Hands-on Workshop : Python and Machine Learning

SSN ACM Student Chapter and SSN Linux User Group

Agenda for the day

- Intro to Python and Setting up the environment
- Basics of Python
- Python Specific Constructs and Features
- Lunch Break
- Web Scraping with Python
  - Python Packages : Urllib, BeautifulSoup, etc.
  - Using the Google Image API with Python
  - Scraping “www.netlingo.com” for Web abbreviations (eg. LOL, ROFL, etc.)
- Hands-on session : Parser for Social Network Text data (Extends after the sessions too)

Python Environment Setup

- ➔ Find all the installation files in **“Py Environment Setup”** directory.
- ➔ Setting Up …
  - Refer to the **“Install-Notes.txt”** file in each of the directories for info regarding installing all the stuff inside it (follow the numbering order in the folder names while setting up).
- ➔ Then move into **“Py Script Files”** directory and try running the **“HelloTest.py”** as instructed in the script file.
- ➔ Python Interpreter.

If you got time: See, **Anaconda**(http://continuum.io/downloads): Anaconda is a completely free Python distribution that includes over 195 of the most popular Python packages for science, math, engineering, data analysis.

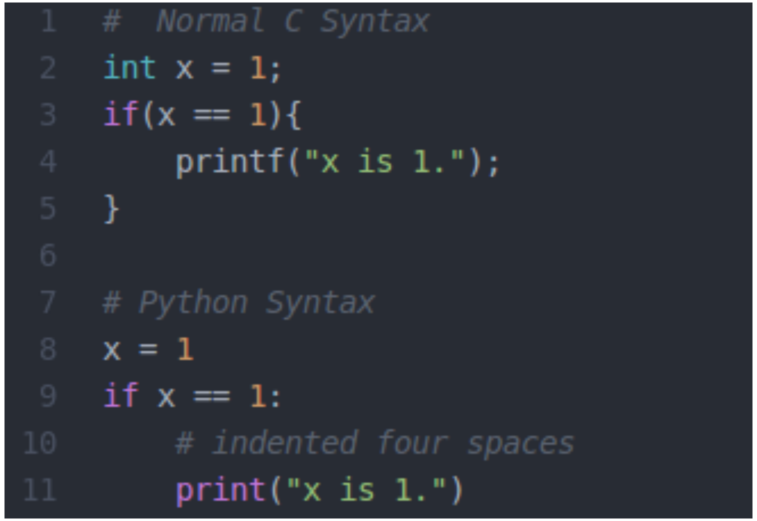
About Python

- ➔ 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

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.



```
#  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.")
```

