

# Exploring Python



**Manjary P. Gangan**

Ph.D. Research Scholar  
Digital Image Forensics  
Department of Computer Science  
University of Calicut, Kerala – 673635  
Email: [manjaryp\\_dcs@uoc.ac.in](mailto:manjaryp_dcs@uoc.ac.in)  
Web: <http://dcs.uoc.ac.in/~manjary>



**Anoop K.**

Ph.D. Research Scholar  
Machine Learning & Big Data Analytics  
Department of Computer Science  
University of Calicut, Kerala – 673635  
Email: [anoopk\\_dcs@uoc.ac.in](mailto:anoopk_dcs@uoc.ac.in)  
Web: <http://dcs.uoc.ac.in/~anoop>

## Motivation:

To drive you into the Pool of  
Python Programming Language



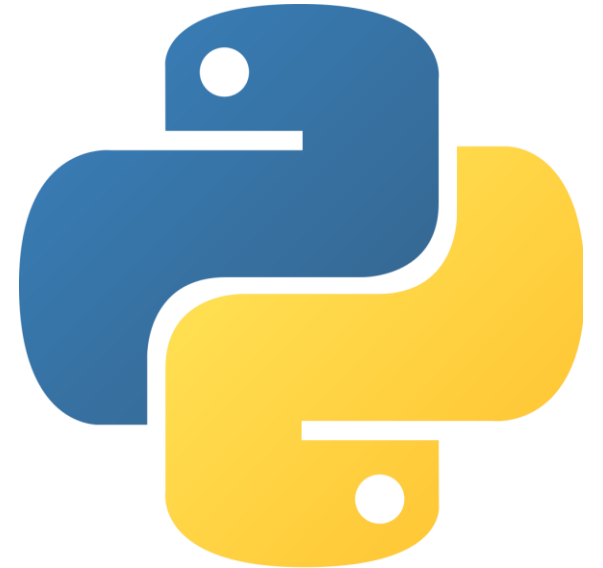
## Outcome:

Familiar to necessary terms and concepts

**Rest of the part:** is to start learn them hardly  
That's not at all difficult, it is web era...

# Outline

- ❑ Anaconda & Spyder - IDE
- ❑ Beauty of Python Code
- ❑ Data Management
- ❑ Decision Making
- ❑ Functions
- ❑ NLTK
- ❑ Data Visualization



# Anaconda

- ❖ The Most Popular Python Data Science Platform
- ❖ With over 4.5 million users, Anaconda is the world's most popular Python data science platform
- ❖ Anaconda, Inc. continues to lead open source projects like Anaconda, NumPy and SciPy that form the foundation of modern data science.

# Installation in Linux

**Link:** <https://www.anaconda.com/download/>

- ❖ `bash Anaconda-latest-Linux-x86_64.sh`
- ❖ `source ~/.bashrc`
- ❖ `anaconda-navigator`
- ❖ Launch Spyder



FileEditSearchSourceRunDebugConsolesProjectsToolsViewHelp

media/manjary/17EB-36C5/notes2learn/python\_programs

media/manjary/17EB-36C5/notes2learn/python\_programs/comparison.py

string.pyXlist.pyXtuple.pyXdictionary.pyXarithmetic.pyXcomparison.py\*

1#!/usr/bin/env python3

2# -\*- coding: utf-8 -\*-

3"""

4Created on Wed Nov 15 16:58:38 2017

5

6@author: manjary

7"""

8

9a = 10

10b = 20

11

12print("a equals b", a==b)

13print("a not equals b", a!=b)

14print("a greater than b", a>b)

15print("a less then b", a<b)

16print("a greater than b", a>b)

17print("a less then b", a<b)]

Variable explorer

Name	Type	Size	Value
a	int	1	10
b	int	1	20

HelpVariable explorerFile explorerStatic code analysisProfiler

IPython console

Console 1/A

```
python_programs/comparison.py', wdir='/media/manjary/17EB-36C5/notes2learn/python_programs')
a equals b False
a not equals b True
a greater than b False

In [2]: runfile('/media/manjary/17EB-36C5/notes2learn/python_programs/comparison.py', wdir='/media/manjary/17EB-36C5/notes2learn/python_programs')
a equals b False
a not equals b True
a greater than b False
a less then b True

18 November 2017

In [3]:
```

IPython consoleHistory log

Permissions: RW

End-of-lines: LF

Encoding: UTF-8

Line: 17

Column: 28

Memory: 71 %

# Beauty of Python



18 November 2017

# Lines and Indentation

---

Python doesn't use braces({}) to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

## Example #1:

```
if True:
    print ("True")
else:
    print ("False")
```

## Example #2:

```
if False:
    print ("True")
else:
    print ("False")
```



# Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.

## Example #1:

```
name_one="Anoop"  
name_two="Lakshmi"  
name_three="Abhi Ram"  
names= name_one + \  
        name_two + \  
        name_three  
print names
```

# Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string. The triple quotes are used to span the string across multiple lines

## Example #1:

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
print word
print sentence
print paragraph
```

# Comments in Python

---

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

```
print "Hello, Python!" # second comment
```

# Multiple Statements on a Single Line

---

The semicolon ( ; ) allows multiple statements on the single line given that neither statement starts a new code block.

```
name="anoop"; print name #prints the output as anoop
```

# Python Input, Output and Import

## I. Output

We use the `print()` function to output data to the standard output device (screen).

### Example #1

```
>>> print('This sentence is output to the screen')
>>> This sentence is output to the screen
>>> a = 5
>>> print('The value of a is', a)
>>> The value of a is 5
```

## II. Input

Up till now, our programs were static. The value of variables were defined or hard coded into the source code. To allow flexibility we might want to take the input from the user. In Python, we have the `input()` function to allow this. The syntax for `input()` is

```
>>> input([prompt])
```

### Example #1:

```
>>> num = input('Enter a number: ')
```

```
Enter a number: 10
```

```
>>> num
```

```
10
```

# Data Management



# Data Types

- A person's age - numeric value
- Address - alphanumeric characters.



# Standard Data types in Python

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary

# Number

- Number data types store numeric values.
- Number objects are created when you assign a value to them.

*Eg: var1 = 1*

*Eg: var2 = 10*

# String

- Contiguous set of characters
- Represented in the quotation marks (single or double quotes)
- Subsets of strings can be taken using the slice operator (`[ ]` and `[:]`) with indexes starting at 0
- `+` : string concatenation operator
- `*` : repetition operator

*[Refer python program: string.py]*

# Lists

- A list contains items separated by commas
- Enclosed within square brackets ([]).
- The items can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:])
- + - list concatenation
- \* - repetition operator

# Tuples

- Similar to the list.
- Values separated by commas
- Enclosed within parentheses
- Tuple cannot be updated.
- Tuples are **read-only** lists.

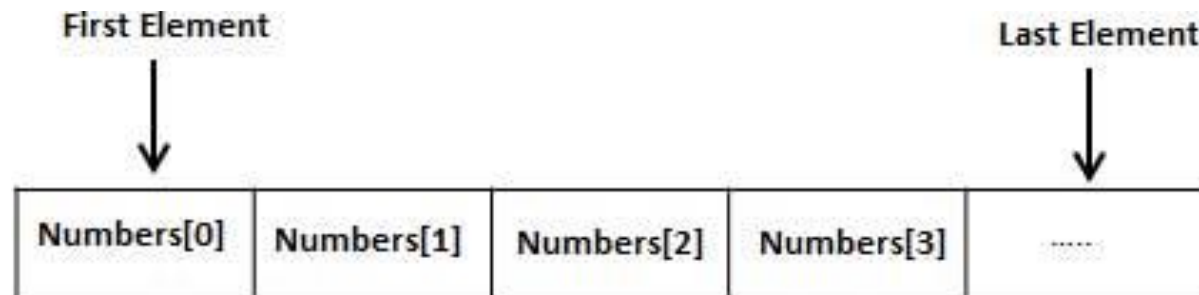
*[Refer python program: tuple.py]*

# Dictionary

- Consist of key-value pairs
- Enclosed by curly braces ({ })
- Values can be assigned and accessed using square braces ([]).

# Array

- A kind of data structure
- Store fixed-size sequential collection
- Store elements of the same type
- Contiguous memory locations



$a = [1, 2, 5, 6]$

1	2	5	6		
---	---	---	---	--	--

$a[0] = 1, \quad a[3] = 6$

0	5	6	8
1	4	6	1
2	8	4	2

$a[0][0] = 0, a[0][1] = 5,$   
 $a[1][0] = 1, a[2][1] = 8$



# Numpy

- <http://www.numpy.org/>
- `import numpy`

*[Refer python program: array.py]*

# Basic Operation in Python

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Membership Operators

# Arithmetic Operators

Operator (a = 10, b = 20)	Example
+ Addition	$a + b = 30$
- Subtraction	$a - b = -10$
* Multiplication	$a * b = 200$
/ Division	$b / a = 2$
% Modulus	$b \% a = 0$
** Exponent	$a ** b = 10 \text{ to the power } 20$
// Floor division	$9 // 2 = 4$ and $9.0 // 2.0 = 4.0$

18 November 2017

27

[Refer python program: [arithmetic.py](#)]

# Comparison Operators

Operator (a = 10, b = 20)	Example
==	(a == b) is not true.
!=	(a != b) is true.
>	(a > b) is not true.
<	(a < b) is true.
>=	(a >= b) is not true.
<=	(a <= b) is true.

# Assignment Operators

Operator (a = 10, b = 20)	Example
=	c = a + b
+= Add AND	c += a
-= Subtract AND	c -= a
*= Multiply AND	c *= a
/= Divide AND	c /= a
%= Modulus AND	c %= a
**= Exponent AND	c **= a
//= Floor Division	c //= a

# Membership Operators

Operator	Example
in	$x \text{ in } y = 1$ if, $x$ is a member of sequence $y$ .
not in	$x \text{ not in } y = 1$ , if $x$ is not a member of sequence $y$ .

# Decision Making

- Conditions
- Looping



## Conditional Operations

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions. Decision structures evaluate multiple expressions which produce **TRUE** or **FALSE** as outcome.

**You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise**



---

Python programming language assumes **any non-zero and non-null values as TRUE**, and if it is either **zero or null**, then it is assumed as **FALSE** value. Python programming language provides following types of decision making statements.

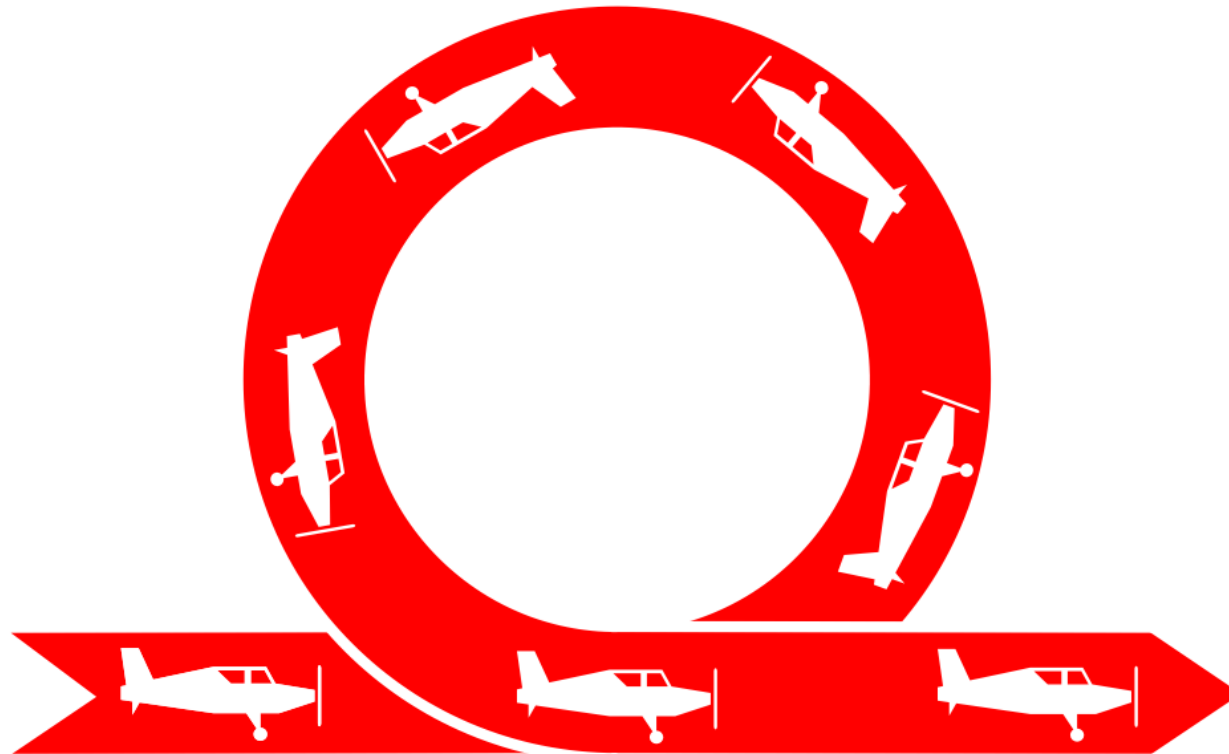
**1. IF Statement**

**2. IF...ELIF...ELSE Statements**

**3. Nested IF statements**

*[Refer python program: `if_else.py`, `if_elseif`, `nested_if.py`]*

# Looping in Python



# Iteration and Looping in Python

---

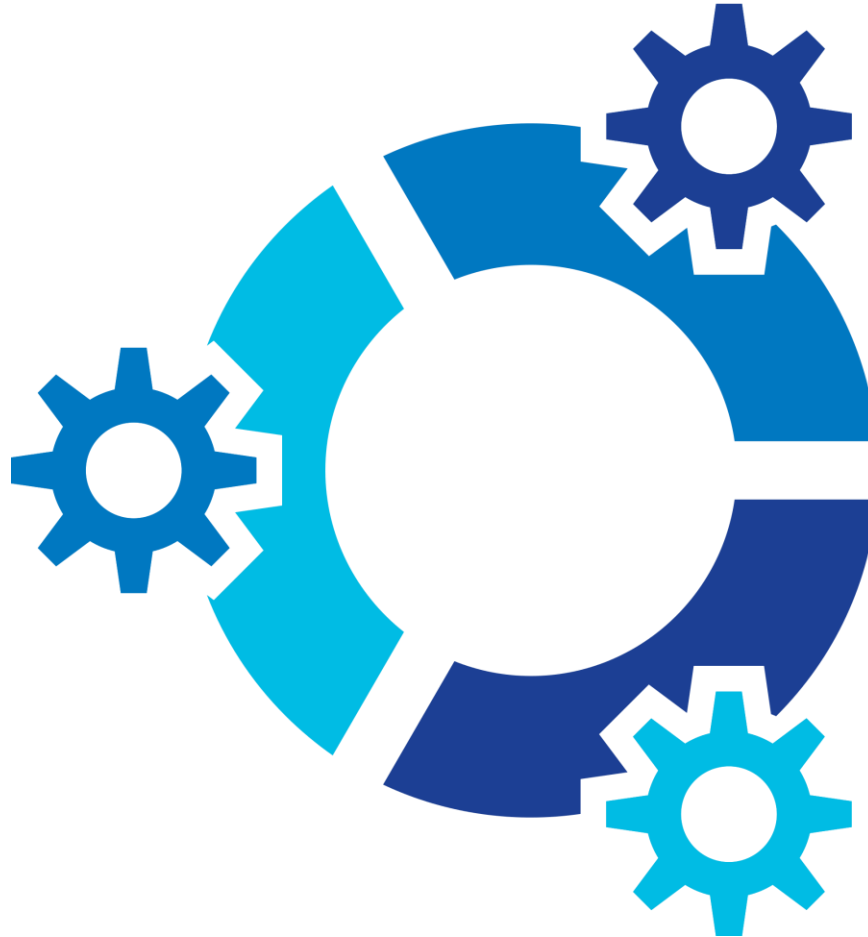
In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times. Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement –


Python programming language provides following types of loops to handle looping requirements.

- 1. while loop**
- 2. for loop**

*[Refer python program: [for.py](#), [while.py](#)]*

# Functions





A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. As you already know, Python gives you many built-in functions like `print()`, etc. but you can also create your own functions. These functions are called user-defined functions.

# Defining a Function

---

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword `def` followed by the function name and parentheses `( )`.
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The code block within every function starts with a colon `(:)` and is indented.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The statement `return [expression]` exits a function, optionally passing back an expression to the caller.
- A return statement with no arguments is the same as `return None`.

# Syntax

---

```
def functionname( parameters ):  
    function_suite  
return [expression]
```

By default, parameters have a positional behavior and you need to inform them in the same order that they were defined

## Example #1: Calling a Function

The following function takes a string as input parameter and prints it on standard screen.

```
def printme( str ):  
    print str  
return  
Printme("anoop")
```

## Example #2:

```
total = 0; # This is global variable.  
def sum( arg1, arg2 ):  
    "Add both the parameters and return them."  
    total = arg1 + arg2; # Here total is local variable.  
    print "Inside the function local total : ", total  
    return total;  
# Now you can call sum function  
sum( 10, 20 );  
print "Outside the function global total : ", total
```

When the above code is executed, it produces the following result:

Inside the function local total : 30

Outside the function global total : 0



# Built-in Functions

---

- `pow(x,y)`: This method returns value of  $x^y$ .

```
#!/usr/bin/python
import math # This will import math module
```

```
print "math.pow(100, 2) : ", math.pow(100, 2)
print "math.pow(2, 4) : ", math.pow(2, 4)
print "math.pow(3, 0) : ", math.pow(3, 0)
```

# Python Modules

---

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

## Example #1:

```
#save the python file as support.py
def print_func( par ):
    print "Hello : ", par
    Return
```

---

**Syntax:** `import module1[, module2[,... moduleN]`

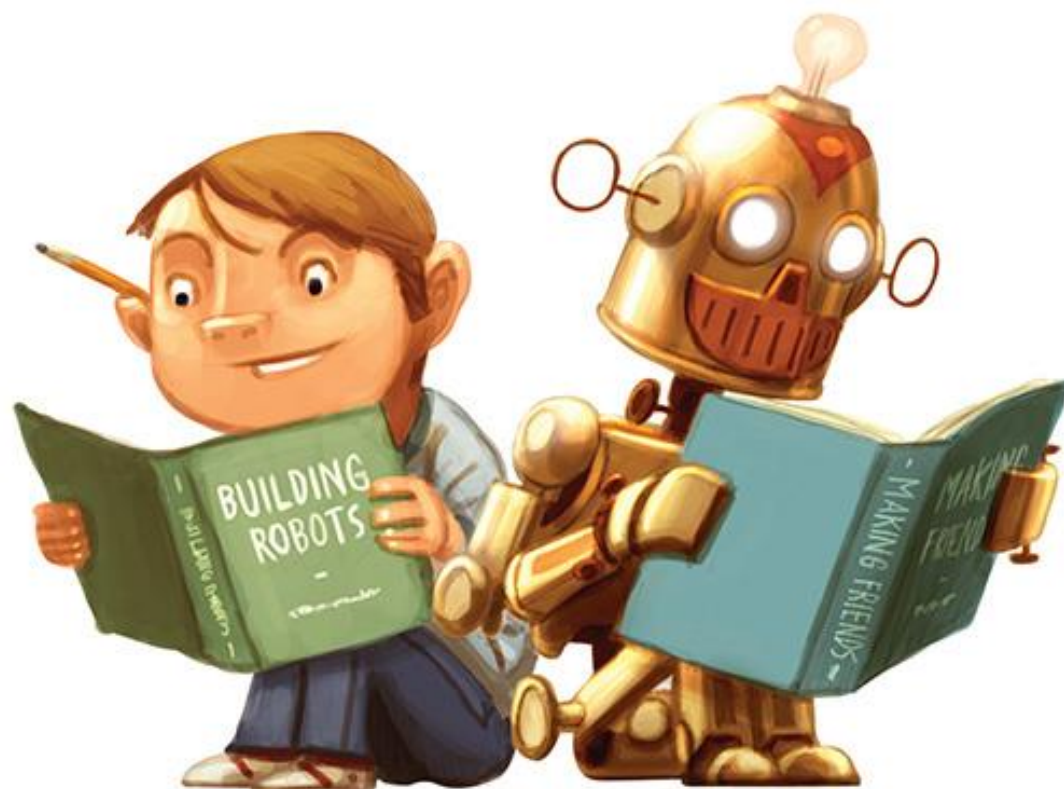
**Example #2:**

```
# Import module support
import support
# Now you can call defined function that module as follows
support.print_func("Appu")
```

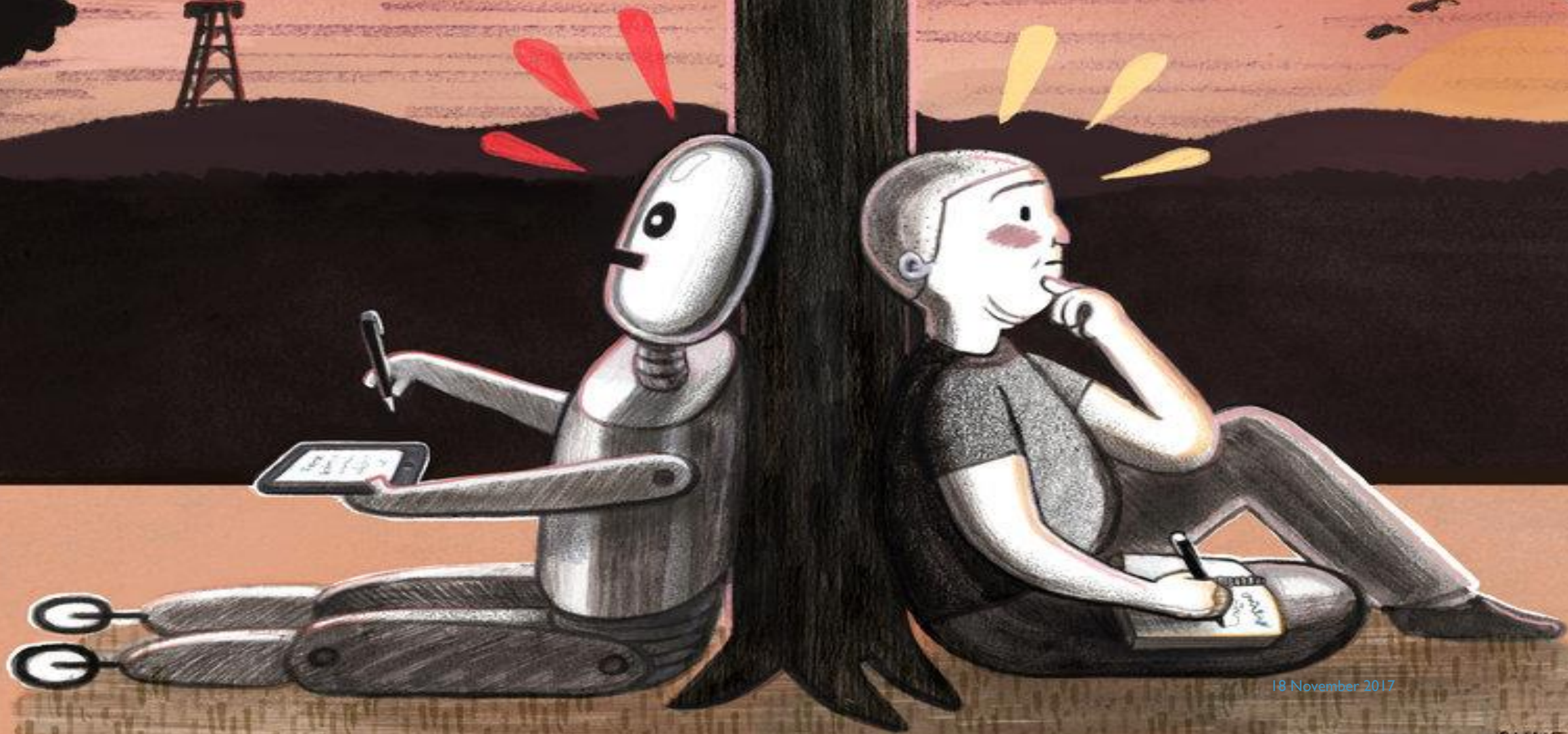
**Output:**

Hello :Appu

# Textual Data Preprocessing with Python NLTK







18 November 2017

ANNELISE  
CAPOSSELA

# Outline

- ☐ Introduction NLTK
- ☐ Tokenization
- ☐ Stop Words
- ☐ Stemming
- ☐ Lemmatization



# Introduction Natural language

Natural language processing (NLP) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages and, in particular, concerned with programming computers to fruitfully process large natural language corpora.

## **Applications :**

- 1. Question Answering**
- 2. Sentiment Analysis**
- 3. Machine Translation**



# Computing with Language: Texts

*Posted by: John Smith*

*Date: March 24, 2017*

"I bought a Canon G12 camera six months ago. I simply love it. The picture quality is amazing. The battery life is also long. However, my wife thinks it is too heavy for her."

from this review,  
we notice a few important points.



18 November 2017



“although the **service** is not that great, I still love this **restaurant**”

- ✓ clearly has a positive tone, we cannot say that this sentence is entirely positive.
- ✓ In fact, the sentence is positive about the restaurant (emphasized)
- ✓ but negative about its service (not emphasized).

We process the texts and words to understand the sentiment



18 November 2017

# Sentiment Analysis as an Application of NLP

**Sentiment analysis, also called opinion mining,** is the field of study that analyzes people

*Opinions*

*Evaluations*

*Attitudes*

*Sentiments*

*Appraisals*

*Emotions*

Towards entities such as products, services, organizations, individuals, issues, events, topics, and their attributes

# Language Computing using Python

The **NLTK module** is a **python based massive tool kit**, aimed at helping you with the entire Natural Language Processing (NLP) methodology.

NLTK will aid you with everything from **splitting sentences** from paragraphs, splitting up **words**, recognizing the **part of speech** of those words, highlighting the main subjects, and then even with helping your machine to understand what the text is all about.



Natural Language  
Analyses with NLTK

# Installing the NLTK

- ✓ **NLTK requires Python versions 2.7 or 3.4+**
- ✓ **Downloadable for free from <http://nltk.org/>**

Install NLTK: **\$ sudo pip install -U nltk**

Install Numpy (optional): **\$ sudo pip install -U numpy**

Test installation: **run python then import nltk**

---

```
$python
```

```
>> import nltk
```

```
>> nltk.download()
```

**I. If it shows the interface, download everything (NLTK corpus and all)**

**II. If no interface use:**

☐ **d** (for download)

☐ **all** (for download everything)

☐ That will download everything for you.

# Steps for NLTK Installation is Available @

---



- ❑ Channel: Notes2Learn
- ❑ Link: [https://www.youtube.com/watch?v=Rh\\_n7lWDVB4](https://www.youtube.com/watch?v=Rh_n7lWDVB4)

# Pre-processing with NLTK

Pre-processing is an important task and critical step in **Text mining, Natural Language Processing (NLP) and information retrieval (IR)**.

**Pre-processing technique that involves transforming raw data into an understandable format.**

Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviours or trends, and is likely to contain many errors. Data pre-processing is a proven method of resolving such issues.

**Data pre-processing prepares raw data for further processing.**

**1. Data Cleaning:** Data is cleansed through processes such as filling missing values, smoothing the noisy data, or resolving the inconsistencies in the data.



**Donald J. Trump** ✓  
@realDonaldTrump

 **Following**

“@realDonaldTrump: I would like to extend my best wishes to all, even the haters and losers, on this special date, September 11th.”



realDonaldTrump I would like to extend my best wishes to all even the haters and losers on this special date September 11th



## 2. Normalization

usa  
U.S.A. } U S A

## 3. Tokenization

- ✓ Cut sentence into word tokens
- ✓ Cut paragraphs into sentence tokens

## 4. Stop Word Removal

## 5. Stemming

## 6. Lemmatization

# Outline

- ☒ Introduction ~~NLTK~~
- ☐ Tokenization
- ☐ Stop Words
- ☐ Stemming
- ☐ Lemmatization



# Tokenization

In lexical analysis, **tokenization** is the process of breaking a **stream of text up into words, phrases, symbols, or other meaningful elements called tokens**. The list of tokens becomes input for further processing such as parsing or text mining.

Tokenization is the task of **chopping it up into pieces**, called *tokens* , perhaps at the same time throwing away certain characters, such as punctuation.

*Input:* Friends, Romans, Peter, lend me your ears;

*Output:*

Friends

Romans

Peter

lend

me

your

ears

- ✓ These tokens are often loosely referred to as **terms or words**
- ✓ A *token* is an instance of a **sequence of characters** in some particular document that are grouped together as a **useful semantic unit for processing**

1. `sent_tokenize`
2. `word_tokenize`
3. `RegexTokenizer`

# Tokenization using NLTK

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 15 21:29:30 2017

@author: anoop
"""
from nltk.tokenize import sent_tokenize, word_tokenize, RegexpTokenizer

data="artificial intelligence is my favorite subject. I am also love machine lea

token1=sent_tokenize(data)
token2=word_tokenize(data)

for word in token2:
    print word

for sent in token1:
    print sent
```

*Refer Program: tokeniz.py*

# Outline

- ☒ Introduction NLTK
- ☒ Tokenization
- ☐ Stop words
- ☐ Stemming
- ☐ Lemmatization



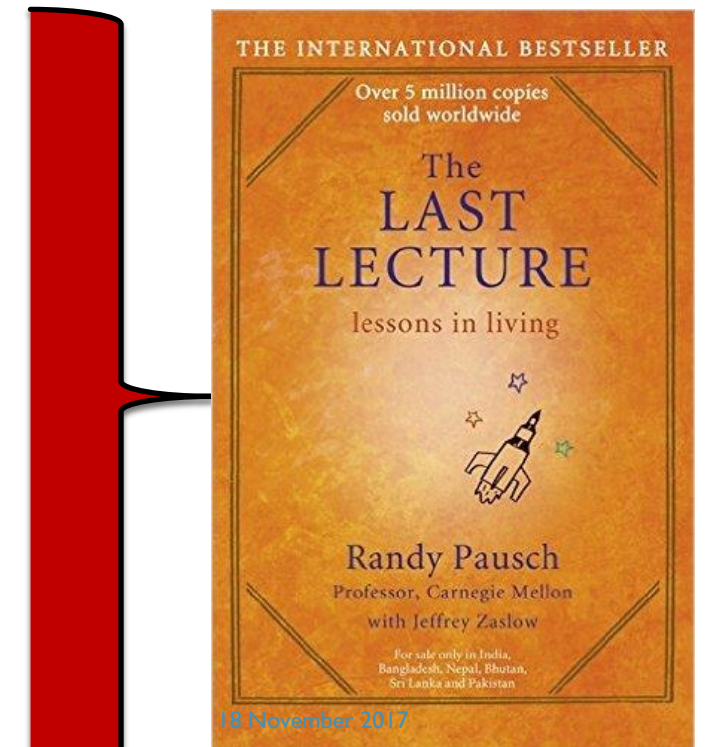
# Stop words

- ✓ Some words, which are very frequent and do not carry meaning is called stop words

**Example:** the, to, an...

**What a** great book! Simple **and** splendid writing.

- ❑ Awesome book, you will approach your life differently after reading **this...a** must read book **in a** life time
- ❑ **This was a** great read! I think my favorite quote from Randy **was** "We cannot change **the** cards **we are** dealt, just how we play **the** hand." --Randy Pausch **That** my friends **is so** powerful **so if** your **in** sales **or** any kind **of** people business you should **get this** book!



# Stop words

- ✓ Stop words are filtered out before or after processing of natural language data [1]
- ✓ Any group of words can be chosen as the stop words for a given purpose.
- ✓ For some search engines, these are some of the most common, short function words, such as the, is, at, which, and on.
- ✓ we can remove some of the most common words(stop words)in order to improve performance.



