BASIC CONCEPTS IN TEXT MINING

Text Mining

- Text Mining-文件探勘
- 文字探勘（Text Mining）被視為是資料探勘（Data Mining）的一環，其中有個關鍵的差別，在於傳統資料探勘所處理的資料，都是「結構性」的資料，也就是說，資料本身具有明確的結構，
- 例如，像是一個固定結構的表格，每個欄位有其明確的定義及值。而資料探勘技術中的演算法，則是以這些結構性的資料為輸入，經過演算過程之後計算得到結果。
- 但文字探勘不同於資料探勘的地方，則在於它的原始輸入資料，都是沒有特定結構的純文字，這些文字的內容，都是用人類的自然語言所寫成的，所以，無法直接套用資料探勘的演算法，來計算出些什麼有意義的東西。

Source:王建興

- 在我們生活當中，除了具結構性的資料，也有相當大量的文字資料，**不具結構性的資料**像是每天的新聞、人們在 Facebook、Twitter、微博上所發表的近況更新、部落格文章、專利文件等等。
- 這些自然語言文字型的資料中，同樣蘊藏可觀、極具潛力的「礦產」，也就是有價值的資訊，等著我們用資訊技術去開採。這就是文字探勘技術及應用所希望達成的目標。
- 另一方面，大約在數十幾年前，搜尋引擎的技術開始普及了起來，除了有一些商業公司開始釋出搜尋引擎的實作，授權其他公司使用之外，也有開放原始碼的專案持續的開發並釋出，其中最有名的，莫過於 Apache Project 的Lucene了。
- 隨著開放原始碼的搜尋引擎專案的壯大，除了讓開發社群愈來愈輕易就能開發運用搜尋引擎能力的應用系統之外，也讓世人更容易得窺搜尋引擎設計之堂奧。同時，在開放原始碼的社群裡，許多以搜尋引擎為核心的程式庫及系統，也得以建立在Lucene的基礎上，陸續問世。像是基於Lucene、提供分散式索引搜尋能力的Solr，還有提供crawling——即俗稱「爬網頁」系統的Nutch、……等等。這些在Apache 之下的開放原始碼專案，以Lucene為核心，形成了一個生態系，也為想要運用搜尋引擎機制的開發者提供了各種便利的工具。

文字表示成數字

- 在正式介紹 Text Mining 的演算法前，我們先來談談如何把**文字表示成數字**
- 為什麼要用數字表示呢？因為電腦只能理解數字，不論在後續的演算法處理，亦或是近期火紅的深度學習，都必須使用數字來表示文字，讓電腦讀懂意涵，畢竟電腦還是看不懂文字的！
- Example: 這裡用一個最簡單的例子來理解，如何把文字編碼成數字：
  - 以下有三個句子 (已經先做好斷詞處理)

```
1. 我 愛 吃 香蕉
2. 我 愛 吃 蘋果
3. 香蕉 好 吃
```

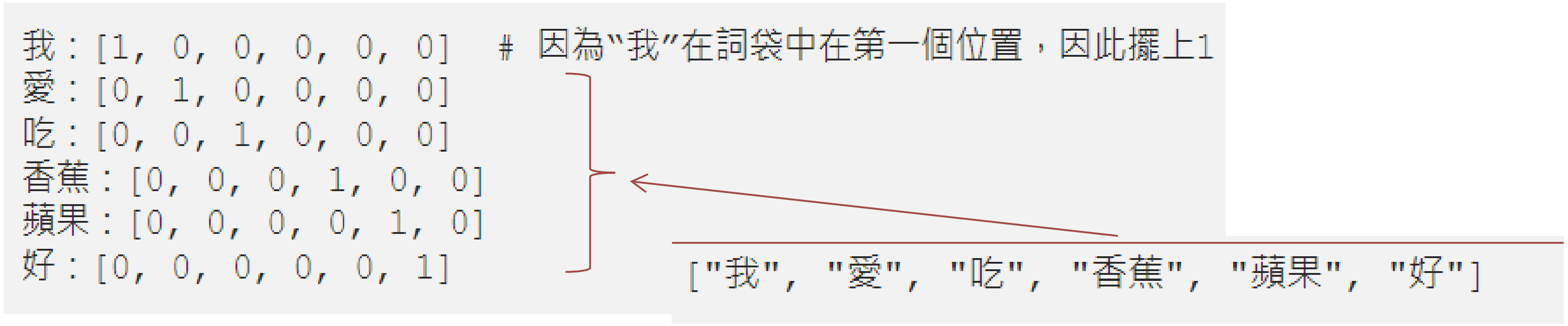
  - 我們抓出上面兩個句子中**不重複的詞**，就像把詞都丟到袋子裡一樣**Bag-of-Word (詞袋模型編碼)** ，我們就會得到有五個詞在我們的袋子裡 (這裡用Python的list表示)

```
["我", "愛", "吃", "香蕉", "蘋果", "好"]
```

  - 因為所有的文檔裡面，總共只有六個單詞，因此每個單詞就可以用**長度為6的一維向量(vector)**去表示

Source:蘇靖軒

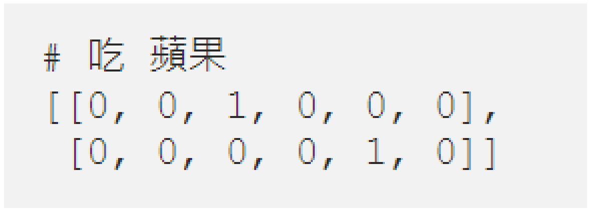
- Example
  - 因為所有的文檔裡面，總共只有六個單詞，因此每個單詞就可以用**長度為6的一維向量(vector)**去表示. 以下是每個文字的向量表示，也某種程度的把文字轉換成數字



```
我：[1, 0, 0, 0, 0, 0]   # 因為"我"在詞袋中在第一個位置，因此擺上1
愛：[0, 1, 0, 0, 0, 0]
吃：[0, 0, 1, 0, 0, 0]
香蕉：[0, 0, 0, 1, 0, 0]
蘋果：[0, 0, 0, 0, 1, 0]
好：[0, 0, 0, 0, 0, 1]
```

```
["我", "愛", "吃", "香蕉", "蘋果", "好"]
```

  - 假設現在有一個句子是 "吃蘋果" ，經過斷詞後，這個句子就可以被編碼成



```
# 吃 蘋果
[[0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 0]]
```

  - 以上的文字編碼方式稱為 **One-Hot Encoding (獨熱編碼)**，是很簡單將文字轉換成數字的方式

- Example
  - 這時，你發現到了，這只是給他一個編號而已，純粹代表這個詞有在文檔中出現過.也許我們嘗試著找一些**特徵**，看看怎麼來編碼
  - 這次我們採用 **詞袋模型 (Bag-of-Word) 編碼.**首先拿出剛剛的詞袋，並統計一下每個詞在所有句子中出現的次數，也就是計算詞頻(word frequecy, wf)，然後把詞依照詞頻大小去排序，並給他一個索引(index)
  - 計算詞頻

```
{"我": 2, "愛": 2, "吃": 3, "香蕉": 2, "蘋果": 1, "好": 1}
```

```
1. 我 愛 吃 香蕉
2. 我 愛 吃 蘋果
3. 香蕉 好 吃
```

  - 詞依照詞頻**大小**去**排序**，並給他一個索引(index)，也就是給他一個流水號

```
{"吃": 0, "我": 1, "愛": 2, "香蕉": 3, "蘋果": 4, "好": 5}
```

  - 其中這個索引(index)，就表示向量陣列中的第index位置，並在這個index填上1，表示這個詞在句子中**有出現.**以”蘋果”這個詞為例，索引為4，因此在陣列中(向量長度與詞袋長度相同) **index為4的地方填上1來表示這個詞**

```
# 蘋果
[0, 0, 0, 0, 1, 0]
```

  - **“蘋果好吃，我愛吃”** 這句來說，句子會被編碼成

```
''' 解析一下句子編碼怎麼運作的
輪次   詞       句子向量
0              [0, 0, 0 ,0, 0, 0] -> 準備好跟詞袋長度一樣的向量list，並都填上0
1     蘋果      [0, 0, 0, 0, 1, 0] -> "蘋果"的index=4
2     好        [0, 0, 0, 0, 1, 1]
3     吃        [1, 0, 0, 0, 1, 1]
4     我        [1, 1, 0, 0, 1, 1]
5     愛        [1, 1, 1, 0, 1, 1]
6     吃        [2, 1, 1, 0, 1, 1]
'''
```

累計**summarized**

Text Mining in Python: Basic Concepts

Source: Dhilip Subramanian

相關社群軟體

- 根據估計，在當今世界，只有 20% 的數據是以結構化格式生成的
- 而在發推文、在 WhatsApp、電子郵件、Facebook、Instagram 或任何文本消息上發送消息時會以非結構化方式生成的。而且這些數據大部分以文本形式存在，這是一種高度非結構化的格式。為了從文本數據中產生有意義的見解，我們需要遵循一種稱為文本分析的方法。

What is Text Mining?

- Text Mining 是從自然語言文本中,獲取有意義資訊的過程。

Text Mining is the process of deriving high quality information from the text.

The overall goal is to turn the texts into data for analysis, via application of Natural Language Processing (NLP)

What is NLP?

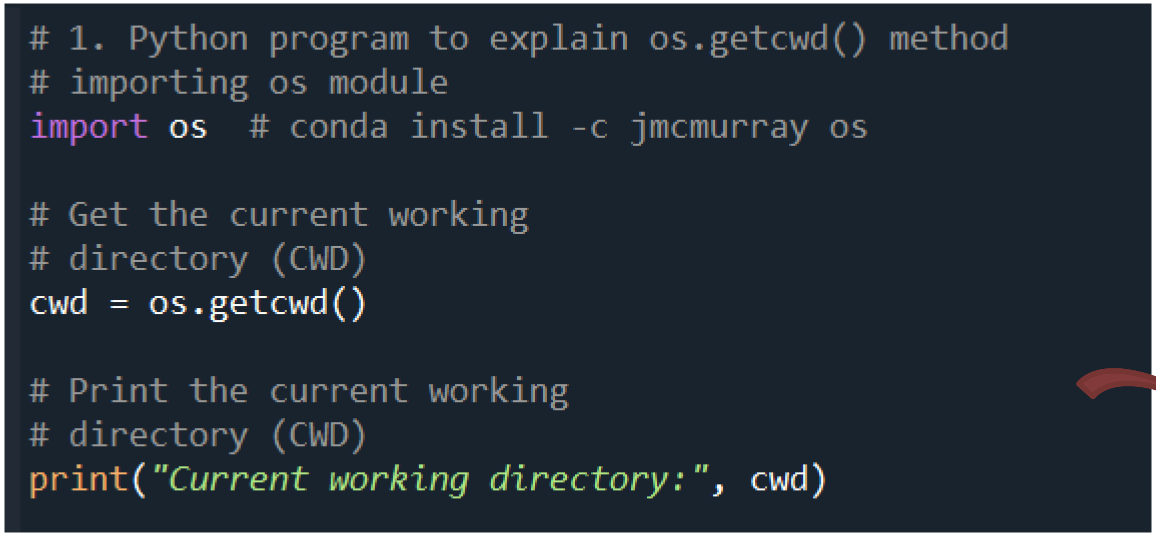
Source: Dhilip Subramanian

- 換句話說，NLP (Natural Language Processing)是本文挖掘的一個組成部分，它執行一種特殊的語言分析，從本質上幫助機器“閱讀”文本。
- 它使用不同的方法來解譯人類語言中的歧義，包括：自動摘要、詞性標記、消歧、分塊以及消歧，以及自然語言理解和識別。底下將使用 Python 逐步查看所有過程。
- 首先，
  - 我們需要安裝 NLTK lib，它是用於構建 Python 程序,以處理人類語言數據的自然語言工具包，它還提供易於使用的界面。
  - **nltk.corpus package**:   sample text for performing tokenization
  - **os package**: The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc. You first need to import the os module to interact with the underlying operating system.

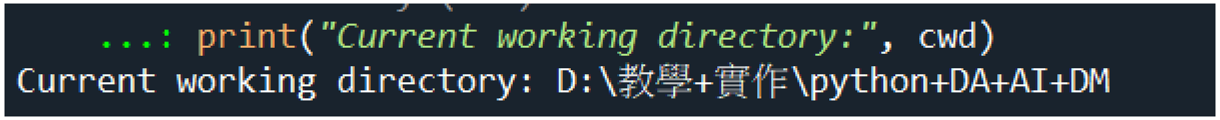
OS Module in Python with Examples

- The OS module in Python provides functions for interacting with the operating system. OS comes under Python’s standard utility modules. This module provides a portable way of using operating system-dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.

