PRAKTIKUM PEMROGRAMAN MOBILE

Kotlin Dasar

Pertemuan Ke-2

Disusun Oleh :

NAMA	**: YOHANA DJAWA SEMAH**
NIM	**: 195410104**
PRODI	**: INFORMATIKA**

MODUL 2

Kotlin Dasar

A. TUJUAN

Mahasiswa mampu memahami dan mengembangkan aplikasi sederhana dengan bahasa pemrograman Kotlin.

B. DASAR TEORI

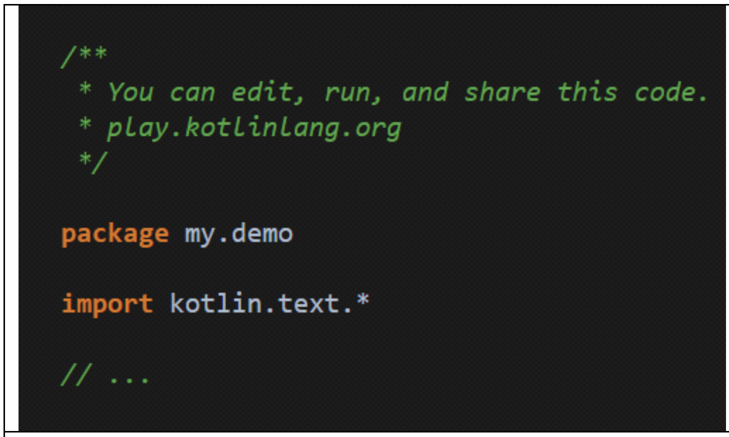
Penggunaan Kotlin untuk Pengembangan Android. Kotlin/Native memungkinkan developer untuk menggunakannya sebagai bahasa pemrograman dalam pengembangan aplikasi di platform lain seperti embedded system, desktop, macOS, dan iOS. Bahkan tak menutup kemungkinan Kotlin juga bisa digunakan untuk data science dan machine learning. Kotlin sangat cocok untuk mengembangkan aplikasi Android, membawa semua keunggulan bahasa modern ke platform Android tanpa memperkenalkan batasan baru:

- Compatibility. Kotlin sepenuhnya kompatibel dengan JDK 6. Ini memastikan bahwa aplikasi yang dibangun dengan Kotlin dapat berjalan pada perangkat Android yang lebih lama tanpa ada masalah. Android Studio pun mendukung penuh pengembangan dengan bahasa Kotlin.
- Performance. Dengan struktur bytecode yang sama dengan Java, aplikasi yang dibangun dengan Kotlin dapat berjalan setara dengan aplikasi yang dibangun dengan Java. Terdapat juga fitur seperti inline function pada Kotlin yang membuat 19 kode yang dituliskan dengan lambda bisa berjalan lebih cepat dibandingkan kode yang sama dan dituliskan dengan Java.
- Interoperability. Semua library Android yang tersedia, dapat digunakan pada Kotlin.
- Compilation Time. Kotlin mendukung kompilasi inkremental yang efisien. Oleh karena itu, proses build biasanya sama atau lebih cepat dibandingkan dengan Java.

C. PRAKTIK

Kita akan membuat program kotlin dengan dibandingkan dengan java. Gunakan laman web (https://try.kotlinlang.org) untuk mencoba menjalankan program kotlin.

Dikutip dari https://kotlinlang.org/docs/reference/basic-syntax.html Defining packages. Package specification should be at the top of the source file:



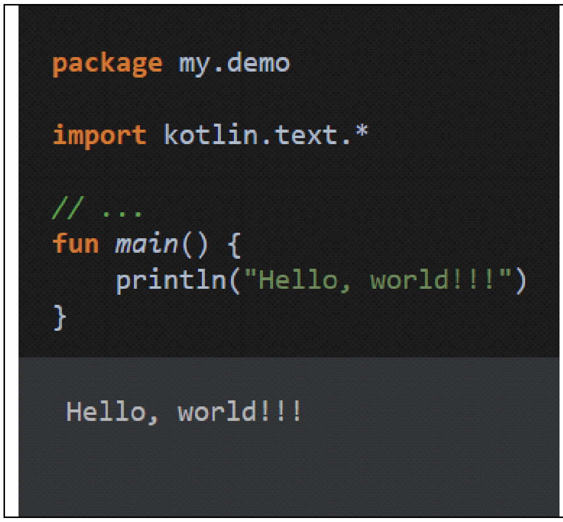
```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

package my.demo

import kotlin.text.*

// ...
```

Disaat di run maka tampiannya no project karena belum ada project yang mau dibuat

It is not required to match directories and packages: source files can be placed arbitrarily in the file system.

Defining functions

Program

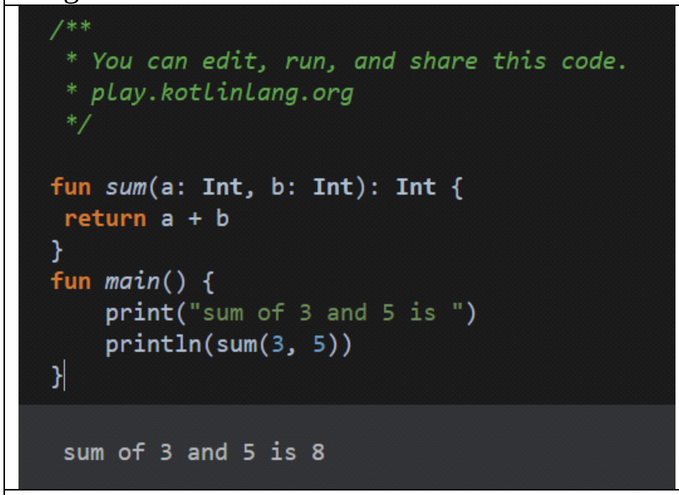


```
package my.demo

import kotlin.text.*

// ...
fun main() {
    println("Hello, world!!!")
}
```

```
Hello, world!!!
```

Function with an expression body and inferred return type:

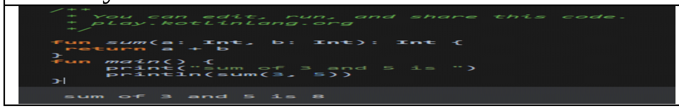
Program



```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun sum(a: Int, b: Int): Int {
 return a + b
}
fun main() {
    print("sum of 3 and 5 is ")
    println(sum(3, 5))
}
```

```
sum of 3 and 5 is 8
```

Hasilnya

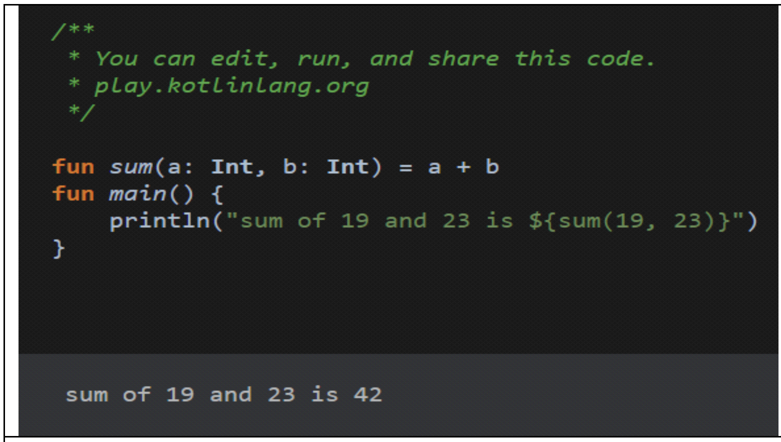


```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun sum(a: Int, b: Int): Int {
 return a + b
}
fun main() {
    print("sum of 3 and 5 is ")
    println(sum(3, 5))
}
```

```
sum of 3 and 5 is 8
```

