PEP Scala (4)

Email: christian.urban at kcl.ac.uk
Slides & Code: KEATS

https://pollev.com/cfltutoratki576

Hints in CW

Hints

For Preliminary Part: useful operations involving regular expressions:

```
reg.findAllIn(s).toList
```

finds all substrings in `s` according to a regular regular expression `reg`; useful list operations: `.distinct` removing duplicates from a list, `.count` counts the number of elements in a list that satisfy some condition, `.toMap` transfers a list of pairs into a Map, `.sum` adds up a list of integers, `.max` calculates the maximum of a list.

For Core Part: use `.split(",").toList` for splitting strings according to commas (similarly `\n`), `.getOrElse(..,..)` allows to query a Map, but also gives a default value if the Map is not defined, a Map can be 'updated' by using `+`, `.contains` and `.filter` can test whether an element is included in a list, and respectively filter out elements in a list, `.sortBy(_._2)` sorts a list of pairs according to the second elements in the pairs—the sorting is done from smallest to highest, `.take(n)` for taking some elements in a list (takes fewer if the list contains less than `n` elements).

Scala Library, e.g. span in

https://www.scala-lang.org/api/current/scala/collection/

Discussion Forum

Re: Core 6 - Getting a little off the target numbers for Part7

by Christian Urban - Saturday, 23 November 2019, 1:06 AM

Hi,

It is a subtle problem, but unfortunately Scala calculates different results according to when you round numbers. As a result yearly_yield needs to be careful when numbers are rounded to Longs. For example, if your balance is $100 and your calculated profit is negative, say -20.5, then

100 + ((-20.5).toLong) = 80

while

(100 + (-20.5)).toLong = 79

Hope this helps,
Christian

Preliminary 7

Raw marks (261 submissions):

4%:	236
3%:	10
2%:	1
1%:	0
0%:	15

(plagiarism/collusion interviews ongoing!)

```
def is_legal(dim: Int, p: Path, x: Pos) = {
  if (...some_really_long_condition...) false
  else true
}
```

```
def is_legal(dim: Int, p: Path, x: Pos) = {
  if (...some_really_long_condition...) false
  else true
}
```

```
def is_legal(dim: Int, p: Path, x: Pos) =
  !(...some_really_long_condition...)
```

```
def foobar(...) = {
  val cs = for (c <- str) yield c.toLowerCase
  ...
}
```

```
def foobar(...) = {
  val cs = for (c <- str) yield c.toLowerCase
  ...
}
```

```
def foobar(...) = {
  val cs = str.map(_.toLowerCase)
  ...
}
```

```
def RomanNumeral2Int(rs: RomanNumeral): Int =
 rs match {
    case Nil => 0
    case M::r     => 1000 + RomanNumeral2Int(r)
    case C::M::r => 900 + RomanNumeral2Int(r)
    case D::r     => 500 + RomanNumeral2Int(r)
    case C::D::r => 400 + RomanNumeral2Int(r)
    case C::r     => 100 + RomanNumeral2Int(r)
    case X::C::r => 90 + RomanNumeral2Int(r)
    case L::r     => 50 + RomanNumeral2Int(r)
    case X::L::r => 40 + RomanNumeral2Int(r)
    case X::r     => 10 + RomanNumeral2Int(r)
    case I::X::r => 9 + RomanNumeral2Int(r)
    case V::r     => 5 + RomanNumeral2Int(r)
    case I::V::r => 4 + RomanNumeral2Int(r)
    case I::r     => 1 + RomanNumeral2Int(r)
 }
```

Last Week: Pattern Matching

```
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
  }
```

Reverse Polish Notation

$$(3 + 1) * (2 + 9)$$

$$\Rightarrow$$

$$3 \;\; 1 \;\; + \;\; 2 \;\; 9 \;\; + \;\; *$$

Reverse Polish Notation

$$(3 + 1) * (2 + 9)$$

$$\Rightarrow$$

$$3\ \ 1\ +\ \ 2\ \ 9\ +\ \ *$$

```
ldc 3
ldc 1
iadd
ldc 2
ldc 9
iadd
imul
```

Sudoku

A very simple-minded version on 110 problems:

1 core: 800 secs

2 cores: 400 secs

8 cores: 290 secs

18 cores: 142 secs

Mind-Blowing Regular Expressions: in Python, JavaScript, Java

Regular Expressions

Suppose you have the regular expression `(a*)b` :

```
"aaaaaaaaaaaaaaab"
```

Regular Expressions

Suppose you have the regular expression `(a*)b`:

`"aaaaa.....aaaaaaaaaaaaaaaaaaaab"`

Regular Expressions

Suppose you have the regular expression `(a*)b`:

`"aaaaa.....aaaaaaaaaaaaaaaaaaaa"`

Regular Expressions

Suppose you have the regular expression `(a*)*b`:

`"aaaaa.....aaaaaaaaaaaaaaaaaaaab"`

Regular Expressions

Suppose you have the regular expression `(a*)*b`:

```
"aaaaa.............aaaaaaaaaaaaaaaaaaaaa"
```

Regular Expressions

Suppose you have the regular expression `(a*)*b`:

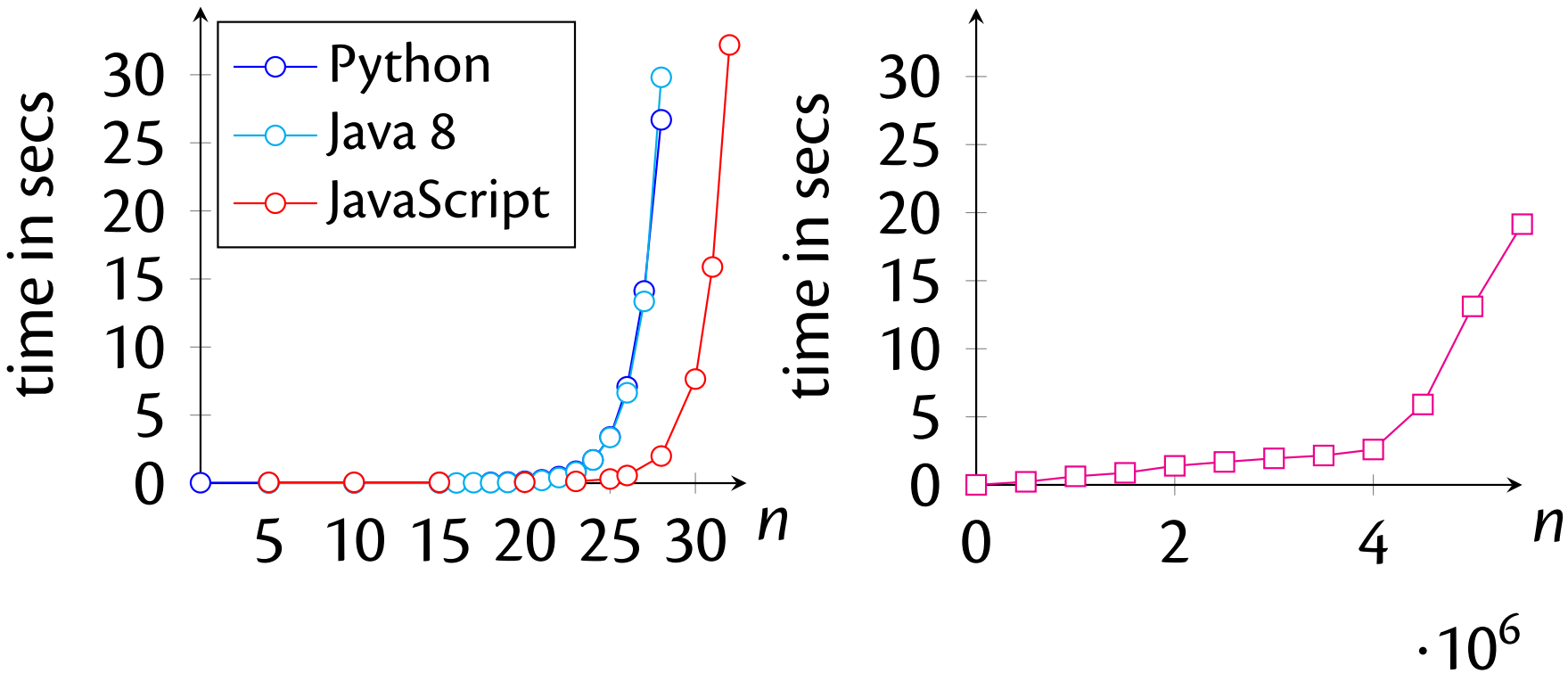
```
"aaaaa.............aaaaaaaaaaaaaaaaaaaaa"
```

How long does Python need to find out?

CW 9: Regexes

Graphs: `(a*)*b` and strings

$$\underbrace{a \ldots a}_{n}$$



https://vimeo.com/112065252