Lecture 2

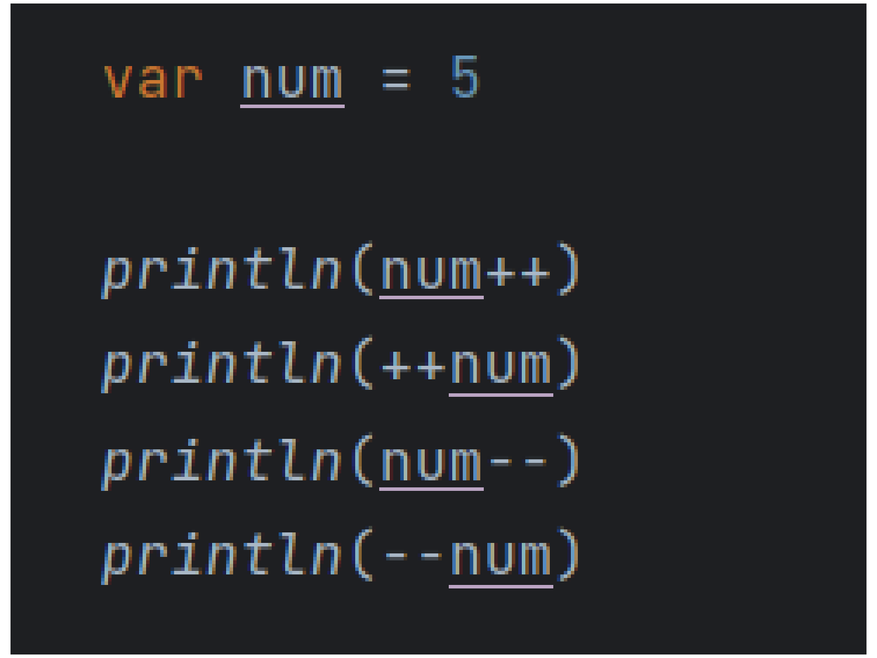
COMP 3717- Mobile Dev with Android Tech

第2讲

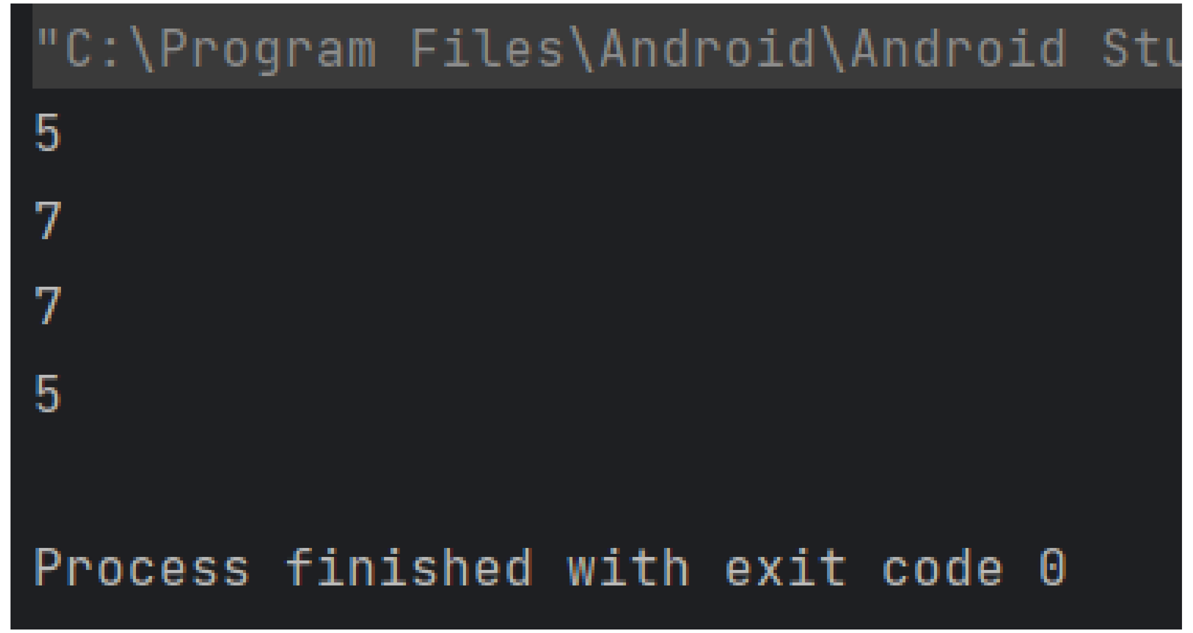
COMP 3717 - 使用Android技术进行移动开发

++ and -- operators

- Notice the position of the operators are tied to when incrementing or decrementing happens



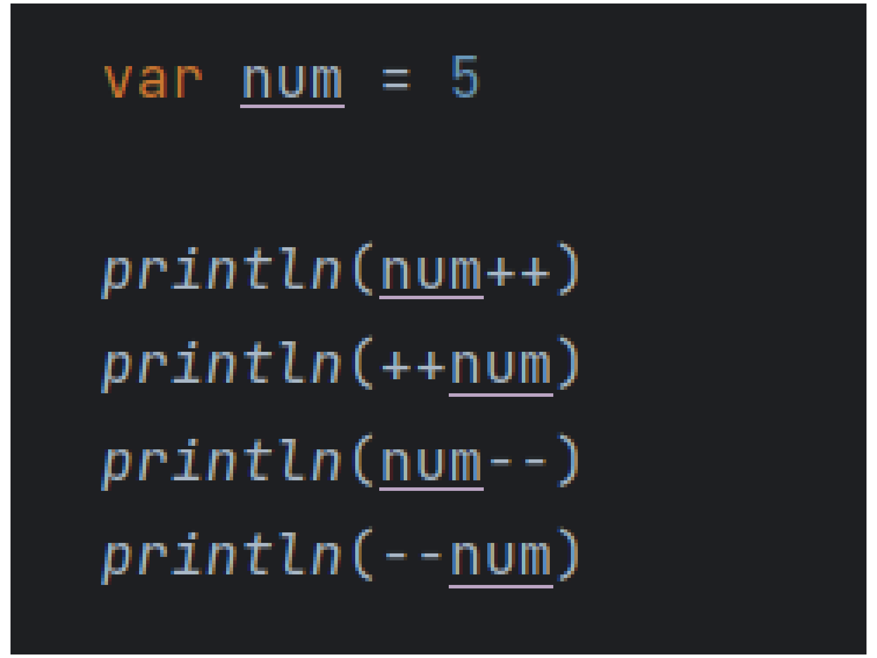
```
var num = 5

println(num++)
println(++num)
println(num--)
println(--num)
```



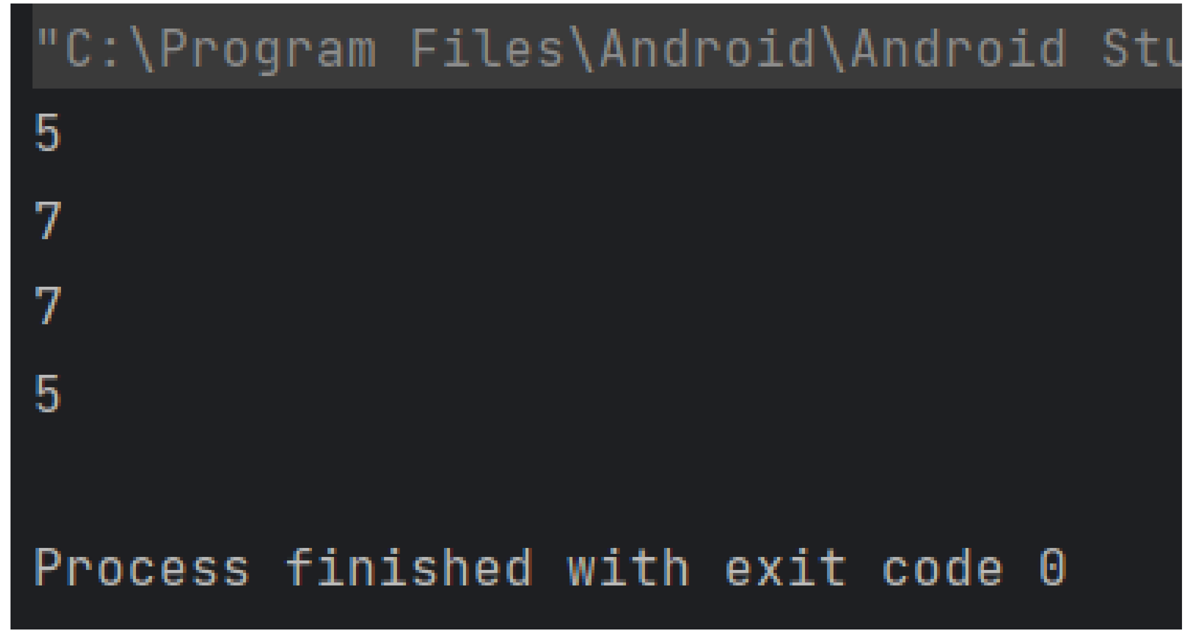
```
"C:\Program Files\Android\Android Stu
5
7
7
5

Process finished with exit code 0
```

++ 和 -- 运算符

- 注意运算符的位置与递增或递减发生的时间相关



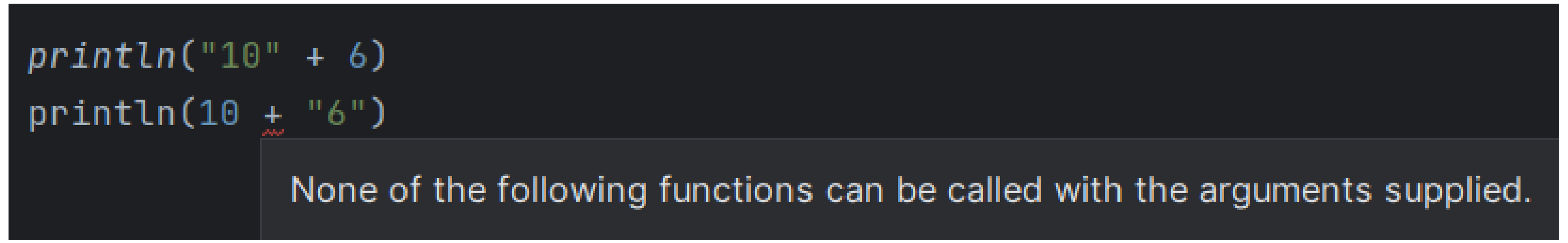
```
var num = 5

println(num++)
println(++num)
println(num--)
println(--num)
```



```
"C:\Program Files\Android\Android Stu
5
7
7
5

Process finished with exit code 0
```

Converting data types

- Notice we can use the *+ operator* with line 1 but not line 2



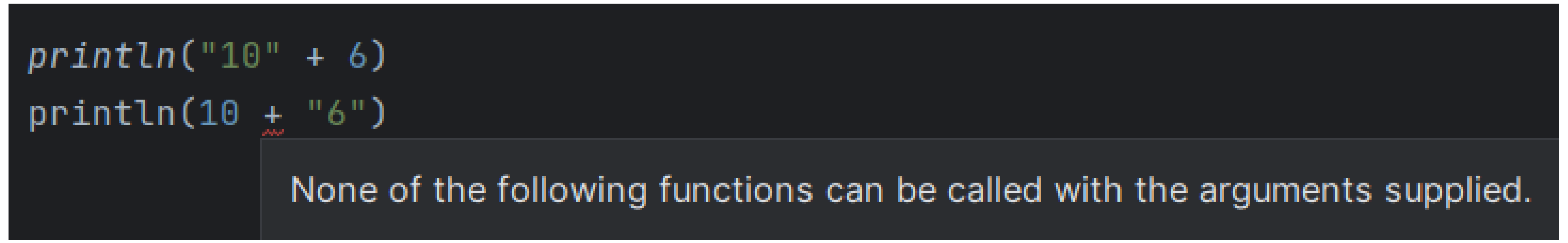
```
println("10" + 6)
println(10 + "6")
```

None of the following functions can be called with the arguments supplied.

- When a string is first type using the *+ operator*, all other types are converted into their string representation

转换数据类型

- 注意，我们可以对第1行使用*+ operator*，但不能对第2行使用



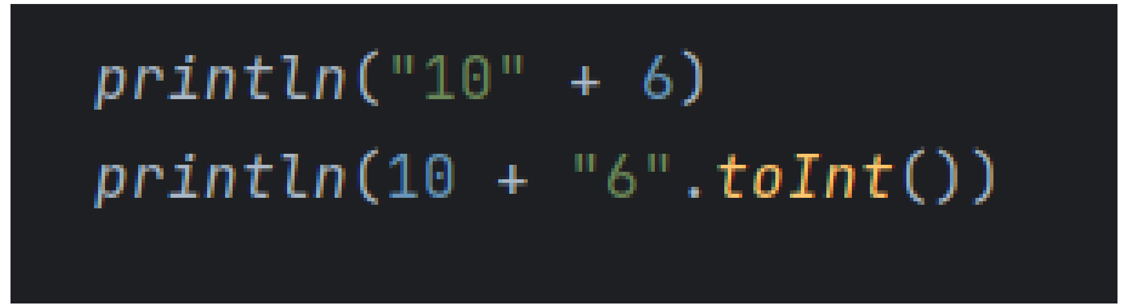
```
println("10" + 6)
println(10 + "6")
```

None of the following functions can be called with the arguments supplied.

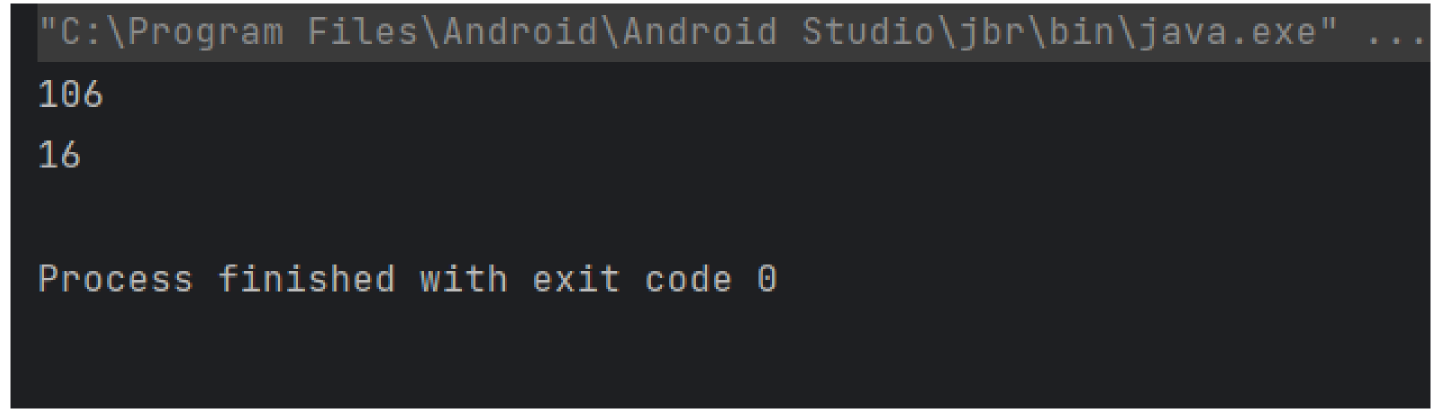
- 当首次对字符串使用*+ operator*时，所有其他类型都将被转换为其字符串表示形式

Converting data types (cont.)

- To add the two numbers together on the second line we have to convert it ourselves



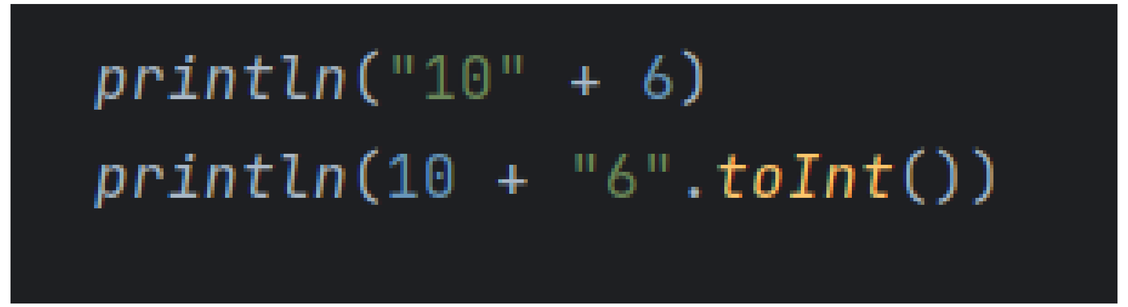
```
println("10" + 6)
println(10 + "6".toInt())
```



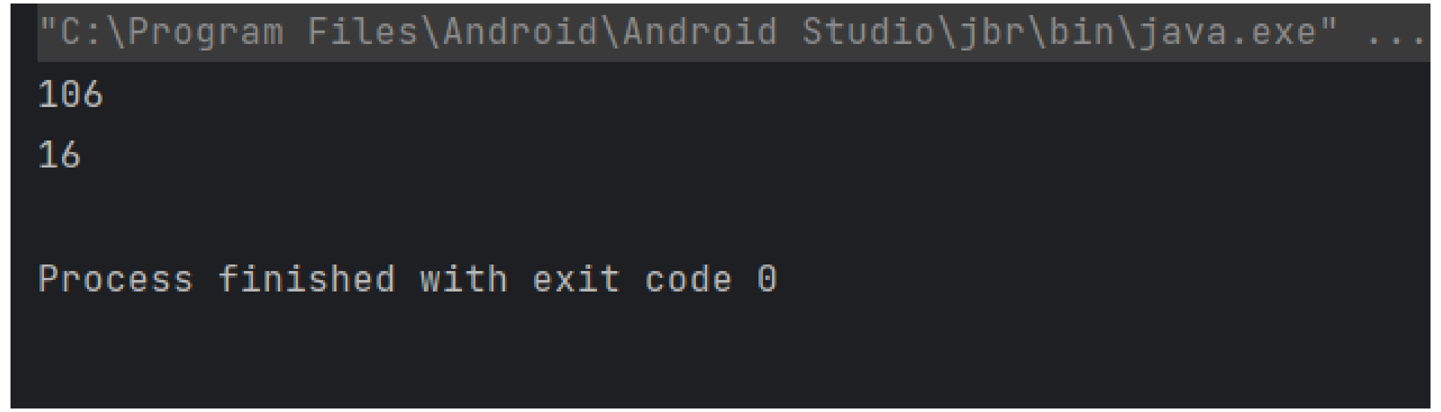
```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
106
16

Process finished with exit code 0
```

转换数据类型（续）

- 要在第二行将这两个数字相加，我们必须自己进行转换



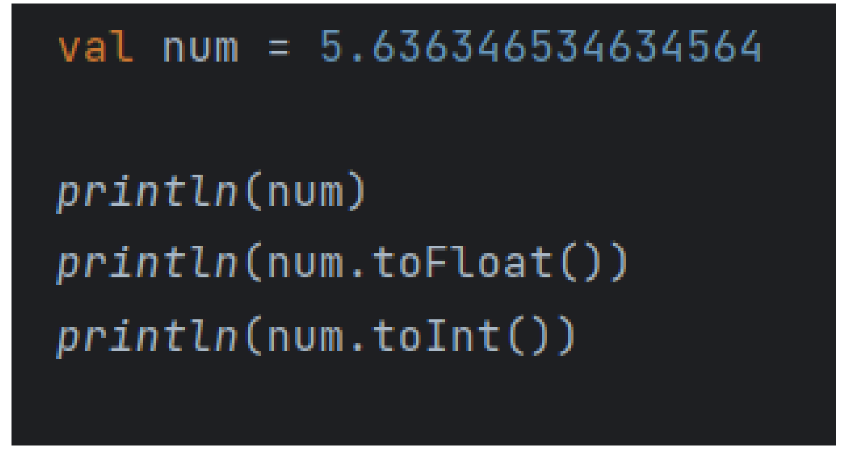
```
println("10" + 6)
println(10 + "6".toInt())
```



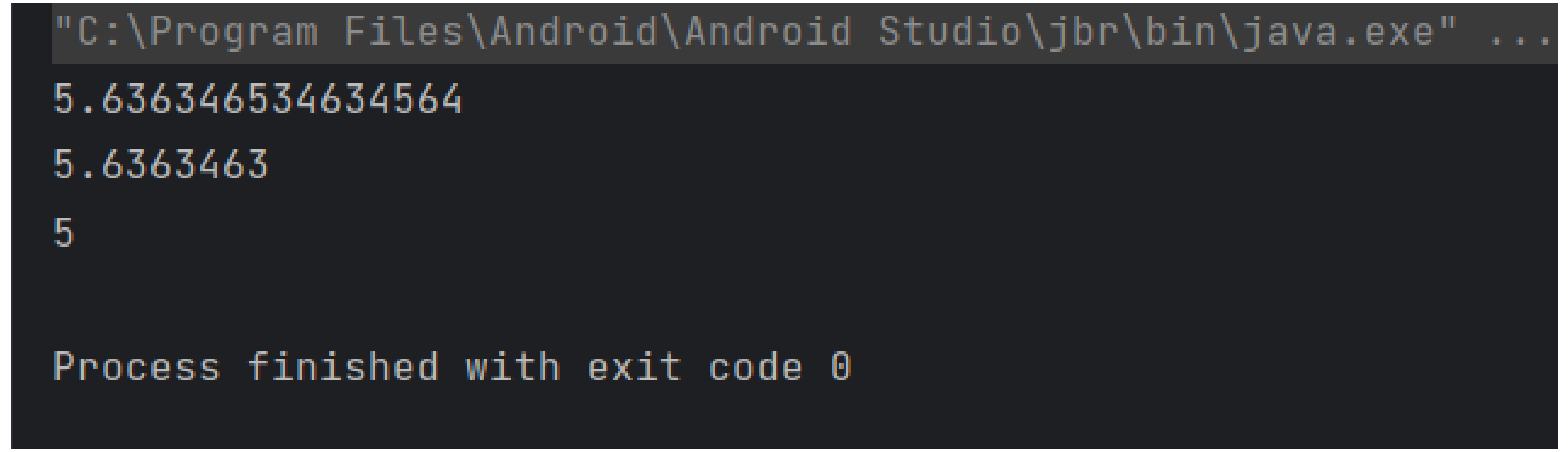
```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
106
16

Process finished with exit code 0
```

Converting data types (cont.)

- In this example, we are converting a double to different types



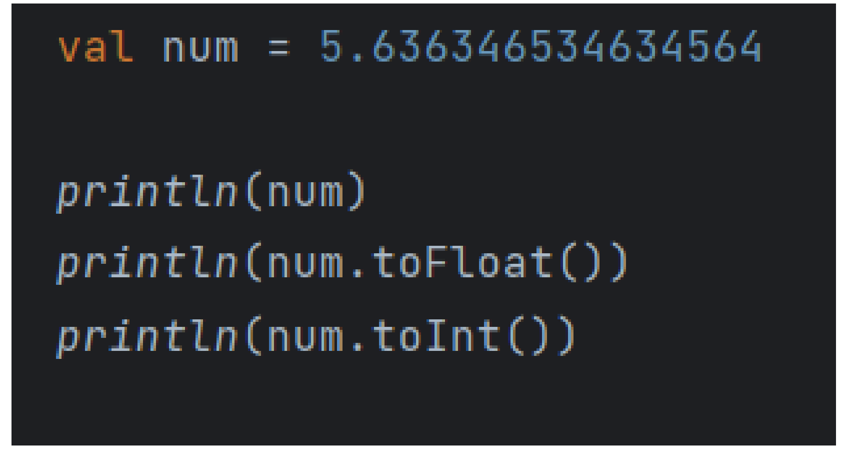
```
val num = 5.636346534634564

println(num)
println(num.toFloat())
println(num.toInt())
```



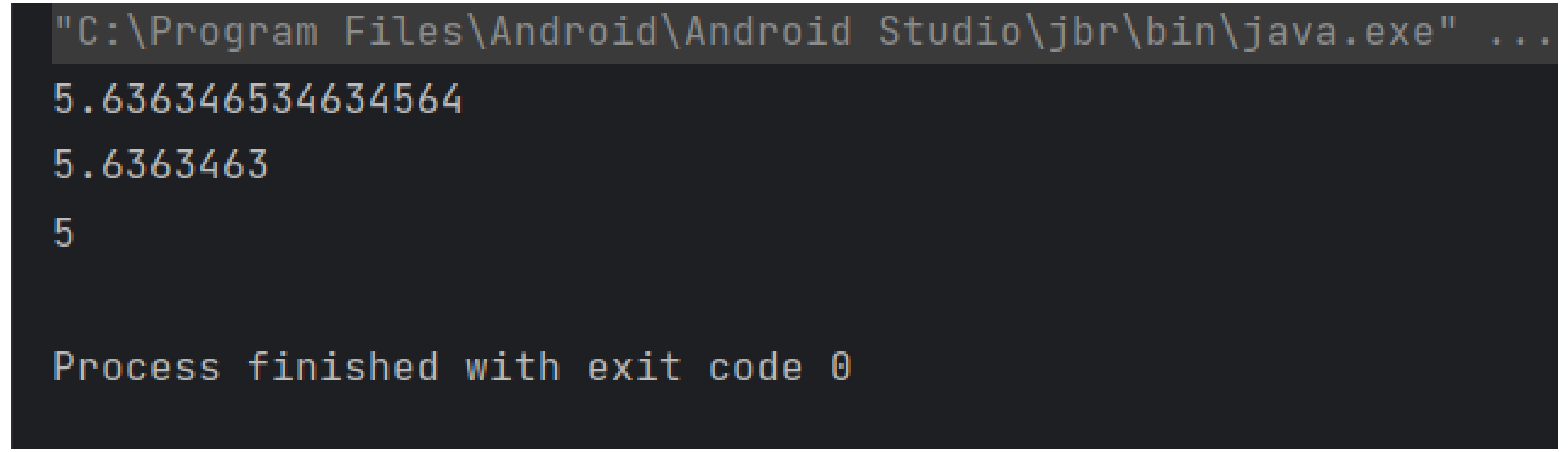
```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
5.636346534634564
5.6363463
5

Process finished with exit code 0
```

数据类型转换（续）

- 在此示例中，我们将一个 double 转换为不同类型



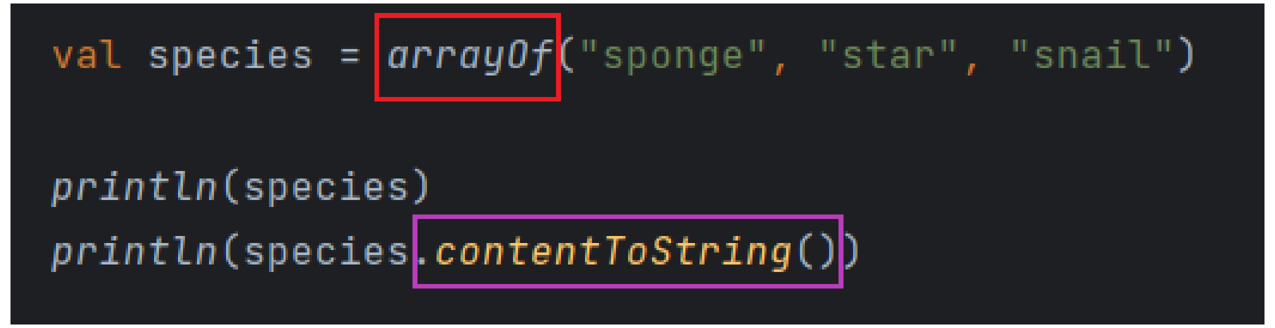
```
val num = 5.636346534634564

println(num)
println(num.toFloat())
println(num.toInt())
```



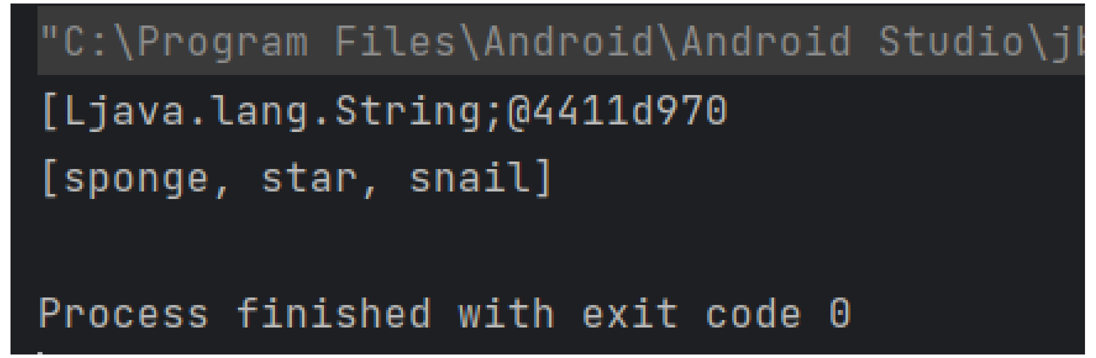
```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
5.636346534634564
5.6363463
5

Process finished with exit code 0
```

Arrays

- To hold multiple values in a datatype we can use an array
- To see the contents in an array we can use the *.contentToString* function



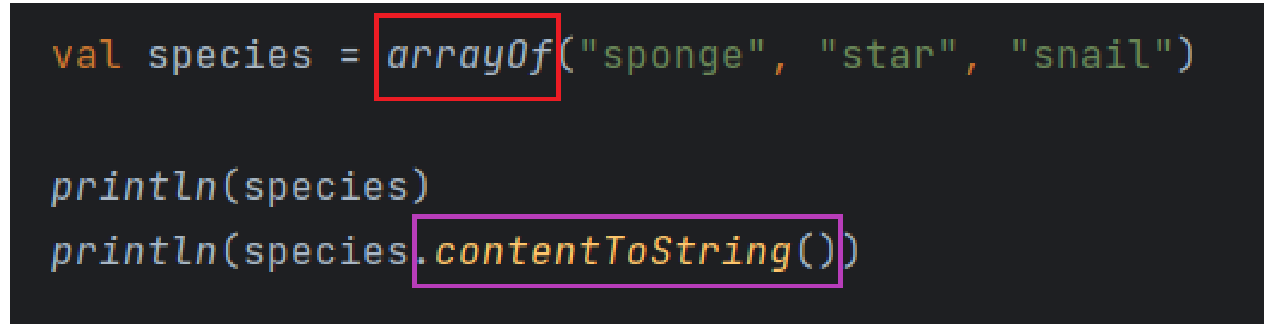
```
val species = arrayOf("sponge", "star", "snail")

println(species)
println(species.contentToString())
```



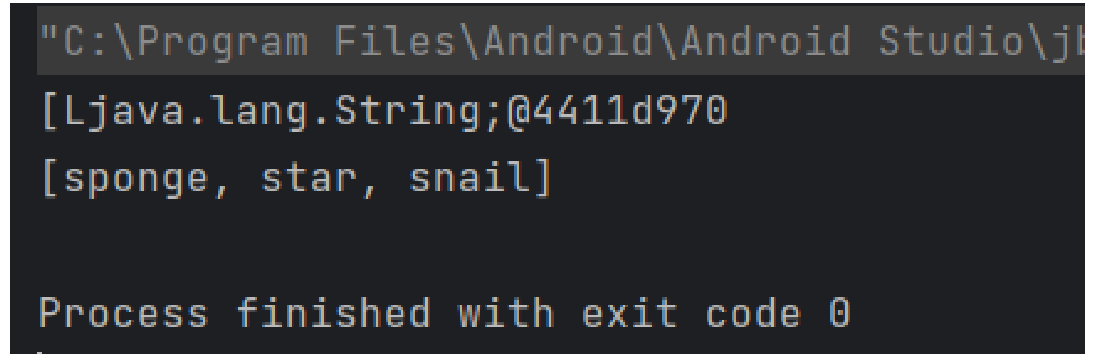
```
"C:\Program Files\Android\Android Studio\jt
[Ljava.lang.String;@4411d970
[sponge, star, snail]

Process finished with exit code 0
```

数组

- 要在一个数据类型中存储多个值，我们可以使用 数组
- 要查看数组中的内容，我们可以使用 *.contentToString* 函数



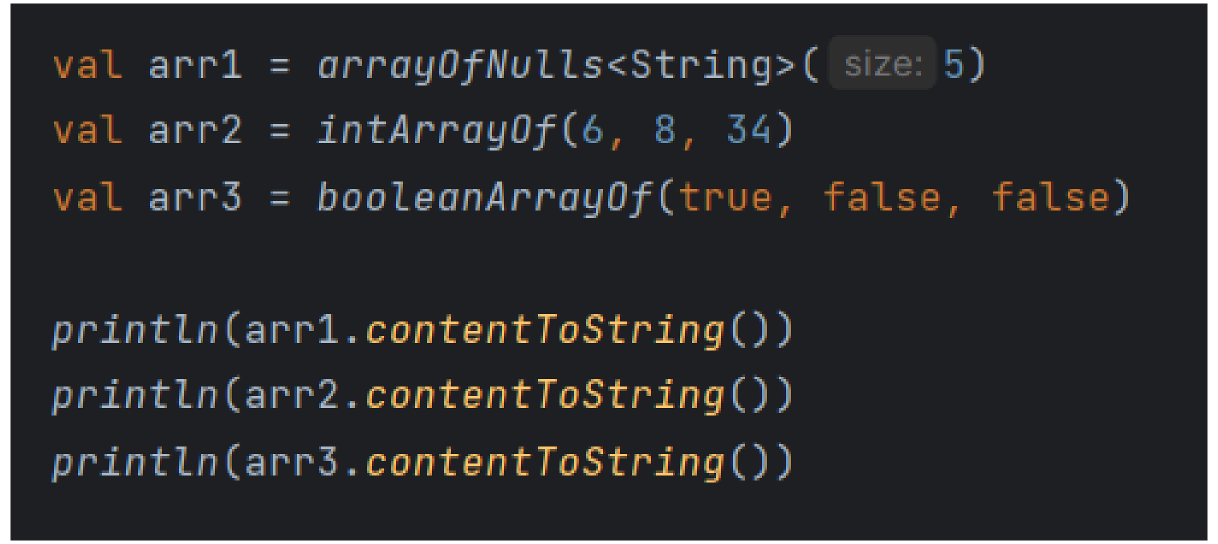
```
val species = arrayOf("sponge", "star", "snail")

println(species)
println(species.contentToString())
```



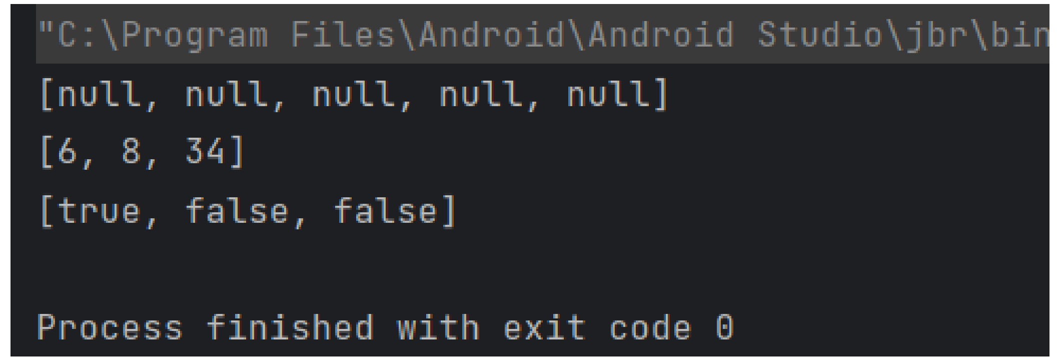
```
"C:\Program Files\Android\Android Studio\jt
[Ljava.lang.String;@4411d970
[sponge, star, snail]

Process finished with exit code 0
```

Arrays (cont.)

