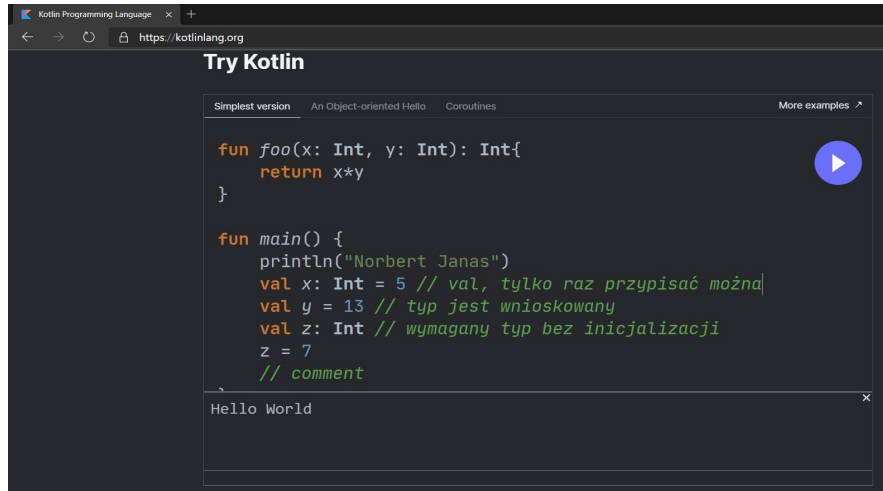


# Sprawozdanie o języku Kotlin

Norbert Janas

Po pierwszym spojrzeniu na kod “Hello World” w kotlinie rzuca się w oczy słowo kluczowe definiujące funkcje, natomiast podstawowa instrukcja wyjścia bardzo przypomina prostotę *pythona*.

Typy deklaruje się po dwukropku nawet podczas typu zwracanego przez funkcję. Same zmienne poprzedzone są słowami kluczowymi **val** lub **var**. (do var można ponownie przypisywać wartości)

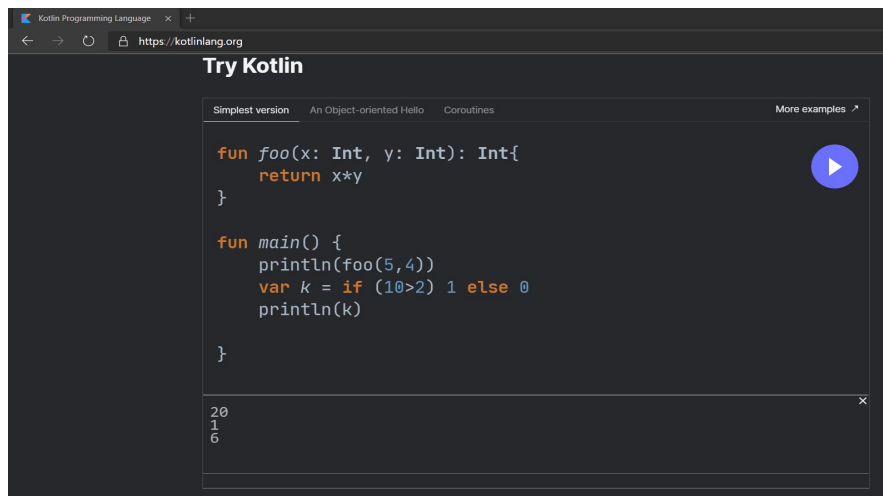


```
fun foo(x: Int, y: Int): Int {
    return x*y
}

fun main() {
    println("Norbert Janas")
    val x: Int = 5 // val, tylko raz przypisać można
    val y = 13 // typ jest wnioskowany
    val z: Int // wymagany typ bez inicjalizacji
    z = 7
    // comment
}
```

Hello World

If może być jako wyrażenie (podobnie np. w pythonie)

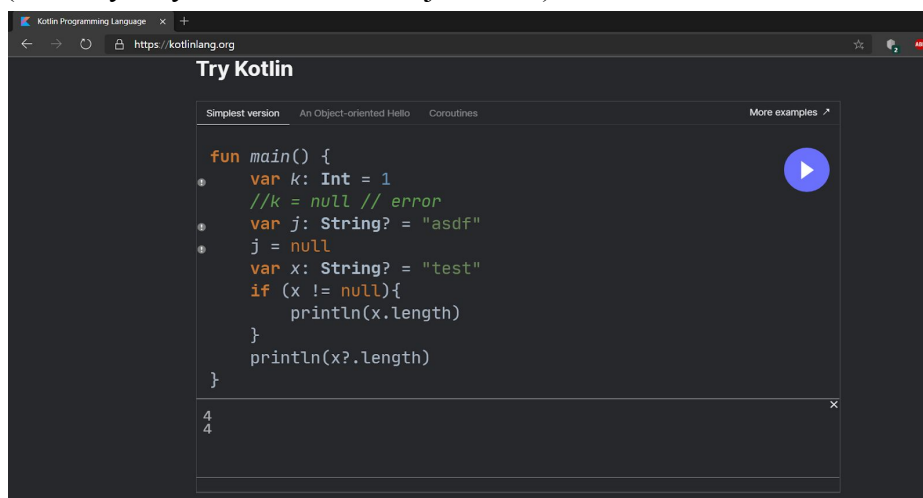


```
fun foo(x: Int, y: Int): Int {
    return x*y
}

fun main() {
    println(foo(5,4))
    var k = if (10>2) 1 else 0
    println(k)
}
```

