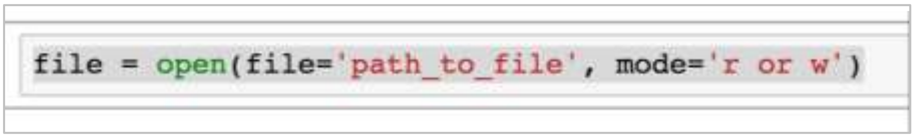
5.3 Natural Language Processing with Python

Topics Covered:

- Working with files
- Natural Language Toolkit (NLTK)
    - Sentence Splitting
    - Tokenisation
    - POS Tagging
    - Bigram Extraction

Activity 1: Working with Files

Python handles files natively so you don’t need to import any special libraries whilst working with files. As in any other languages, you firstly need to open a file, then do some processing and finally close the file. In order to open a file, use the open function as show in the figure below:

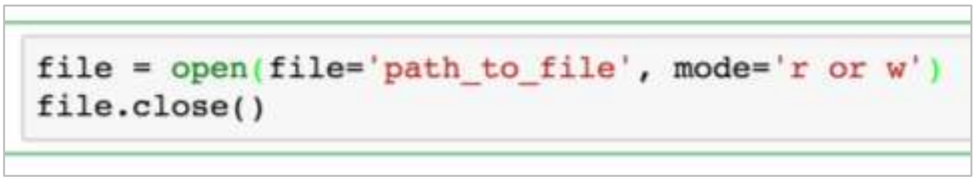


```
file = open(file='path_to_file', mode='r or w')
```

The open function has three main arguments:

File: path to the file you want to process
Mode: access mode of that file
  r: reading the file
  w: writing into file

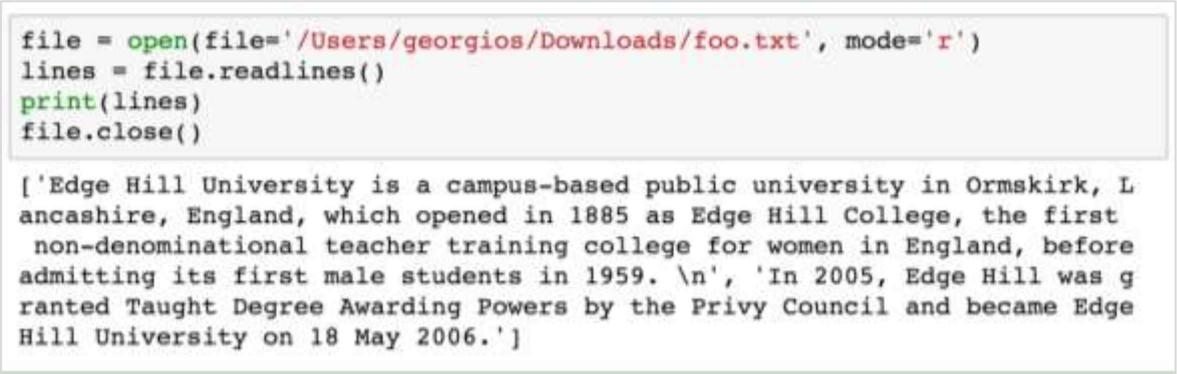
Once you have finished processing the file, you the close function as follows:



```
file = open(file='path_to_file', mode='r or w')
file.close()
```

1.1: Reading from a text file

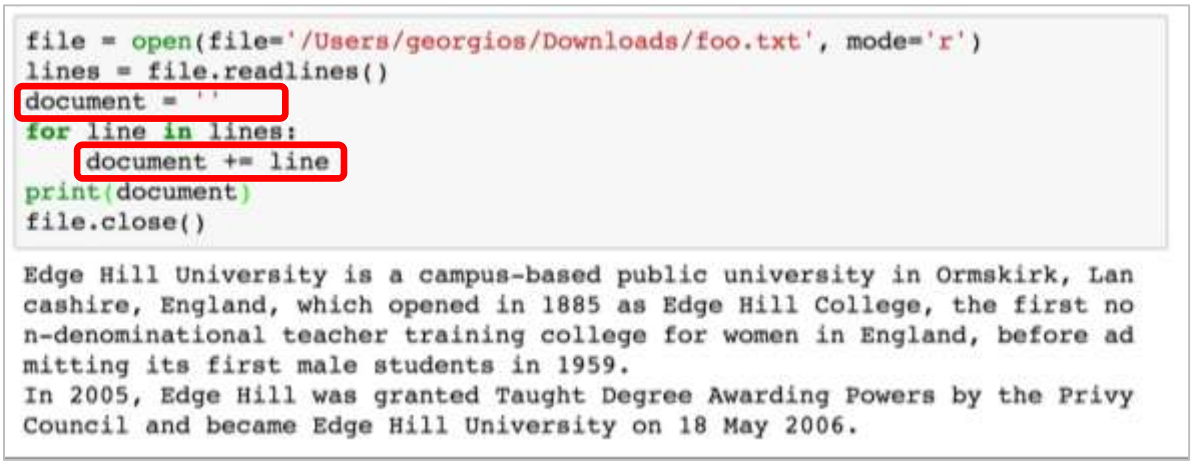
a) Firstly, download the text file **foo.txt** save it into your hard disk.
b) In order to read the content of the foo.txt file we will use the readlines() function which reads a file line by line and stores the content of the file into a list. Each element of the list corresponds to a line.



```
file = open(file='/Users/georgios/Downloads/foo.txt', mode='r')
lines = file.readlines()
print(lines)
file.close()
```

```
['Edge Hill University is a campus-based public university in Ormskirk, L
ancashire, England, which opened in 1885 as Edge Hill College, the first
 non-denominational teacher training college for women in England, before
admitting its first male students in 1959. \n', 'In 2005, Edge Hill was g
ranted Taught Degree Awarding Powers by the Privy Council and became Edge
Hill University on 18 May 2006.']
```

