Regular expressions

Regexp class

Regexp key

Extra exercises:

Regex\_exercises

1.DIY : find years

-eventueel niet doen : oefening om jaartal te vinden zonder regexp

**(10 min)**

## 2. re.findall()

## (5min)

### **DIY: A's and B's**

**(5 min)**

### 3. **What if we need to capture the meta-characters in text?**

**(5min)**

### **3.The Task: Analyze Spam Emails**

Extract information from fraudulent\_email.txt

Completely to be done by students : extract email, extract name on 300 items

# Collect lines starting with "From:"

pattern = "^From:.\*$"

result = re.findall(pattern, file\_content)

print(result)

won’t work without multiline tag

**(20 min)**

### 4. **Greedy vs. non-greedy search**

Principle of list comprehension is explained.

List comprehension to be converted into ‘classic’ for loop

**(5 min)**

From multiline to one line, with list comprehension, and join

To be done by students

Info about list comprehension

[Python Basics: List Comprehensions | by Ventsislav Yordanov | Towards Data Science](https://towardsdatascience.com/python-basics-list-comprehensions-631278f22c40)

**EXTRA (5 min)**

#EXTRA: rewrite aforementioned list comprehension code in a loop:

## ****Basic regex operations/patterns (continued)****

 \s -- (lowercase s) matches a single whitespace character -- space, newline, return, tab, form [ \n\r\t\f].

 \S (upper case S) matches any non-whitespace character.

 \w (lowercase w): matches a "word" character: a letter or digit or underbar [a-zA-Z0-9\_]. Note that although "word" is the mnemonic for this, it only matches a single word char, not a whole word.

 \W (upper case W) matches any non-word character.

 \t, \n, \r -- tab, newline, return

*5.* In "result" list, collect all the first and the last words in each line. The last words are in fact the email addresses surrounded by the angle brackets.

**(10 min)**

Find 1st and last words,

Bit tricky, not to use \w but \S

**Basic regex operations/patterns (continued)**

()

 \d -- decimal digit (0-9)

 {m} -- specifies that exactly m copies of the previous RE should be matched; fewer matches cause the entire RE not to match. For example, a{6} will match exactly six 'a' characters, but not five.

 {m,n} -- causes the resulting RE to match from m to n repetitions of the preceding RE, attempting to match as many repetitions as possible.

 {m,n}? -- causes the resulting RE to match from m to n repetitions of the preceding RE, attempting to match as few repetitions as possible. This is the non-greedy version of the previous qualifier. For example, on the 6-character string 'aaaaaa', what will the following patterns match?

Difficult one to explain:

['aa', 'aa', 'aa', '']

"a{,2}"

**(10 min)**

Now, back to analysing the e-mails!

6. collect all (and only) the years appearing in the lines that start with:

(10 min)

7.FLAGS

Exercise re.dotall

**(5 min)**

## DIY: Basic regex operations

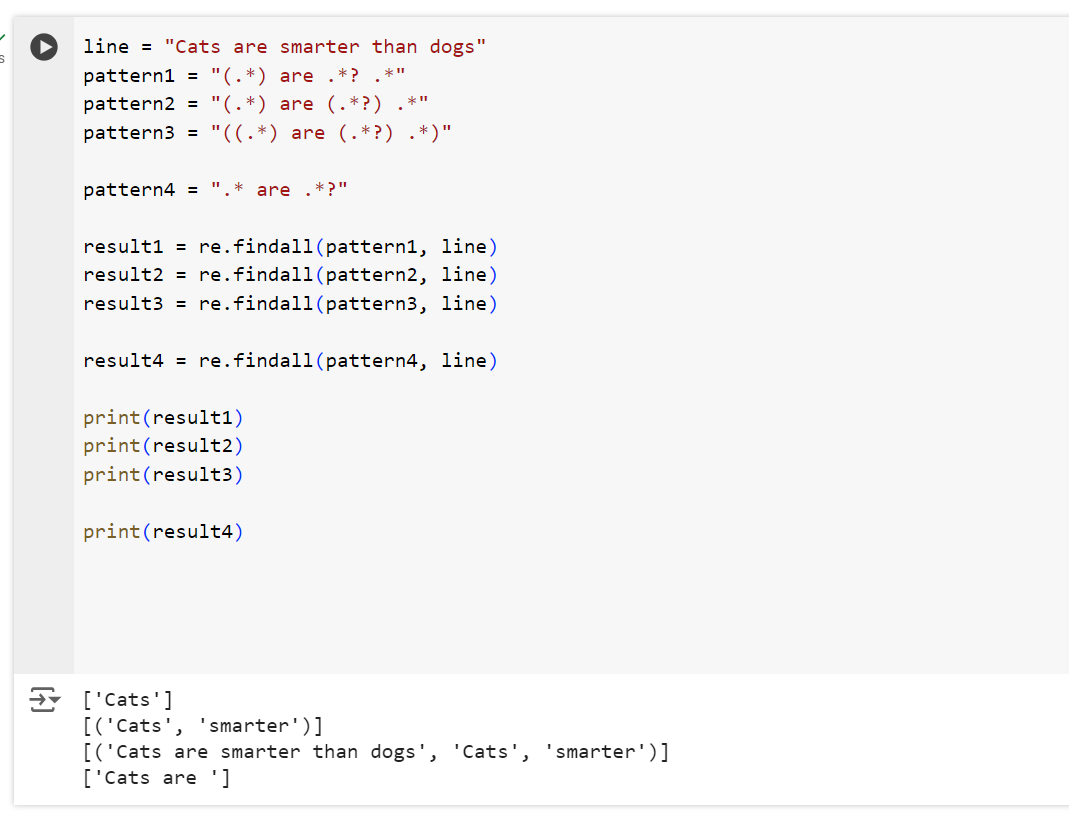
(DIY 1-5)

**(15 min)**

GROUPS

**(10 min)**

To well understand



In pattern 4

.\*?

This is NON greedy matching

### **DIY. Groups**

**(10 min)**

LOOKING AROUND OPTIONAL (15 MIN)

SPLIT and substitution

**(15 min)**

**TOTAL 130 min**

**EXTRA**

**Regex\_exercises**

**Assignment: dictionary for emails**

**Creative opdracht, eliza robot.**

**PROJECT**

Clean data set with regexp:

-imdb film data set

-CV data set

**To continue with**

[imdb-dataset | Text Classification (kaggle.com)](https://www.kaggle.com/code/uvinir/imdb-dataset-text-classification) for stemming