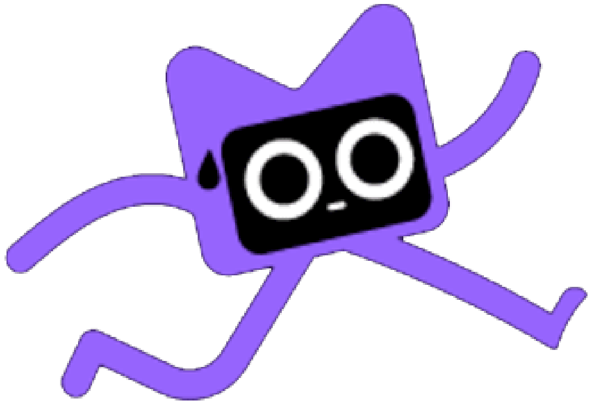
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming

(p.2 - functions)



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

Functions

- Functions are a core part of many programming languages.
- A function lets you define a block of code that performs a task.
- Then, whenever your app needs to execute that task, you can run the function instead of having to copy and paste the same code everywhere.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Function basics

- Imagine you have an app that frequently needs to print your name.
- You can write a function to do this:

```
fun printMyName() {
    println("My name is Eugeny")
}
```

The code above is known as a function declaration. You define a function using the **fun** keyword.

With your function defined, you can use it like so:

```
printMyName()
```

This prints out the following:

```
My name is Eugeny
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Function parameters

- Sometimes you want to **parameterize** your function, which lets the function perform differently depending on the data passed into it via its **parameters**.
- As an example, consider the following function:

```
fun printMultipleOfFive(value: Int) {
    println("$value * 5 = ${value * 5}")
}
```

And after declaring it, you can use this function, as shown here:

```
printMultipleOfFive(10)
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Function parameters

- In any function, the parentheses contain what’s known as the **parameter list**.
- These parentheses are required both when declaring and when invoking the function, even if the parameter list is empty.
- In the example, you call the function with an **argument** of 10

So, as result, you can see:

```
10 * 5 = 50
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Function parameters

- Take care not to confuse the terms "**parameter**" and "**argument**".
- A function declares its ***parameters*** in its parameter list.
- When you call a function, you provide values as ***arguments*** for the functions parameters.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Function parameters

You can take this one step further and make the function more general. With two parameters, the function can print out a multiple of any two values.

```
fun printMultipleOf(multiplier: Int, andValue: Int) {
    println("$multiplier * $andValue = ${multiplier * andValue}")
}
```

Than you can call it with line

```
printMultipleOf(4, 2)
```

There are now two parameters inside the parentheses after the function name: one named `multiplier` and the other named `andValue`, both of type `Int`

Named arguments

- Sometimes it is helpful to use **named arguments** when calling a function to make it easier to understand the purpose of each argument

```
printMultipleOf(multiplier = 4, andValue = 2)
```

This is especially helpful when a function has several parameters

Default values

- You can also give **default values** to parameters:

```
fun printMultipleOf(multiplier: Int, value: Int = 1) {
    println("$multiplier * $value = ${multiplier * value}")
}


printMultipleOf(4)
```

The difference is the = 1 after the second parameter, which means that if no value is provided for the second parameter, it defaults to 1.
Therefore, this code prints the following:

```
4 * 1 = 4
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Return values

- You can use a function to manipulate data. You simply take in data through parameters, manipulate it and then return it.
- Here's how you define a function that returns a value:

```
fun multiply(number: Int, multiplier: Int): Int {
    return number * multiplier
}
```

Inside the function, you use a return statement to return the value. In this example, you return the product of the two parameters.

Using of Pair

- It’s also possible to return multiple values through the use of Pairs:

```
fun multiplyAndDivide(number: Int, factor: Int): Pair<Int, Int> {
    return Pair(number * factor, number / factor)
}

val (product, quotient) = multiplyAndDivide(4, 2)
```

This function returns *both* the product and quotient of the two parameters by returning a Pair containing two Int values.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Function in expression

- If a function consists solely of a single expression, you can assign the expression to the function using = while at the same time not using braces, a return type, or a return statement:

```
fun multiplyInferred(number: Int, multiplier: Int) =
                                         number * multiplier
```

In such a case, the type of the function return value is *inferred* to be the type of the expression assigned to the function.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Parameters as values

- Function parameters are constants by default, which means they can’t be modified.

```
fun incrementAndPrint(value: Int) {
    value += 1
    print(value)
}
```

And result will be:

```
val cannot be reassigned
```

Parameters as values

- If you want a function to alter a parameter and return it, you must do so indirectly by declaring a new variable like so:

```
fun incrementAndPrint(value: Int): Int {
    val newValue = value + 1
    println(newValue)
    return newValue
}
```

Overloading

- What if you want more than one function with the same name?

```
fun getValue(value: Int): Int {
    return value + 1
}

fun getValue(value: String): String {
    return "The value is $value"
}
```

This is called **overloading** and lets you define similar functions using a single name.

Overloading

- The compiler must still be able to tell the difference between these functions within a given scope.
- Whenever you call a function, it should always be clear which function you’re calling.
- This is usually achieved through a difference in the parameter list:
  - A different number of parameters.
  - Different parameter types.

Note: The return type alone is not enough to distinguish two functions.