Getting the Current working directory



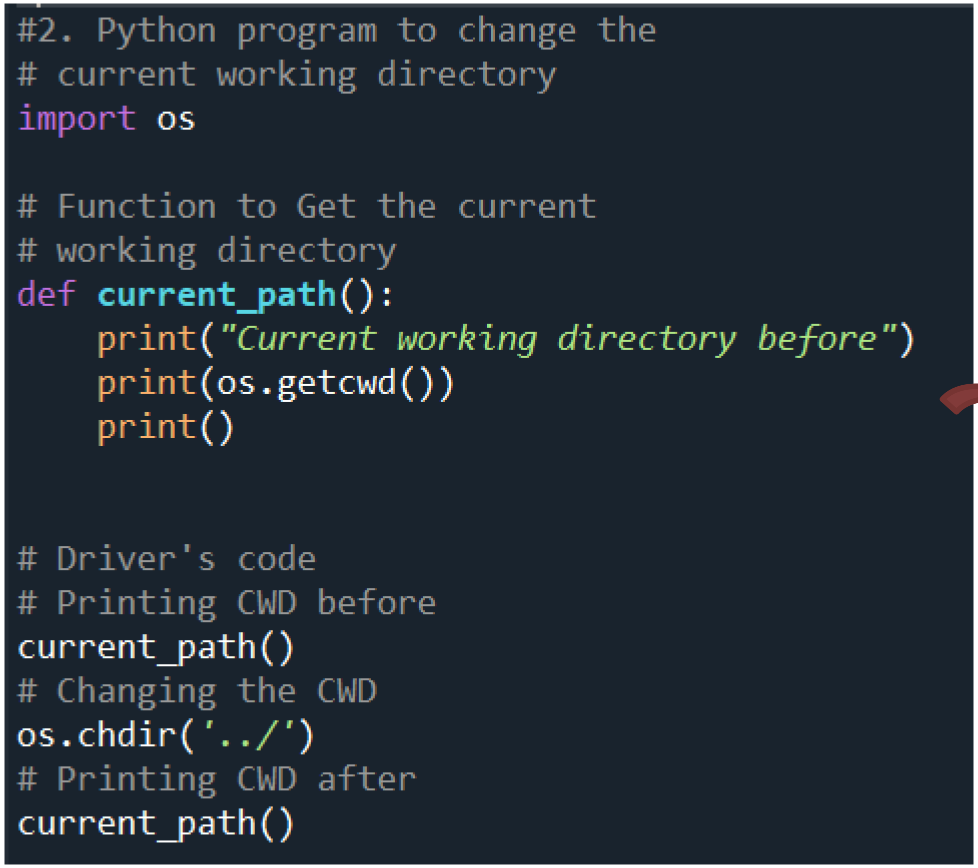
```python
# 1. Python program to explain os.getcwd() method
# importing os module
import os  # conda install -c jmcmurray os

# Get the current working
# directory (CWD)
cwd = os.getcwd()

# Print the current working
# directory (CWD)
print("Current working directory:", cwd)
```



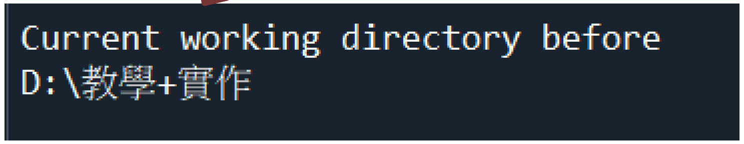
```
    ...: print("Current working directory:", cwd)
Current working directory: D:\教學+實作\python+DA+AI+DM
```

DA_ex1803a.py

Changing the Current working directory



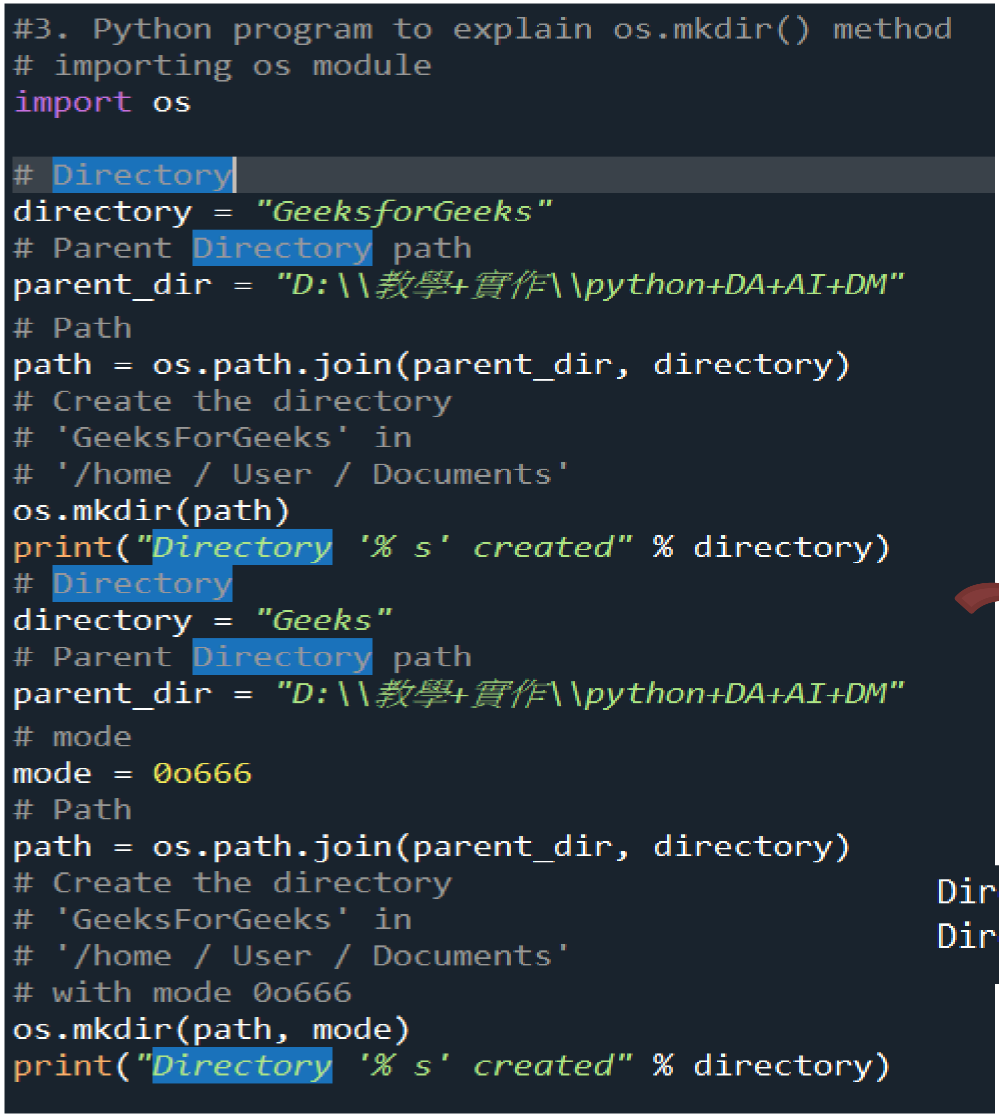
```python
#2. Python program to change the
# current working directory
import os

# Function to Get the current
# working directory
def current_path():
    print("Current working directory before")
    print(os.getcwd())
    print()


# Driver's code
# Printing CWD before
current_path()
# Changing the CWD
os.chdir('../')
# Printing CWD after
current_path()
```



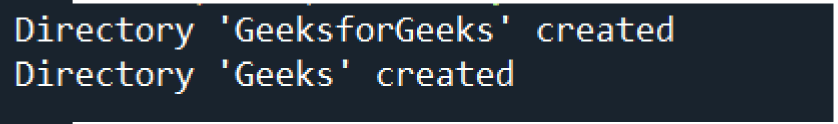
```
Current working directory before
D:\教學+實作
```

DA_ex1803a.py

Creating a Directory



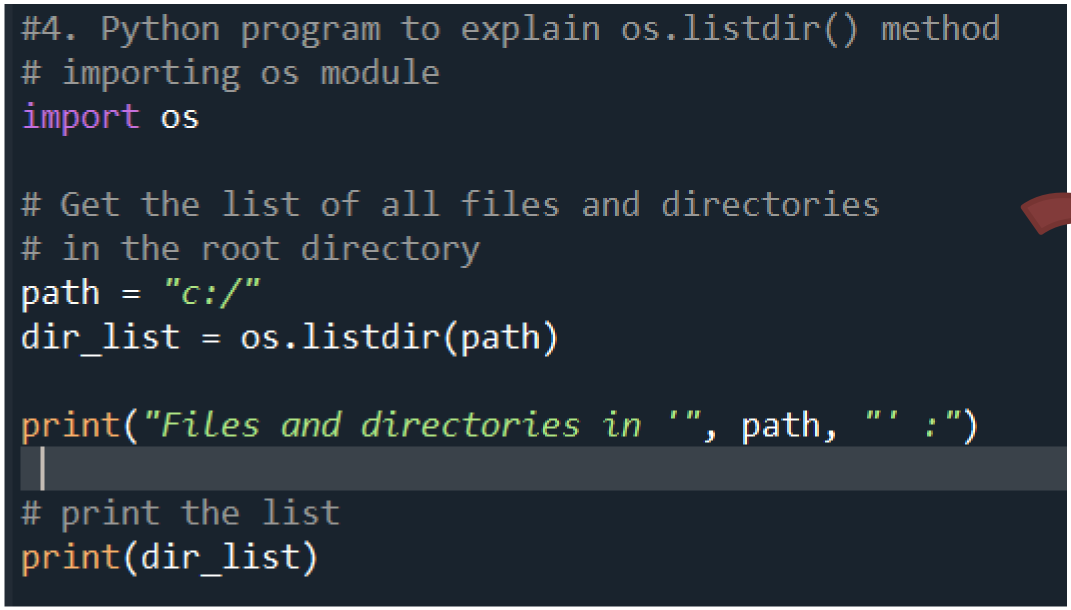
```python
#3. Python program to explain os.mkdir() method
# importing os module
import os

# Directory
directory = "GeeksforGeeks"
# Parent Directory path
parent_dir = "D:\\教學+實作\\python+DA+AI+DM"
# Path
path = os.path.join(parent_dir, directory)
# Create the directory
# 'GeeksForGeeks' in
# '/home / User / Documents'
os.mkdir(path)
print("Directory '% s' created" % directory)
# Directory
directory = "Geeks"
# Parent Directory path
parent_dir = "D:\\教學+實作\\python+DA+AI+DM"
# mode
mode = 0o666
# Path
path = os.path.join(parent_dir, directory)
# Create the directory
# 'GeeksForGeeks' in
# '/home / User / Documents'
# with mode 0o666
os.mkdir(path, mode)
print("Directory '% s' created" % directory)
```



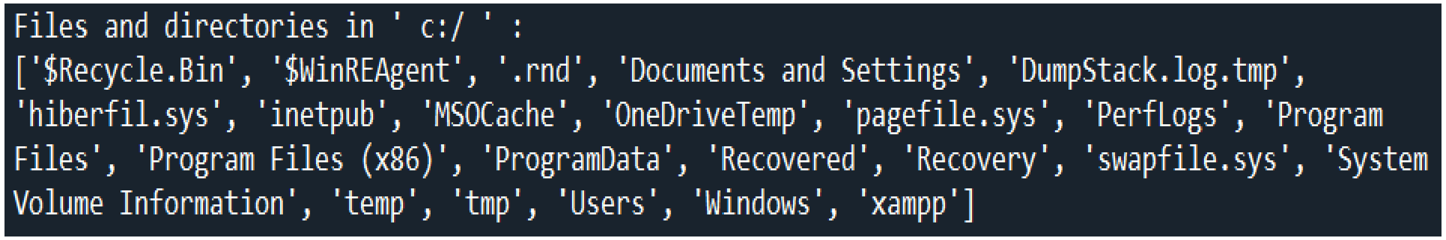
```
Directory 'GeeksforGeeks' created
Directory 'Geeks' created
```

DA_ex1803a.py

Listing out Files and Directories



```
#4. Python program to explain os.listdir() method
# importing os module
import os

# Get the list of all files and directories
# in the root directory
path = "c:/"
dir_list = os.listdir(path)

print("Files and directories in '", path, "' :")

# print the list
print(dir_list)
```



