Hands-on Workshop : Python Programming

SSN ACM Student Chapter

Agenda for the day

- Intro to Python
- Basics of Python (3 Coding Sessions)
- Lunch Break
- Python Specific Constructs and Features (2 Coding Sessions)
- After Session:
  - Google Image Search API with Python
  - Scraping “www.netlingo.com” for Web abbreviations (eg. LOL, ROFL, etc.)
  - Parser for Social Network Text data

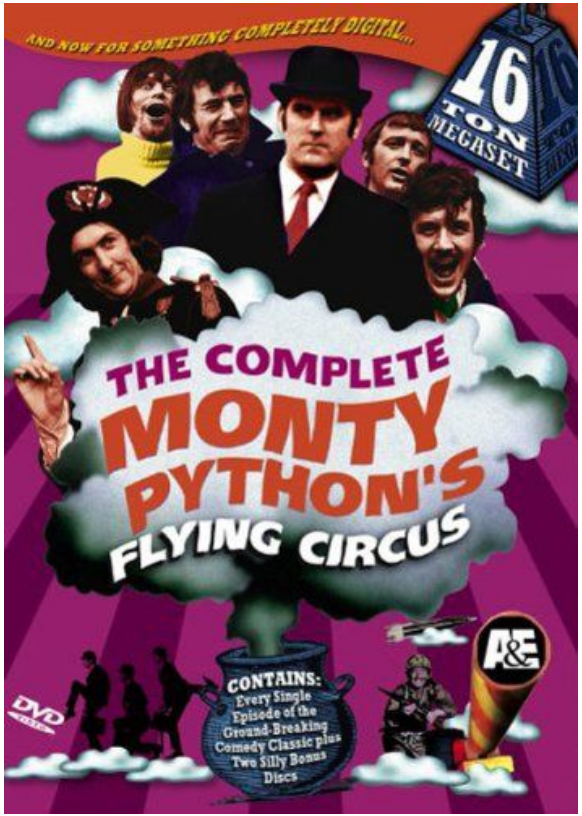
What motivated me ?

Formal Intro

- ➔ Created by **Guido Van Rossum.**
- ➔ Extremely Simple + Intuitive + Minimalistic + Readable + Expressive => **Very Pythonic** !!
- ➔ Interpreted Language
- ➔ Multi-Paradigm programming language
  - Object Oriented
  - Structural
  - And Functional
- ➔ Dynamic Typing, Dynamic Name Resolution, Cycle Detecting Garbage Collector

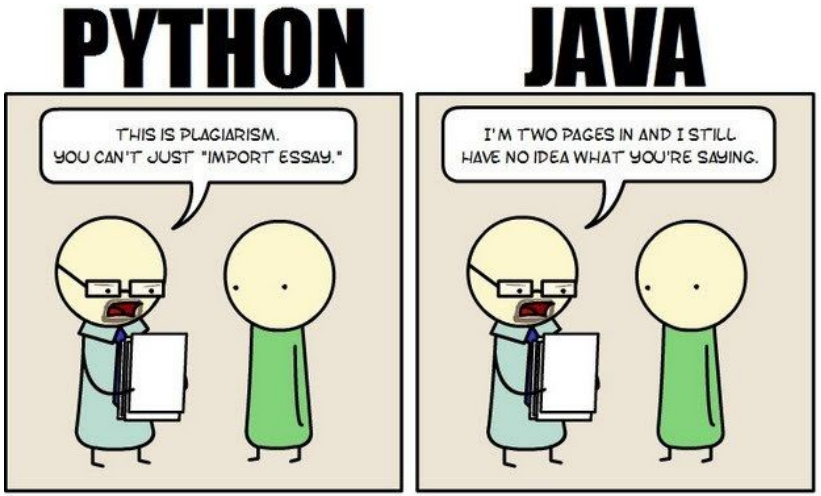
Guido Van Rossum
Worked in,
Google - 2005 - 2012
DropBox - 2013 - Now

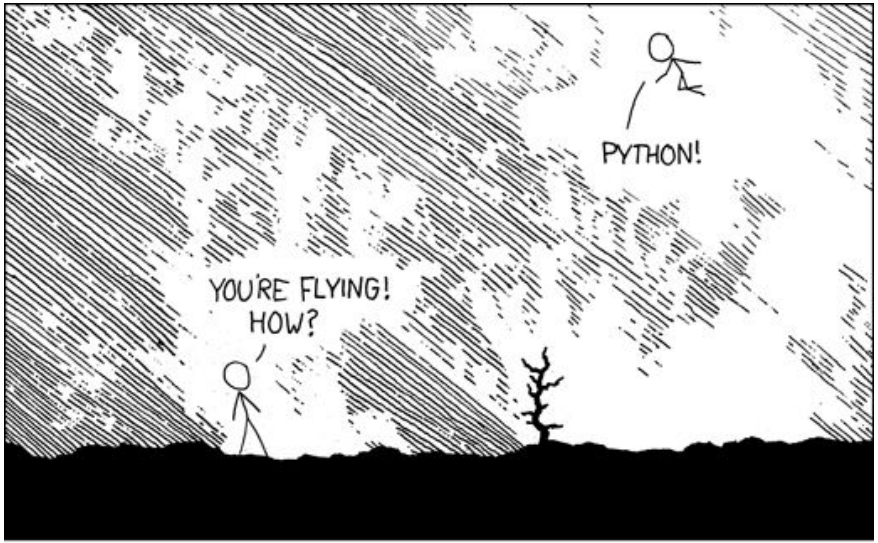
Why the name "Python"?



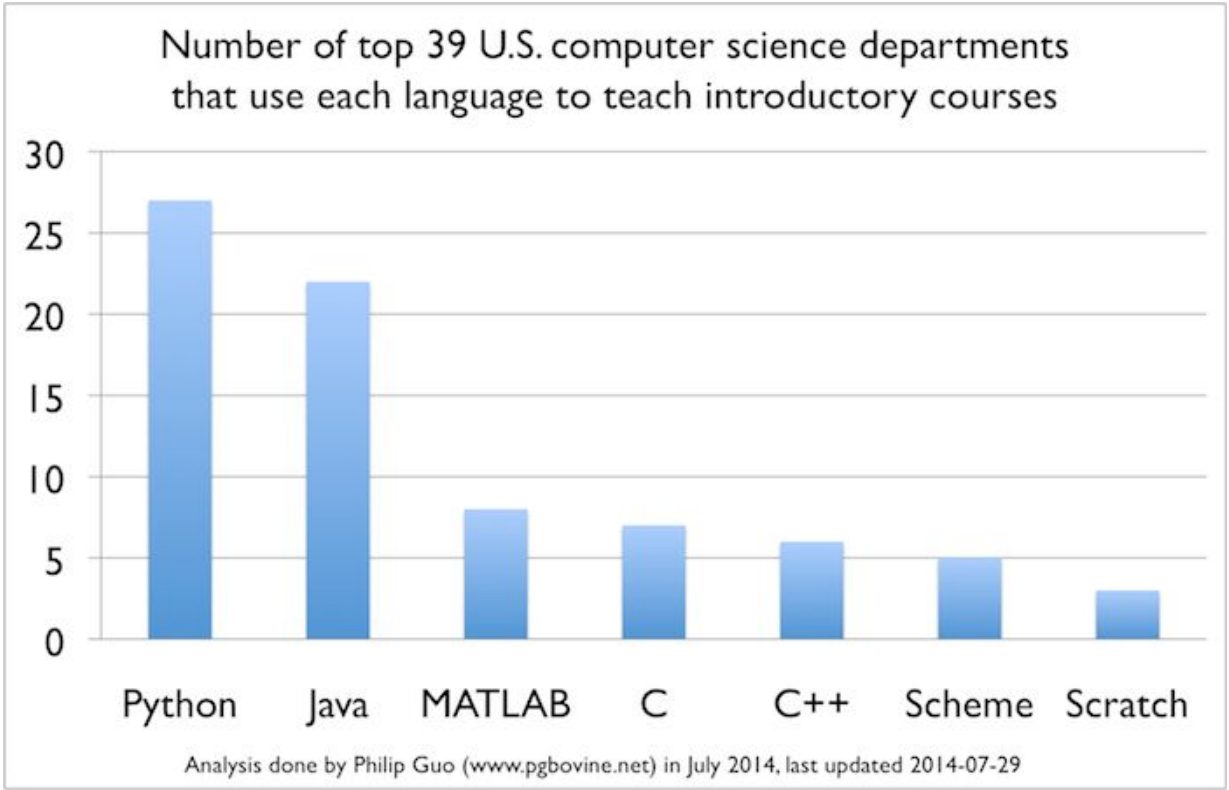
Why Learn Python?