c) If you want to store the content of a file into a variable rather than a list, iterate through lines and append each line to the variable as shown below

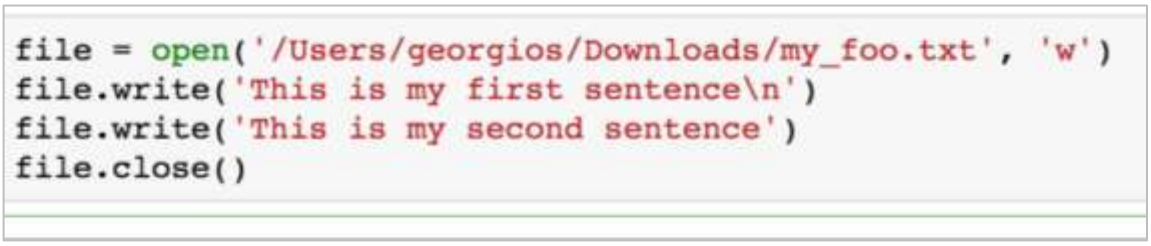


```
file = open(file='/Users/georgios/Downloads/foo.txt', mode='r')
lines = file.readlines()
document = ''
for line in lines:
    document += line
print(document)
file.close()
```

```
Edge Hill University is a campus-based public university in Ormskirk, Lan
cashire, England, which opened in 1885 as Edge Hill College, the first no
n-denominational teacher training college for women in England, before ad
mitting its first male students in 1959.
In 2005, Edge Hill was granted Taught Degree Awarding Powers by the Privy
Council and became Edge Hill University on 18 May 2006.
```

1.2: Writing into a text file

- In order to write content into a file you can use the write() function as shown below:



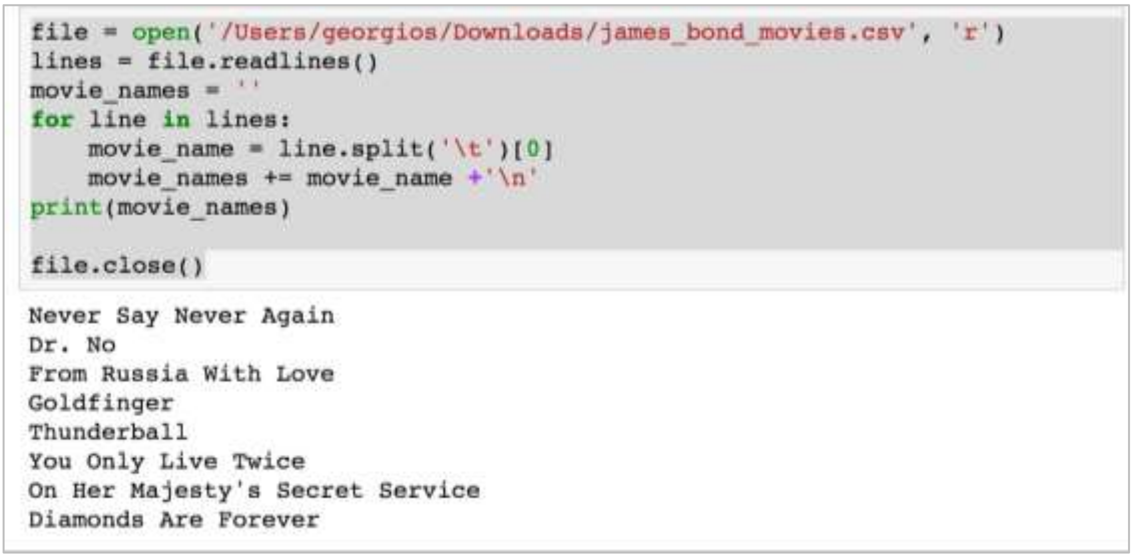
```
file = open('/Users/georgios/Downloads/my_foo.txt', 'w')
file.write('This is my first sentence\n')
file.write('This is my second sentence')
file.close()
```

1.3: Reading from a CSV file

CSV is a popular format for making available tabular data. Each line in a CSV file determines a record while each record consists of different fields (i.e., columns) that are separated by character delimiter (e.g., tab, space, comma.).