Function with an expression body and inferred return type:

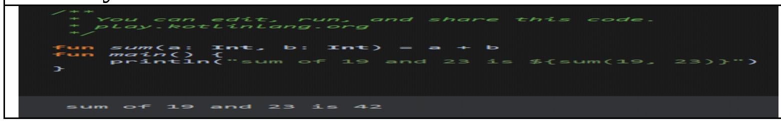
Program



```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun sum(a: Int, b: Int) = a + b
fun main() {
    println("sum of 19 and 23 is ${sum(19, 23)}")
}
```

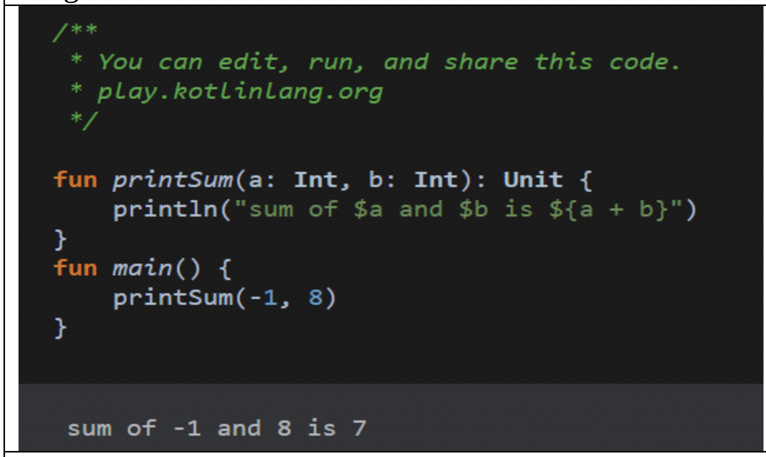
```
sum of 19 and 23 is 42
```

Hasilnya



```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun sum(a: Int, b: Int) = a + b
fun main() {
    println("sum of 19 and 23 is ${sum(19, 23)}")
}
sum of 19 and 23 is 42
```

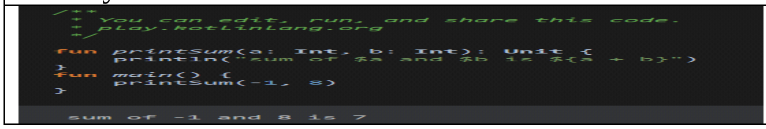
Program



```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun printSum(a: Int, b: Int): Unit {
    println("sum of $a and $b is ${a + b}")
}
fun main() {
    printSum(-1, 8)
}
```

```
sum of -1 and 8 is 7
```

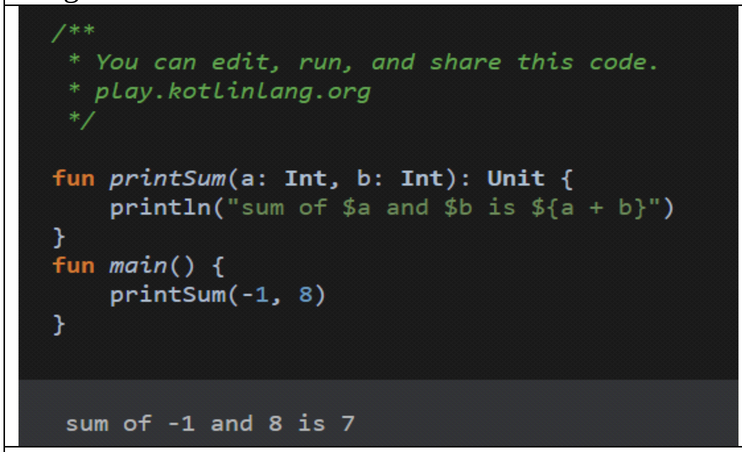
Hasilnya



```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun printSum(a: Int, b: Int): Unit {
    println("sum of $a and $b is ${a + b}")
}
fun main() {
    printSum(-1, 8)
}
sum of -1 and 8 is 7
```

Unit return type can be omitted:

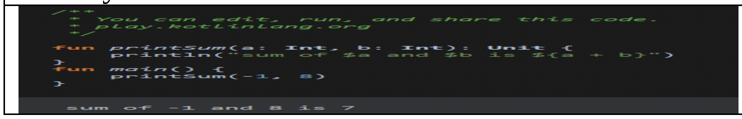
Program



```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun printSum(a: Int, b: Int): Unit {
    println("sum of $a and $b is ${a + b}")
}
fun main() {
    printSum(-1, 8)
}
```

```
sum of -1 and 8 is 7
```

Hasilnya

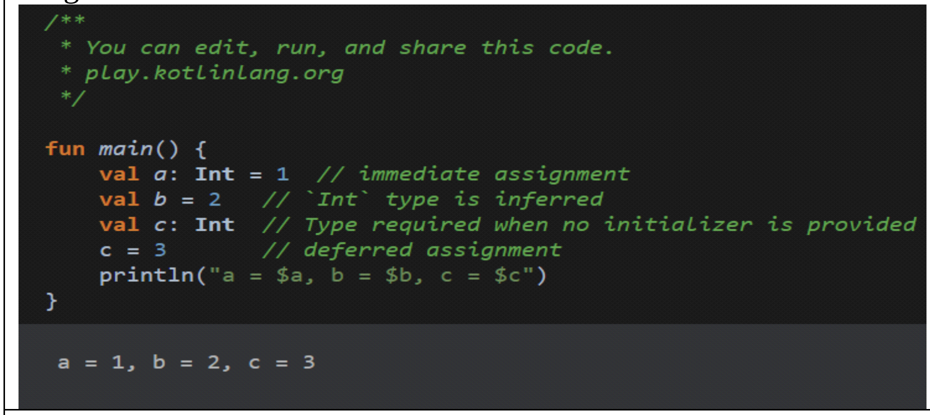


```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun printSum(a: Int, b: Int): Unit {
    println("sum of $a and $b is ${a + b}")
}
fun main() {
    printSum(-1, 8)
}
```

```
sum of -1 and 8 is 7
```

Defining variables Read-only local variables are defined using the keyword val. They can be assigned a value only once.

