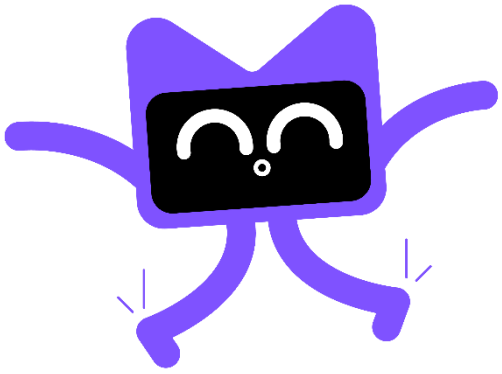
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming

(p.3 – control flow)



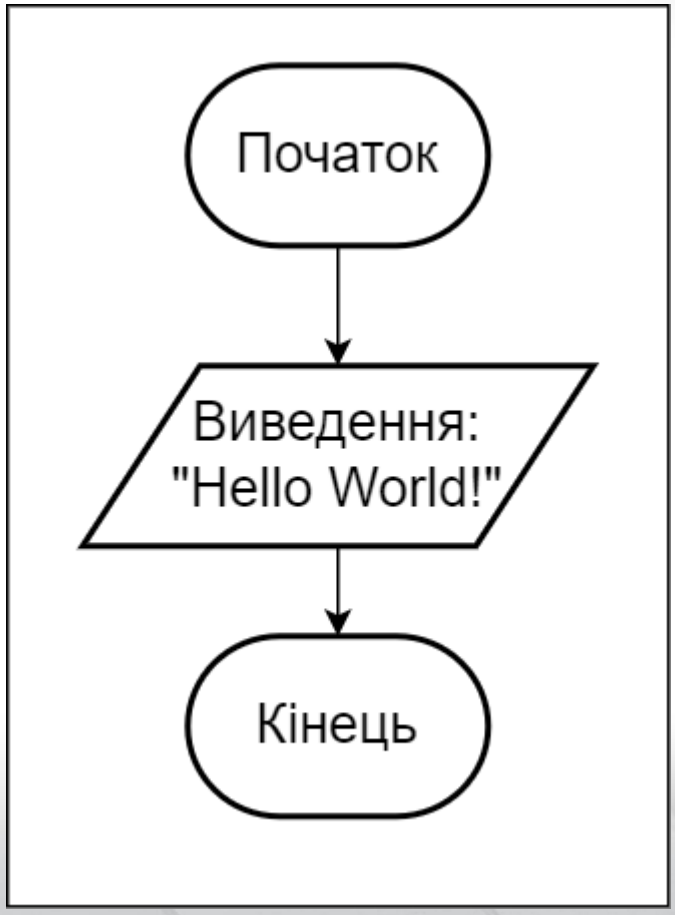
Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

But before we start...

- Flowcharts – what are they?

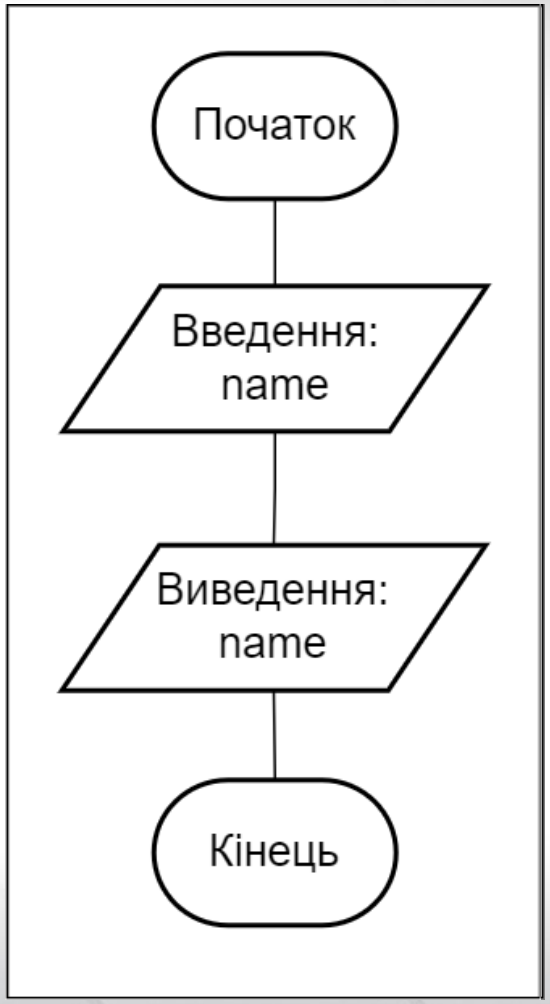
```
fun main() {
    println("Hello World!")
}
```



