PEP Scala (2)

Email:	christian.urban at kcl.ac.uk
Office:	N7.07 (North Wing, Bush House)

Slides & Code:	KEATS
Office Hours:	Thursdays 12:00 – 14:00
Additionally:	(for Scala) Tuesdays 10:45 – 11:45

Scala 2.13.1

```
$ scala

Welcome to Scala 2.13.1 (Java HotSpot(TM)
64-Bit Server VM, Java 9). Type in expressions
for evaluation. Or try :help.

scala>
```

With older versions you will get strange results with my reference implementation.

Reference Implementation

Keep your implementation and my reference implementation separate.

```
$ scala -cp collatz.jar

scala> CW6a.collatz(6)
res0: Long = 8


scala> import CW6a._
scala> collatz(9)
res1: Long = 19
```

Preliminary Part 7

$$\texttt{overlap}(d_1, d_2) = \frac{d_1 \cdot d_2}{max(d_1^2, d_2^2)}$$

where d_1^2 means $d_1 \cdot d_1$ and so on

Discussion Forum

"Since we can't use `var`s I was wondering if we could use a stack?"

My `collatz` and `collatz_max` functions are 4 loc each.

Email: Hate 'val'

Subject: **Hate '`val`'** 01:00 AM

Hello Mr Urban,

I just wanted to ask, how are we suppose to work with the completely useless **`val`**, that can't be changed ever? Why is this rule active at all? I've spent 4 hours not thinking on the coursework, but how to bypass this annoying rule. What's the whole point of all these coursework, when we can't use everything Scala gives us?!?

Regards.

«deleted»

Subject: **Re: Hate '`val`'** 01:02 AM

« my usual rant about fp...
concurrency bla bla... better programs yada »

PS: What are you trying to do where you desperately want to use `var`?

Subject: **Re: Re: Hate 'val'** 01:04 AM

Right now my is_legal function works fine:

```
def is_legal(dim: Int, path: Path)(x: Pos): Boolean = {
  var boolReturn = false
  if(x._1 > dim || x._2 > dim || x._1 < 0 || x._2 < 0) {
  else { var breakLoop = false
         if(path == Nil) { boolReturn = true }
         else { for(i <- 0 until path.length) {
                   if(breakLoop == false) {
                     if(path(i) == x) {
                       boolReturn = true
                       breakLoop = true
                     }
                     else { boolReturn = false }
                   } else breakLoop
              }
         }
         boolReturn
  }
```

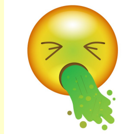
...but I can't make it work with boolReturn being val. What approach would you recommend in this case, and is using var in this case justified?

Subject: **Re: Re: Hate 'val'** 01:04 AM

Right now my is_legal function works fine:

```
def is_legal(dim: Int, path: Path)(x: Pos): Boolean = {
  var boolReturn = false
  if(x._1 > dim || x._2 > dim || x._1 < 0 || x._2 < 0) {
  else { var breakLoop = false
         if(path == Nil) { boolReturn = true }
         else { for(i <- 0 until path.length) {
                   if(breakLoop == false) {
                     if(path(i) == x) {
                       boolReturn = true
                       breakLoop = true
                     }
                     else { boolReturn = false }
                   } else breakLoop

turn
```

Me:



...but I can't make it work with boolReturn being val. What approach would you recommend in this case, and is using var in this case justified?

Subject: **Re: Re: Re: Hate ’val’** 01:06 AM

OK. So you want to make sure that the x-position is not outside the board....and furthermore you want to make sure that the x-position is not yet in the path list. How about something like

```
def is_legal(dim: Int, path: Path)(x: Pos): Boolean =
  ...<<some board conditions>>... && !path.contains(x)
```

Does not even contain a `val`.

(This is all on one line)

Subject: **Re: Re: Re: Re: Hate ’`val`’** 11:02 AM

THANK YOU! You made me change my coding perspective. Because of you, I figured out the next one…

Subject: **Re: Re: Re: Re: Hate 'val'** 11:02 AM

THANK YOU! You made me change my coding perspective. Because of you, I figured out the next one...

Me:

Assignments

Don't change any names or types in the templates!

Avoid at all costs:

- `var`
- `return`
- `ListBuffer`
- `mutable`
- `.par`

I cannot think of a good reason to use stacks.

For-Comprehensions Again

```
for (n <- List(1, 2, 3, 4, 5)) yield n*n
```

For-Comprehensions Again

```
for (n <- List(1, 2, 3, 4, 5)) yield n*n

n*n:    List(1,  4,  9,  16,  25)
```

For-Comprehensions Again

```
for (n <- List(1, 2, 3, 4, 5)) yield n*n

n*n:  List(1,  4,  9,  16,  25)
```

This is for when the for-comprehension **yields / produces** a result.