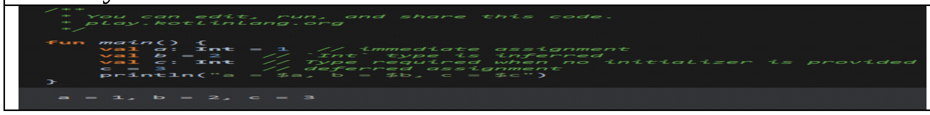
Program



```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun main() {
    val a: Int = 1  // immediate assignment
    val b = 2   // `Int` type is inferred
    val c: Int  // Type required when no initializer is provided
    c = 3       // deferred assignment
    println("a = $a, b = $b, c = $c")
}
```

```
a = 1, b = 2, c = 3
```

Hasilnya

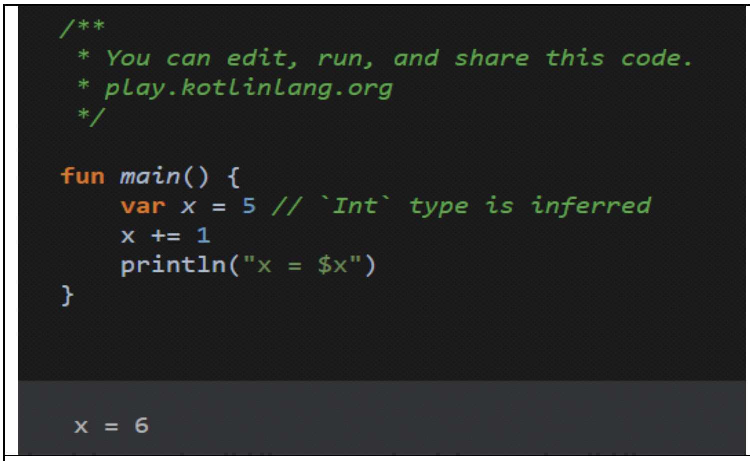


```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    val a: Int = 1  // immediate assignment
    val b = 2   // `Int` type is inferred
    val c: Int  // Type required when no initializer is provided
    c = 3       // deferred assignment
    println("a = $a, b = $b, c = $c")
}
```

```
a = 1, b = 2, c = 3
```

Variables that can be reassigned use the var keyword:

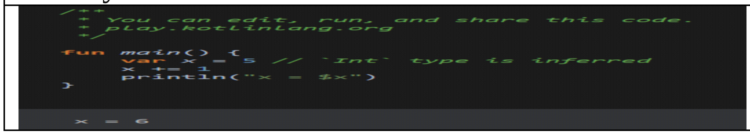
```
var x = 5 // `Int` type is inferred x += 1
```



```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun main() {
    var x = 5 // `Int` type is inferred
    x += 1
    println("x = $x")
}
```

```
x = 6
```

Hasilnya

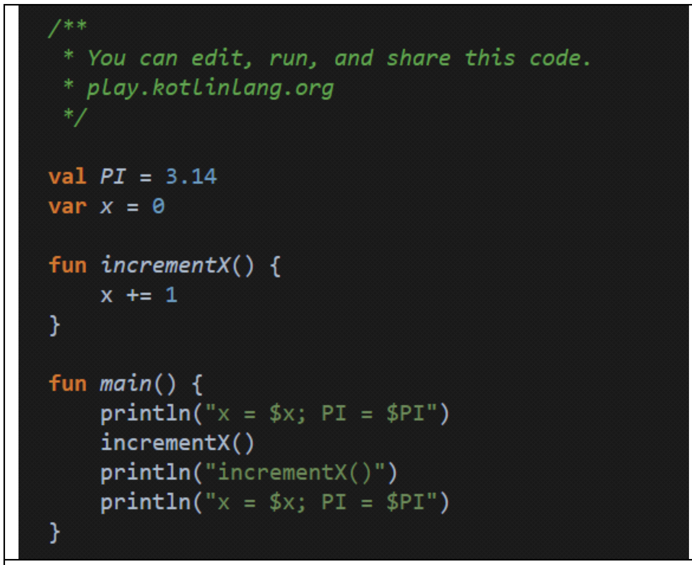


```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var x = 5 // `Int` type is inferred
    x += 1
    println("x = $x")
}
```

```
x = 6
```

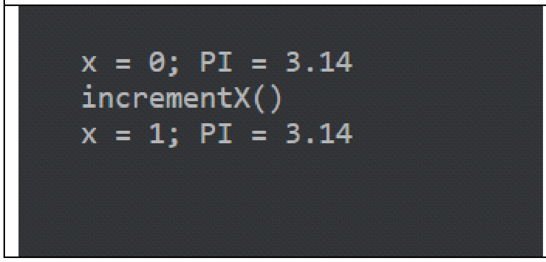
Top-level variables:

Programnya



```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

val PI = 3.14
var x = 0

fun incrementX() {
    x += 1
}

fun main() {
    println("x = $x; PI = $PI")
    incrementX()
    println("incrementX()")
    println("x = $x; PI = $PI")
}
```

Hasil setelah dirun



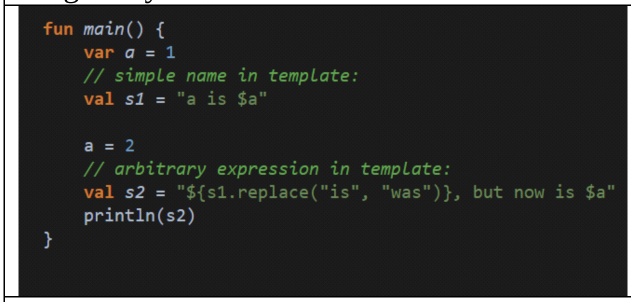
```
x = 0; PI = 3.14
incrementX()
x = 1; PI = 3.14
```

// This is an end-of-line comment /*
This is a block comment on multiple lines. */
Unlike Java, block comments in Kotlin can be nested.

See Documenting Kotlin Code for information on the documentation comment syntax

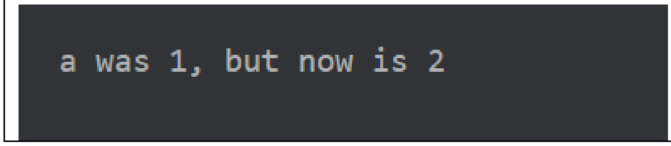
Using string templates

Programnya



```
fun main() {
    var a = 1
    // simple name in template:
    val s1 = "a is $a"

    a = 2
    // arbitrary expression in template:
    val s2 = "${s1.replace("is", "was")}, but now is $a"
    println(s2)
}
```

Hasil setelah dirun

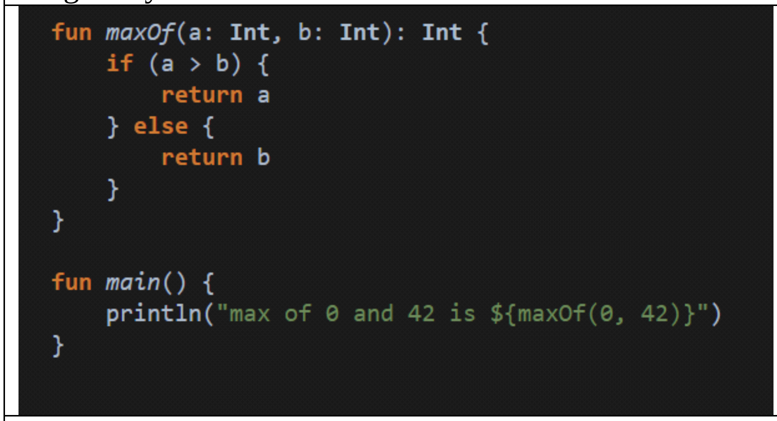


```
a was 1, but now is 2
```

Using conditional expressions

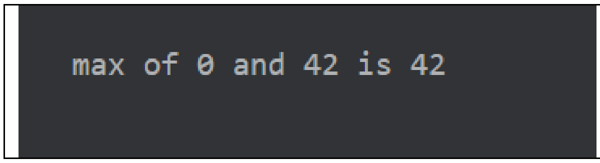
```
fun maxOf(a: Int, b: Int): Int {
 if (a > b) {
 return a
}
 else {
 return b
 }
}
```

