

Basic Syntax II



More of the basics of
Kotlin



Using nullable values and checking for `null`

A reference must be explicitly marked as nullable when `null` value is possible.

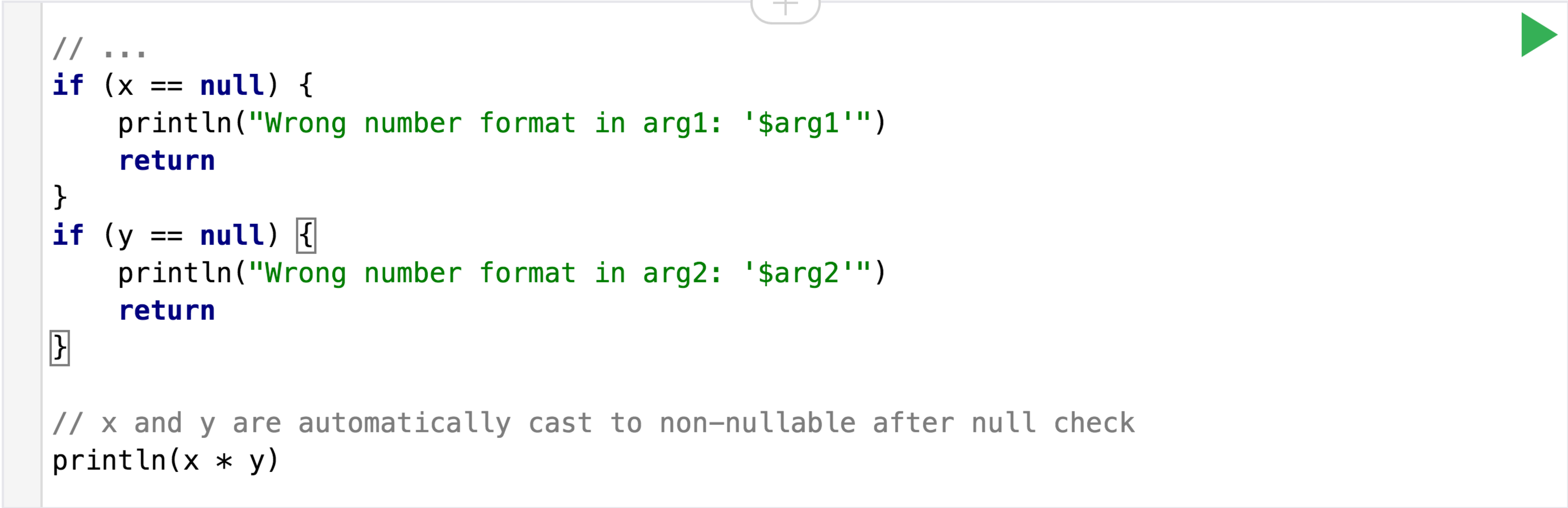
Return `null` if `str` does not hold an integer:

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

Use a function returning nullable value:



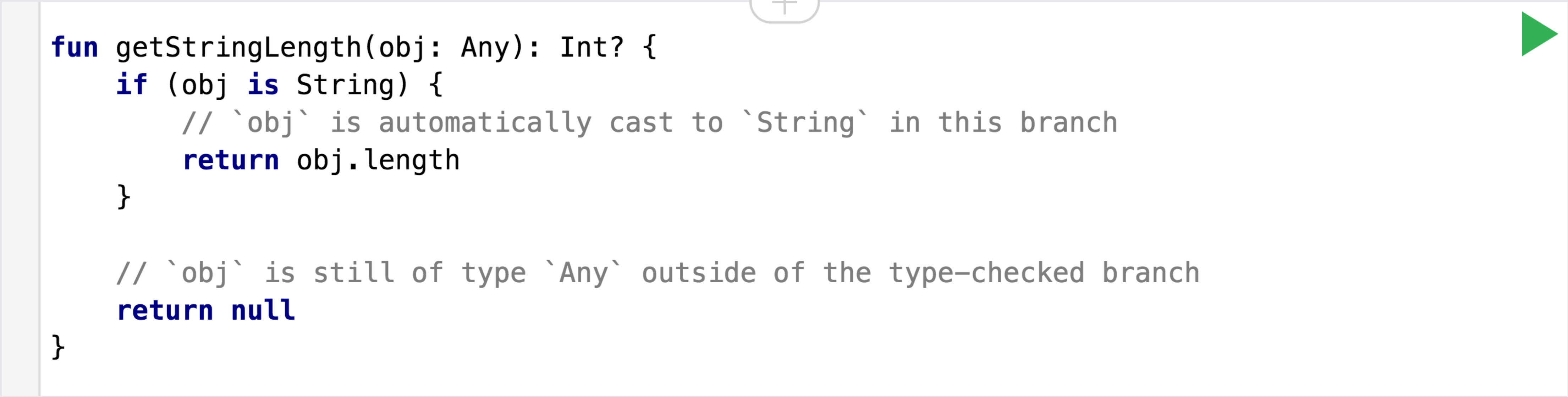
```
fun printProduct(arg1: String, arg2: String) {  
    val x = parseInt(arg1)  
    val y = parseInt(arg2)  
  
    // Using `x * y` yields error because they may hold nulls.  
    if (x != null && y != null) {  
        // x and y are automatically cast to non-nullable after null check  
        println(x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