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Flowcharts

```
fun main() {
    print("Enter your name:")
    var name = readln()

    println("Hello $name!")
}
```



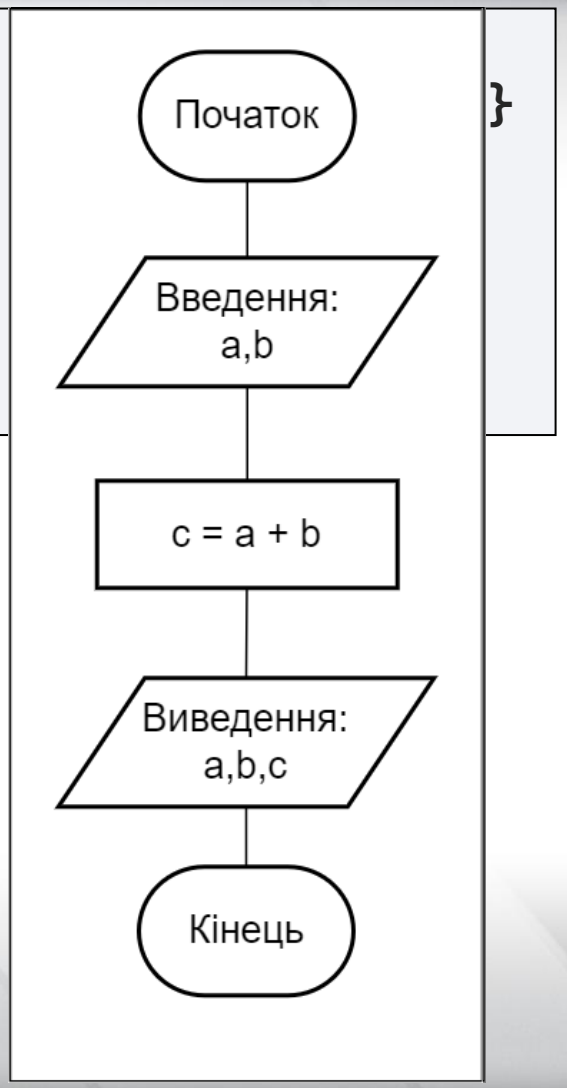
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Flowcharts

```
fun main() {
    val (a,b) = readln()
        .split(" ")
        .map { it.toDouble() }

    val c = a + b

    println("$a + $b = $c")
}
```

```
" ").map
```