- Download the csv james_bond_movies.csv it into your hard disk.
    - *The james_bond_movies.csv contains 8 record (8 lines) and each line consists of two fields: the name of the movie and the director of the movie.*
- In order to read a csv file you can use the **readlines()** function as explained above. If you are interested in a specific field of the CSV file you can split each line using the character delimiter.

- *The script below shows how you can read only the movie names from the james_bond_movies.csv file*



```
file = open('/Users/georgios/Downloads/james_bond_movies.csv', 'r')
lines = file.readlines()
movie_names = ''
for line in lines:
    movie_name = line.split('\t')[0]
    movie_names += movie_name +'\n'
print(movie_names)

file.close()
```

```
Never Say Never Again
Dr. No
From Russia With Love
Goldfinger
Thunderball
You Only Live Twice
On Her Majesty's Secret Service
Diamonds Are Forever
```

Exercise 2.1: Write a Python script that:

- reads the james_bond_movies.csv file
- and stores the director names into:
  1) a list
     and
  2) into a single variable

Exercise 2.2: Write a Python script that:

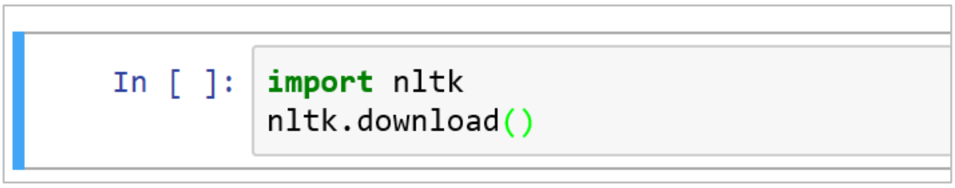
- reads the director names from the james_bond_movies.csv file and saves the director names into a new file called 'director_names.txt'

Activity 2: Using the Natural Language Toolkit (NLTK)

NLTK is a popular library in Python for developing Natural Language Processing applications. NLTK is readily available within your python installation but we need to install some additional modules.

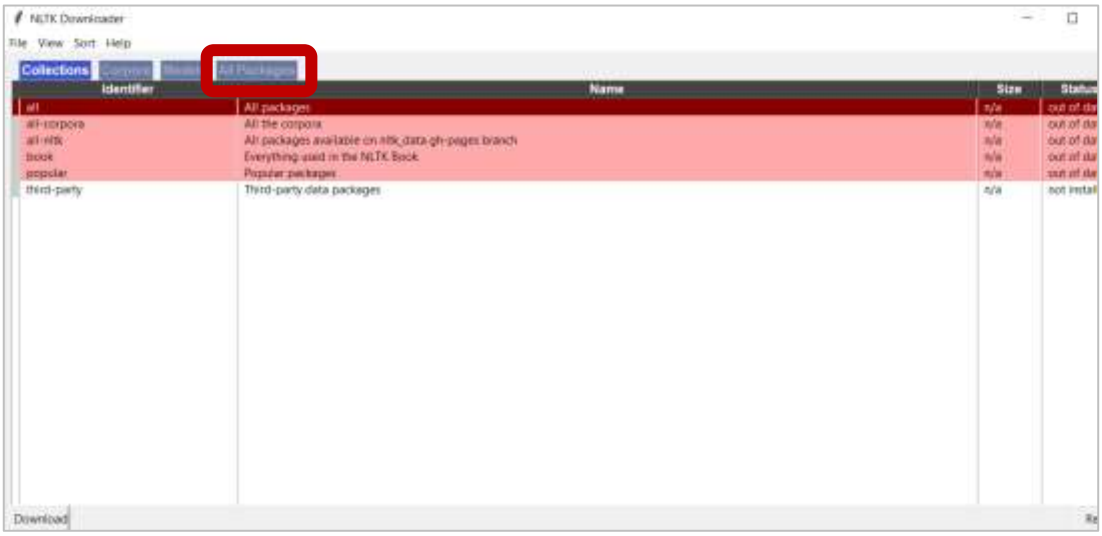
2.1: Setting up the NLTK

- Create a new Python3 file and enter the commands as shown below
- Execute the following script



```
In [ ]: import nltk
        nltk.download()
```

