ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI
SCIENZE STATISTICHE "PAOLO FORTUNATI"

4 - Basic Text Analysis

Giovanni Della Lunga
giovanni.dellalunga@unibo.it

Halloween Conference in Quantitative Finance

Bologna - October 26-28, 2021

We will talk about...

- What is text mining
- What is a Corpus
- Creating your own corpus
- Regular Expressions

Subsection 1

What is Text Mining

And why is so important!

What is Text Mining

- At some level, text analysis is the act of breaking up larger bodies of work into their constituent components - unique vocabulary words, common phrases, syntactical patterns - then applying statistical analysis to them;
- We will soon see that there are many levels to which we can apply our analysis, all of which revolve around a central text dataset: the **corpus**.

What is a Corpus

- Corpora are collections of related documents that contain natural language.
- A corpus can be large or small, though generally they consist of dozens or even hundreds of gigabytes of data inside of thousands of documents.
- A corpus can be broken down into categories of documents or individual documents.

What is a Corpus

- Corpora can be **annotated** *meaning that the text or documents are labeled with the correct responses for supervised learning algorithms* , or **unannotated**, making them candidates for topic modeling and document clustering.
- For example, one common type of annotation is the addition of tags, or labels, indicating the word class to which words in a text belong

```
sentence = "My name is Giovanni"
token = nltk.word_tokenize(sentence)
token
```

```
['My', 'name', 'is', 'Giovanni']
```

```
nltk.pos_tag(token)
```

```
[('My', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), ('Giovanni', 'NNP')]
```

Domain Specific Corpora

- The best applications tend to use language models trained on domain-specific corpora (collections of related documents containing natural language).
- The reason for this is that language is highly contextual, and words can mean different things in different contexts.
- For example, depending on context, the word **bank** can refer a place where you put your money, the side of a river, the surface of a mine shaft, or the cushion of a pool table!
- With domain-specific corpora, we can reduce ambiguity and prediction space to make results more intelligible.

Subsection 2

Create your own corpus

Getting text from the web

Web Scraping

- The need and importance of extracting data from the web is becoming increasingly loud and clear.
- There are several ways to extract information from the web.
- Use of APIs being probably the best way to extract data from a website. If you can get what you need through an API, it is almost always preferred approach over web scrapping.

Web Scraping

- Sadly, not all websites provide an API.
- Some do it because they do not want the readers to extract huge information in structured way, while others do not provide APIs due to lack of technical knowledge. What do you do in these cases?
- Well, we need to scrape the website to fetch the information.

Web Scraping

- Ok, but what is Web Scraping?
- Web scraping is a computer software technique of extracting information from websites.
- This technique mostly focuses on the transformation of unstructured data (HTML format) on the web into structured data (database or spreadsheet).
- You can perform web scrapping in various way.
- We will resort to Python because of its ease and rich ecosystem. It has a library known as **BeautifulSoup** which assists this task.

Downloading Files from the Web with the requests Module

- The requests module lets you easily download files from the Web without having to worry about complicated issues such as network errors, connection problems, and data compression.
- The requests module does not come with Python, so you will have to install it first. From the command line, run pip install requests.
- Next, do a simple test to make sure the requests module installed itself correctly. Enter the following into the interactive shell:
- ¿¿¿ import requests
- If no error messages show up, then the requests module has been successfully installed.

Downloading a Web Page with the requests.get() Function

- The requests.get() function takes a string of a URL to download.
- By calling type() on requests.get() return value, you can see that it returns a Response object, which contains the response that the web server gave for your request

```
>>> import requests
>>> res = requests.get('https://automatetheboringstuff.com/files/rj.txt')
>>> type(res)
<class 'requests.models.Response'>
>>> res.status_code == requests.codes.ok
True
>>> len(res.text)
178981
>>> print(res.text[:250])
```

Libraries required for web scraping

- Urllib2: It is a Python module which can be used for fetching URLs.
- It defines functions and classes to help with URL actions (basic and digest authentication, redirections, cookies, etc).
- For more detail refer to the documentation page.
- BeautifulSoup: It is an incredible tool for pulling out information from a webpage.
- You can use it to extract tables, lists, paragraph and you can also put filters to extract information from web pages.
- You can look at the installation instruction in its documentation page.