Functions as variables

- Functions in Kotlin are simply another data type.
- You can assign them to variables and constants just as you can any other type of value, such as an Int or a String.

```
fun add(a: Int, b: Int): Int {
    return a + b
}
```

This function takes two parameters and returns the sum of their values. You can assign this function to a variable using the **method reference operator**, ::, like so:

```
var function = ::add
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Functions as variables

- The fact that you can assign functions to variables comes in handy because it means you can pass functions to other functions.
- Here's an example of this in action:

```
fun printResult(function: (Int, Int) -> Int, a: Int, b: Int) {
    val result = function(a, b)
    print(result)
}


printResult(::add, 4, 2)
```

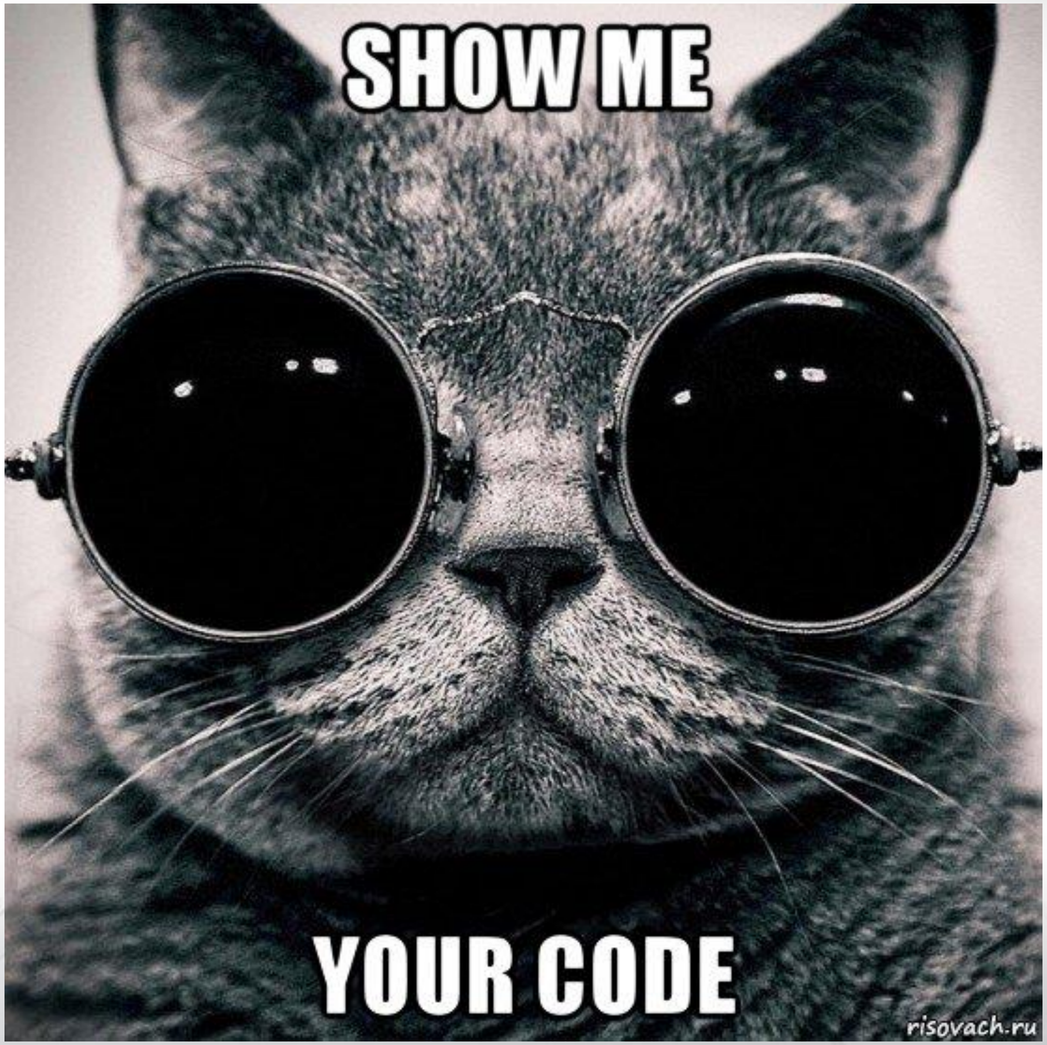
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Writing good functions

- The best (easiest to use and understand) functions do *one simple task* rather than trying to do many.
- This makes them easier to mix and match and assemble into more complex behaviors.
- Good functions also have a well defined set of inputs that produce the same output every time.
- This makes them easier to reason about and test in isolation.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Let's code!



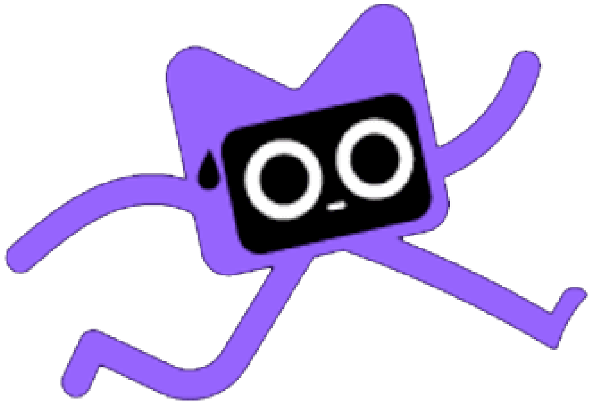
НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ
КОРАБЛЕБУДУВАННЯ
ІМЕНІ АДМІРАЛА МАКАРОВА

Questions?

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming

(p.2 - functions)



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