Introduction to Regex
Learn to recognize when you should use it.

Jørgen Høgberget

- `anyNumber = r"^?\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*\b?"`

- `anyNumber = r"^?\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*\b?"`
- Above: Regex pattern for any number on the form 1, 123.456, $123.456e-123$, $123.456e+123$, etc.

```
>>> import re
>>> text = "asd1 foo 2.0 0.123213 foo foo 1E-2 foo 1e-04 1.2313e123"
>>> anyNumber = "\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"
>>> re.findall(anyNumber, text)
[’1’, ’2.0’, ’0.123213’, ’1E-2’, ’1e-04’, ’1.2313e123’]
```

Cracking the code

- Regular expressions takes as input a *pattern* and the *source string*.

Cracking the code

- Regular expressions takes as input a *pattern* and the *source string*.
- `anyNumber` is a pattern reprisenting any number.
- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`

Cracking the code

- Regular expressions takes as input a *pattern* and the *source string*.
- `anyNumber` is a pattern reprisenting any number.
- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- `r"string"` is a *raw string*. These strings does not read special string identifiers such as `\n` (which might mess up the regexp).

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- `? + * . ) (` has special functions. Let's focus on them later.

Cracking the code

- `anyNumber = r"\s\d\.\d[eE][\-\+]\d)\s"`

Cracking the code

- `anyNumber = r"\s\d\.\d[eE][\-\+]\d)\s"`
- `\s` - matches any whitespace character (tabs, newlines, etc.)

Cracking the code

- `anyNumber = r"\s\d\.\d[eE][\-\+]\d)\s"`
- `\s` - matches any whitespace character (tabs, newlines, etc.)
- `\d` - matches any integer

Cracking the code

- `anyNumber = r"\s\d\.\d[eE][\-\+]\d)\s"`
- `\s` - matches any whitespace character (tabs, newlines, etc.)
- `\d` - matches any integer
- `\.` - matches a comma (we have to use \ since . has a special function. (like % in TeX))

Cracking the code

- `anyNumber = r"\s\d\.\d[eE][\-\+]\d)\s"`
- `\s` - matches any whitespace character (tabs, newlines, etc.)
- `\d` - matches any integer
- `\.` - matches a comma (we have to use \ since . has a special function. (like % in TeX))
- `[eE]` - matches either e or E
- `[\+\-]` - matches either + or - (again backslash since + and - has special function.)

Cracking the code

- `anyNumber = r"\s\d\.\d[eE][\-\+]\d)\s"`
- `\s` - matches any whitespace character (tabs, newlines, etc.)
- `\d` - matches any integer
- `\.` - matches a comma (we have to use \ since . has a special function. (like % in TeX))
- `[eE]` - matches either e or E
- `[\+\-]` - matches either + or - (again backslash since + and - has special function.)
- Summarized: Any number is isolated by spaces, starts with an integer, then a comma, then an integer, then e or E, then + or -, then an integer.
- This is obviously not true, but in the *right combinations* it is true. Let's go back to the special characters.

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- `anyNumber =     *(  +  ?  *    ?      ?  *)  *`
- These special characters describe allowed combinations of the previous regexp.

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- `anyNumber =     *(  +  ?  *    ?       ?  *)   *`
- These special characters describe allowed combinations of the previous regexp.
- ? : Optional. It is either there or not. In other words: The comma is optional, e or E is optional, + or - is optional.

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- `anyNumber =     *(  +  ?  *    ?      ?  *)  *`
- These special characters describe allowed combinations of the previous regexp.
- ? : Optional. It is either there or not. In other words: The comma is optional, e or E is optional, + or - is optional.
- + : Not Optional. Allows for multiple successing regexp of the previous statement. \d+ matches any integer.

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- `anyNumber =     *(  +  ?  *    ?      ?  *)  *`
- These special characters describe allowed combinations of the previous regexp.
- ? : Optional. It is either there or not. In other words: The comma is optional, e or E is optional, + or - is optional.
- + : Not Optional. Allows for multiple successing regexp of the previous statement. \d+ matches any integer.
- * : Optional combined with allowing any amount of the previous regex to success eachother.

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- `anyNumber =      *(  +  ?  *   ?      ?  *)  *`
- These special characters describe allowed combinations of the previous regexp.
- `?` : Optional. It is either there or not. In other words: The comma is optional, e or E is optional, + or - is optional.
- `+` : Not Optional. Allows for multiple successing regexp of the previous statement. `\d+` matches any integer.
- `*` : Optional combined with allowing any amount of the previous regex to success eachother.
- `(...)` : Identifies a *group*. When we apply `re.findall`, we want the numbers returned, not the spaces (`\s*`) (even though they are a part of the regexp pattern); we put the matching number in a group.

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- Does this expression allow for negative numbers?

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- Does this expression allow for negative numbers?
- Frank, Svenn-Arne; you are not allowed to answer.

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- Does this expression allow for negative numbers?
- Frank, Svenn-Arne; you are not allowed to answer.
- Anyone else?

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- Does this expression allow for negative numbers?
- Frank, Svenn-Arne; you are not allowed to answer.
- Anyone else?
- You are right! It does not!
- So what do we need to add?

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- Does this expression allow for negative numbers?
- Frank, Svenn-Arne; you are not allowed to answer.
- Anyone else?
- You are right! It does not!
- So what do we need to add?
- A precurring -? (NOTE: Inside the group!)

