

# COMP348 — Document Processing and the Semantic Web

## Week 01 Lecture 2: Text Processing in Python

Diego Mollá

Department of Computer Science  
Macquarie University

COMP348 2019H1

# Programme

- 1 A Review of Python
  - Practicalities
  - Basic Python
  - Simple Statistics in NLTK
- 2 Text Processing with Python
  - Sorting
  - String Handling
  - Text Preprocessing with NLTK

## Reading

- NLTK Chapter 1

## Additional Reading

- <http://docs.python.org/3/tutorial/index.html>

# Programme

- 1 A Review of Python
  - Practicalities
  - Basic Python
  - Simple Statistics in NLTK
- 2 Text Processing with Python
  - Sorting
  - String Handling
  - Text Preprocessing with NLTK

# Programme

- 1 A Review of Python
  - Practicalities
  - Basic Python
  - Simple Statistics in NLTK
- 2 Text Processing with Python
  - Sorting
  - String Handling
  - Text Preprocessing with NLTK

# Why Python

## Scripting Language

- Rapid prototyping.
- Platform neutral.

## Python

- Even easier prototyping.
  - jupyter notebooks.
- Clean, object oriented.
- Good text manipulation.
- Wide range of libraries.
  - NLTK for NL algorithms.
  - pandas, sklearn, tensorflow for data mining.
  - NumPy and SciPy for scientific computing.
  - matplotlib and pyplot for plotting.

# Installing Python

- Official Python at <http://www.python.org>.
- We will use the Anaconda Python environment from <https://www.anaconda.com/distribution/>.
- Current version is 3.x **do not use 2.x**.
  - Due to compatibility issues, we will use 3.6 (not the latest version).
- Windows/Mac/Linux versions.
- Download includes many libraries.
  - cgi.
  - email.
  - HTMLParser.
- Anaconda includes Jupyter notebooks and Spyder, a useful IDE.

# Installing Python 3.6

- Anaconda supports different environments.
- Each environment can have a different Python interpreter.
- The creation of environments and installation of Python packages is managed by conda.
- Full instructions at <https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html>

## Creating a Python 3.6 Environment

```
$ conda create -n python36 python=3.6
```

# Spyder

The screenshot displays the Spyder Python IDE interface. The main window is divided into several panes:

- File Explorer (Left):** Shows a project structure with folders like `__pycache__`, `Abstracts`, `crossvalida`, `hpcdata`, `hpcdata4b`, `hpcdata10d`, `hpcdataasm`, `logs`, `nnmodels`, `__pycache__`, `logs`, `init`, `allMeSH`, `attention`, `cnn.py`, `compare`, `Explorat`, `golden`, `iris_test`, `iris_train`, `lstm.py`, `my_token`, `simple.p`, `simpleK`, `sklearn`, `tensorfl`, `test_cnn`, `test_lstm`, `savedmode`, `summarise`, `Task4bPub`, `Task5bPub`, `tmp`, `allMeSH_2k`, `allMeSH_2k`, `allMeSH_2k`, `alta2016ta`, `BinningTes`, and `bioasq-out`.
- Code Editor (Center):** Contains the file `scatterplots.py`. The code defines a `scatterplot` function and a `main` function. The `main` function calls `scatterplot` with various parameters to create a scatter plot of the MSE of the target vs. prediction. The code is as follows:
 

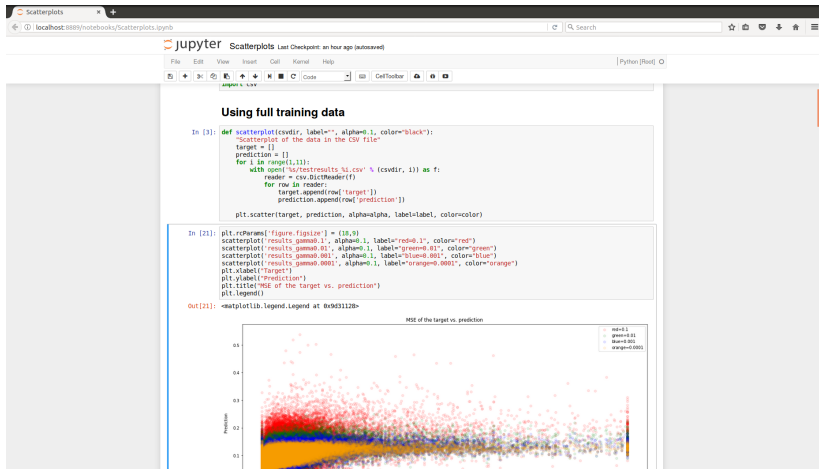
```

1 from matplotlib import pyplot as plt
2 import csv
3
4 def scatterplot(csvdir, label="", alpha=0.1, color="black")
5     "Scatterplot of the data in the CSV file"
6     target = []
7     prediction = []
8     for i in range(1,11):
9         with open("%s/testresults_%i.csv" % (csvdir, i)) as
10             reader = csv.DictReader(f)
11             for row in reader:
12                 target.append(row['target'])
13                 prediction.append(row['prediction'])
14
15     plt.scatter(target, prediction, alpha=alpha, label=label)
16
17 if __name__ == "__main__":
18     scatterplot('results_gamma0.1', alpha=0.1, label='red=0
19     scatterplot('results_gamma0.01', alpha=0.1, label='gree
20     scatterplot('results_gamma0.001', alpha=0.1, label='blu
21     scatterplot('results_gamma0.0001', alpha=0.1, label='or
22     plt.xlabel("Target")
23     plt.ylabel("Prediction")
24     plt.title("MSE of the target vs. prediction")
25
26     plt.show()
27
28 #scatterplot(f, gamma=args.gamma, alpha=args.alpha)
29
30

```
- Object Inspector (Right):** Shows the `scatterplots` object. A tooltip is visible with the text: "Usage: Here you can get help of any object by pressing Ctrl+H in front of it, either on the Editor or the Console. Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Object Inspector."
- IPython Console (Bottom):** Shows the command `runfile('/home/diego/Research/BioASQ/gridsrvr/scatterplots.py', wdir='/home/diego/Research/BioASQ/gridsrvr')` and the resulting scatter plot. The plot is titled "MSE of the target vs. prediction" and shows four data series: red (alpha=0.1), green (alpha=0.01), blue (alpha=0.001), and orange (alpha=0.0001). The x-axis is labeled "Target" and the y-axis is labeled "Prediction".



# Jupyter Notebooks



# Programme

- 1 A Review of Python
  - Practicalities
  - **Basic Python**
  - Simple Statistics in NLTK
- 2 Text Processing with Python
  - Sorting
  - String Handling
  - Text Preprocessing with NLTK

# Beginning Python

This and other Python code available as Jupyter notebooks in github: <https://github.com/dmollaaliod/comp348>.

```
def hello (who):                                # 1
    """ Greet somebody """                     # 2
    print(" Hello_" + who + " !")              # 3

hello(" Steve" )                                # 4
hello( 'World' )                                # 5
people = [ 'Steve' , "Mark" , 'Diego' ]        # 6
for person in people:                           # 7
    hello(person)                               # 8
```

# Core Data Types

- Strings.
- Numbers (integers, float, complex).
- Lists.
- Tuples (immutable sequences).
- Dictionaries (associative arrays).

# Lists I

```
>>> a = ['one', 'two', 3, 'four']
>>> a[0]
'one'
>>> a[-1]
'four'
>>> a[0:3]
['one', 'two', 3]
>>> len(a)
4
>>> a[1]=2
>>> a
['one', 2, 3, 'four']
>>> a.append('five')
>>> a
['one', 2, 3, 'four', 'five']
```

# Lists II

```
>>> top = a.pop()
>>> a
['one', 2, 3, 'four']
>>> top
'five'
```

# List Comprehensions

```
>>> a = ['one', 'two', 'three', 'four']
>>> len(a[0])
3
>>> b = [w for w in a if len(w) > 3]
>>> b
['three', 'four']
>>> c = [[1, 'one'], [2, 'two'], [3, 'three']]
>>> d = [w for [n,w] in c]
>>> d
['one', 'two', 'three']
```

# Tuples

- Tuples are a sequence data type like lists but are immutable:
  - Once created, elements cannot be added or modified.
- Create tuples as literals using parentheses:  
`a = ('one', 'two', 'three')`
- Or from another sequence type:  
`a = ['one', 'two', 'three']`  
`b = tuple(a)`
- Use tuples as fixed length sequences: memory advantages.



# Dictionaries

- Associative array datatype (hash).
- Store values under some hash key.
- Key can be any immutable type: string, number, tuple.

```
>>> names = dict()
>>> names['madonna'] = 'Madonna'
>>> names['john'] = ['Dr.', 'John', 'Marshall']
>>> list(names.keys())
['madonna', 'john']
>>> ages = {'steve':41, 'john':22}
>>> 'john' in ages
True
>>> 41 in ages
False
>>> for k in ages:
...     print(k, ages[k])
steve 41
john 22
```

# Organising Source Code: Modules

- In Python, a **module** is a single source file which defines one or more procedures or classes.
- Load a module with the **import** directive.

**import** mymodule

- This loads the file **mymodule.py** and evaluates its contents.
- By default, all procedures are put into the **mymodule** namespace, accessed with a dotted notation:
  - **mymodule.test()** — calls the **test()** procedure defined in **mymodule.py**

# Modules

- Can import names into global namespace.

```
from mymodule import test , doodle  
from mymodule import *
```

- The Python distribution comes with many useful modules.

```
from math import *  
x = 20 * log(y)  
import webbrowser  
webbrowser.open( 'http://www.python.org' )
```

# Defining Modules

- A module is a source file containing Python code.
  - Usually class/function definitions.
- First non-comment item can be a docstring for the module.

```
# my python module  
"""This is a python module to  
do something interesting"""  
  
def foo(x):  
    'foo_the_x'  
    print('the_foo_is_' + str(x))
```

# Documentation in Python

- Many Python objects have associated documentation strings.
- Good practice is to use these to document your modules, classes and procedures.
- Docstring can be retrieved as the `__doc__` attribute of a module/class/procedure name:

```
def hello (who):  
    """ Greet somebody """  
    print(" Hello_" + who + " !")
```

```
>>> hello.__doc__  
' Greet_somebody '
```

- The function `help()` uses the docstring to generate interactive help.

# Programme

- 1 A Review of Python
  - Practicalities
  - Basic Python
  - Simple Statistics in NLTK
- 2 Text Processing with Python
  - Sorting
  - String Handling
  - Text Preprocessing with NLTK

# NLTK

## What is NLTK?

- Natural Language Toolkit.
- <http://www.nltk.org>.
- A collection of Python 3 libraries.

## Installing NLTK

- <http://www.nltk.org/install.html>.
- Pre-installed in Anaconda.

# Importing NLTK modules

All NLTK modules are under the `nltk` namespace.

```
>>> import nltk
>>> for id in nltk.corpus.gutenberg.fileids():
...     print(id)
...
austen-emma.txt
austen-persuasion.txt
austen-sense.txt
bible-kjv.txt
blake-poems.txt
bryant-stories.txt
burgess-busterbrown.txt
carroll-alice.txt
chesterton-ball.txt
chesterton-brown.txt
```



# Counting Words I

```
>>> import nltk
>>> emma = nltk.corpus.gutenberg.words('austen-emma.txt')
>>> len(emma)
192427
>>> emma[:10]
['[', 'Emma', 'by', 'Jane', 'Austen', '1816', ']', 'VOLUME',
>>> import collections
>>> emma_counter = collections.Counter(emma)
>>> emma_counter.most_common(10)
[(',', 11454), ('.', 6928), ('to', 5183), ('the', 4844),
('and', 4672), ('of', 4279), ('I', 3178), ('a', 3004),
('was', 2385), ('her', 2381)]
```

# Exercises

- 1 Find the most frequent word with length of at least 7 characters.
- 2 Find the words that are longer than 7 characters and occur more than 7 times.

# Counting Bigrams

A bigram is a sequence of two words.

```
>>> list(nltk.bigrams([1,2,3,4,5,6]))  
[(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]  
>>> list(nltk.bigrams(emma))[:3]  
[( '[', 'Emma'), ('Emma', 'by'), ('by', 'Jane')]
```

## Exercises

- 1 Find the most frequent bigram.
- 2 Find the most frequent bigram that begins with “the”.

# Counting Bigrams

A bigram is a sequence of two words.

```
>>> list(nltk.bigrams([1,2,3,4,5,6]))  
[(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]  
>>> list(nltk.bigrams(emma))[:3]  
[( '[', 'Emma'), ('Emma', 'by'), ('by', 'Jane')]
```

## Exercises

- 1 Find the most frequent bigram.
- 2 Find the most frequent bigram that begins with “the”.

# Ngrams

- A bigram is an ngram where n is 2.
- A trigram is an ngram where n is 3.

```
>>> list(nltk.ngrams(emma, 4))[:5]
[(['', 'Emma', 'by', 'Jane'),
 ('Emma', 'by', 'Jane', 'Austen'),
 ('by', 'Jane', 'Austen', '1816'),
 ('Jane', 'Austen', '1816', ')],
 ('Austen', '1816', ']', 'VOLUME'))]
```

# Programme

- 1 A Review of Python
  - Practicalities
  - Basic Python
  - Simple Statistics in NLTK
- 2 Text Processing with Python
  - Sorting
  - String Handling
  - Text Preprocessing with NLTK

# Programme

- 1 A Review of Python
  - Practicalities
  - Basic Python
  - Simple Statistics in NLTK
- 2 Text Processing with Python
  - Sorting
  - String Handling
  - Text Preprocessing with NLTK

# Sorting

- The function **sorted()** returns a sorted copy.
- Sequences can be sorted in place with the **sort()** method.
- Python 3 does not support sorting of lists with mixed contents.

```
>>> foo = [2,5,9,1,11]
>>> sorted(foo)
[1, 2, 5, 9, 11]
>>> foo
[2, 5, 9, 1, 11]
>>> foo.sort()
>>> foo
[1, 2, 5, 9, 11]
>>> foo2 = [2,5,9,1,'a']
>>> sorted(foo2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: str() < int()
```



# Example

```
>>> l = ['a', 'abc', 'b', 'c', 'aa', 'bb', 'cc']
>>> sorted(l)
['a', 'aa', 'abc', 'b', 'bb', 'c', 'cc']
>>> sorted(l, key=len)
['a', 'b', 'c', 'aa', 'bb', 'cc', 'abc']
>>> sorted(l, key=len, reverse=True)
['abc', 'aa', 'bb', 'cc', 'a', 'b', 'c']
>>> sorted(l, key=lambda x: -len(x))
['abc', 'aa', 'bb', 'cc', 'a', 'b', 'c']
```

# Exercises

You're given data of the following form:

```
namedat = dict()  
namedat['mc'] = ('Madonna', 45)  
namedat['sc'] = ('Steve', 41)
```

- ❶ How would you print a list ordered by name?

```
('Madonna', 45)  
( 'Steve', 41)
```

- ❷ How would you print out a list ordered by age?

```
('Steve', 41)  
( 'Madonna', 45)
```

# Programme

- 1 A Review of Python
  - Practicalities
  - Basic Python
  - Simple Statistics in NLTK
- 2 Text Processing with Python
  - Sorting
  - **String Handling**
  - Text Preprocessing with NLTK

# Strings in Python

- String is a base datatype.
- Strings are sequences and can use operations like:
  - `foo = "A_string"`
  - `len(foo)`
  - `foo[0]`
  - `foo[0:3]`
  - `multifoo = """A multiline  
string"""`
- In addition, there are some utility functions in the `string` module.

## Some Useful String Functions

```
>>> "my_string".capitalize()  
'My_string'  
>>> "my_string".upper()  
'MY_STRING'  
>>> "My_String".lower()  
'my_string'  
>>> a = "my_string_with_my_other_text"  
>>> a.count('my')  
2  
>>> a.find('with')  
10  
>>> a.find('nothing')  
-1
```

# Split

- `split` (`sep`) is a central string operation.
- It splits a string wherever `sep` occurs (blank space by default).
- It is either a function in the string module or a method of string objects.

```
>>> foo="one::two::three"
>>> foo.split()
['one', '::', 'two', '::', 'three']
>>> foo.split('::')
['one_', '_two_', '_three']
>>> import string
>>> string.split("this_is_a_test")
['this', 'is', 'a', 'test']
```

# Join

- Join is another useful function/method in the string module.
- It takes a list and joins the elements using some delimiter.

```
>>> text=" this _is _some _text _to _analyse"  
>>> words=text.split()  
>>> words.sort()  
>>> print(" ,".join(words))  
analyse , is , some , text , this , to  
>>> print(" ".join(words))  
analyse is some text this to
```

# Replace

```
def censor(text):  
    'replace bad words in a text with XXX'  
    badwords = ['poo', 'bottom']  
    for b in badwords:  
        text = text.replace(b, 'XXX')  
    return text
```



# Programme

- 1 A Review of Python
  - Practicalities
  - Basic Python
  - Simple Statistics in NLTK
- 2 Text Processing with Python
  - Sorting
  - String Handling
  - Text Preprocessing with NLTK

# NLTK Packaged Tools

Some NLTK tools that are useful for text pre-processing are:

- `word_tokenize(text)`
- `sent_tokenize(text)`
- `pos_tag(tokens)`
- `pos_tag_sents(sentences)`
- `PorterStemmer()`

# Sentence and Word Tokenisation with NLTK

- NLTK can split text into sentences and words.
  - **Sentence segmentation** splits text into a list of sentences.
  - **Word tokenisation** splits text into a list of words (tokens).
- NLTK's default word tokeniser works best after splitting the text into sentences.

```
In [1]: import nltk
```

```
In [2]: text = "This_is_a_sentence._This_is_another_sentence."
```

```
In [3]: nltk.sent_tokenize(text)
```

```
Out[3]: ['This_is_a_sentence.', 'This_is_another_sentence.']
```

```
In [4]: for s in nltk.sent_tokenize(text):  
...:     for w in nltk.word_tokenize(s):  
...:         print(w)  
...:     print()  
...:
```

# Part of Speech Tagging

- Often it is useful to know whether a word is a noun, or an adjective, etc. These are called **parts of speech**.
- NLTK has a part of speech tagger that tags a list of tokens.
- The default list of parts of speech is fairly detailed but we can set a simplified version (called **universal** by NLTK).