Example : Processing Raw Text

- Notebook:
  04-basic-text-analysis
- Focus: Processing HTML Files
- Libraries: NLTK, URLIB, BeautifulSoup

Subsection 3

Regular Expressions

Regular Expressions

- In computing, a regular expression, also referred to as "regex" or "regexp", provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters.
- A regular expression is written in a formal language that can be interpreted by a regular expression processor.
- Really clever "wild card" expressions for matching and parsing strings.

Regular Expressions

- Very powerful and quite cryptic
- Fun once you understand them
- Regular expressions are a language unto themselves
- A language of "marker characters" - programming with characters

Regular Expressions quick guide

^	Matches the beginning of a line
$	Matches the end of the line
.	Matches any character
\s	Matches whitespace
\S	Matches any non-whitespace character
*	Repeats a character zero or more times
*?	Repeats a character zero or more times (non-greedy)
+	Repeats a character one or more times
+?	Repeats a character one or more times (non-greedy)
[aeiou]	Matches a single character in the listed set
[^XYZ]	Matches a single character not in the listed set
[a-z0-9]	The set of characters can include a range
(	Indicates where string extraction is to start
)	Indicates where string extraction is to end

Regular Expressions Module

- Before you can use regular expressions in your program, you must import the library using **import re**
- You can use **re.search()** to see if a string matches a regular expression similar to using the **find()** method for strings
- You can use **re.findall()** extract portions of a string that match your regular expression similar to a combination of **find()** and slicing.

Wild-Card Characters

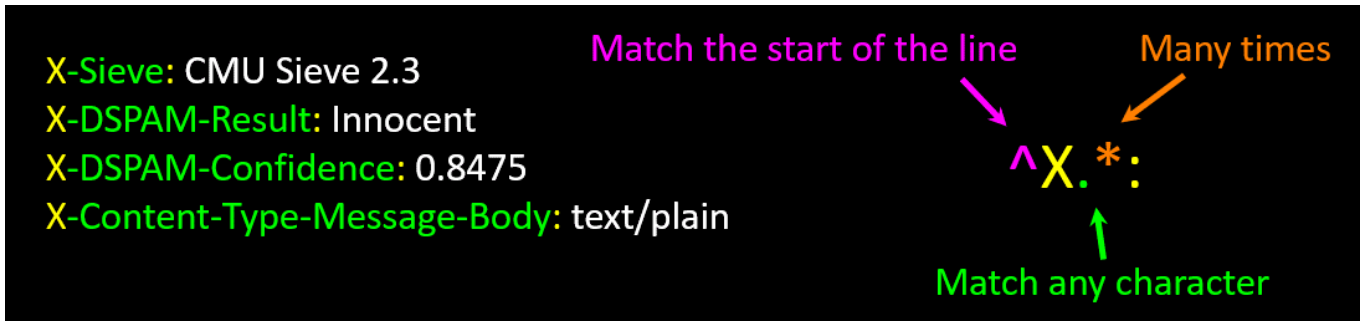
- The dot character matches any character
- If you add the asterisk character, the character is "any number of times"

X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-DSPAM-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain

^X.*:

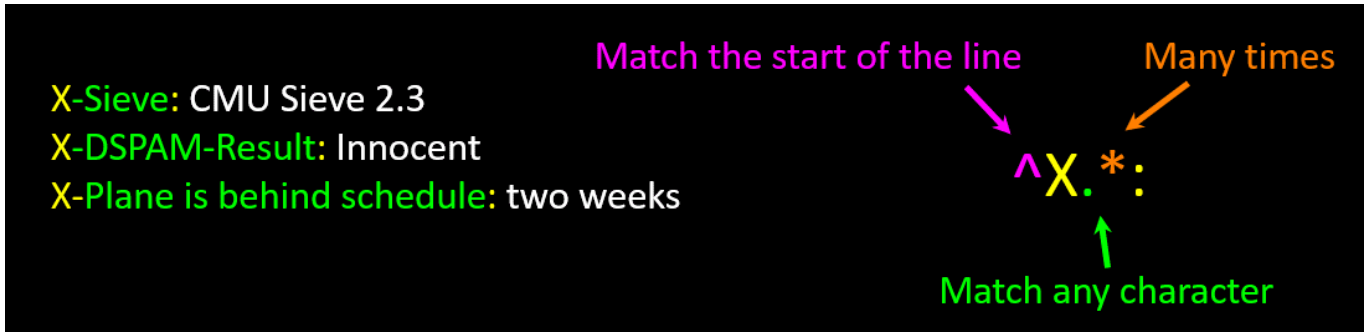
Wild-Card Characters

- The dot character matches any character
- If you add the asterisk character, the character is "any number of times"



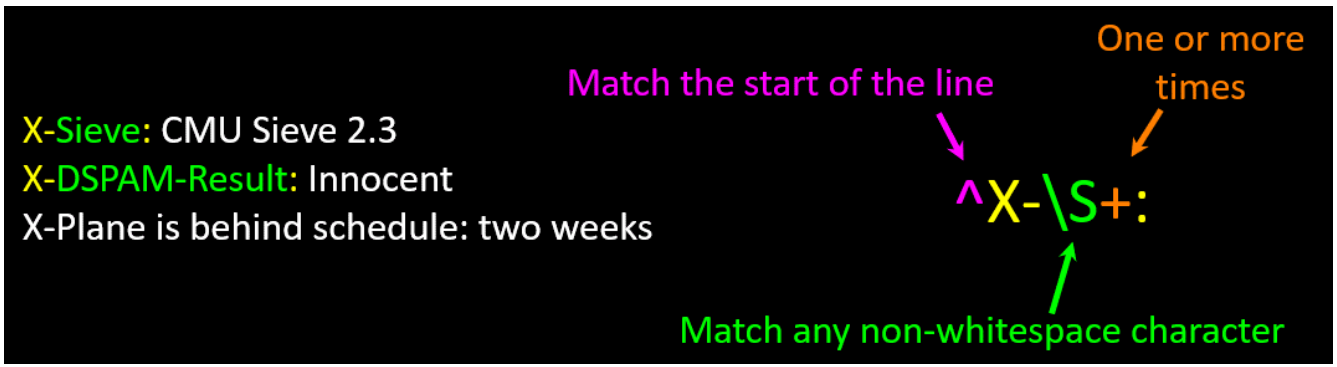
Fine-Tuning Your Match

Depending on how "clean" your data is and the purpose of your application, you may want to narrow your match down a bit



Fine-Tuning Your Match

Depending on how "clean" your data is and the purpose of your application, you may want to narrow your match down a bit



Matching and Extracting Data

- The re.search() returns a True/False depending on whether the string matches the regular expression
- If we actually want the matching strings to be extracted, we use re.findall()

[0-9]+

One or more digits

```
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'
>>> y = re.findall('[0-9]+',x)
>>> print y
['2', '19', '42']
```