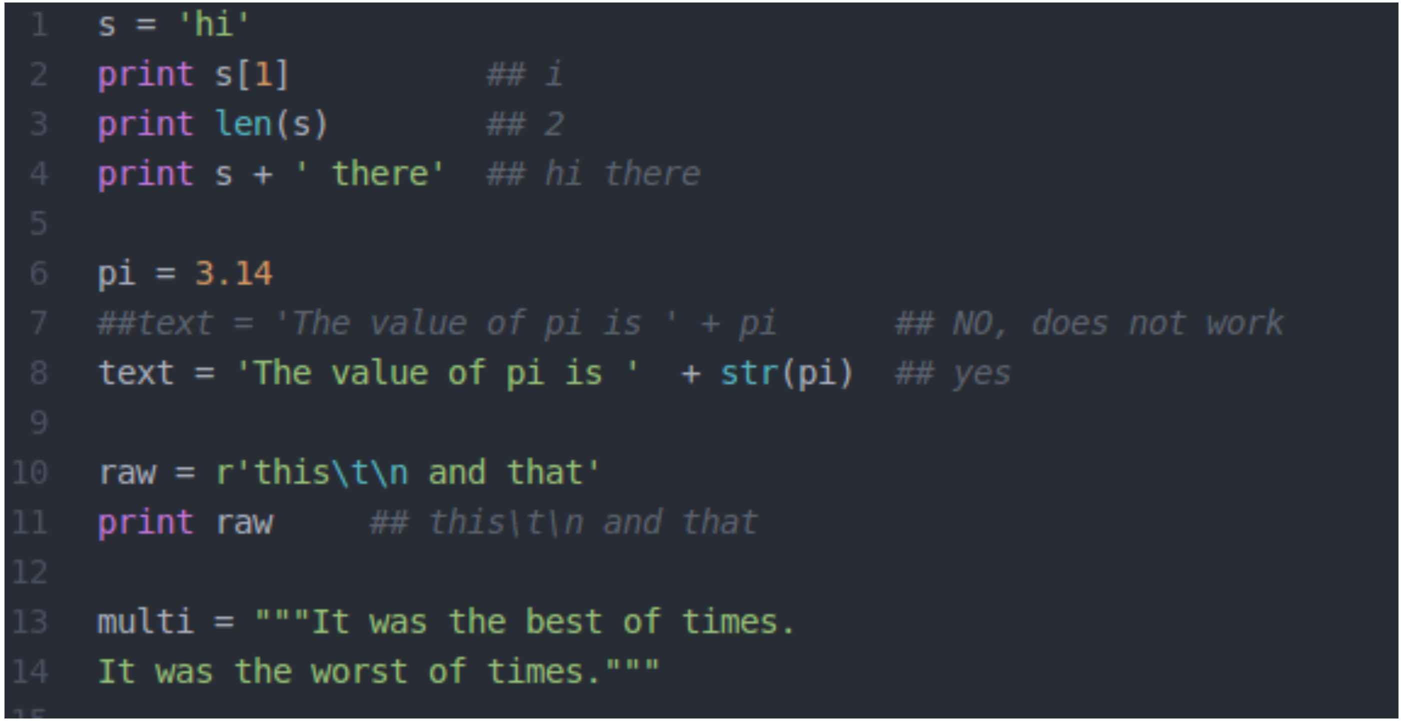
Basics of Python

- 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



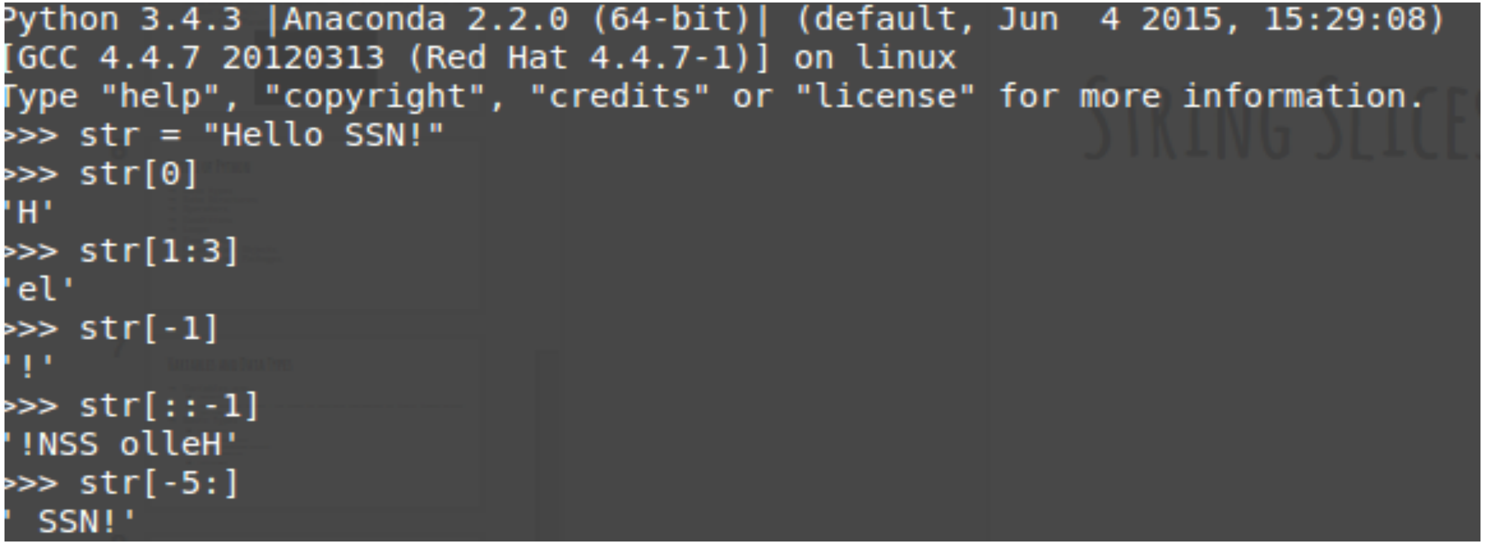
```python
s = 'hi'
print s[1]          ## i
print len(s)        ## 2
print s + ' there'  ## hi there

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)`

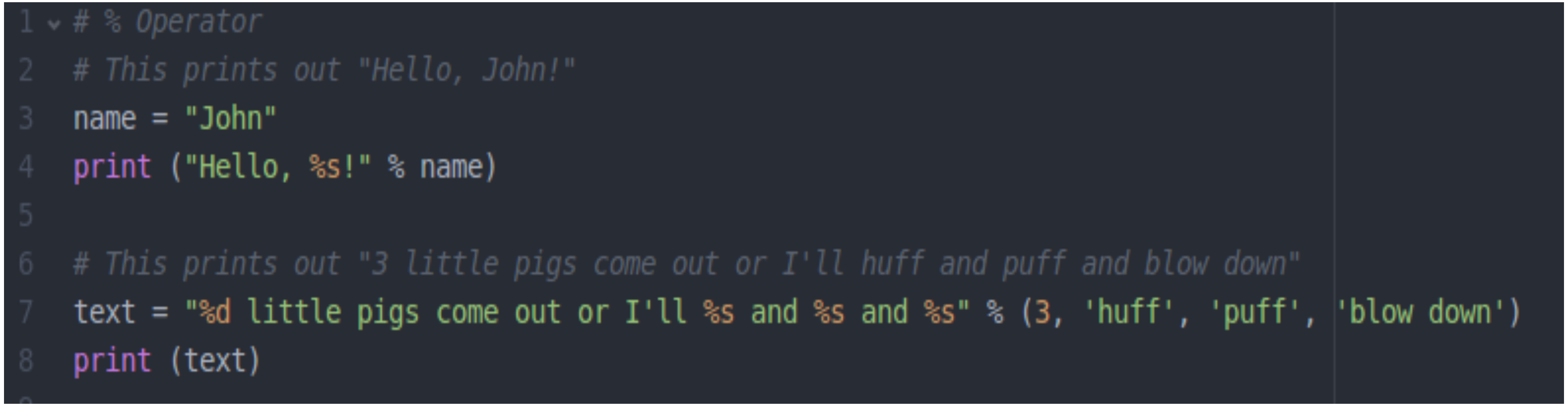
String Slices

```
 H    e    l    l    o

 0    1    2    3    4

-5   -4   -3   -2   -1
```



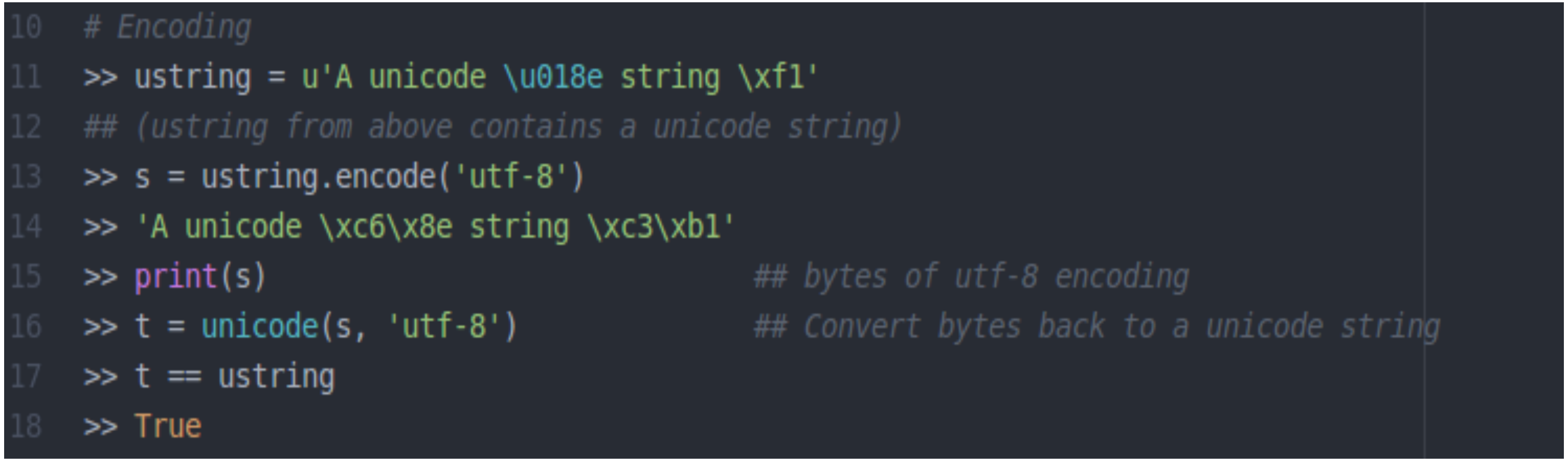
```
Python 3.4.3 |Anaconda 2.2.0 (64-bit)| (default, Jun  4 2015, 15:29:08)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> str = "Hello SSN!"
>>> str[0]
'H'
>>> str[1:3]
'el'
>>> str[-1]
'!'
>>> str[::-1]
'!NSS olleH'
>>> str[-5:]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

String Formatting



```python
# % 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)
```

String Encoding



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

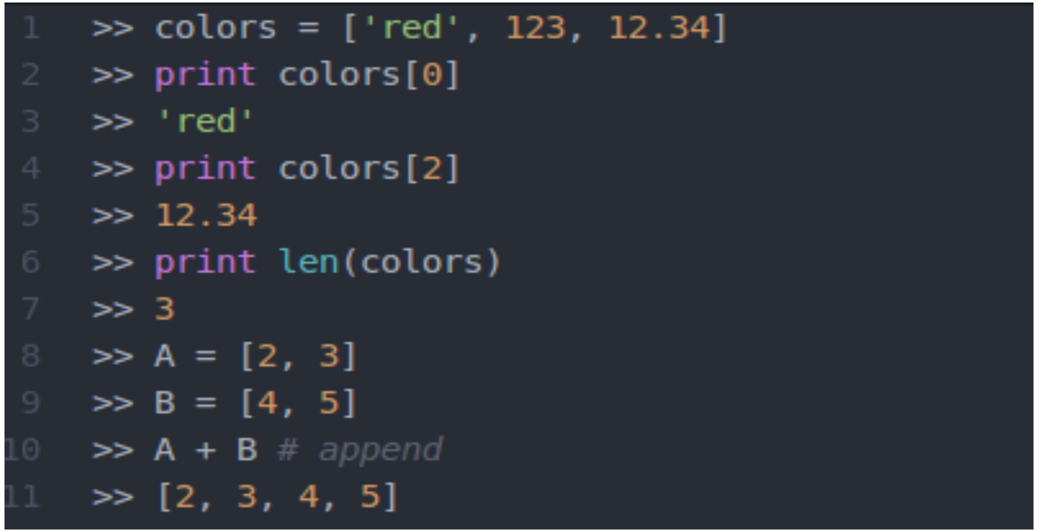
ISO-8859-1 is graphic character set that is a superset of “UTF-8”