Programnya



```
fun maxOf(a: Int, b: Int): Int {
    if (a > b) {
        return a
    } else {
        return b
    }
}

fun main() {
    println("max of 0 and 42 is ${maxOf(0, 42)}")
}
```

Hasil setelah dirun

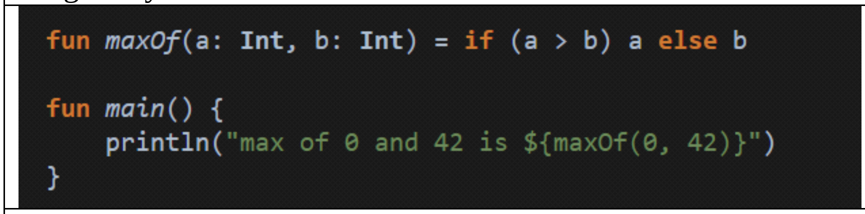


```
max of 0 and 42 is 42
```

Using if as an expression:

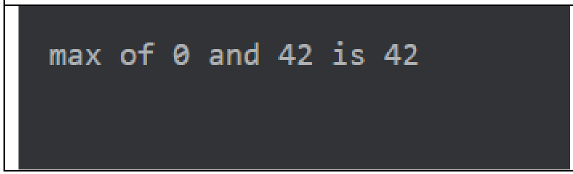
```
fun maxOf(a: Int, b: Int) = if (a > b) a else b
```

Programnya



```
fun maxOf(a: Int, b: Int) = if (a > b) a else b

fun main() {
    println("max of 0 and 42 is ${maxOf(0, 42)}")
}
```

Hasil setelah dirun



```
max of 0 and 42 is 42
```

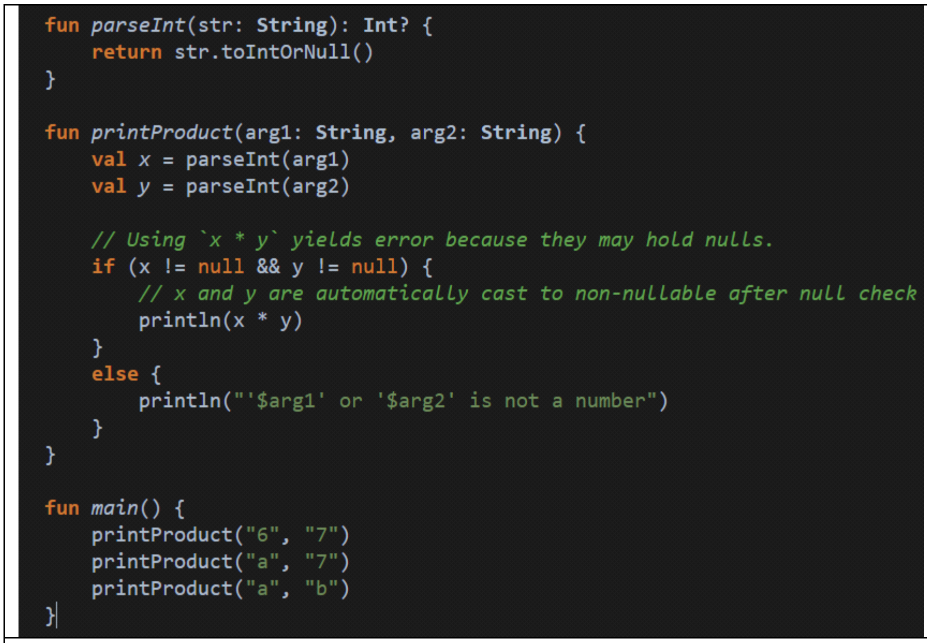
Using nullable values and checking for null

A reference must be explicitly marked as nullable when null value is possible. Return null if str does not hold an integer:

```
fun parseInt(str: String): Int? {
 // ...
}
```

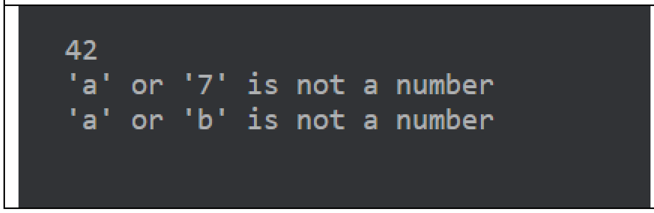
Use a function returning nullable value:

Programnya



```kotlin
fun parseInt(str: String): Int? {
    return str.toIntOrNull()
}

fun printProduct(arg1: String, arg2: String) {
    val x = parseInt(arg1)
    val y = parseInt(arg2)

    // Using `x * y` yields error because they may hold nulls.
    if (x != null && y != null) {
        // x and y are automatically cast to non-nullable after null check
        println(x * y)
    }
    else {
        println("'$arg1' or '$arg2' is not a number")
    }
}

fun main() {
    printProduct("6", "7")
    printProduct("a", "7")
    printProduct("a", "b")
}
```

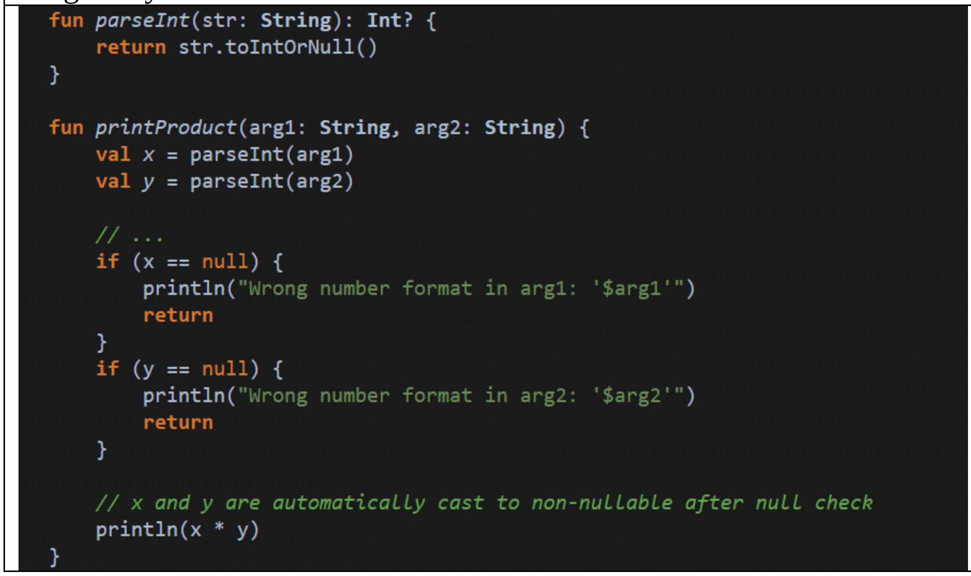
Hasil setelah dirun



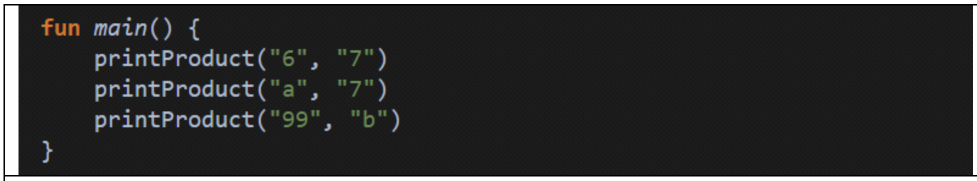
```
42
'a' or '7' is not a number
'a' or 'b' is not a number
```