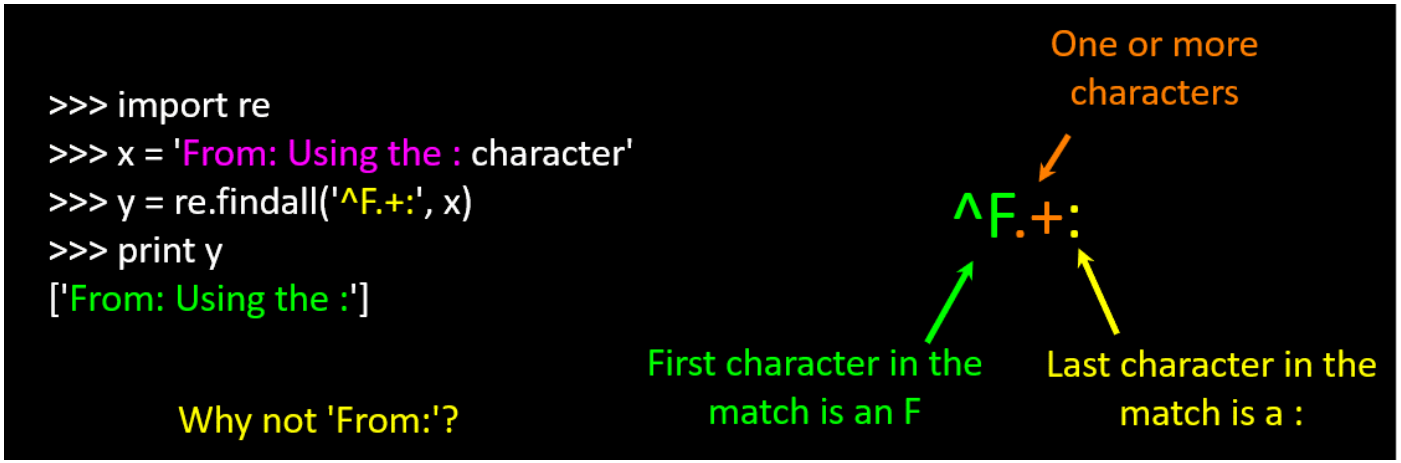
Matching and Extracting Data

When we use re.findall() it returns a list of zero or more sub-strings that match the regular expression

```
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'>>> y = re.findall('[0-9]+',x)>>> print y['2', '19', '42']
>>> y = re.findall('[AEIOU]+',x)
>>> print y
[]
```

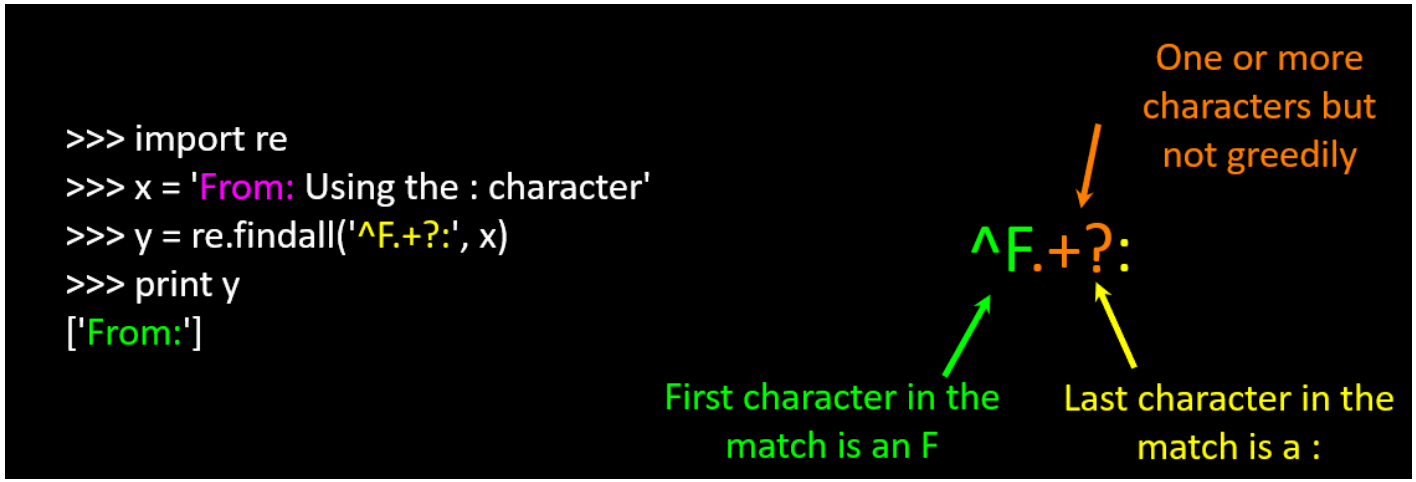
Warning: Greedy Matching

The repeat characters (* and +) push outward in both directions (greedy) to match the largest possible string