```
// ...  
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)
```

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:




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



Using a for loop




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




Target platform: JVM Running on kotlin v. 1.2.70

or



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



Using a while loop



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



Using when expression




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



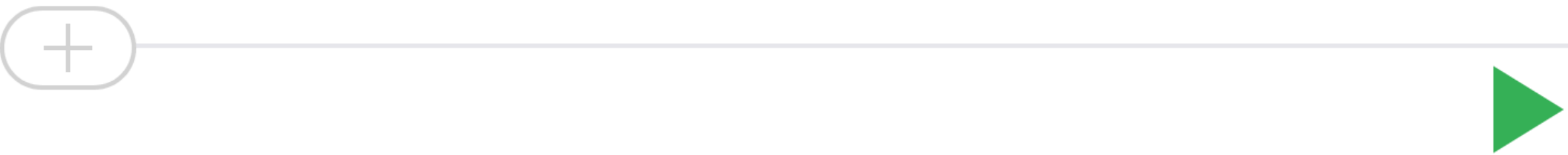
Using ranges

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



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

Check if a number is out of range:

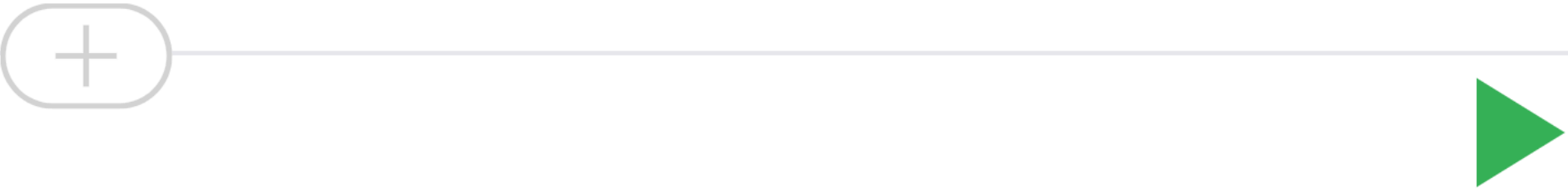


```
val list = listOf("a", "b", "c")

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")
}
```

Using collections

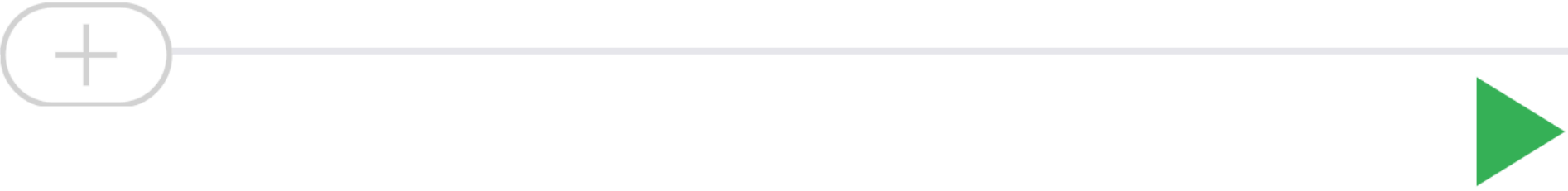
Iterating over a collection:



```
for (item in items) {  
    println(item)  
}
```

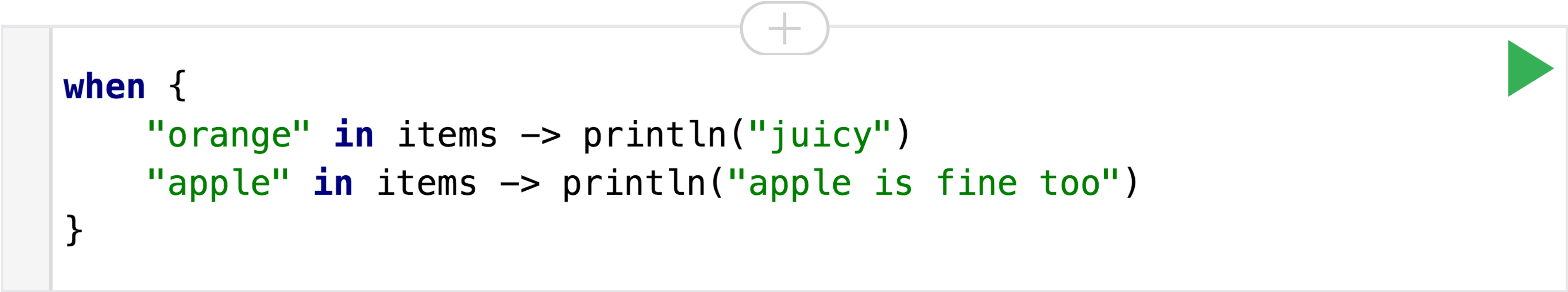
Target platform: JVM Running on kotlin v. 1.2.70

Checking if a collection contains an object using **in** operator:



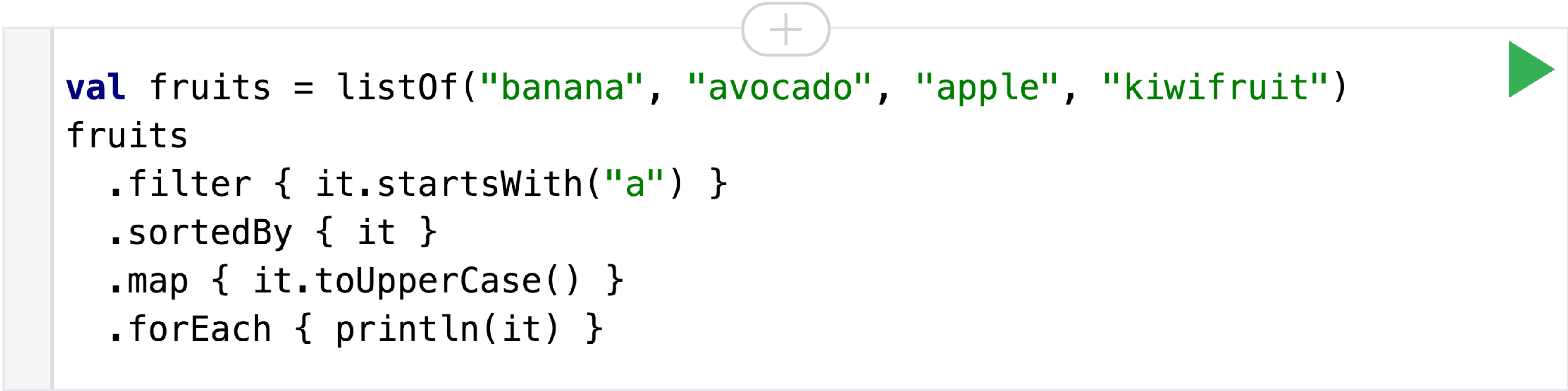
```
when {  
    "orange" in items -> println("juicy")  
    "apple" in items -> println("apple is fine too")  
}
```

Checking if a collection contains an object using `in` operator:





```
when {  
  "orange" in items -> println("juicy")  
  "apple" in items -> println("apple is fine too")  
}
```

Using lambda expressions to filter and map collections:



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

Creating basic classes and their instances:

```
val rectangle = Rectangle(5.0, 2.0) //no 'new' keyword required  
val triangle = Triangle(3.0, 4.0, 5.0)
```