Or

Programnya

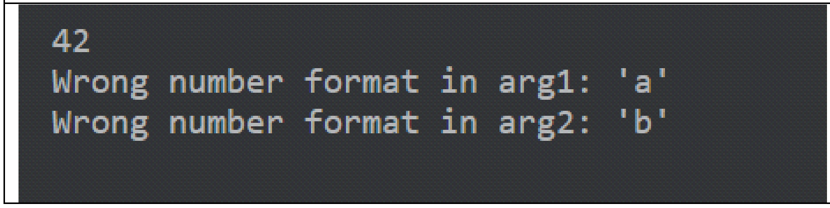


```kotlin
fun parseInt(str: String): Int? {
    return str.toIntOrNull()
}

fun printProduct(arg1: String, arg2: String) {
    val x = parseInt(arg1)
    val y = parseInt(arg2)

    // ...
    if (x == null) {
        println("Wrong number format in arg1: '$arg1'")
        return
    }
    if (y == null) {
        println("Wrong number format in arg2: '$arg2'")
        return
    }

    // x and y are automatically cast to non-nullable after null check
    println(x * y)
}
```



```
fun main() {
    printProduct("6", "7")
    printProduct("a", "7")
    printProduct("99", "b")
}
```

Hasil setelah dirun

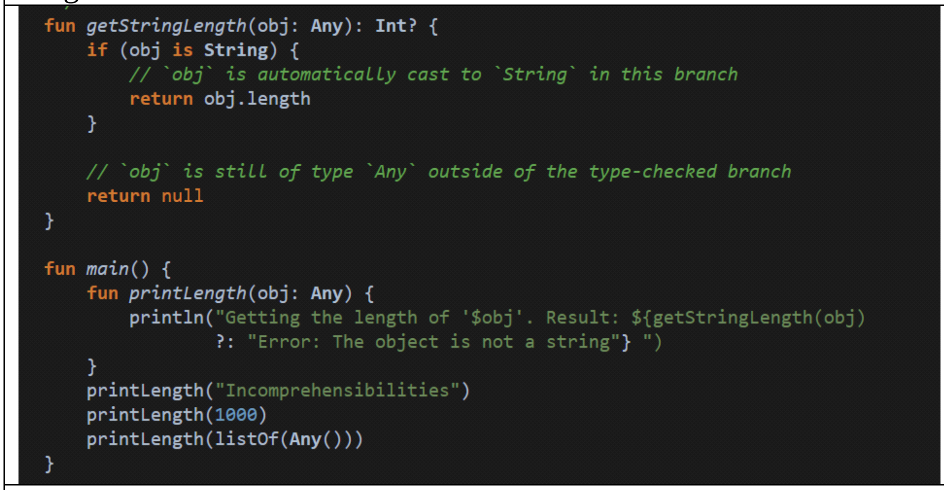


```
42
Wrong number format in arg1: 'a'
Wrong number format in arg2: 'b'
```

Using type checks and automatic casts

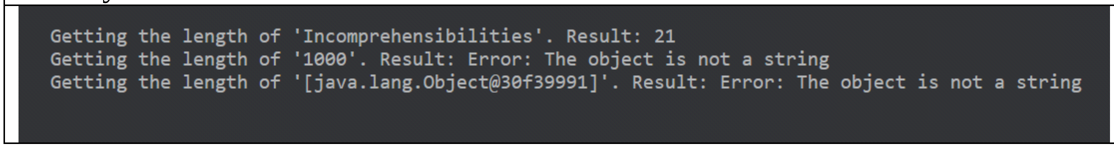
The is operator checks if an expression is an instance of a type. If an immutable local variable or property is checked for a specific type, there's no need to cast it explicitly:

Program



```
fun getStringLength(obj: Any): Int? {
    if (obj is String) {
        // `obj` is automatically cast to `String` in this branch
        return obj.length
    }

    // `obj` is still of type `Any` outside of the type-checked branch
    return null
}

fun main() {
    fun printLength(obj: Any) {
        println("Getting the length of '$obj'. Result: ${getStringLength(obj)
              ?: "Error: The object is not a string"} ")
    }
    printLength("Incomprehensibilities")
    printLength(1000)
    printLength(listOf(Any()))
}
```

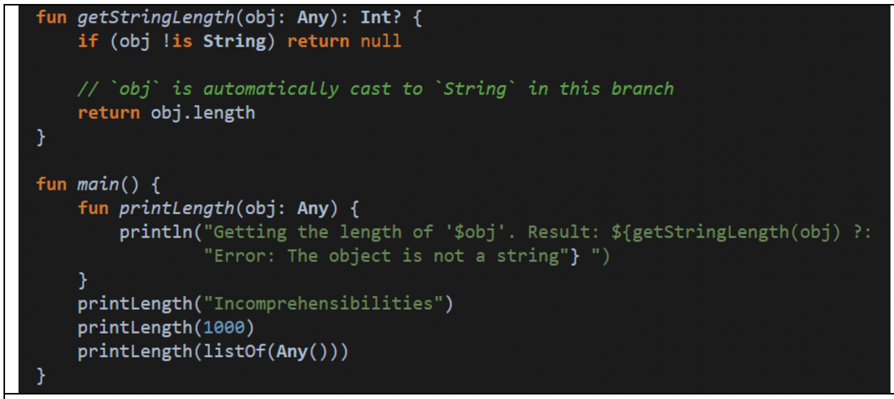
Hasilnya



```
Getting the length of 'Incomprehensibilities'. Result: 21
Getting the length of '1000'. Result: Error: The object is not a string
Getting the length of '[java.lang.Object@30f39991]'. Result: Error: The object is not a string
```

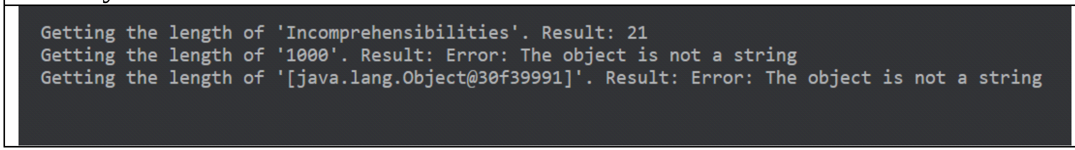
Or

Program



```
fun getStringLength(obj: Any): Int? {
    if (obj !is String) return null

    // `obj` is automatically cast to `String` in this branch
    return obj.length
}

fun main() {
    fun printLength(obj: Any) {
        println("Getting the length of '$obj'. Result: ${getStringLength(obj) ?:
                "Error: The object is not a string"} ")
    }
    printLength("Incomprehensibilities")
    printLength(1000)
    printLength(listOf(Any()))
}
```

