



# PEP Scala (1)

- 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

# Why Scala?

twitter 

Linked 

theguardian

Morgan Stanley

CREDIT SUISSE 

...



edf  
ENERGY

Novell.

foursquare™

HSBC 

...

developed since 2004 by Martin Odersky (he was behind Generic Java which was included in Java 5 ...I am using Scala since maybe 2008?)

# Why Scala?

- compiles to the JVM  
(also JavaScript, native X86 in the works)
- integrates seamlessly with Java
- combines functional and **object-oriented** programming
- it is a bit on the “theory” / “mathematical” side  
(no pointers, no null, but expressions)
- often one can write very concise and elegant code

# Java vs Scala

```
public class Point {
    private final int x, y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int x() { return x; }

    public int y() { return y; }
}
```

Java

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12

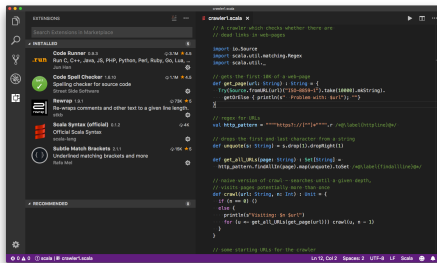
---

```
case class Point(val x: Int, val y: Int)
```

Scala

# First Steps: Scala Tools

- contains a REPL
- I use VS Code and a Scala extension (M'place)



The screenshot shows the Visual Studio Code interface. On the left, the 'EXTENSIONS' sidebar is open, displaying a search for 'Scala' in the marketplace. The 'INSTALLED' section lists several extensions, including 'Code Runner', 'Code Spell Checker', 'Rewrap', 'Scala System (official)', and 'Subtle Match Brackets'. The 'RECOMMENDED' section is also visible. The main editor area shows a Scala file named 'crawler.scala' with the following code:

```
// A crawler which checks whether there are
// dead links in webpages

import scala.util.matching.Regex
import scala.util._

// gets the first 100 of a webpage
def get_page(url: String): String = {
  Try(Sources.fromURL(url).take(10000).asString)
  getOrElse { println(" Problem with: " + url); "" }
}

// regex for URLs
val http_pattern = """http://[^\s]*\.[^\s]{3,4}"""

// drops the first and last character from a string
def unquote(s: String) = s.drop(1).dropRight(1)

def get_all_urls(pages: String): Set[String] =
  http_pattern.findAllIn(pages).map(unquote).toSet

// main version of crawl - searches until a given depth,
// prints page addresses until then stop
def crawl(url: String, so far: Int) = {
  if (so far == 0) ()
  else {
    println("visiting: " + url)
    for (u <- get_all_urls(get_page(url)) crawl(u, so far - 1))
  }
}

// some starting URLs for the crawler
```

- there is a plugin for Eclipse (called Scala IDE)
- there is also a plugin for IntelliJ

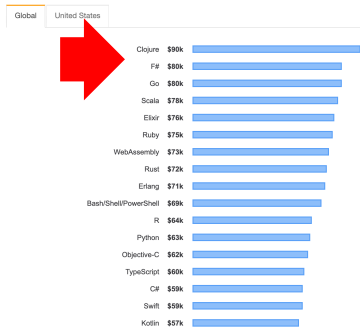
# Why Scala?

Elm, Rust, Haskell, Ocaml, F#, Erlang, ML, Lisp (Racket)...

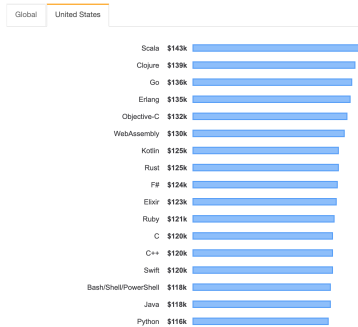
# Why Scala?

## Money?

What Languages Are Associated with the Highest Salaries Worldwide?



What Languages Are Associated with the Highest Salaries Worldwide?



\* source: Stackoverflow Developer Survey, 2019

Elm, Rust, Haskell, Ocaml, F#, Erlang, ML, Lisp (Racket)...

# Why Scala?

## Money?

What Languages Are Associated with the Highest Salaries Worldwide?

Global United States

What Languages Are Associated with the Highest Salaries Worldwide?

Global United States



\* source: Stackoverflow Developer Survey, 2019

Elm, Rust, Haskell, Ocaml, F#, Erlang, ML, Lisp (Racket)...



# Why Functional Programming?

Elm, Haskell, Ocaml, F#, Erlang, ML, Lisp (Racket)...

# Why Functional Programming?

“If you want to see which features will be in mainstream programming languages tomorrow, then take a look at functional programming languages today.”

—Simon Peyton Jones (works at Microsoft)  
main developer of the Glasgow Haskell Compiler

Elm, Haskell, Ocaml, F#, Erlang, ML, Lisp (Racket)...

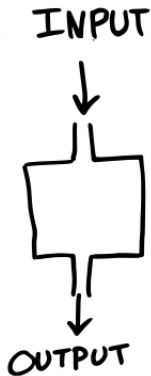
# Why Functional Programming?



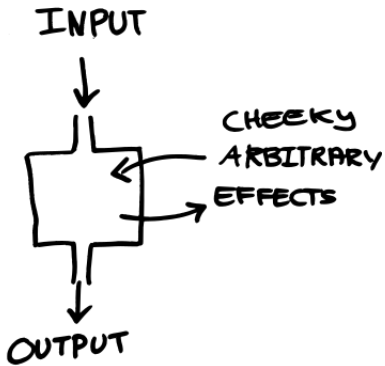
## Immutability

Elm, Haskell, Ocaml, F#, Erlang, ML, Lisp (Racket)...

## Functions



## Procedures



\* from "What pure functional programming is all about?"

Why bother? or

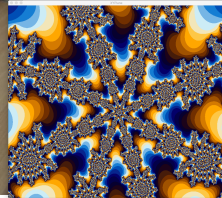
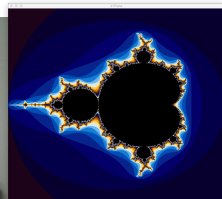
What is wrong with this?

```
for (int i = 10; i < 20; i++) {  
  
    //...Do something interesting  
    //    with i...  
  
}
```

1986



3 days



64K RAM, no HD, no monitor, lots of cables

1986



1988, C



1986



1988, C



1992, Linux





1986



1988, C



1992, Linux



1996

1986



1988, C



1992, Linux



2000



1996



1986



1988, C



1992, Linux



2012?

