Uniwersytet Wrocławski

Scala in Practice

Uniwersytet Wrocławski

Local functions

```
def processFile ( filename : String , width : Int ) = {

  def procLine ( l : String ) ={
    if ( l.length <= width) println ( filename + " : " + l.trim )
  }

  val source = Source.fromFile ( filename )
  for ( line <- source.getLines ) procLine ( line )
}
```

Uniwersytet Wrocławski

Repeated Parameters

```
scala > def echo ( args : String *) =
            for ( arg <- args ) println ( arg )

echo : ( args : String *) Unit

scala > echo ( " Scala ", " In ", " Practice ")
Scala
In
Practice
```

Uniwersytet Wrocławski

Repeated Parameters

```
scala > def echo ( args : String *) =
            print ( args.slice (1 , args.size ))

scala > echo ( " Scala ", " In ", " Practice ")
WrappedArray ( In , Practice )
```

Uniwersytet Wrocławski

Named Parameters

```
scala > def speed ( distance : Double , time : Double ) =
            distance / time
speed : ( distance : Double , time : Double ) Double

scala > speed (100 , 20)
res0 : Double = 5.0

scala > speed ( time = 20 , distance = 100)
res1 : Double = 5.0
```

Uniwersytet Wrocławski

Default parameters

```
def log(message: String, level: String = "INFO") =
  println(s"$level: $message")

scala > log("Object created")
INFO: Object created

scala > printTime ( "Object created", "DEBUG" )
DEBUG: Object created
```

Uniwersytet Wrocławski

First Class Functions

- Functions are values
  - Can be passed as arguments to higher order functions
  - Can be returned by other functions
  - Can be assigned to a variable

Uniwersytet Wrocławski

Back to high school

$$f(x) = x * x - 3$$
$$g(x) = 2 * x + 6$$

$$g \circ f = g(f(x)) = 2 * (x * x - 3) + 6 = 2 * x * x$$

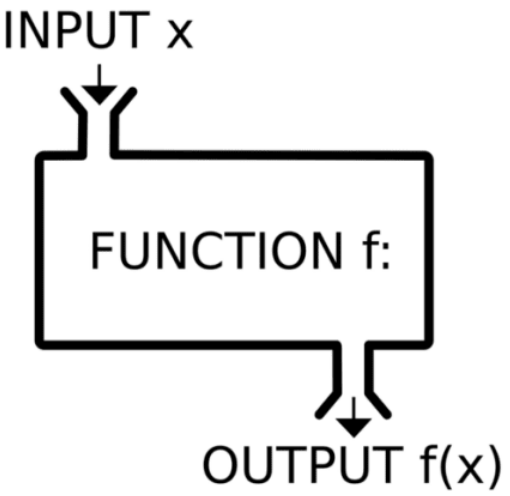
Uniwersytet Wrocławski

Back to high school

$$f(x) = f_1 f_2 f_3 \ldots f_N(x)$$

Uniwersytet Wrocławski

Functional programming



Uniwersytet Wrocławski

Higher Order Functions

```
def myMap ( lst : List [ Int ] , fun : Int => Int ) : List [ Int ] =
        for ( l <- lst ) yield fun ( l )

def addOne ( n : Int ) = n + 1

scala > myMap ( List(11, 3, 4), addOne )
res0 : List [ Int ] = List (12, 4, 5)
```

Uniwersytet Wrocławski

Mapping collections

```
def addOne ( n : Int ) = n + 1

scala > myMap ( List(11, 3, 4), addOne )

scala > List(11, 3, 4).map(addOne)
res0 : List [ Int ] = List (12, 4, 5)
```

Function Literals / Lambdas

```
scala > ( x : Int ) => x + 1
res0  : Int => Int = $$Lambda$1031/2118984327@3f4b840d
```

```
scala > val numbers = List ( 10 , -10 , 5)
scala > numbers.map ((x: Int) => x + 1)
res1 : List [ Int ] = List ( 11 , -9 , 6)
```

Uniwersytet Wrocławski

Lambdas as values

```
scala > val addFun = ( x : Int ) => x + 1
res0 : Int = > Int = $$Lambda$1053/498023236@7fd8c559

scala > val numbers = List ( 10 , -10 , 5)
scala > numbers.map (addFun)
res1 : List [ Int ] = List ( 11 , -9 , 6)
```

Uniwersytet Wrocławski

Lambda tricks

```
scala > numbers.map (( x : Int ) => x * 2)
scala > numbers.map ( x  => x * 2)

scala > var addfun = x => x +1
< console >:7: error : missing parameter type var addfun = x => x +1
```

Uniwersytet Wrocławski

Lambda tricks

```
scala > numbers.map ( x  => x * 2)

scala > numbers.map ( _ + 2)

scala > val add = ( _ : Int ) + ( _ : Int )
add: (Int, Int) => Int = $$Lambda$1033/73351708@1816e24a
```

Uniwersytet Wrocławski

Lambda tricks

```
scala > numbers.map ( x => println(x) )

scala > numbers.map ( println(_) )

scala > numbers.map ( println _ )

scala > numbers.map ( println )
```

Uniwersytet Wrocławski

Closures

```
scala > var more = 1
more : Int = 1

scala > def addMore ( x : Int ) = x + more
addMode : ( x : Int ) Int

scala > addMore (1)
res3 : Int = 2
```

Uniwersytet Wrocławski

Changing captured vars

```
scala > var sum = 0
sum : Int = 0

scala > def addToSum ( x : Int ) { sum += x }
addToSum : ( x : Int ) Unit

scala > addToSum (42)
scala > addToSum (23)
scala > sum
res7 : Int = 65
```

Creating and Returning Closures

```
def makeIncreaser ( more : Int ) = ( x : Int ) => x + more

scala > val inc9999 = makeIncreaser (9999)
res0: Int => Int = $$Lambda$1044/1362435880@21618fa7

scala > inc9999 (1)
res0 : Int = 10000
```

Uniwersytet Wrocławski

Assignment constraints

```
scala > def squareDef ( x : Int ) = x * x
squareDef ( x : Int ) Int

scala > val squareDef2 = squareDef
<console>:12: error: missing argument list for method squareDef
Unapplied methods are only converted to functions when a function type is expected.
You can make this conversion explicit by writing `squareDef _` or `squareDef(_)` instead of
`squareDef`.
    val squareDef2 = squareDef

scala > val squareDef2 = squareDef _
squareDefToo : Int => Int = $$Lambda$1146/769342184@79518e00
```

Uniwersytet Wrocławski

Partially applied functions

```
scala > val squareDef2 = squareDef _
scala > val squareDef2 = ( x : Int ) => squareDef ( x )

scala > def sumThree ( a : Int , b : Int , c : Int ) = a + b + c
sumThree : ( a : Int , b : Int , c : Int ) Int

scala > val sumThree2 = sumThree _
sumThree2 : (Int, Int, Int) => Int = $$Lambda$1207/76306072@17884d
```

Uniwersytet Wrocławski

Currying

```
scala > def curriedSum ( x : Int, y : Int ) = x + y
curriedSum : ( x : Int, y : Int ) Int

scala > def curriedSum ( x : Int )( y : Int ) = x + y
curriedSum : ( x : Int )( y : Int ) Int

scala > curriedSum (1)(2)
res0 : Int = 3
```

Uniwersytet Wrocławski

Currying

scala > def curriedSum (x : Int)(y : Int) = x + y

curriedSum : (x : Int)(y : Int) Int

scala > curriedSum (1)(2)

res0 : Int = 3

scala > curriedSum (1)

< console >:14: error : missing arguments for method curriedSum ; follow this method with ‘_’ if you want to treat it as a partially applied function