Python Is Easy!



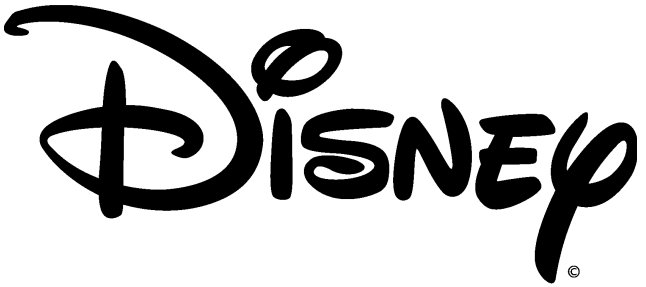


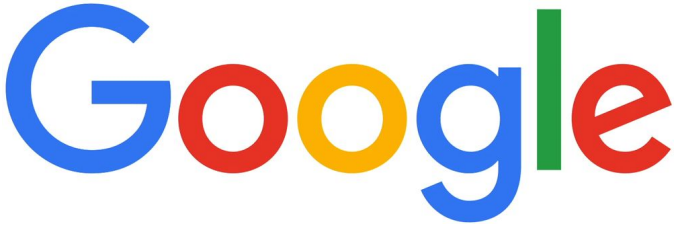
Python is the "first" language!



Source: Communications of the ACM

Who uses Python?





Python Is Efficient!

```
def read_in_chunks(file_object, chunk_size=1024):
    """Lazy function (generator) to read a file piece by piece.
    Default chunk size: 1k."""
    while True:
        data = file_object.read(chunk_size)
        if not data:
            break
        yield data


f = open('really_big_file.dat')
for piece in read_in_chunks(f):
    process_data(piece)

```

Generators in C!

```
def pow2s():
    yield 1
    for i in map((lambda x: 2 * x), pow2s()):
        yield i


def mymap(func, iteration):
    for i in iteration:
        yield func(i)

```

```c
#include <stdio.h>
#include <stdlib.h>

struct gen { // generic structure, the base of all generators
  int (*next)() ;
  int continue_from ;
} ;

typedef int (*fptr)() ;

struct mapgen { // structure for map
  int (*next)() ;
  int continue_from ; // not really required, provided for compatibility
  fptr f ;
  struct gen *g ;
} ;

int map_next(struct mapgen *p) { // next function for map
  return p->f(p->g->next(p->g)) ;
}

struct gen *map(fptr f, struct gen *g) { // constructor for map iterator
  struct mapgen *p = (struct mapgen *)malloc(sizeof(struct mapgen));
  p->next = map_next;
  p->continue_from = 0;
  p->f = f;
  p->g = g;
  return (struct gen *)p ;
}

struct pow2s { // structure
  int (*next)() ;
  int continue_from ;
  struct gen *g ;
};

int times2(int x) { // anonymous lambda is translated into this
  return 2*x ;
}

struct gen *pow2() ; // forward declaration of constructor

int pow2next(struct pow2s * p){ // next function for iterator
  switch(p->continue_from) {
  case 0:
    p->continue_from = 1;
    return 1;
  case 1:
    p->g = map(times2,pow2()) ;
    p->continue_from = 2;
    return p->g->next(p->g) ;
  case 2:
    p->continue_from = 2;
    return p->g->next(p->g) ;
  }
}

struct gen * pow2() { // constructor for pow2
  struct pow2s * p = (struct pow2s *)malloc(sizeof(struct pow2s));
  p->next = pow2next;
  p->continue_from = 0;
  return (struct gen *)p;
}

int main() {
  int i ;
  struct gen * p = pow2() ;
  for(i=0;i<10;i++)
    printf("%d ",p->next(p)) ;
  printf("\n");
}
```

Python is "fast" !

Native Python

```
import numpy as np

def pairwise_python(X, D):
    M = X.shape[0]
    N = X.shape[1]
    for i in range(M):
        for j in range(M):
            d = 0.0
            for k in range(N):
                tmp = X[i, k] - X[j, k]
                d += tmp * tmp
            D[i, j] = np.sqrt(d)
```

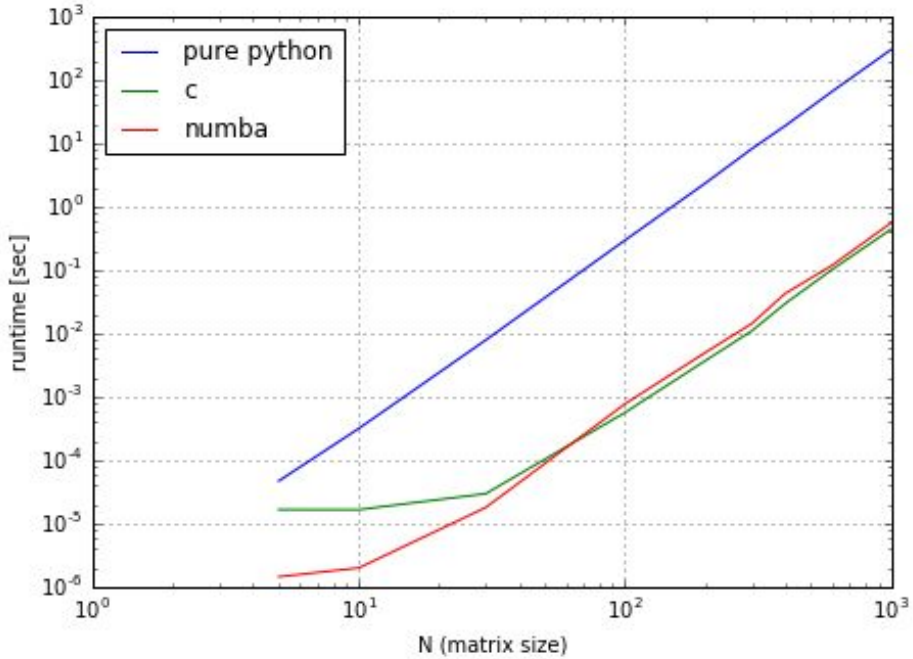
```
In [2]: import numpy as np
In [3]: X = np.random.random((1000, 3))
In [4]: D = np.empty((1000, 1000))
In [5]: %timeit pairwise_python(X, D)
1 loops, best of 3: 12.1 s per loop
```