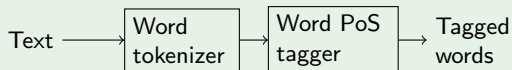
Tag	Meaning	English Examples
ADJ	adjective	new, good, high, special, big, local
ADP	adposition	on, of, at, with, by, into, under
ADV	adverb	really, already, still, early, now
CONJ	conjunction	and, or, but, if, while, although
DET	determiner, article	the, a, some, most, every, no, which
NOUN	noun	year, home, costs, time, Africa
NUM	numeral	twenty-four, fourth, 1991, 14:24
PRT	particle	at, on, out, over per, that, up, with
PRON	pronoun	he, their, her, its, my, I, us
VERB	verb	is, say, told, given, playing, would
.	punctuation marks	. , ; !
X	other	ersatz, esprit, dunno, gr8, univeristy

# NLP Pipelines

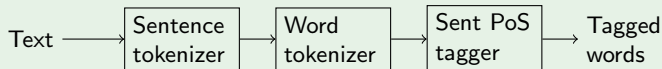
Often, text processing works in **pipelines**.

- The output of one module is used as the input to the following module.

## NLP pipeline for Word PoS tagging in NLTK



## NLP pipeline for Sentence PoS tagging in NLTK



Note that the above pipelines may differ in other environments.

# Examples in Python

```
In [28]: nltk.pos_tag(["this", "is", "a", "test"])
Out[28]: [('this', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('test', 'NN')]

In [29]: nltk.pos_tag(["this", "is", "a", "test"], tagset="universal")
Out[29]: [('this', 'DET'), ('is', 'VERB'), ('a', 'DET'), ('test', 'NOUN')]

In [30]: nltk.pos_tag(nltk.word_tokenize("this_is_a_test"), tagset="universal")
Out[30]: [('this', 'DET'), ('is', 'VERB'), ('a', 'DET'), ('test', 'NOUN')]

In [31]: text
Out[31]: 'This_is_a_sentence._This_is_another_sentence._'

In [34]: text_sent_tokens = [nltk.word_tokenize(s) for s in nltk.sent_tokenize(t
...: ext)]

In [35]: text_sent_tokens
Out[35]:
[['This', 'is', 'a', 'sentence', '.'],
 ['This', 'is', 'another', 'sentence', '.']]

In [38]: nltk.pos_tag_sents(text_sent_tokens, tagset="universal")
Out[38]:
[[('This', 'DET'),
 ('is', 'VERB'),
 ('a', 'DET'),
 ('sentence', 'NOUN'),
 ('.', '.')],
 [('This', 'DET'),
 ('is', 'VERB'),
```

# Stemming

- Often it is useful to remove information such as verb form, or the difference between singular and plural.
- NLTK offers stemming, which removes suffixes.
  - The Porter stemmer is a popular stemmer.
- The remaining stem is not a word but can be used, for example, by search engines (we'll see more of this in another lecture).

# Examples of NLTK's Porter Stemmer

```
In [46]: s = nltk.PorterStemmer()
```

```
In [47]: s.stem("books")
```

```
Out[47]: 'book'
```

```
In [48]: s.stem("is")
```

```
Out[48]: 'is'
```

```
In [50]: s.stem("runs")
```

```
Out[50]: 'run'
```

```
In [51]: s.stem("running")
```

```
Out[51]: 'run'
```

```
In [52]: s.stem("run")
```

```
Out[52]: 'run'
```

```
In [53]: s.stem("goes")
```

```
Out[53]: 'goe'
```



# Exercises

- 1 What is the sentence with the largest number of tokens in Austen's "Emma"?
- 2 What is the most frequent part of speech in Austen's "Emma"?
- 3 What is the number of distinct stems in Austen's "Emma"?
- 4 What is the most ambiguous stem in Austen's "Emma"? (meaning, which stem in Austen's "Emma" maps to the largest number of distinct tokens?)

# Take-home Messages

- Get to know `nltk`.
- Practice with NLP pipelines.
- Do the exercises listed in the lecture notes.

# What's Next

## Week 2

- Information Retrieval

## Additional Reading

- “Introduction to Information Retrieval”,  
<http://www-nlp.stanford.edu/IR-book/>