- There are lots of ways to create an array



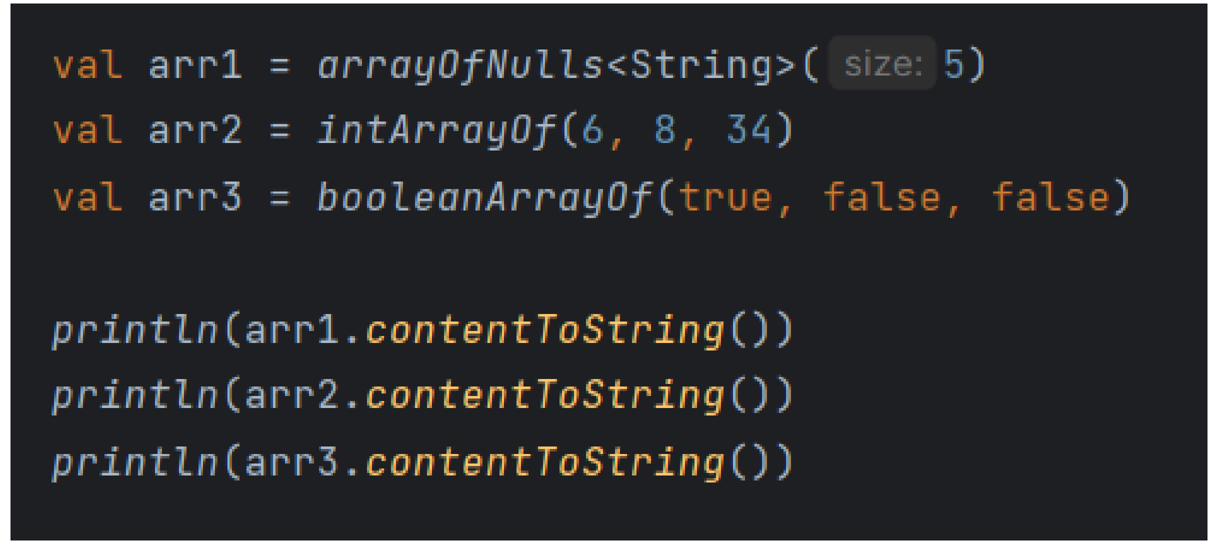
```kotlin
val arr1 = arrayOfNulls<String>( size: 5)
val arr2 = intArrayOf(6, 8, 34)
val arr3 = booleanArrayOf(true, false, false)

println(arr1.contentToString())
println(arr2.contentToString())
println(arr3.contentToString())
```



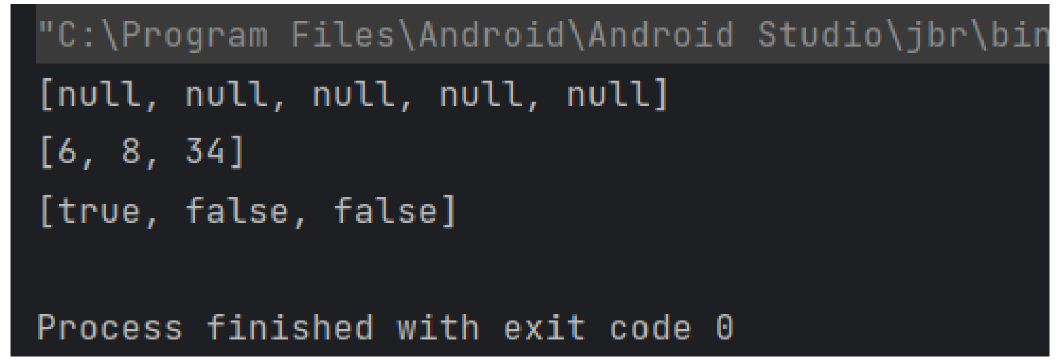
```
"C:\Program Files\Android\Android Studio\jbr\bin
[null, null, null, null, null]
[6, 8, 34]
[true, false, false]

Process finished with exit code 0
```

数组（续）

- 创建数组的方法有很多



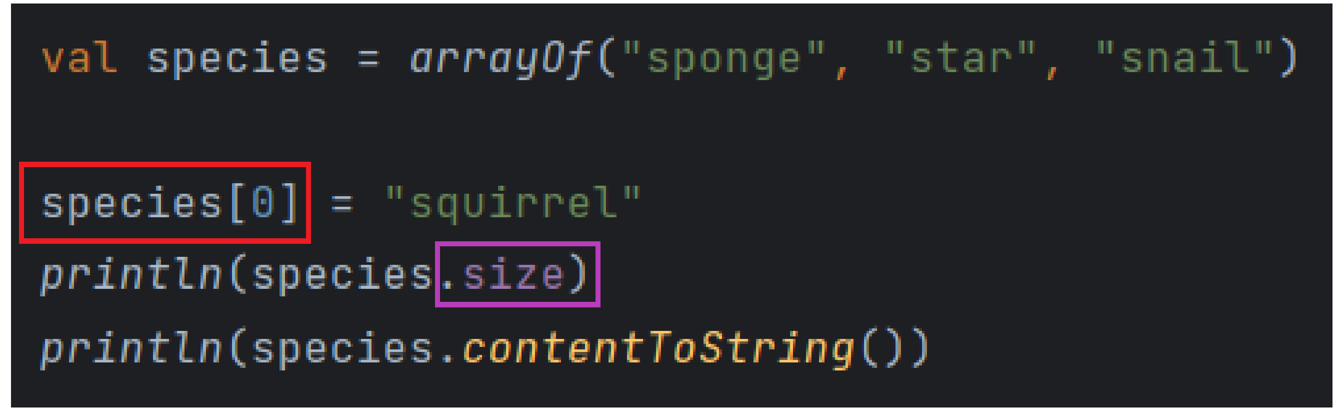
```kotlin
val arr1 = arrayOfNulls<String>( size: 5)
val arr2 = intArrayOf(6, 8, 34)
val arr3 = booleanArrayOf(true, false, false)

println(arr1.contentToString())
println(arr2.contentToString())
println(arr3.contentToString())
```



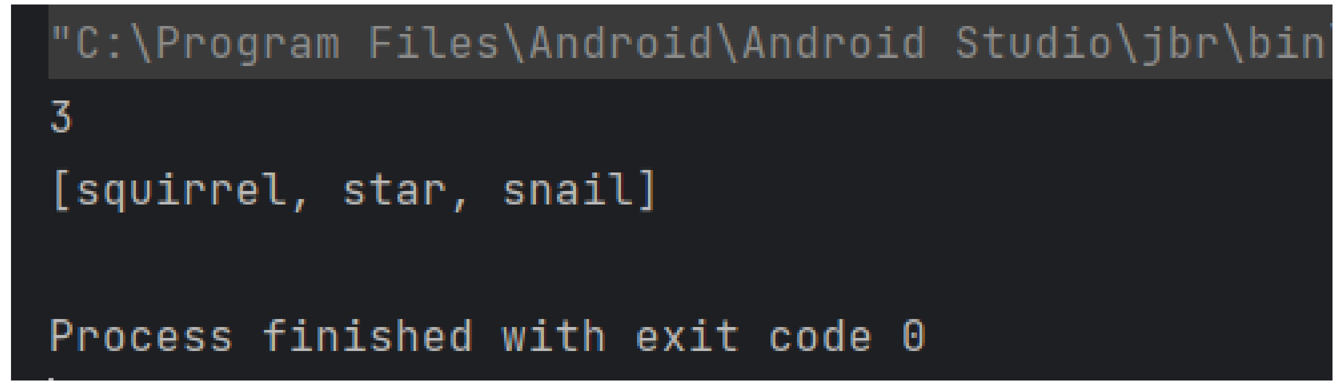
```
"C:\Program Files\Android\Android Studio\jbr\bin
[null, null, null, null, null]
[6, 8, 34]
[true, false, false]

Process finished with exit code 0
```

Arrays (cont.)

- You can change specific values of an array by accessing its index []
- You can also see how many elements are in the array using the *.size property*



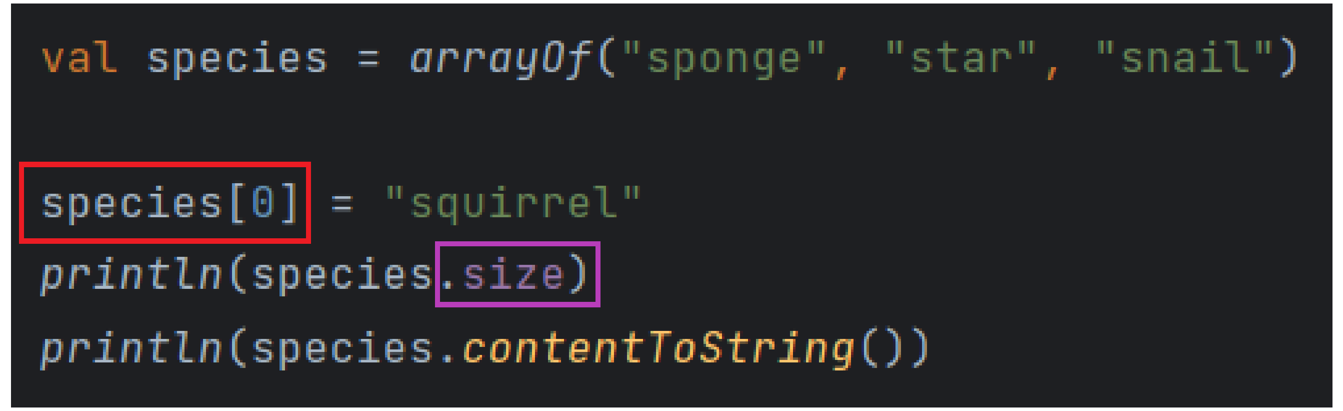
```
val species = arrayOf("sponge", "star", "snail")

species[0] = "squirrel"
println(species.size)
println(species.contentToString())
```



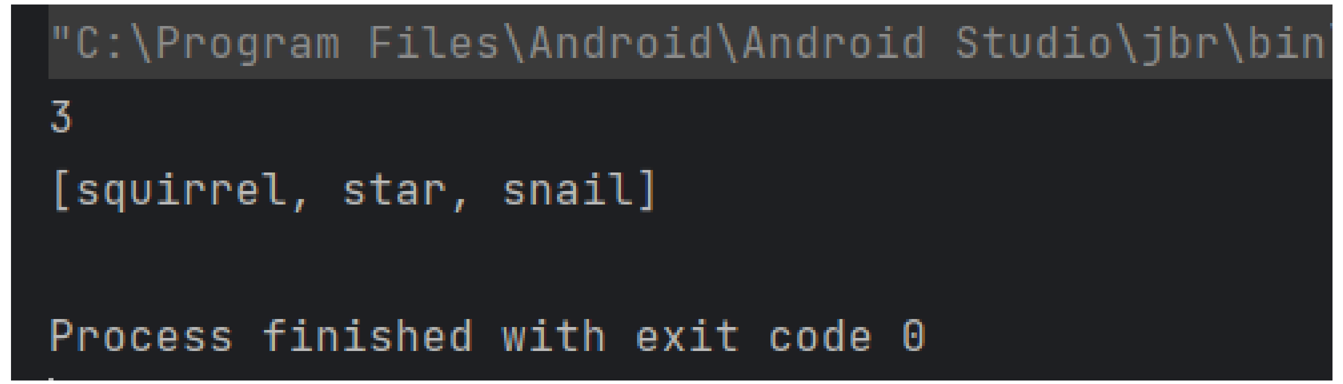
```
"C:\Program Files\Android\Android Studio\jbr\bin
3
[squirrel, star, snail]

Process finished with exit code 0
```

数组（续）

- 可以通过访问其 索引 []
- 还可以使用 *.size* 属性



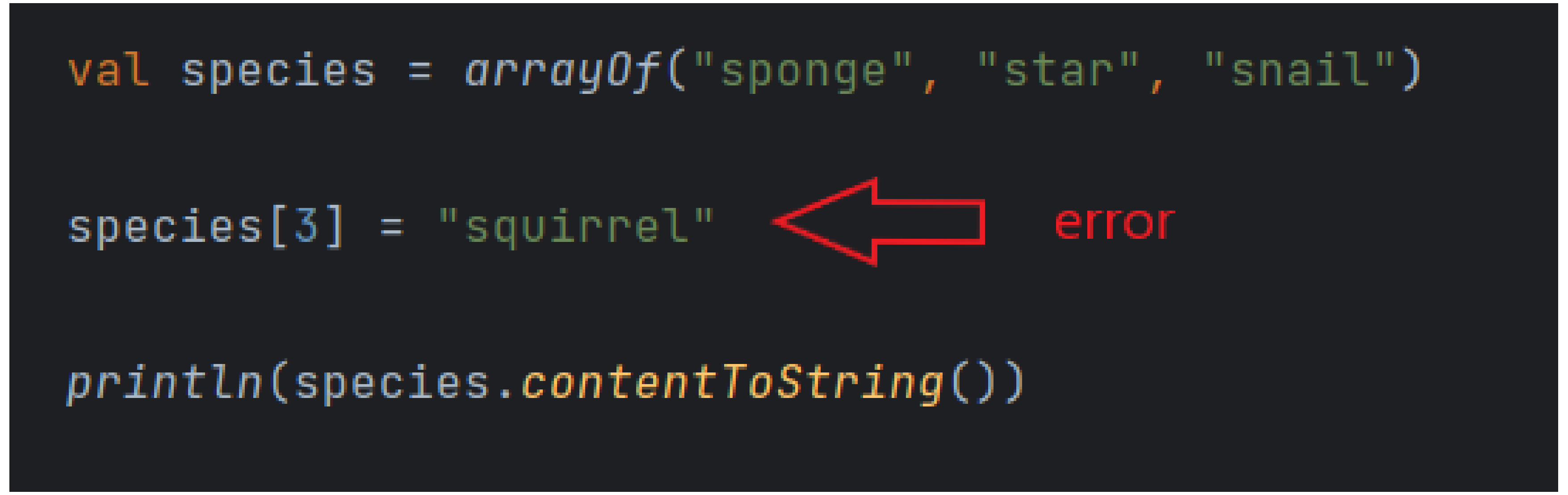
```
val species = arrayOf("sponge", "star", "snail")

species[0] = "squirrel"
println(species.size)
println(species.contentToString())
```



```
"C:\Program Files\Android\Android Studio\jbr\bin
3
[squirrel, star, snail]

Process finished with exit code 0
```

Arrays (cont.)

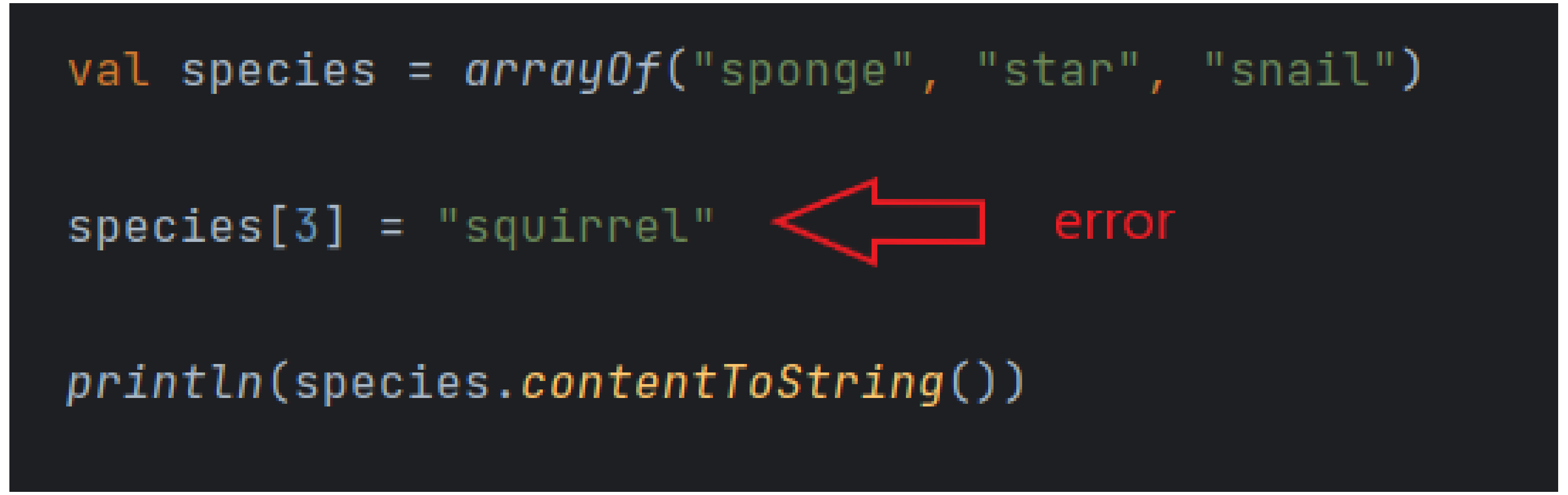
- Arrays can't be accessed or modified out of its bounds



```
val species = arrayOf("sponge", "star", "snail")

species[3] = "squirrel"

println(species.contentToString())
```

数组（续）

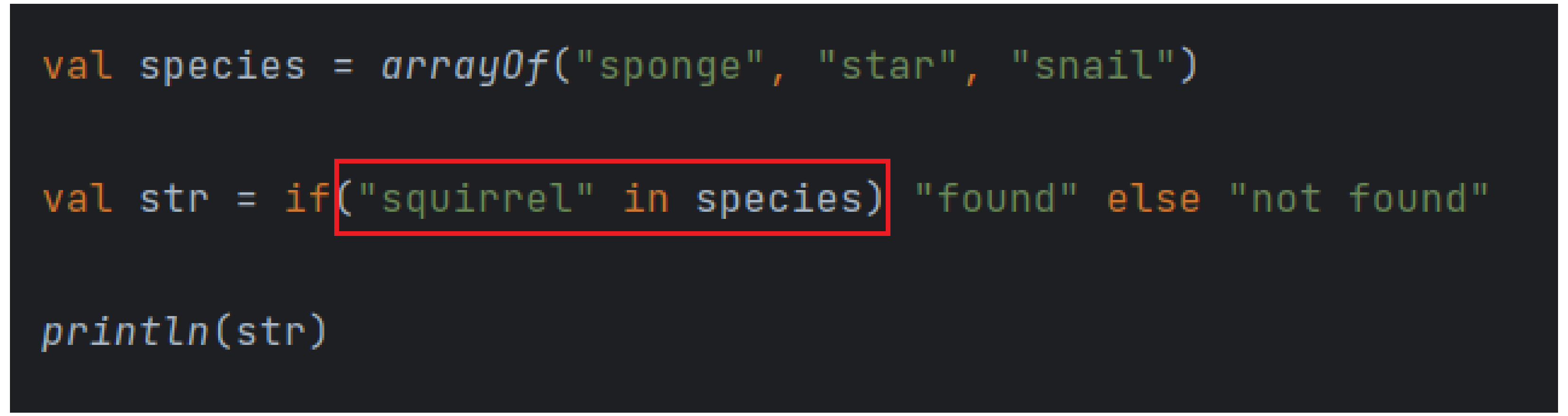
- 数组不能在其边界之外被访问或修改



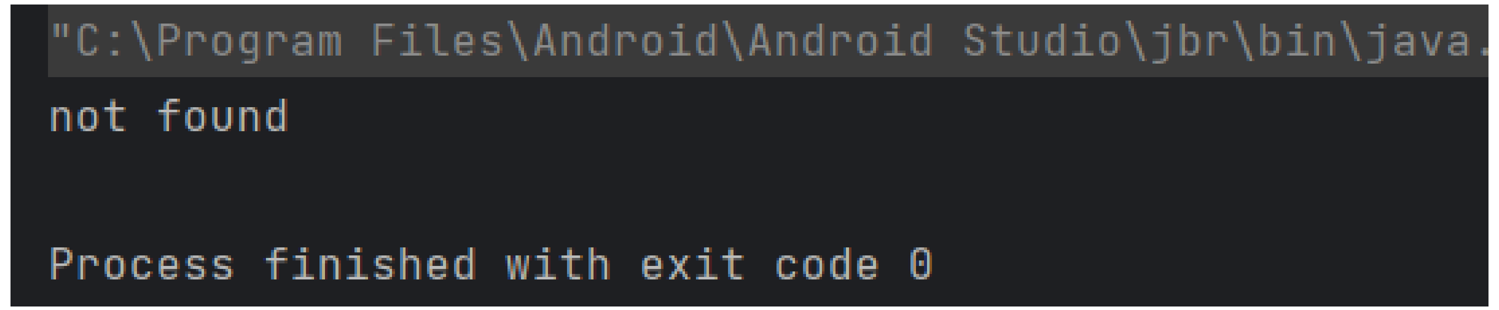
```
val species = arrayOf("sponge", "star", "snail")

species[3] = "squirrel"

println(species.contentToString())
```

Arrays (cont.)

- To check if an array contains a certain element, we can use the in keyword, or the *contains* function



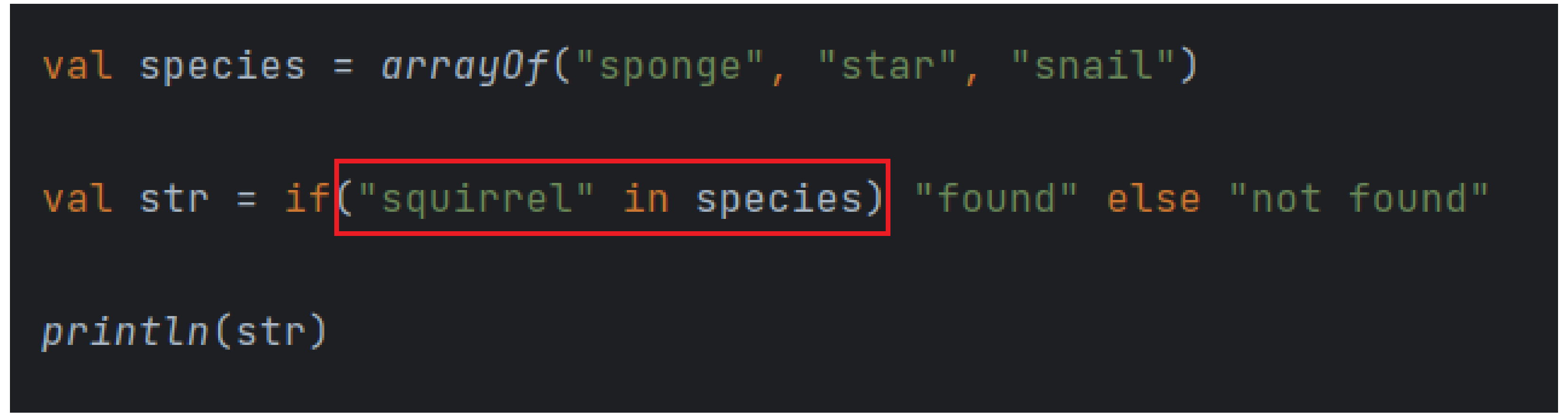
```
val species = arrayOf("sponge", "star", "snail")

val str = if("squirrel" in species) "found" else "not found"

println(str)
```



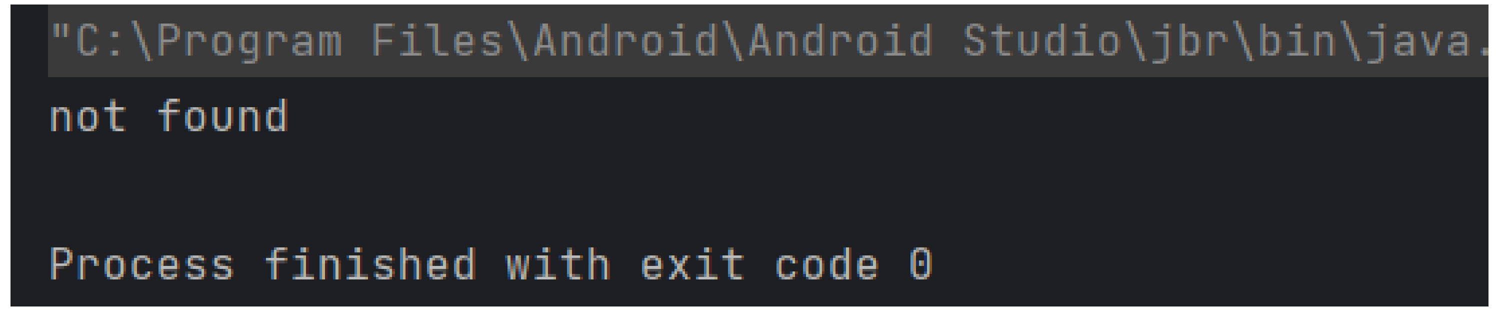
```
"C:\Program Files\Android\Android Studio\jbr\bin\java
not found

Process finished with exit code 0
```

数组（续）

- 要检查数组是否包含某个元素，我们可以使用 in 关键字，或 *contains* 函数



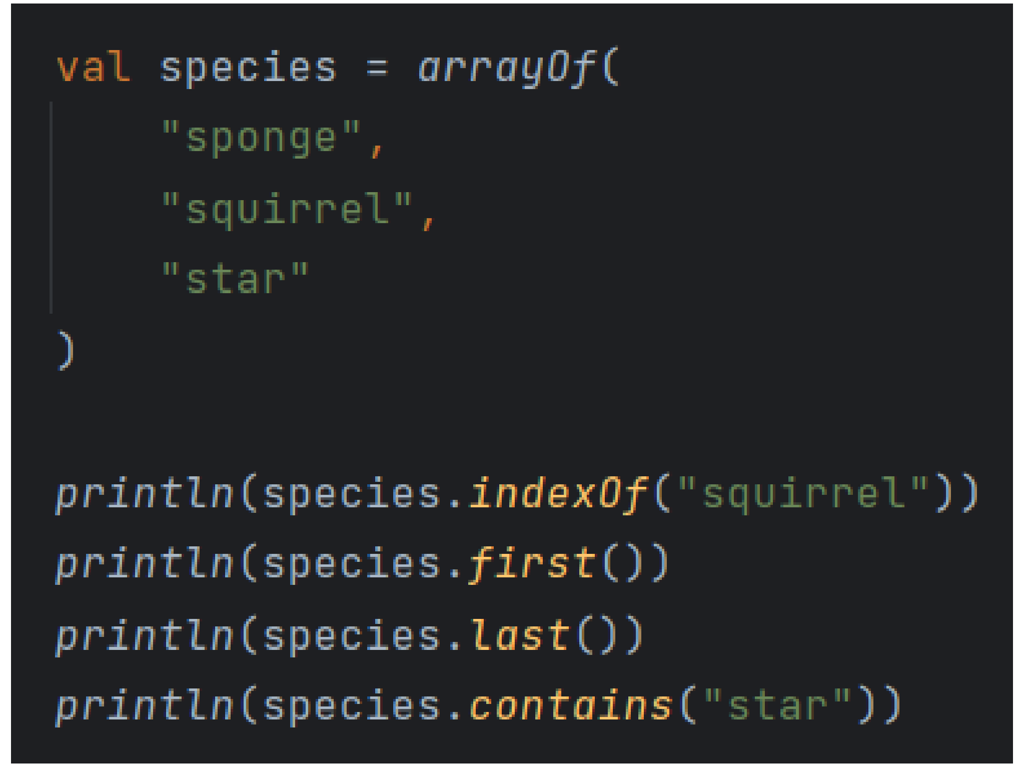
```
val species = arrayOf("sponge", "star", "snail")

val str = if("squirrel" in species) "found" else "not found"

println(str)
```



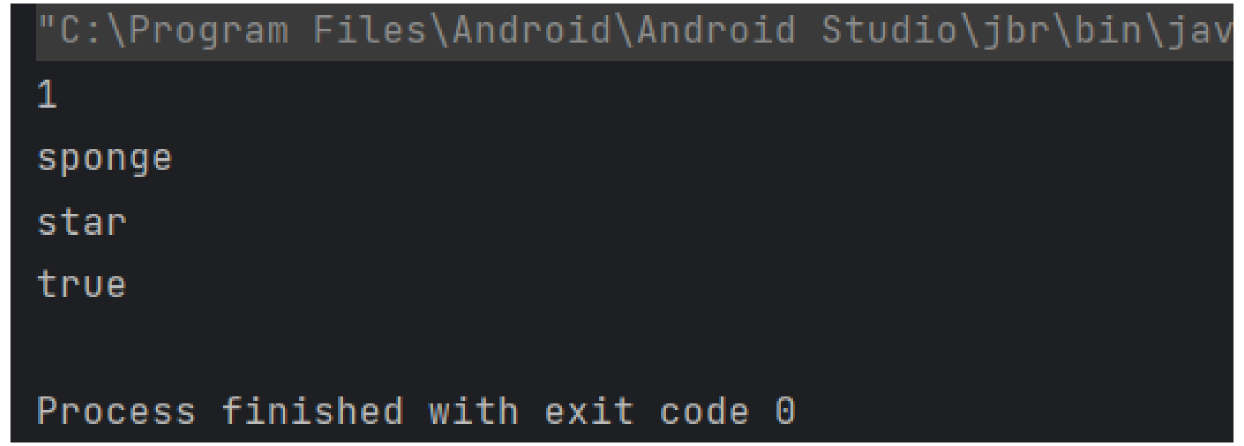
```
"C:\Program Files\Android\Android Studio\jbr\bin\java
not found

Process finished with exit code 0
```

Arrays (cont.)