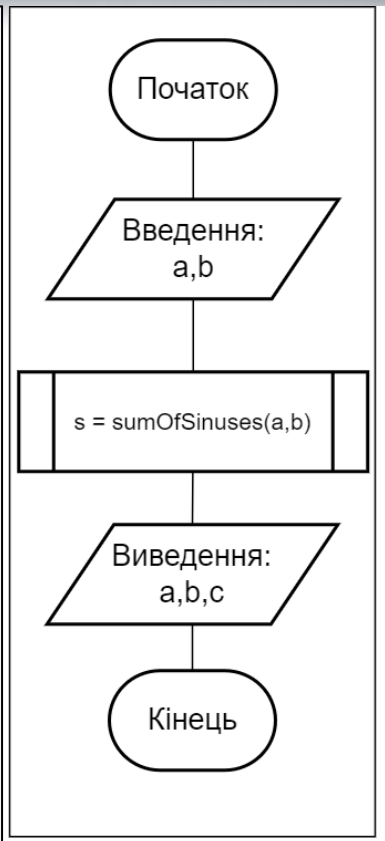
```
}
```

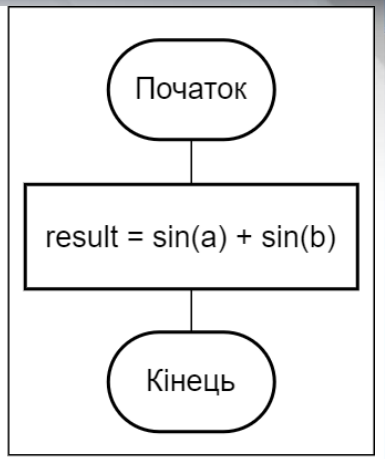


НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Flowcharts

```
import kotlin.math.sin

fun main() {
    val (a,b) = readln()
        .split(" ")
        .map { it.toDouble() }

    val s = sumOfSinuses(a, b)

    println("sum of sinuses = $s")
}

fun sumOfSinuses(
        a: Double,
        b: Double): Double {

    return sin(a) + sin(b)
}
```





Control flow

- When writing a computer program, you need to be able to tell the computer what to do in different scenarios.
- For example, a calculator app would need to do one thing if the user tapped the addition button and another thing if the user tapped the subtraction button

In computer-programming terms, this concept is known as **control flow**.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Comparison operators

- When you perform a comparison, such as looking for the greater of two numbers, the answer is either *true* or *false*.
- Kotlin has a data type just for this! It's called a Boolean
- This is how you use a Boolean in Kotlin:

```
val yes: Boolean = true
val no: Boolean = false
```

And because of Kotlin's type inference, you can leave off the type annotation:

```
val yes = true
val no = false
```

Boolean operators

- Booleans are commonly used to compare values. For example, you may have two values and you want to know if they're equal: either they are (true) or they aren't (false).
- In Kotlin, you do this using the **equality operator**, which is denoted by ==:

```
val doesOneEqualTwo = (1 == 2)          // false
```

Similarly, you can find out if two values are *not* equal using the != operator:

```
val doesOneNotEqualTwo = (1 != 2)       // true
```

Boolean operators

- The prefix ! operator, also called the not-operator, toggles true to false and false to true. Another way to write the above is:

```
val alsoTrue = !(1 == 2)
```

- Two more operators let you determine if a value is greater than (>) or less than (<) another value. You'll likely know these from mathematics:

```
val isOneGreaterThanTwo = (1 > 2)
val isOneLessThanTwo = (1 < 2)
```

Boolean operators

- There's also an operator that lets you test if a value is less than *or* equal to another value: <=.
- It's a combination of < and ==, and will therefore return true if the first value is either less than the second value or equal to it.
- Similarly, there's an operator that lets you test if a value is greater than or equal to another — you may have guessed that it's >=.

Boolean logic

- One way to combine conditions is by using **AND**.
- When you **AND** together two Booleans, the result is another Boolean.
- If **both** input Booleans are **true**, then the result is **true**. Otherwise, the result is **false**.

```
val and = true && true
```

In this case, **and** will be true.
If either of the values on the right was false, then and would be false.

Boolean logic

- Another way to combine conditions is by using **OR**.
- When you **OR** together two Booleans, the result is **true** if ***either*** of the input Booleans is **true**.
- Only if ***both*** input Booleans are **false** will the result be **false**.

```
val or = true || false
```

In this case, **or** will be true. If both values on the right were false, then or would be false. If both were true, then or would still be true.