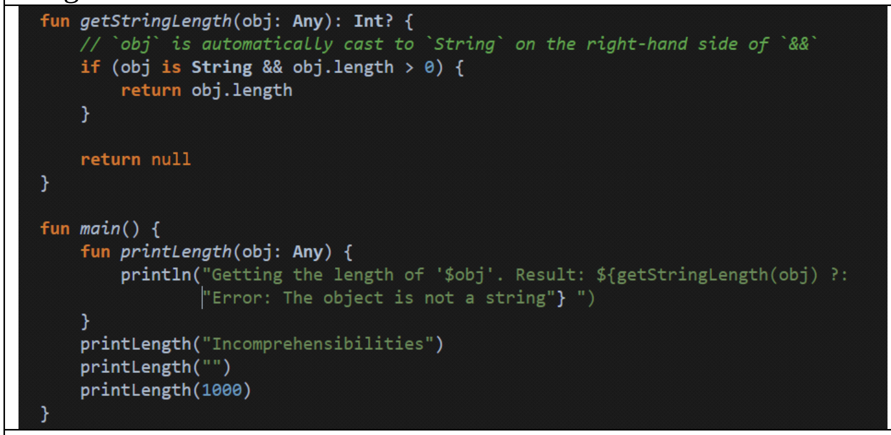
Hasilnya



```
Getting the length of 'Incomprehensibilities'. Result: 21
Getting the length of '1000'. Result: Error: The object is not a string
Getting the length of '[java.lang.Object@30f39991]'. Result: Error: The object is not a string
```

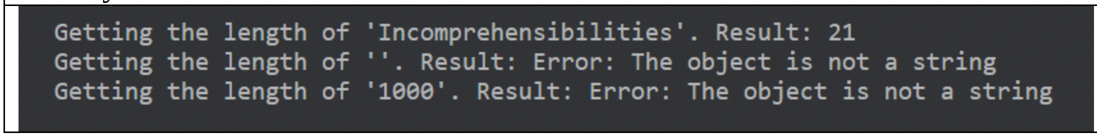
Or even

Program



```
fun getStringLength(obj: Any): Int? {
    // `obj` is automatically cast to `String` on the right-hand side of `&&`
    if (obj is String && obj.length > 0) {
        return obj.length
    }

    return null
}

fun main() {
    fun printLength(obj: Any) {
        println("Getting the length of '$obj'. Result: ${getStringLength(obj) ?:
                "Error: The object is not a string"} ")
    }
    printLength("Incomprehensibilities")
    printLength("")
    printLength(1000)
}
```

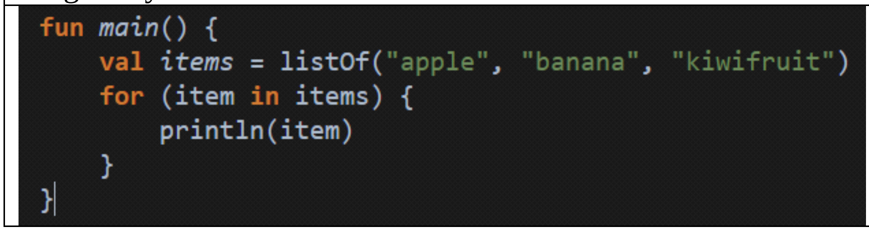
Hasilnya



```
Getting the length of 'Incomprehensibilities'. Result: 21
Getting the length of ''. Result: Error: The object is not a string
Getting the length of '1000'. Result: Error: The object is not a string
```

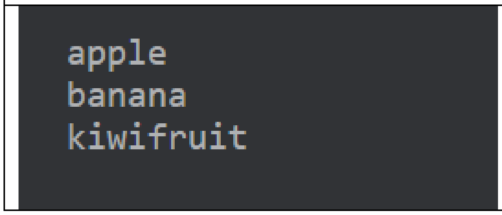
Using a for loop

Programnya



```
fun main() {
    val items = listOf("apple", "banana", "kiwifruit")
    for (item in items) {
        println(item)
    }
}
```

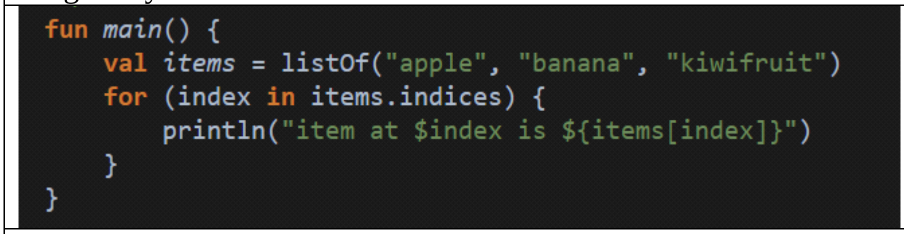
Hasil setelah dirun



```
apple
banana
kiwifruit
```

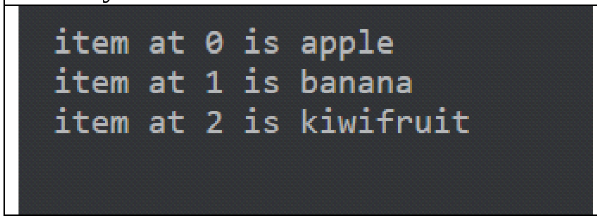
Or

Programnya



```kotlin
fun main() {
    val items = listOf("apple", "banana", "kiwifruit")
    for (index in items.indices) {
        println("item at $index is ${items[index]}")
    }
}
```

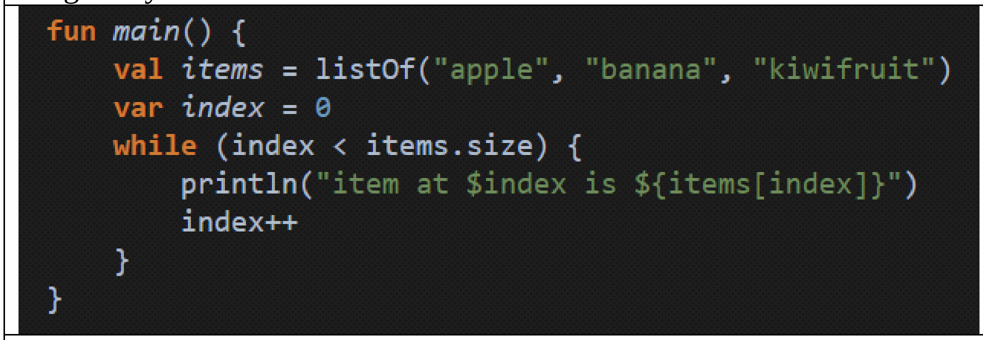
Hasilnya



```
item at 0 is apple
item at 1 is banana
item at 2 is kiwifruit
```

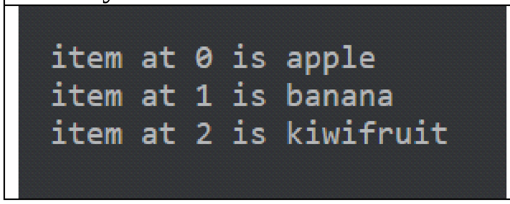
Using a while loop

Programnya



```kotlin
fun main() {
    val items = listOf("apple", "banana", "kiwifruit")
    var index = 0
    while (index < items.size) {
        println("item at $index is ${items[index]}")
        index++
    }
}
```