"Numba" Python

```
import numpy as np
from numba import double
from numba.decorators import jit

@jit(arg_types=[double[:,:], double[:,:]])
def pairwise_numba(X, D):
    M = X.shape[0]
    N = X.shape[1]
    for i in range(M):
        for j in range(M):
            d = 0.0
            for k in range(N):
                tmp = X[i, k] - X[j, k]
                d += tmp * tmp
            D[i, j] = np.sqrt(d)
```

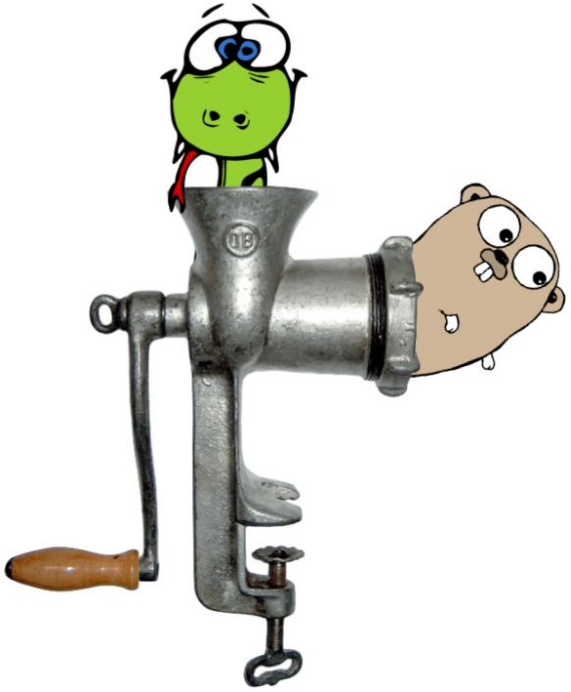
```
In [2]: import numpy as np
In [3]: X = np.random.random((1000, 3))
In [4]: D = np.empty((1000, 1000))
In [5]: %timeit pairwise_numba(X, D)
100 loops, best of 3: 15.5 ms per loop
```

Benchmarking



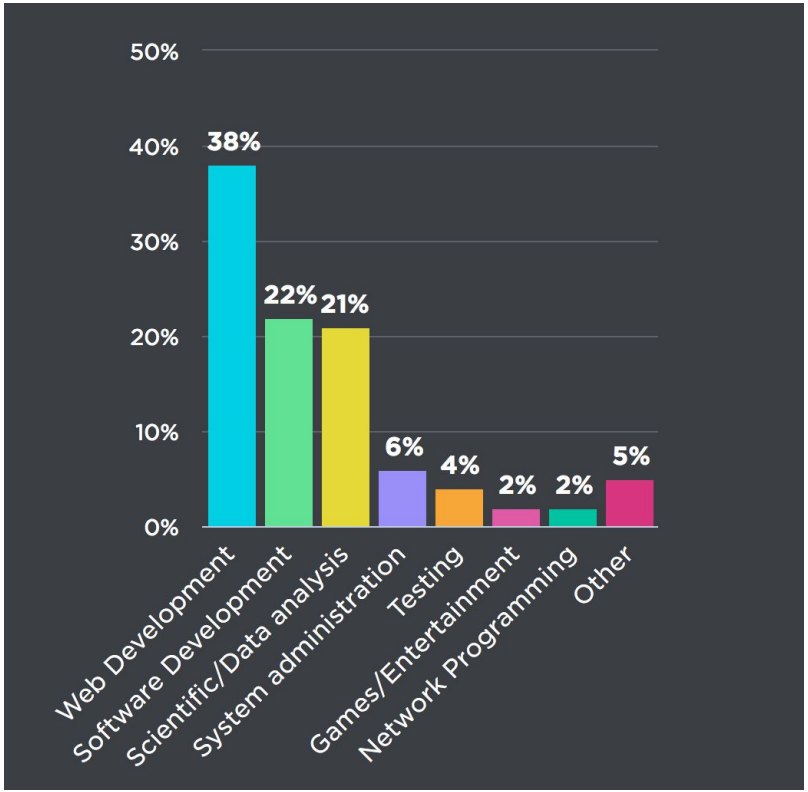
Source: IBM developerWorks Blog

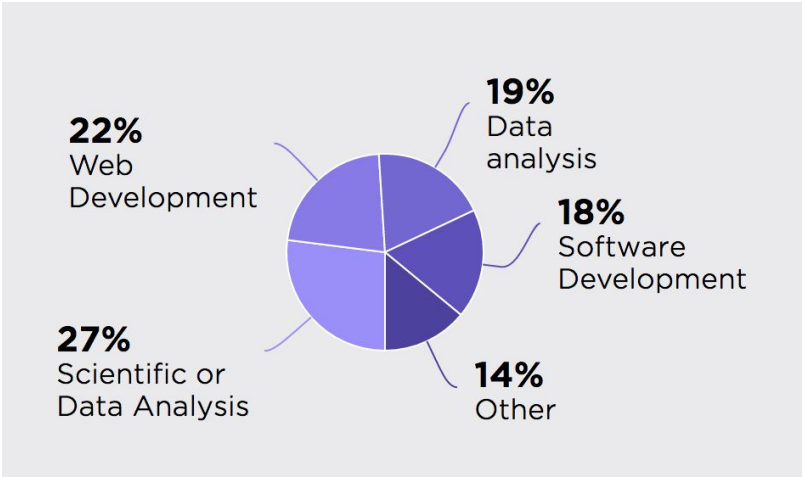
Python Is Not Python Anymore!