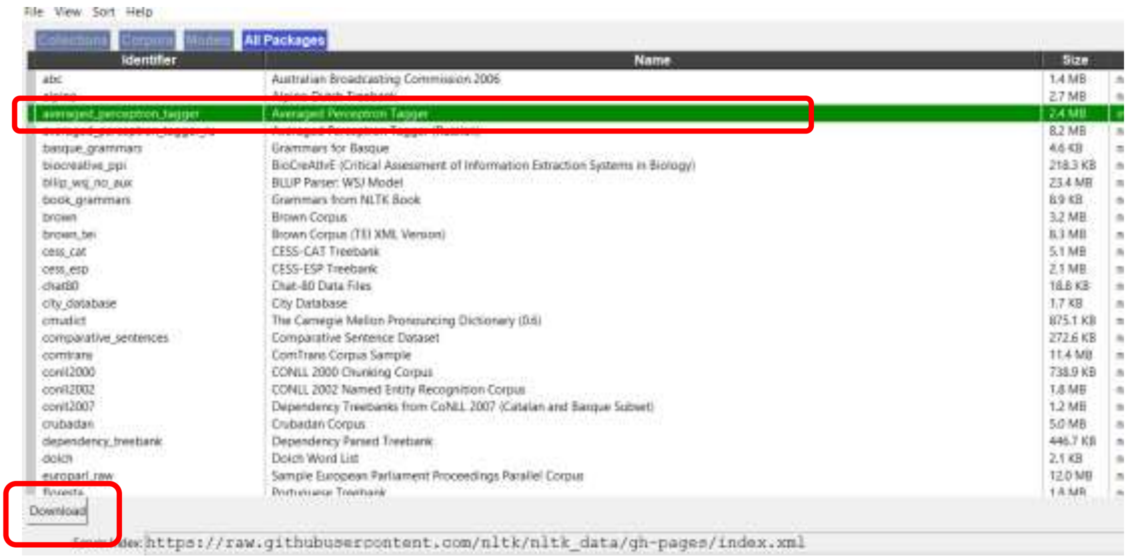
- In the new window that appears, go to the ‘All Packages’ tab



Identifier	Name	Size	Status
all	All packages	n/a	out of da
all-corpora	All the corpora	n/a	out of da
all-nltk	All packages available on nltk_data gh-pages branch	n/a	out of da
book	Everything used in the NLTK Book	n/a	out of da
popular	Popular packages	n/a	out of da
third-party	Third-party data packages	n/a	not instal

- And download the following three packages:

1. ‘averaged_perceptron_tagger’ 2 .‘punkt’ 3. stopwords



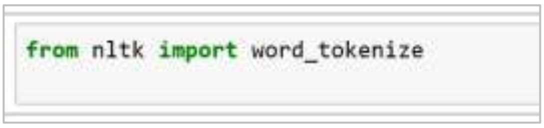
Identifier	Name	Size
abc	Australian Broadcasting Commission 2006	1.4 MB
alpino	Alpino Dutch Treebank	2.7 MB
averaged_perceptron_tagger	Averaged Perceptron Tagger	2.4 MB
averaged_perceptron_tagger_ru	Averaged Perceptron Tagger (Russian)	8.2 MB
basque_grammars	Grammars for Basque	4.6 KB
biocreative_ppi	BioCreAtIvE (Critical Assessment of Information Extraction Systems in Biology)	218.3 KB
bllip_wsj_no_aux	BLLIP Parser: WSJ Model	23.4 MB
book_grammars	Grammars from NLTK Book	8.9 KB
brown	Brown Corpus	3.2 MB
brown_tei	Brown Corpus (TEI XML Version)	8.3 MB
cess_cat	CESS-CAT Treebank	5.1 MB
cess_esp	CESS-ESP Treebank	2.1 MB
chat80	Chat-80 Data Files	18.8 KB
city_database	City Database	1.7 KB
cmudict	The Carnegie Mellon Pronouncing Dictionary (0.6)	875.1 KB
comparative_sentences	Comparative Sentence Dataset	272.6 KB
comtrans	ComTrans Corpus Sample	11.4 MB
conll2000	CONLL 2000 Chunking Corpus	738.9 KB
conll2002	CONLL 2002 Named Entity Recognition Corpus	1.8 MB
conll2007	Dependency Treebanks from CoNLL 2007 (Catalan and Basque Subset)	1.2 MB
crubadan	Crubadan Corpus	5.0 MB
dependency_treebank	Dependency Parsed Treebank	446.7 KB
dolch	Dolch Word List	2.1 KB
europarl_raw	Sample European Parliament Proceedings Parallel Corpus	12.0 MB
floresta	Portuguese Treebank	1.4 MB

https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml

2.2: Tokenisation in NLTK

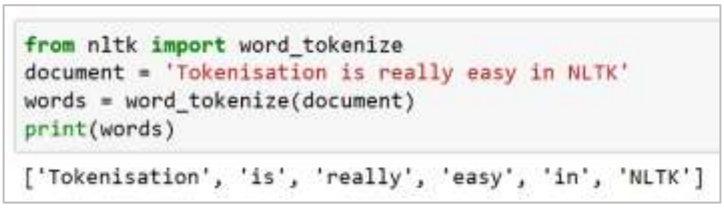
NLTK allows you to perform tokenisation using one command only

- First import the word_tokenize function using the command:



```
from nltk import word_tokenize
```

- The word_tokenize() function takes as an input a document (single string variable not list) and returns a list of words contained in that document



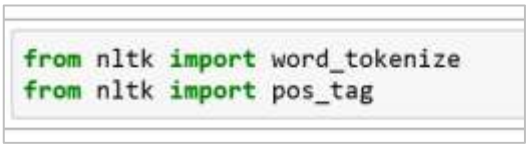
```
from nltk import word_tokenize
document = 'Tokenisation is really easy in NLTK'
words = word_tokenize(document)
print(words)
```

```
['Tokenisation', 'is', 'really', 'easy', 'in', 'NLTK']
```