Uniwersytet Wrocławski

More specific functions

```
scala > def curriedSum ( x : Int )( y : Int ) = x + y
curriedSum : ( x : Int )( y : Int ) Int

scala > val sum5 = curriedSum (5)(_)
scala > val sum44 = curriedSum (44) _
...
```

Uniwersytet Wrocławski

Tricks

```
scala > def sum ( x : Int, y : Int ) = x + y
sum: (x: Int, y: Int)Int

scala > val s = sum _
s: (Int, Int) => Int = $$Lambda$1302/494021631@72c29d87

scala > val s = (sum _).curried
s: Int => (Int => Int) = scala.Function2
$Lambda$1279/1951379728@50fe5df2
```

Uniwersytet Wrocławski

Example

```
scala > def twice ( op : Double => Double )( x : Double ) =
            op ( op ( x ))
twice : ( op : Double = > Double )( x : Double ) Double

scala > twice ( x => x + 2 )(3)
res16 : Double = 7.0

scala > twice ( _ + 2 )(3)
res16 : Double = 7.0
```

Uniwersytet Wrocławski

Generic functions

```
def randomName(names: Seq[String]): String = {
    val randomNum = util.Random.nextInt(names.length)
    names(randomNum)
}

def randomElement[A](seq: Seq[A]): A = {
    val randomNum = util.Random.nextInt(seq.length)
    seq(randomNum)
}
```

Uniwersytet Wrocławski

Parameters

- By default Scala is call-by-value
- Any expression is evaluated before it is passed as a function parameter

- Can force call-by-name by prefixing parameter types with =>
- Expression passed to parameter is evaluated every time it is used

Uniwersytet Wrocławski

By-Value Parameters

```
def add (x : Int, y : Int ) ={
  println ( s" add: $x + $y" )
  x + y
}

scala > add ( add (1 , 2) , add (2, 3))
add: 1 + 2
add: 2 + 3
add: 3 + 5
res20: Int = 8
```

Uniwersytet Wrocławski

By-Name Parameters

```
def lazyAdd (x : => Int, y : => Int ) ={
  println ( s" lazy add: $x + $y" )
  x + y
}

scala > lazyAdd ( lazyAdd (1 , 2) , lazyAdd (2, 3))
lazy add: 1 + 2
lazy add: 2 + 3
lazy add: 3 + 5
lazy add: 1 + 2
lazy add: 2 + 3
res22: Int = 8
```

Uniwersytet Wrocławski

By-Name Parameters

```
def notSoLazyAdd ( x : => Int , y : => Int ) ={
  val a = x
  val b = y
  println( " not so lazy add : " + a + " + " + b )
  a + b
}

scala> lazyAdd3 ( lazyAdd3 (1 ,2) , lazyAdd3 (2 ,3))
not so lazy add : 1 + 2
not so lazy add : 2 + 3
not so lazy add : 3 + 5
res24: Int = 8
```

Uniwersytet Wrocławski

Tricks

```
def trueOrException(exMessage: String)(block: => Boolean): Unit =
  if (!block) throw new Exception(exMessage)

def assert(x: Int, y: Int): Unit =  trueOrException(s"Incorrect parameters: $x & $y") {
  x > y & x < 20
}

def assert(msg: String): Unit = trueOrException("Incorrect message: $msg") {
  msg.length > 50
}
```

Uniwersytet Wrocławski

Lazy vals

```
def makeString() = { println ( " in makeString " ); " hello " + " world " }

def printString() = {
  lazy val s = makeString()
  print ( " in printString " )
  println ( s )
}

scala > printString()
in printString
in makeString
hello world
```

Uniwersytet Wrocławski

Constraints validation

```
def calculate(x: Int, y: Int) = {
  x / y + 200
}

def anyMethod(...) = {
  require (someBooleanChecks, “Some message if not fulfilled”)
   . . .
}
```