IronPython

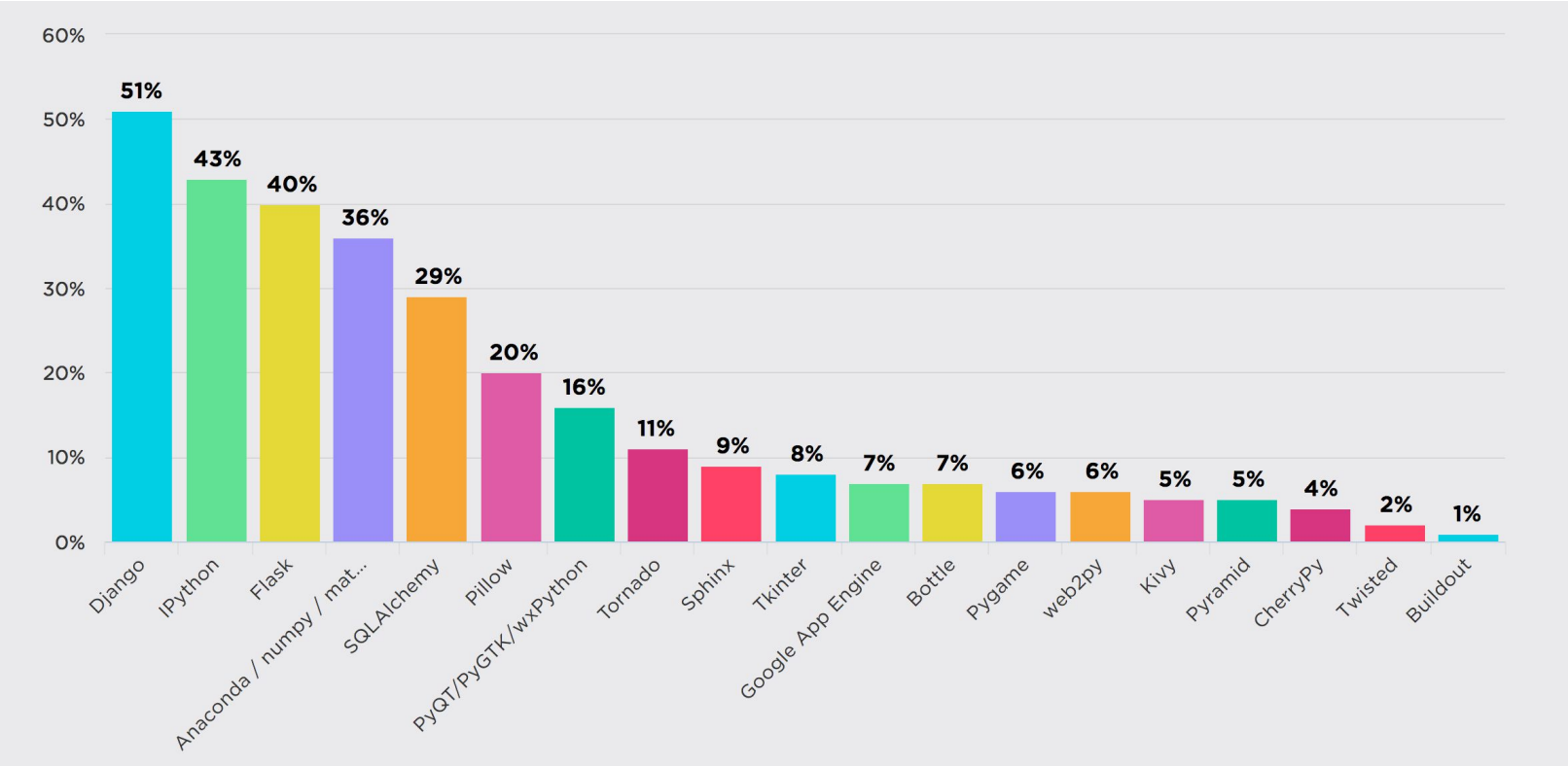
Python is Everywhere!





Source: PyCharm

Python is Open Source!



Source: PyCharm

Are You All Set?

- Python Environment Setup
  - Refer to the **“Setup.txt”** file in the **“Py Environment Setup”** directory.
  - **Anaconda**: It is a completely free Python distribution tha 195+ of the most popular Python packages for science, mat engineering, data analysis.
- If, incase you have problems with setting up, try the **“Alternate”** in **“Py Environment Setup”** folder.
- All the **‘Time to Code …!’** slides have supporting files in the “**./Py Script Files/**” folder.

Hello World!

Java

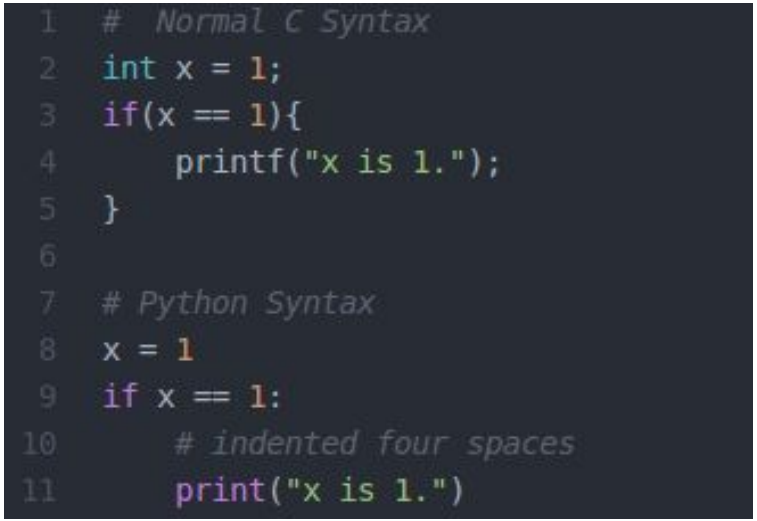
```java
public class Main {
  public static void main(String[] args) {
     System.out.println("hello world");
   }
}
```

Python

```python
print "hello world";
```

Indentation ...

- ➔ Python uses tabs/spaces for representing blocks rather than curly braces or keywords - off-side rule(Standard is - four spaces).
- ➔ Codestyling Paradigms: PEP-8, Flake-8, etc.



```c
#  Normal C Syntax
int x = 1;
if(x == 1){
    printf("x is 1.");
}

# Python Syntax
x = 1
if x == 1:
    # indented four spaces
    print("x is 1.")
```

Basics of Python

- ➔ Variables and Data types
- ➔ Data Structures
- ➔ Operators
- ➔ Conditions
- ➔ Loops
- ➔ Functions
- ➔ Classes and Objects
- ➔ Modules and Packages

Variables and Data Types

- Variables are,
  - Objects
  - Dynamically Typed - No need to declare variables or their types prior to use.
- Basic Types
  - Numbers
    - Integers
    - Floating Point
    - Complex
  - Strings

Strings

```
s = 'hi'
print s[1]          ## i
print len(s)        ## 2
print s + ' there'  ## hi there

pi = 3.14
##text = 'The value of pi is ' + pi      ## NO, does not work
text = 'The value of pi is '  + str(pi)  ## yes

raw = r'this\t\n and that'
print raw     ## this\t\n and that

multi = """It was the best of times.
It was the worst of times."""
```

String Operations

- `s.lower(), s.upper()`
- `s.strip()`
- `s.isalpha()/s.isdigit()/s.isspace()`
- `s.startswith('other')/s.endswith('other')`
- `s.find('other')`
- `s.replace('old', 'new')`
- `s.split('delim')`
- `s.join(list)`

```
>>> "\n Harry Potter ".strip()
'Harry Potter'
>>>
>>> "Harry Potter".lower()
'harry potter'
>>> "Harry Potter".upper()
'HARRY POTTER'
>>>
>>>
>>> "abc".isalpha()
True
>>> "abc ".isalpha()
False
>>>
>>>
>>> "123".isdigit()
True
>>> "123a".isdigit()
False
>>>
>>>
>>> " \t\n".isspace()
True
>>> " 1\t\n".isspace()
False
```

```
>>>
>>> "Star Wars".startswith("S")
True
>>> "Star Wars".startswith("s")
False
>>>
>>>
>>> "Star Wars".find("Wars")
5
>>> "Star Wars".find("wars")
-1
>>>
>>>
>>> "Star Wars".replace("Wars", "Battle")
'Star Battle'
>>>
>>>
>>> "Star Wars".split(" ")
['Star', 'Wars']
>>>
>>>
>>> "||".join(["Monty", "Python's", "Circus"])
"Monty||Python's||Circus"
```

String Slices


```
Python 3.4.3 |Anaconda 2.2.0 (64-bit)| (default, Jun  4 2015, 15:29:08)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> str = "Hello SSN!"
>>> str[0]
'H'
>>> str[1:3]
'el'
>>> str[-1]
'!'
>>> str[::-1]
'!NSS olleH'
>>> str[-5:]
' SSN!'
```

String Formatting

```
# % Operator
# This prints out "Hello, John!"
name = "John"
print ("Hello, %s!" % name)

# This prints out "3 little pigs come out or I'll huff and puff and blow down"
text = "%d little pigs come out or I'll %s and %s and %s" % (3, 'huff', 'puff', 'blow down')
print (text)
```

```
# This prints out: 'First Value-Second Value'
'{0}-{1}'.format('First Value', 'Second Value')

# This prints out: 'My name is Chris. I like Machine Learning'
'My name is {name}. I like {interest}'.format(name='Chris', interest="Machine Learning")
```

String Encoding

```
10  # Encoding
11  >> ustring = u'A unicode \u018e string \xf1'
12  ## (ustring from above contains a unicode string)
13  >> s = ustring.encode('utf-8')
14  >> 'A unicode \xc6\x8e string \xc3\xb1'
15  >> print(s)                              ## bytes of utf-8 encoding
16  >> t = unicode(s, 'utf-8')               ## Convert bytes back to a unicode string
17  >> t == ustring
18  >> True
```

ISO-8859-1 is graphic character set that is a superset of “UTF-8”

Data Structures

List	Mutable, Mixed Types
Dictionary	Mutable, Mixed Types (key and value)
Set	Mutable, Hashable, Unordered, Mixed Types
Tuple	Mutable, Mixed Types

Python Lists

- Lists are similar to arrays but with mixed-type support.
- They have the *len()* function and *[]* to access data, similar to strings.

```
>> colors = ['red', 123, 12.34]
>> print colors[0]
>> 'red'
>> print colors[2]
>> 12.34
>> print len(colors)
>> 3
>> A = [2, 3]
>> B = [4, 5]
>> A + B # append
>> [2, 3, 4, 5]
```

"FOR" AND "IN" IN LISTS

```
>>> squares = [1, 4, 9, 16]
>>> sum = 0
>>> for num in squares:
...     sum += num
...
>>> print (sum)
30
>>> list = ['larry', 'curly', 'moe']
>>> if 'curly' in list:
...    print("Found !")
...
Found !
```

List Methods

- list.append(elem)
- list.extend(list2)
- list.index(elem)
- list.remove(elem)
- list.sort()
- list.reverse()
- list.pop(index)
- list.insert(index, elem)

```
>>> list = ['larry', 'curly', 'moe']
>>> list.append('shemp')          ## append elem at end
>>> list.insert(0, 'xxx')         ## insert elem at index 0
>>> list.extend(['yyy', 'zzz'])   ## add list of elems at end
>>> print (list)
['xxx', 'larry', 'curly', 'moe', 'shemp', 'yyy', 'zzz']
>>> print (list.index('curly'))
2
>>> list.remove('curly')          ## search and remove that element
>>> list.pop(1)                   ## removes and returns 'larry'
'larry'
>>> print (list)
['xxx', 'moe', 'shemp', 'yyy', 'zzz']
>>>
```

List Slicing works the same as Strings.

```
>>> a = ["one", "two", "three"]
>>> a.append("four")
>>> a
['one', 'two', 'three', 'four']
>>>
>>>
>>> a = ["one", "two", "three"]
>>> a.append(["four", "five"])
>>> a
['one', 'two', 'three', ['four', 'five']]
>>>
>>>
>>> a = ["one", "two", "three"]
>>> a.extend(["four", "five"])
>>> a
['one', 'two', 'three', 'four', 'five']
>>>
>>>
>>> a = ["one", "two", "three"]
>>> print(a.index("two"))
1
>>>
>>>
```

```
>>> a = ["one", "two", "three"]
>>> a.remove("one")
>>> a
['two', 'three']
>>>
>>>
>>> a = [5, -1, 10]
>>> a.sort()
>>> a
[-1, 5, 10]
>>>
>>>
>>> a = ["one", "two", "three"]
>>> a.reverse()
>>> a
['three', 'two', 'one']
>>>
>>>
>>> a = ["one", "two", "three"]
>>> a.pop(2)
'three'
>>>
>>>
>>> a = ["one", "two", "three"]
>>> a.insert(1, "four")
>>> a
['one', 'four', 'two', 'three']
```

Time to code ...!

- **Edit Words**: Given a word W, perform the following edit operations and output a single unique list of edited words.
- Eg. “pokemon”
  - **Splits** - ‘p’, ‘okemon’; ‘po’, ‘kemon’; ‘pok’, ‘emon’; etc.
  - **Deletes** - ‘pkemon’, ‘poemon’, ‘pokmon’, etc.
  - **Transposes** - ‘pkoemon’, ‘poekmon’, ‘pokmeon’, etc.
  - **Replaces** - ‘pakemon’, ‘pobemon’, ‘pokcmon’, etc.
  - **Inserts** - ‘paokemon’, ‘pobkemon’, ‘pokcemon’, etc.

```
>>>
>>> input_word = "pokemon"
>>> output_word_list = list()
>>> for i in range(1, len(input_word)):
...     output_word_list.extend([input_word[:i], input_word[i:]])
...
>>> print(output_word_list)
['p', 'okemon', 'po', 'kemon', 'pok', 'emon', 'poke', 'mon', 'pokem', 'on', 'pokemo', 'n']
>>>
>>>
```

Python Dictionaries

- Efficient key/value hash tables
- {} are used to denote dict

```
## Can build up a dict by starting with the the empty dict {}
## dict[key] = value-for-that-key
dict = {}
dict['a'] = 'alpha'
dict['g'] = 'gamma'
dict['o'] = 'omega'
# (or)
dict = {'a': 'alpha', 'g': 'gamma', 'o': 'omega'}
print dict  ## {'a': 'alpha', 'o': 'omega', 'g': 'gamma'}

print dict['a']     ## Simple lookup, returns 'alpha'
dict['a'] = 6       ## Put new key/value into dict
'a' in dict         ## True
## print dict['z']                  ## Throws KeyError
if 'z' in dict: print dict['z']     ## Avoid KeyError
print dict.get('z')  ## None (instead of KeyError)
```

Continued...

```
## By default, iterating over a dict iterates over its keys.
## Note that the keys are in a random order.
for key in dict: print (key) # prints a g o
## Exactly the same as above
for key in dict.keys(): print (key)

## Get the .keys() list:
print (dict.keys())  ## ['a', 'o', 'g']

## Likewise, there's a .values() list of values
print (dict.values())  ## ['alpha', 'omega', 'gamma']

## .items() is the dict expressed as (key, value) tuples
print (dict.items())  ##  [('a', 'alpha'), ('o', 'omega'), ('g', 'gamma')]

for k, v in dict.items(): print (k + '>' + v)
## a > alpha    o > omega     g > gamma
```

Mutable Vs Immutable

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Mutable Vs Immutable

```
>>> def func1(val):
...     val += 'bar'
...
>>> x = 'foo'
>>> x
'foo'
>>> func1(x)
>>> x
'foo'
>>>
>>>
>>> def func2(val):
...     val += [3, 2, 1]
...
>>> x = [1, 2, 3]
>>> x
[1, 2, 3]
>>> func2(x)
>>> x
[1, 2, 3, 3, 2, 1]
```

Operators

```
# Just as any other programming languages, the addition, subtraction, multiplication,
# and division operators can be used with numbers.
>> 1 + 2 * 3 / 4.0
2.5
>> 11 % 3 # remainder
>> 2
# Using two multiplication symbols makes a power relationship.
>> 7 ** 2 # squared
>> 49
>> 2 ** 3 # cubed
>> 8
# Python supports concatenating strings using the addition operator:
>> "hello" + " " + "world"
>> 'hello world'
# Python also supports multiplying strings to form a string with a repeating sequence:
>> "hello" * 10
>> 'hellohellohellohellohellohellohellohellohellohello'
>> print ([1,2,3] * 3)
>> [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

DEL Operator

```
1  var = 6
2  del var  # var no more!
3
4  list = ['a', 'b', 'c', 'd']
5  del list[0]     ## Delete first element
6  del list[-2:]   ## Delete last two elements
7  print (list)    ## ['b']
8
9  dict = {'a':1, 'b':2, 'c':3}
10 del dict['b']   ## Delete 'b' entry
11 print (dict)    ## {'a':1, 'c':3}
```

Conditions

```
>>> x = 2
>>> x == 2
True
>>> x == 3
False
>>> x < 3
True
>>> x != 2
False
```

The "and" and "or" boolean operators allow building complex boolean expressions.

