



## PEP Scala (2)

Email: christian.urban at kcl.ac.uk

Slides & Code: KEATS

Office Hour: Fridays 11:30 – 12:30

Location: N7.07 (North Wing, Bush House)

# Scala 3

**scala**  $\Rightarrow$  <https://www.scala-lang.org/download/>

Installation problems:

- Zoltan Meszaros (zoltan.meszaros@kcl.ac.uk, Linux)
- Zishan Rahman (zishan.rahman@kcl.ac.uk, Linux)
- Bofan Zhang (bofan.zhang@kcl.ac.uk, MacOSX)
- Aidan Dakhama (aidan.dakhama@kcl.ac.uk, Linux)

Github problems:

- Zishan Rahman (zishan.rahman@kcl.ac.uk)

```
def fname(arg1: ty1, arg2: ty2,..., argn: tyn): rty = {  
    ....  
}
```

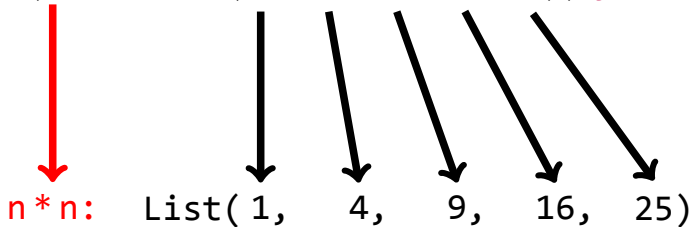
```
def average(xs: List[Int]) : Int = {  
    val s = xs.sum  
    val n = xs.length  
    s / n  
}
```

# For-Comprehensions

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

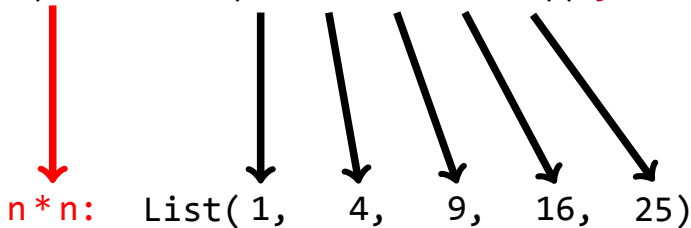
# For-Comprehensions

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



# For-Comprehensions

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



This is for when the for-comprehension  
**yields / produces** a 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)
```

# Anonymous Functions

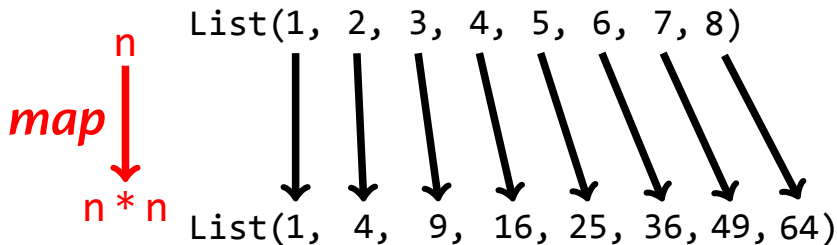
```
def less4(x: Int) = x < 4
```

vs

```
(x: Int) => x < 4
```

# 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  
  }
```

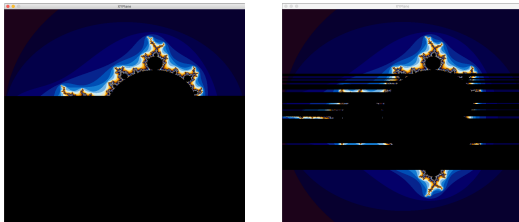
# Recursion

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

# Recursion

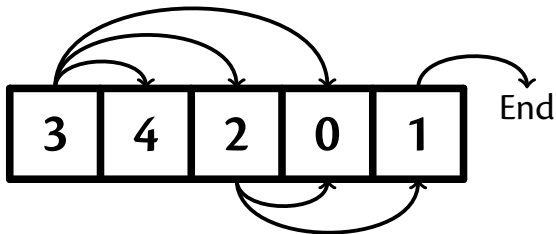
```
def my_flatten(xs: List[Option[Int]]): List[Int] =  
  xs match {  
    case Nil => Nil  
    case None :: rest => my_flatten(rest)  
    case Some(v) :: rest => v :: my_flatten(rest)  
  }
```

# Questions?



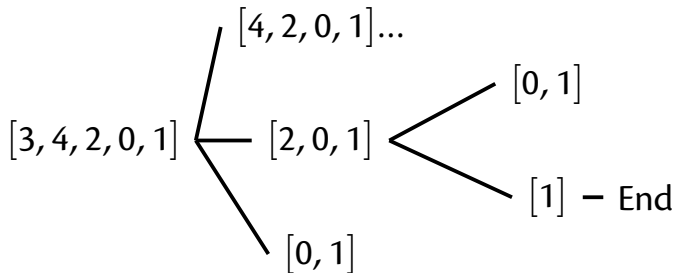
My Office Hours: Thursdays 12 – 14  
And specifically for Scala: Tuesdays 10:45 – 11:45

# Jumping Towers



shortest:  $3 \rightarrow 4 \rightarrow \text{End}$

# “Children” / moves



























???