20  
1  
6

W kotlinie musimy jawnie określać, co może przetrzymywać null a co nie. Określamy to za pomocą ‘?’, jeśli coś jest null to musimy to jawnie sprawdzić, żeby nie powodować błędu (możemy użyć odwołania z ‘?.’ jak w C#)



```
fun main() {
    var k: Int = 1
    //k = null // error
    var j: String? = "asdf"
    j = null
    var x: String? = "test"
    if (x != null){
        println(x.length)
    }
    println(x?.length)
}
```

4  
4

Listy tworzymy z podziałem na stałe albo zmienne listy, w których typ przechowywanych elementów może zostać wywnioskowany. (Wygodna opcja wypisywania elementów listy za pomocą zwykłego printa jak w pythonie)

```
fun main() {
    val list = listOf(1, 2, 3) // Tylko do odczytu
    println(list[0])
    // list[0] = "test" // error
    val mutableList = mutableListOf<Int>(4, 5, 6, 7)
    val mutableList2 = mutableListOf("ab", "cd", "ef")
    mutableList2[2] = "test"
    println(mutableList2)
}
```

1  
[ab, cd, test]

Iteracja w kotlinie wygląda jak ‘for each’, posiadając również wbudowaną funkcjonalność iteracji jednocześnie po elementach i indeksach

(jak w *pythonie* `for idx, el in enumerate(list):` )

(Formatowanie łańcuchów znaków podobne jak w C#)

```
fun main() {
    val list = listOf(1, 2, 3, 4, 5)
    for ((idx, el) in list.withIndex()){
        println("i: $idx \t el: ${el}")
    }
}
```

i: 0 el: 1  
i: 1 el: 2  
i: 2 el: 3  
i: 3 el: 4  
i: 4 el: 5

Switch jest tutaj w formie ‘when’ i ma składnię opartą na ‘->’, a słowo default zastępuje else.

Podobnie do list tworzy się również tablice za pomocą `arrayOf`, jednak aby je wypisać musimy jawnie skorzystać z pakietu Javy używając metody `toString()`.

```
fun main() {
    val x = -2
    when (x) {
        0 -> println("zero")
        in 1..5 -> println("od 1 do 5")
        else -> println("default")
    }
    val k = arrayOf(1, 2, 3, 4)
    println(k)
    println(java.util.Arrays.toString(k))
}
```

default  
[Ljava.lang.Integer;@5a07e868  
[1, 2, 3, 4]

Szybsza inicjalizacja tablicy za pomocą 'it' odwołania do indeksu tablicy  
(wyrażenia lambda w klamrach podobnie jak C#)

```
fun main() {  
    val a = Array<Int>(6) {it+5} // it -> index  
    println(java.util.Arrays.toString(a))  
    // Lambda  
    val square = {x: Int -> x*x}  
    val suma: (Int, Int) -> Int = {a, b -> a+b}  
    println(square(5))  
    println(suma(2, 3))  
}
```

Bardzo przydatna funkcjonalność List Comprehension (python)

```
fun main() {  
    val suma: (Int, Int) -> Int = {a, b -> a+b}  
    println(square(5))  
    println(suma(2, 3))  
    // List Comprehension  
    val numbers = Array<Int>(10) {it+1}  
    val even = numbers  
        .filter {it%2 == 0}  
        .map {it}  
    println(even)  
}
```

```
[5, 6, 7, 8, 9, 10]  
25  
5  
[2, 4, 6, 8, 10]
```

Zasięgi zapisywane za pomocą '..', if może ich również używać. Składnia z *downTo* czy *step* do iteracji.

```
fun main() {  
    // Używanie zasięgów ..  
    for (i in 1..5){  
        println(i)  
    }  
    if (3 in 1..5) println("w zasięgu") else println("poza")  
    for (i in 5 downTo 1) print(i) // iteracja w dół  
    println()  
    for (i in 1..9 step 3) print(i) // określenie kroku  
    println()  
    for (l in 'a'..'f' step 2) print(l)  
}
```

```
1  
2  
3  
4  
5  
w zasięgu  
54321  
147  
ace
```