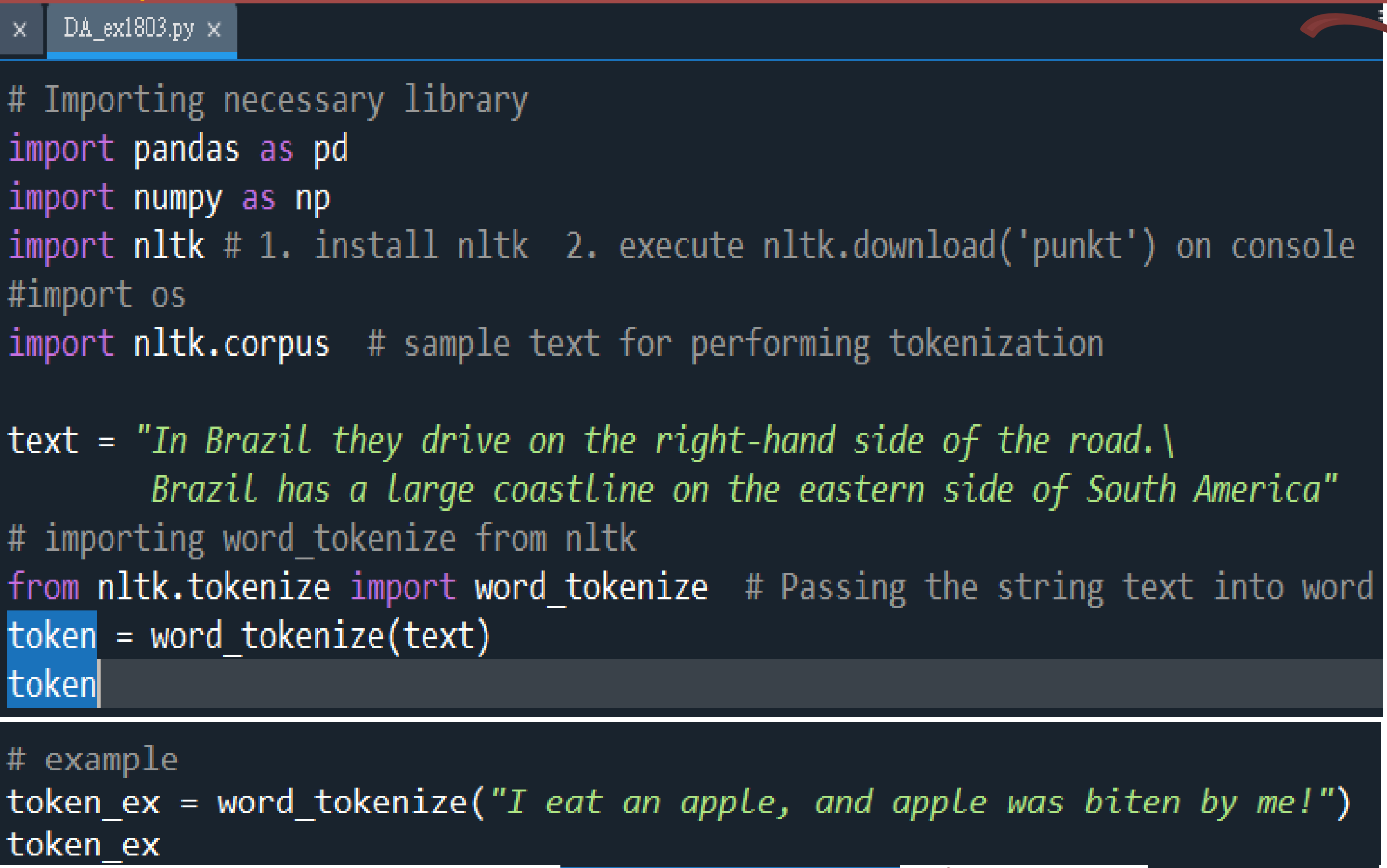
```
Files and directories in ' c:/ ' :
['$Recycle.Bin', '$WinREAgent', '.rnd', 'Documents and Settings', 'DumpStack.log.tmp',
'hiberfil.sys', 'inetpub', 'MSOCache', 'OneDriveTemp', 'pagefile.sys', 'PerfLogs', 'Program
Files', 'Program Files (x86)', 'ProgramData', 'Recovered', 'Recovery', 'swapfile.sys', 'System
Volume Information', 'temp', 'tmp', 'Users', 'Windows', 'xampp']
```

DA_ex1803a.py

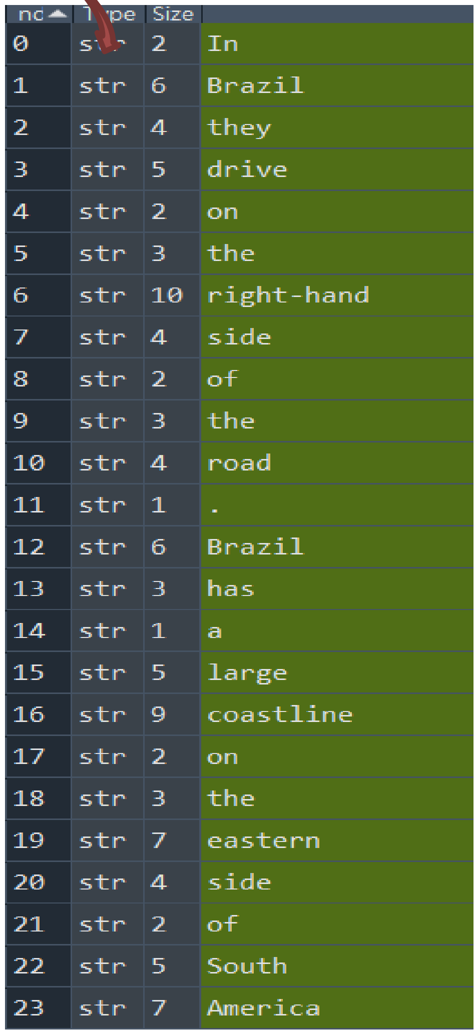
NPL - Tokenization

- 標記化(Tokenization)是 NPL 的第一步
  - 這是將字符串**分解**為標記的過程，標記又是小結構或單元。
  - 標記化涉及三個步驟，將復雜的句子分解為單詞，了解每個單詞相對於句子的重要性，最後對輸入句子產生結構描述。

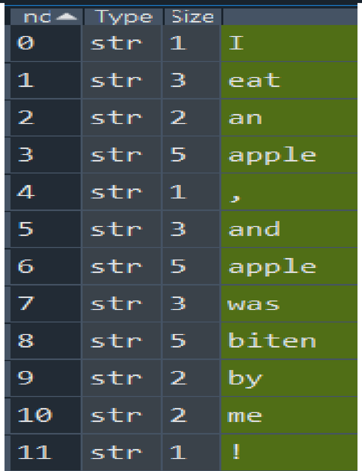
Step 1. Tokenization



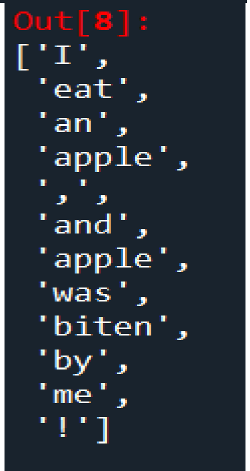
```
x   DA_ex1803.py x

# Importing necessary library
import pandas as pd
import numpy as np
import nltk # 1. install nltk  2. execute nltk.download('punkt') on console
#import os
import nltk.corpus  # sample text for performing tokenization

text = "In Brazil they drive on the right-hand side of the road.\
        Brazil has a large coastline on the eastern side of South America"
# importing word_tokenize from nltk
from nltk.tokenize import word_tokenize  # Passing the string text into word
token = word_tokenize(text)
token

# example
token_ex = word_tokenize("I eat an apple, and apple was biten by me!")
token_ex
```



nc	Type	Size	
0	str	2	In
1	str	6	Brazil
2	str	4	they
3	str	5	drive
4	str	2	on
5	str	3	the
6	str	10	right-hand
7	str	4	side
8	str	2	of
9	str	3	the
10	str	4	road
11	str	1	.
12	str	6	Brazil
13	str	3	has
14	str	1	a
15	str	5	large
16	str	9	coastline
17	str	2	on
18	str	3	the
19	str	7	eastern
20	str	4	side
21	str	2	of
22	str	5	South
23	str	7	America



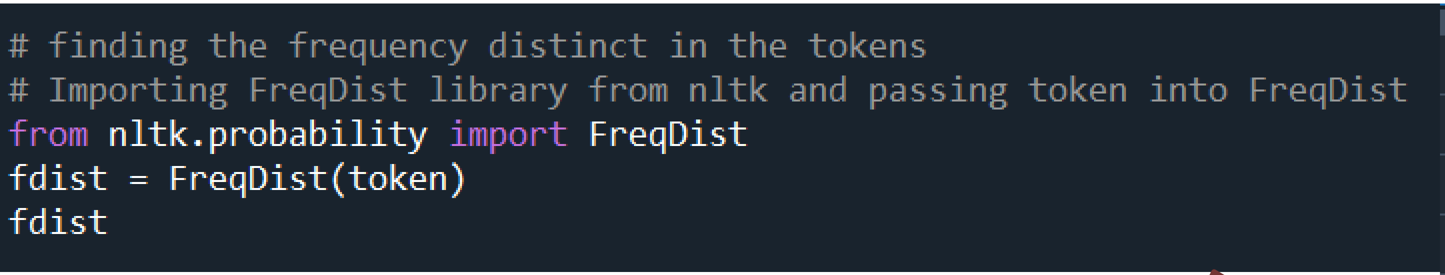
nc	Type	Size	
0	str	1	I
1	str	3	eat
2	str	2	an
3	str	5	apple
4	str	1	,
5	str	3	and
6	str	5	apple
7	str	3	was
8	str	5	biten
9	str	2	by
10	str	2	me
11	str	1	!



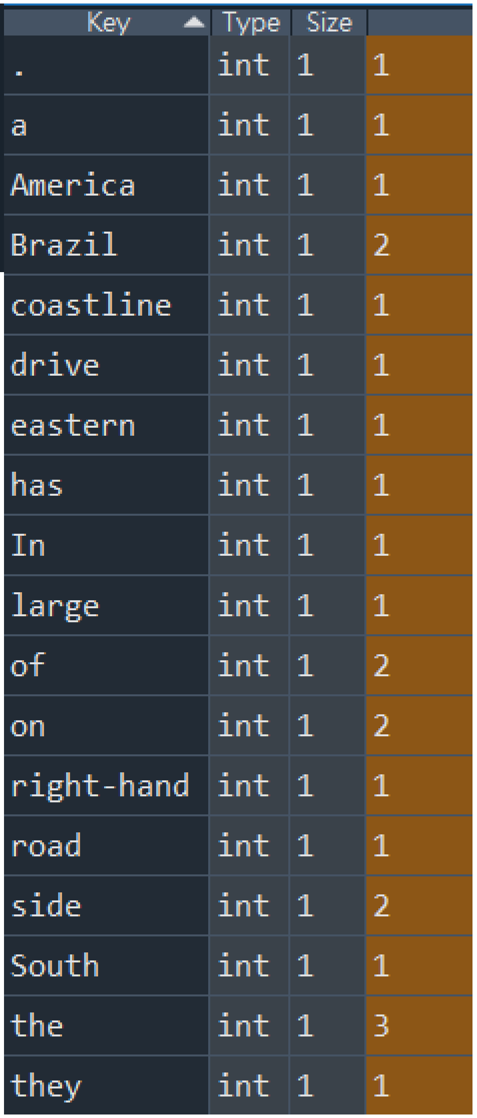
```
Out[8]:
['I',
 'eat',
 'an',
 'apple',
 ',',
 'and',
 'apple',
 'was',
 'biten',
 'by',
 'me',
 '!']
```

DA_ex1803.py

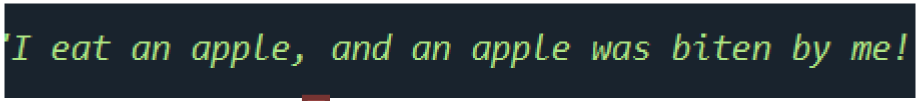
Step 2. Finding frequency distinct in the text



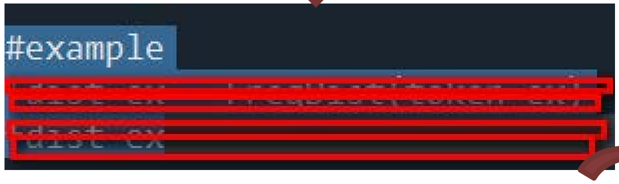
```python
# finding the frequency distinct in the tokens
# Importing FreqDist library from nltk and passing token into FreqDist
from nltk.probability import FreqDist
fdist = FreqDist(token)
fdist
```



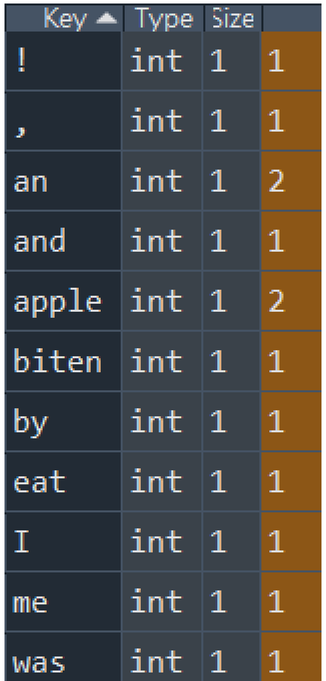
Key	Type	Size	
.	int	1	1
a	int	1	1
America	int	1	1
Brazil	int	1	2
coastline	int	1	1
drive	int	1	1
eastern	int	1	1
has	int	1	1
In	int	1	1
large	int	1	1
of	int	1	2
on	int	1	2
right-hand	int	1	1
road	int	1	1
side	int	1	2
South	int	1	1
the	int	1	3
they	int	1	1



```
'I eat an apple, and an apple was biten by me!
```

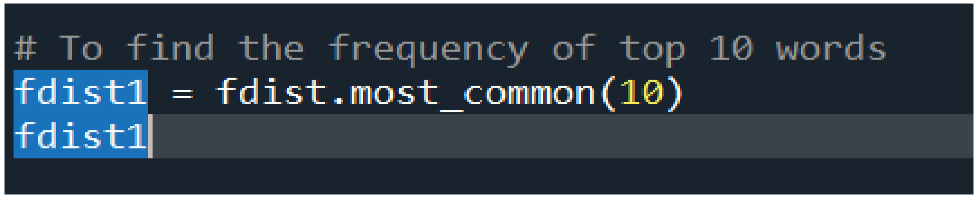


```
#example
~~fdist_ex = FreqDist(token_ex)~~
~~fdist_ex~~
```