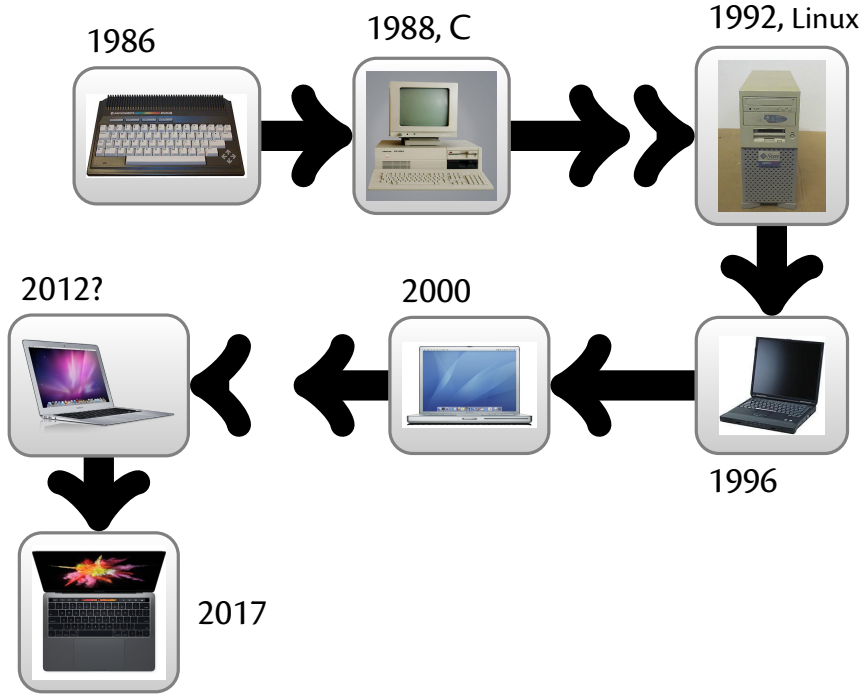


2000



1996





1986



1988, C



1992, Linux



2012?



2000



1996

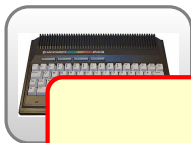


2017

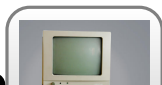


**1986: no Internet  
no Amazon  
no FB, no mobiles,...**

1986



1988, C



1992, Linux



### Speedup by Moore's Law

<b>1986:</b>	3 days	<b>1996:</b>	135 mins
<b>1988:</b>	1.5 days	<b>1998:</b>	67 mins
<b>1990:</b>	18 hs	<b>2000:</b>	33 mins
<b>1992:</b>	9 hs	<b>2002:</b>	16 mins
<b>1994:</b>	4.5 hs		???

2012?



Every two years, computers got twice as powerful.

2017



**no Amazon**  
**no FB, no mobiles,...**

1986



1988, C



1992, Linux



2012?



2000



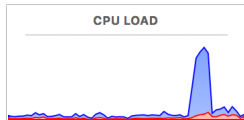
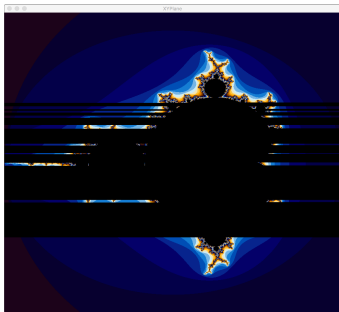
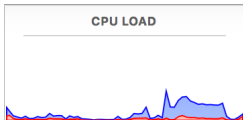
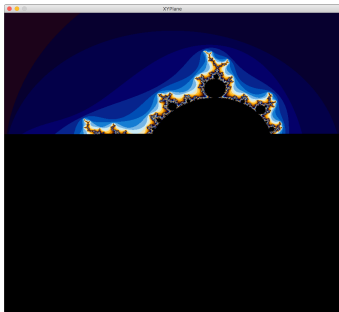
1996



2017

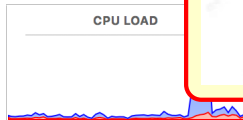
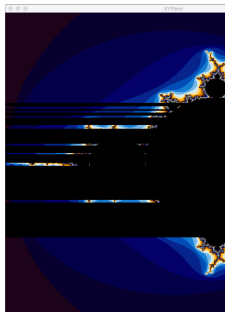
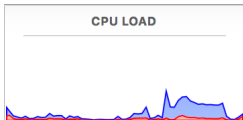
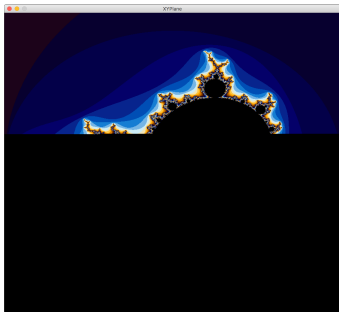
**1986: no Internet  
no Amazon  
no FB, no mobiles,...**

