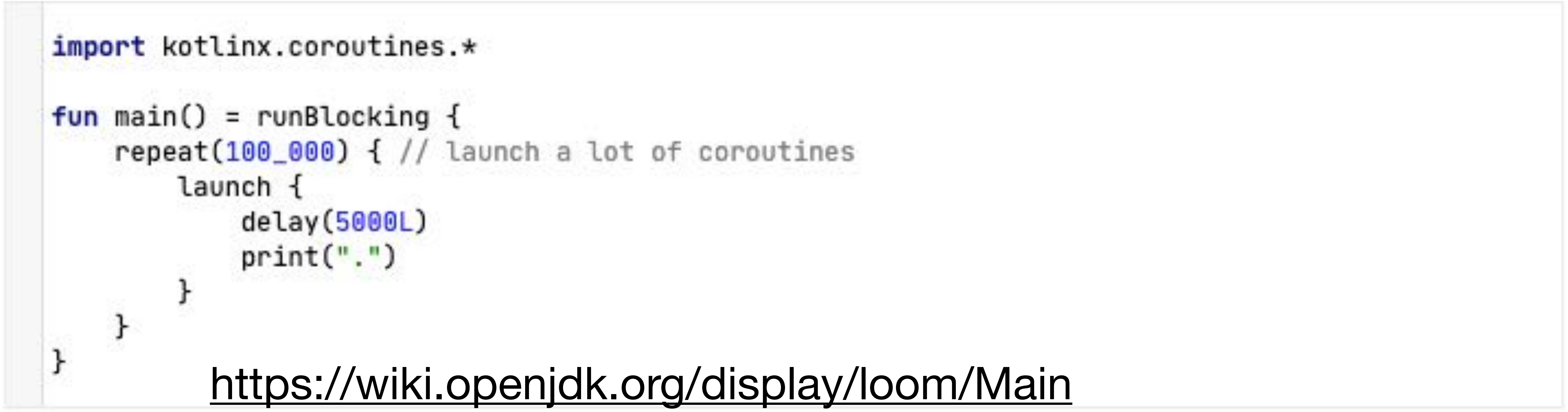


```
import kotlinx.coroutines.*

fun main() = runBlocking {
    launch { doWorld() }
    println("Hello,")
}

// this is your first suspending function
suspend fun doWorld() {
    delay(1000L)
    println("World!")
}
```

suspend -> Ausführung muss nicht zwingend blockierend sein

https://kotlinlang.org/docs/flow.html#suspending-functions

Kotlin



```
import kotlinx.coroutines.*

fun main() = runBlocking {
    repeat(100_000) { // launch a lot of coroutines
        launch {
            delay(5000L)
            print(".")
        }
    }
}
```

https://wiki.openjdk.org/display/loom/Main

Perfomance: ähnlicher Java Code würde vermutlich eine Out-Of-Memory Exception werfen!

(Ansatz für Java: Project Loom)
https://wiki.openjdk.org/display/loom/Main

weitere Informationen:

https://kotlinlang.org/docs/reference/coroutines/basics.html