Data Structures

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

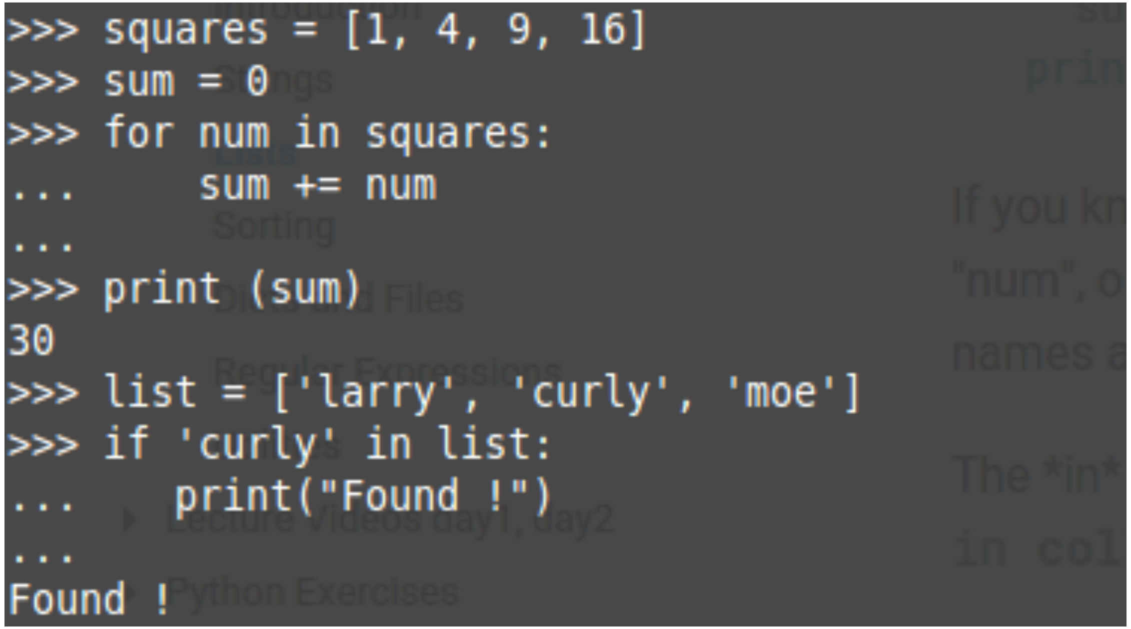
Python Lists

➔ 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]
```

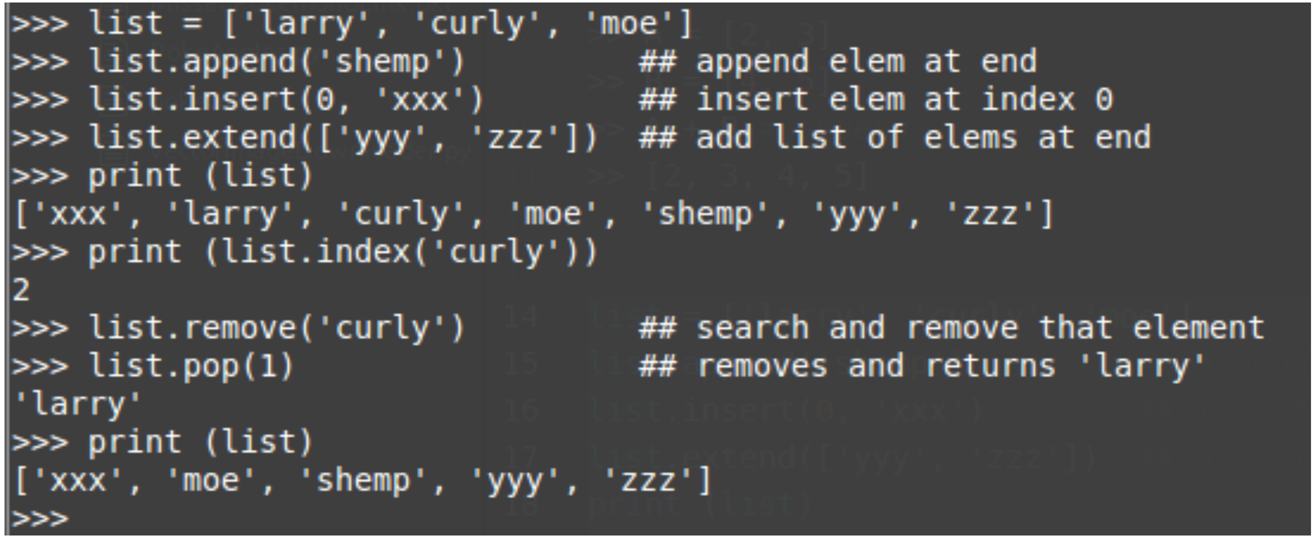
"FOR" AND "IN" IN LISTS



```
>>> 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)