- There are lots of helpful functions we can use with arrays



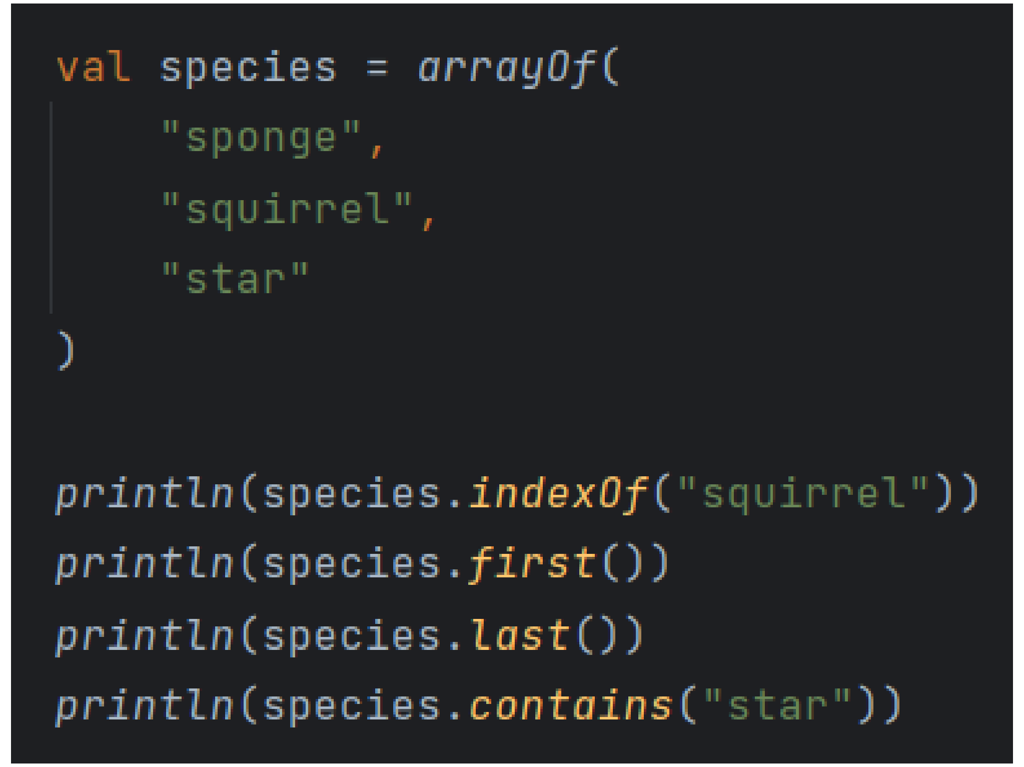
```
val species = arrayOf(
    "sponge",
    "squirrel",
    "star"
)

println(species.indexOf("squirrel"))
println(species.first())
println(species.last())
println(species.contains("star"))
```



```
"C:\Program Files\Android\Android Studio\jbr\bin\jav
1
sponge
star
true

Process finished with exit code 0
```

数组（续）

- 我们可以使用许多有助于处理数组的函数

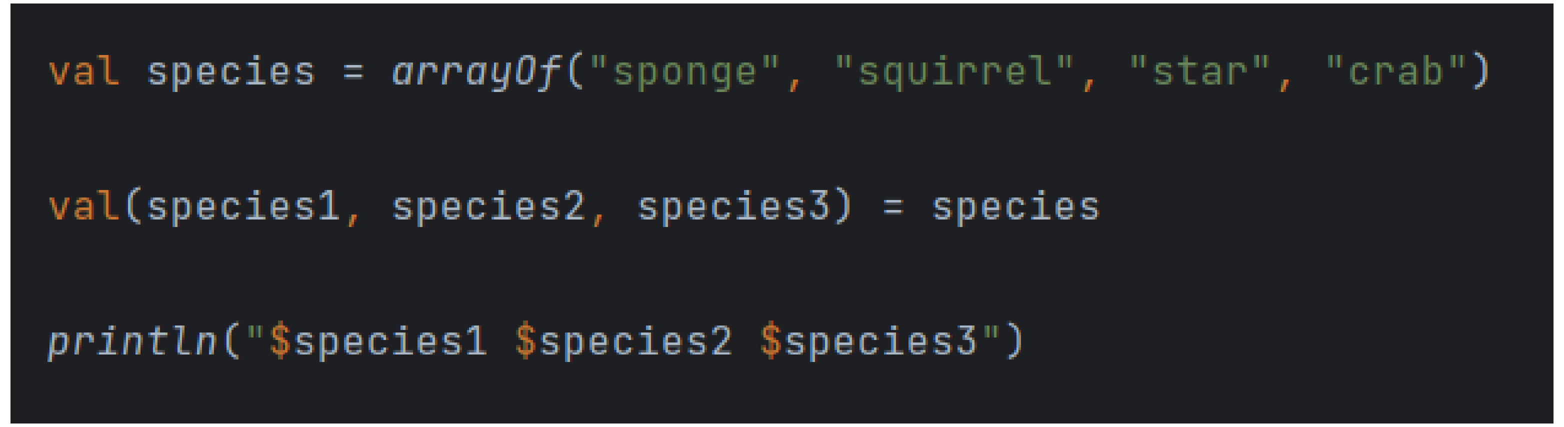


```
val species = arrayOf(
    "sponge",
    "squirrel",
    "star"
)

println(species.indexOf("squirrel"))
println(species.first())
println(species.last())
println(species.contains("star"))
```

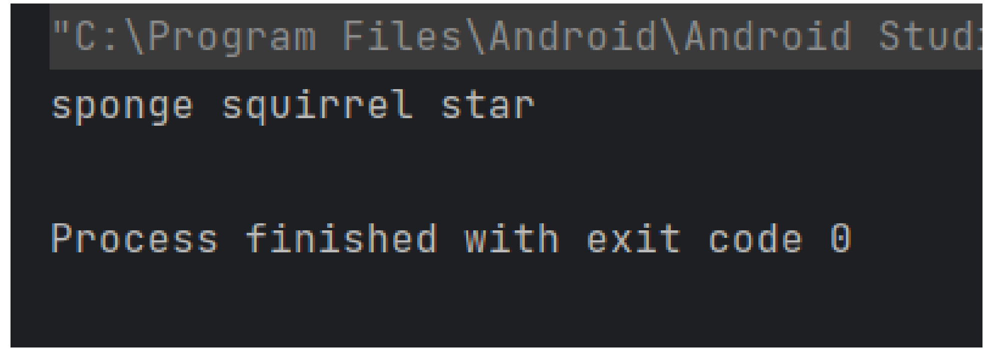
```
"C:\Program Files\Android\Android Studio\jbr\bin\jav
1
sponge
star
true

Process finished with exit code 0
```

Deconstructing an Array

- To separate elements in an array into separate variables, we can do



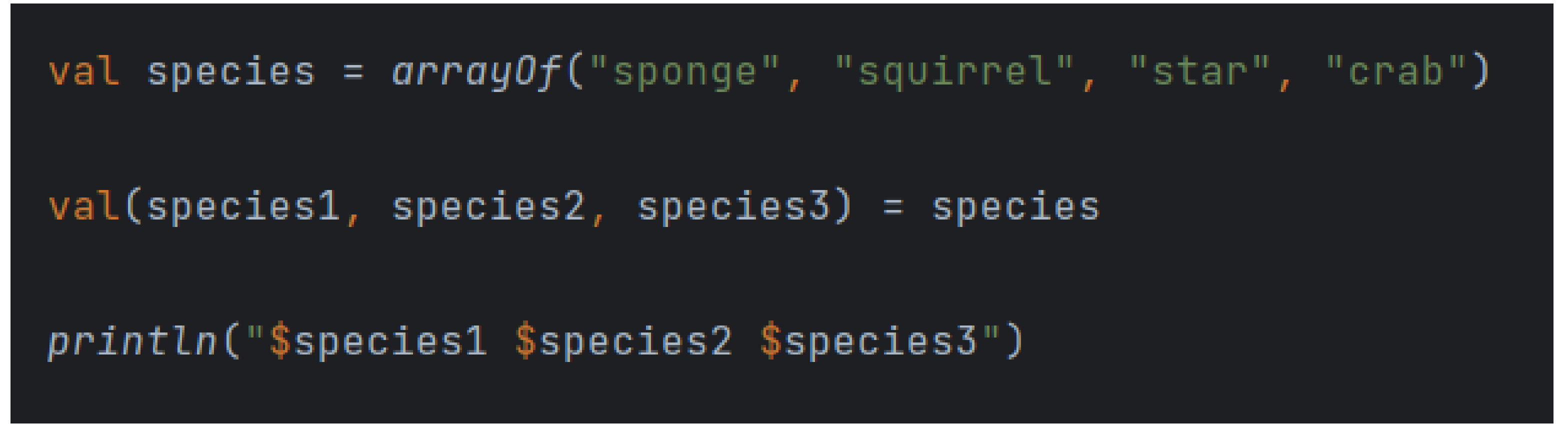
```
val species = arrayOf("sponge", "squirrel", "star", "crab")

val(species1, species2, species3) = species

println("$species1 $species2 $species3")
```



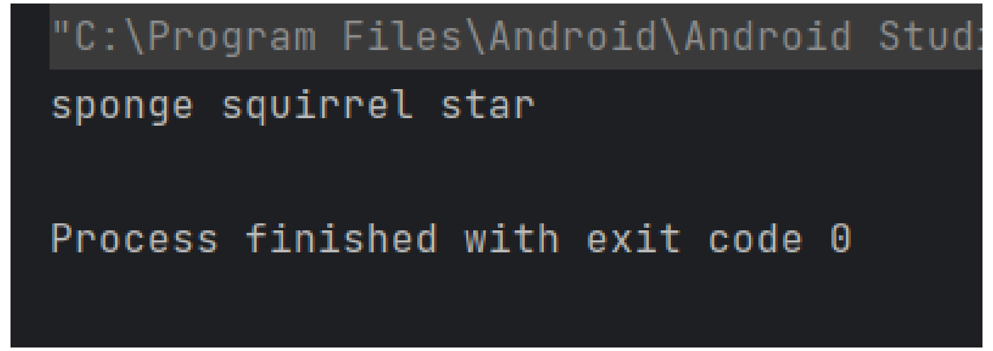
```
"C:\Program Files\Android\Android Stud
sponge squirrel star

Process finished with exit code 0
```

解构数组

- 要将数组中的元素拆分为单独的变量，我们可以这样做



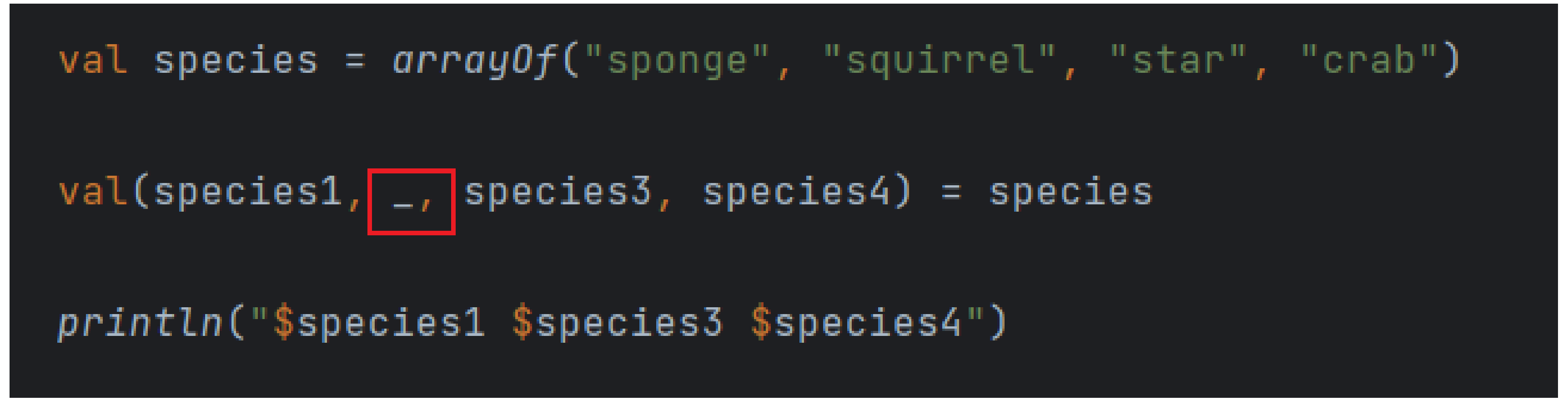
```
val species = arrayOf("sponge", "squirrel", "star", "crab")

val(species1, species2, species3) = species

println("$species1 $species2 $species3")
```



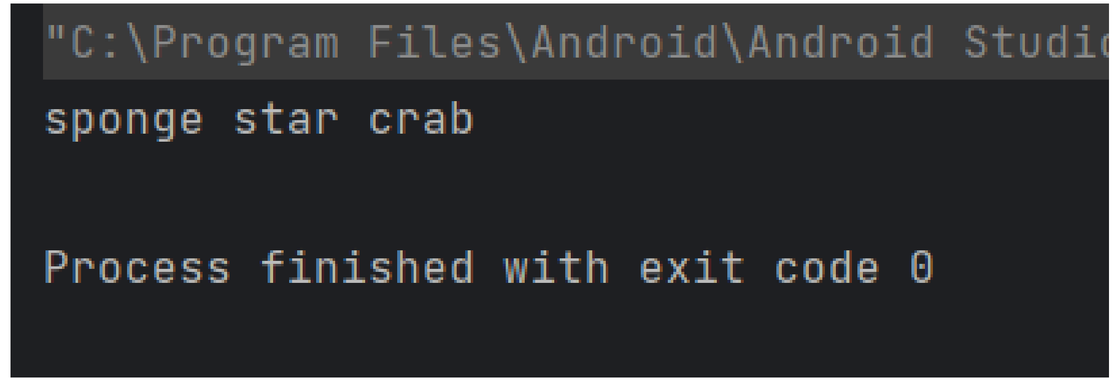
```
"C:\Program Files\Android\Android Stud
sponge squirrel star

Process finished with exit code 0
```

Deconstructing an Array (cont.)

- To omit certain variables we can use underscore _



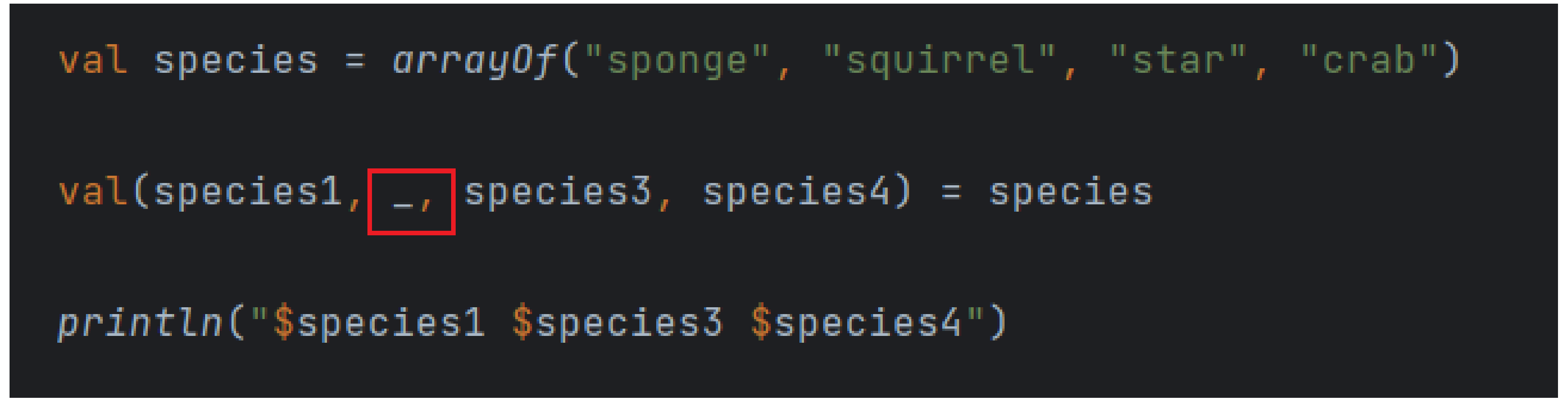
```
val species = arrayOf("sponge", "squirrel", "star", "crab")

val(species1, _, species3, species4) = species

println("$species1 $species3 $species4")
```



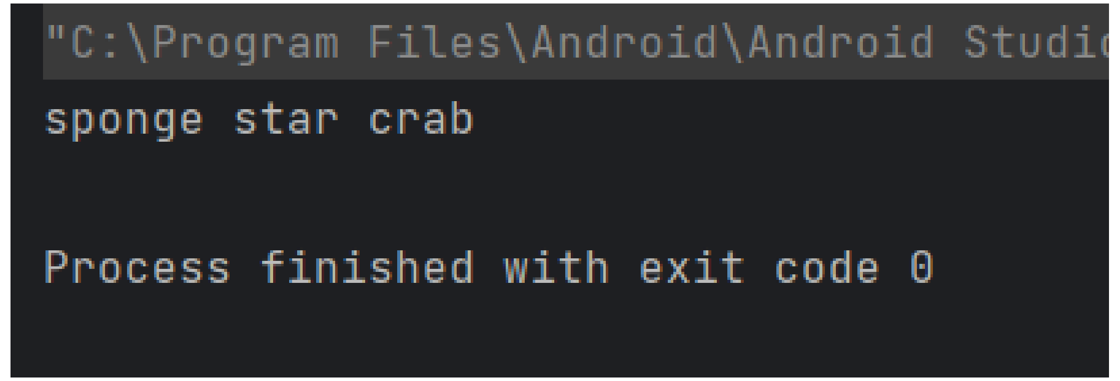
```
"C:\Program Files\Android\Android Studi
sponge star crab

Process finished with exit code 0
```

数组的解构（续）

- 要忽略某些变量，我们可以使用下划线 _



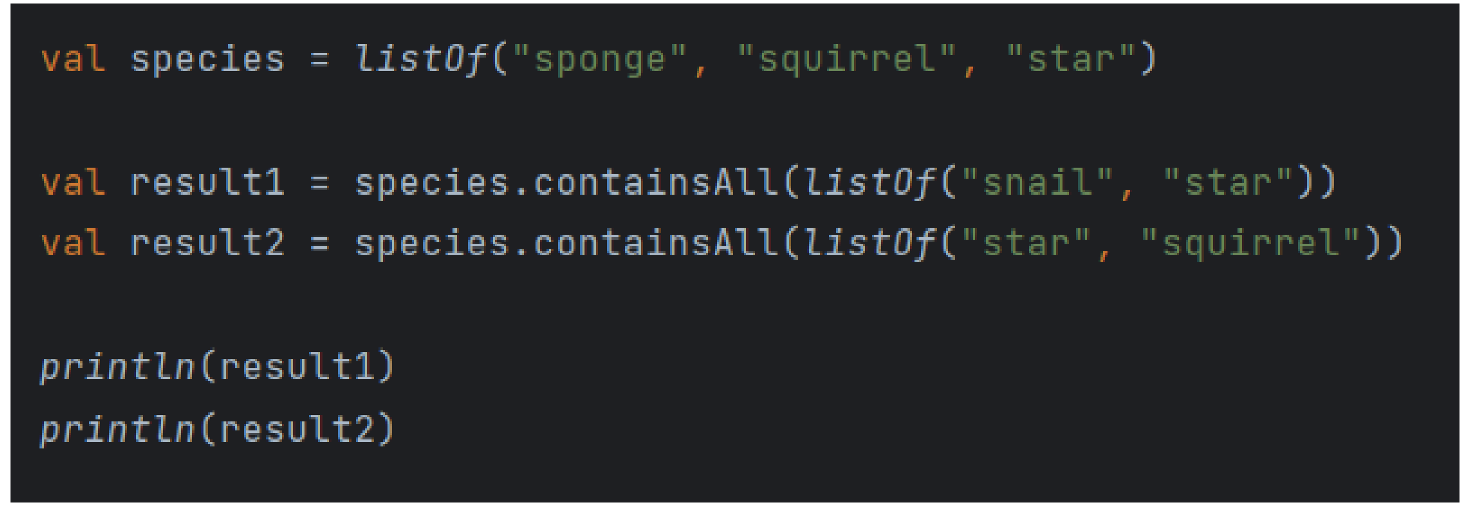
```
val species = arrayOf("sponge", "squirrel", "star", "crab")

val(species1, _, species3, species4) = species

println("$species1 $species3 $species4")
```



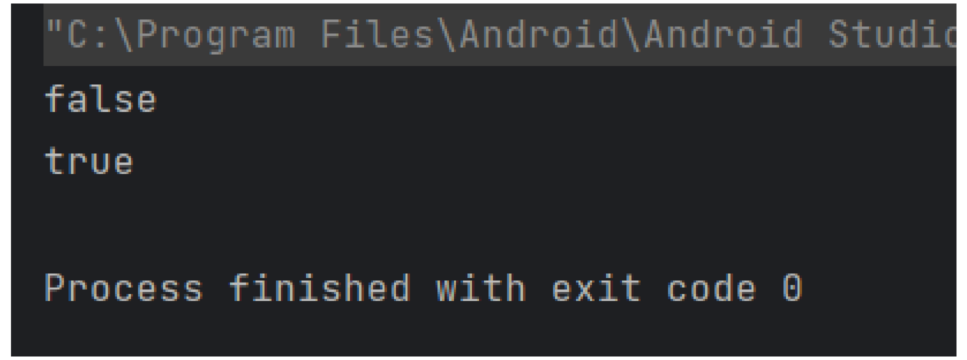
```
"C:\Program Files\Android\Android Studi
sponge star crab

Process finished with exit code 0
```

Lists

- Lists are built on top of arrays and provide more flexibility
- For instance, we can use more functions like *containsAll*



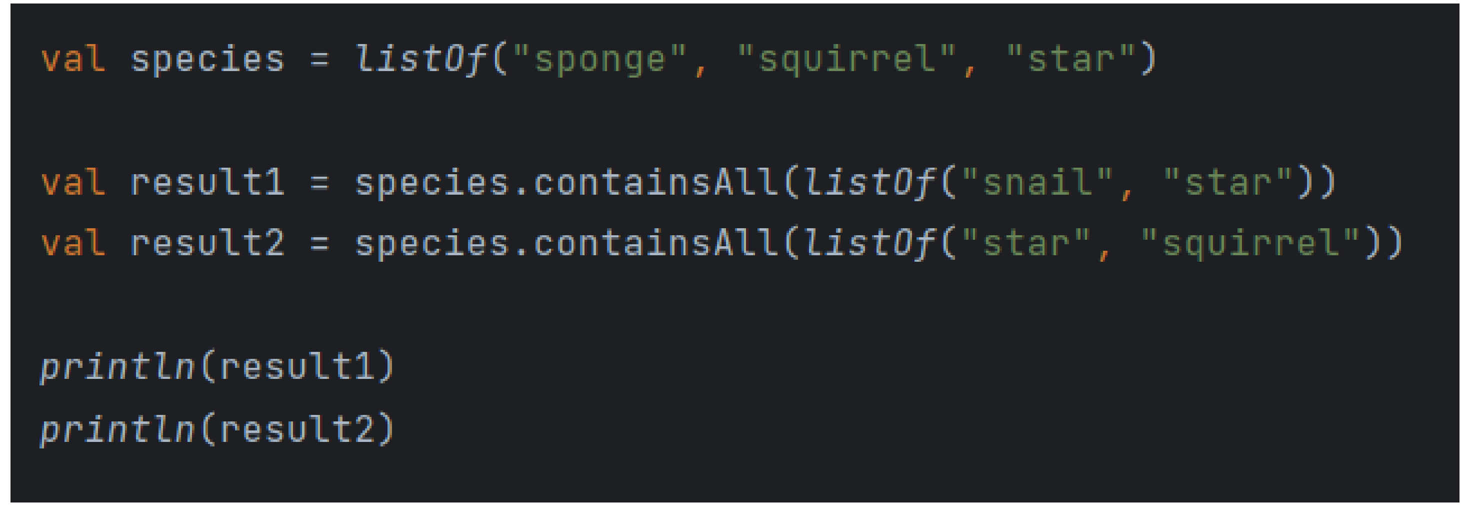
```kotlin
val species = listOf("sponge", "squirrel", "star")

val result1 = species.containsAll(listOf("snail", "star"))
val result2 = species.containsAll(listOf("star", "squirrel"))

println(result1)
println(result2)
```



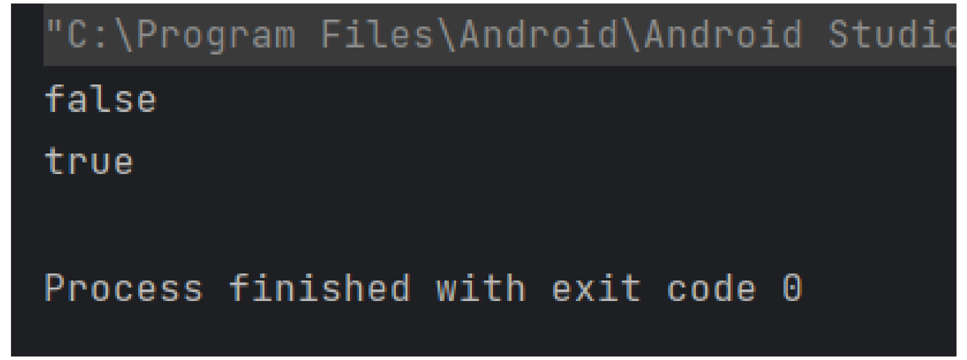
```
"C:\Program Files\Android\Android Studi
false
true

Process finished with exit code 0
```

列表

- 列表基于数组构建，并提供了更大的灵活性
- 例如 , 我们可以使用更多函数，如 *containsAll*



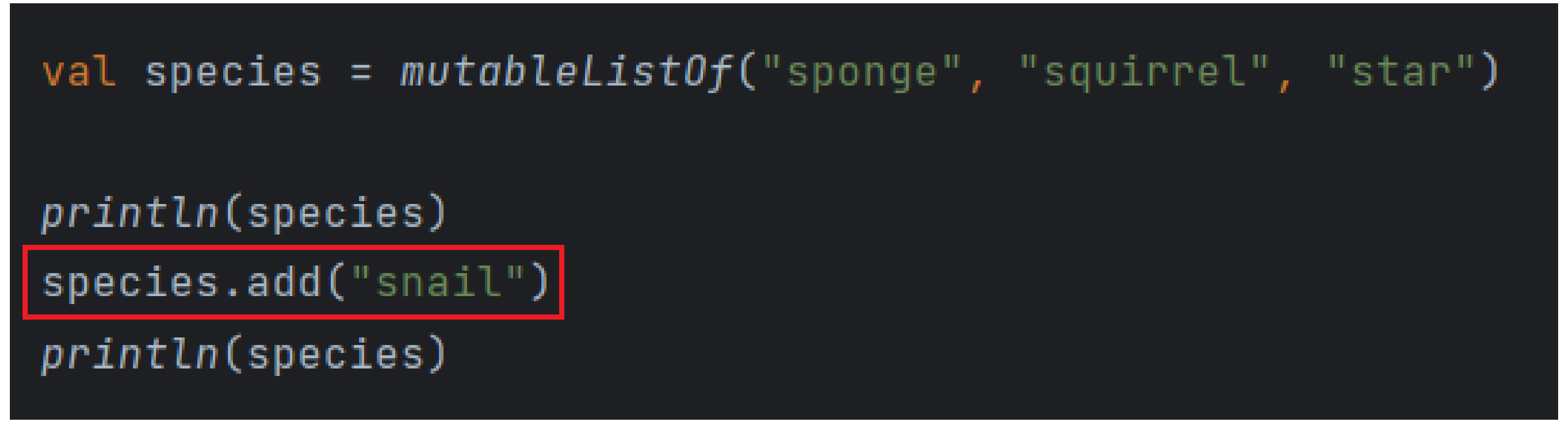
```kotlin
val species = listOf("sponge", "squirrel", "star")

val result1 = species.containsAll(listOf("snail", "star"))
val result2 = species.containsAll(listOf("star", "squirrel"))

println(result1)
println(result2)
```



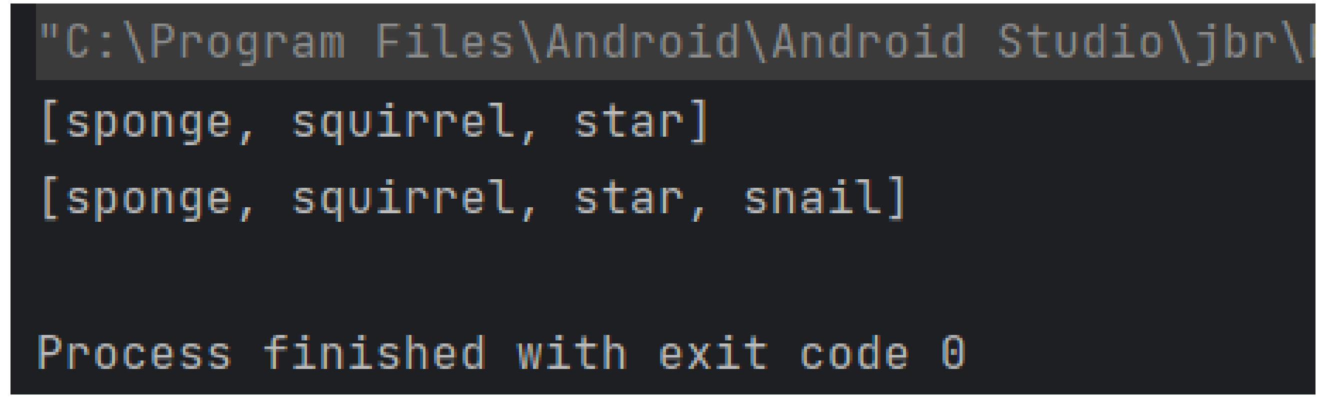
```
"C:\Program Files\Android\Android Studi
false
true

Process finished with exit code 0
```

Mutable lists

- Lists can be resized where as arrays cannot
  - Keep in mind that the *listOf* function is read-only, so we use *mutableListOf*



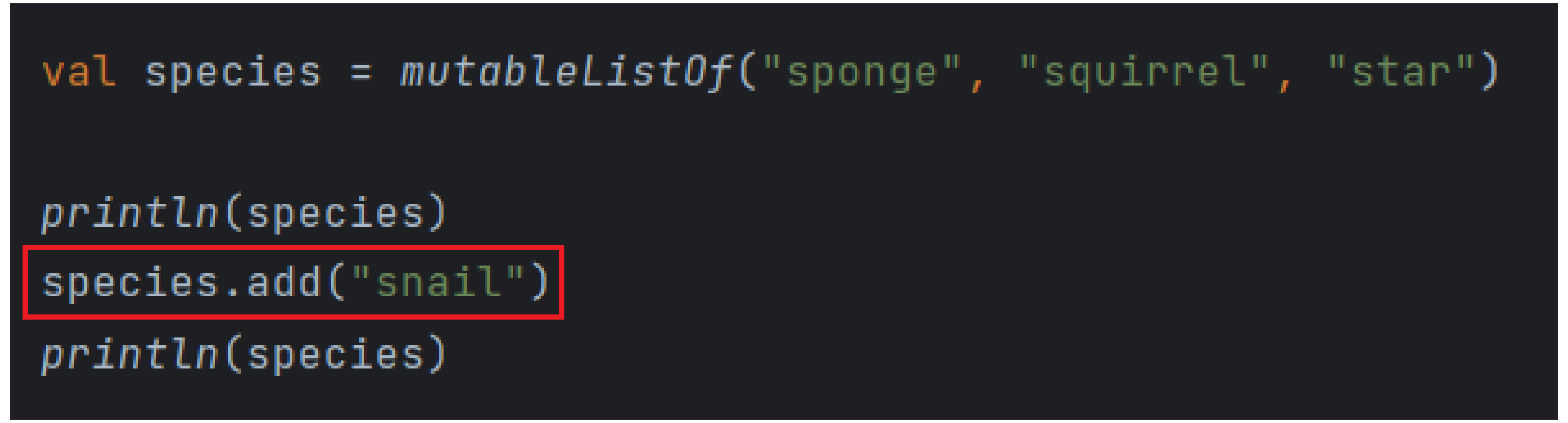
```
val species = mutableListOf("sponge", "squirrel", "star")

println(species)
species.add("snail")
println(species)
```