# Stop words using NLTK

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 15 23:21:45 2017

@author: anoop
"""

import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

data="artificial intelligence is my favorite subject. I am also love machine lea
words=word_tokenize(data)

#defining stopwords
stop_words=set(stopwords.words("english"))

stop_words.add("is")

#filtering
filterData=[]
for w in words:
    if w not in stop_words:
        filterData.append(w)
```

*Refer Program: stopwordRemov.py*

# Outline

- ☒ Introduction NLTK
- ☒ Tokenization
- ☒ Stop words
- ☐ Stemming
- ☐ Lemmatization



# Stemming

For a Query processing system:

Frequently, the user specifies a word in a query but only a **variant** of this word is present in a relevant document

Query

car



Document

car, cars,  
car's, cars'

plurals, past tense suffixes, etc.

# Stemming

The syntactical variations prevents a **perfect match** between a query word and a respective document word.

This problem can be partially overcome with the substitution of the words by their stems.

A **stem** is the portion of a word which is left after the removal of its affixes (i.e., prefixes and suffixes).

**Connect** is the stem for its variants:  
**connected, connecting, connection, and connections**

# Stemming using NLTK

PorterStemmer, SnowballStemmer

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 15 23:36:56 2017

@author: anoop
"""

from nltk.stem import PorterStemmer, SnowballStemmer
from nltk.tokenize import word_tokenize

#example_words = ["python", "pythoner", "pythoning", "pythoned", "pythonly"]
word="cacti"

ps=PorterStemmer()
stemmedData1=ps.stem(word)

ss=SnowballStemmer("english", ignore_stopwords=True)
stemmedData2=ss.stem(word)
```

*Refer Program: stemmer.py*

# Outline

- ☒ Introduction NLTK
- ☒ Tokenization
- ☒ Stop Word Removal
- ☒ Stemming
- ☐ Lemmatization



# Lemmatization

*Lemmatization* usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*.

## Example #1:

If confronted with the token **saw**, stemming might return just **s**, whereas lemmatization would attempt to return either **see** or **saw** depending on whether the use of the token was as a verb or a noun.

Lemmatization = morphological analysis of a word that returns its *lemma*

This is the form in which a word appears in the dictionary

# Lemmatization using NLTK

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 15 23:47:49 2017

@author: anoop

Note:cacti:a succulent plant with a thick fleshy stem which typically bears spin
"""
from nltk.tokenize import sent_tokenize, word_tokenize, RegexpTokenizer
from nltk.stem import WordNetLemmatizer

#example_words = ["python", "pythoner", "pythoning", "pythoned", "pythononly"]
word="pythoner"
lemmatizer = WordNetLemmatizer()

lemmatizedWord=lemmatizer.lemmatize(word)