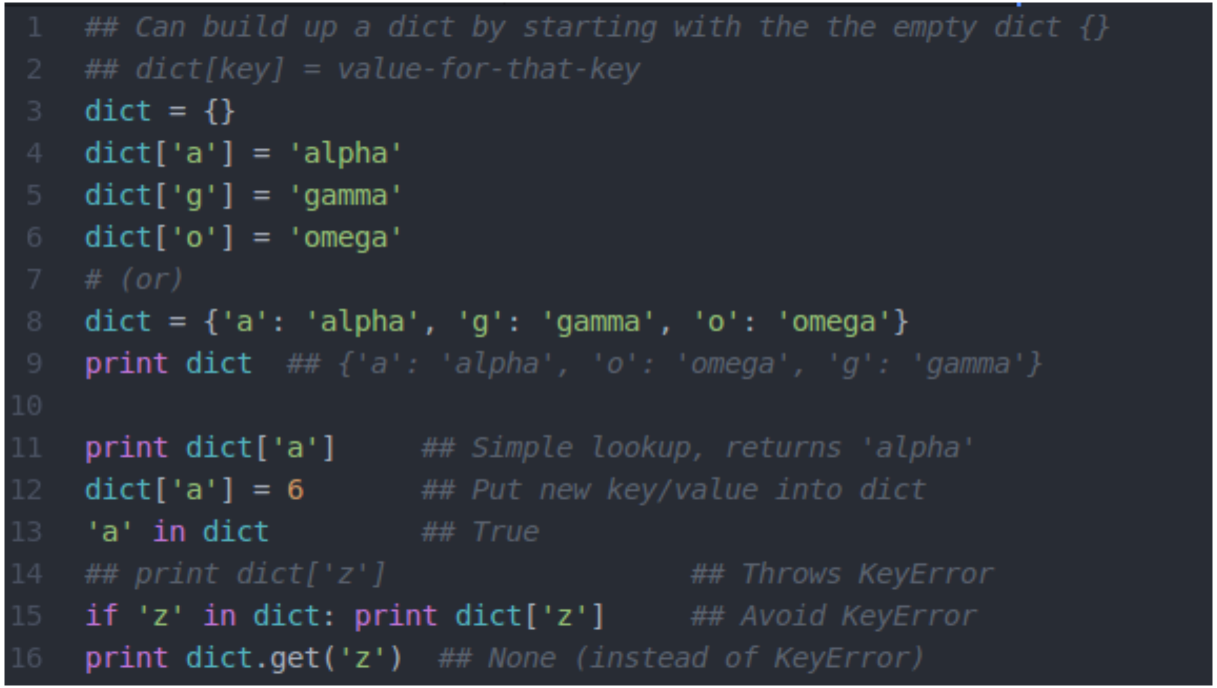


```
>>> list = ['larry', 'curly', 'moe']
>>> list.append('shemp')         ## append elem at end
>>> list.insert(0, 'xxx')        ## insert elem at index 0
>>> list.extend(['yyy', 'zzz'])  ## add list of elems at end
>>> print (list)
['xxx', 'larry', 'curly', 'moe', 'shemp', 'yyy', 'zzz']
>>> print (list.index('curly'))
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.

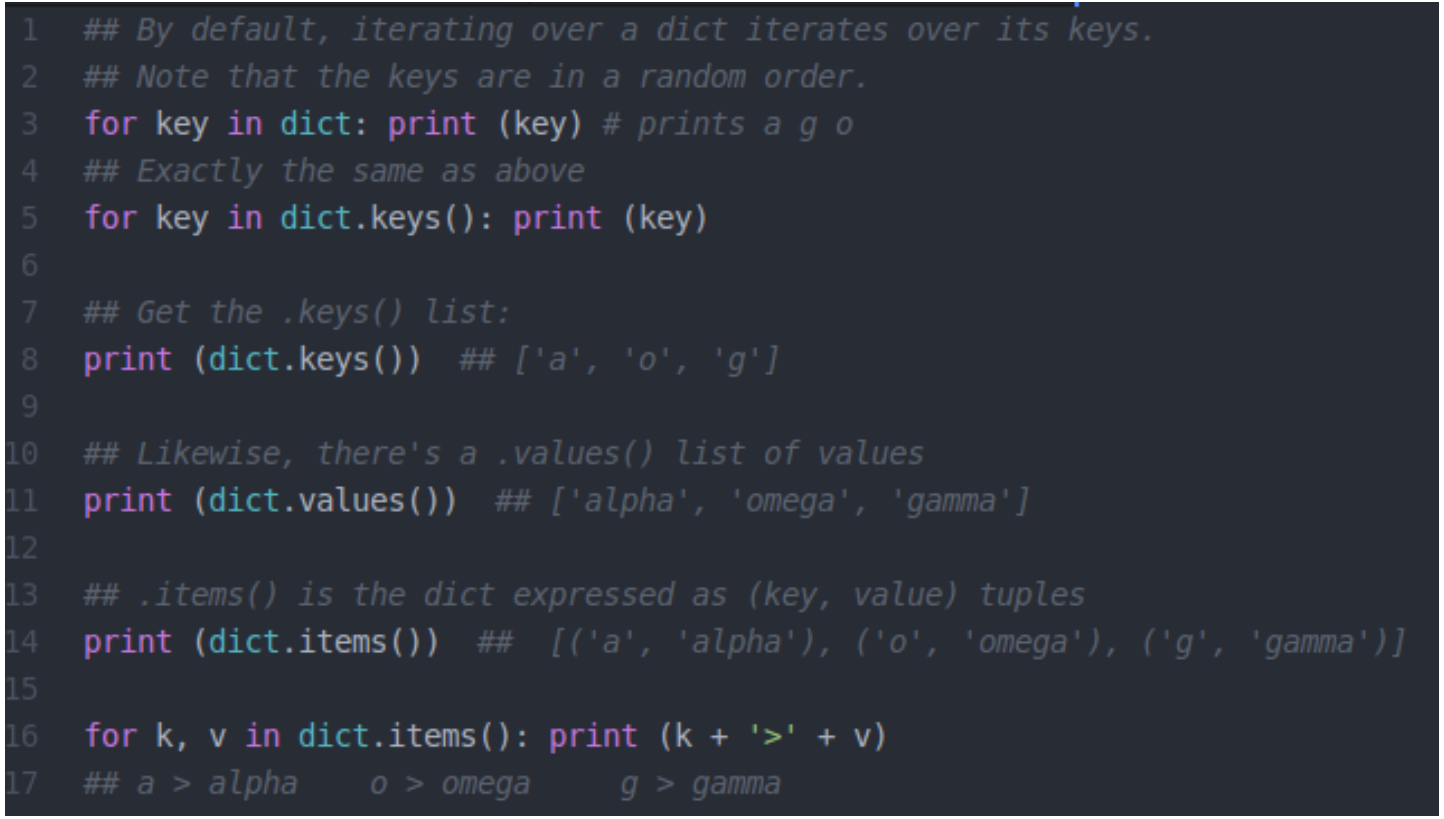
Python Dictionaries

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



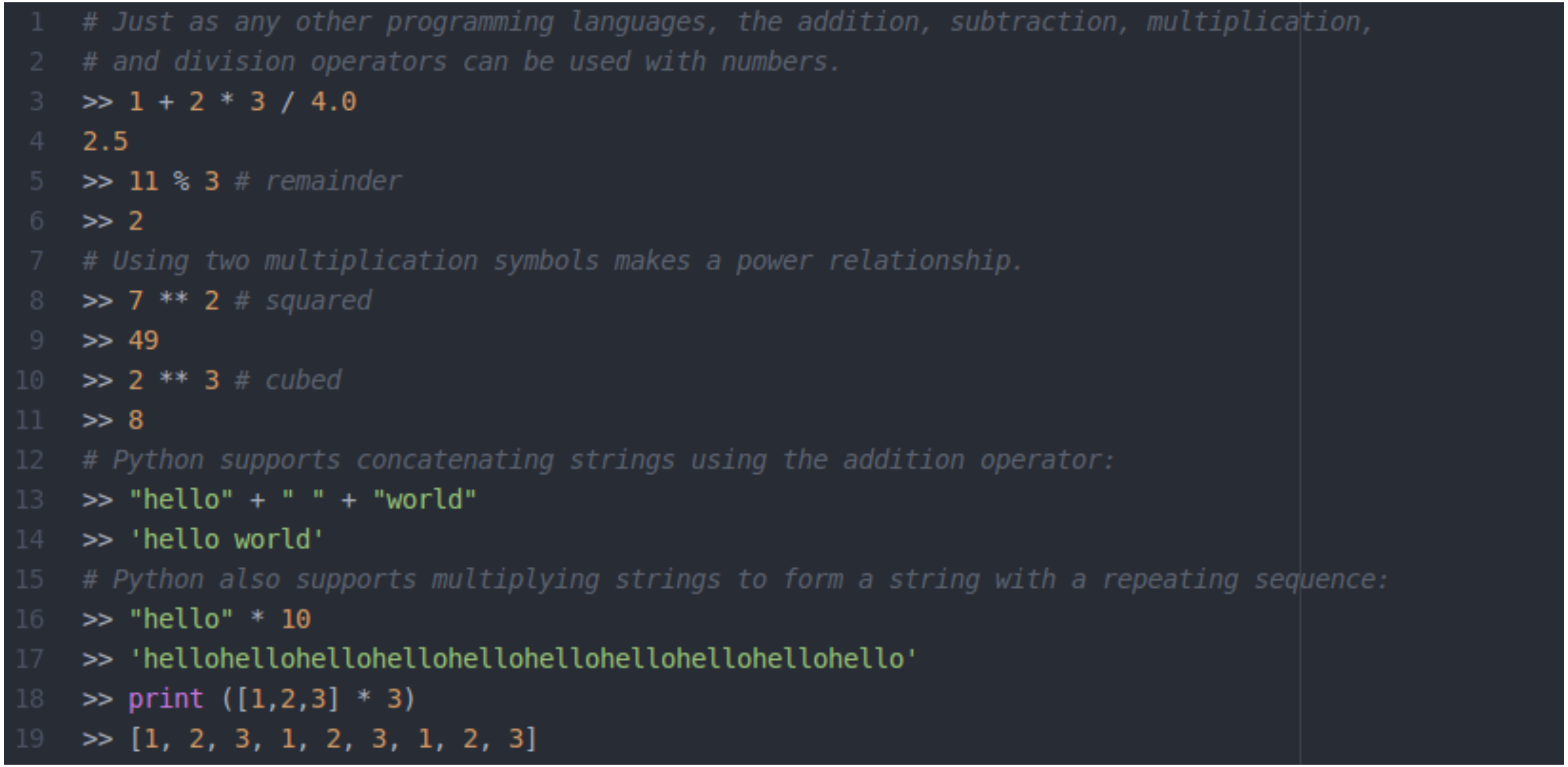
```
## Can build up a dict by starting with the the empty dict {}
## dict[key] = value-for-that-key
dict = {}
dict['a'] = 'alpha'
dict['g'] = 'gamma'
dict['o'] = 'omega'
# (or)
dict = {'a': 'alpha', 'g': 'gamma', 'o': 'omega'}
print dict  ## {'a': 'alpha', 'o': 'omega', 'g': 'gamma'}