Boolean logic

- In Kotlin, Boolean logic is usually applied to multiple conditions.
- Maybe you want to determine if two conditions are true; in that case, you'd use AND.
- If you only care about whether one of two conditions is true, then you'd use OR.
- For example, consider the following code:

```
val andTrue = 1 < 2 && 4 > 3
val andFalse = 1 < 2 && 3 > 4
val orTrue = 1 < 2 || 3 > 4
val orFalse = 1 == 2 || 3 == 4
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Boolean logic

- It's also possible to use Boolean logic to combine more than two comparisons.
- For example, you can form a complex comparison like so:

```
val andOr = (1 < 2 && 3 > 4) || 1 < 4
```

The parentheses disambiguates the expression.
First Kotlin evaluates the subexpression inside the parentheses, and then it evaluates the full expression, following these steps:

```
1. (1 < 2 && 3 > 4) || 1 < 4
2. (true && false) || true
3. false || true
4. true
```

Short circuit

- What about evaluation of this expression?

```
val orAnd = 1 < 4 || (1 < 2 && 3 > 4)
```

After evaluating first subexpression, it's not need to evaluate last one, because all result is determined. This called "short-circuit evaluation"

String equality

- Sometimes you want to determine if two strings are equal.
- For example, a children's game of naming an animal in a photo would need to determine if the player answered correctly.
- In Kotlin, you can compare strings using the standard equality operator "==" in exactly the same way as you compare numbers. For example:

```
val guess = "dog"
val dogEqualsCat = guess == "cat"
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

String comparision

- Just as with numbers, you can compare not just for equality, but also to determine if one value is greater than or less that another value. For example:

```
val order = "cat" < "dog"
```

- This syntax checks if one string comes before another alphabetically.
- In this case, order equals true because "cat" comes before "dog".

The if expression

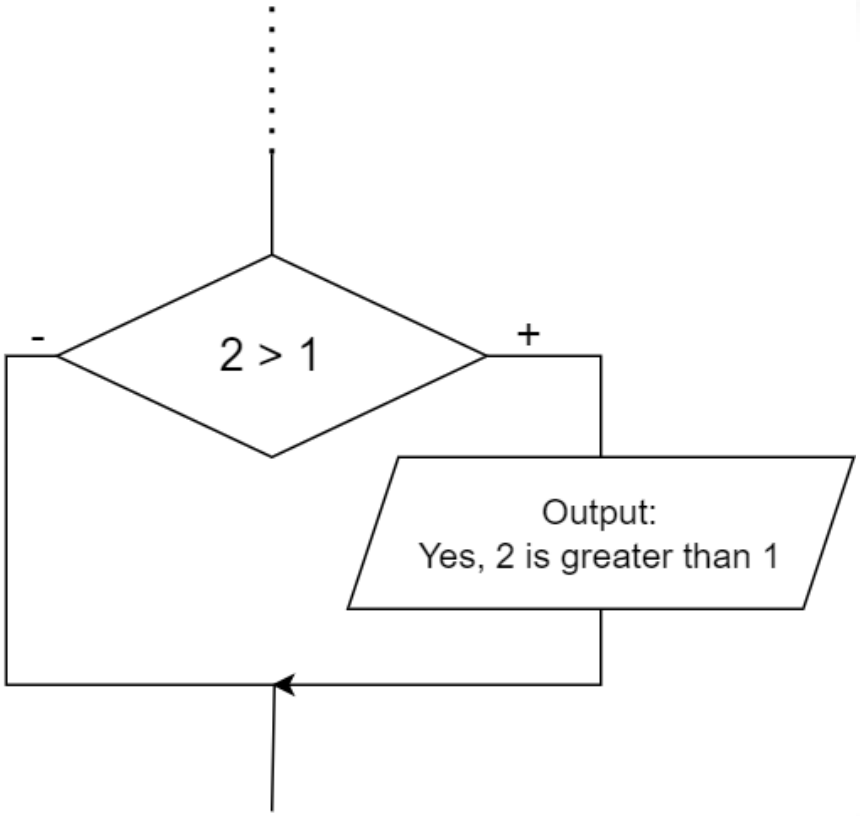
- The first and most common way of controlling the flow of a program is through the use of an **if expression**, which allows the program to do something only *if* a certain condition is true.
- For example, consider the following:

```
if (2 > 1) {
    println("Yes, 2 is greater than 1")
}
```

