НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming

(p.4 – control flow, loops)

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

Loops

- Loops are Kotlin's way of executing code multiple times. In this lecture, you'll learn about three types of loops: the while loop, the do-while loop, for loop.
- If you know another programming language, you'll find the concepts and maybe even the syntax to be familiar.

While loop

- A while loop repeats a block of code while a condition is true. You create a while loop this way:

```
while (<CONDITION>) {
    <LOOP CODE>
}
```

While loop

- The loop checks the condition for every iteration.
- If the condition is true, then the loop executes and moves on to another iteration.
- If the condition is false, then the loop stops.
- Just like if expressions, while loops introduce a scope.
- The simplest while loop takes this form:

```
while (true) {
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

While loop

- The simplest while loop takes this form:

```
while (true) {
}
```

- This is a while loop that never ends because the condition is always true. Of course, you would never write such a while loop, because your program would spin forever!
- This situation is known as an **infinite loop**, and while it might not cause your program to crash, it will very likely cause your computer to freeze.

While loop

- Here's a more useful example of a while loop:

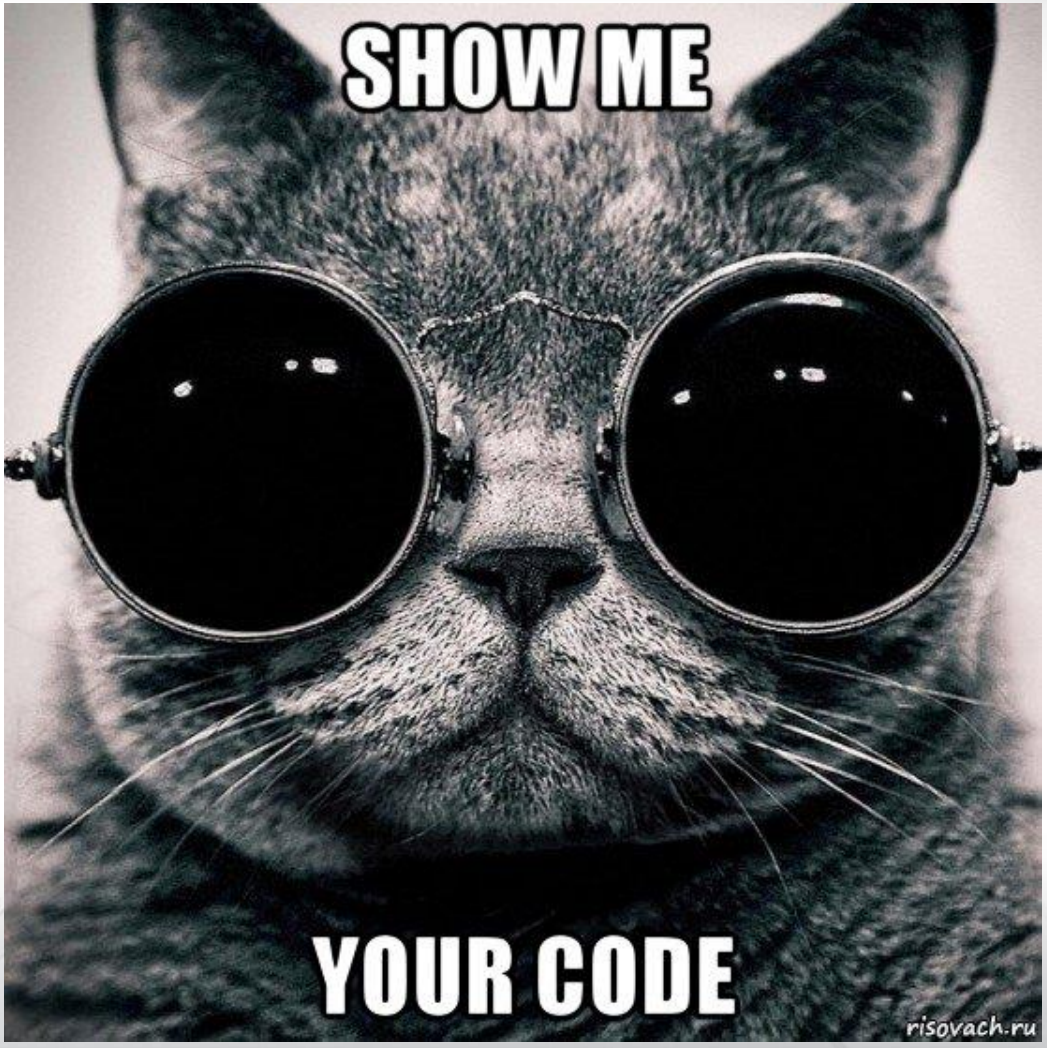
```
var sum = 1
while (sum < 1000) {
    sum = sum + (sum + 1)
}
```

This code calculates a mathematical sequence, up to the point where the value is greater than 1000.

What value will be in sum variable after this code was executed?

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Let’s code!



do-while loop

- A variant of the while loop is called the **do-while loop**.
- It differs from the while loop in that the condition is evaluated *at the end* of the loop rather than at the beginning.
- You construct a do-while loop like this:

```
do {
     <LOOP CODE>
} while (<CONDITION>)
```

do-while loop

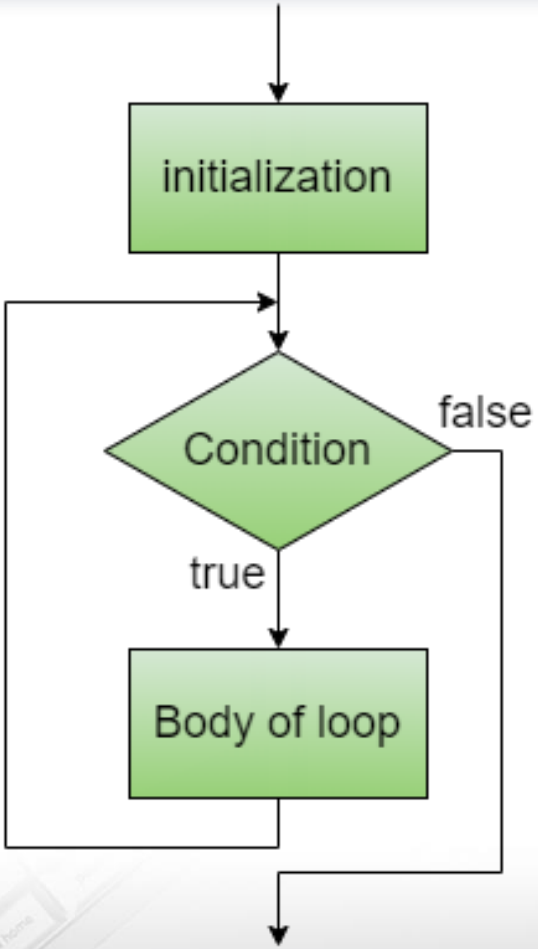
- Here's the example from the last section, but using a do-while loop:

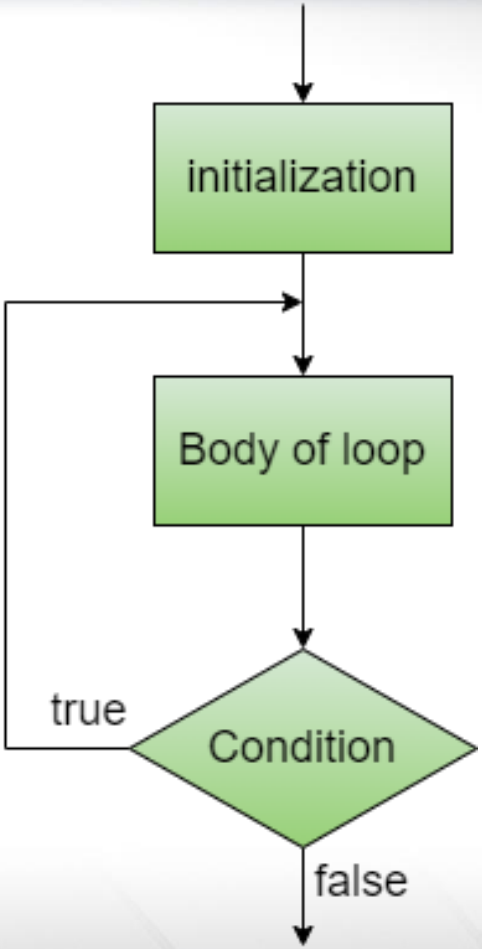
```
sum = 1
do {
     sum = sum + (sum + 1)
} while (sum < 1000)
```

In this example, the outcome is the same as before. However, that isn't always the case; you might get a different result with a different condition.

Why? And when?

while vs do-while





Breaking out of a loop

- Sometimes you want to break out of a loop early.
- You can do this using the break statement, which immediately stops the execution of the loop and continues on to the code after the loop.
- For example, consider the following code:

```
sum = 1
while (true) {
    sum = sum + (sum + 1)
    if (sum >= 1000) {
        break
    }
}
```

For loops

- This is probably the most common loop you'll see, and you'll use it to run code a certain number of times.
- You construct a for loop like this:

```
for (<CONSTANT> in <RANGE>) {
    <LOOP CODE>
}
```

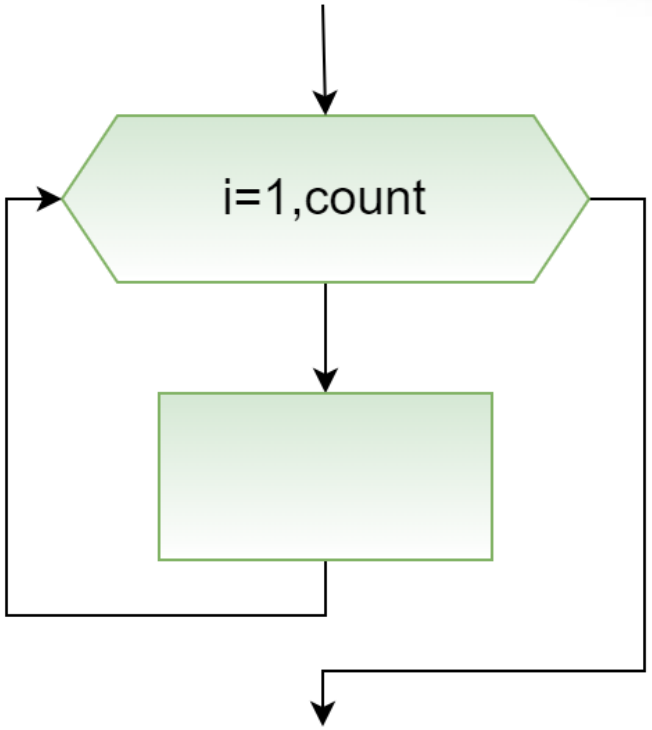
For loops

- The loop begins with the for keyword, followed by a name given to the loop constant (more on that shortly), followed by in, followed by the range to loop through. Here's an example:

```
val count = 10
var sum = 0
for (i in 1..count) {
    sum += i
}
```

In the code above, the for loop iterates through the range 1 to count. At the first iteration, i will equal the first element in the range: 1. Each time around the loop, i will increment until it's equal to count; the loop will execute one final time and then finish.

For loops - Flowchart



Notes about for loop

- If you'd used a half-open range, the last iteration would see i equal to count - 1.
- In terms of scope, the i constant is only visible inside the scope of the for loop, which means it's not available outside of the loop.

repeat loop

- Sometimes you only want to loop a certain number of times, and so you don't need to use the loop constant at all.
- In that case, you can employ a repeat loop, like so:

```
sum = 1
var lastSum = 0
repeat(10) {
    val temp = sum
    sum += lastSum
    lastSum = temp
}
```

for loops

- It’s also possible to only perform certain iterations in the range.
- For example, imagine you wanted to compute a sum similar to that of triangle numbers, but only for odd numbers:

```
sum = 0
for (i in 1..count step 2) {
    sum += i
}
```

for loops

- You can even count down in a for loop using downTo.
- In this case if count is 10 then the loop will iterate through the following values (10, 8, 6, 4, 2).

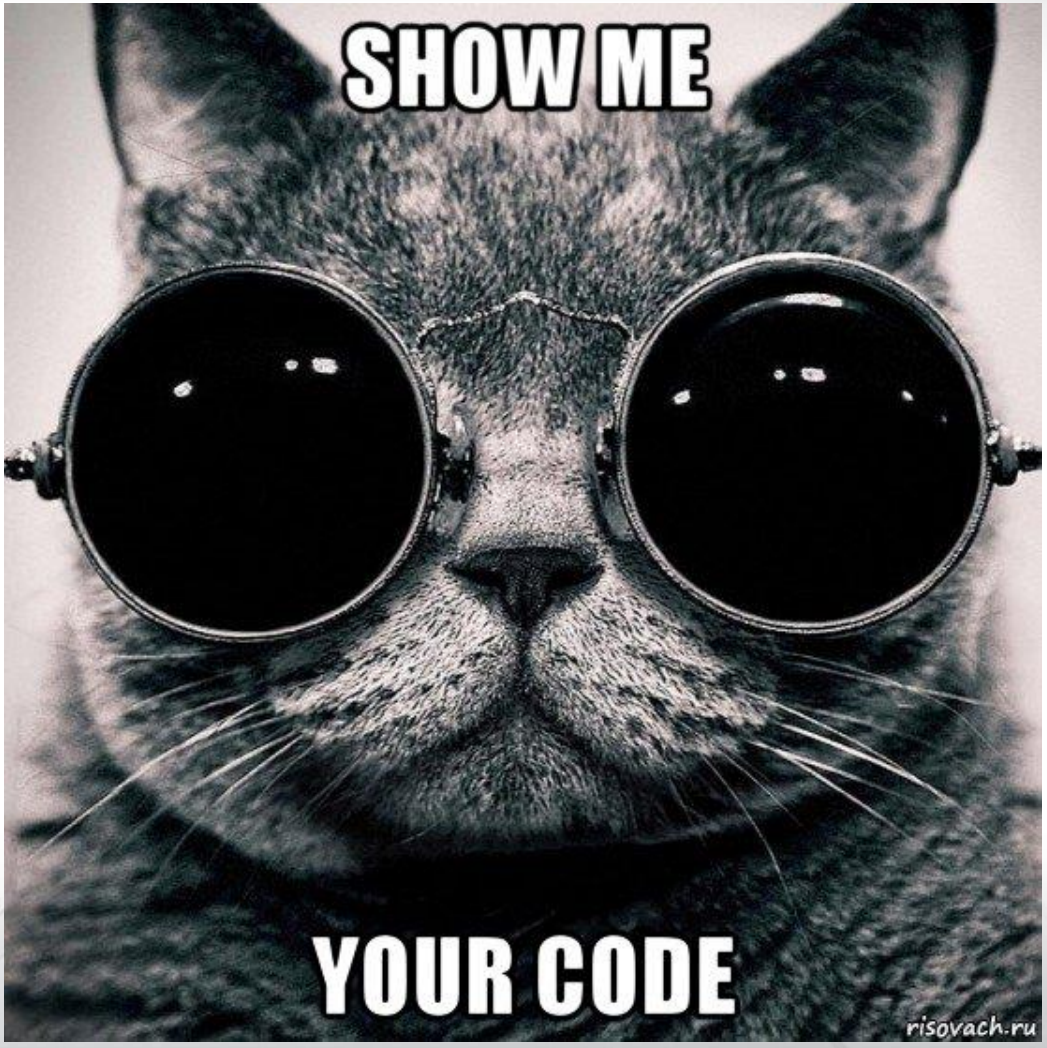
```
sum = 0
for (i in count downto 1 step 2) {
    sum += i
}
```

continue statement

- Sometimes you'd like to skip a loop iteration for a particular case without breaking out of the loop entirely.
- You can do this with the continue statement, which immediately ends the current iteration of the loop and starts the next iteration.
- The continue statement gives you a higher level of control, letting you decide where and when you want to skip an iteration.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Let’s code!



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Questions?

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming

(p.4 – control flow, loops)

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua