



# Scala

Wprowadzenie do języka.

# whoami

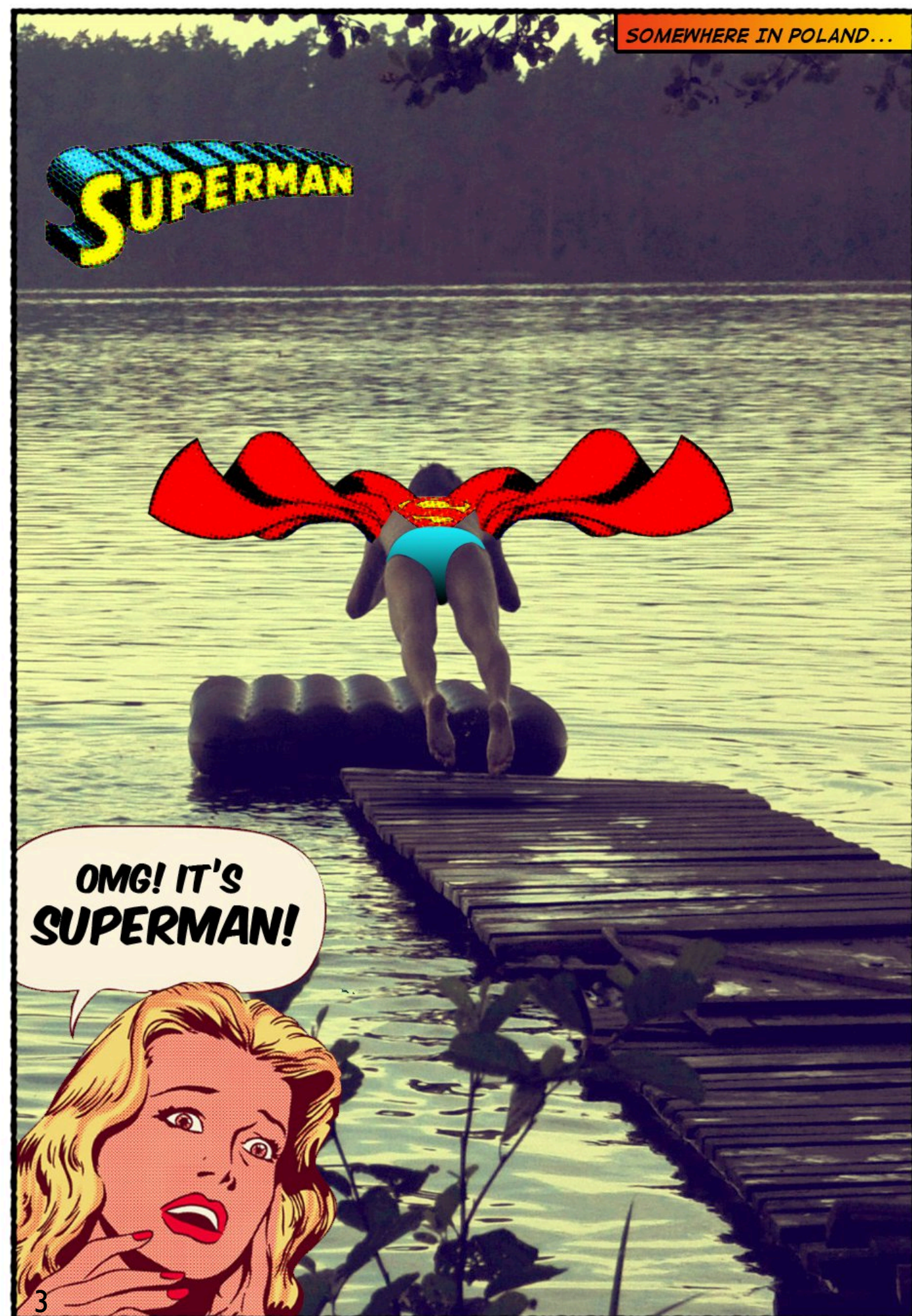
Michał Januszewski

[twitter.com/mjanuszewski](https://twitter.com/mjanuszewski)

[michal@mjanuszewski.pl](mailto:michal@mjanuszewski.pl)

# Other activities

\* [www.kropkapodi.pl](http://www.kropkapodi.pl)



# Krótką historia...

...czyli co, po co i dlaczego...



# Features (ficzery)

- Statyczne typowanie
- Wieloparadygmatowość
- JVM + ... .NET ;-)
- Zwięzła, elastyczna i elegancka składnia
- Inferencja typu
- Skalowalność
- Wydajność

# Static typing

- Typ zmiennej przechowywany w zmiennej przez cały cykl życia.
- Możliwość przechowywania obiektów tego samego typu lub pochodnych.
- Dlaczego nie dynamiczne?
  - słabsza wydajność
  - brak optymalizacji ze strony kompilatora
  - trudniejsza implementacja w IDE

# Paradygmaty

- Object-oriented - wszystko jest obiektem, brak typów podstawowych, traitsy oraz singletony.
- Functional - immutable variables (wielowątkowość), funkcje jako obiekty, domknięcia (ang. "closures").

# Platformy

- JVM
- .NET



# Inferencja typu

- Scala posiada wbudowany mechanizm inferencji typu, dzięki któremu nie jesteśmy zmuszani do definiowania typu zmiennej.
- Kompilator przypisuje zmiennej typ, na podstawie wartości przypisanej do tej zmiennej.

# Skalowalność

- Scala jest językiem skalowalnym, co oznacza, że doskonale nadaje się do pisania zarówno prostych skryptów, jak i rozbudowanych aplikacji.

# Jak zacząć?

- [www.scala-lang.org/downloads](http://www.scala-lang.org/downloads)
- REPL
- plugin do IDE (Intellij IDEA, Eclipse, etc.)

Show me the code!

# Zmienne

```
val s: String = "string"  
var s = "kolejny string"
```

```
lazy val i = 10  
val map = Map(1 -> "jeden", 2 -> "dwa")
```

**val** – odpowiednik javowego "final"  
**var** – zwykła zmienna

# Funkcje

```
def join(strings: List[String], sep: String): String =  
  strings.mkString(sep)
```

```
def show(list: String*) = for (x <- list) println(x)
```

```
val increase = (x: Int) => x + 1  
increase(10)
```

```
val sum(a: Int, b: Int, c: Int) = a + b + c
```

```
val otherSum = sum _ otherSum(2, 4, 5)
```

```
val one = sum(1, _:Int, 3)
```

```
val result = one(2)
```

# Inferencja typu

Java:

```
Map<Integer,String> mapa = new HashMap<Integer, String>();
```

Scala:

```
val mapa: Map[Integer,String] = new HashMap
```

```
val mapa2 = new HashMap[Integer,String]
```



# Pakiety

```
import scala.collection.mutable._  
  
import java.util.{Map, HashMap}  
  
package scala.collection {  
    import immutable._  
}  
  
def function() = {  
    import scala.collection.immutable.{ Stack => Stos,  
        TreeMap => _  
    }  
}
```

# Nowe typy

```
val s = """To jest przykładowy  
napis rozdzielony na kilka wierszy.  
Koniec """
```

```
val krotka = (1, "tekst", new Car)  
val nr = krotka._1  
val car = krotka._3  
val krotka2 = 1 -> "napis" // (1,"napis")
```

# Notacja operatorowa

```
def parzysta(x: Int) = (x % 2) == 0
```

```
List(1,2,3,4,5).filter(x => parzysta(x)).foreach(y => println(y))
```

```
List(1,2,3,4,5).filter(parzysta( _ )).foreach(println( _ ))
```

```
List(1,2,3,4,5) filter parzysta foreach println
```

# Instrukcje warunkowe

```
val y = 4
```

```
val x = if (y > 0) true else false
```

```
val file = new File("in.txt")
```

```
val filePath = if (file.exists()) {
```

```
    file.getAbsolutePath()
```

```
} else {
```

```
    file.createNewFile() file.getAbsolutePath()
```

```
}
```

# Pętla “for”

```
val fruits = List("apple", "orange", "banana")
```

```
for (fruit <- fruits) // for(String s : fruits)  
  println(fruit)
```

```
for {  
  fruit <- fruits  
  if fruit.startsWith("a")  
  if fruit.length == 5  
} println("Czy to jabłko?")
```

# Pętla “for” od środka

```
for (fruit <- fruits) println(fruit)
fruits.foreach( fruit => println( fruit ) )
```

```
for {
  fruit <- fruits
  if fruit.startsWith("a")
  if fruit.length == 5
} println("jabłko")
```

```
fruits.filter( x => x.startsWith("a") && x.length == 5).foreach
( x => println("jabłko") )
```

# Generator

```
for (i <- 1 to 5) println(i)  
for (j <- 1 until 10) println(i)
```

```
val owoce = for {  
    fruit <- List("apple", "orange", "banana")  
} yield fruit
```

```
owoce.foreach(println)
```



# Pattern matching

```
val x = List(2.0, "sesja", new Praca(), false)
for(elem <- x)
  elem match {
    case d: Double => println("Why?")
    case s: String => println("When?!")
    case p: Praca => println("Really?")
    case b: Boolean => println("uff...")
    case _ => println("Just kidding ;-)")
  }
```

# Pattern matching (cd)

```
val n1 = List(1,2,3,4)
```

```
val n2 = List(2,4,6,8)
```

```
val empty = List()
```

```
for (list <- List(n1, n2, empty)) {
```

```
  list match {
```

```
    case List(_, 2, _, _) => println("4 elementy i drugi to 2")
```

```
    case List(2, _*) => println("Lista z 2 na początku")
```

```
    case List(_*) => println("Zero lub więcej elementów")
```

```
  }
```

```
}
```

# Pattern matching (cd.)

```
val t1 = ("A", "B")
```

```
val t2 = ("B", "A")
```

```
for( t <- List(t1, t2))
```

```
  t match {
```

```
    case (x, y) if x == "A" => println("Krotka z A")
```

```
    case (x, y) => println("Inna krotka")
```

```
}
```

# Obsługa wyjątków

```
import java.util.Calendar
val then = null
val now = Calendar.getInstance()
try {
    now.compareTo(then)
} catch {
    case e: NullPointerException => ("Null!")
    case _ => ("Inny wyjątek")
} finally {
    println("Wszystko ok!")
}
```

# Traits

```
trait Car {  
  def start() = println("wrrruum")  
  def horn() = println("beeeep!")  
}  
trait Engine {  
  def accel() = println("Go faster!")  
}  
class GasCar extends Car with Engine {  
  def carType() = "GasCar"  
}  
val car = new GasCar  
car.start()  
car.accel()
```

# Klasy i obiekty

```
class Rational(val n: Int, val d: Int) {  
  def this(n: Int) = this(n, 1)  
  def + (that: Rational): Rational =  
    new Rational(n * that.d + that.n * d, d * that.d)  
  override def toString() = n + "/" + d  
  override def hashCode = 25 + (25 + n) + d  
}
```

```
object Rational {  
  def apply(n: Int, d: Int) = new Rational(n, d)  
  def apply(n: Int) = new Rational(n)  
}
```

# Case class

```
case class X(x: Int, y: String)
```

Możliwość tworzenia obiektów bez użycia "new":

```
val x = X(4, "cztery")
```

Parametry konstruktora otrzymują "val", więc tworzone są odpowiednie pola w klasie.

Kompilator automatycznie dodaje metody "toString()", "hashCode()" oraz "equals", więc można w prosty sposób porównywać obiekty



# Case class - PM

```
case class Osoba(imie: String, nazwisko: String, wiek: Double)
```

```
val zenek = Osoba("Zenek", "Augustyński", 21.4)
```

```
val kasia = Osoba("Kasia", "Bywalska", 24.3)
```

```
val gosia = Osoba("Gosia", "Cichociemna", 24.)
```

```
for(osoba <- List(zenek, kasia, gosia))
```

```
  osoba match {
```

```
    case Osoba(_, _, 24) => println("Gosia?")
```

```
    case Osoba("Kasia", _, _) => println("Jakaś Kasia")
```

```
}
```

# Kolekcje

krótki przegląd

# List

```
val l1 = List("apple", "orange", "mandarin")  
val l2 = "apple" :: "orange" :: "mandarin" :: Nil
```

```
val numbers = List(1,2,3,4,5)  
val sorted = l1.sort(_ < _)
```

```
val lengths = l2.map(_ .length )  
val filtered = l1.filter(_ .startsWith("o"))  
val sum = (0 /: numbers) { _ + _ }  
val product = numbers.foldLeft(1) { _ * _ }
```

# Map

```
import scala.collection.mutable._  
val map = Map[Int, String]()
```

```
map += List(1 -> "apple", 2 -> "orange")
```

```
map += (3 -> "banana")  
map.contains(2)  
val orange = map(2)
```

```
map.keys, map.values
```

# Array

```
val zeros = Array[String](0)
```

```
val oneToFive = Array(1,2,3,4,5)
```

```
import scala.collection.mutable._
```

```
val buffer = Array(1,2,3,4)  
    buffer += 5  
    buffer -= 2
```

# Gdzie szukać info?

[www.scala-lang.org](http://www.scala-lang.org) / Documentation / Manuals

[ibm.com/developerworks/java/library/j-scala01228.html](http://ibm.com/developerworks/java/library/j-scala01228.html)

[scala.sygneca.com](http://scala.sygneca.com)

Książki wydawnictw: O'reilly, Artima, Apress oraz Pragmatic

Kanał irc [#scala.pl](#) oraz [#scala](#) @ freenode  
(aka blackrain)

# Pytania ?



**Dziękuję za uwagę!**