The term **if expression** is used here instead of **if statement**, since, unlike many other programming languages, a value is returned from the if expression. The value returned is the value of the last expression in the if block.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

The if expression



The if expression

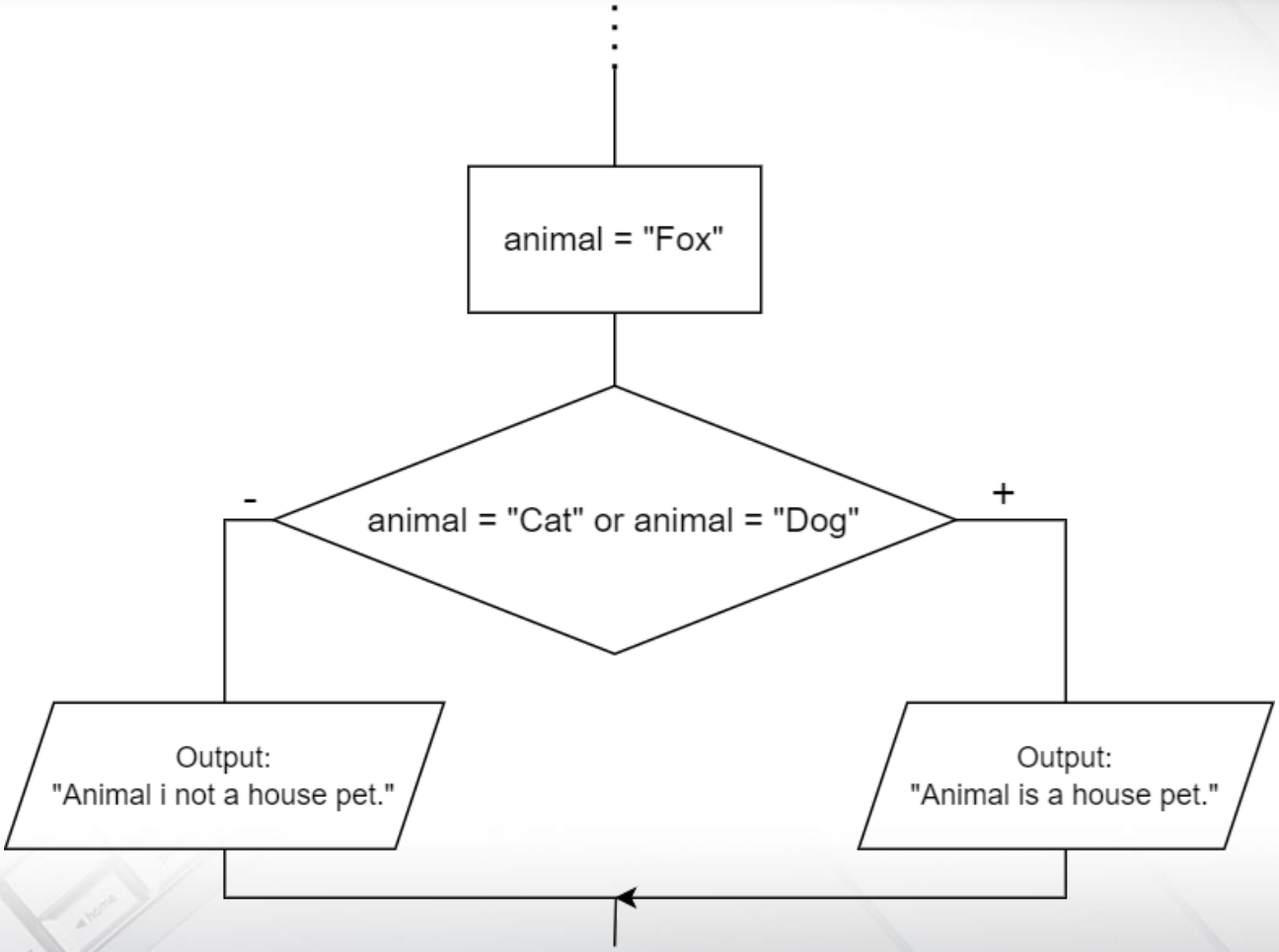
- You can extend an if expression to provide code to run in case the condition turns out to be false.
- This is known as the **else clause**. Here's an example:

```
val animal = "Fox"
if (animal == "Cat" || animal == "Dog") {
    println("Animal is a house pet.")
} else {
    println("Animal is not a house pet.")
}
```

```
Animal is not a house pet.
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

The if expression



The if expression

- You can also use an if-else expression on one line.
- If you wanted to determine the minimum and maximum of two variables, you *could* use if expressions like so:

```
val a = 5
val b = 10
val min = if (a < b) a else b
val max = if (a > b) a else b
```

```
val a = 5
val b = 10
val min: Int
if (a < b) {
    min = a
} else {
    min = b
}
val max: Int
if (a > b) {
    max = a
} else {
    max = b
}
```

The if expression

- Sometimes you want to check one condition, then another. This is where **else-if** comes into play, nesting another if clause in the **else** clause of a previous if clause.
- You can use else-if like so:

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

The if expression

- You can use else-if like so:

```
val hourOfDay = 12
val timeOfDay = if (hourOfDay < 6) {
    "Early morning"
} else if (hourOfDay < 12) {
    "Morning"
} else if (hourOfDay < 17) {
    "Afternoon"
} else if (hourOfDay < 20) {
    "Evening"
} else if (hourOfDay < 24) {
    "Late evening"
} else {
    "INVALID HOUR!"
}
println(timeOfDay)
```

Short circuiting

- An important fact about if expressions and the Boolean operators is what happens when there are multiple Boolean conditions separated by ANDs (&&) or ORs (||).
- Consider the following code:

```
if (1 > 2 && name == "Ivan Ivanov") {
    // ...
}
```

- The first condition of the if expression, 1 > 2 is false.
- Therefore the whole expression cannot ever be true.
- So Kotlin will not even bother to check the second part of the expression, namely the check of name.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Short circuiting

- Similarly, consider the following code:

```
if (1 < 2 || name == "Ivan Ivanov") {
    // ...
}
```

- Since 1 < 2 is true, the whole expression must be true as well.
- Therefore once again, the check of name is not executed.
- This will come in handy later on when you start dealing with more complex data types.

Encapsulating variables

- if expressions introduce a new concept **scope**, which is a way to encapsulate variables through the use of braces.
- Imagine you want to calculate the fee to charge your client. Here's the deal you've made:

> You earn $25 for every hour up to 40 hours, and $50 for every hour thereafter.

Encapsulating variables

- Imagine you want to calculate the fee to charge your client. Here's the deal you've made:

You earn $25 for every hour up to 40 hours, and $50 for every hour thereafter.

Using Kotlin, you can calculate your fee in this way:

```
var hoursWorked = 45
var price = 0
if (hoursWorked > 40) {
    val hoursOver40 = hoursWorked - 40
    price += hoursOver40 * 50
    hoursWorked -= hoursOver40
}
price += hoursWorked * 25
println(price)
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Encapsulating variables