Cracking the code

- `anyNumber = r"\s*(\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- Does this expression allow for negative numbers?
- Frank, Svenn-Arne; you are not allowed to answer.
- Anyone else?
- You are right! It does not!
- So what do we need to add?
- A precurring `-?` (NOTE: Inside the group!)
- We do not need a backslash since - is only special inside brackets. (`[a-z]` means any character between a and z will match)

Cracking the code

- `anyNumber = r"\s*(-?\d+\.?\d*[eE]?[\-\+]?\d*)\s*"`
- Does this expression allow for negative numbers?
- Frank, Svenn-Arne; you are not allowed to answer.
- Anyone else?
- You are right! It does not!
- What do we need to add?
- A precurring `-?`
- We do not need a backslash since - is only special inside brackets. (`[a-z]` means any character between a and z will match)

Today's show

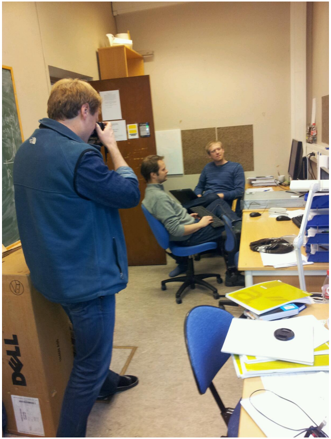
1. Regular expressions. What the..?
2. Examples
    - Retrieving data from raw output
    - Analyzing an Austrian's master thesis

```
...
dmcE: 2.99999| Nw:  998| 92.30000%
dmcE: 3.00000| Nw:  995| 92.40000%
dmcE: 3.00000| Nw:  995| 92.50000%
dmcE: 3.00000| Nw:  996| 92.60000%
...
dmcE: 2.99999| Nw:  998| 94.00000%
dmcE: 3.00000| Nw:  999| 94.10000%
dmcE: 3.00000| Nw:  999| 94.20000%
dmcE: 3.00000| Nw:  998| 94.30000%
dmcE: 3.00000| Nw:  995| 94.40000%
dmcE: 3.00001| Nw:  992| 94.50000%
dmcE: 3.00001| Nw:  993| 94.60000%
...
dmcE: 3.00002| Nw:  984| 99.80000%
dmcE: 3.00001| Nw:  984| 99.90000%
dmcE: 3.00002| Nw:  985| 100.00000%
DMC FIN.
Job fin
```

```
def getDmcE(path):
    stdout = open(path + "/stdout.txt", ’r’)
    stdoutRaw = "\n".join(stdout.readlines())
    stdout.close()

    pattern = "dmcE:\s*(\d+\.?\d*)\s*\|\s*Nw:\s*\d+\|\s*100\.?[0]*%"

    r = re.findall(pattern, stdoutRaw)

    if r:
        #r[0] = therm; r[1] = production

        #Both thermalization and main cycles succeeded.
        if len(r) ==2:
               return float(r[1])
        else:
        #Run aborted after thermalization.
               return "~" + r[0]
    else:
        #Run aborted.
        return "N/A"
```

Today's show

1. Regular expressions. What the..?
2. Examples
   - Retrieving data from raw output
   - Analyzing an Austrian's master thesis


DELL

AUSTRIAN

- Step one: Concatinate all the his tex-files into one file:

```
~$ find . -name *.tex -exec cat {} \; > ~/.../christoffer_raw.tex
```

- Use `count_words.py` to count the accurances of important strings/words such as ...

- Jørgen, Coupled Cluster and Monte Carlo

```
~$ python count_words.py -i -b christoffer_raw.tex Jørgen
Number of occurances of word ’Jørgen’ (case insensitive): 1

~$ python count_words.py -i christoffer_raw.tex "coupled cluster"
Number of occurances of string ’coupled cluster’ (case insensitive): 15

~$ python count_words.py -i christoffer_raw.tex "monte carlo"
Number of occurances of string ’monte carlo’ (case insensitive): 2
```

```
...cml-line parsing...

#Adding bounds to the word (pattern=any spacing, word, any ending)
stringOrWord = "string"
if bounded:
    stringOrWord = "word"
    word = "[\^\s]" + word + "[\s\b]"

#Flaging case insensitivity
if not caseSense:
    regExtObj = re.compile(word, re.IGNORECASE)
    printCase = " (case insensitive)"
else:
    regExtObj = re.compile(word)
    printCase = ""

Nmatches = len(regExtObj.findall(rawFile))

printfArgs = (stringOrWord, rawWord, printCase, Nmatches)
print "Number of occurances of %s '%s'%s: %d" %  printfArgs
```

- Use `list_occurances.py` to reveal his most used words:

```
christoffer_raw.tex
word            : n

electron        : 498
tension         : 182
omega           : 134
basis           : 130
particle        : 126
operators       : 126
operator        : 117
delta           : 114
state           : 102
electrons       : 92
states          : 89
...
```

```
import sys, re

#load entire file into a string. Convert to lower case letters.
f1 = open(sys.argv[1]); allwords = f1.read().lower(); f1.close()

max_length = 15; min_length = 5

#Recognize all words
rePattern = r"[a-zA-Z]{%d,%d}" % (min_length, max_length)
allwords = re.findall(rePattern, allwords)

texWords = ... list of texWords such as begin, end, left, right..

#count words
wordCount = {}
for word in allwords:
    if word not in texWords and not re.findall(’.*(fmf).*’, word):
        if word not in wordCount.keys():
            wordCount[word] = 1
        elif word in wordCount.keys():
            wordCount[word] += 1

#Sort scores from largest to lowest
sort = sorted(wordCount.items(), key=lambda x: x[1], reverse=True)

#Output:
s = max_length
print sys.argv[1].ljust(len(sys.argv[1]))
print "%s: %s" % ("word".ljust(s), "n".ljust(s))
print
for key, value in sort:
    print "%s: %s" % (key.ljust(s), str(value).ljust(s))
```