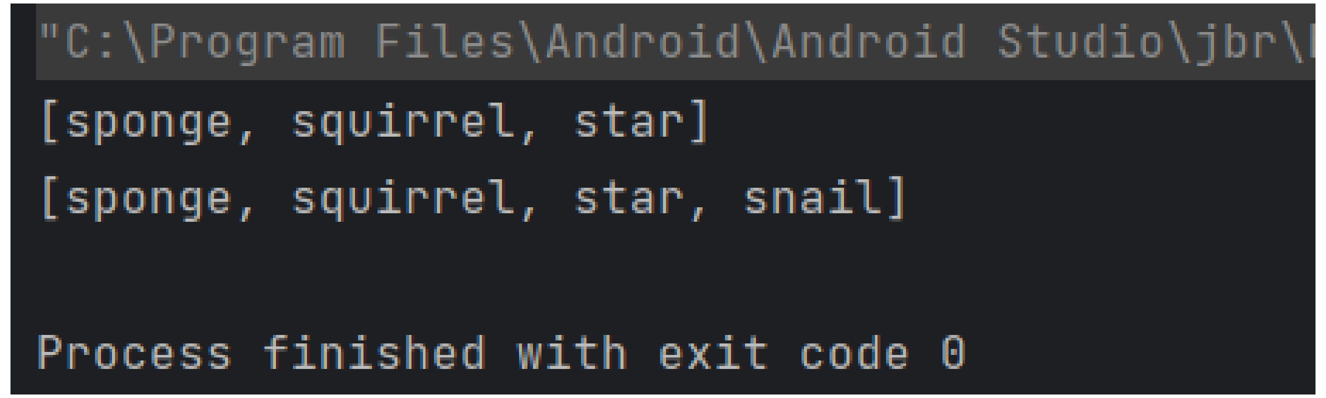
```
"C:\Program Files\Android\Android Studio\jbr\
[sponge, squirrel, star]
[sponge, squirrel, star, snail]

Process finished with exit code 0
```

可变列表

- 列表可以 调整大小，而数组不能
  - 请注意，*listOf* 函数是只读的，因此我们使用 *mutableListOf*



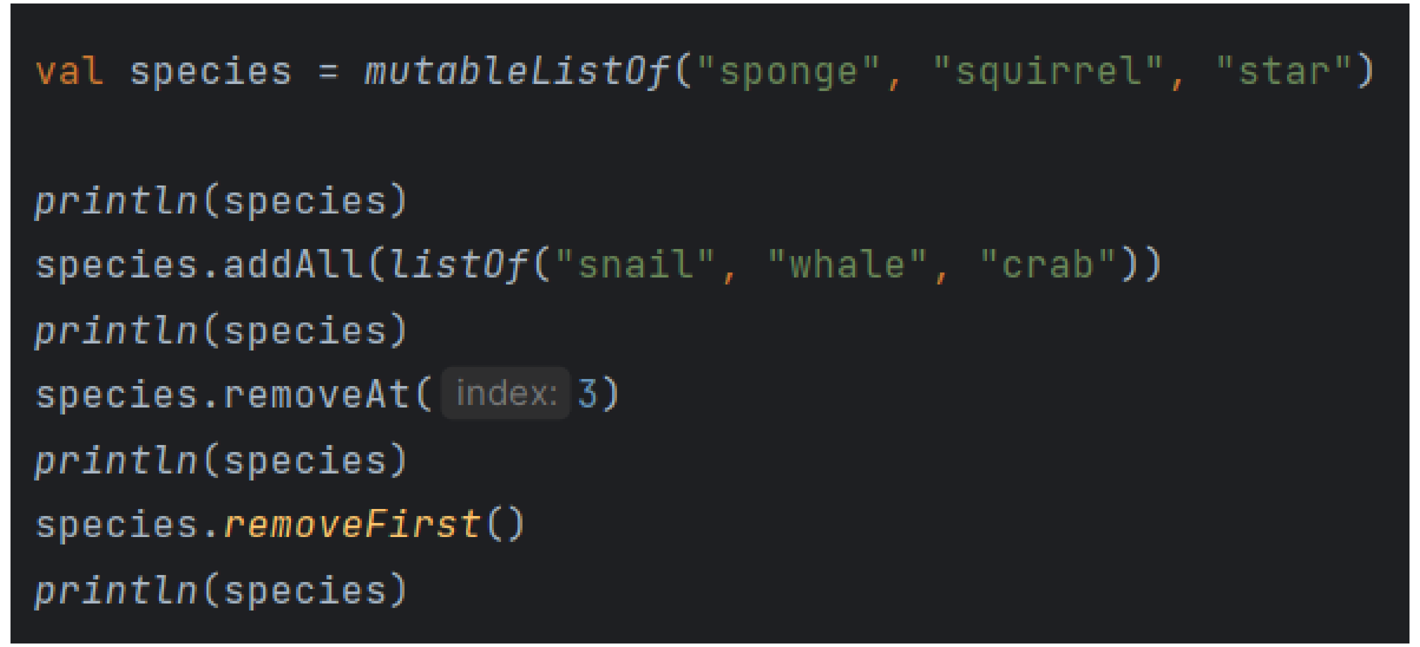
```
val species = mutableListOf("sponge", "squirrel", "star")

println(species)
species.add("snail")
println(species)
```



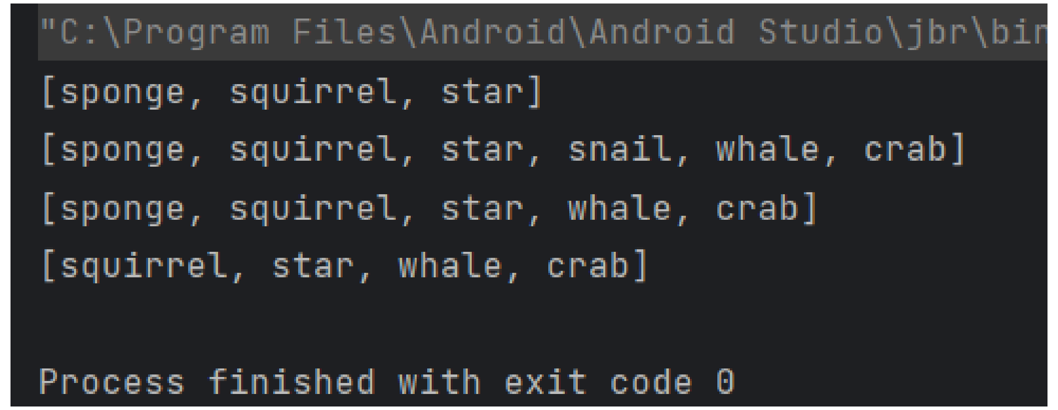
```
"C:\Program Files\Android\Android Studio\jbr\
[sponge, squirrel, star]
[sponge, squirrel, star, snail]

Process finished with exit code 0
```

Mutable lists (cont.)

- There are lots of ways to resize a mutable list



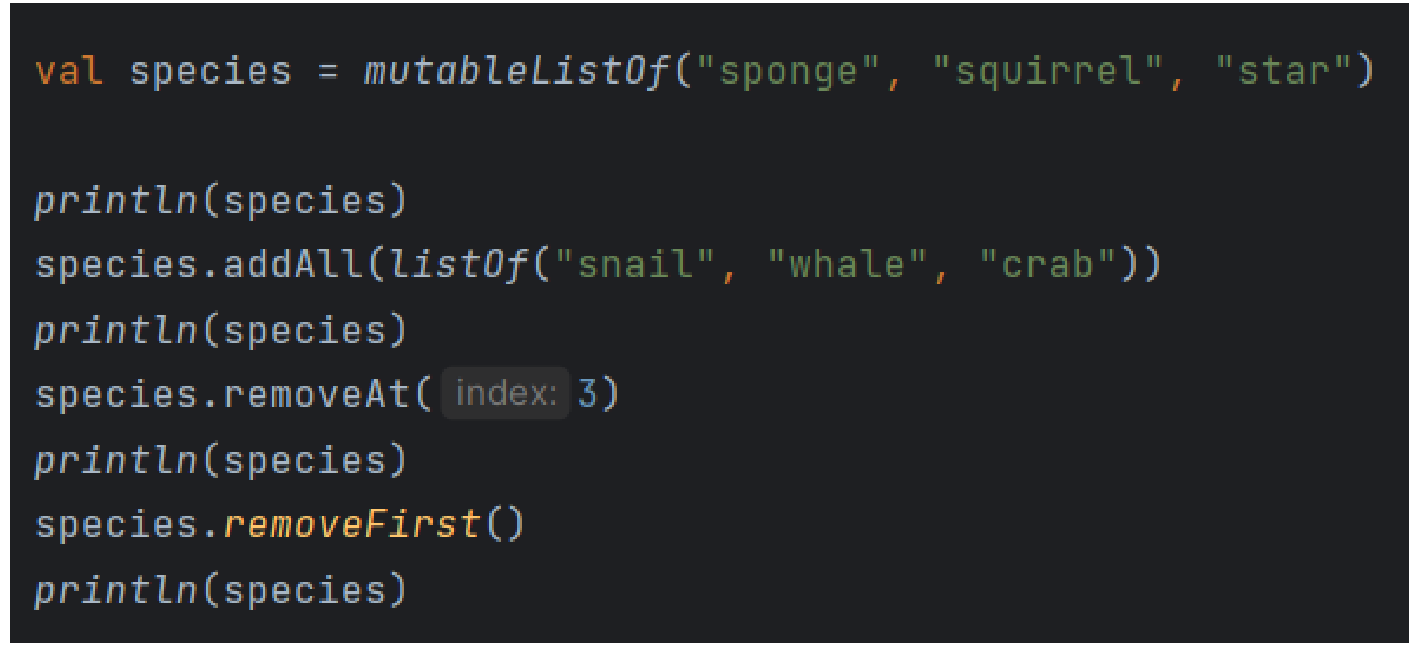
```kotlin
val species = mutableListOf("sponge", "squirrel", "star")

println(species)
species.addAll(listOf("snail", "whale", "crab"))
println(species)
species.removeAt( index: 3)
println(species)
species.removeFirst()
println(species)
```



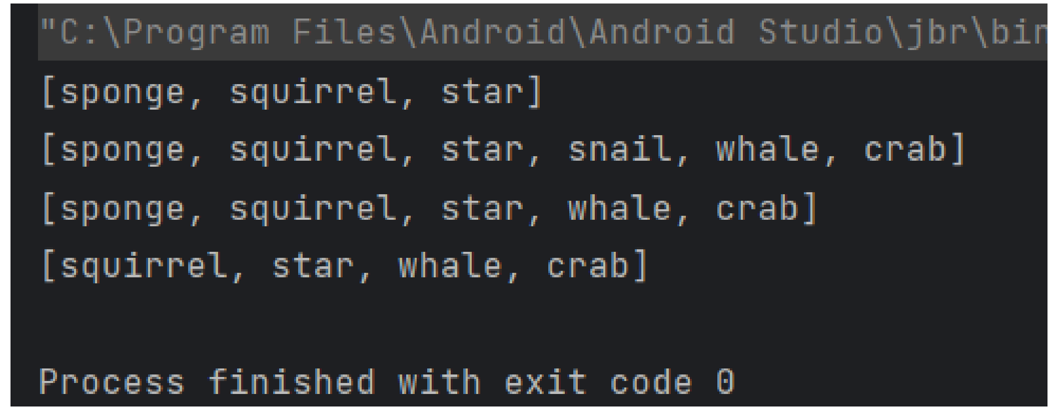
```
"C:\Program Files\Android\Android Studio\jbr\bi
[sponge, squirrel, star]
[sponge, squirrel, star, snail, whale, crab]
[sponge, squirrel, star, whale, crab]
[squirrel, star, whale, crab]

Process finished with exit code 0
```

可变列表（续）

- 有多种方法可以调整可变列表的大小



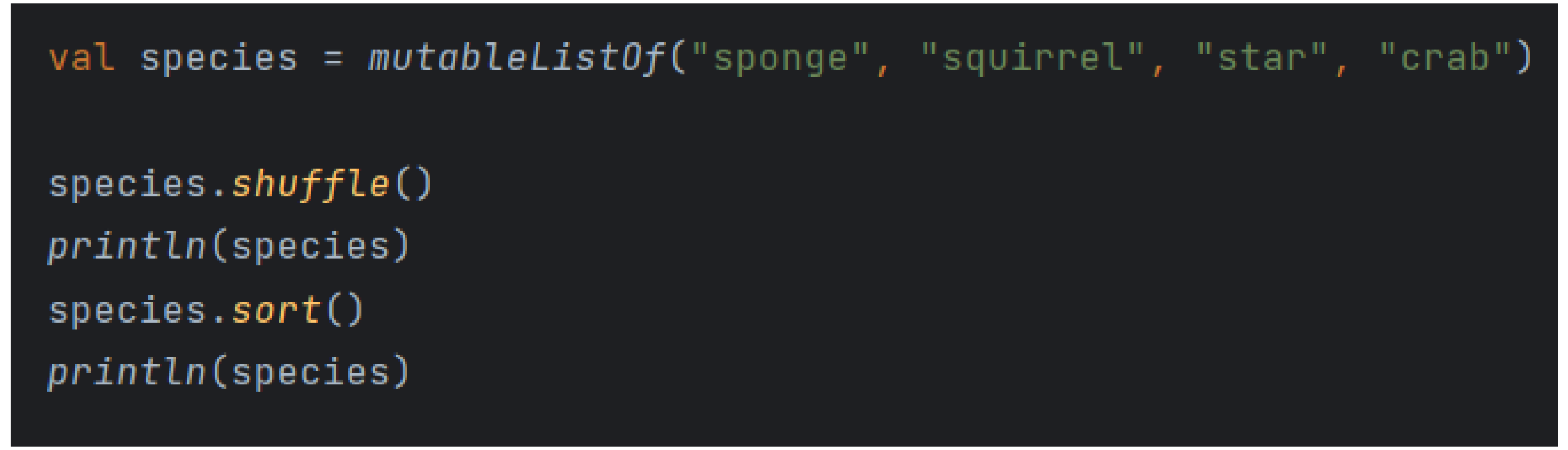
```kotlin
val species = mutableListOf("sponge", "squirrel", "star")

println(species)
species.addAll(listOf("snail", "whale", "crab"))
println(species)
species.removeAt( index: 3)
println(species)
species.removeFirst()
println(species)
```



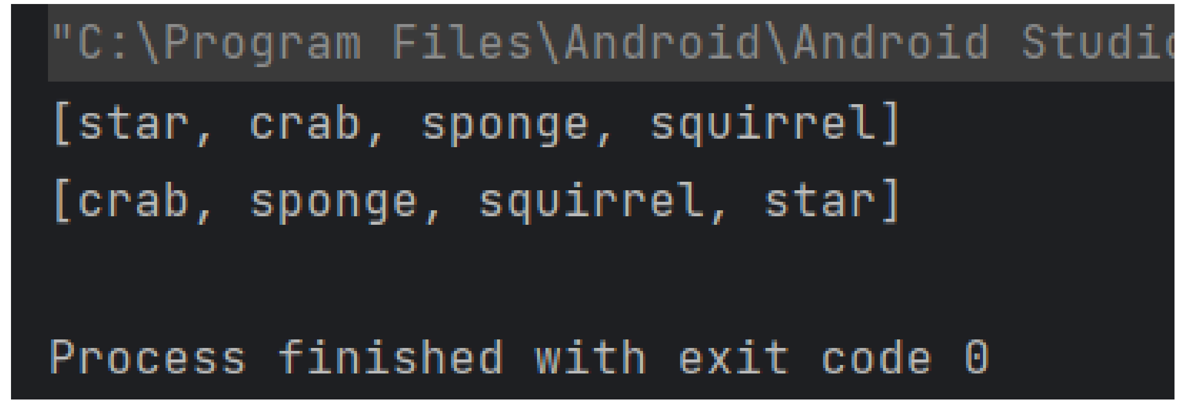
```
"C:\Program Files\Android\Android Studio\jbr\bi
[sponge, squirrel, star]
[sponge, squirrel, star, snail, whale, crab]
[sponge, squirrel, star, whale, crab]
[squirrel, star, whale, crab]

Process finished with exit code 0
```

Mutable lists (cont.)

- Mutable lists also have helpful functions like *sort* and *shuffle*



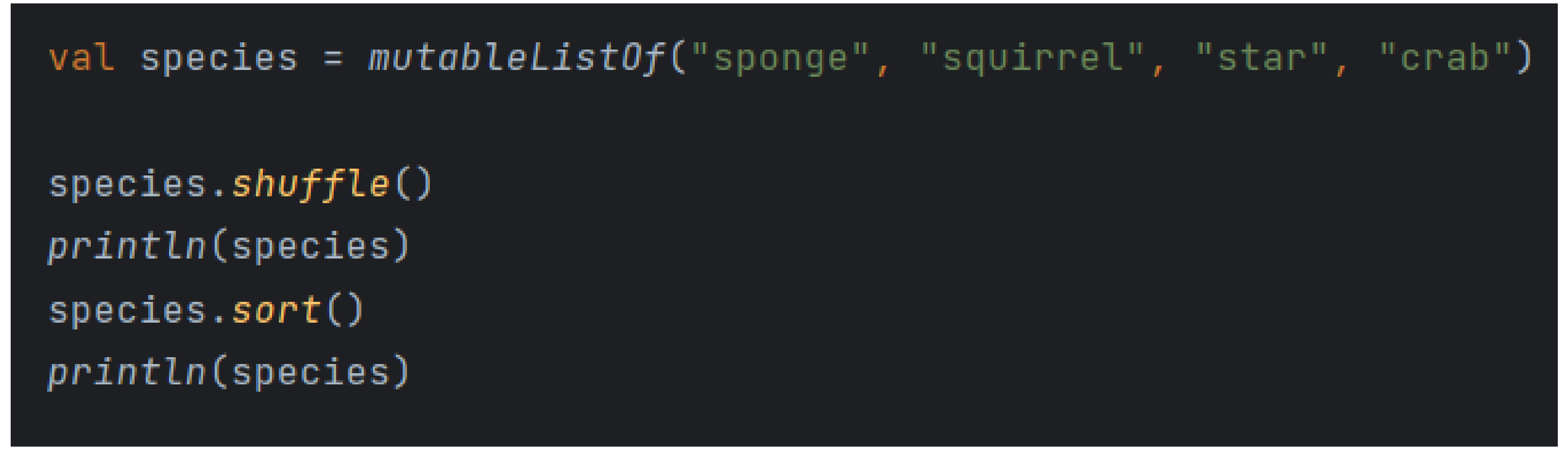
```kotlin
val species = mutableListOf("sponge", "squirrel", "star", "crab")

species.shuffle()
println(species)
species.sort()
println(species)
```



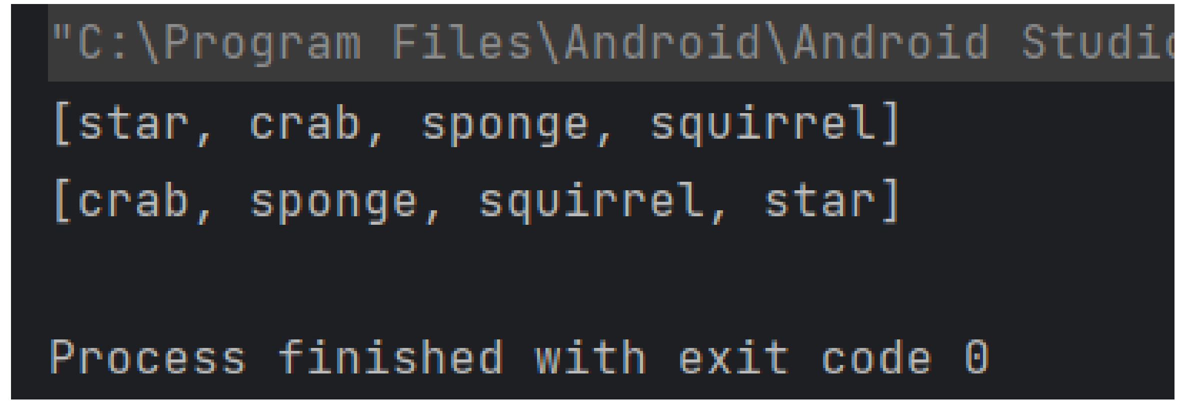
```
"C:\Program Files\Android\Android Studi
[star, crab, sponge, squirrel]
[crab, sponge, squirrel, star]

Process finished with exit code 0
```

可变列表（续）

- 可变列表还拥有一些有用的功能，例如 *sort* 和 *shuffle*



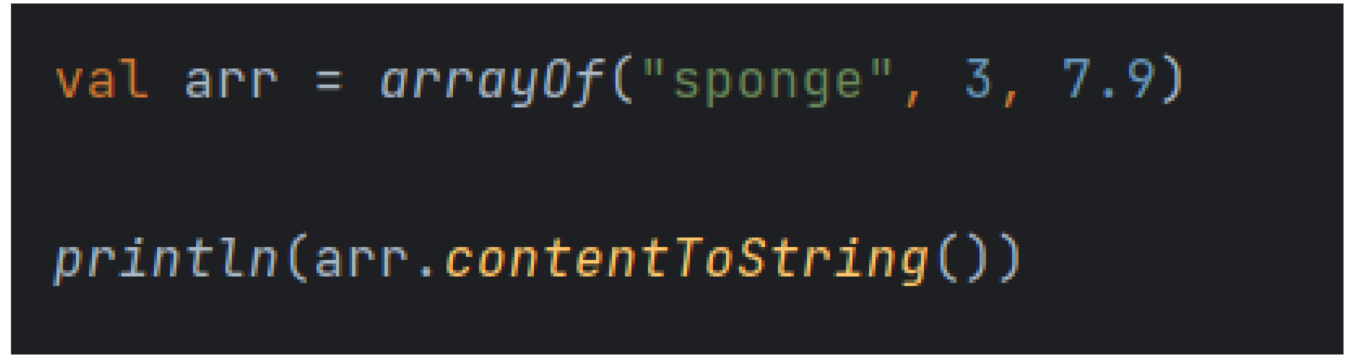
```kotlin
val species = mutableListOf("sponge", "squirrel", "star", "crab")

species.shuffle()
println(species)
species.sort()
println(species)
```



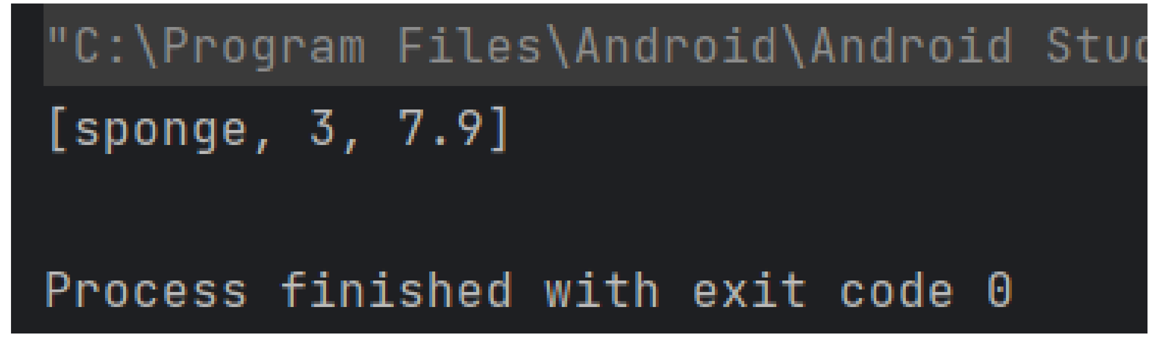
```
"C:\Program Files\Android\Android Studi
[star, crab, sponge, squirrel]
[crab, sponge, squirrel, star]

Process finished with exit code 0
```

Arrays and Lists

- When working with different collections it is possible to mix data types



```kotlin
val arr = arrayOf("sponge", 3, 7.9)

println(arr.contentToString())
```

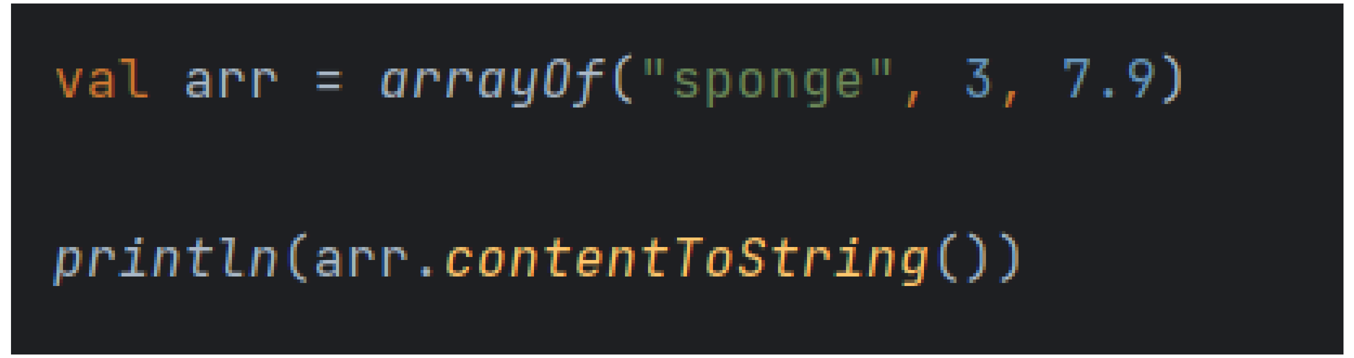


```
"C:\Program Files\Android\Android Stu
[sponge, 3, 7.9]

Process finished with exit code 0
```

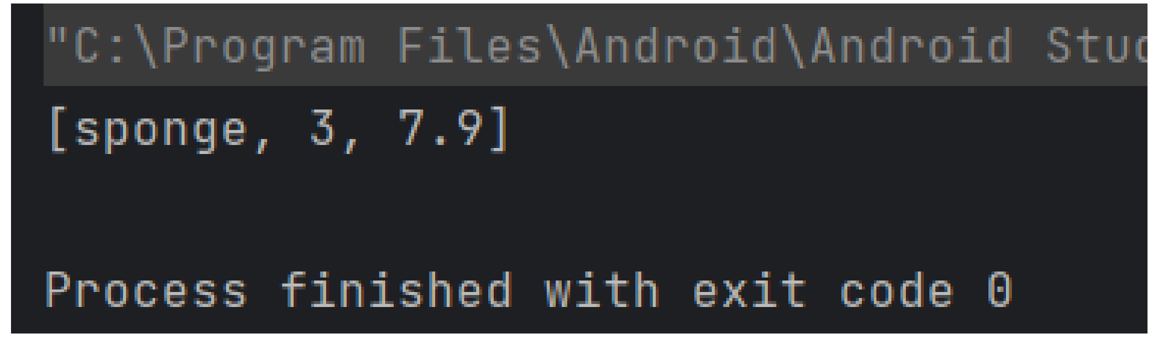
- Sometimes this may seem tempting but <u>try to avoid if possible!</u>

数组和列表

- 在处理不同的集合时，可以混合使用数据类型



```kotlin
val arr = arrayOf("sponge", 3, 7.9)

println(arr.contentToString())
```

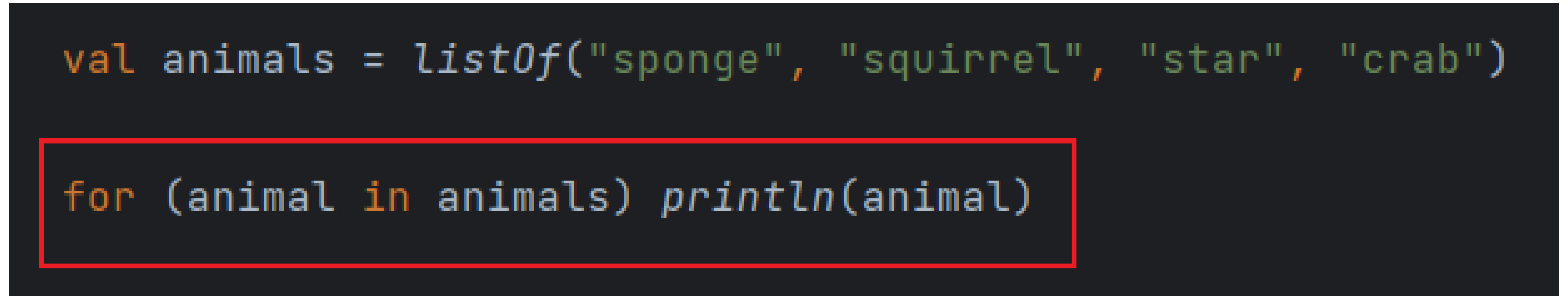


```
"C:\Program Files\Android\Android Stu
[sponge, 3, 7.9]

Process finished with exit code 0
```

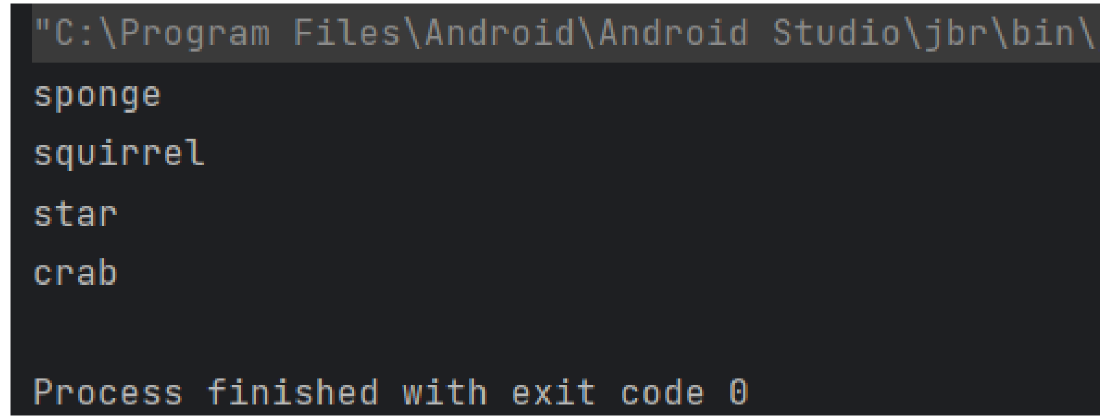
- 有时这可能看起来很诱人，但应尽量避免！

For loop

- Here we are looping through a collection using a classic for loop
  - If there is only one line in the block, we can put everything on one line



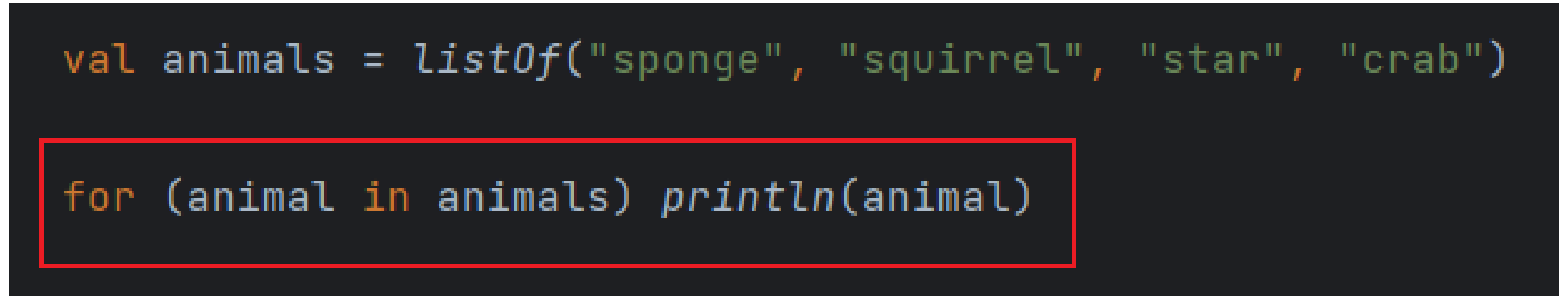
```
val animals = listOf("sponge", "squirrel", "star", "crab")

for (animal in animals) println(animal)
```



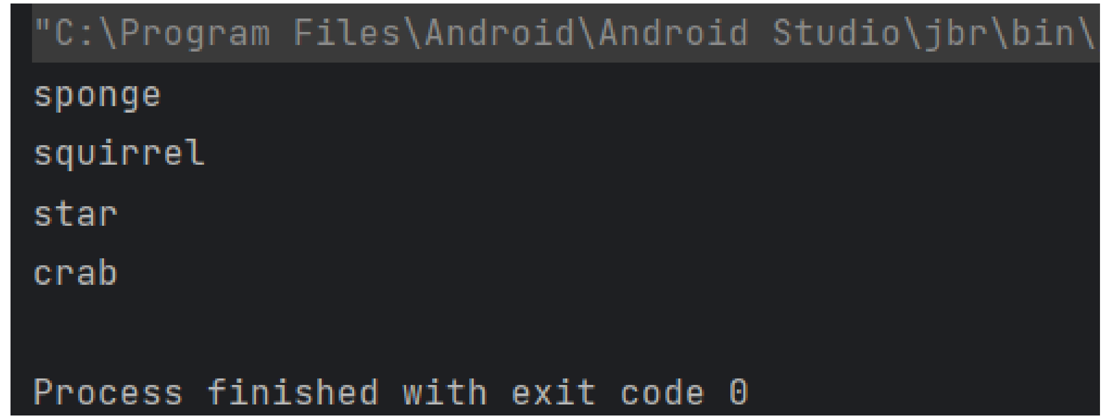
```
"C:\Program Files\Android\Android Studio\jbr\bin\
sponge
squirrel
star
crab

Process finished with exit code 0
```

for循环

- 这里我们使用经典的for循环遍历一个集合
  - 如果代码块中只有一行，我们可以将所有内容放在一行上



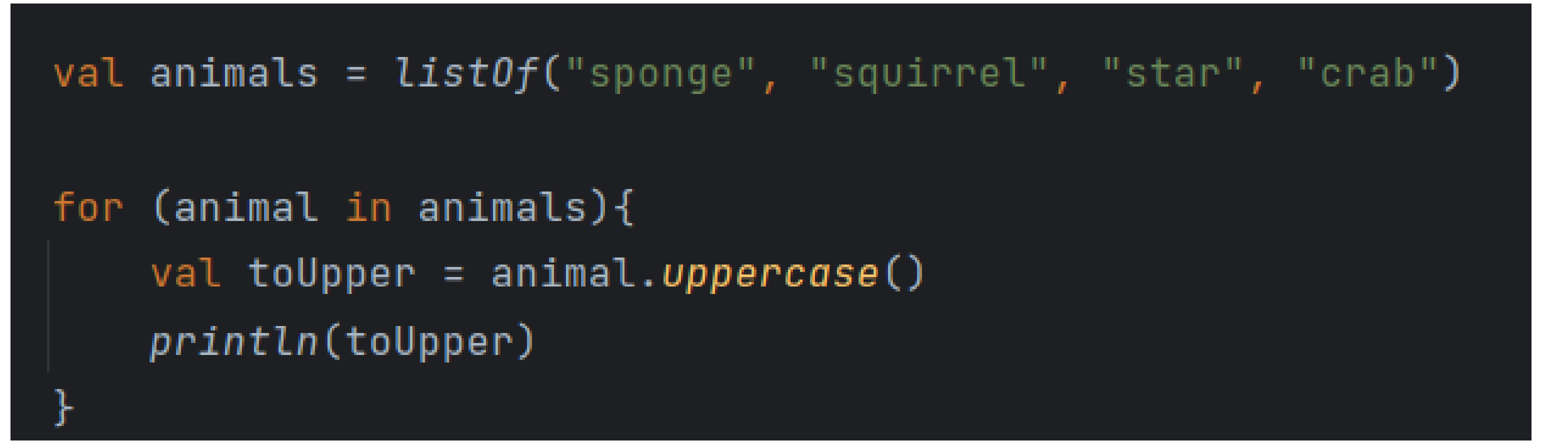
```
val animals = listOf("sponge", "squirrel", "star", "crab")

for (animal in animals) println(animal)
```



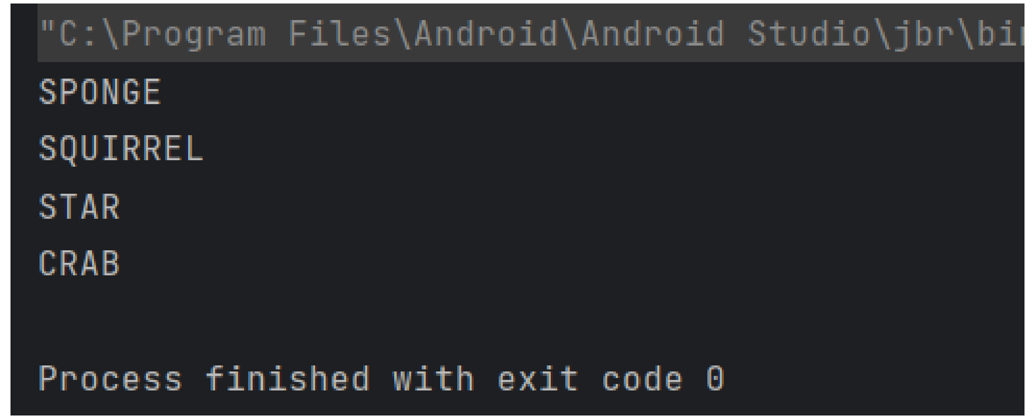
```
"C:\Program Files\Android\Android Studio\jbr\bin\
sponge
squirrel
star
crab

Process finished with exit code 0
```

For loop (cont.)