- The interesting thing here is the code inside the if expression. There is a declaration of a new constant, hoursOver40, to store the number of hours over 40.
- Clearly, you can use it inside the if statement. Since 1 < 2 is true, the whole expression must be true as well.
- Therefore once again, the check of name is not executed.
- This will come in handy later on when you start dealing with more complex data types. But what happens if you try to use it at the end of the above code?

```
println(price)
println(hoursOver40)
```

This would result in the following error:

```
Unresolved reference: 'hoursOver40'
```

when expressions

- You can also control flow via the when expression. It executes different code depending on the value of a variable or constant.
- Here's a when expression that acts on an integer:

```
val number = 10
when (number) {
    0 -> println("Zero")
    else -> println("Non-zero")
}
```

In this example, the code will print the following:

```
Non-zero
```

when expressions

- when expressions also work with data types other than integers.
- Here's an example using a string:

```
val string = "Dog"
when (string) {
    "Cat", "Dog" -> println("Animal is a house pet.")
    else -> println("Animal is not a house pet.")
}
```

This will print the following:

```
Animal is a house pet.
```

Returning values

- If you want to determine the name of the number, you can assign the value with a when expression as follows:

```
val numberName = when (number) {
    2 -> "two"
    4 -> "four"
    6 -> "six"
    8 -> "eight"
    10 -> "ten"
    else -> {
        println("Unknown number")
        "Unknown"
    }
}
println(numberName) // > ten
```

Advanced when expressions

- In the one of previous slides, you saw an if expression that used multiple else clauses to convert an hour of the day to a string describing that part of the day.
- You could rewrite that more succinctly with a when expression

Advanced when expressions

```
val hourOfDay = 12
val timeOfDay: String
timeOfDay = when (hourOfDay) {
    0, 1, 2, 3, 4, 5 -> "Early morning"
    6, 7, 8, 9, 10, 11 -> "Morning"
    12, 13, 14, 15, 16 -> "Afternoon"
    17, 18, 19 -> "Evening"
    20, 21, 22, 23 -> "Late evening"
    else -> "INVALID HOUR!"
}
println(timeOfDay)
```

Ranges

- Before you dive into the when expression, you need to know about the **range** data types, which let you represent a sequence of countable integers.
- Let's look at two types of ranges.
- First, there's a **closed range**, which you represent like so:

```
val closedRange = 0..5
```

The two dots (..) indicate that this range is closed, which means the range goes from 0 to 5 inclusive. That's the numbers (0, 1, 2, 3, 4, 5).

Ranges

- Second, there's a **half-open range**, which you represent like so:

```
val halfOpenRange = 0 until 5
```

- Here, you replace the two dots with until. Half-open means the range goes from 0 up to, but not including, 5. That's the numbers (0, 1, 2, 3, 4).
- Open and half-open ranges created with the .. and until operators are always increasing.
- In other words, the second number must always be greater than or equal to the first.

Ranges

- To create a decreasing range, you can use downTo, which is inclusive:

```
val decreasingRange = 5 downTo 0
```

- That will include the numbers (5, 4, 3, 2, 1, 0).
- Ranges are commonly used in both:
  - when expressions and
  - for loops

