CSCI 3450: NLP
Fall 2022

Topic: Regular Expression and Text Processing
Prepared by: Fahmida Hamid

12-Sept-2022

1 Introduction

Regular expression (RE) is a language for specifying text search strings. This practical language is used in every computer language, word processor, and text processing tools like the Unix tools *grep* or Emacs.

Definition 1. *A regular expression is an algebraic notation for characterizing a set of strings.*

Regular expressions come in many variants. We'll be describing regular expressions recognized by Python's *re* library. We will use the following sample (*sent*) as our test string.

1.1 Initialization

```
import re
sent = "This is a test sentence that contains 11 numbers & \
it was created on 09/02/2020 evening.  \
Can you extract some useful information? this (10-31-2222) \
is not a date though! Nothing new about it :) \
1990-34-12 is another way. Try to extract email \
ids like fhamid@ncf.edu, pioneer111@yahoo.com, \
or binary@gmail.com."
```

1.2 Expression 01: Exact matching

```
re.findall(r'[Tt]his', sent)

>>>['This', 'this']
```

- Note:
  - I am using *Jupyter Notebook*. If you are using other python IDE you will need to call the *print*() method in order to print the output.
  - $[Tt]$ means match with either character T or t.
  - $[T|t]$ is equivalent to $[Tt]$

1.3 Expression 02: Alphanumeric character, including _ (underscore)

```
chars = re.findall(r"\w", sent)
print(chars)

>>> ['T', 'h', 'i', 's', 'i', 's', 'a', 't', ...
```

1.4 Expression 03: A sequence of Alphanumeric characters

```
words = re.findall(r"\w+", sent)
print(words)

>>>['This', 'is', 'a', 'test', 'sentence', 'that', 'contains', '11',
'numbers', 'it', 'was', 'created', 'on', '09', '02', '2020',
'evening', 'Can', 'you', 'extract', 'some', 'useful', 'information',
'this', '10', '31', '2222', 'is', 'not', 'a', 'date', 'though',
'Nothing', 'new', 'about', 'it', '1990', '34', '12', 'is',
'another', 'way', 'Try', 'to', 'extract', 'email', 'ids',
'like', 'fhamid', 'ncf', 'edu', 'pioneer111', 'yahoo',
'com', 'or', 'binary', 'gmail', 'com']
```

- Note
  - Carefully note the difference between $\backslash w$ and $\backslash w+$.
  - $\backslash w$ stands for alphanumeric characters including _.
  - $+$ indicates 1 or more occurrences of a pattern.
  - $*$ indicates 0 or more occurrences of a pattern.
  - after r, we use single or double quotes to represent the pattern.

1.5 Expression 04: Anything but letters, excluding _

```
special_chars = re.findall(r"\W+", sent)
print(special_chars)

>>>[' ', ' ', ... '&' , ' ', ..., '? ', ...]
```

Note: $\backslash W$ stands for anything but alphanumeric characters, excluding _.

1.6 Expression 05: Extracting Digits and Numbers

```
digits = re.findall(r"\d", sent)
print(digits)
>>>['1', '1', '0', '9', '0', '2', ...]
numbers = re.findall(r"\d+", sent)
print(numbers)
>>> ['11', '09', '02', '2020', '10', '31', '2222', '1990',
'34', '12', '111']
```

1.7 Expression 06: Extracting Dates

```
dates = re.findall(r'\d+/\d+/\d+', sent)
print(dates)

>>>['09/02/2020']
various_dates = re.findall(r'\d{2}[/-]\d{2}[/-]\d{4}|
                                        \d{4}[/-]\d{2}[/-]\d{2}',
                                        sent)
print(various_dates)

>>>['09/02/2020', '10-31-2222', '1990-34-12']
```

- Note:
  - $\{n\}$ means exactly n repeats where n is a positive integer.
  - $\{n,\}$ means at least n repeats
  - $\{,n\}$ means no more than n repeats

1.8 Expression 07: Substrings that contain at least one n

```
words_with_an_n = re.findall(r'[nN]\w+', sent)
print(words_with_an_n)

>>>['ntence', 'ntains', 'numbers', 'ning', 'nformation', 'not',
'Nothing', 'new', 'nother', 'ncf', 'neer111', 'nary']
```