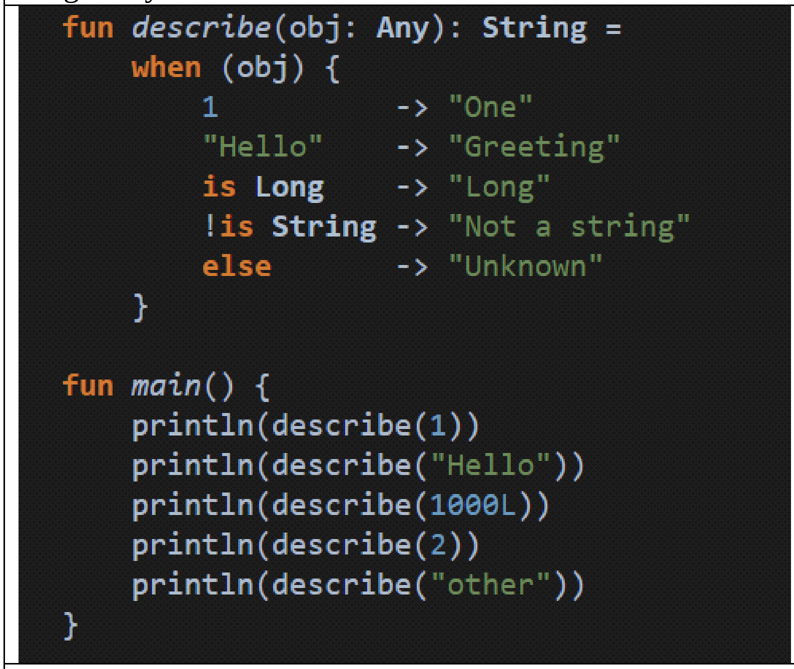
Hasilnya



```
item at 0 is apple
item at 1 is banana
item at 2 is kiwifruit
```

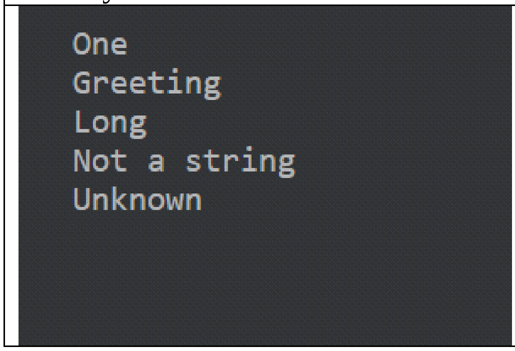
Using when expression

Programnya



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

fun main() {
    println(describe(1))
    println(describe("Hello"))
    println(describe(1000L))
    println(describe(2))
    println(describe("other"))
}
```

Hasilnya

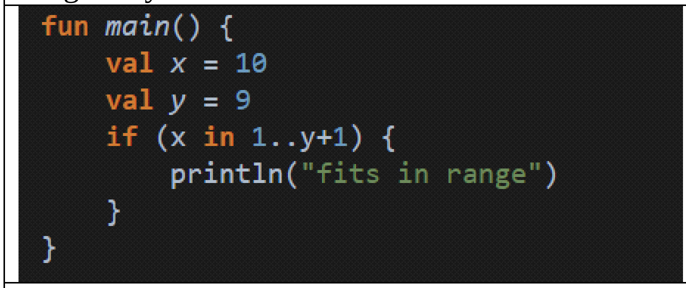


```
One
Greeting
Long
Not a string
Unknown
```

Using ranges

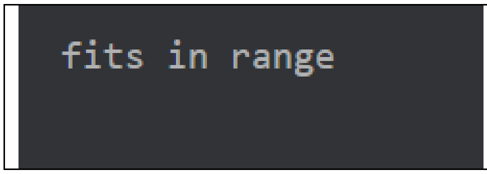
Check if a number is within a range using in operator:

Programnya



```
fun main() {
    val x = 10
    val y = 9
    if (x in 1..y+1) {
        println("fits in range")
    }
}
```

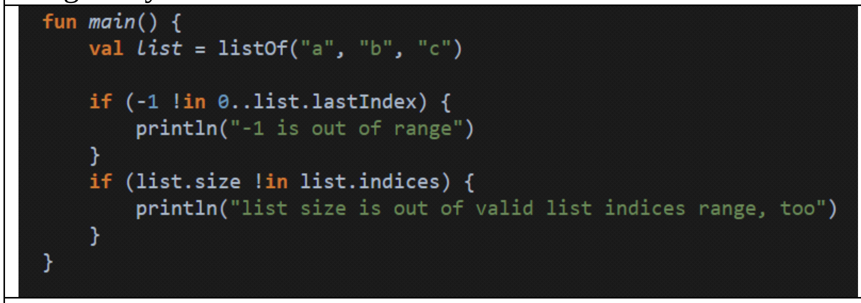
Hasilnya



```
fits in range
```

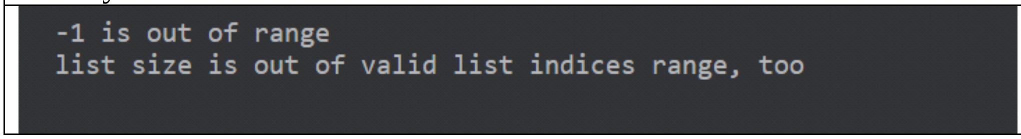
Check if a number is out of range:

Programnya



```
fun main() {
    val list = listOf("a", "b", "c")

    if (-1 !in 0..list.lastIndex) {
        println("-1 is out of range")
    }
    if (list.size !in list.indices) {
        println("list size is out of valid list indices range, too")
    }
}
```

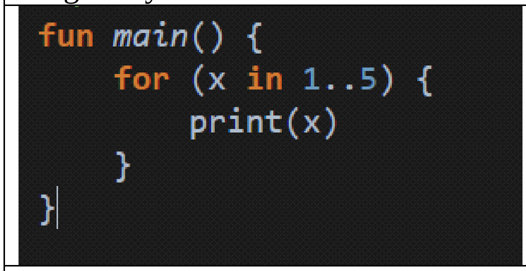
Hasilnya



```
-1 is out of range
list size is out of valid list indices range, too
```

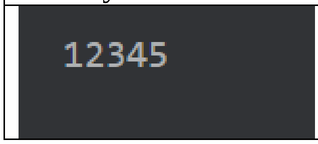
Iterating over a range:

Programnya



```
fun main() {
    for (x in 1..5) {
        print(x)
    }
}
```

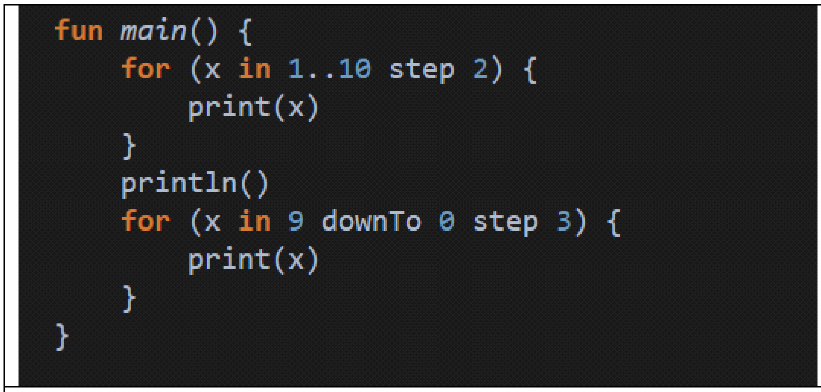
Hasilnya



```
12345
```

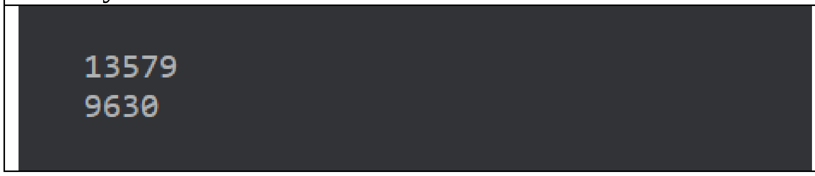
or over a progression:

Program