2.3: Part-of-speech (POS) tagging in NLTK

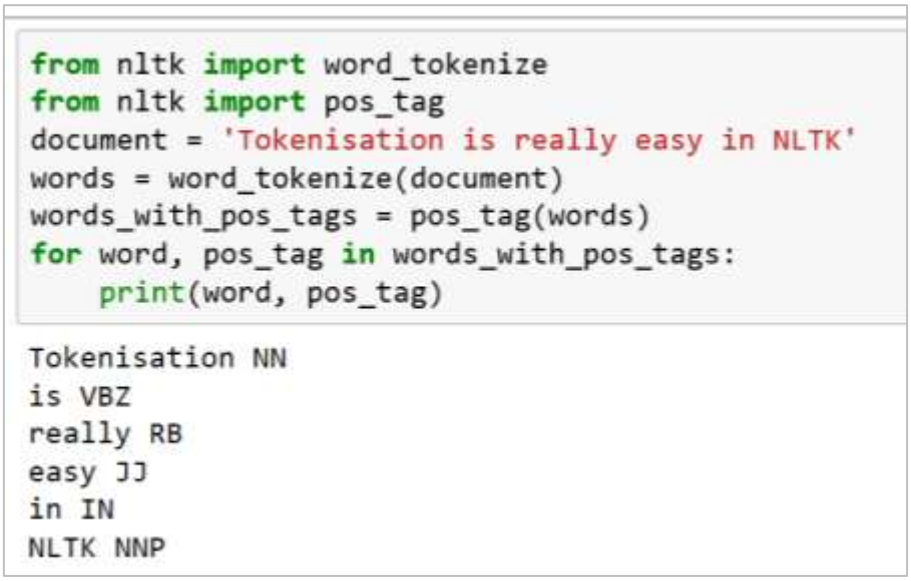
Similar to tokenisation, POS tagging is performed with one command only

- First import the pos_tag function using the command: ‘from nltk import pos_tag’



```
from nltk import word_tokenize
from nltk import pos_tag
```

- The pos_tag() function takes as an input a list of words, previously extracted using the word_tokenise() function and it returns a list of tuples, where the first item of the tuple is the word and the second item the POS tag



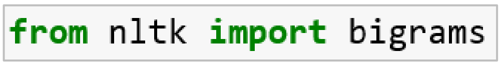
```
from nltk import word_tokenize
from nltk import pos_tag
document = 'Tokenisation is really easy in NLTK'
words = word_tokenize(document)
words_with_pos_tags = pos_tag(words)
for word, pos_tag in words_with_pos_tags:
    print(word, pos_tag)
```

```
Tokenisation NN
is VBZ
really RB
easy JJ
in IN
NLTK NNP
```

Activity 3 - Extracting Bigrams in NLTK

3.1 Using only one NLTK command you can extract bigrams

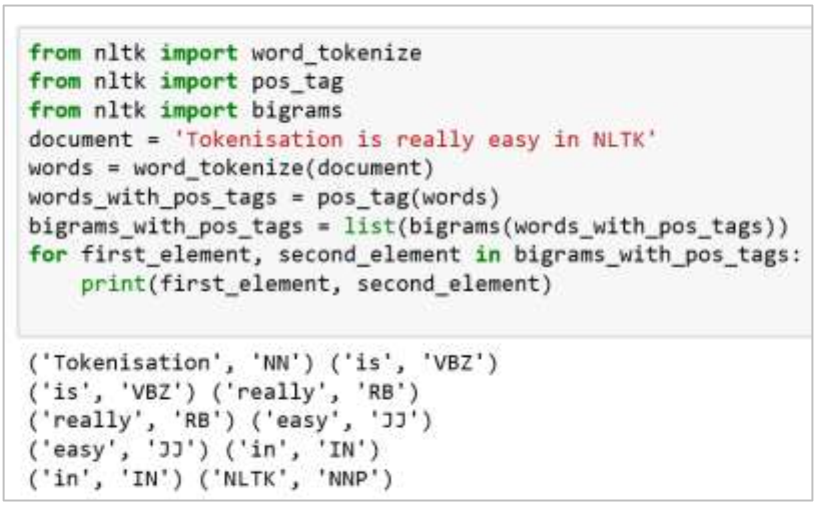
- First import the pos_tag function using the command: ‘from nltk import pos_tag’



```
from nltk import bigrams
```

- The bigrams() function takes as an input a list of tuples, previously extracted using the pos_tag() function, and returns a Python iterable object which can be easily converted into a list of pairs/tuples using the list() function.
- Each pair in the list consist of two items:
    - the first item is a <word, pos tag> tuple which corresponds to the first word of the bigram
    - the second item is another <word, pos tag> tuple which corresponds to the second word of the bigram