- We can also perform operations on each index in the collection



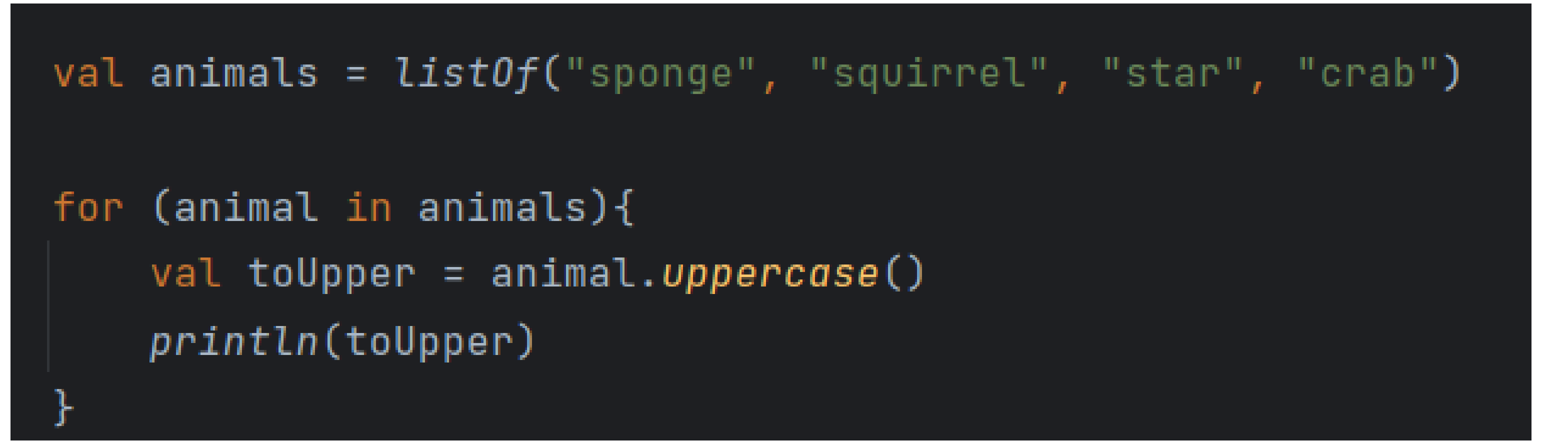
```kotlin
val animals = listOf("sponge", "squirrel", "star", "crab")

for (animal in animals){
    val toUpper = animal.uppercase()
    println(toUpper)
}
```



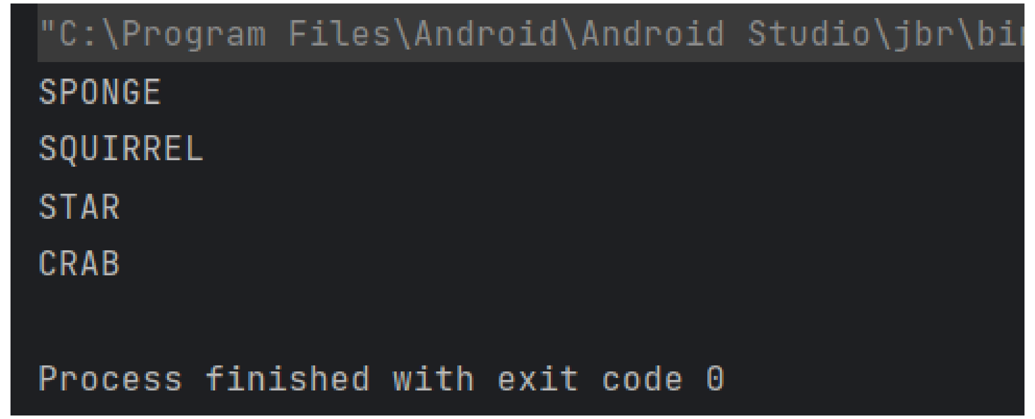
```
"C:\Program Files\Android\Android Studio\jbr\bi
SPONGE
SQUIRREL
STAR
CRAB

Process finished with exit code 0
```

for循环（续）

- 我们还可以对集合中的每个索引执行操作



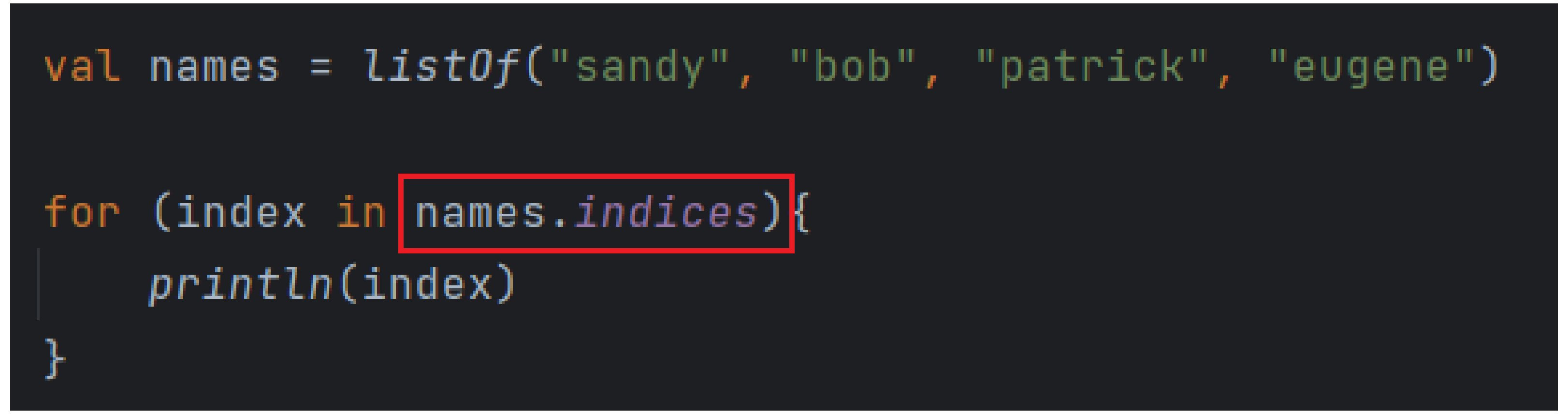
```kotlin
val animals = listOf("sponge", "squirrel", "star", "crab")

for (animal in animals){
    val toUpper = animal.uppercase()
    println(toUpper)
}
```



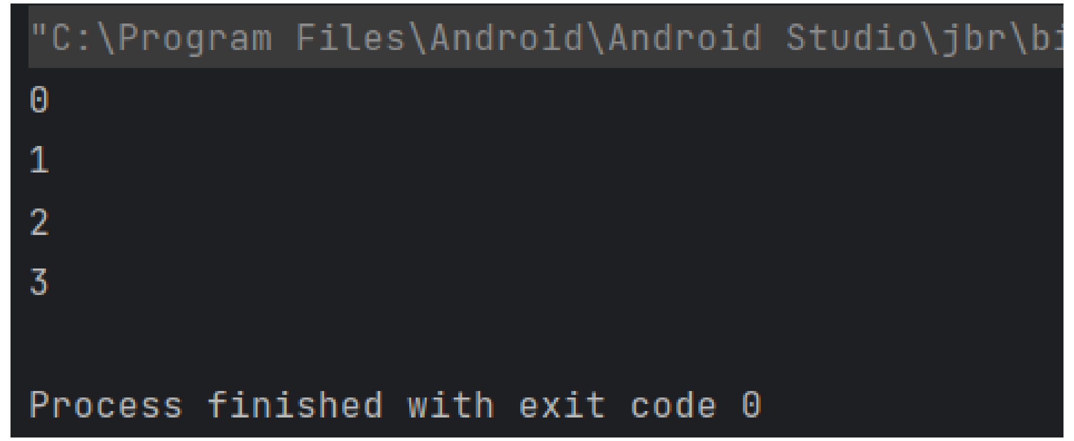
```
"C:\Program Files\Android\Android Studio\jbr\bi
SPONGE
SQUIRREL
STAR
CRAB

Process finished with exit code 0
```

For loop (cont.)

- We can also loop the indices rather than the values



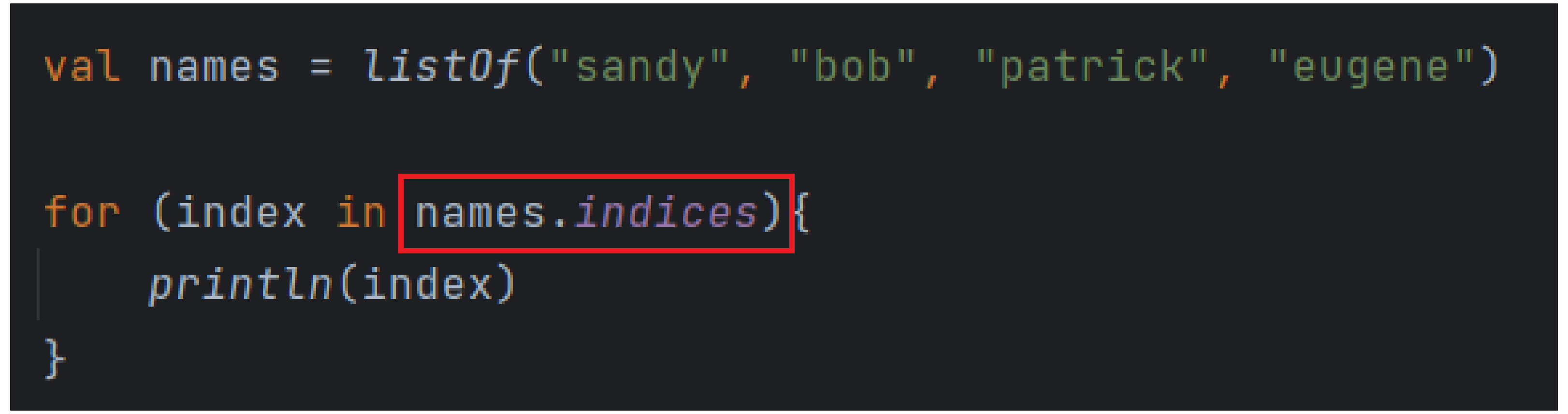
```kotlin
val names = listOf("sandy", "bob", "patrick", "eugene")

for (index in names.indices){
    println(index)
}
```



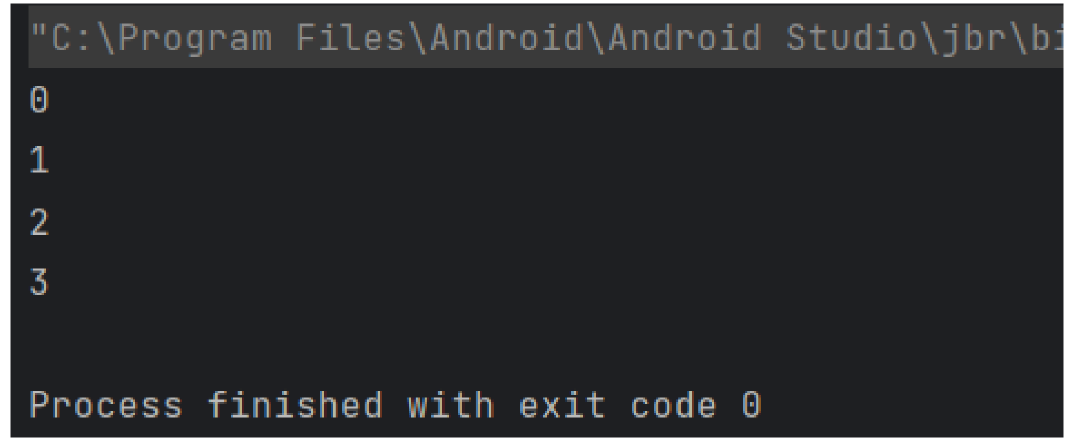
```
"C:\Program Files\Android\Android Studio\jbr\bi
0
1
2
3

Process finished with exit code 0
```

for循环（续）

- 我们也可以遍历索引而不是值



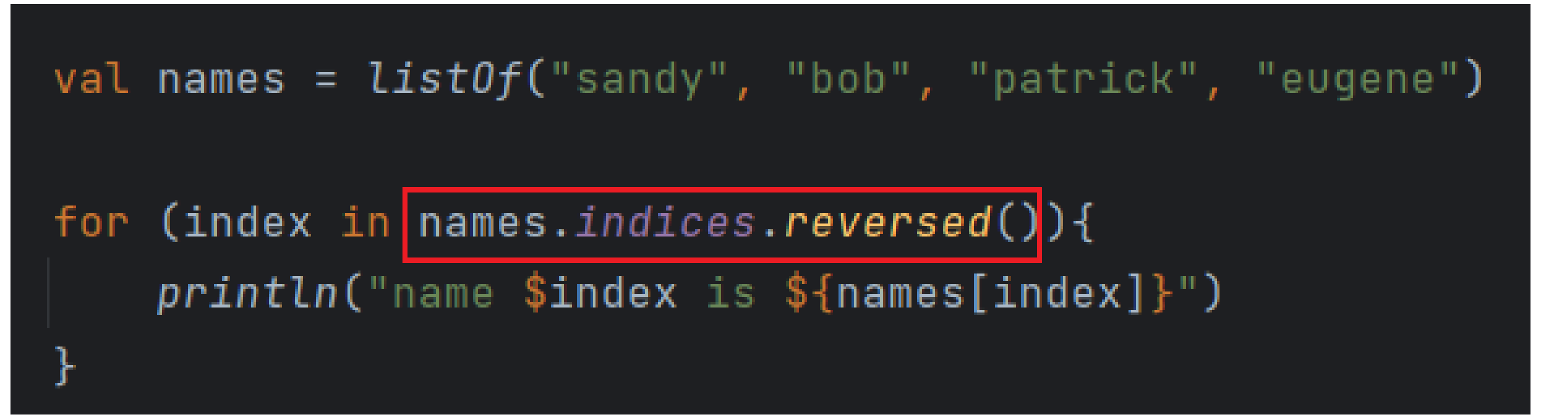
```kotlin
val names = listOf("sandy", "bob", "patrick", "eugene")

for (index in names.indices){
    println(index)
}
```



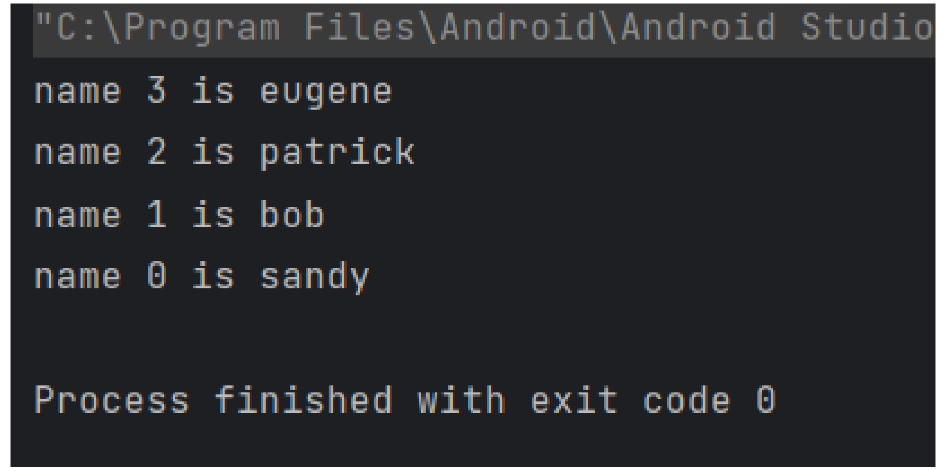
```
"C:\Program Files\Android\Android Studio\jbr\bi
0
1
2
3

Process finished with exit code 0
```

For loop (cont.)

• When looping the indices we can do it in reverse



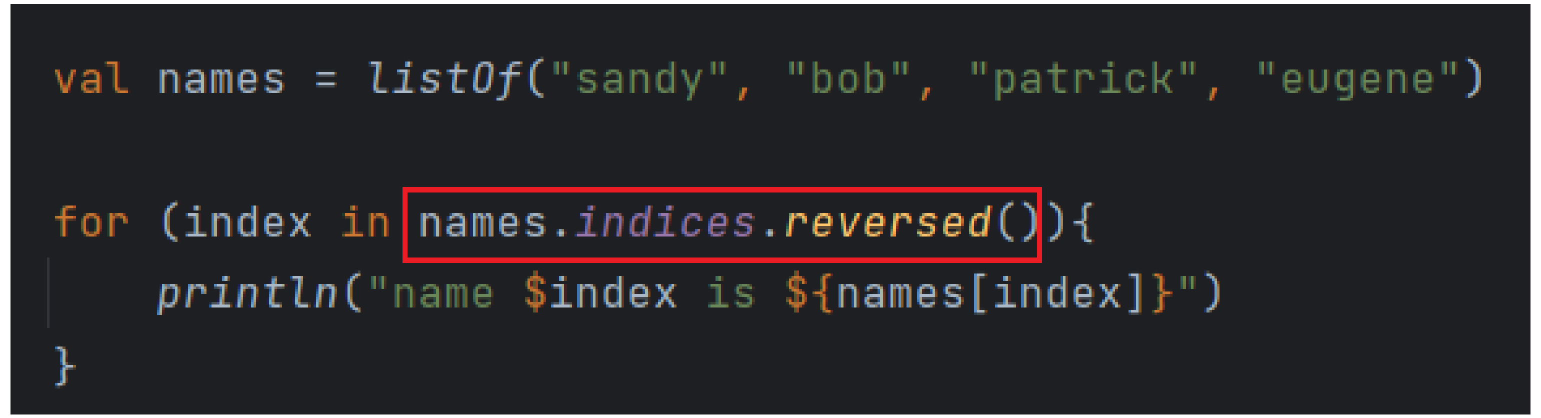
```
val names = listOf("sandy", "bob", "patrick", "eugene")

for (index in names.indices.reversed()){
    println("name $index is ${names[index]}")
}
```



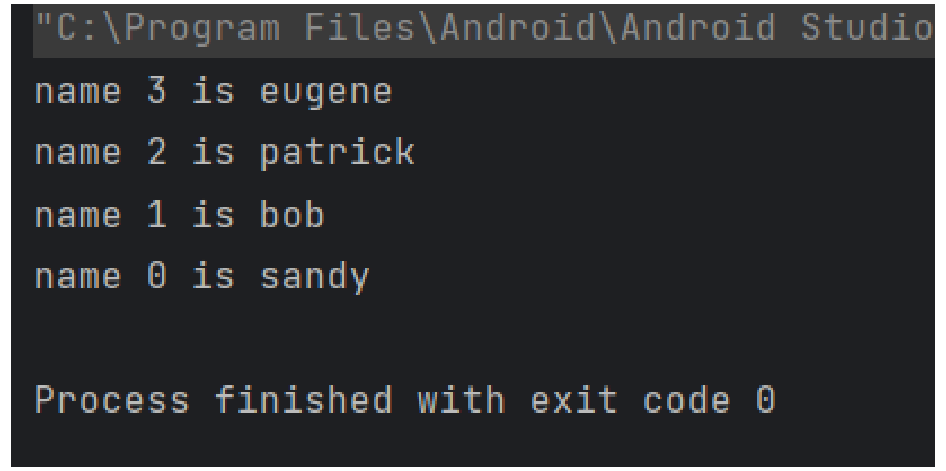
```
"C:\Program Files\Android\Android Studio
name 3 is eugene
name 2 is patrick
name 1 is bob
name 0 is sandy

Process finished with exit code 0
```

for循环（续）

• 遍历索引时，我们可以 逆序 进行



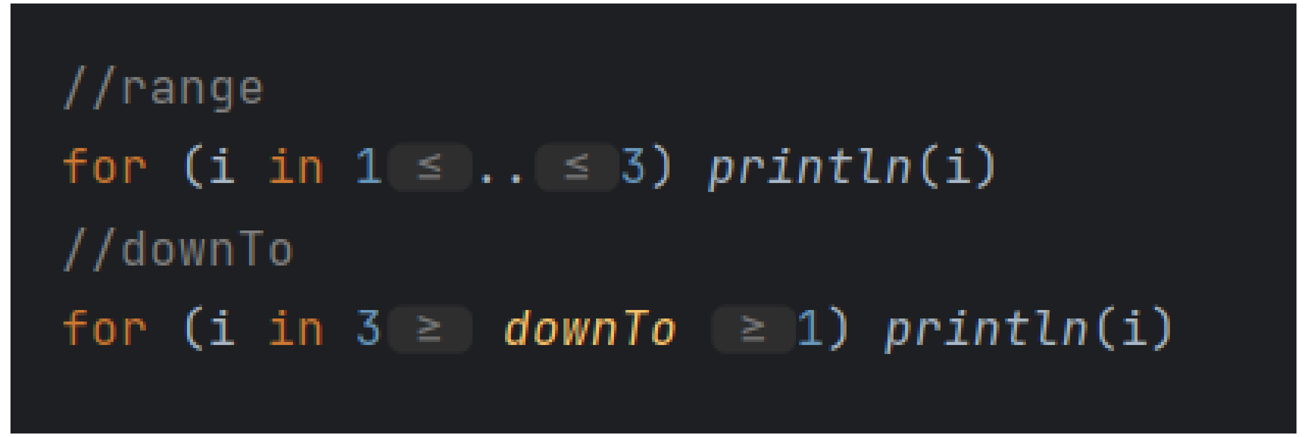
```
val names = listOf("sandy", "bob", "patrick", "eugene")

for (index in names.indices.reversed()){
    println("name $index is ${names[index]}")
}
```



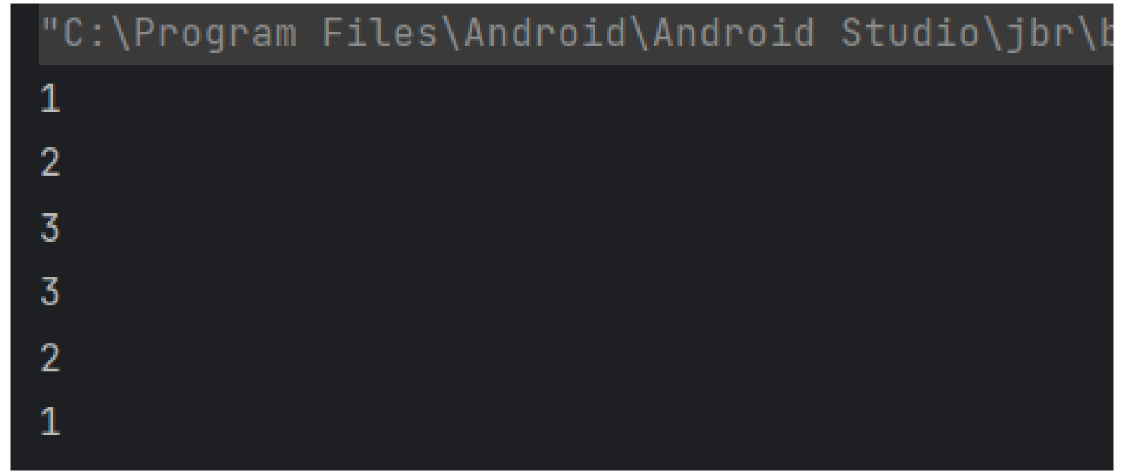
```
"C:\Program Files\Android\Android Studio
name 3 is eugene
name 2 is patrick
name 1 is bob
name 0 is sandy

Process finished with exit code 0
```

Ranges & downTo

- We can easily loop ranges up or down like so



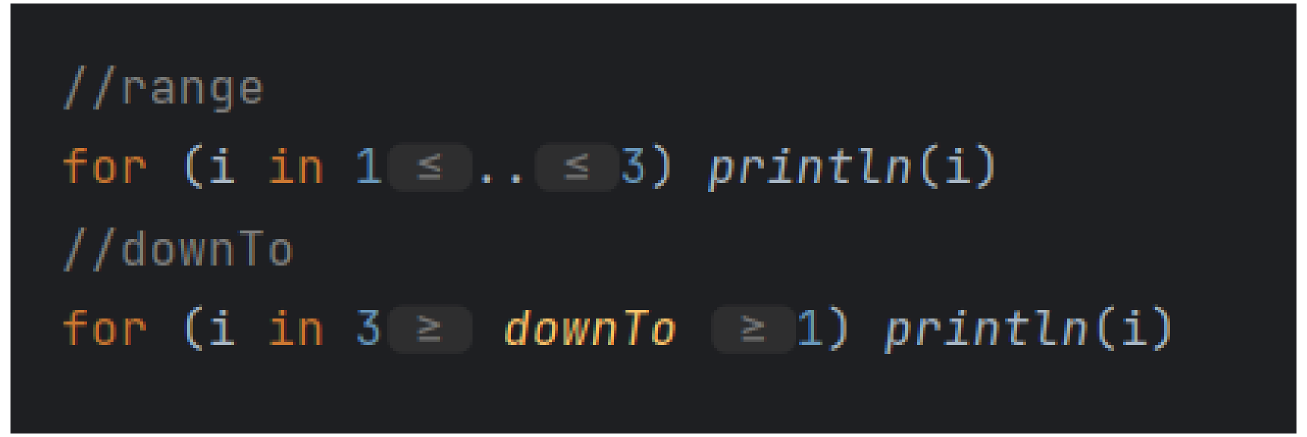
```
//range
for (i in 1 ≤ .. ≤ 3) println(i)
//downTo
for (i in 3 ≥  downTo  ≥ 1) println(i)
```



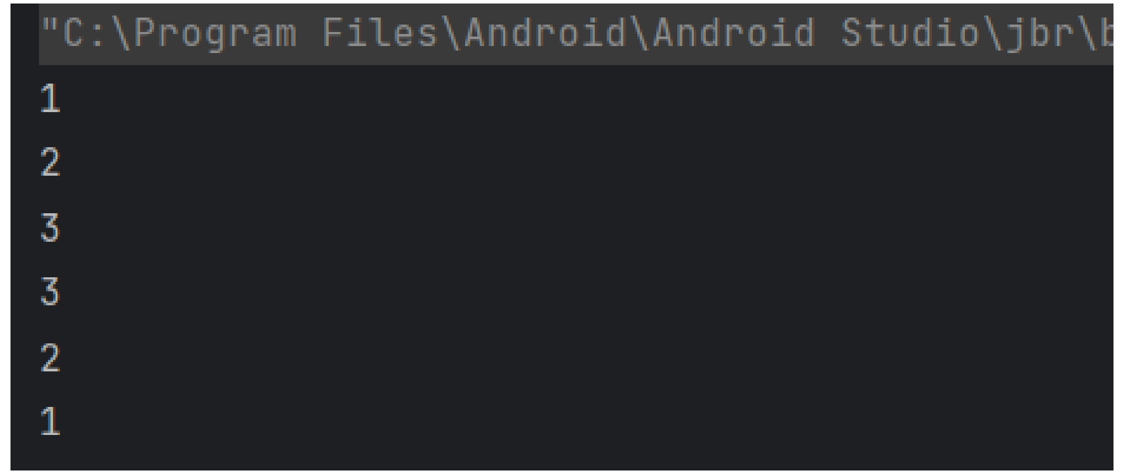
```
"C:\Program Files\Android\Android Studio\jbr\
1
2
3
3
2
1
```