1.9 A white space and n

```
words_with_space_and_n = re.findall(r'\s[nN]\w+', sent)
print(words_with_space_and_n)

>>>[' numbers', ' not', ' Nothing', ' new']
```

1.10 Expression 09: Boundary

```
boundary_words = re.findall(r'\b[cC]\w+', sent)
print(boundary_words)

>>>['contains', 'created', 'Can', 'com', 'com']

sent2 = "Testing a new sentence. Object-oriented
design should have a hyphen."

words = re.findall(r'\w+[e|g]n\b', sent2)
print(words)

>>>['design', 'hyphen']
```

- Note:
  - $\backslash b$ indicates boundary.

1.11 Expression 10: Hyphenated words

```
hwords = re.findall(r'\w+[-]?\w+', sent2)
print(hwords)

>>>['Testing', 'new', 'sentence', 'Object-oriented',
'design', 'should', 'have', 'hyphen']
```

- Note:
  - ? indicates 0 or 1 occurrence of a character/pattern.

1.12 Expression 11: Extracting two consecutive words per Sentence

```
bigrams = re.findall(r'\w+[\s|-|/]\w+', sent)
print(bigrams)

>>>['This is', 'a test', 'sentence that', 'contains 11', 'it was',
'created on', '09/02', '2020 evening', 'Can you', 'extract some',
'useful information', 'is not', 'a date', 'Nothing new',
'about it', '12 is', 'another way', 'Try to', 'extract email',
'ids like', 'or binary']
```

1.13 Exercise 12: Email id Extraction

```
emails = re.findall(r'\w+\@\w+.\w{3}', sent)
print(emails)

>>>['fhamid@ncf.edu', 'pioneer111@yahoo.com', 'binary@gmail.com']

testemails = "these (rosy.gray@amazon.co.uk) \
or rosy.gray@amazon.com are also valid emails."

emails2 = re.findall(r'\w+[.]?\w+@\w+[.]\w+[.]?\w+' , testemails)
print(emails2)

>>>['rosy.gray@amazon.co.uk', 'rosy.gray@amazon.com']
```

Class Activity 04, Task 01

1.14 Task 01: 20 points

Write necessary regular expressions to find patterns from the following text (*string*). Submit your source code (Name it *class_activity_04task01.py*). Copy and paste your tested result after each regular expression as multiline comments in the source file.

```
string = "This test001 is a simple test. \
You should be able to parse it by 4:50 PM. \
If needed, check the references. The regular \
```

```
time formats may also look like 12:12:12. \
12xyz or 12.34 is not a valid variable name in \
C/C++/Java. Python is a way more flexible \
than Java. Python may accept 12xyz as a valid name. \
Earlier, people used to consider \
C as the state-of-the art for learning a programming \
language. Now, C is mostly used for low-level \
programming. Extracting urls like \
(https://docs.python.org/3/library/re.html) is not easy. \
 We are using Python 3.5 for our class."
```

---

1. Extract all the time-related information (4:50, 12:12:12).
2. Extract words that contain some digits along with other alphanumeric characters (e.g. '12xyz' or 'test001's).
3. Find all the words that start with letter l or L.
4. Find all the words of at least length 4 that start with letter l or L.
5. Find words that contain at least two vowels.
6. Find words that start with an Uppercase letter.
7. Find numbers (decimals and floating points).
8. Extract urls.
9. Extract domain names from an url
10. Extract domain names from an email id.

2 Beautiful Soup

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work. Your next class activity is to install Beautiful Soup (check the reference) on your machine and use it to clean any html file. Once cleaned, run POS-tagger and list the nouns. You may decide to store the html file locally or access/read it directly using *urllib* or any other suitable python libraries that will let you get the *html*/*htm*/*xml* file.

Class Activity 04, Task 02

2.1 Task 02: 10 points

Write a script (Python code, (Name it *class_activity_04task02.py*)) to access the url: https://en.wikipedia.org/wiki/Virus. Then extract all *Nouns* from the text and all the *url links*. Print them. While printing the nouns, add frequency. You may apply lemmatization.

3 Reference

- Regular Expression
- Beautiful Soup