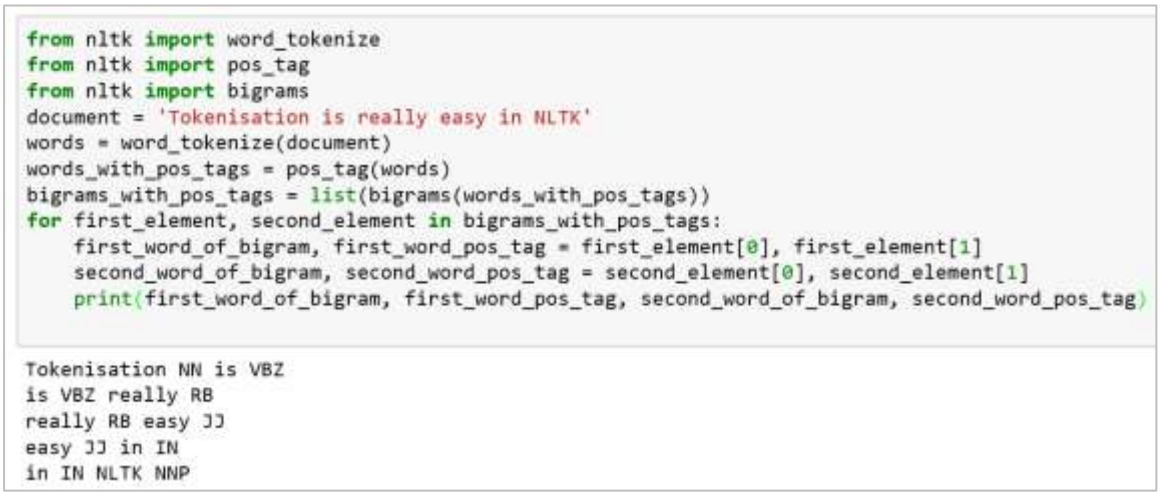
Printing bigram elements



```
from nltk import word_tokenize
from nltk import pos_tag
from nltk import bigrams
document = 'Tokenisation is really easy in NLTK'
words = word_tokenize(document)
words_with_pos_tags = pos_tag(words)
bigrams_with_pos_tags = list(bigrams(words_with_pos_tags))
for first_element, second_element in bigrams_with_pos_tags:
    print(first_element, second_element)
```

```
('Tokenisation', 'NN') ('is', 'VBZ')
('is', 'VBZ') ('really', 'RB')
('really', 'RB') ('easy', 'JJ')
('easy', 'JJ') ('in', 'IN')
('in', 'IN') ('NLTK', 'NNP')
```

Printing parts of bigram elements

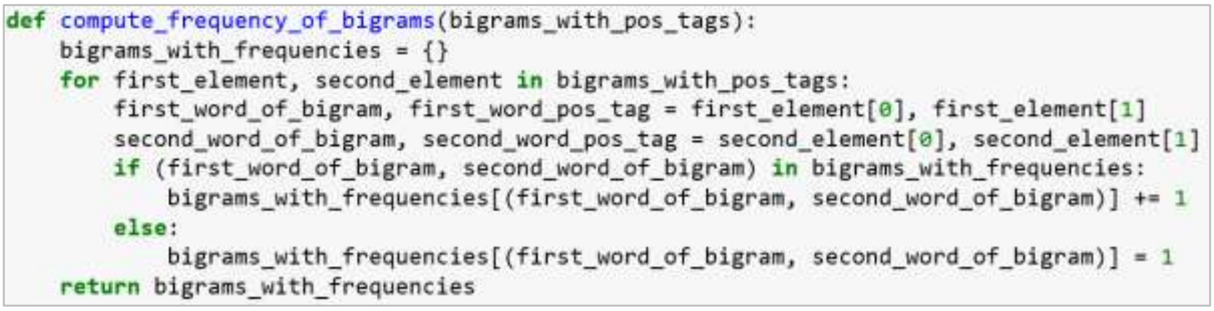


```
from nltk import word_tokenize
from nltk import pos_tag
from nltk import bigrams
document = 'Tokenisation is really easy in NLTK'
words = word_tokenize(document)
words_with_pos_tags = pos_tag(words)
bigrams_with_pos_tags = list(bigrams(words_with_pos_tags))
for first_element, second_element in bigrams_with_pos_tags:
    first_word_of_bigram, first_word_pos_tag = first_element[0], first_element[1]
    second_word_of_bigram, second_word_pos_tag = second_element[0], second_element[1]
    print(first_word_of_bigram, first_word_pos_tag, second_word_of_bigram, second_word_pos_tag)
```

```
Tokenisation NN is VBZ
is VBZ really RB
really RB easy JJ
easy JJ in IN
in IN NLTK NNP
```