```
fun main() {
    for (x in 1..10 step 2) {
        print(x)
    }
    println()
    for (x in 9 downTo 0 step 3) {
        print(x)
    }
}
```

Hasilnya

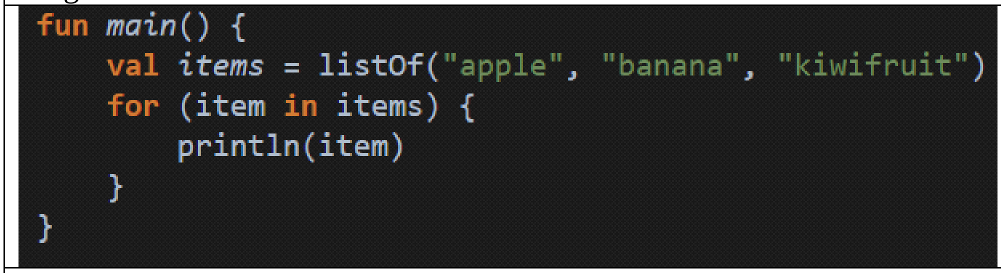


```
13579
9630
```

Using collections

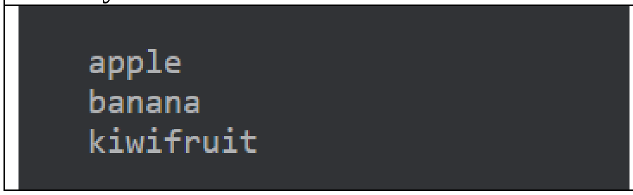
Iterating over a collection:

Program



```
fun main() {
    val items = listOf("apple", "banana", "kiwifruit")
    for (item in items) {
        println(item)
    }
}
```

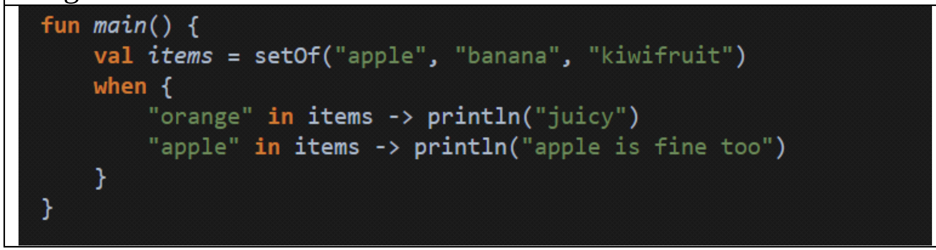
Hasilnya



```
apple
banana
kiwifruit
```

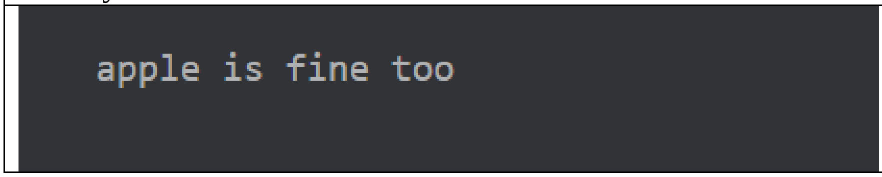
Checking if a collection contains an object using in operator:

Program



```
fun main() {
    val items = setOf("apple", "banana", "kiwifruit")
    when {
        "orange" in items -> println("juicy")
        "apple" in items -> println("apple is fine too")
    }
}
```

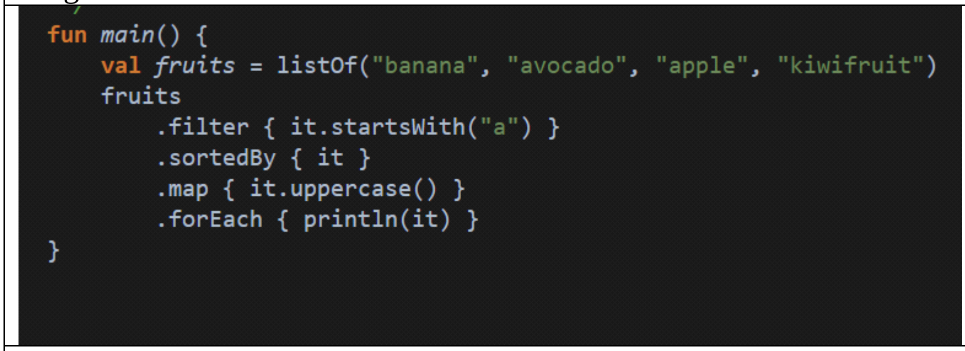
Hasilnya



```
apple is fine too
```

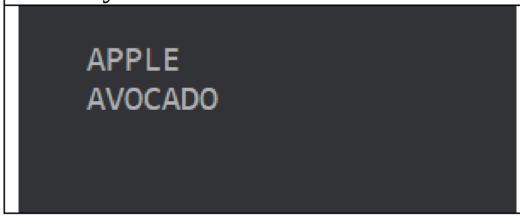
Using lambda expressions to filter and map collections:

Program



```
fun main() {
    val fruits = listOf("banana", "avocado", "apple", "kiwifruit")
    fruits
        .filter { it.startsWith("a") }
        .sortedBy { it }
        .map { it.uppercase() }
        .forEach { println(it) }
}
```

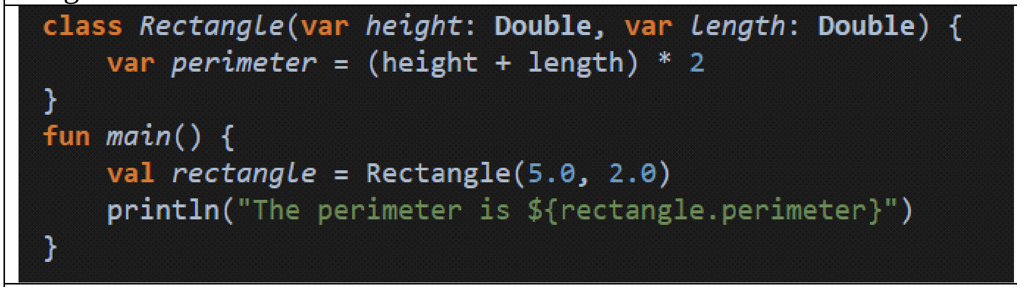
Hasilnya



```
APPLE
AVOCADO
```

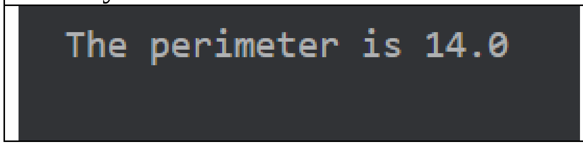
Creating basic classes and their instances:

Program



```
class Rectangle(var height: Double, var length: Double) {
    var perimeter = (height + length) * 2
}
fun main() {
    val rectangle = Rectangle(5.0, 2.0)
    println("The perimeter is ${rectangle.perimeter}")
}
```

Hasilnya



```
The perimeter is 14.0
```

D. KESIMPULAN

Mahasiswa mampu memahami dan mengembangkan aplikasi sederhana dengan bahasa pemrograman Kotlin. Kotlin/Native memungkinkan developer untuk menggunakannya sebagai bahasa pemrograman dalam pengembangan aplikasi di platform lain seperti embedded system, desktop, macOS, dan iOS.