```
1  name = "John"
2  age = 23
3  if name == "John" and age == 23:
4      print ("Your name is John, and you are also 23 years old.")
5
6  if name == "John" or name == "Rick":
7      print ("Your name is either John or Rick.")
```

'Is' vs '=='

```
>>> # 'is' compares object level identity
... # '==' compares value level identity
...
>>> a = [1, 2, 3]
>>> b = a
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
>>>
>>>
>>> a is b
True
>>> a == b
True
>>>
>>>
>>> a = [1, 2, 3]
>>> b = list(a)
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
>>>
>>> a is b
False
>>> a == b
True
```

If ... Else ...

```
if <statement is true>:
    <do something>
    ....
    ....
elif <another statement is true>: # else if
    <do something else>
    ....
    ....
else:
    <do another thing>
    ....
    ....
```

Loops - For, While

```
1  primes = [2, 3, 5, 7]
2  for prime in primes:
3      print prime
4  # 2 3 5 7
5
6  for x in range(5):
7      print x
8  # 0 1 2 3 4
9
10 count = 0
11 while count < 5:
12     print count
13     count += 1  # This is the same as count = count + 1
14 # 0 1 2 3 4
```

"Break" and "Continue"

```
# Prints out 0,1,2,3,4

count = 0
while True:
    print (count)
    count += 1
    if count >= 5:
        break

# Prints out only odd numbers - 1,3,5,7,9
for x in xrange(10):
    # Check if x is even
    if x % 2 == 0:
        continue
    print (x)
```

Time to code ...!

- **Download Progress**: Open “**download-progress**”, you can find a ‘**download_1.log**’ and ‘**download_2.log**’ files in it.
- You have to read this file and sort all of the log entries based on time in ascending order. Each log entry occupies one line as is in Dict format.
- Then, you will have to look-up the ‘**progress_percentage**’ key in each of the Dict entries and verify whether the download has been smooth.
- Smooth download (1,2,3,3,10,90,100) => Output ‘True’
- Non-smooth downlaod (1,2,3,10,8,90,100) => Output ‘False’

```
{"timestamp": "2017-02-24 01:12:03.234892", "progress_percentage": 10}
{"timestamp": "2017-02-24 01:12:04.234892", "progress_percentage": 11}
```

Hints...

```
# read from file
logs = open("download_1.log").readlines()

# sorting logs based on time
sorted_logs = sorted(logs, key=lambda x: x["timestamp"])

# conveting string to variables
import ast
log_str = '''{"timestamp": "2017-02-24 01:12:03.234892", "progress_percentage": 10}'''
log_dict = ast.literal_eval(log_str)
print(log_dict["progress_percentage"])  # 10

# converting string date time datetime object
from datetime import datetime
time1 = datetime.strptime("2017-02-24 01:12:03.234892", "%Y-%m-%d %H:%M:%S.%f")
time2 = datetime.strptime("2017-02-25 01:12:03.234892", "%Y-%m-%d %H:%M:%S.%f")
print(time1 < time2)    # True
```

Functions

```
1  def my_function():
2      print ("Hello From My Function!")
3
4  def my_function_with_args(username, greeting):
5      print ("Hello, %s , From My Function!, I wish you %s" % (username, greeting))
6
7  def sum_two_numbers(a, b):
8      return a + b
9
10 # print a simple greeting
11 my_function()
12
13 #prints - "Hello, John, From My Function!, I wish you a great year!"
14 my_function_with_args("John", "a great year!")
15
16 # after this line x will hold the value 3!
17 x = sum_two_numbers(1,2)
```

Classes and Objects

```
1  class Person:
2      def __init__(self, n):
3          self.name = n
4
5      def printMessage(self):
6          print ("Hello Mr./Ms. " + self.name)
7
8  # Create an object
9  p1 = Person("John")
10 p2 = Person("Angel")
11
12 # Function call
13 p1.printMessage() # Hello Mr./Ms. John
14 p2.printMessage() # Hello Mr./Ms. Angel
```

Modules and Packages

- **Modules** are Python files with a .py extension that implements a set of functions; Collection of modules is a **Package**.
- Installation of Python Packages
  - Use ***pip install <Module-or-Package-Name>***
  - Download Source from PyPi and install using, ***python setup.py install***
- *“**import**” is the keyword used.*
- ```
  # import the library
  import urllib.request
  ```
- ```
  # use it
  urllib.request.urlopen(...)
  ```

Exploring built-in Modules

- "*dir*" and "*help*" functions are used to explore.

```
>>> from urllib import request
>>> dir(request)
['AbstractBasicAuthHandler', 'AbstractDigestAuthHandler', 'AbstractHTTPHandler',
 'BaseHandler', 'CacheFTPHandler', 'ContentTooShortError', 'DataHandler', 'FTPHa
ndler', 'FancyURLopener', 'FileHandler', 'HTTPBasicAuthHandler', 'HTTPCookiePro
essor', 'HTTPDefaultErrorHandler', 'HTTPDigestAuthHandler', 'HTTPError', 'HTTPEr
rorProcessor', 'HTTPHandler', 'HTTPPasswordMgr', 'HTTPPasswordMgrWithDefaultReal
m', 'HTTPRedirectHandler', 'HTTPSHandler', 'MAXFTPCACHE', 'OpenerDirector', 'Pro
xyBasicAuthHandler', 'ProxyDigestAuthHandler', 'ProxyHandler', 'Request', 'URLEr
ror', 'URLopener', 'UnknownHandler', '__all__', '__builtins__', '__cached__', '_
_doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '__ver
sion__', '_cut_port_re', '_ftperrors', '_have_ssl', '_localhost', '_noheaders',
'_opener', '_parse_proxy', '_proxy_bypass_macosx_sysconf', '_randombytes', '_saf
e_gethostbyname', '_thishost', '_url_tempfiles', 'addclosehook', 'addinfourl', '
base64', 'bisect', 'build_opener', 'collections', 'contextlib', 'email', 'ftpcac
he', 'ftperrors', 'ftpwrapper', 'getproxies', 'getproxies_environment', 'hashlib
', 'http', 'install_opener', 'io', 'localhost', 'noheaders', 'os', 'parse_http_l
ist', 'parse_keqv_list', 'pathname2url', 'posixpath', 'proxy_bypass', 'proxy_byp
ass_environment', 'quote', 're', 'request_host', 'socket', 'splitattr', 'splitho
st', 'splitpasswd', 'splitport', 'splitquery', 'splittag', 'splittype', 'splitus
er', 'splitvalue', 'ssl', 'sys', 'tempfile', 'thishost', 'time', 'to_bytes', 'un
quote', 'unquote_to_bytes', 'unwrap', 'url2pathname', 'urlcleanup', 'urljoin', '
urlopen', 'urlparse', 'urlretrieve', 'urlsplit', 'urlunparse', 'warnings']
>>> help(request)
```

Writing Modules and Packages

- ➔ Modules - Create a new “.py” with the Module name and all necessary functions inside and import it using the python file name.
- ➔ Packages
  - Are namespaces that contain multiple packages and Modules themselves.
  - They are simply directories with a special file named ***“__init__.py”***, this can be empty for now.