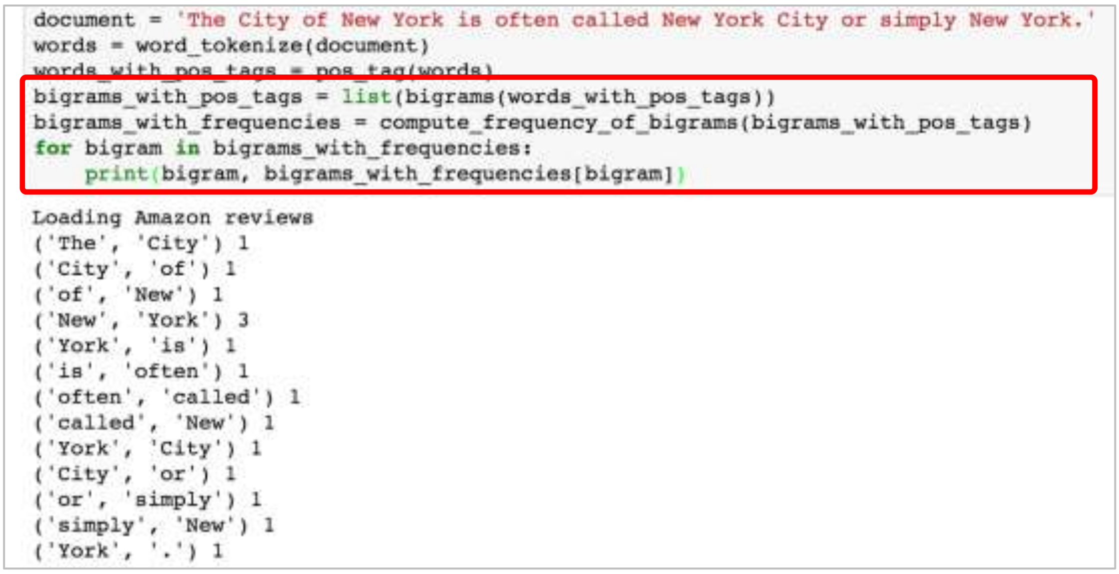
3.2 Computing the Frequency of Bigrams

- ❑ To compute the frequency of bigrams we will need to write our own custom function. Our function should take as an input a list of bigrams with POS tags, previously extracted using the function list(bigrams()) and it should return a dictionary with the following keys/values:
    1. Key of dictionary: Tuple: <first word of bigram, second word of bigram>
    2. Value of dictionary: Frequency of the bigram



```
def compute_frequency_of_bigrams(bigrams_with_pos_tags):
    bigrams_with_frequencies = {}
    for first_element, second_element in bigrams_with_pos_tags:
        first_word_of_bigram, first_word_pos_tag = first_element[0], first_element[1]
        second_word_of_bigram, second_word_pos_tag = second_element[0], second_element[1]
        if (first_word_of_bigram, second_word_of_bigram) in bigrams_with_frequencies:
            bigrams_with_frequencies[(first_word_of_bigram, second_word_of_bigram)] += 1
        else:
            bigrams_with_frequencies[(first_word_of_bigram, second_word_of_bigram)] = 1
    return bigrams_with_frequencies
```

- Finally, we simply need to call the compute_frequency_of_bigrams() function

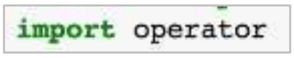


```
document = 'The City of New York is often called New York City or simply New York.'
words = word_tokenize(document)
words_with_pos_tags = pos_tag(words)
bigrams_with_pos_tags = list(bigrams(words_with_pos_tags))
bigrams_with_frequencies = compute_frequency_of_bigrams(bigrams_with_pos_tags)
for bigram in bigrams_with_frequencies:
    print(bigram, bigrams_with_frequencies[bigram])
```

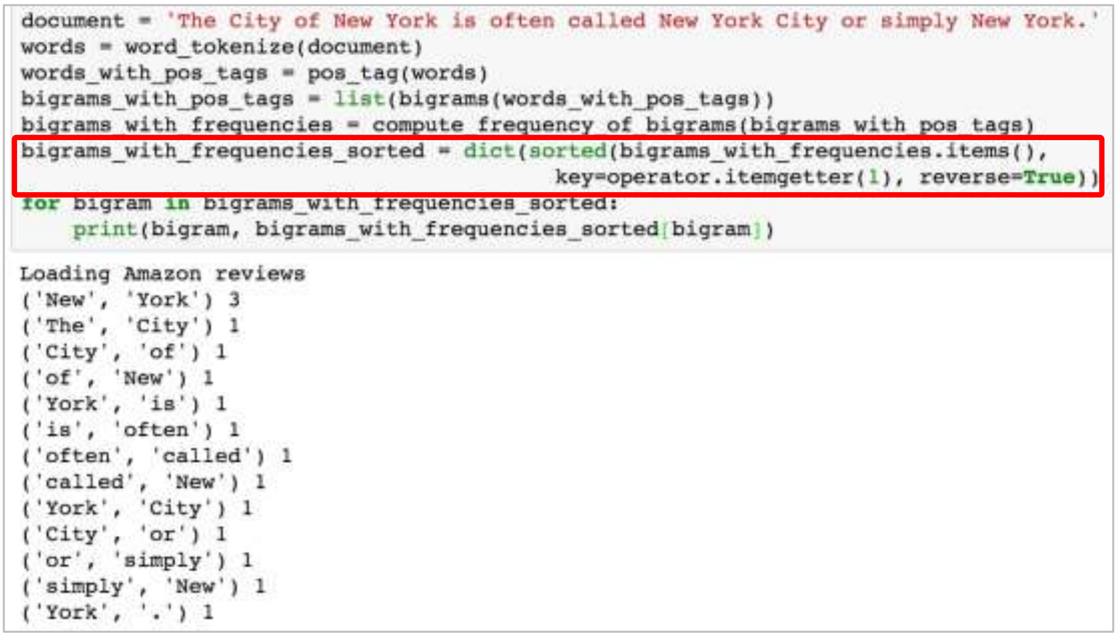
```
Loading Amazon reviews
('The', 'City') 1
('City', 'of') 1
('of', 'New') 1
('New', 'York') 3
('York', 'is') 1
('is', 'often') 1
('often', 'called') 1
('called', 'New') 1
('York', 'City') 1
('City', 'or') 1
('or', 'simply') 1
('simply', 'New') 1
('York', '.') 1
```