For-Comprehensions Again

```
for (n <- List(1, 2, 3, 4, 5)) yield n*n
```

vs

```
for (n <- List(1, 2, 3, 4, 5)) println(n)
```

The second version is in case the for **does not** produce any result.

Find something below 4 in a list. What do you think Scala answers?

```
List(7,2,3,4,5,6).find(_ < 4)
```

```
List(5,6,7,8,9).find(_ < 4)
```

Find something below 4 in a list. What do you think Scala answers?

```
List(7,2,3,4,5,6).find(_ < 4)
res: Option[Int] = Some(2)

List(5,6,7,8,9).find(_ < 4)
res: Option[Int] = None
```

Option Type

- if the value is present, you use

```
Some(value)
```

- if no value is present, you use

```
None
```

e.g. `Option[Int]`, then `Some(42)` and `None`
good for error handling

Option Type

```
Integer.parseInt(”1234”)

// vs.

def get_me_an_int(s: String) : Option[Int] =
 Try(Some(Integer.parseInt(s))).getOrElse(None)
```

in the Scala code it is clear from the type I that have to deal with the None-case; no JavaDoc needed

Higher-Order Functions

In Scala, functions can take other functions as arguments and can return a function as a result.

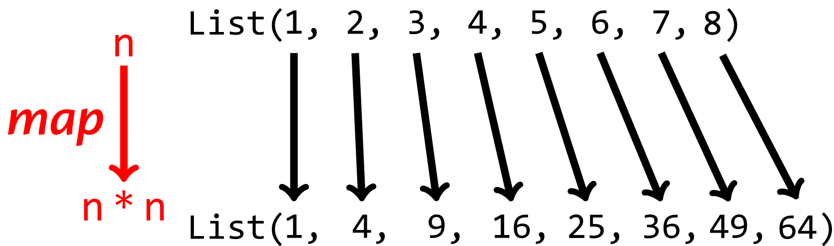
```
List(7,2,3,4,5,6).find(_ < 4)
```

Higher-Order Functions (2)

```
def even(x: Int) : Boolean = x % 2 == 0

List(1, 2, 3, 4, 5).filter(even)
  res : List[Int] = List(2, 4)

List(1, 2, 3, 4, 5).count(even)
  res : Int = 2

List(1, 2, 3, 4, 5).find(even)
  res: Option[Int] = Some(2)
```

map (lower case)

applies a function to each element of a list (and more)




```
List(1,2,3,4,5,6,7,8).map(n => n * n)
```

For-Comprehensions are maps

```
for (n <- List(1,2,3,4,5,6,7,8))
  yield n * n


// is just syntactic sugar for


List(1,2,3,4,5,6,7,8).map(n => n * n)
```

Map (upper case)

a type, representing a key-value association datastructure

```
val ascii =
    ('a' to 'z').map(c => (c, c.toInt))

val ascii_Map = ascii.toMap

ascii_Map.get('a')   // -> 97
```

Pattern Matching

…on pairs:

```
def fizz_buzz(n: Int) : String =
 (n % 3, n % 5) match {
    case (0, 0) => "fizz buzz"
    case (0, _) => "fizz"
    case (_, 0) => "buzz"
    case _ => n.toString
 }
```

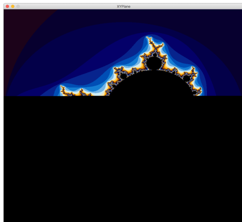
Recursion

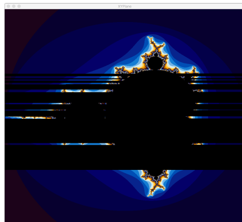
```
def fib(n: Int) : Int = {
  if (n == 0 || n == 1) 1
   else fib(n - 1) + fib(n - 2)
}
```

Recursion

```scala
def my_flatten(xs: List[Option[Int]]): List[Int] =
 xs match {
   case Nil => Nil
   case None :: rest => my_flatten(rest)
   case Some(v) :: rest => v :: my_flatten(rest)
 }
```

Questions?



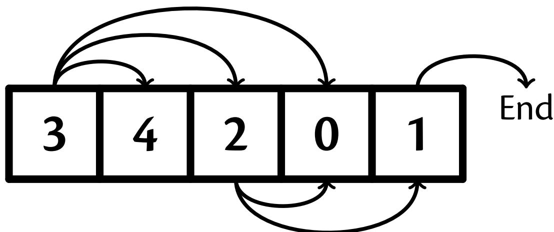


My Office Hours: Thursdays 12 – 14
And specifically for Scala: Tuesdays 10:45 – 11:45

Mind-Blowing Programming Languages: Overloading in any language is great but it makes a difference `10/3` or `10.0/3`

Mind-Blowing Programming Languages: PHP (7.0)

Jumping Towers



shortest: 3 $\rightarrow$ 4 $\rightarrow$ End

“Children” / moves