```
Folder Structure:
-> Foo
    --> bar1.py
    --> bar2.py
    --> __init__.py
-> spam.py

# inside spam.py import Foo package
>> from Foo import bar1
>> import Foo.bar2
```

Some Packages I Have USed:

- ➔ PyCaffe, Keras, etc - Deep Learning Tools
- ➔ Scikit-Learn - Machine Learning Tools
- ➔ Nltk - Natural Language Processing Tools
- ➔ Gensim - Topic Modelling Library
- ➔ Scrapy - Web Scraping
- ➔ Numpy, Scipy - Mathematical and Scientific Library
- ➔ Flask, Cherrypy - lightweight web server
- ➔ Tweepy - Twitter API
- ➔ Twokenize - Tweets Tokenizer Tool.
- ➔ Etc…

Time To Code ... !

- ➔ Open the folder **“top-10”**.
- ➔ Try to parse the contents of the **“content.txt”** file and print the top 10 highly frequent words in the file.
- ➔ Try using looping, dicts (clue: defaultdict, counter), list slicing, files io, sorting, etc.
- ➔ Hint (File IO and Sorting):

```
# file open
f = open("filename","r")  # r for Read mode
lines = f.readlines() # to get the lines in file as a list
f.close()

# sorting dict based on "value"
import operator
sorted_dict = dict(sorted(unsorted_dict.items(), key = operator.itemgetter(1)))
```

Python Specific Constructs and Features

- ➔ Generators
- ➔ List Comprehensions
- ➔ Multiple Function Arguments
- ➔ Regular Expressions
- ➔ Exception Handling
- ➔ Sets
- ➔ Serialization - JSON, Pickle
- ➔ Lambda Operator
- ➔ Decorators

Generators

Simple functions which return an iterable set of items, one at a time, in a special way.

```
import random


def lottery():

    # returns 6 numbers between 1 and 40
    for i in range(6):
        yield random.randint(1, 40)

    # returns a 7th number between 1 and 15
    yield random.randint(1, 15)

for random_number in lottery():
    print("And the next number is... %d !" % random_number)
```

List Comprehensions

Very powerful tool, which creates a new list based on another list, in a single, readable line.

```
sentence = "the quick brown fox jumps over the lazy dog"
words = sentence.split()
word_lengths = []
for word in words:
    if word != "the":
        word_lengths.append(len(word))
print(word_lengths)
# >> [5, 5, 3, 5, 4, 4, 3]
```

Much Clean and Simple Version

```
10  sentence = "the quick brown fox jumps over the lazy dog"
11  words = sentence.split()
12  word_lengths = [len(word) for word in words if word != "the"]
13  print(word_lengths)
14  # >> [5, 5, 3, 5, 4, 4, 3]
```

FUN ! - Find the N-Grams from a list of Words

```
def find_ngrams(listI, n):
    grams = []
    for i in zip(*[listI[j:] for j in range(n)]):
            grams.append(" ".join(i))
    return grams
```

Multiple Function Arguments

```
# Normal Function Call
def myfunction(first, second, third):
    # do something with the 3 variables
    print(first + second + third)

myfunction(1, 2, 3)
# >> 6

def foo(first, second, third, *theRest):
    print "First: %s" % first
    print "Second: %s" % second
    print "Third: %s" % third
    print "And all the rest... %s" % list(theRest)

foo(1, 2, 3, 4, 5, 6, 7)
foo(1, 2, 3, 4, 5)
```

```
First: 1
Second: 2
Third: 3
And all the rest... [4, 5, 6, 7]
First: 1
Second: 2
Third: 3
And all the rest... [4, 5]
```

More ...

```
def bar(first, second, third, **options):
    if options.get("action") == "sum":
        print "The sum is: %d" % (first + second + third)    # The sum is: 6

    if options.get("number") == "first":
        return first

result = bar(1, 2, 3, action="sum", number="first")
print "Result: %d" % result    # Result: 1
```

*ARGS, **KWARGS

```
>>> def foo(a, *args):
...     print(a, args)
...
>>> foo(1, 2, 3, 4)
(1, (2, 3, 4))
>>>
>>> foo(1, *[2, 3, 4])
(1, (2, 3, 4))
>>>
>>>
>>> def bar(a, **kwargs):
...     print(a, kwargs)
...
>>> bar(1, b=2, c=3, d=4)
(1, {'c': 3, 'b': 2, 'd': 4})
>>>
>>>
>>> def foo_bar(a, *args, **kwargs):
...     print(a, args, kwargs)
...
>>> foo_bar(1, 2, 3, 4, b=5, c=6, d=7)
(1, (2, 3, 4), {'c': 6, 'b': 5, 'd': 7})
```

Regular Expressions

```
1  import re
2
3  line = "Cats are smarter than dogs"
4
5  matchObj = re.match(r'(.*) are (.*?) .*', line, re.M | re.I)
6
7  if matchObj:
8      print "matchObj.group() : %s " % matchObj.group()
9      print "matchObj.group(1) : %s " % matchObj.group(1)
10     print "matchObj.group(2) : %s " % matchObj.group(2)
11 else:
12     print("No match!!")
13 # RESULT
14 # matchObj.group() : Cats are smarter than dogs
15 # matchObj.group(1) : Cats
16 # matchObj.group(2) : smarter
```

Further Reading: http://www.tutorialspoint.com/python/python_reg_expressions.htm

Exception Handling

```
1  def do_stuff_with_number(n):
2      print(n)
3
4  the_list = (1, 2, 3, 4, 5)
5
6  for i in range(7):
7      try:
8          do_stuff_with_number(the_list[i])
9      except IndexError:     # Raised when accessing a non-existing index of a list
10         do_stuff_with_number(0)
```

```
1
2
3
4
5
0
0
[Finished in 0.039s]
```

Sets

Similar to lists with no duplicate entries provided all entries must be hashable.