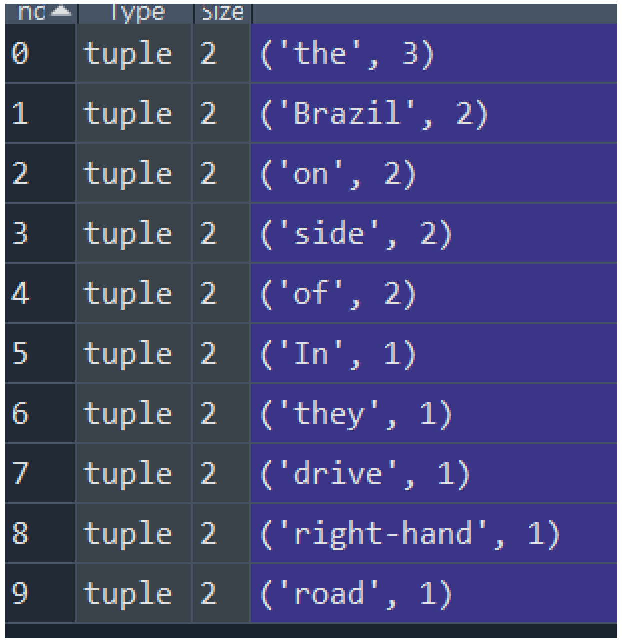


Key	Type	Size	
!	int	1	1
,	int	1	1
an	int	1	2
and	int	1	1
apple	int	1	2
biten	int	1	1
by	int	1	1
eat	int	1	1
I	int	1	1
me	int	1	1
was	int	1	1

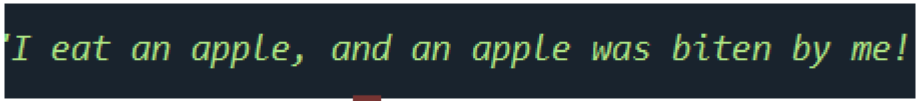
DA_ex1803.py



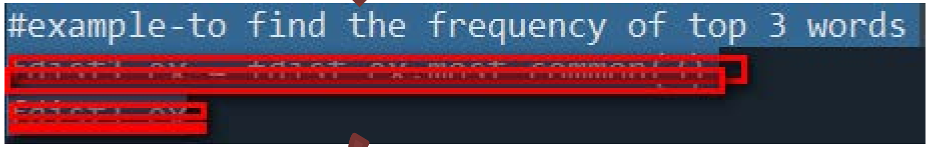
```
# To find the frequency of top 10 words
fdist1 = fdist.most_common(10)
fdist1
```



nc	Type	Size	
0	tuple	2	('the', 3)
1	tuple	2	('Brazil', 2)
2	tuple	2	('on', 2)
3	tuple	2	('side', 2)
4	tuple	2	('of', 2)
5	tuple	2	('In', 1)
6	tuple	2	('they', 1)
7	tuple	2	('drive', 1)
8	tuple	2	('right-hand', 1)
9	tuple	2	('road', 1)

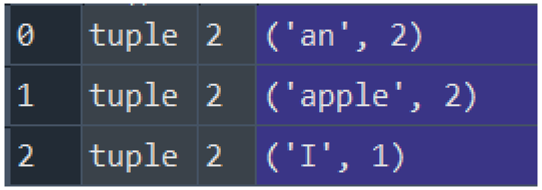


```
'I eat an apple, and an apple was biten by me!
```



```
#example-to find the frequency of top 3 words
```

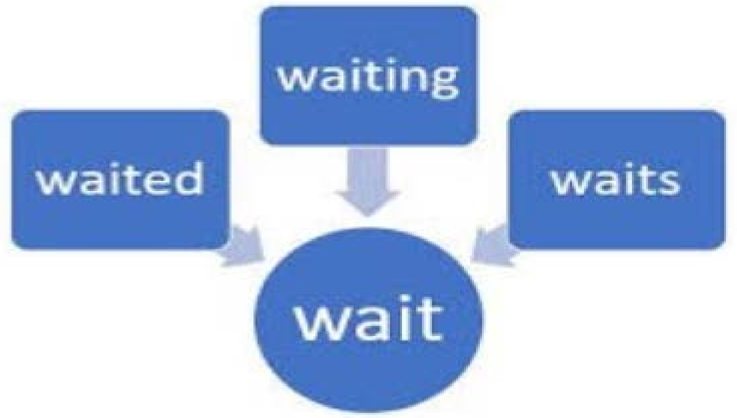
[redacted]



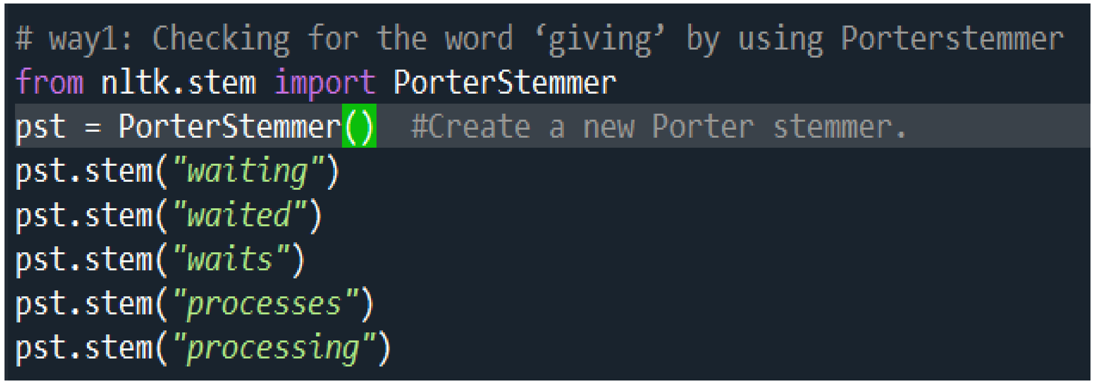
0	tuple	2	('an', 2)
1	tuple	2	('apple', 2)
2	tuple	2	('I', 1)

3. Stemming

- 詞幹提取通常是指將單詞規範化為其基本形式或詞根形式。如

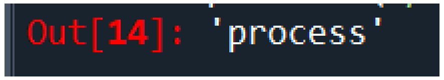


- There are two methods in Stemming, namely, 1)Porter Stemming (removes common morphological (形態的)and inflectional (屈折的) endings from words) and 2) Lancaster Stemming (a more aggressive stemming algrithm).



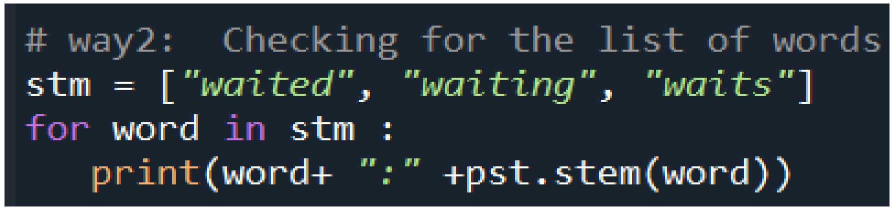
```
# way1: Checking for the word 'giving' by using Porterstemmer
from nltk.stem import PorterStemmer
pst = PorterStemmer()  #Create a new Porter stemmer.
pst.stem("waiting")
pst.stem("waited")
pst.stem("waits")
pst.stem("processes")
pst.stem("processing")
```

Check the last one

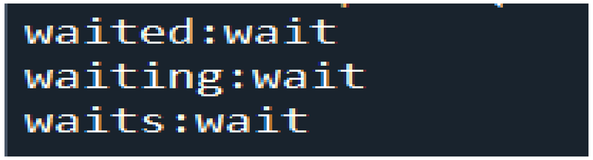


```
Out[14]: 'process'
```

DA_ex1803.py

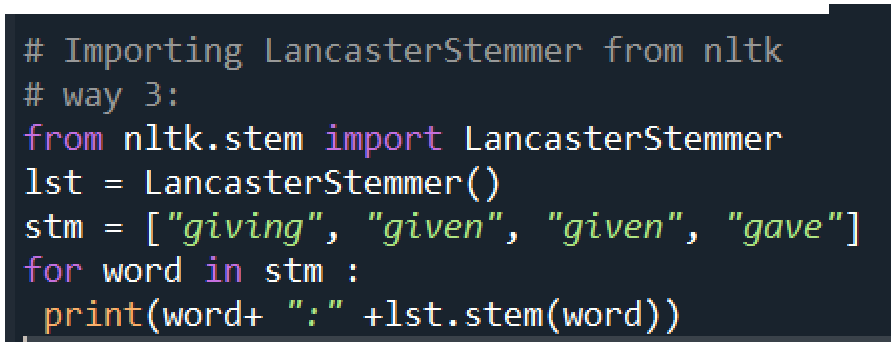


```
# way2:  Checking for the list of words
stm = ["waited", "waiting", "waits"]
for word in stm :
    print(word+ ":" +pst.stem(word))
```

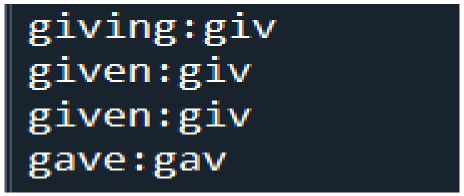


```
waited:wait
waiting:wait
waits:wait
```

Lancaster is more aggressive than Porter stemmer



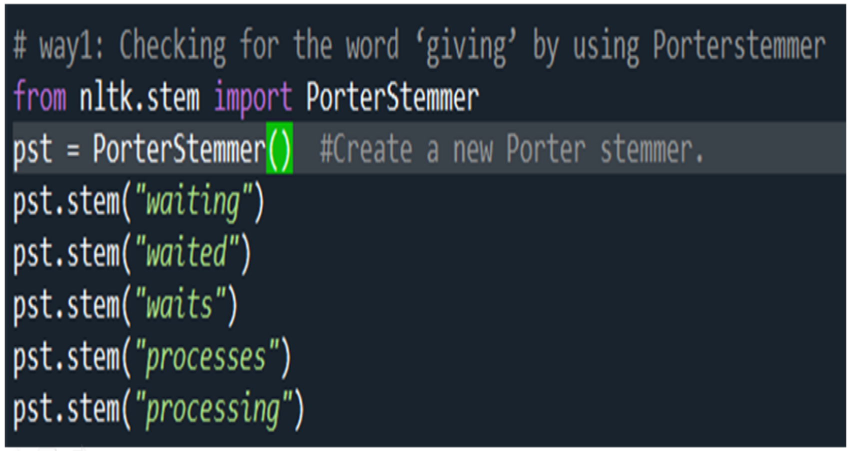
```
# Importing LancasterStemmer from nltk
# way 3:
from nltk.stem import LancasterStemmer
lst = LancasterStemmer()
stm = ["giving", "given", "given", "gave"]
for word in stm :
 print(word+ ":" +lst.stem(word))
```



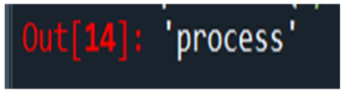
```
giving:giv
given:giv
given:giv
gave:gav
```

如何改進

Check the last one



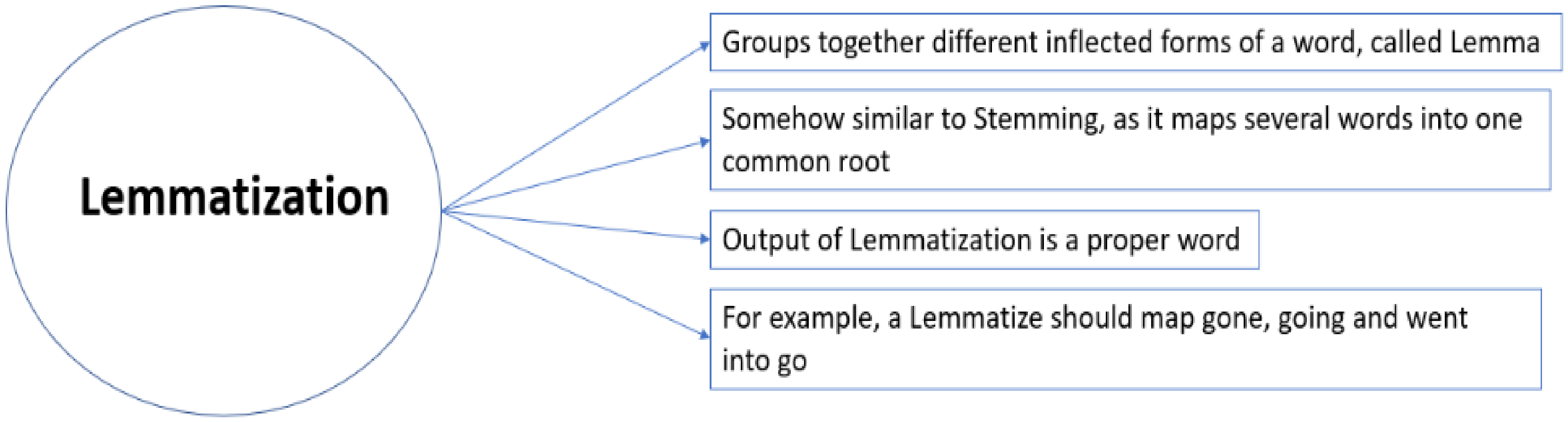
```
# way1: Checking for the word 'giving' by using Porterstemmer
from nltk.stem import PorterStemmer
pst = PorterStemmer()  #Create a new Porter stemmer.
pst.stem("waiting")
pst.stem("waited")
pst.stem("waits")
pst.stem("processes")
pst.stem("processing")
```



```
Out[14]: 'process'
```