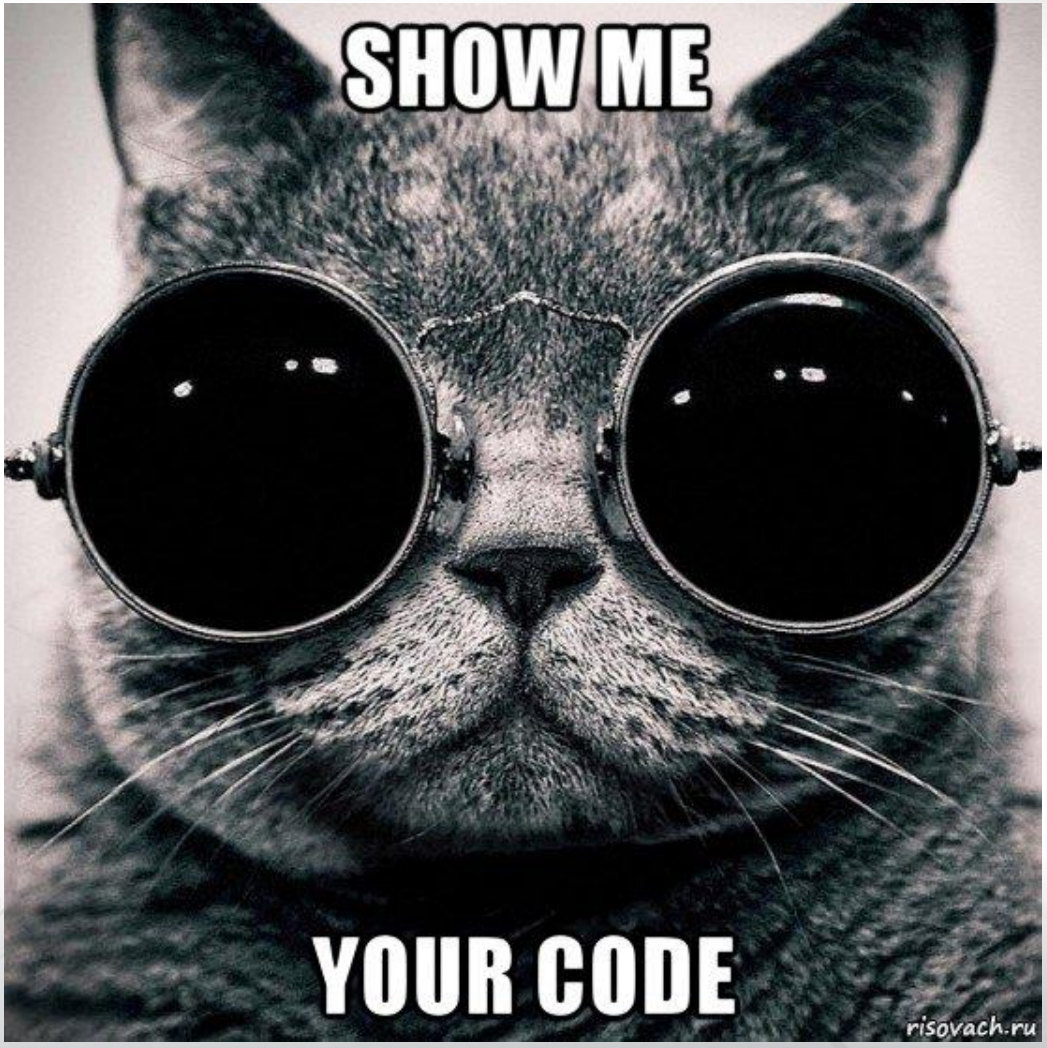
Advanced when expressions

- Well, you can use ranges to simplify this when expression. You can rewrite the above code using ranges:

```
timeOfDay = when (hourOfDay) {
    in 0..5 -> "Early morning"
    in 6..11 -> "Morning"
    in 12..16 -> "Afternoon"
    in 17..19 -> "Evening"
    in 20..23 -> "Late evening"
    else -> "INVALID HOUR!"
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Let’s code!

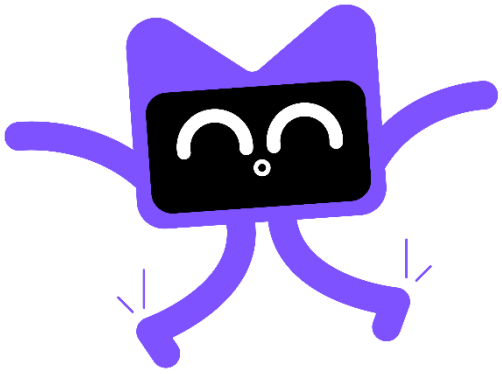


НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Questions?

Algorithms & Programming

(p.3 – control flow)



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