```
>>> print (set("my name is Eric and Eric is my name".split()))
{'my', 'and', 'name', 'is', 'Eric'}
```

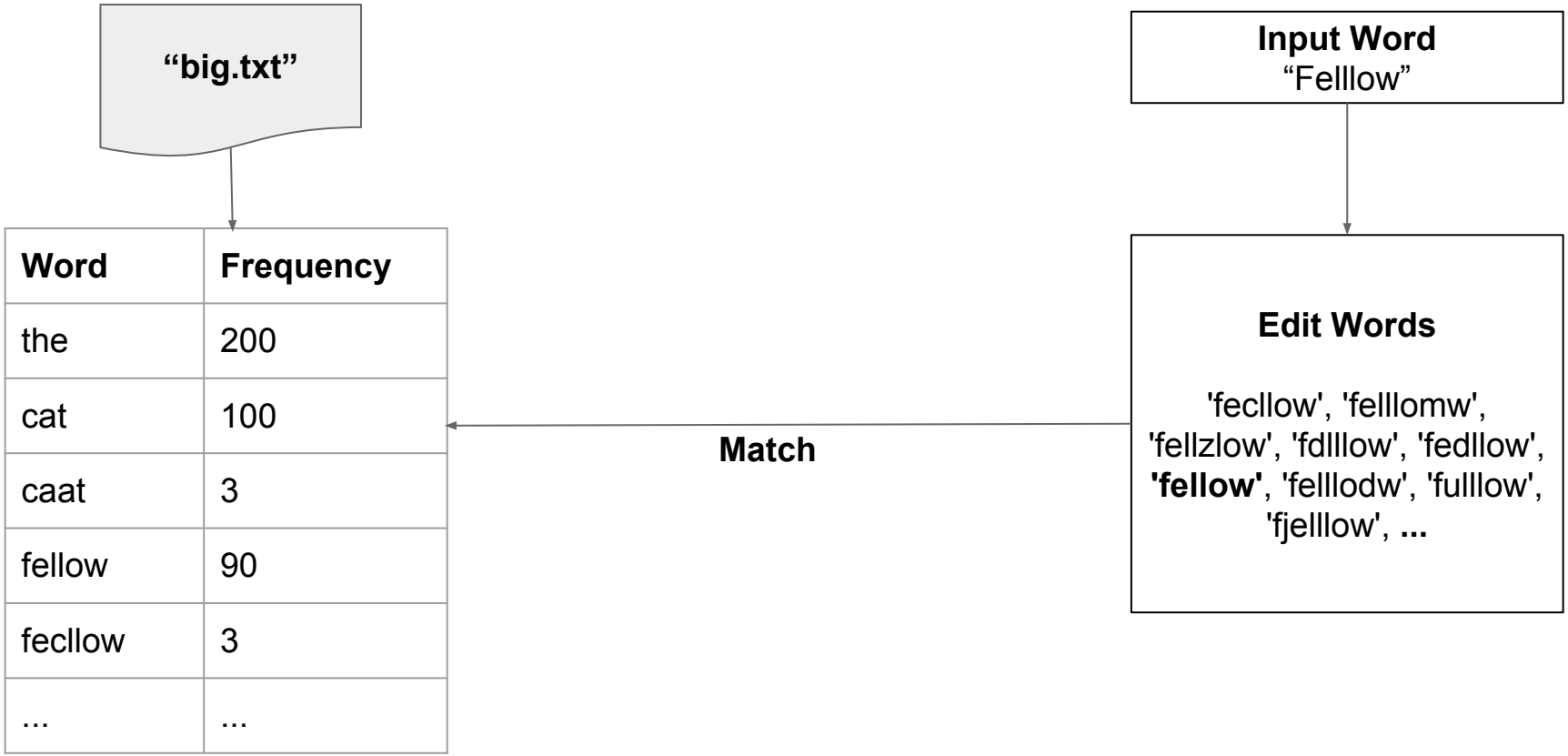
Example:

```python
# list of participants in events A and B
a = set(["Jake", "John", "Eric"])
b = set(["John", "Jill"])

# To find out which members attended both events
a.intersection(b)  # set(['John'])
b.intersection(a)  # set(['John'])

# To find out which members attended only one of the events
a.symmetric_difference(b)  # set(['Jill', 'Jake', 'Eric'])
b.symmetric_difference(a)  # set(['Jill', 'Jake', 'Eric'])

# To find out which members attended only one event and not the other
a.difference(b)  # set(['Jake', 'Eric'])
b.difference(a)  # set(['Jill'])

# To receive a list of all participants
a.union(b)  # set(['Jill', 'Jake', 'John', 'Eric'])
```

Time to Recode ...!

- **Part 1:** Try to convert all the **Edit Words** for loop constructs to use **List Comprehension** and produce a unique list with **Set.**
- **Part 2:** Go into “**spell-corrector**”,
  - Try the command `**python spell.py felllow**`, it will output `**fellow**`
  - We need to extend **Edit Words** to build this spell corrector!

How To Build A Spell Corrector ?



Serialization

```
# JSON
import json

dict = {'a': 'alpha', 'o': 'omega', 'g': 'gamma'}
json_string = json.dumps(dict)
print(json_string)

print(json.loads(json_string))

# PICKLE
import pickle
pickled_string = pickle.dumps([1, 2, 3, "a", "b", "c"])
print(pickle.loads(pickled_string))

```

Saving Objects and Data Structures - Persistence

```
15  # Saving DataStructures as .pkl binary files
16  def save_obj(obj, name):
17      with open(name + '.pkl', 'wb') as f:
18          pickle.dump(obj, f,  protocol=2)
19
20  def load_obj(name):
21      with open(name + '.pkl', 'rb') as f:
22          return pickle.load(f)
23
24  dict = {'a': 'alpha', 'o': 'omega', 'g': 'gamma'}
25  save_obj(dict, "dictSave")
```

Lambda Operator

- Anonymous Functions
- Functional Programming - Pass functions to other functions to do stuff.

```
# Using lambda
addTwo = lambda x: x+2
addTwo(2)
# 4

# the above is similar to
def addTwo(x):
    return x+2
addTwo(2)
# 4
```

Use

```
# Use:
mult3 = filter(lambda x: x % 3 == 0, [1, 2, 3, 4, 5, 6, 7, 8, 9])
print(mult3)
# [3, 6, 9]

# Equivalent to
new = []
for i in [1, 2, 3, 4, 5, 6, 7, 8, 9]:
    if i % 3 == 0:
        new.append(i)
print(new)
# [3, 6, 9]
```

Decorators

```python
from functools import wraps


def authorize(func):
    @wraps(func)
    def check_credentials(*args, **kwargs):
        # check the login credentials
        return func(*args, **kwargs)
    return check_credentials


@authorize
def apiCall(username, password):
    # do operation
    pass
```

Time to Code ... !

- ➔ Open the folder **“imdb”**.
- ➔ Try to read from all the text files in **“MovieReviews”** folder.
- ➔ Try generating a list of all possible unique bi-grams and their frequencies in numbers and save the frequency map in .pkl format for persistence.
- ➔ Code should be resumable. i.e,(If we add new files from **“subMovieReviews”**) into **“MovieReviews”** and compile again, the code should only parse the new files and update the frequencies accordingly and update the .pkl binaries rather than overwriting.

Hints

- To get list of all files in a folder
  - import os
  - os.listdir(“/path/to/folder/”)
- Bi-Grams: They are phrases with exactly 2 consecutive words, from any piece of text.
  - eg. “India is my country!”
  - **All Bi-Grams** - “India is”, “is my”, “my country!”.
- Use “sets” to find undone files list (use it with os.listdir)
- Use pickle to store and load frequency map
  - use DefaultDict from Collections

Python work at QUBE Cinema Technologies!

Web Scraping With Python

- Libraries we will be using
  - **Urllib** - open and read information from urls
  - **Beautifulsoup** - Parse HTML documents (similar to DOM in JS)
- Before building a scraper (“**web-scraping**”), let’s see how to download images from Google Image Search using Google Images API(“**google-image-api**”).

Parser for Social Network Text data

Suggestions?

Thank You!

Satish Palaniappan

Contact:

Email: tpsatish95@gmail.com

Phone: +91 9488515784

GitHub: https://github.com/tpsatish95

LinkedIn: https://in.linkedin.com/in/satishpalaniappan