范围与 downTo

- 我们可以像这样轻松地向上或向下循环范围



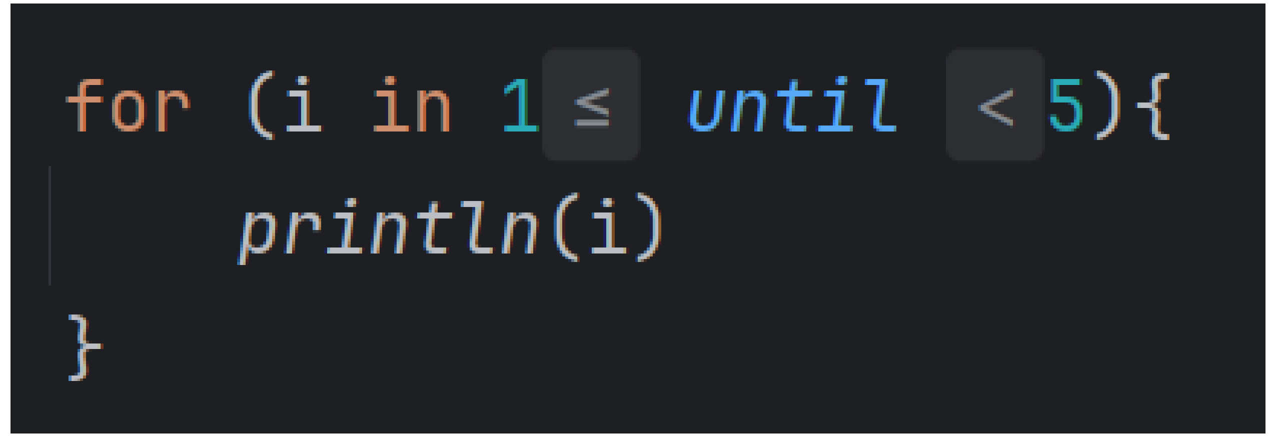
```
//range
for (i in 1 ≤ .. ≤ 3) println(i)
//downTo
for (i in 3 ≥  downTo  ≥ 1) println(i)
```



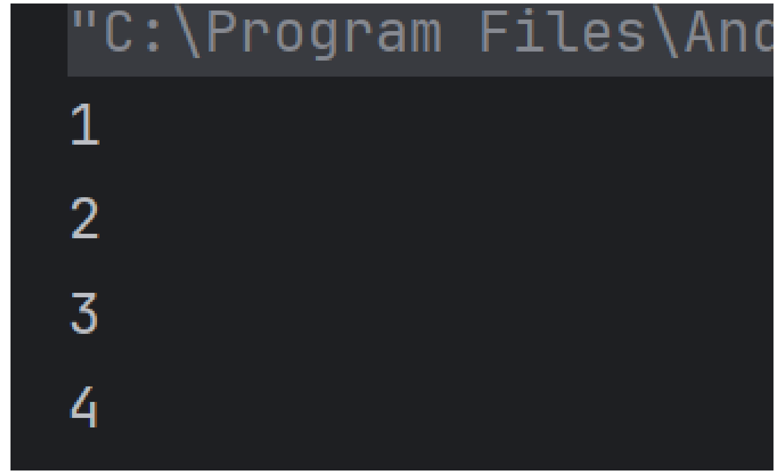
```
"C:\Program Files\Android\Android Studio\jbr\
1
2
3
3
2
1
```

Until

- For exclusive range, we can use *until*



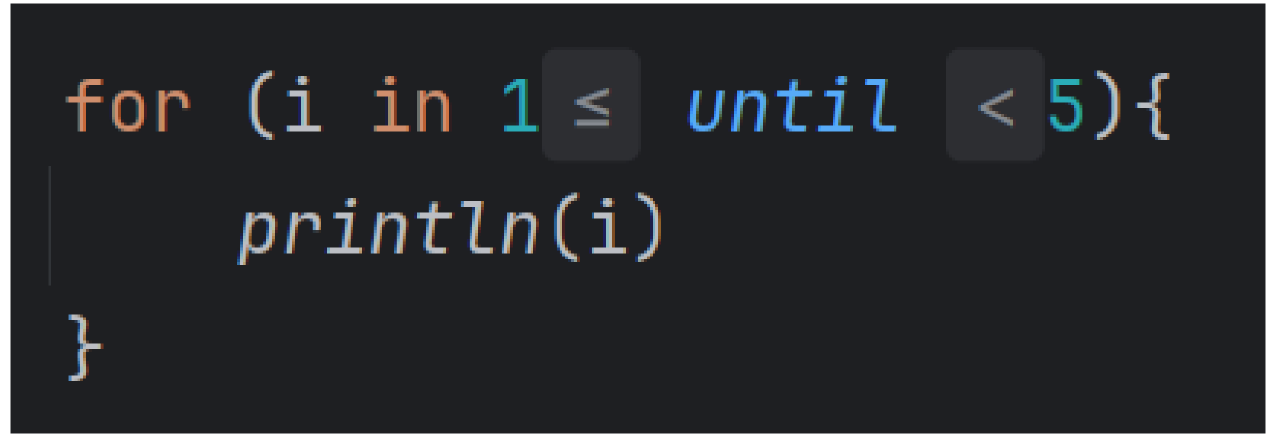
```
for (i in 1 ≤  until  < 5){
    println(i)
}
```



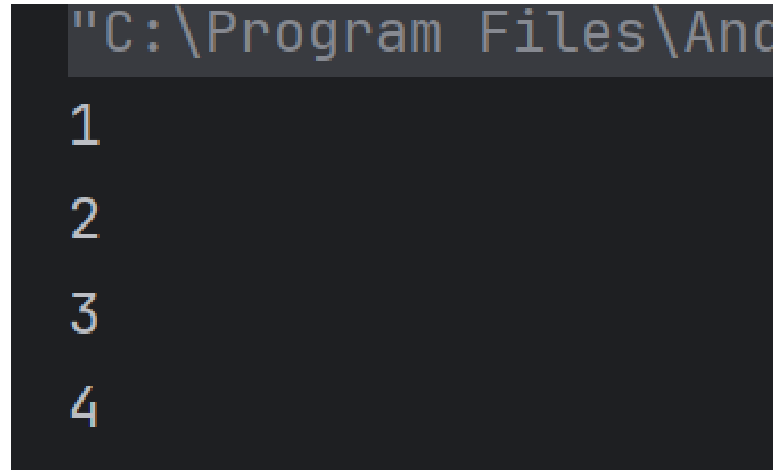
```
"C:\Program Files\And
1
2
3
4
```

直到

- 对于不包含上限的范围，我们可以使用*until*



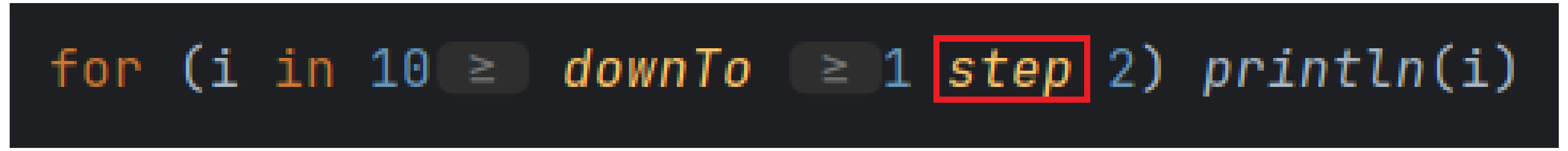
```
for (i in 1 ≤  until  < 5){
    println(i)
}
```



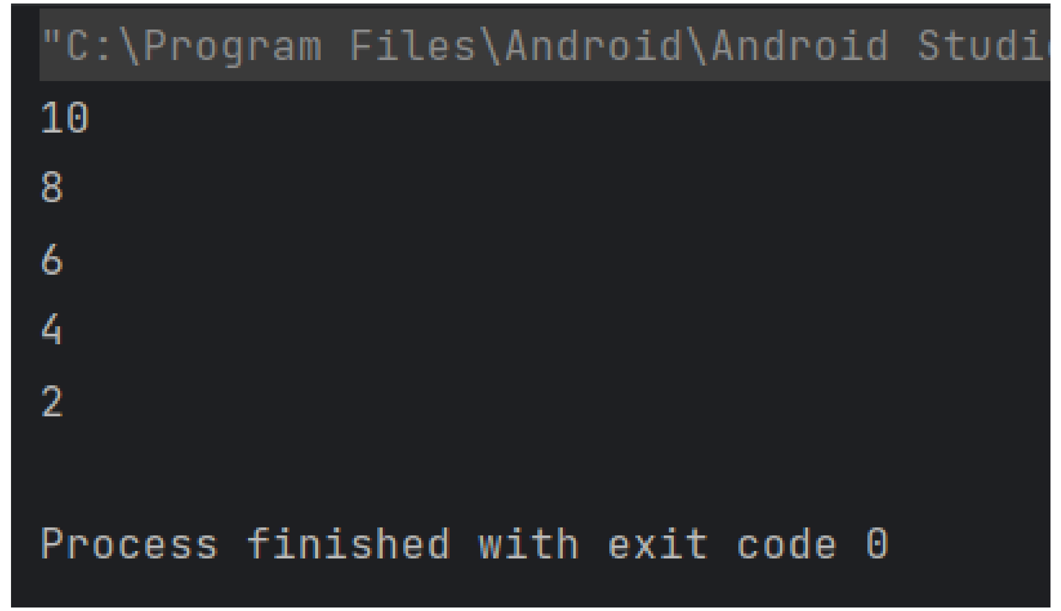
```
"C:\Program Files\And
1
2
3
4
```

Step

- We can also loop a range or downTo using a step



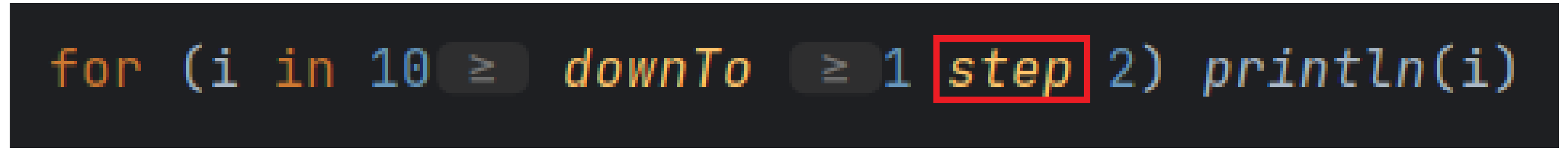
```
for (i in 10 ≥  downTo  ≥ 1 step 2) println(i)
```



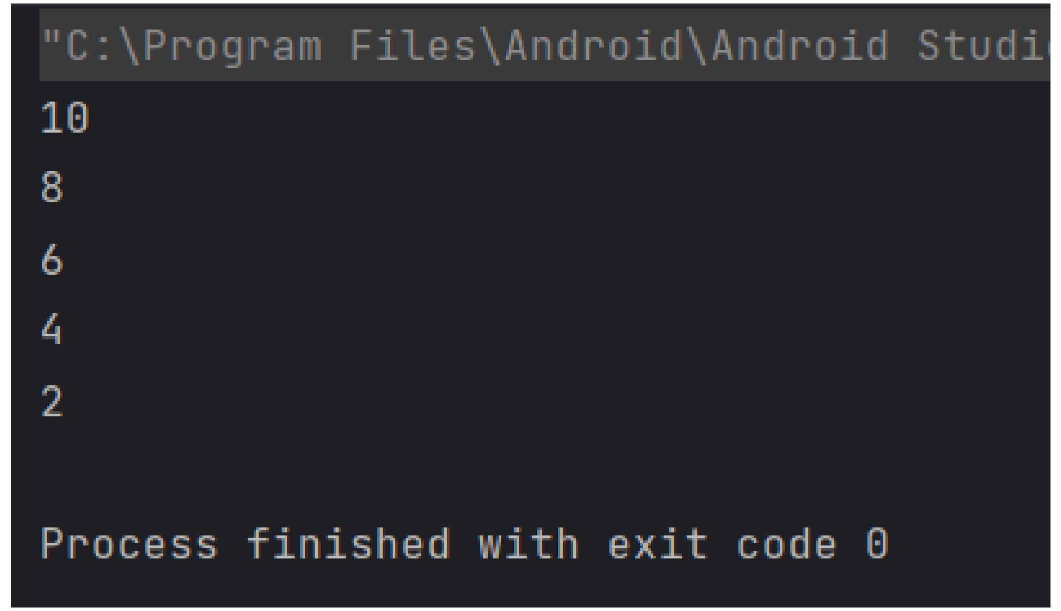
```
"C:\Program Files\Android\Android Studi
10
8
6
4
2

Process finished with exit code 0
```

Step

- 我们还可以使用 step 来循环范围或 downTo



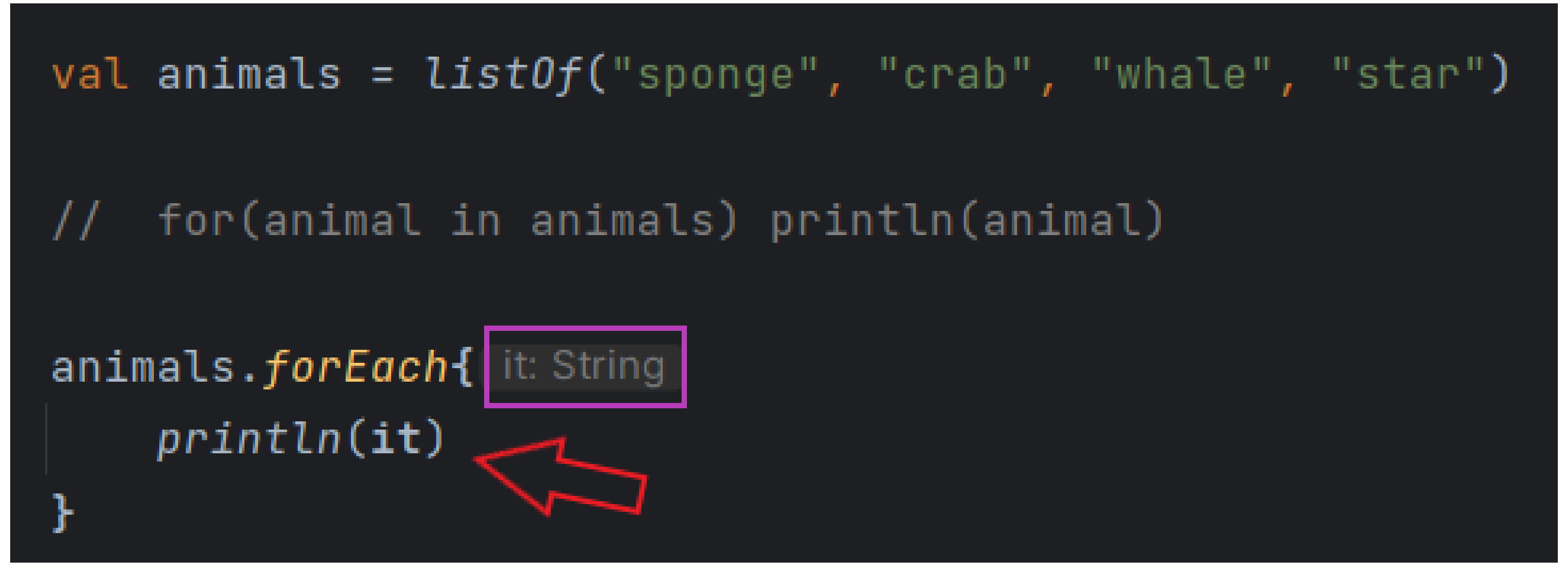
```
for (i in 10 ≥  downTo  ≥ 1 step 2) println(i)
```



```
"C:\Program Files\Android\Android Studi
10
8
6
4
2

Process finished with exit code 0
```

forEach

- *forEach* is similar to the for loop but the syntax is slightly different
  - Notice we use the *it* keyword here instead of 'animal' like we would in a for loop

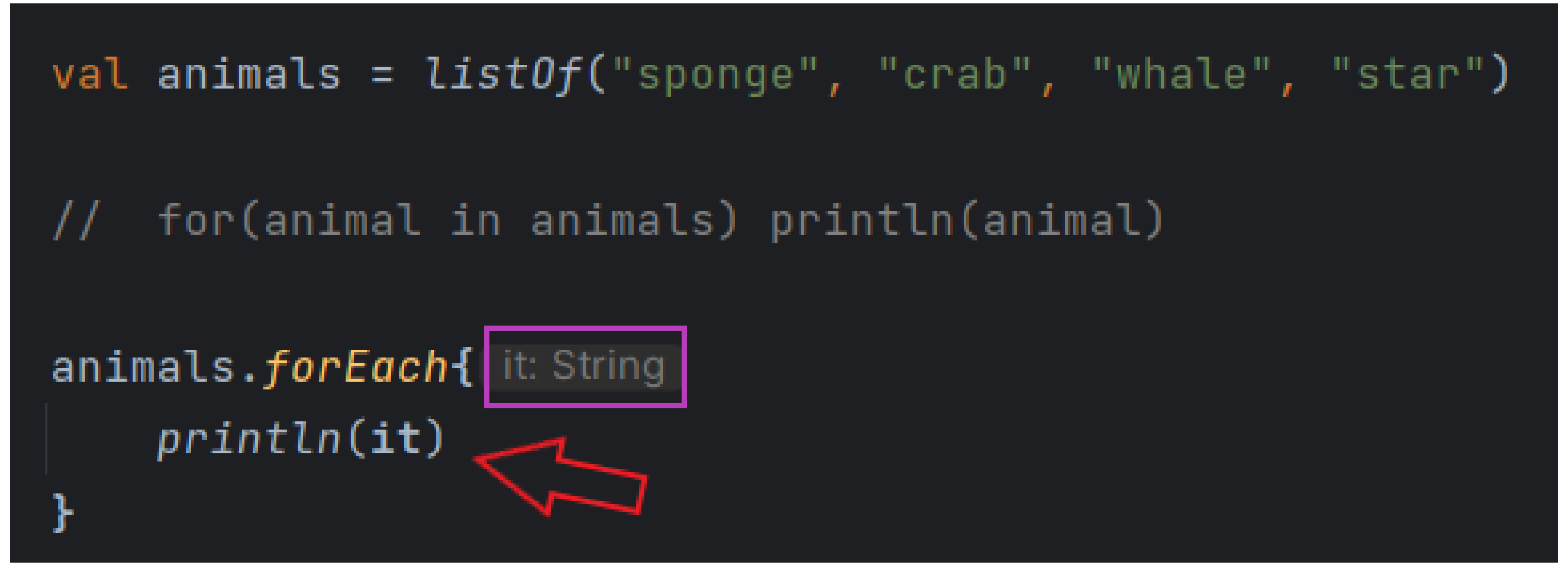


```
val animals = listOf("sponge", "crab", "whale", "star")

//  for(animal in animals) println(animal)

animals.forEach{ it: String
    println(it)
}
```

- The editor tells us *it* is a String

forEach

- *forEach* 与 for 循环类似，但语法略有不同
  - 请注意，我们在这里使用 *it* 关键字，而不是像在 for 循环中那样使用 'animal'

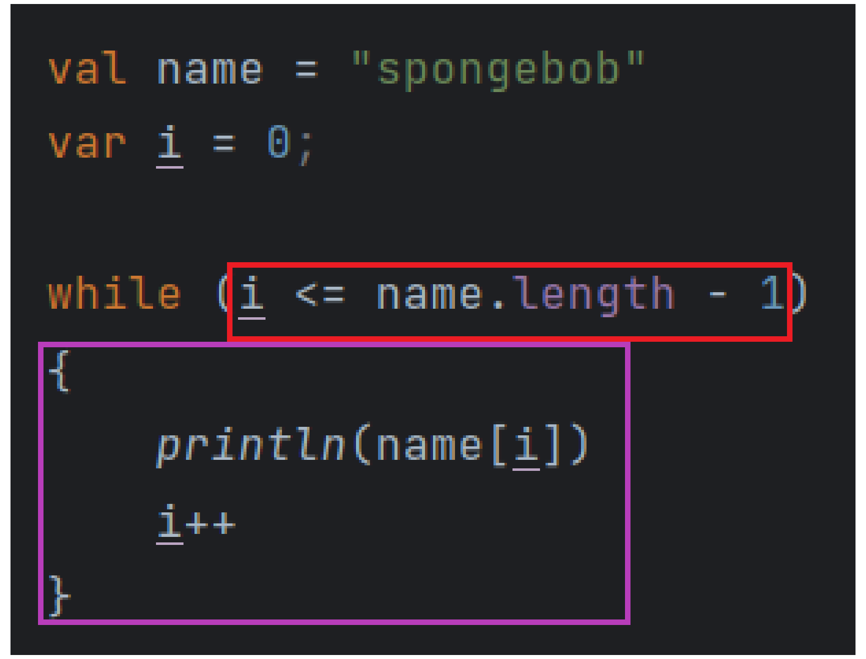


```
val animals = listOf("sponge", "crab", "whale", "star")

//  for(animal in animals) println(animal)

animals.forEach{ it: String
    println(it)
}
```

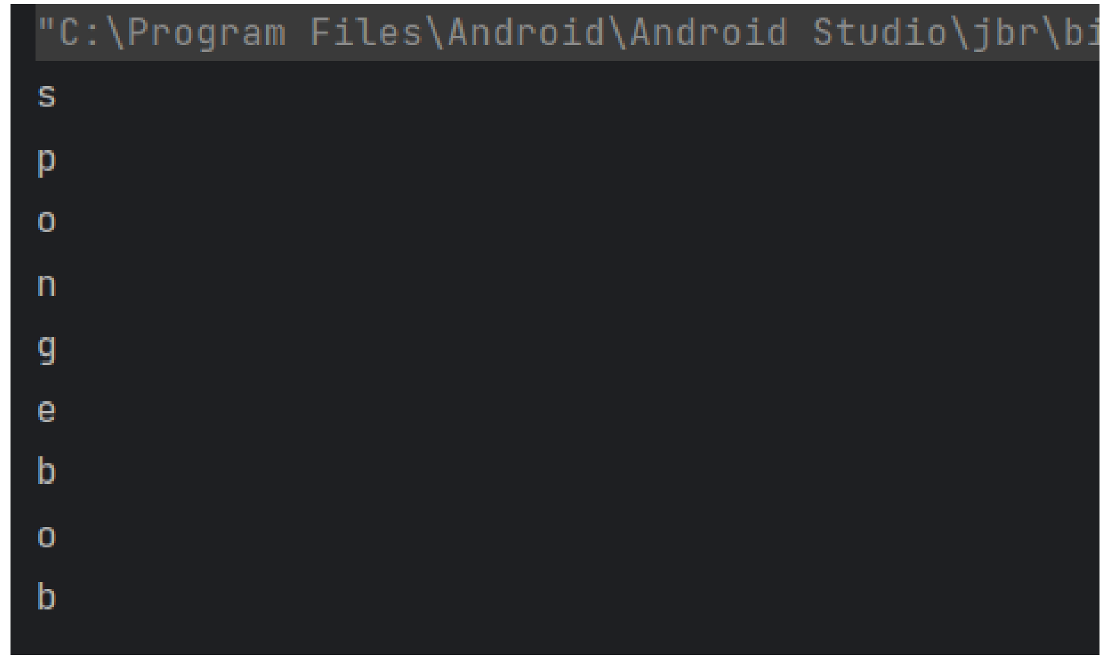
- 编辑器告诉我们 *it* 是 一个字符串

While loop

- While some condition is true, perform an operation
  - keep repeating the operation until the condition is false



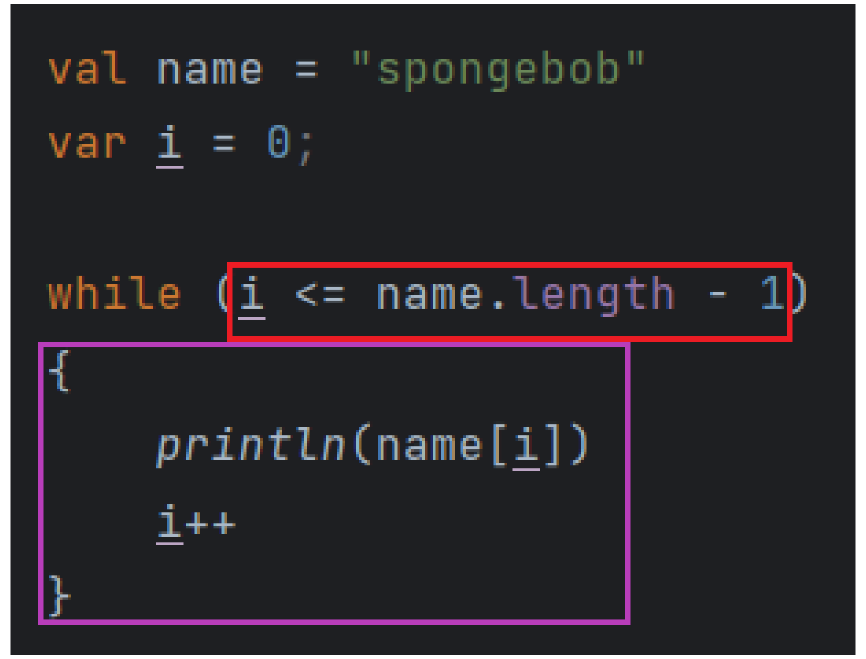
```
val name = "spongebob"
var i = 0;

while (i <= name.length - 1)
{
    println(name[i])
    i++
}
```



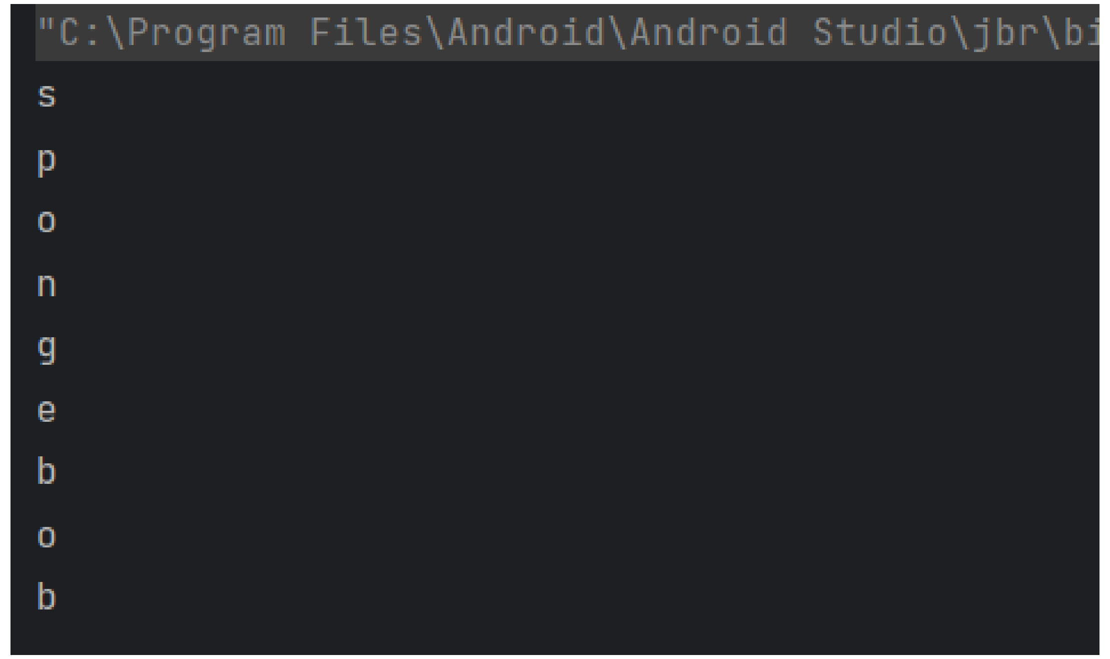
```
"C:\Program Files\Android\Android Studio\jbr\bi
s
p
o
n
g
e
b
o
b
```

while循环

- 当某些 条件 为真时，执行 操作
  - 不断重复 操作，直到 条件 为假



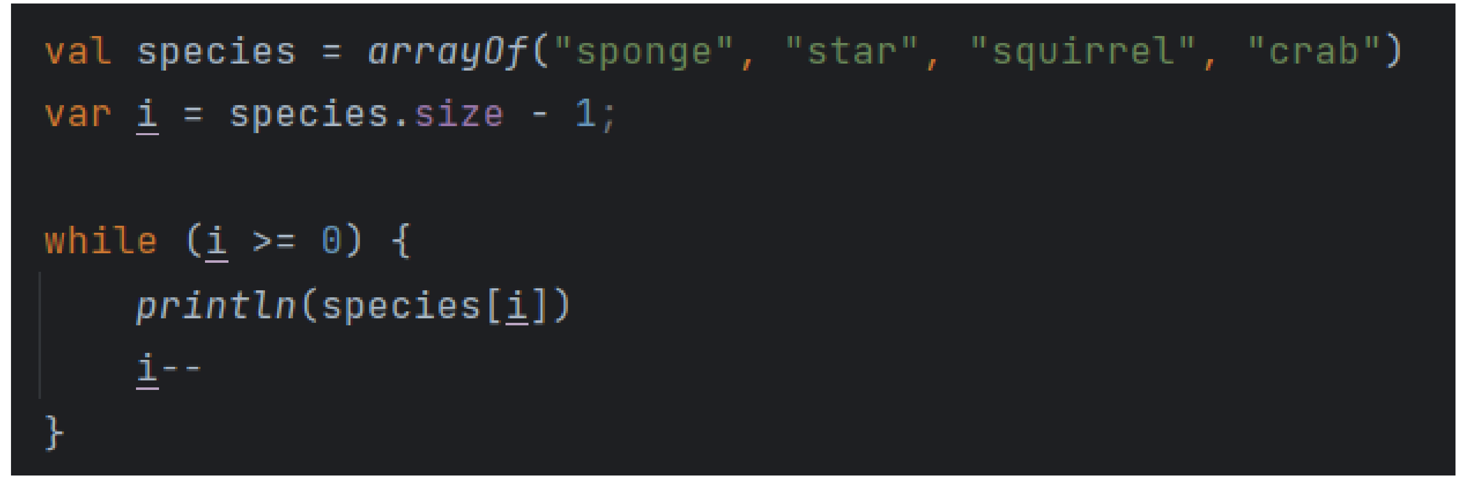
```
val name = "spongebob"
var i = 0;

while (i <= name.length - 1)
{
    println(name[i])
    i++
}
```



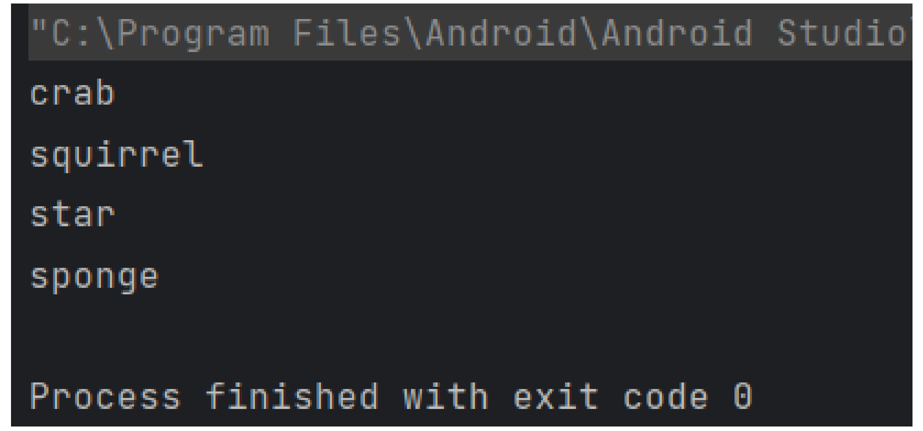
```
"C:\Program Files\Android\Android Studio\jbr\bi
s
p
o
n
g
e
b
o
b
```

While loop (cont.)

- Here is another example reversing through an array



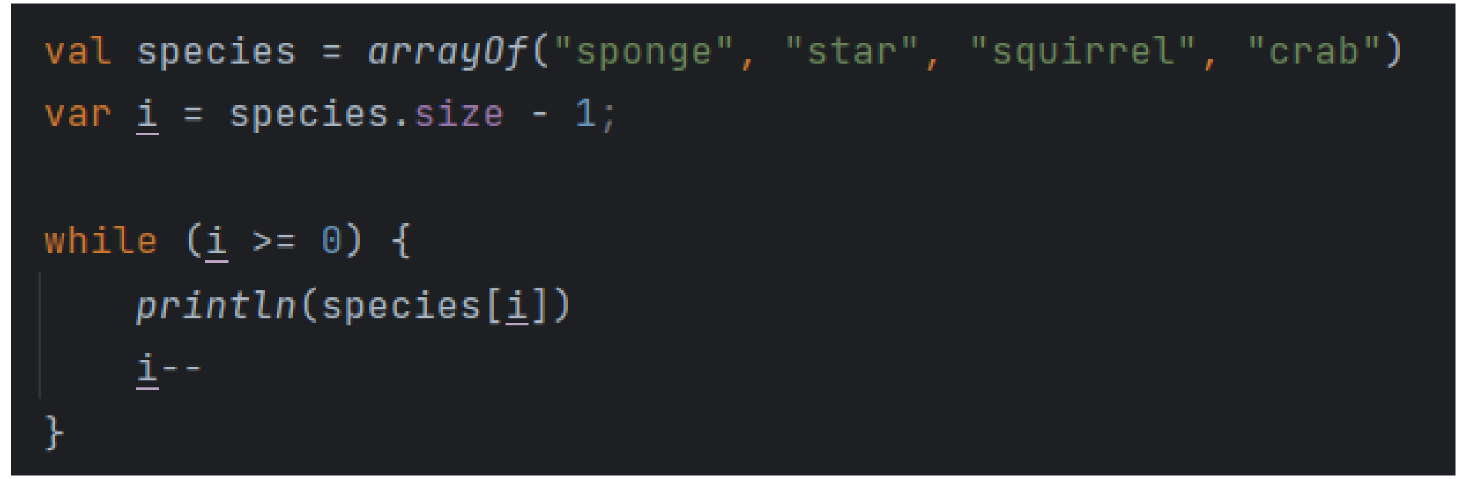
```kotlin
val species = arrayOf("sponge", "star", "squirrel", "crab")
var i = species.size - 1;

while (i >= 0) {
    println(species[i])
    i--
}
```



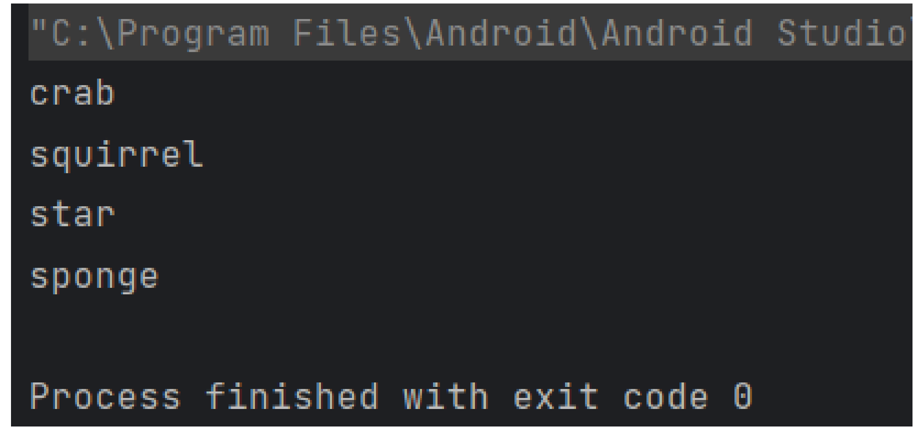
```
"C:\Program Files\Android\Android Studio
crab
squirrel
star
sponge

Process finished with exit code 0
```

while 循环（续）

- 这是另一个逆序遍历数组的示例



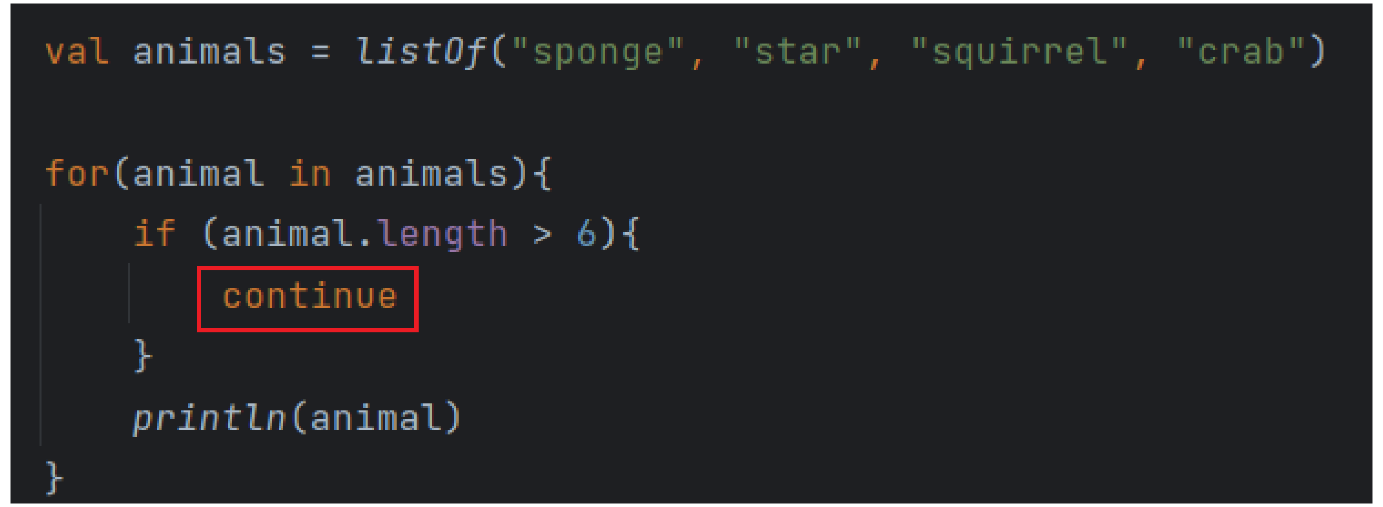
```kotlin
val species = arrayOf("sponge", "star", "squirrel", "crab")
var i = species.size - 1;

while (i >= 0) {
    println(species[i])
    i--
}
```



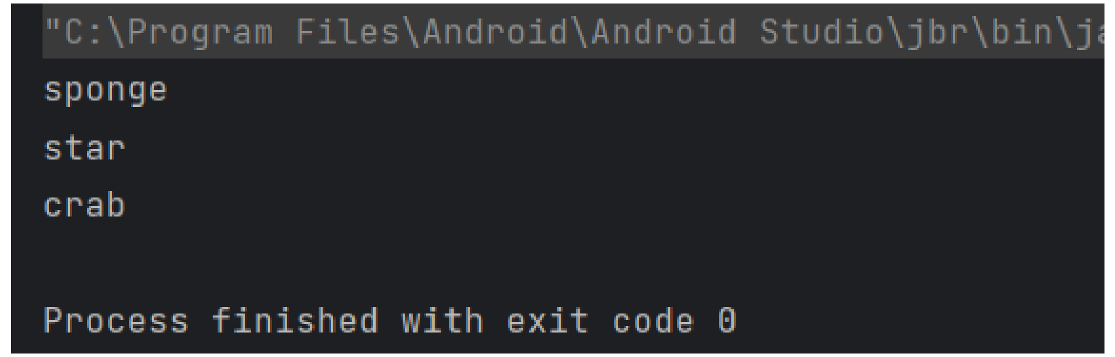
```
"C:\Program Files\Android\Android Studio
crab
squirrel
star
sponge

Process finished with exit code 0
```

continue

- When looping through a collection we can skip operations and continue to the next iteration



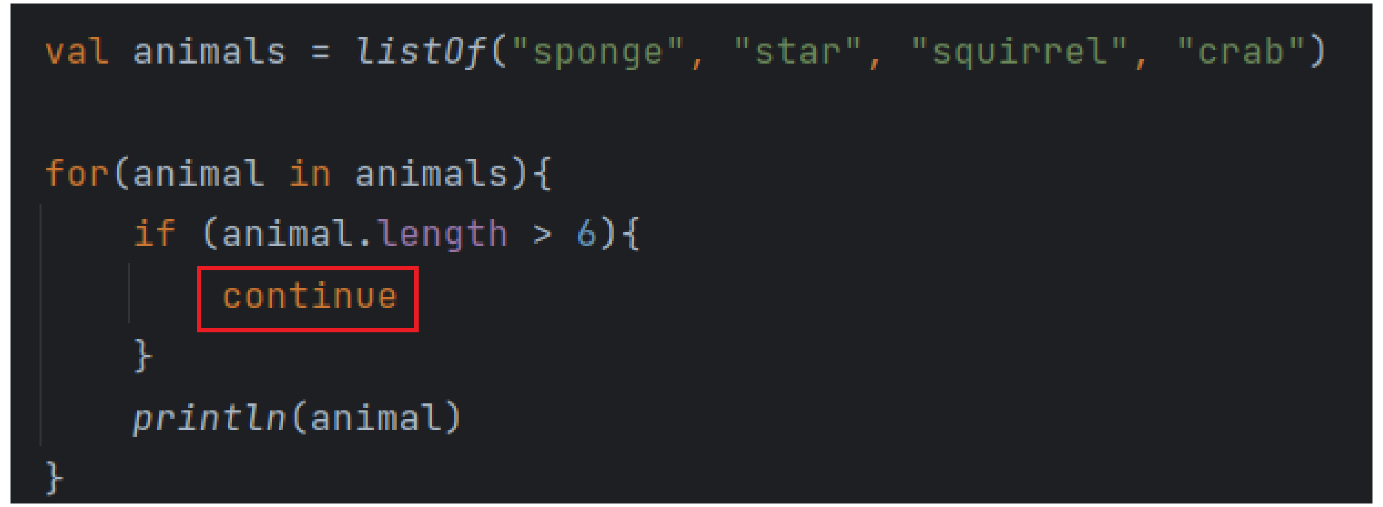
```
val animals = listOf("sponge", "star", "squirrel", "crab")

for(animal in animals){
    if (animal.length > 6){
        continue
    }
    println(animal)
}
```



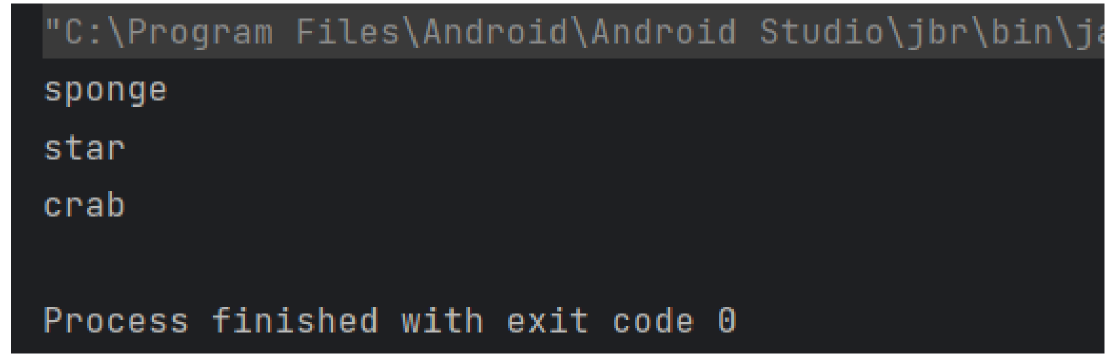
```
"C:\Program Files\Android\Android Studio\jbr\bin\ja
sponge
star
crab

Process finished with exit code 0
```

继续

- 在遍历集合时，我们可以跳过操作并继续 到下一次迭代



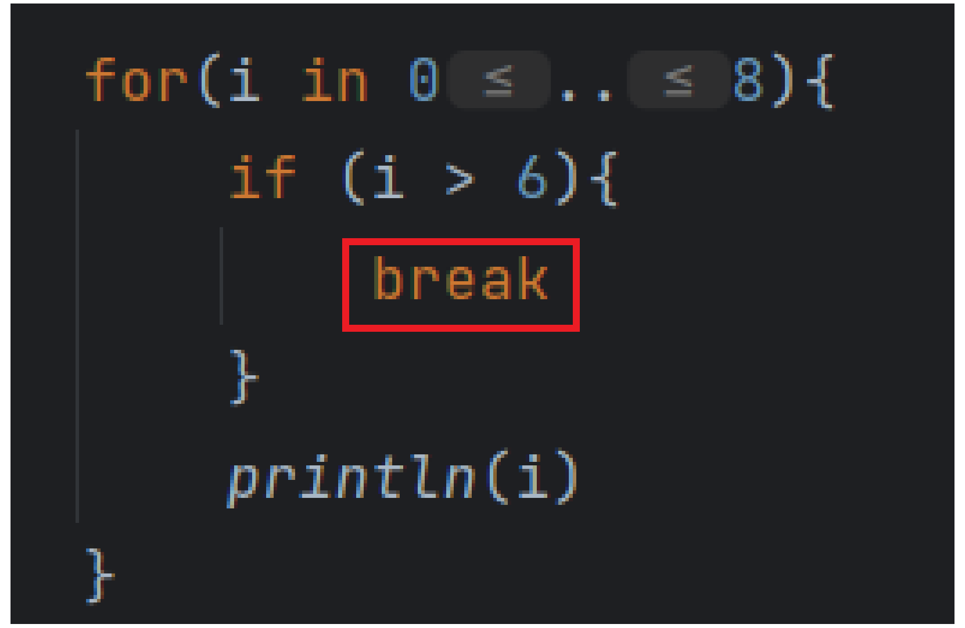
```
val animals = listOf("sponge", "star", "squirrel", "crab")

for(animal in animals){
    if (animal.length > 6){
        continue
    }
    println(animal)
}
```



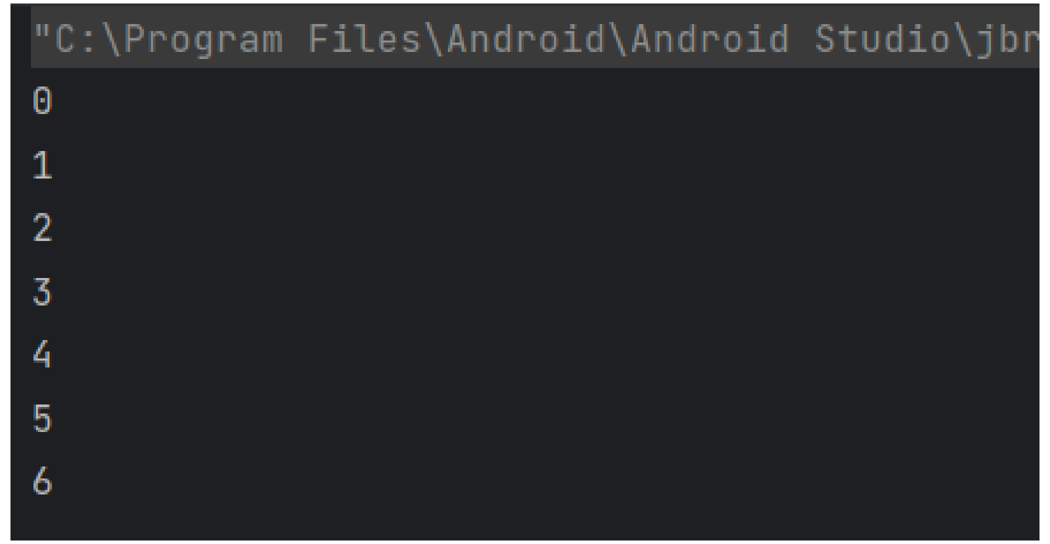
```
"C:\Program Files\Android\Android Studio\jbr\bin\ja
sponge
star
crab

Process finished with exit code 0
```

break

- break will exit the loop



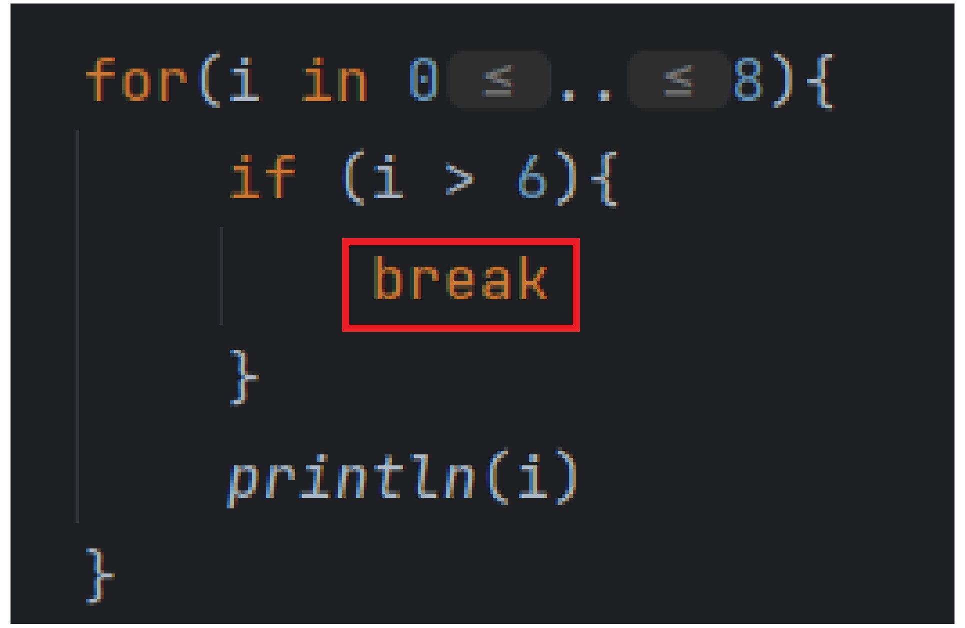
```
for(i in 0 ≤ .. ≤ 8){
    if (i > 6){
        break
    }
    println(i)
}
```



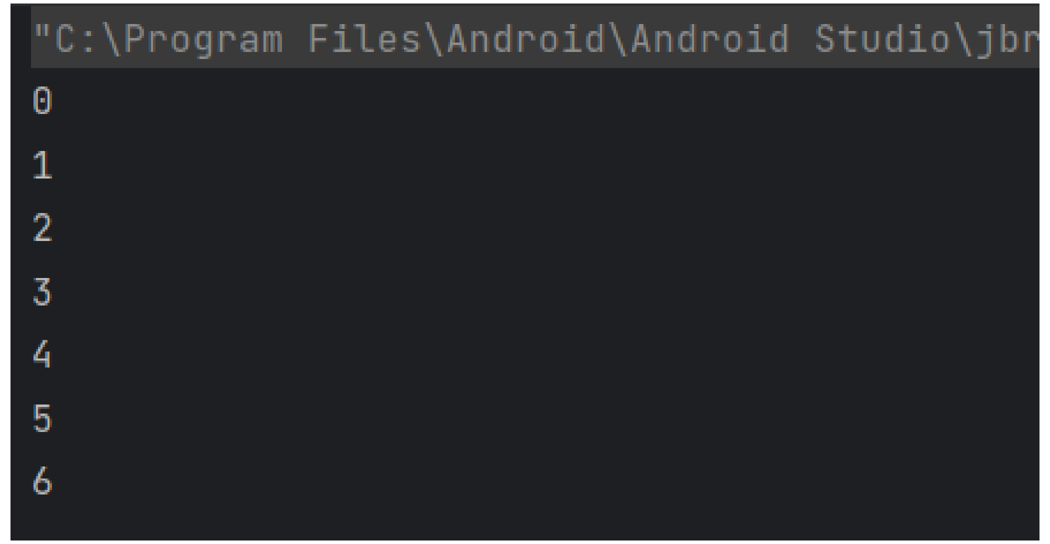
```
"C:\Program Files\Android\Android Studio\jbr
0
1
2
3
4
5
6
```

break

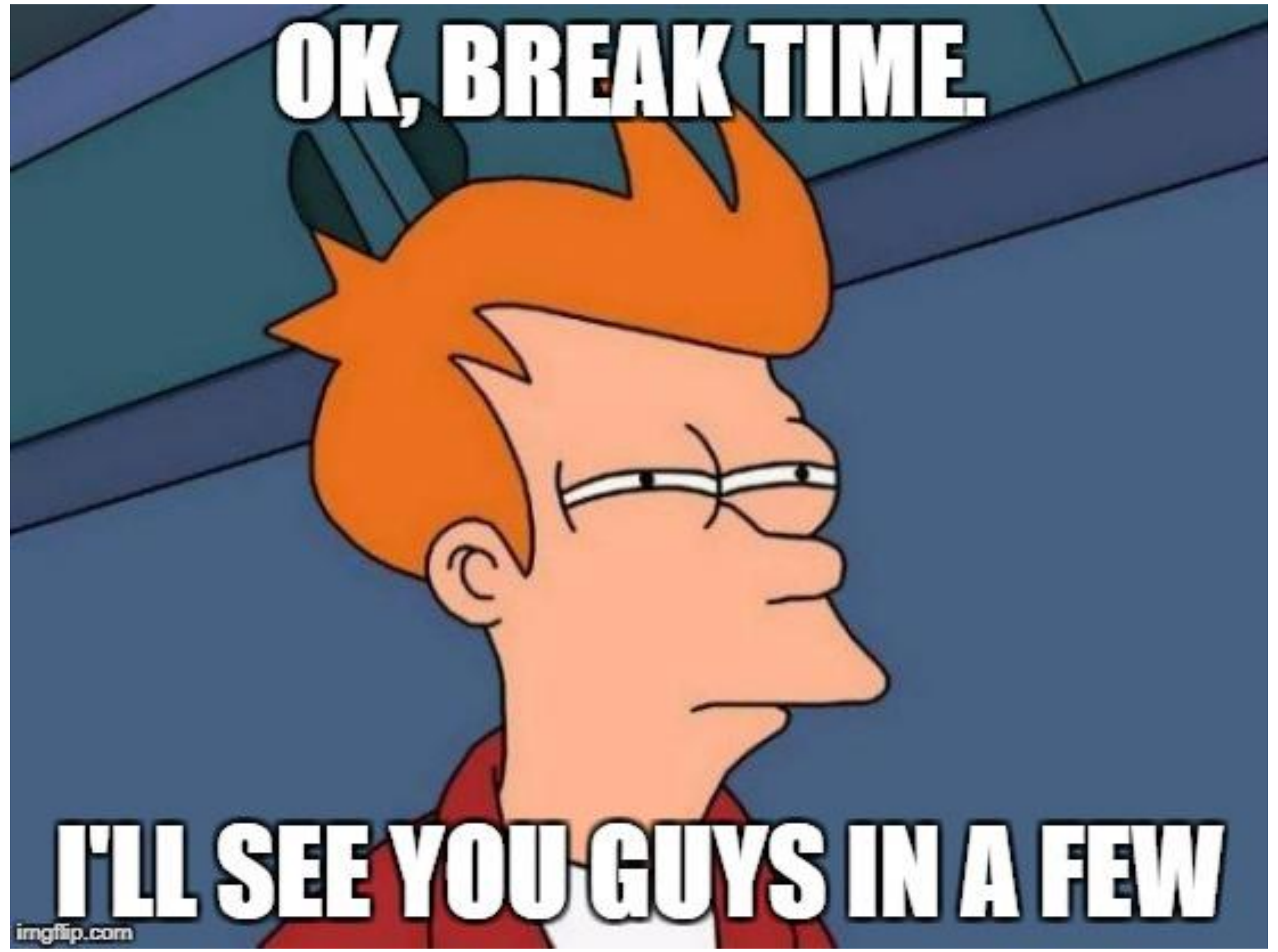
- break 将退出循环

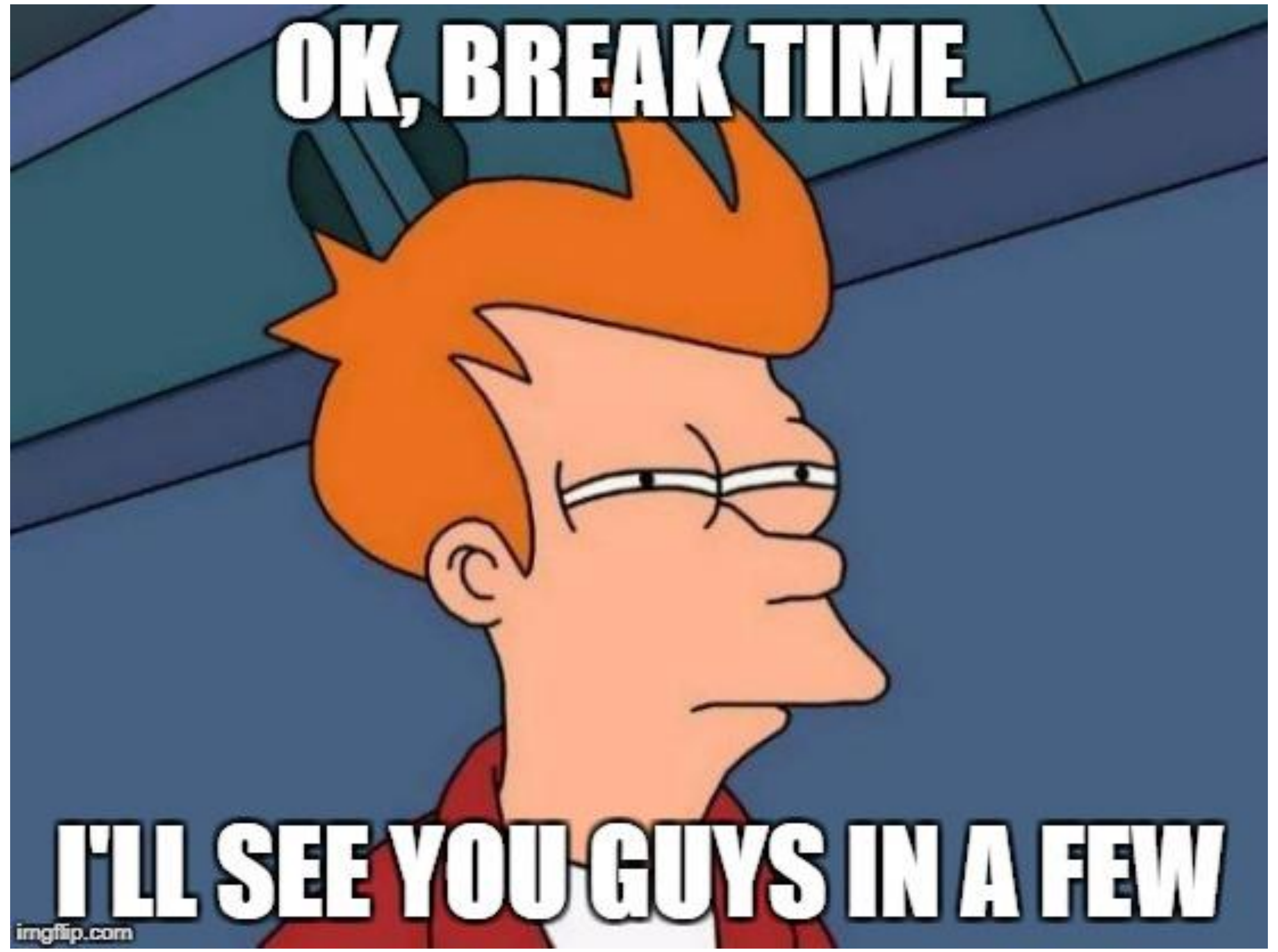


```
for(i in 0 ≤ .. ≤ 8){
    if (i > 6){
        break
    }
    println(i)
}
```



```
"C:\Program Files\Android\Android Studio\jbr
0
1
2
3
4
5
6
```


OK, BREAK TIME.
I'LL SEE YOU GUYS IN A FEW
imgflip.com


OK, BREAK TIME.
I'LL SEE YOU GUYS IN A FEW
imgflip.com

Class Activity 1

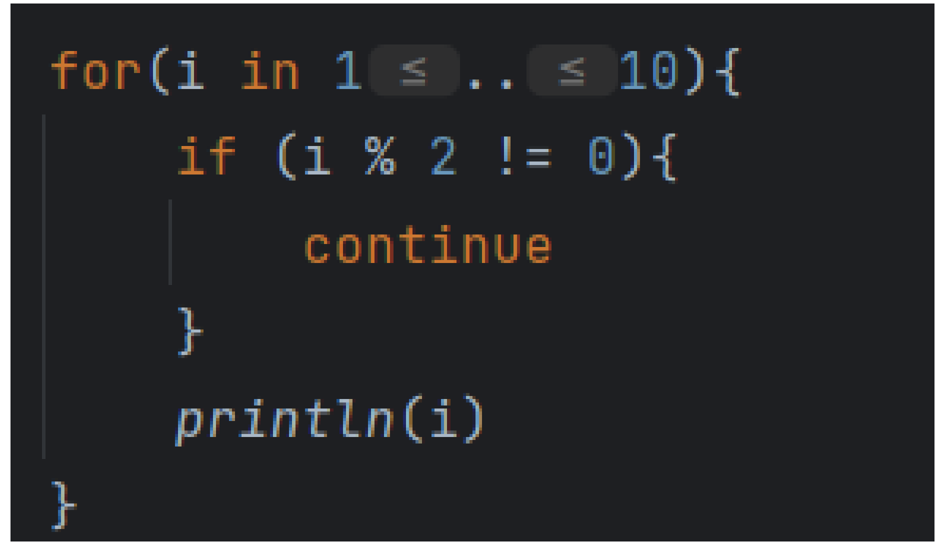
- Using *continue* and a *for loop*, iterate through 1-10 only printing even numbers

课堂活动 1

- 使用 *continue* 和一个 for 循环，遍历 1 到 10，仅打印偶数

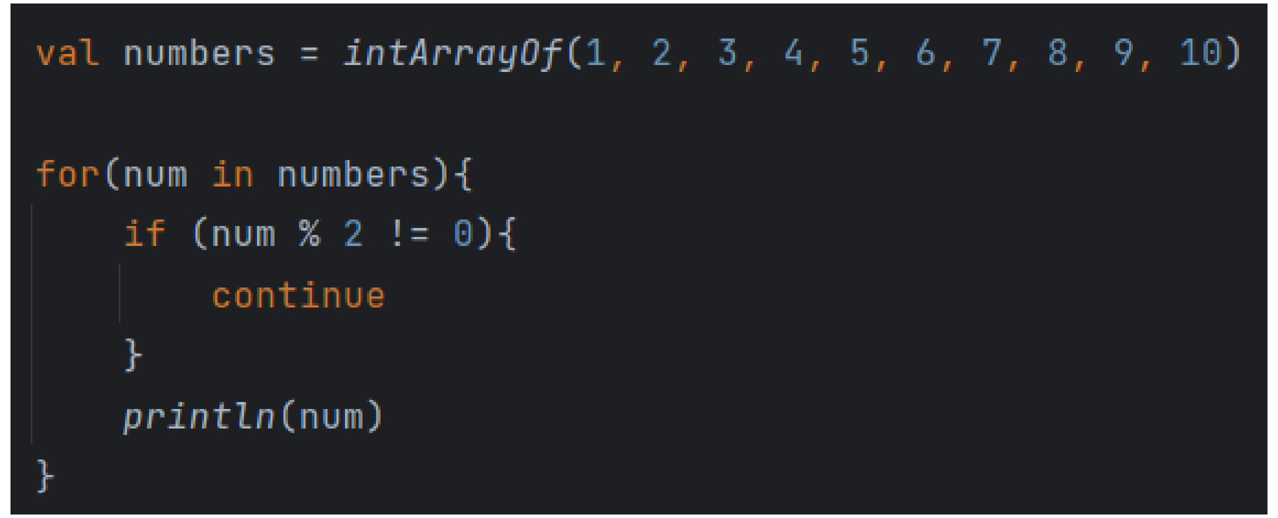
Class Activity 1 Answer

Using ranges



```
for(i in 1 ≤ .. ≤ 10){
    if (i % 2 != 0){
        continue
    }
    println(i)
}
```

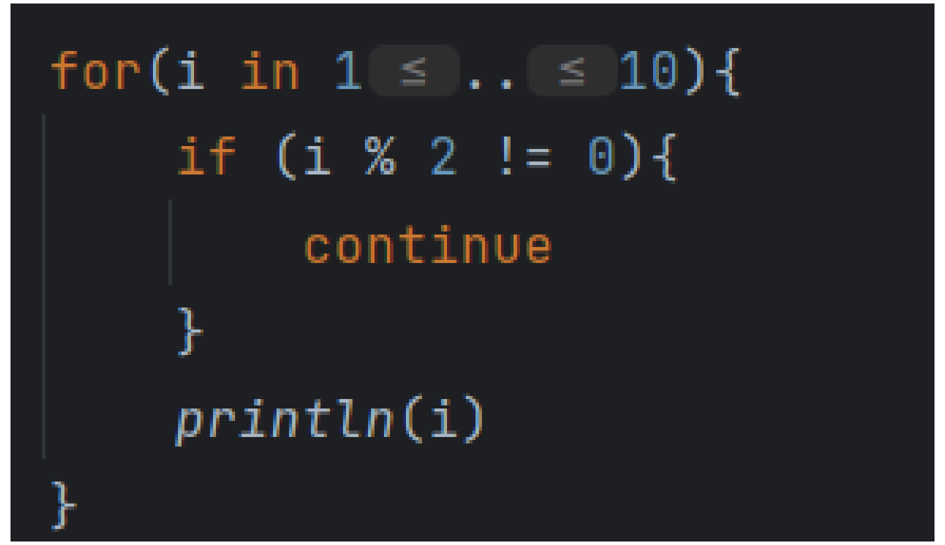
Using an array



```
val numbers = intArrayOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

for(num in numbers){
    if (num % 2 != 0){
        continue
    }
    println(num)
}
```

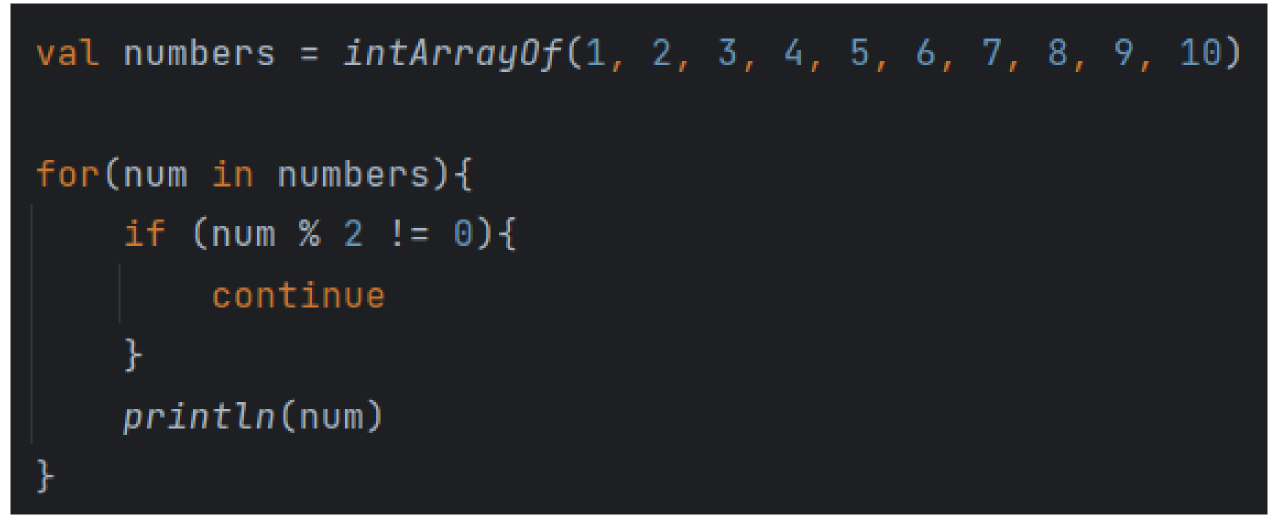
课堂活动1答案

使用范围



```
for(i in 1 ≤ .. ≤ 10){
    if (i % 2 != 0){
        continue
    }
    println(i)
}
```

使用数组



```
val numbers = intArrayOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

for(num in numbers){
    if (num % 2 != 0){
        continue
    }
    println(num)
}
```

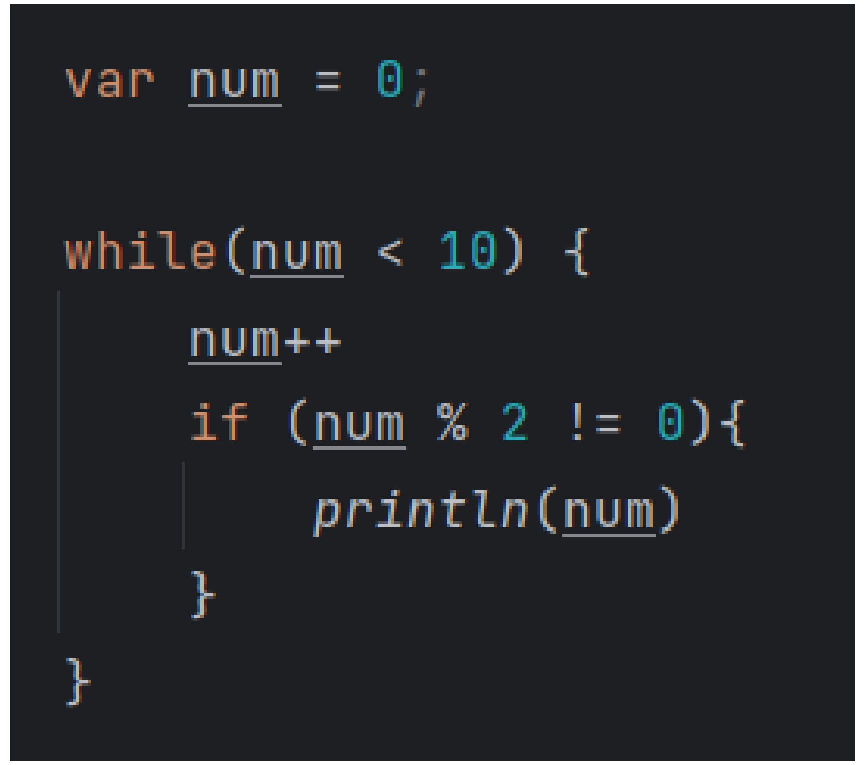
Class Activity 2

- Using a *while loop*, iterate through 1-10 only printing odd numbers

课堂活动 2

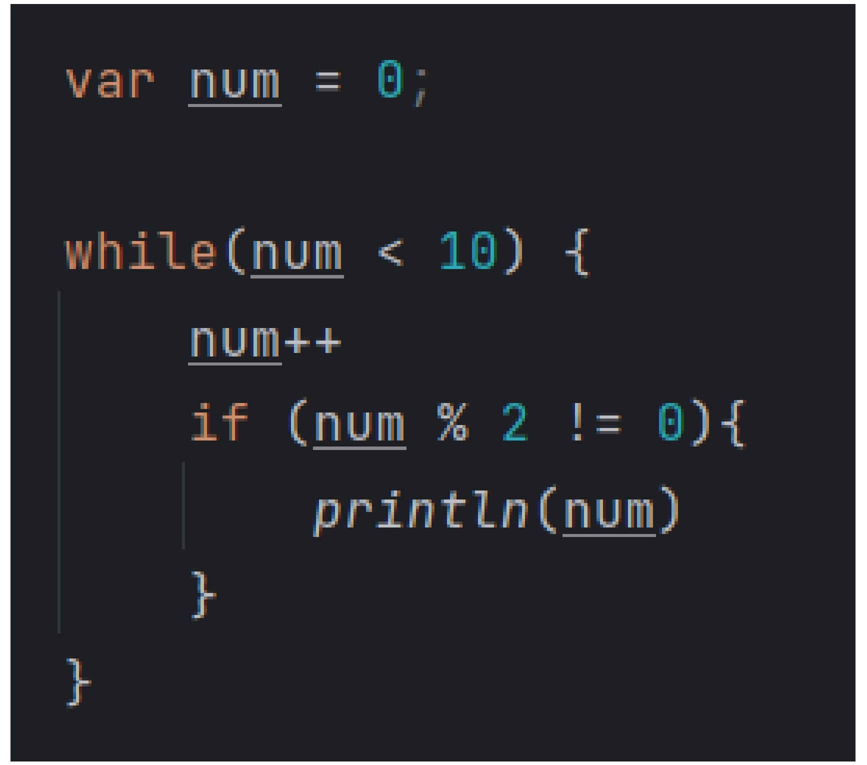
- 使用while循环，遍历1-10，仅打印奇数

Class Activity 2 Answer



```
var num = 0;

while(num < 10) {
    num++
    if (num % 2 != 0){
        println(num)
    }
}
```

课堂活动2答案



```
var num = 0;

while(num < 10) {
    num++
    if (num % 2 != 0){
        println(num)
    }
}
```

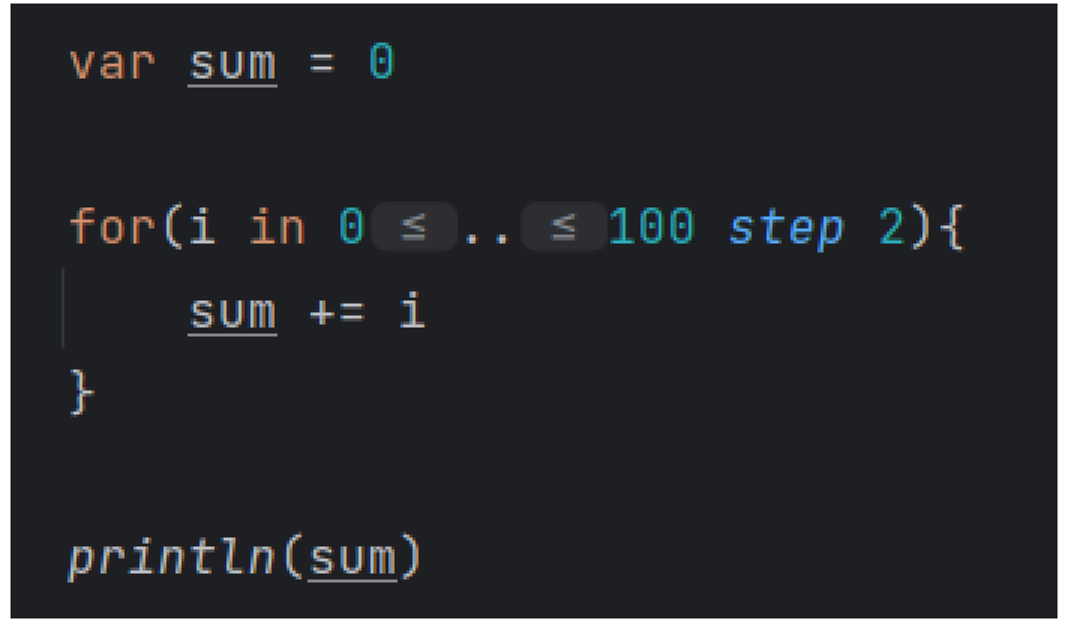
Class Activity 3

- Calculate the sum of every second number between 1 and 100

课堂活动 3

- 计算 1 到 100 之间每隔一个数的数字之和

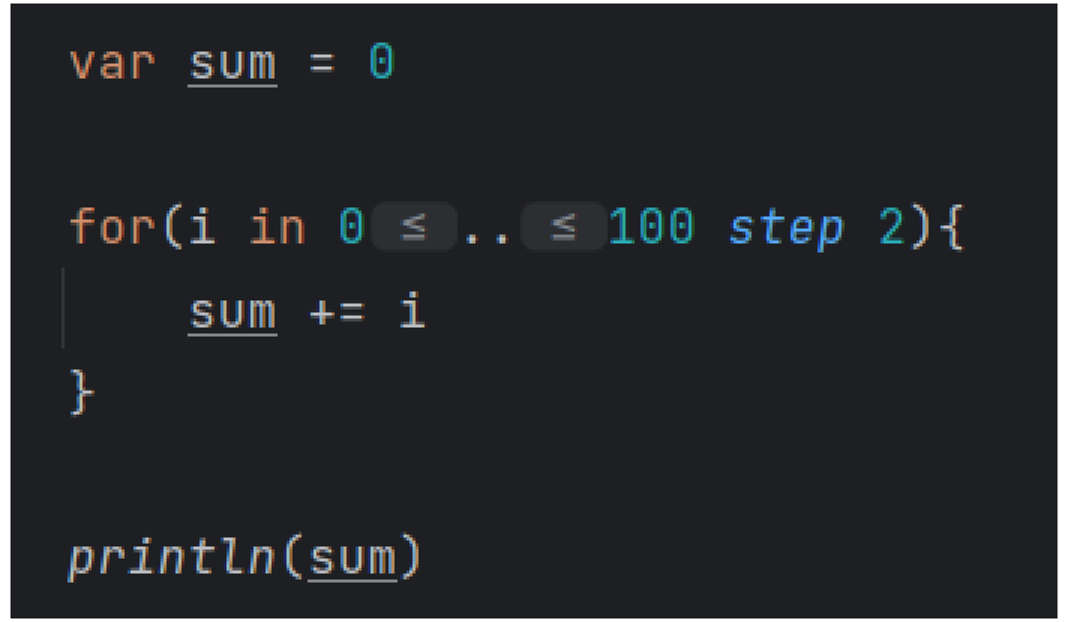
Class Activity 3 Answer



```
var sum = 0

for(i in 0 ≤ .. ≤ 100 step 2){
    sum += i
}

println(sum)
```

2550

课堂活动3答案



```
var sum = 0

for(i in 0 ≤ .. ≤ 100 step 2){
    sum += i
}

println(sum)
```

2550

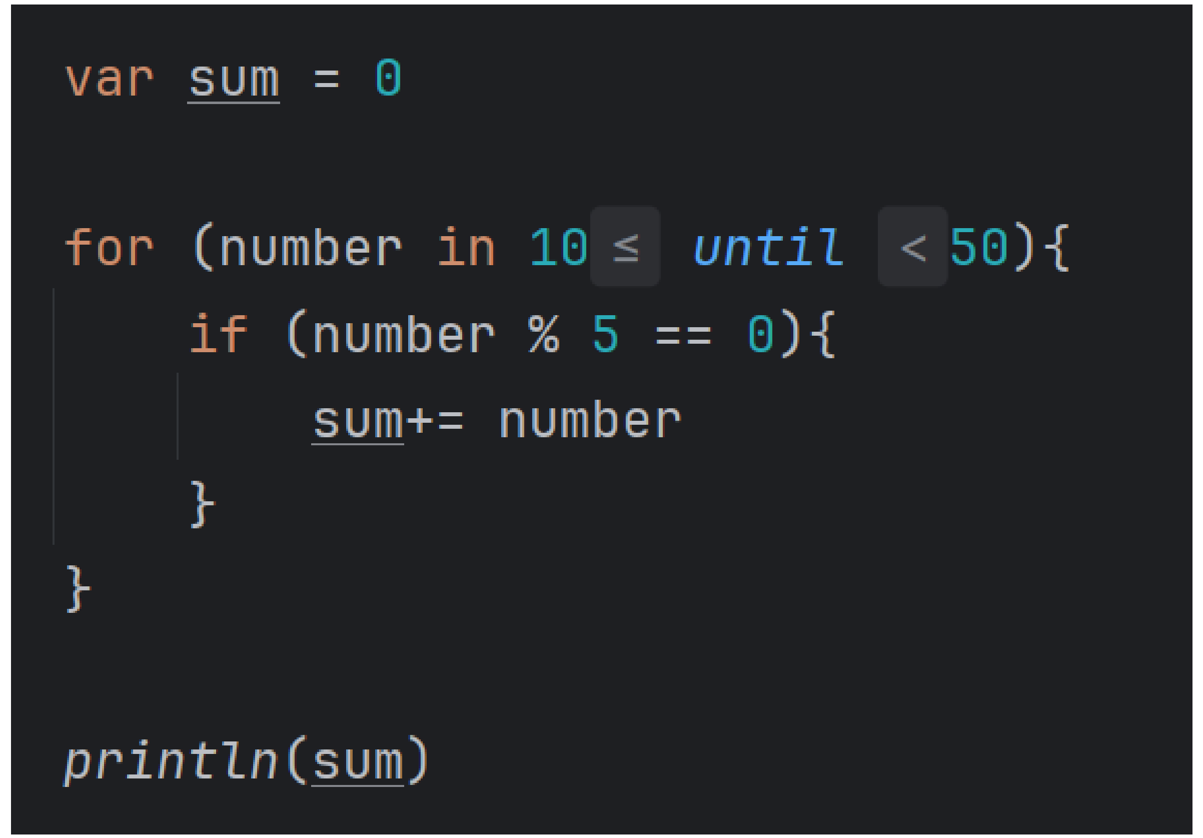
Class Activity 4

- Using exclusive range *until*, calculate the sum of all numbers between 10 to 50 that are divisible by 5

课堂活动 4

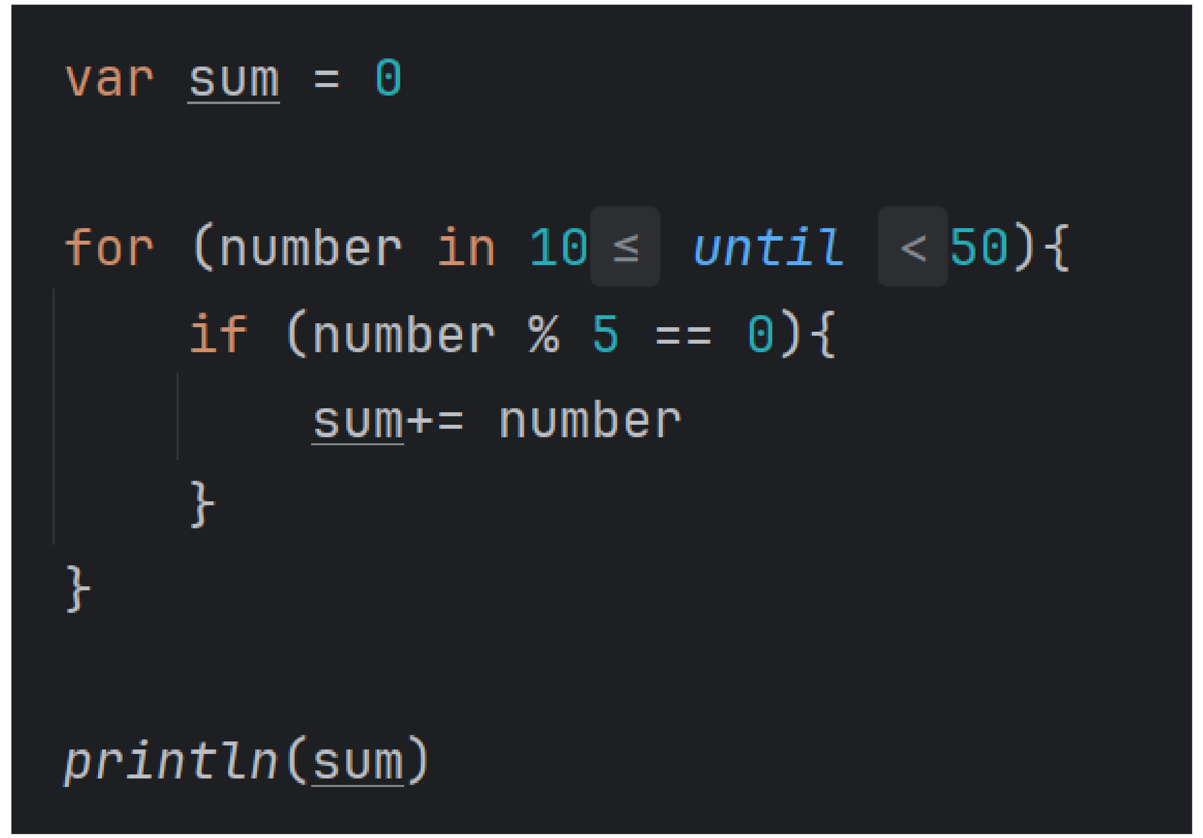
- 使用独占范围 *until*，计算 10 到 50 之间所有能被 5 整除的数的和

Class Activity 4 Answer



```
var sum = 0

for (number in 10 ≤ until < 50){
    if (number % 5 == 0){
        sum+= number
    }
}

println(sum)
```

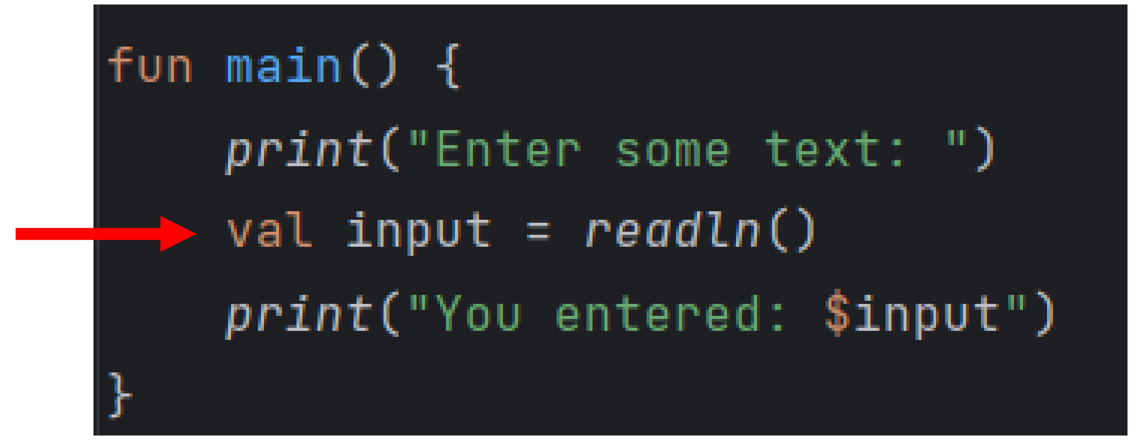
课堂活动4答案



```
var sum = 0

for (number in 10 ≤ until < 50){
    if (number % 5 == 0){
        sum+= number
    }
}

println(sum)
```

Input

- Reading input in Kotlin can be done using readln

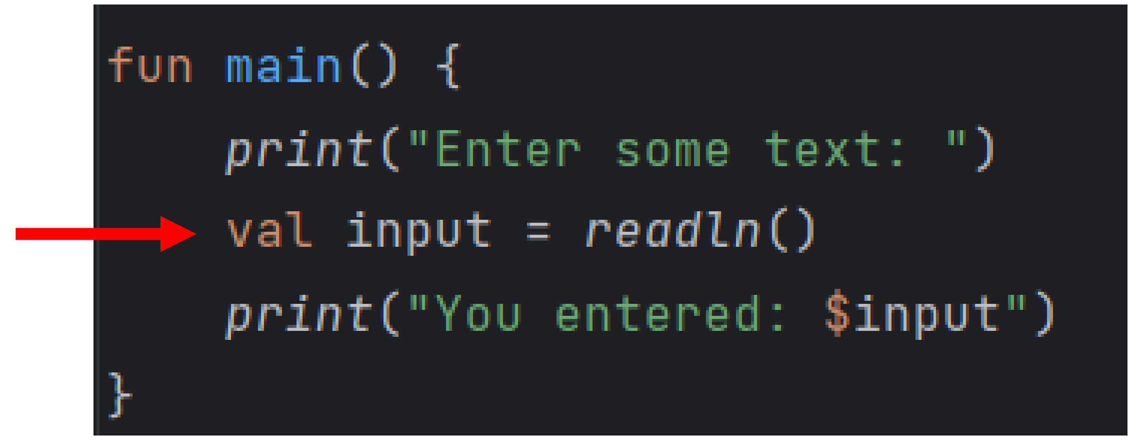


```
fun main() {
    print("Enter some text: ")
    val input = readln()
    print("You entered: $input")
}
```

- *readln* returns a *String*

输入

- 在 Kotlin 中可以通过 readln 进行输入读取

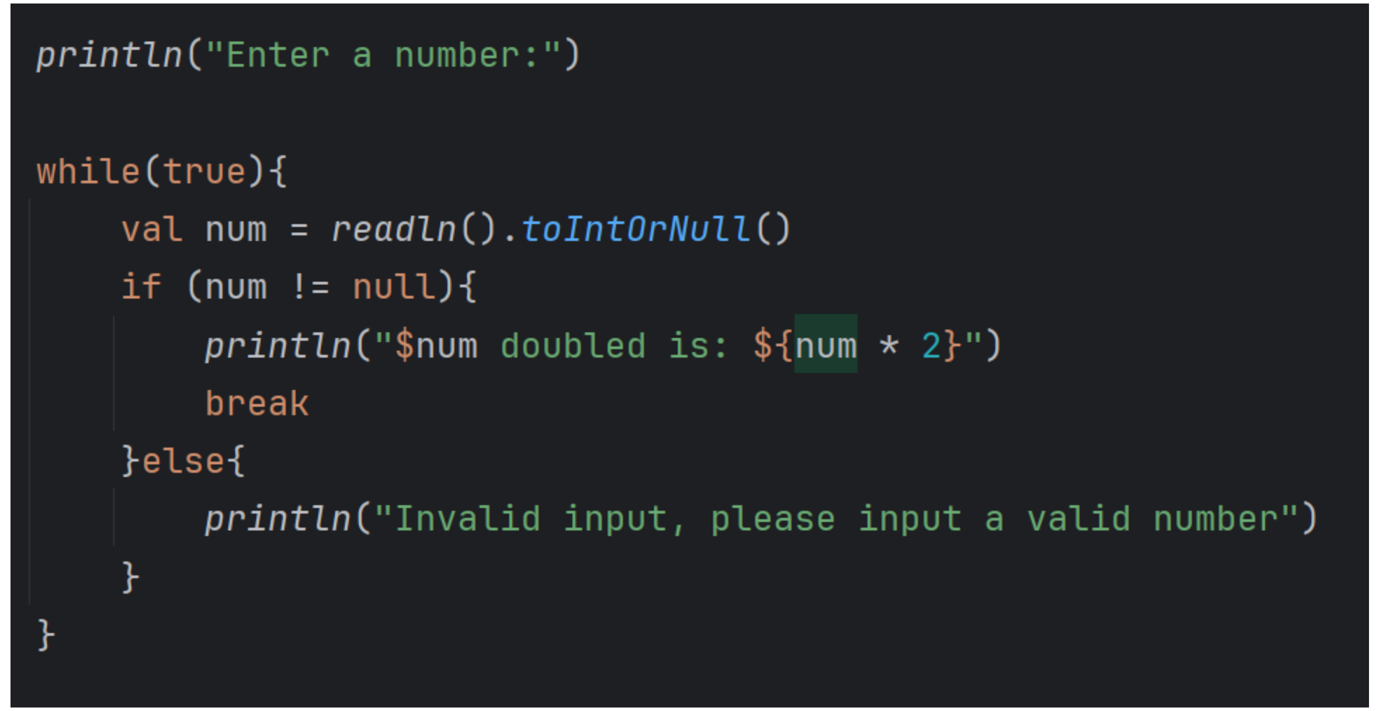


```
fun main() {
    print("Enter some text: ")
    val input = readln()
    print("You entered: $input")
}
```

- *readln* 返回一个 *String*

Input (cont.)

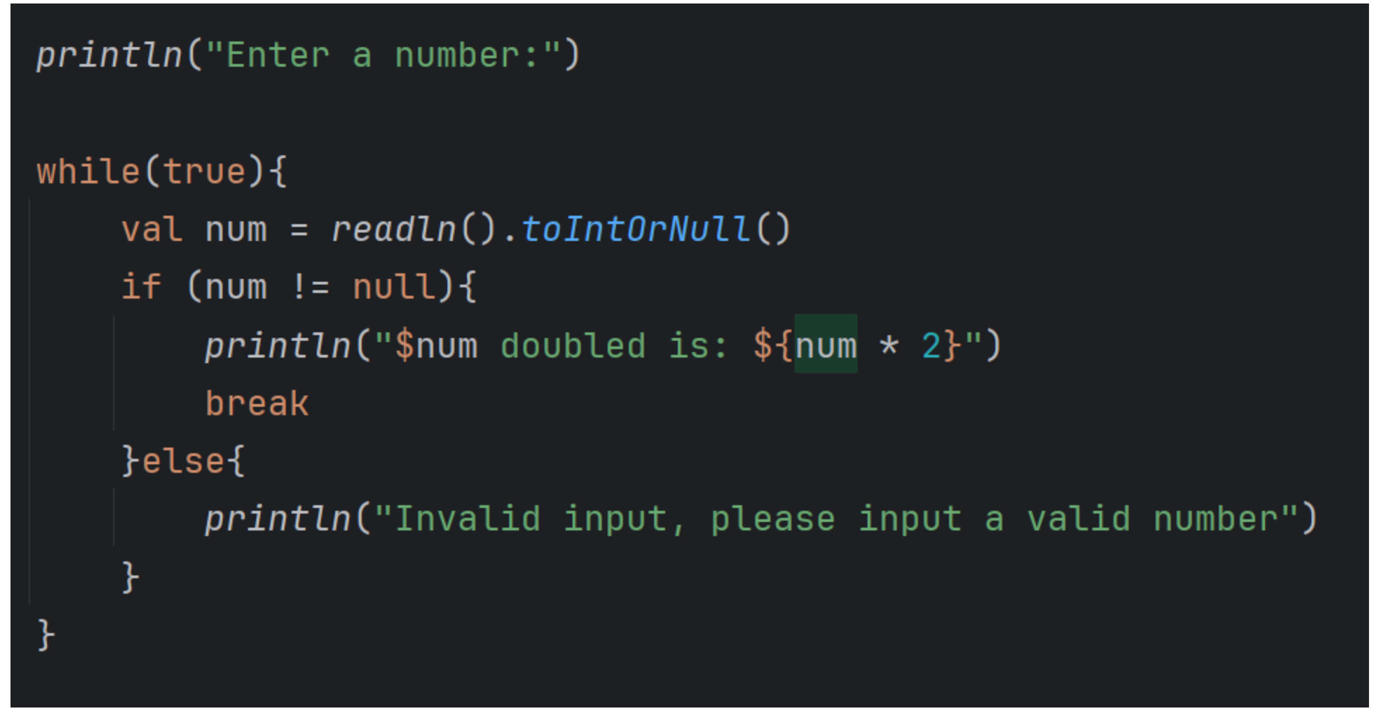
- Here is a simple program that checks for a number and doubles it



```
println("Enter a number:")

while(true){
    val num = readln().toIntOrNull()
    if (num != null){
        println("$num doubled is: ${num * 2}")
        break
    }else{
        println("Invalid input, please input a valid number")
    }
}
```

输入（续）

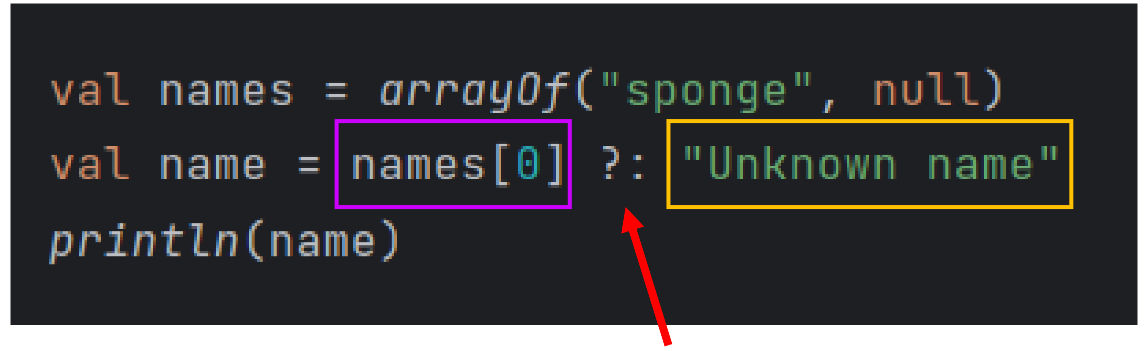
- 这是一个检查一个数字并将其加倍的简单程序



```
println("Enter a number:")

while(true){
    val num = readln().toIntOrNull()
    if (num != null){
        println("$num doubled is: ${num * 2}")
        break
    }else{
        println("Invalid input, please input a valid number")
    }
}
```

Elvis operator

- The Elvis operator ?: is helpful when working with nullable types

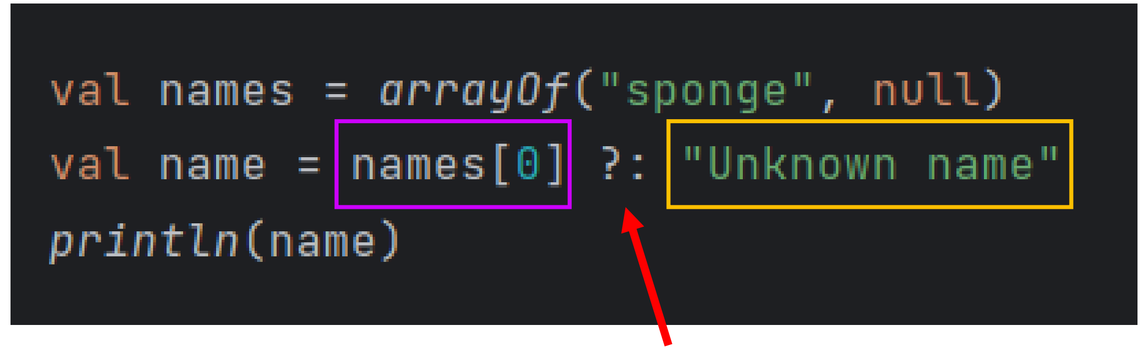


```
val names = arrayOf("sponge", null)
val name = names[0] ?: "Unknown name"
println(name)
```

- The first side of the expression will be returned if it is not null
- The second side of the expression will be returned if it is null

Elvis 操作符

- 空值合并运算符 ?: 在处理可空类型时非常有用



```
val names = arrayOf("sponge", null)
val name = names[0] ?: "Unknown name"
println(name)
```

- 如果 表达式的第一个部分 不为 null，则返回该部分
- 如果 表达式的第二个部分 为 null，则返回该部分