3.3 Sorting bigrams by frequency

- The sorted() function of the Python's operator library can be used to sort a dictionary by values.



```
import operator
```



```
document = 'The City of New York is often called New York City or simply New York.'
words = word_tokenize(document)
words_with_pos_tags = pos_tag(words)
bigrams_with_pos_tags = list(bigrams(words_with_pos_tags))
bigrams_with_frequencies = compute_frequency_of_bigrams(bigrams_with_pos_tags)
bigrams_with_frequencies_sorted = dict(sorted(bigrams_with_frequencies.items(),
                                        key=operator.itemgetter(1), reverse=True))
for bigram in bigrams_with_frequencies_sorted:
    print(bigram, bigrams_with_frequencies_sorted[bigram])
```

```
Loading Amazon reviews
('New', 'York') 3
('The', 'City') 1
('City', 'of') 1
('of', 'New') 1
('York', 'is') 1
('is', 'often') 1
('often', 'called') 1
('called', 'New') 1
('York', 'City') 1
('City', 'or') 1
('or', 'simply') 1
('simply', 'New') 1
('York', '.') 1
```

Activity 4: Advanced (not compulsory)

Exercise:

Given the sentence:

"The City of New York is often called New York City or simply New York."

- filter the bigram list by keeping only those bigrams whose POS tag matches one of the following:
  - "JJ NN"
  - "NN NN"
  - "NNP NNP"

    - “NNP NN”

Hint: In the compute_frequency_of_bigrams() function add an if statement to filter bigrams

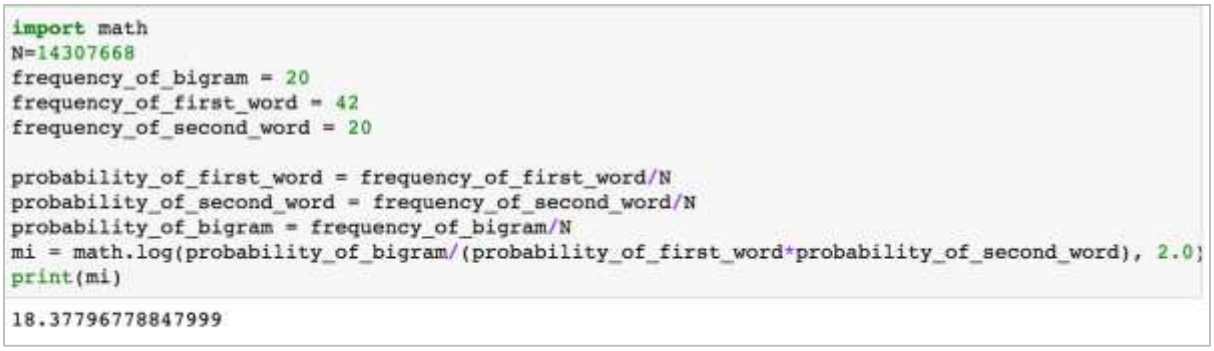
4.1 Computing Mutual information

- To compute the mutual information, you first need to import Python’s math library

```
import math
```

$$pmi(x;y) = log_2 \frac{P(x,y)}{P(x)P(y)}$$

- Then you need to compute the following:
  1. N=number of unique words
  2. Probability of first word = frequency of first word/N
  3. Probability of second word = frequency of second word/N
  4. Probability of bigram = frequency of bigram/N



```
import math
N=14307668
frequency_of_bigram = 20
frequency_of_first_word = 42
frequency_of_second_word = 20

probability_of_first_word = frequency_of_first_word/N
probability_of_second_word = frequency_of_second_word/N
probability_of_bigram = frequency_of_bigram/N
mi = math.log(probability_of_bigram/(probability_of_first_word*probability_of_second_word), 2.0)
print(mi)
```

```
18.37796778847999
```