print(lemmatizer.lemmatize("cats"))
print(lemmatizer.lemmatize("cacti"))
print(lemmatizer.lemmatize("geese"))
print(lemmatizer.lemmatize("rocks"))
print(lemmatizer.lemmatize("python"))
print(lemmatizer.lemmatize("better", pos="a"))
print(lemmatizer.lemmatize("best", pos="a"))
print(lemmatizer.lemmatize("run"))
print(lemmatizer.lemmatize("run", 'v'))
```



# Visualization

- **Matplotlib**
- **seaborn**

<https://matplotlib.org/>

<https://seaborn.pydata.org/index.html>



# Matplotlib – for Data Visualization



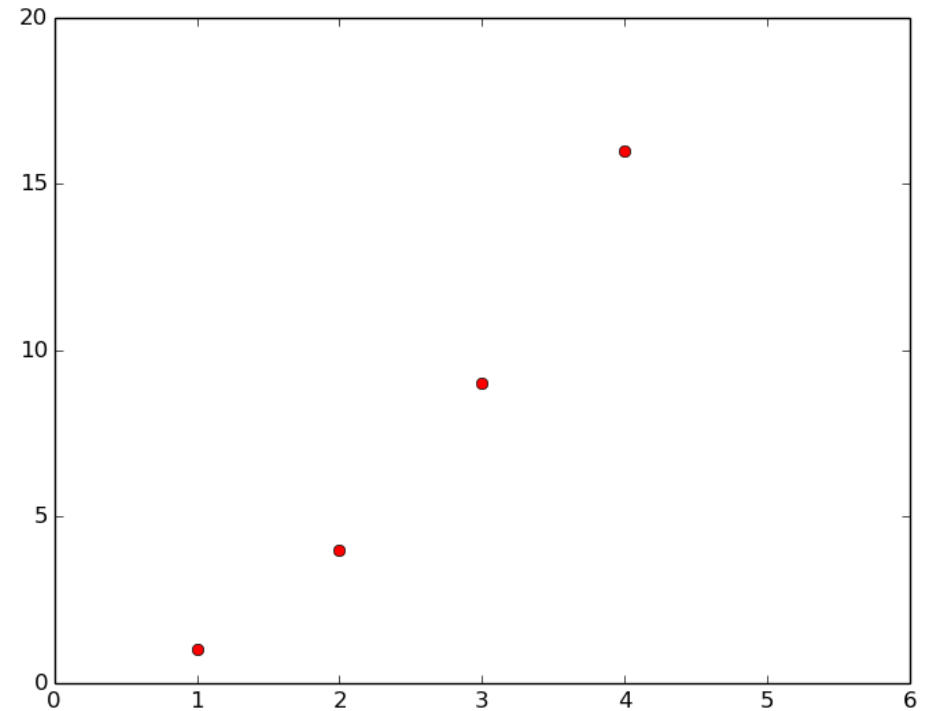
**Matplotlib:** Python based plotting library offers matplotlib with a **complete 2D support along with limited 3D graphic support**. It is useful in producing publication quality figures in interactive environment across platforms. **It can also be used for animations as well**. It provides both a very quick way to visualize data from Python and publication-quality figures in many formats.

## **Installing the module Matplotlib in ubuntu**

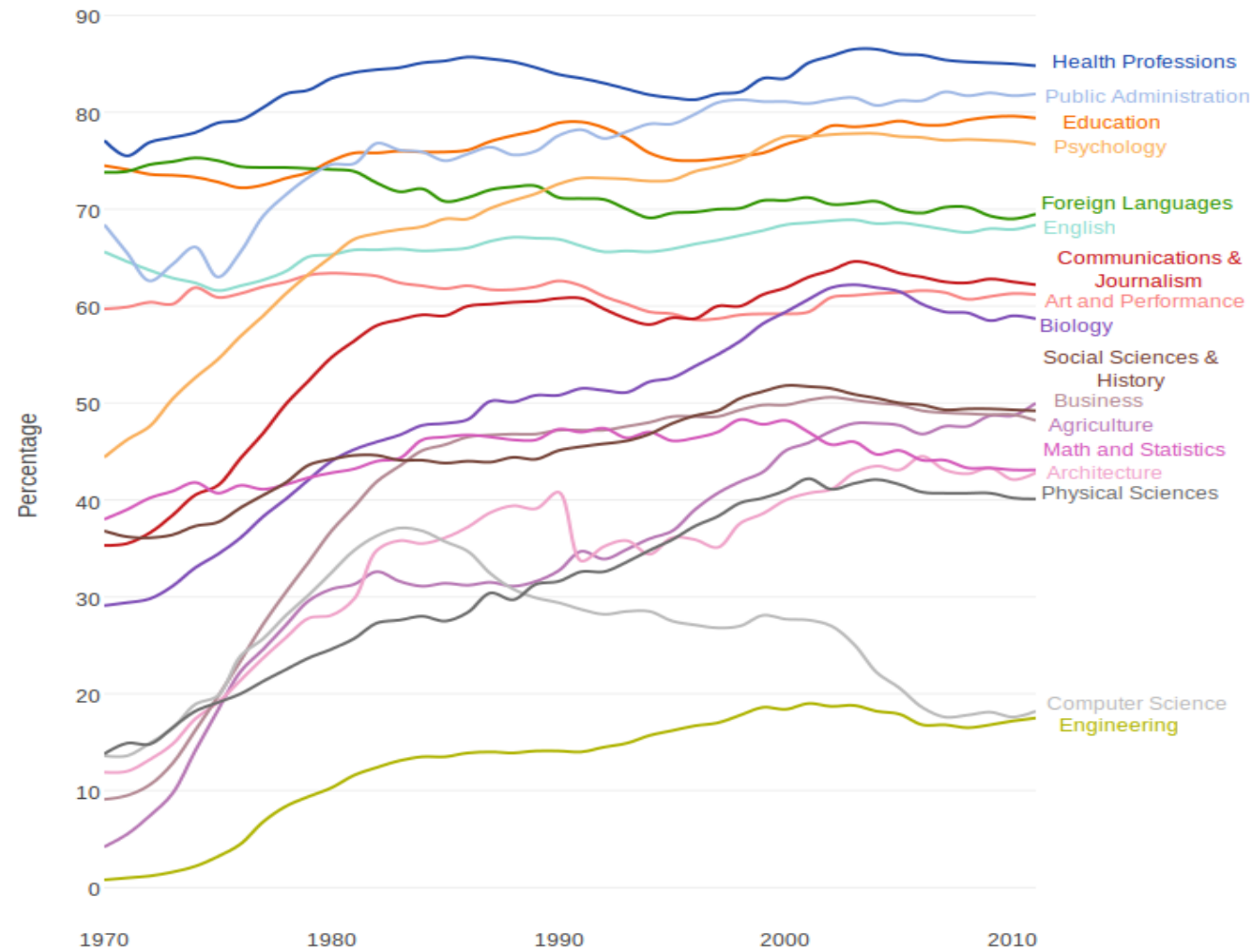
```
sudo apt-get install python-matplotlib  
or if you prefer pip or easy_install,  
pip install matplotlib  
or  
easy_install matplotlib
```

## Example #1:

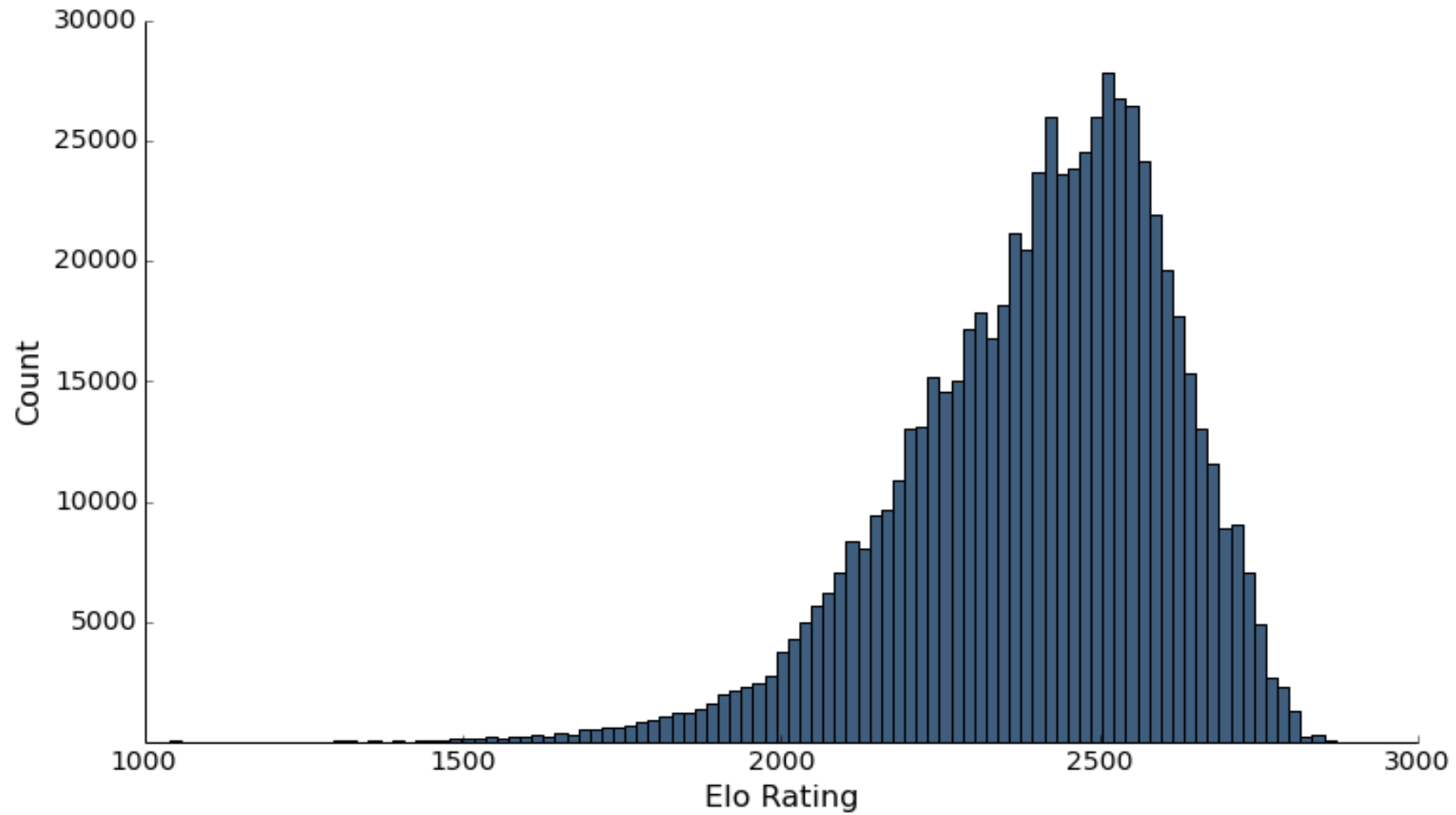
```
import matplotlib.pyplot as plt  
plt.plot([1,2,3,4], [1,4,9,16], 'ro') #g^  
another style  
plt.axis([0, 6, 0, 20])  
plt.show()
```



Percentage of Bachelors Degrees Conferred to Women  
in the U.S.A. by Major (1970-2012)

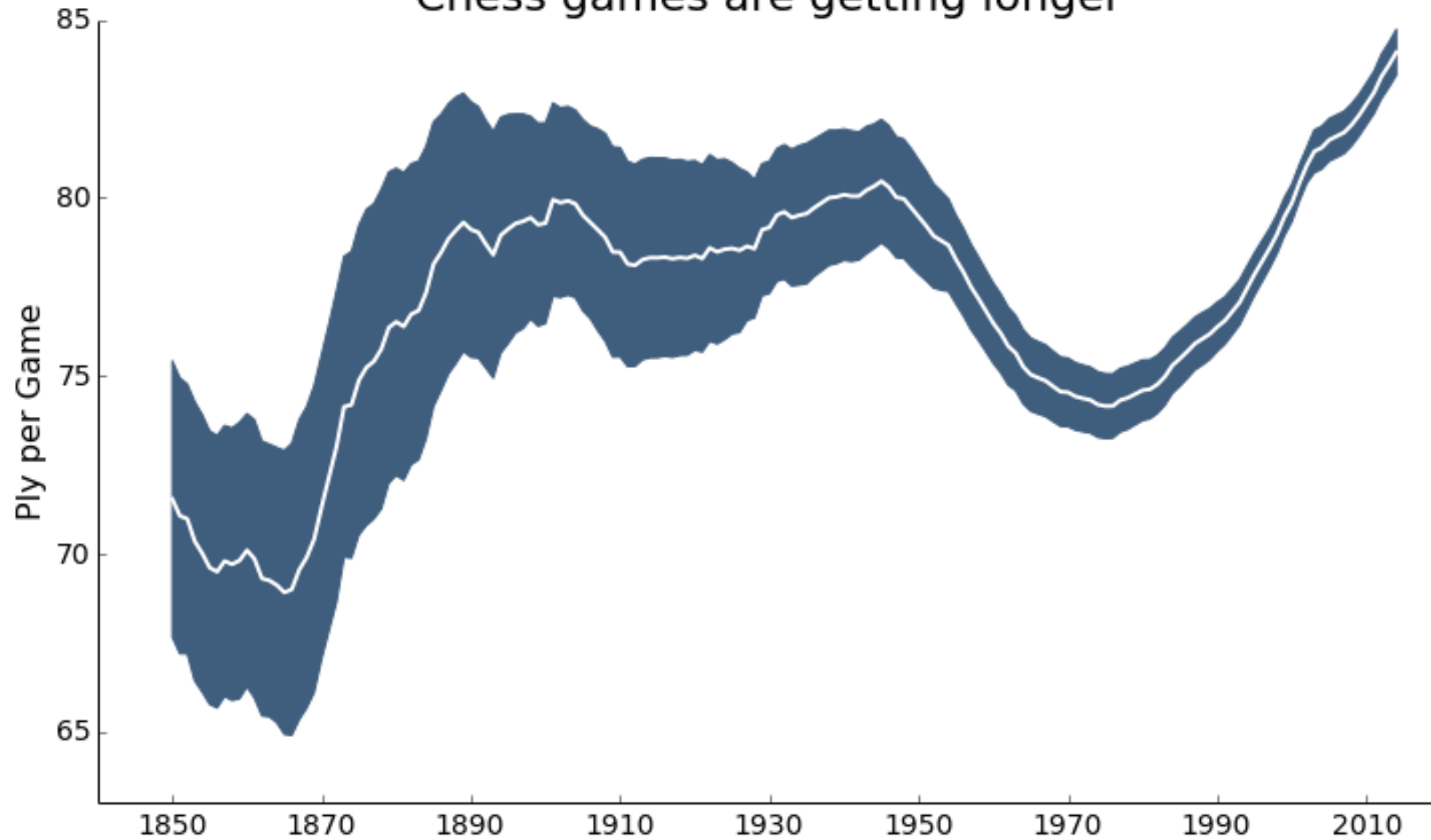


Source & Data: Randal S. Olson  
 randalolson.com / @randal\_olson  