# Seq vs Par





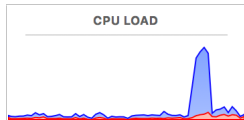
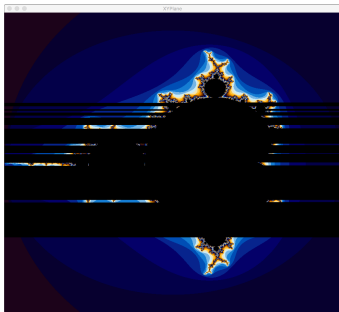
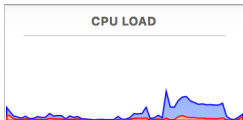
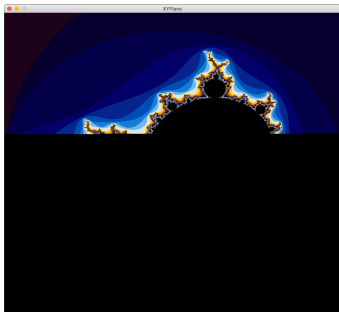
# Seq vs Par



in Java or C++



# Seq vs Par



In FP: Once a variable is created, it is assigned a value and then never changed again  $\Rightarrow$  no synchronisation needed

# Types

- Base types

Int, Long, BigInt, Float, Double  
String, Char  
Boolean

- Compound types

List[Int]

lists of Int's

Set[Double]

sets of Double's

(Int, String)

Int-String pair

List[(BigInt, String)]

lists of BigInt-String  
pairs

List[List[Int]]

list of lists of Int's

Option[Int]

options of Int's

# Coursework Dates

Similar to C++:

- Preliminary Parts: Wednesdays 4pm
  - Preliminary Part 6: 3% (13 November)
  - Preliminary Part 7: 4% (20 November)
  - Preliminary Part 8: 4% (27 November)
  - Preliminary Part 9: 4% (5 December)
- Core Part: 35% (15 January 2020)

# Coursework

- Sorry, I might have been a bit wordy:  
Part 6 of CW description is 7 pages, but I only needed < 100 loc for *all* Part 6.
- there is feedback when pushing code to github
- there are jar-files you can use to test my reference implementation
- we want you to learn FP!  
**no vars**, no mutable data-structures  
e.g. no Arrays, no ListBuffer



# The Joy of Immutability

- If you need to manipulate some data in a list say, then you make a new list with the updated values, rather than revise the original list. Easy!

```
val old_list = List(1, 2, 3, 5)
val new_list = 0 :: old_list
                // -> List(0, 1, 2, 3, 4, 5)
```

- You do not have to be defensive about who can access the data.
- You can look at your code in isolation.

# 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?



Right now my is\_legal function works fine:

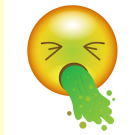
```
def is_legal(dim: Int, path: Path)(x: Pos): Boolean = {
  var boolReturn = false
  if(x._1 > dim || x._2 > dim || x._1 < 0 || x._2 < 0) {
  else { var breakLoop = false
        if(path == Nil) { boolReturn = true }
        else { for(i <- 0 until path.length) {
              if(breakLoop == false)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

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
```

Me:



turn

...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 'va1'**

11:02 AM

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

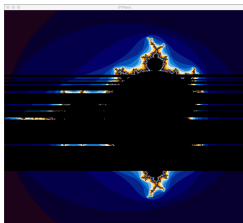
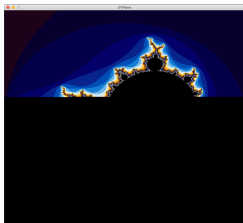
**Me:**



# Conclusion for Today

- Scala is still under development, 2.13.1 came out in Sept. (the compiler is terribly slow)
- <http://www.scala-lang.org/>
- it is a rather **deep** language...i.e. gives you a lot of rope to shoot yourself
- learning functional programming is not easy...when you have spent all of your career thinking in an imperative way, it is hard to change
- hope you have fun with Scala and the assignments

# Questions?



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