DA_ex1803.py

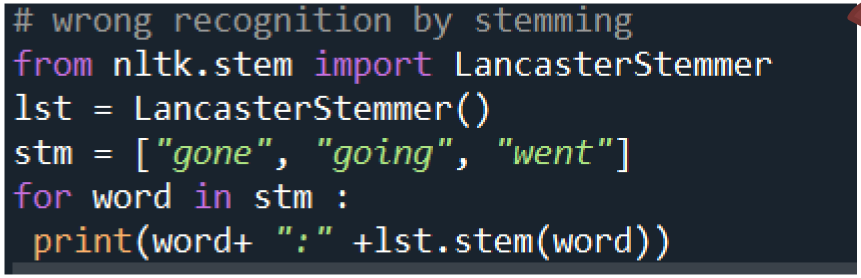
4. Lemmatization


Lemmatization
Groups together different inflected forms of a word, called Lemma
Somehow similar to Stemming, as it maps several words into one common root
Output of Lemmatization is a proper word
For example, a Lemmatize should map gone, going and went into go

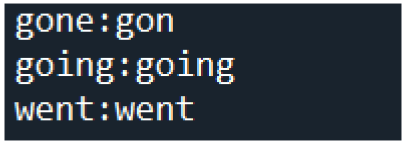
- In simpler terms, it is the process of converting a word to **its base form**. The **difference** between stemming and lemmatization is, lemmatization considers the context and converts the word to its **meaningful base form**, whereas stemming just removes the last few characters, often leading to incorrect meanings and spelling errors.
- For example, lemmatization would correctly identify the base form of 'feet' to 'foot,' whereas stemming would recover it and convert it into a "foot". Lemmatization can be implemented in python by using Wordnet Lemmatizer, Spacy Lemmatizer, TextBlob, Stanford CoreNLP

辨識字形能力

Wrong Recognition by Stemming

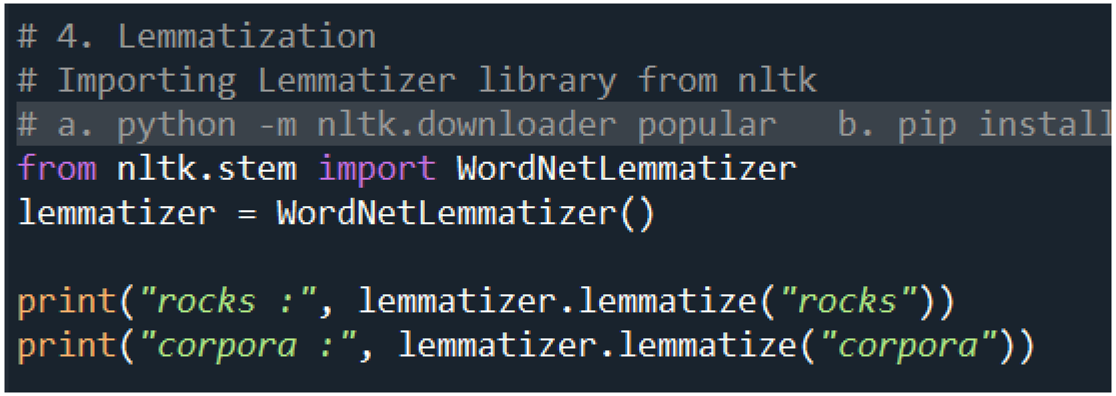


```python
# wrong recognition by stemming
from nltk.stem import LancasterStemmer
lst = LancasterStemmer()
stm = ["gone", "going", "went"]
for word in stm :
 print(word+ ":" +lst.stem(word))
```



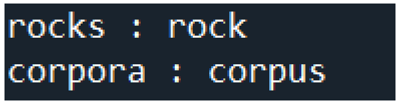
```
gone:gon
going:going
went:went
```

Lemmatization

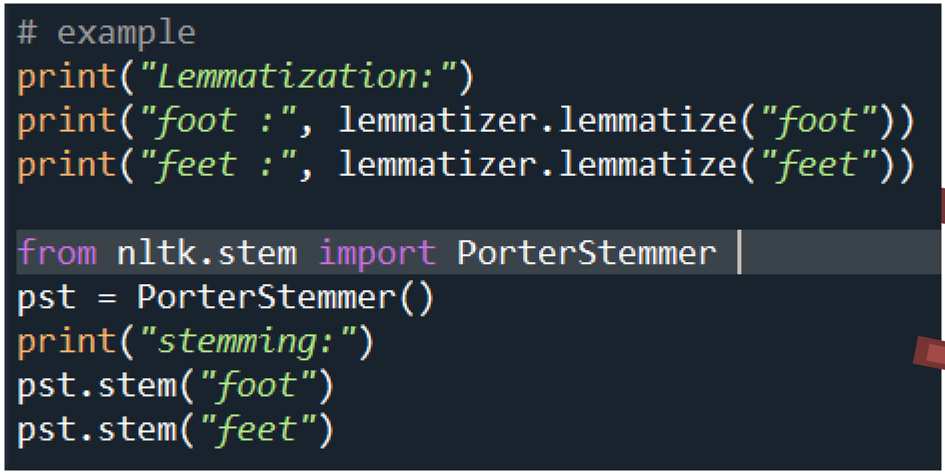


```
# 4. Lemmatization
# Importing Lemmatizer library from nltk
# a. python -m nltk.downloader popular   b. pip install
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

print("rocks :", lemmatizer.lemmatize("rocks"))
print("corpora :", lemmatizer.lemmatize("corpora"))
```

All right



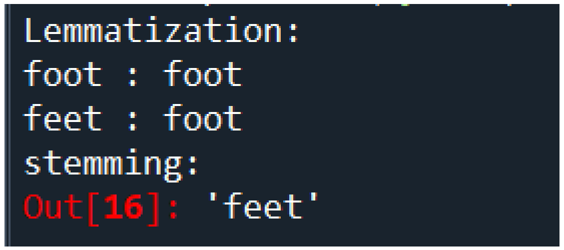
```
rocks : rock
corpora : corpus
```



```
# example
print("Lemmatization:")
print("foot :", lemmatizer.lemmatize("foot"))
print("feet :", lemmatizer.lemmatize("feet"))

from nltk.stem import PorterStemmer
pst = PorterStemmer()
print("stemming:")
pst.stem("foot")
pst.stem("feet")
```

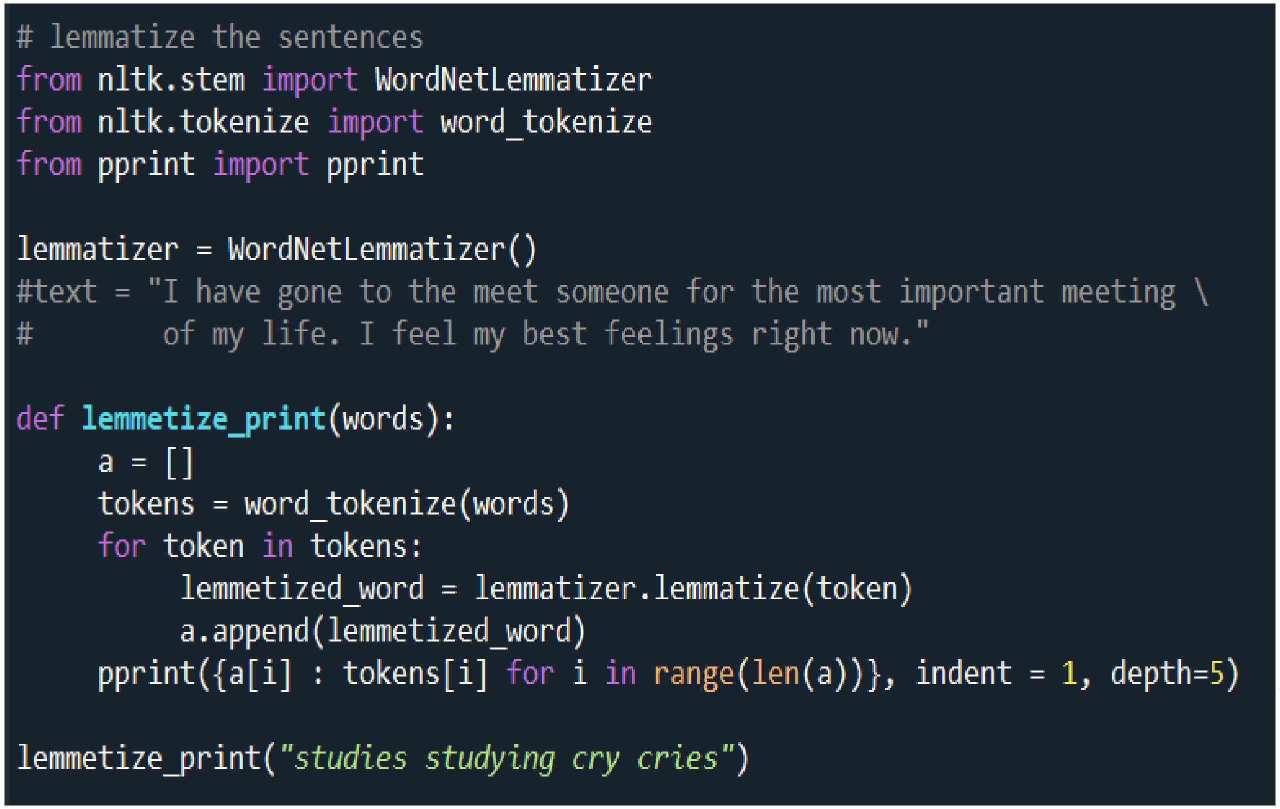
All right

Wrong way



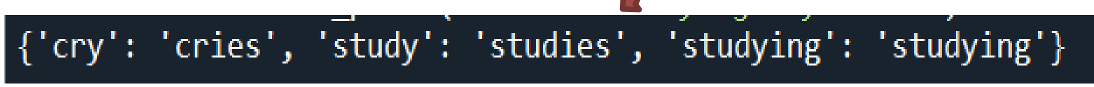
```
Lemmatization:
foot : foot
feet : foot
stemming:
Out[16]: 'feet'
```

- Advanced talks to Lemmatization: To perform Lemmatization with Natural Language Tool Kit (NLTK), “WordNetLemmatizer()” method should be used. “Nltk.stem.WordNetLemmatizer.lemmatize” method will lemmatize a word based on its context and its usage within the sentence. An NLTK Lemmatization example can be found below.



```python
# lemmatize the sentences
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from pprint import pprint

lemmatizer = WordNetLemmatizer()
#text = "I have gone to the meet someone for the most important meeting \
#        of my life. I feel my best feelings right now."

def lemmetize_print(words):
     a = []
     tokens = word_tokenize(words)
     for token in tokens:
          lemmetized_word = lemmatizer.lemmatize(token)
          a.append(lemmetized_word)
     pprint({a[i] : tokens[i] for i in range(len(a))}, indent = 1, depth=5)

lemmetize_print("studies studying cry cries")
```

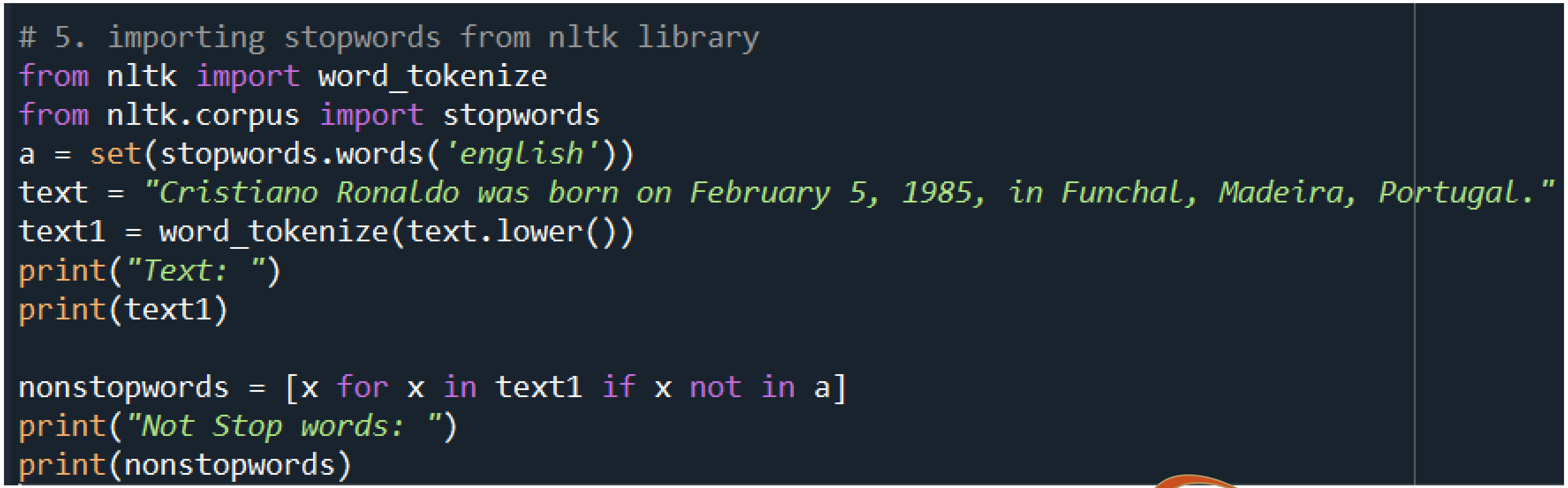
3 of them are ok!



```
{'cry': 'cries', 'study': 'studies', 'studying': 'studying'}
```