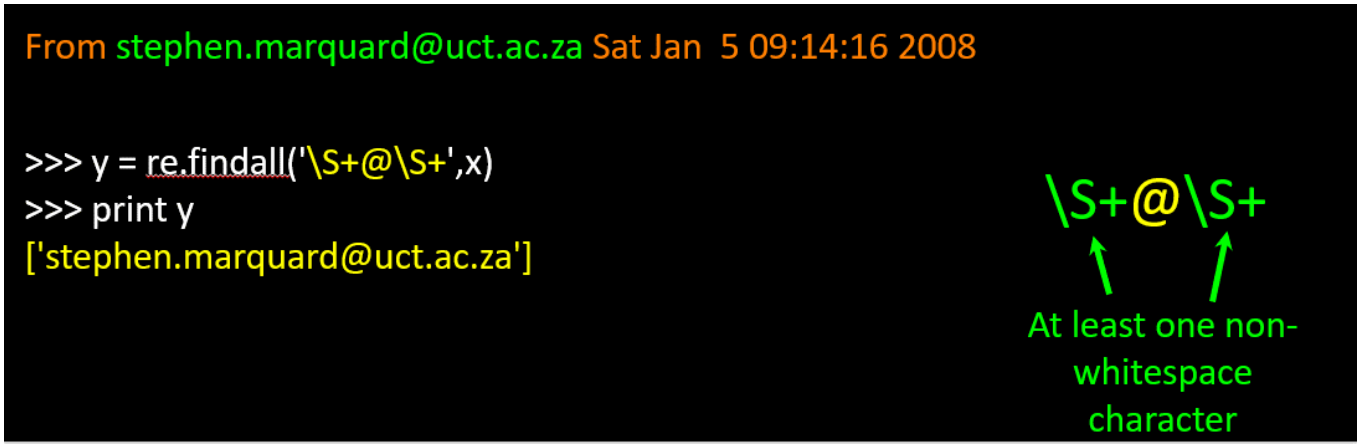
Non-Greedy Matching

Not all regular expression repeat codes are greedy! If you add a ? character - the + and * chill out a bit...



Fine Tuning String Extraction

You can refine the match for re.findall() and separately determine which portion of the match that is to be extracted using parenthesis

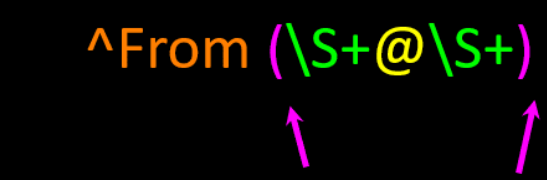


Fine Tuning String Extraction

Parenthesis are not part of the match - but they tell where to start and stop what string to extract

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

>>> y = re.findall('\S+@\S+',x)
>>> print y
['stephen.marquard@uct.ac.za']
>>> y = re.findall('^From (\S+@\S+)',x)
>>> print y
['stephen.marquard@uct.ac.za']
```



The Regex Version

Sometimes we split a line one way and then grab one of the pieces of the line and split that piece again

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('@([^ ]*)',lin)
print y['uct.ac.za']
```

'@([^]*)'