 Some majors missing due to unavailable data



Data source: [www.ChessGames.com](http://www.ChessGames.com) | Author: Randy Olson ([randalolson.com](http://randalolson.com) / [@randal\\_olson](https://twitter.com/randal_olson))

## Chess games are getting longer



Data source: [www.ChessGames.com](http://www.ChessGames.com) | Author: Randy Olson ([randalolson.com](http://randalolson.com) / @randal\_olson)

# References

- [1] <http://www.tutorialspoint.com/python>
- [2] <http://www.programiz.com/python-programming/input-output-import>
- [3] <https://pythonprogramming.net/>
- [4] <http://www.programiz.com/python-programming/set>
- [5] <http://www.python-course.eu/numpy.php>
- [6] <http://cs231n.github.io/python-numpy-tutorial/#numpy>
- [7] <http://www.randalolson.com/2014/06/28/how-to-make-beautiful-data-visualizations-in-python-with-matplotlib/>

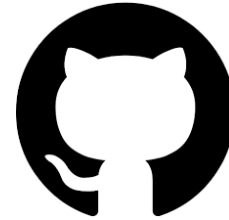




# Resources and Materials

**Github:**

<https://github.com/anoopkdc/exploringPython>



**YouTube:**

<https://www.youtube.com/channel/UCQnmg6-VdlygjVo9N2tPrkg>



**Manjary P. Gangan**

Ph.D. Research Scholar

Digital Image Forensics

Department of Computer Science

University of Calicut, Kerala – 673635

Email: manjaryp\_dcs@uoc.ac.in

Web: <http://dcs.uoc.ac.in/~manjary>

**Anoop K.**

Ph.D. Research Scholar

Machine Learning & Big Data Analytics

Department of Computer Science

University of Calicut, Kerala – 673635

Email: anoopk\_dcs@uoc.ac.in

Web: <http://dcs.uoc.ac.in/~anoop>

---

*The best preparation for tomorrow is doing your best today*

thank you!

Have a Great Day