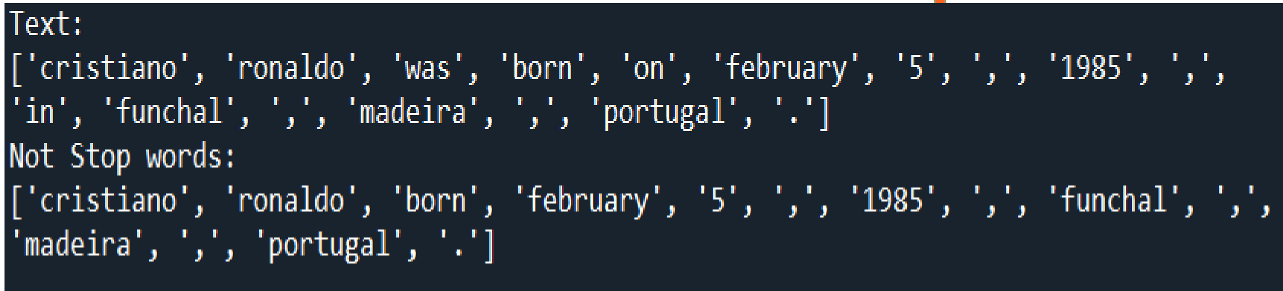
Stop words

- “Stop words” are the most common words in a language like “the”, “a”, “at”, “for”, “above”, “on”, “is”, “all”. These words do not provide any meaning and are usually removed from texts. We can remove these stop words using nltk library

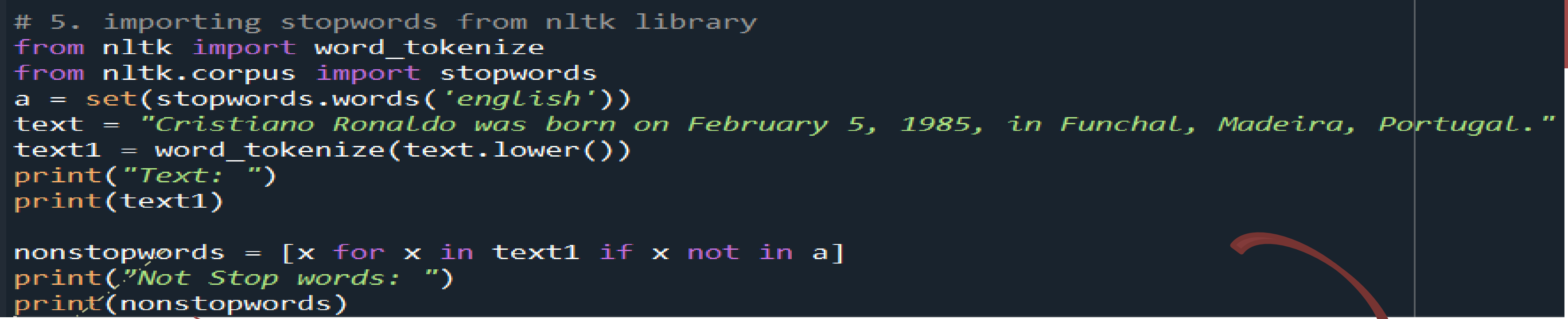


```python
# 5. importing stopwords from nltk library
from nltk import word_tokenize
from nltk.corpus import stopwords
a = set(stopwords.words('english'))
text = "Cristiano Ronaldo was born on February 5, 1985, in Funchal, Madeira, Portugal."
text1 = word_tokenize(text.lower())
print("Text: ")
print(text1)

nonstopwords = [x for x in text1 if x not in a]
print("Not Stop words: ")
print(nonstopwords)
```

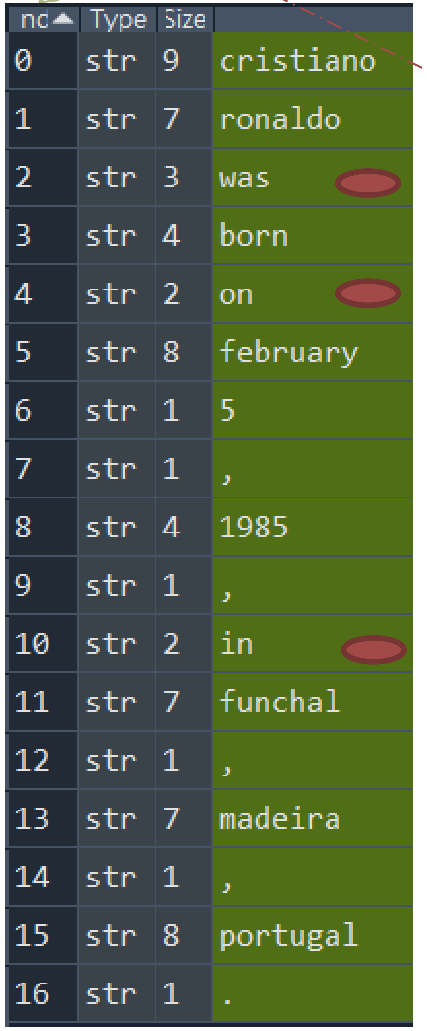
Find nonstop words



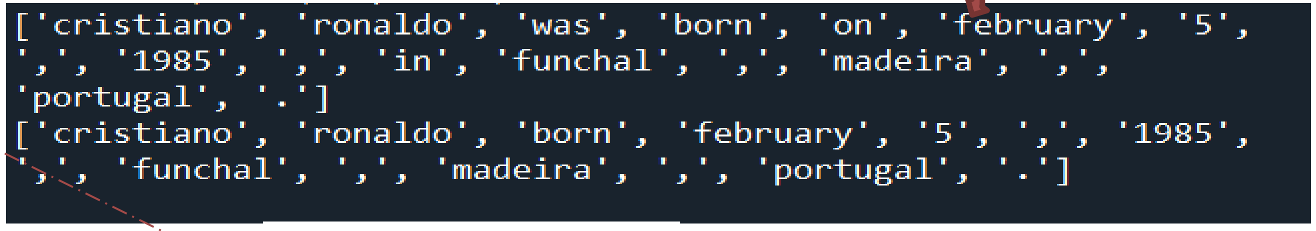
```
Text:
['cristiano', 'ronaldo', 'was', 'born', 'on', 'february', '5', ',', '1985', ',',
'in', 'funchal', ',', 'madeira', ',', 'portugal', '.']
Not Stop words:
['cristiano', 'ronaldo', 'born', 'february', '5', ',', '1985', ',', 'funchal', ',',
'madeira', ',', 'portugal', '.']
```



```python
# 5. importing stopwords from nltk library
from nltk import word_tokenize
from nltk.corpus import stopwords
a = set(stopwords.words('english'))
text = "Cristiano Ronaldo was born on February 5, 1985, in Funchal, Madeira, Portugal."
text1 = word_tokenize(text.lower())
print("Text: ")
print(text1)

nonstopwords = [x for x in text1 if x not in a]
print("Not Stop words: ")
print(nonstopwords)
```

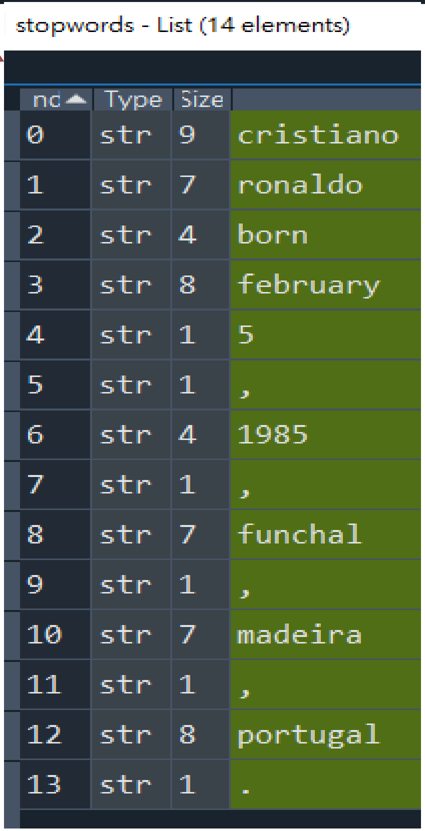


nc	Type	Size	
0	str	9	cristiano
1	str	7	ronaldo
2	str	3	was ●
3	str	4	born
4	str	2	on ●
5	str	8	february
6	str	1	5
7	str	1	,
8	str	4	1985
9	str	1	,
10	str	2	in ●
11	str	7	funchal
12	str	1	,
13	str	7	madeira
14	str	1	,
15	str	8	portugal
16	str	1	.



```
['cristiano', 'ronaldo', 'was', 'born', 'on', 'february', '5',
',', '1985', ',', 'in', 'funchal', ',', 'madeira', ',',
'portugal', '.']
['cristiano', 'ronaldo', 'born', 'february', '5', ',', '1985',
',', 'funchal', ',', 'madeira', ',', 'portugal', '.']
```

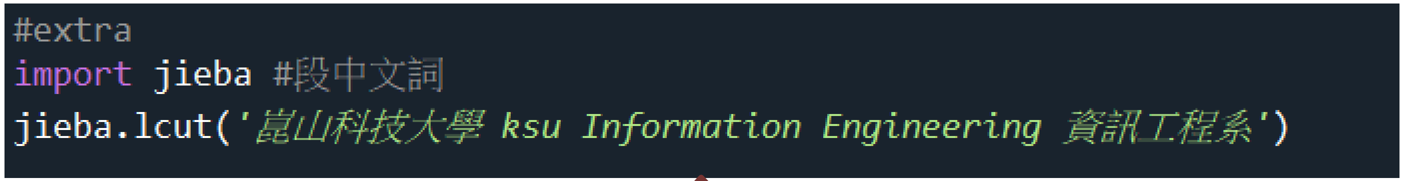
● Stop words have been removed!



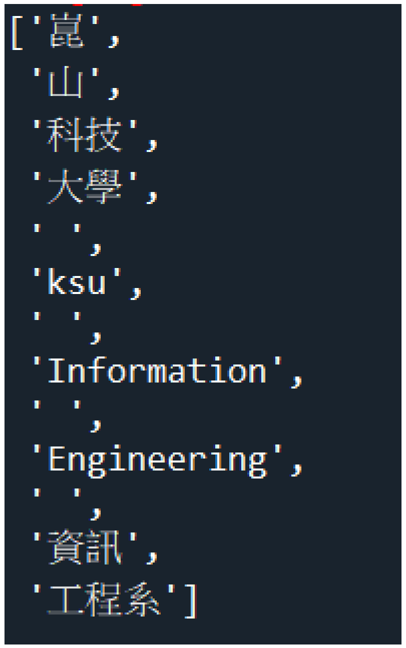
stopwords - List (14 elements)

nc	Type	Size	
0	str	9	cristiano
1	str	7	ronaldo
2	str	4	born
3	str	8	february
4	str	1	5
5	str	1	,
6	str	4	1985
7	str	1	,
8	str	7	funchal
9	str	1	,
10	str	7	madeira
11	str	1	,
12	str	8	portugal
13	str	1	.