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…



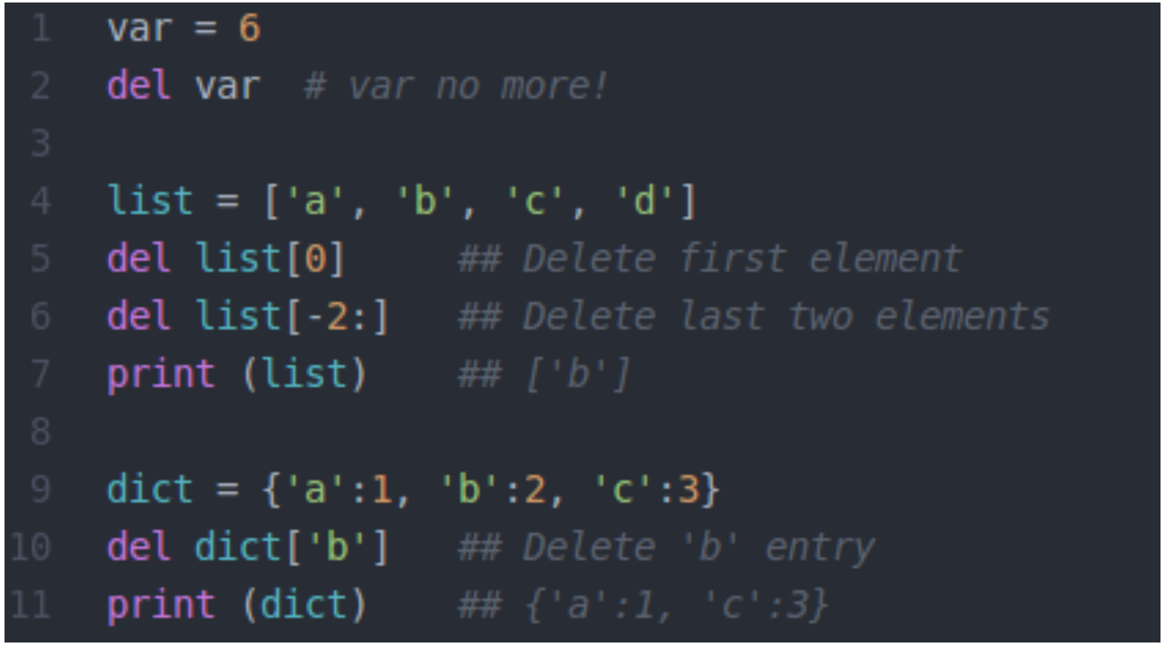
```python
## By default, iterating over a dict iterates over its keys.
## Note that the keys are in a random order.
for key in dict: print (key) # prints a g o
## Exactly the same as above
for key in dict.keys(): print (key)

## 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
```

Operators



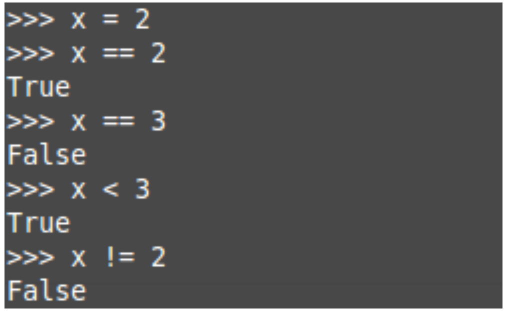
```
# Just as any other programming languages, the addition, subtraction, multiplication,
# and division operators can be used with numbers.
>> 1 + 2 * 3 / 4.0
2.5
>> 11 % 3 # remainder
>> 2
# Using two multiplication symbols makes a power relationship.
>> 7 ** 2 # squared
>> 49
>> 2 ** 3 # cubed
>> 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



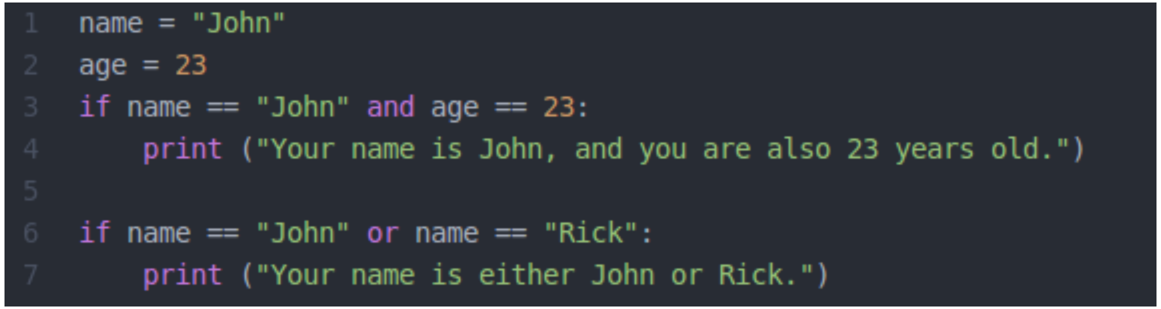
```python
var = 6
del var  # var no more!

list = ['a', 'b', 'c', 'd']
del list[0]       ## Delete first element
del list[-2:]     ## Delete last two elements
print (list)      ## ['b']