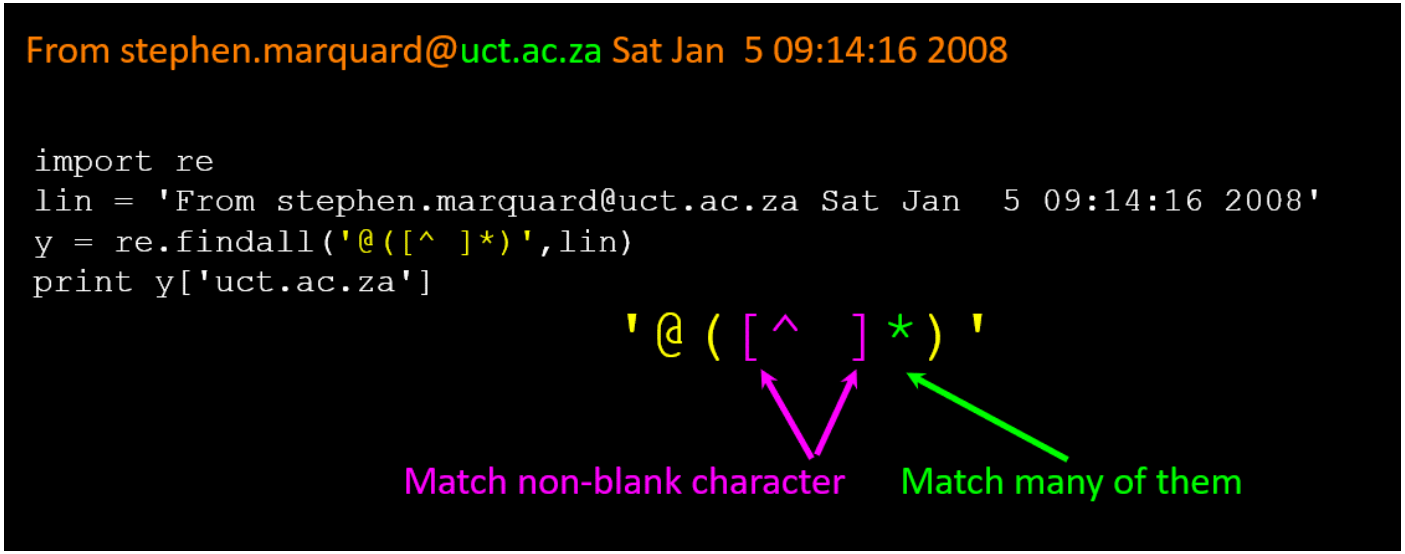
Look through the string until you find an at-sign

The Regex Version

Sometimes we split a line one way and then grab one of the pieces of the line and split that piece again

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('@([^ ]*)',lin)
print y['uct.ac.za']
```



The Regex Version

Sometimes we split a line one way and then grab one of the pieces of the line and split that piece again

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

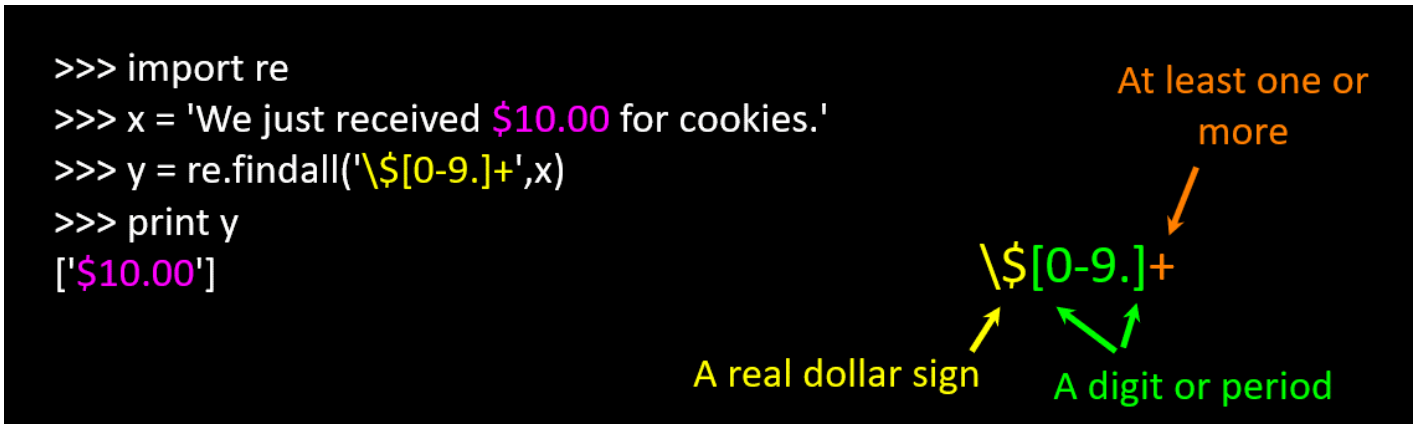
```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('@([^ ]*)',lin)
print y['uct.ac.za']
```

'@([^]*)'

Extract the non-blank characters

Escape Character

If you want a special regular expression character to just behave normally (most of the time) you prefix it with ’
’



Example : Regular Expressions

- Notebook: 04-basic-text-analysis
- Focus: Using Regular Expressions
- Libraries: python re module