中文段辭



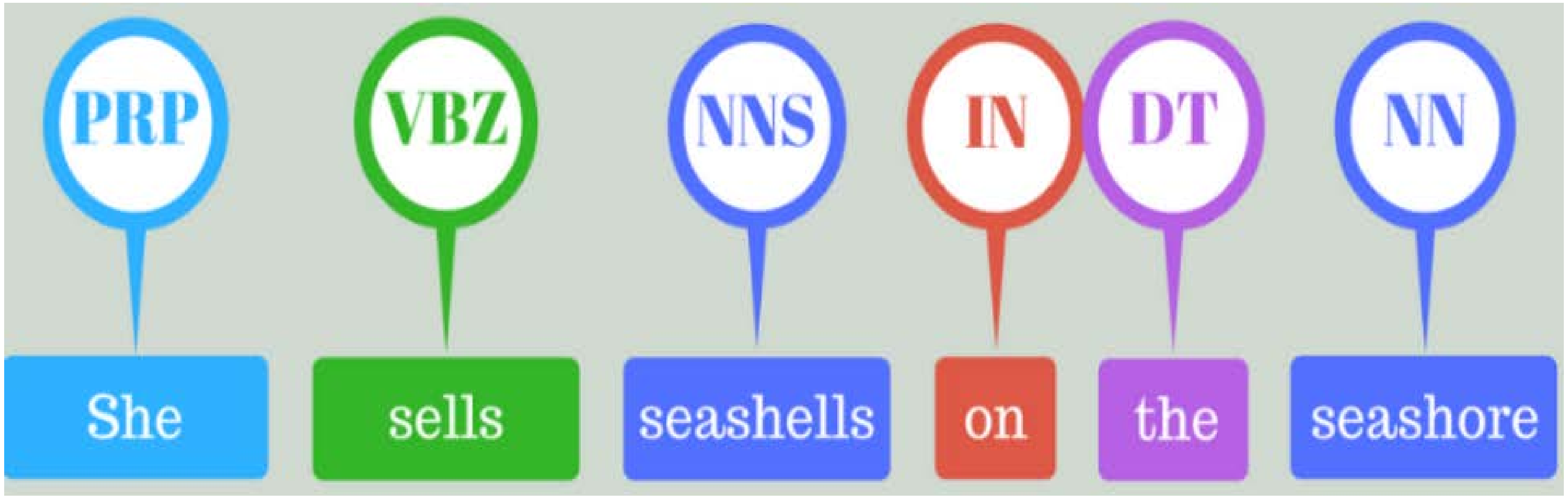
```
#extra
import jieba #段中文詞
jieba.lcut('崑山科技大學 ksu Information Engineering 資訊工程系')
```

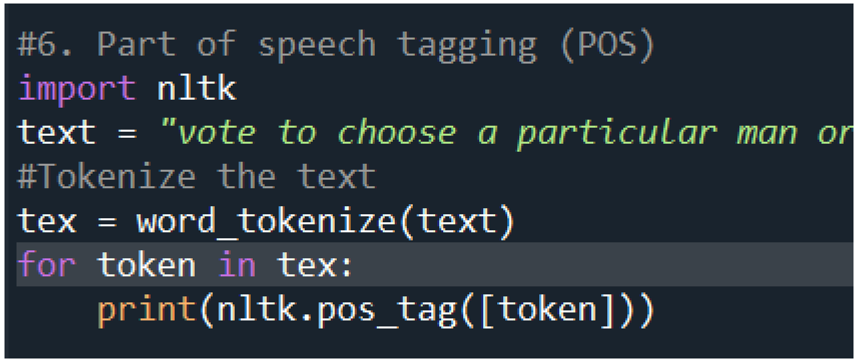


```
['崑',
 '山',
 '科技',
 '大學',
 ' ',
 'ksu',
 ' ',
 'Information',
 ' ',
 'Engineering',
 ' ',
 '資訊',
 '工程系']
```

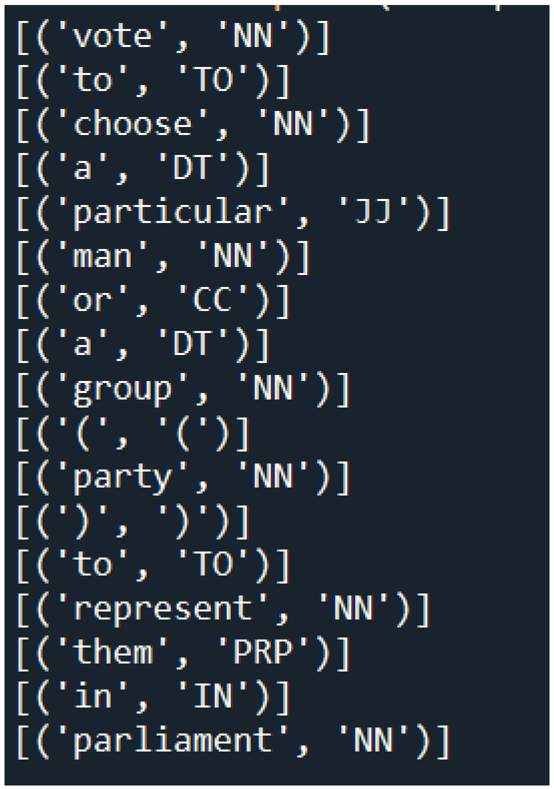
POS

- Part-of-speech tagging (詞性標註) is used to assign parts of speech to each word of a given text (such as nouns, verbs, pronouns, adverbs, conjunction, adjectives, interjection) based on its definition and its context. There are many tools available for POS taggers and some of the widely used taggers are NLTK, Spacy, TextBlob, Standford CoreNLP, etc.

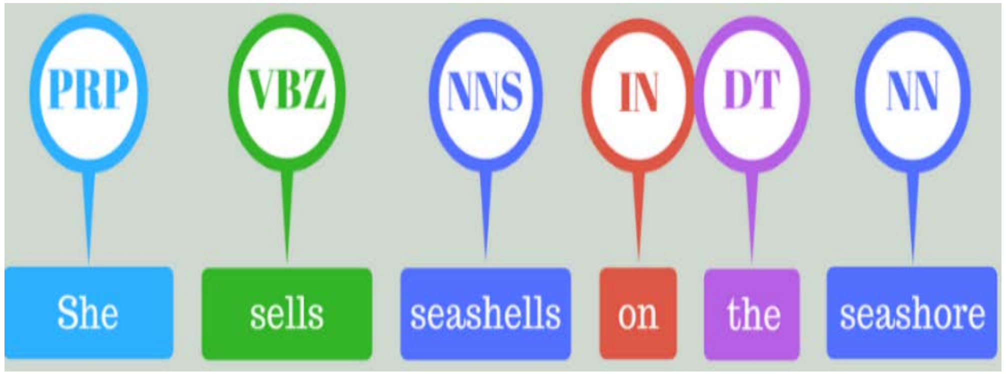




```
#6. Part of speech tagging (POS)
import nltk
text = "vote to choose a particular man or
#Tokenize the text
tex = word_tokenize(text)
for token in tex:
    print(nltk.pos_tag([token]))
```

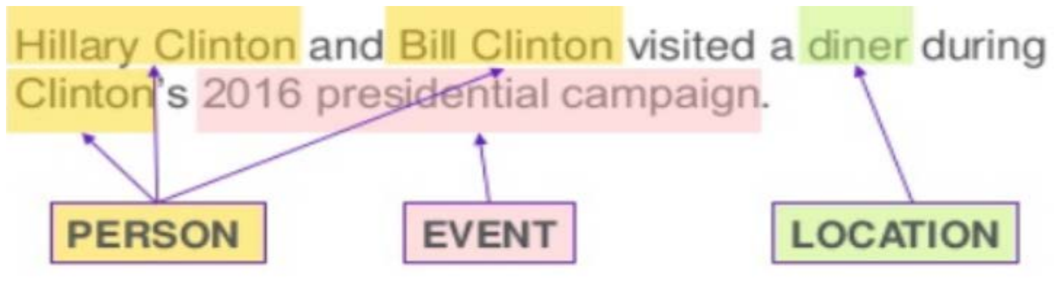


```
[('vote', 'NN')]
[('to', 'TO')]
[('choose', 'NN')]
[('a', 'DT')]
[('particular', 'JJ')]
[('man', 'NN')]
[('or', 'CC')]
[('a', 'DT')]
[('group', 'NN')]
[('(', '(')]
[('party', 'NN')]
[(')', ')')]
[('to', 'TO')]
[('represent', 'NN')]
[('them', 'PRP')]
[('in', 'IN')]
[('parliament', 'NN')]
```



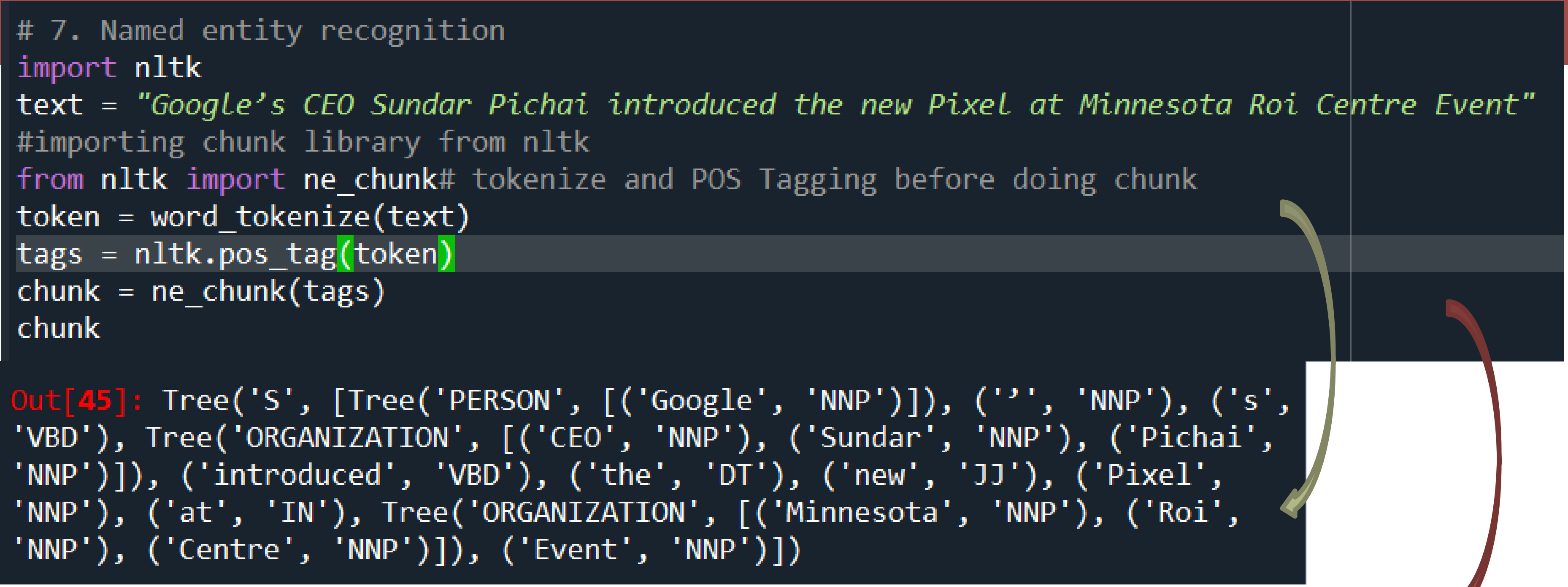
7. Named entity recognition

- It is the process of detecting the named entities such as the person name, the location name, the company name, the quantities, and the monetary value.

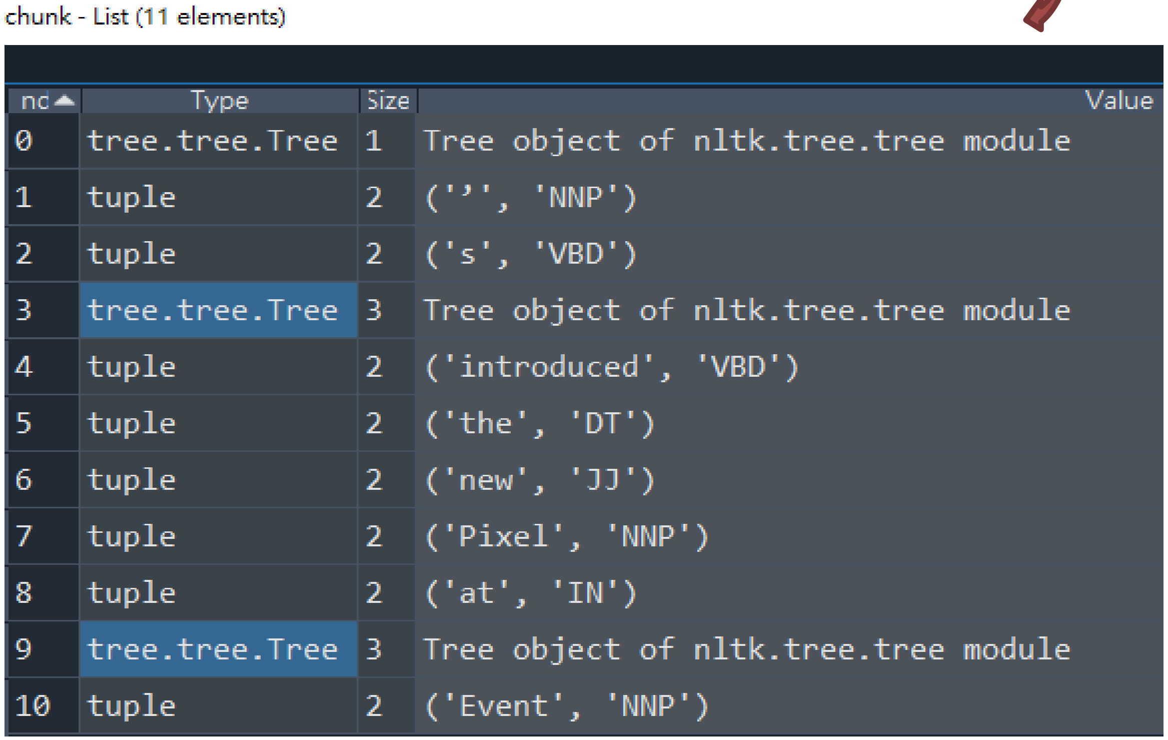


Hillary Clinton and Bill Clinton visited a diner during Clinton's 2016 presidential campaign.
PERSON
EVENT
LOCATION