dict = {'a':1, 'b':2, 'c':3}
del dict['b']     ## Delete 'b' entry
print (dict)      ## {'a':1, 'c':3}
```

Conditions



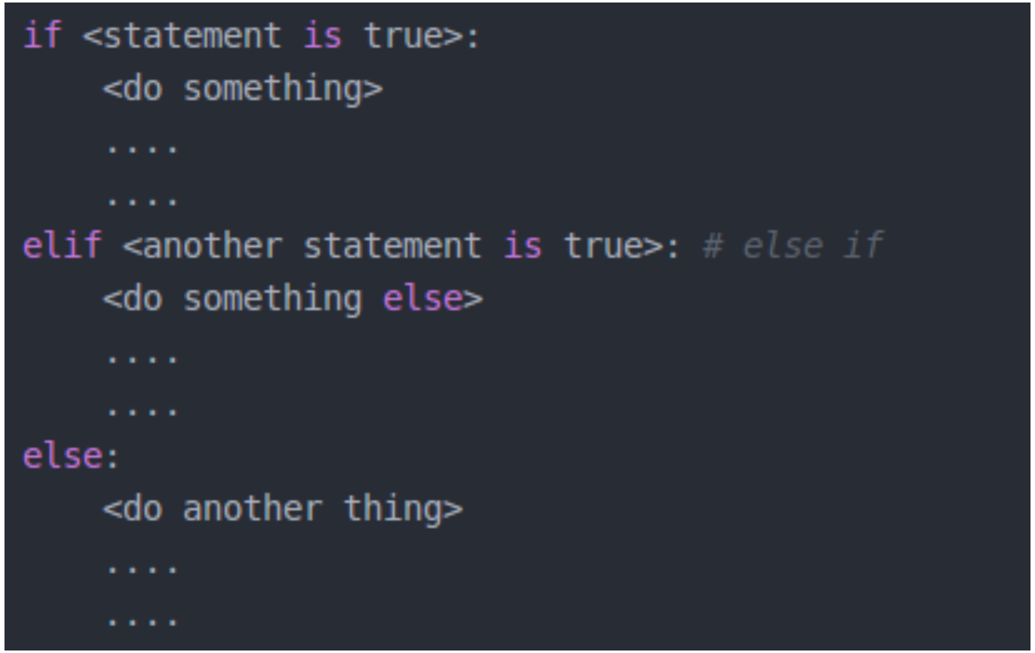
```
>>> x = 2
>>> x == 2
True
>>> x == 3
False
>>> x < 3
True
>>> x != 2
False
```

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



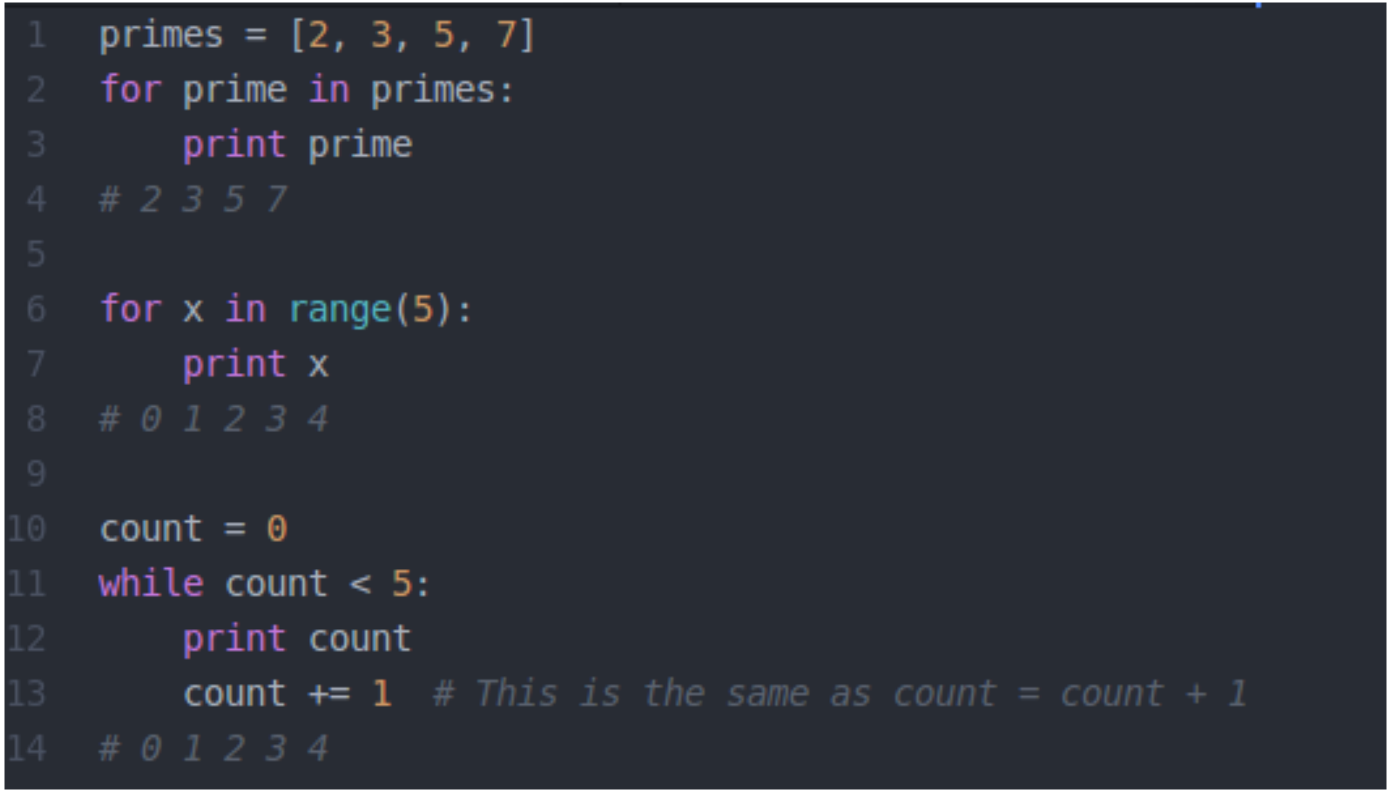
```
name = "John"
age = 23
if name == "John" and age == 23:
    print ("Your name is John, and you are also 23 years old.")