Ref: Sujit Pal



```
# 7. Named entity recognition
import nltk
text = "Google’s CEO Sundar Pichai introduced the new Pixel at Minnesota Roi Centre Event"
#importing chunk library from nltk
from nltk import ne_chunk# tokenize and POS Tagging before doing chunk
token = word_tokenize(text)
tags = nltk.pos_tag(token)
chunk = ne_chunk(tags)
chunk

Out[45]: Tree('S', [Tree('PERSON', [('Google', 'NNP')]), ('’', 'NNP'), ('s',
'VBD'), Tree('ORGANIZATION', [('CEO', 'NNP'), ('Sundar', 'NNP'), ('Pichai',
'NNP')]), ('introduced', 'VBD'), ('the', 'DT'), ('new', 'JJ'), ('Pixel',
'NNP'), ('at', 'IN'), Tree('ORGANIZATION', [('Minnesota', 'NNP'), ('Roi',
'NNP'), ('Centre', 'NNP')]), ('Event', 'NNP')])
```

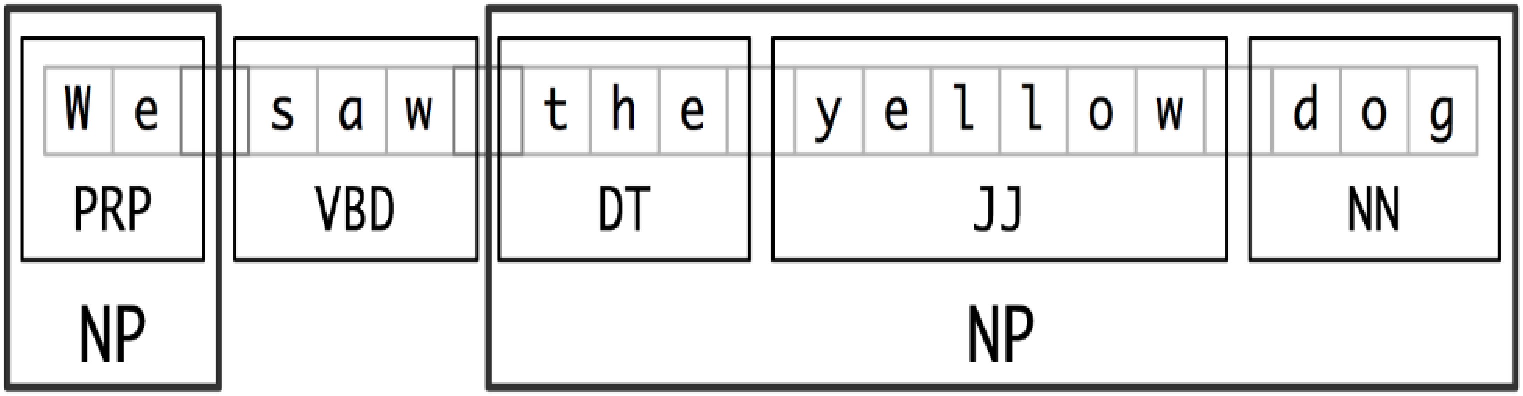


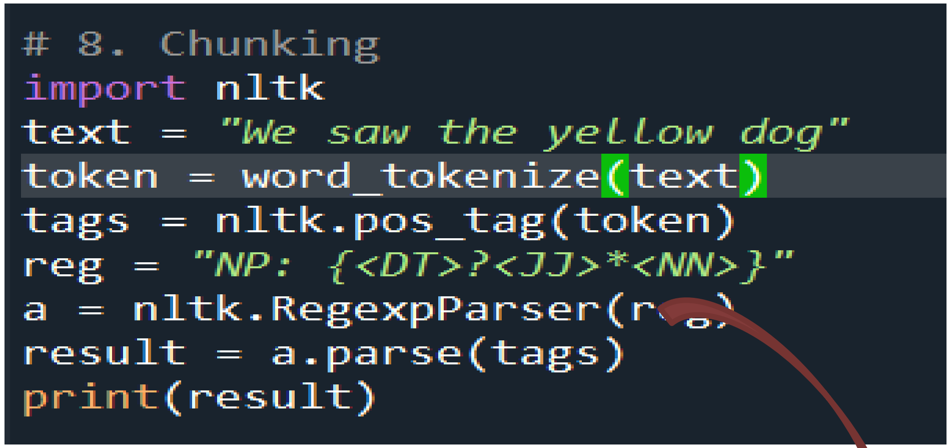
chunk - List (11 elements)

nc	Type	Size	Value
0	tree.tree.Tree	1	Tree object of nltk.tree.tree module
1	tuple	2	('’', 'NNP')
2	tuple	2	('s', 'VBD')
3	tree.tree.Tree	3	Tree object of nltk.tree.tree module
4	tuple	2	('introduced', 'VBD')
5	tuple	2	('the', 'DT')
6	tuple	2	('new', 'JJ')
7	tuple	2	('Pixel', 'NNP')
8	tuple	2	('at', 'IN')
9	tree.tree.Tree	3	Tree object of nltk.tree.tree module
10	tuple	2	('Event', 'NNP')

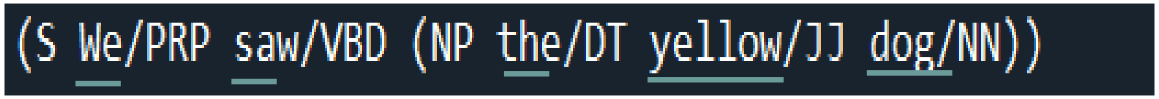
8. Chunking

- Chunking means picking up individual pieces of information and grouping them into bigger pieces. In the context of NLP and text mining, chunking means a grouping of words or tokens into chunks.





```python
# 8. Chunking
import nltk
text = "We saw the yellow dog"
token = word_tokenize(text)
tags = nltk.pos_tag(token)
reg = "NP: {<DT>?<JJ>*<NN>}"
a = nltk.RegexpParser(reg)
result = a.parse(tags)
print(result)
```



```
(S We/PRP saw/VBD (NP the/DT yellow/JJ dog/NN))
```

DA_ex1803.py

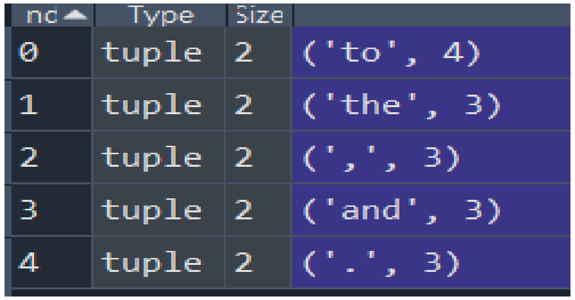
Final exam

(1)

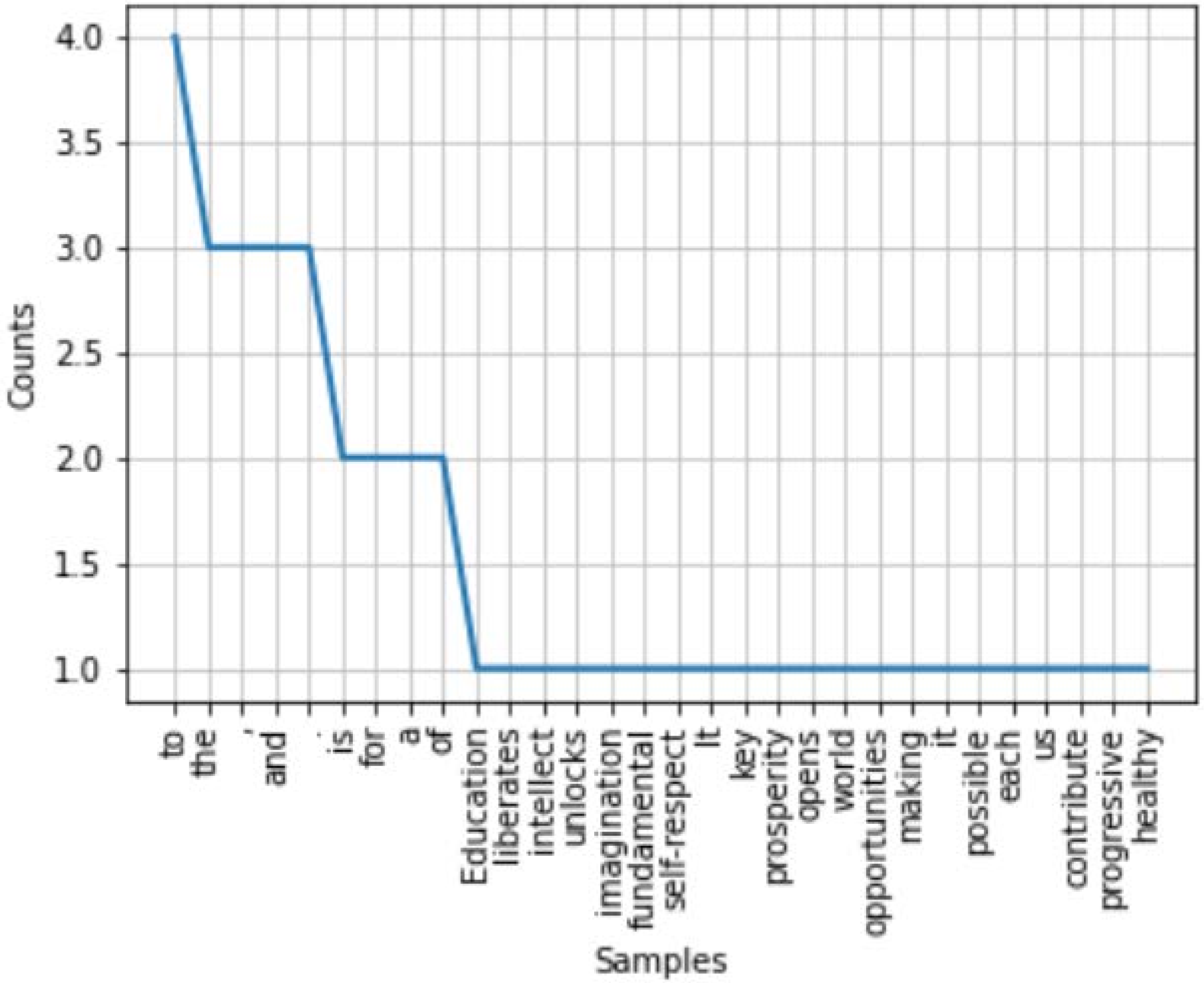
- 請使用Text Mining中的python方法，剖析底下這一段文章，並找出前面5個頻率最高的tokens (字詞)，如表中輸出(ps結果一樣即可,不須列表)。最後列出前面30個tokens (字詞)的頻率圖形如圖顯示。(繳交程式碼 DM_finalexam1.py)

「Education liberates the intellect, unlocks the imagination and is fundamental for self-respect. It is the key to prosperity and opens a world of opportunities, making it possible for each of us to contribute to a progressive, healthy society. Learning benefits every human being and should be available to all.」

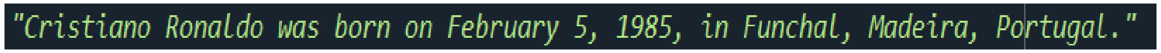
Outputs



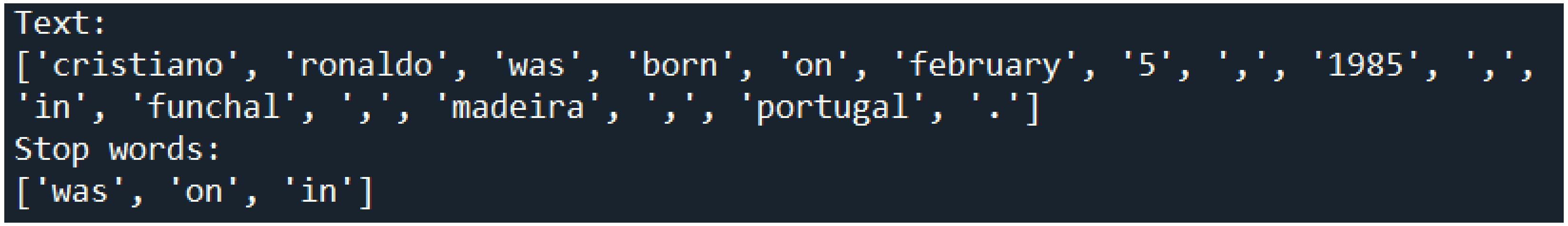
nc▲	Type	Size	
0	tuple	2	('to', 4)
1	tuple	2	('the', 3)
2	tuple	2	(',', 3)
3	tuple	2	('and', 3)
4	tuple	2	('.', 3)



(2) 寫一python (DM_finalexam2.py)列出如下 stop words



```
"Cristiano Ronaldo was born on February 5, 1985, in Funchal, Madeira, Portugal."
```



```
Text:
['cristiano', 'ronaldo', 'was', 'born', 'on', 'february', '5', ',', '1985', ',',
'in', 'funchal', ',', 'madeira', ',', 'portugal', '.']
Stop words:
['was', 'on', 'in']
```

The End