if name == "John" or name == "Rick":
    print ("Your name is either John or Rick.")
```

If ... Else ...



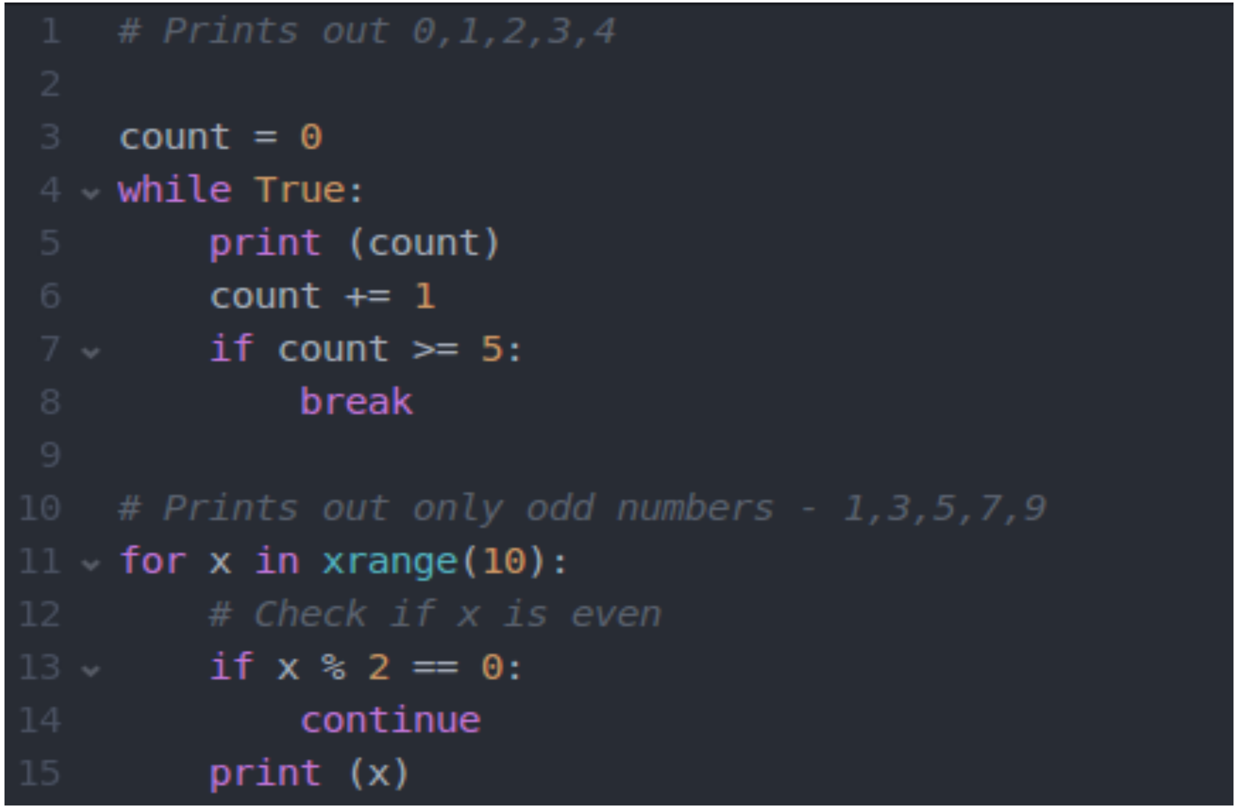
```
if <statement is true>:
    <do something>
    ....
    ....
elif <another statement is true>: # else if
    <do something else>
    ....
    ....
else:
    <do another thing>
    ....
    ....
```

Loops - For, While



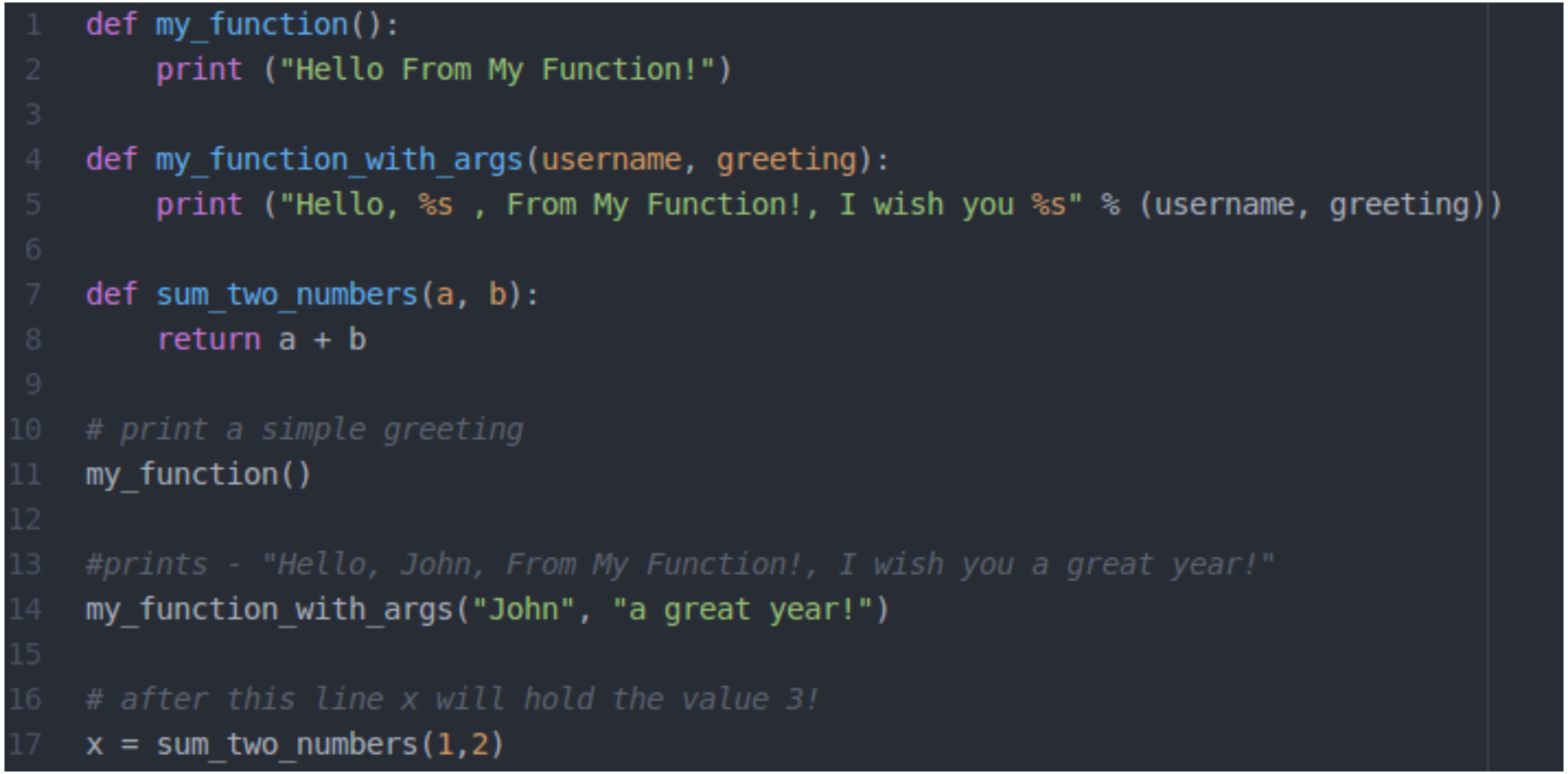
```python
primes = [2, 3, 5, 7]
for prime in primes:
    print prime
# 2 3 5 7

for x in range(5):
    print x
# 0 1 2 3 4

count = 0
while count < 5:
    print count
    count += 1  # This is the same as count = count + 1
# 0 1 2 3 4
```

"Break" and "Continue"



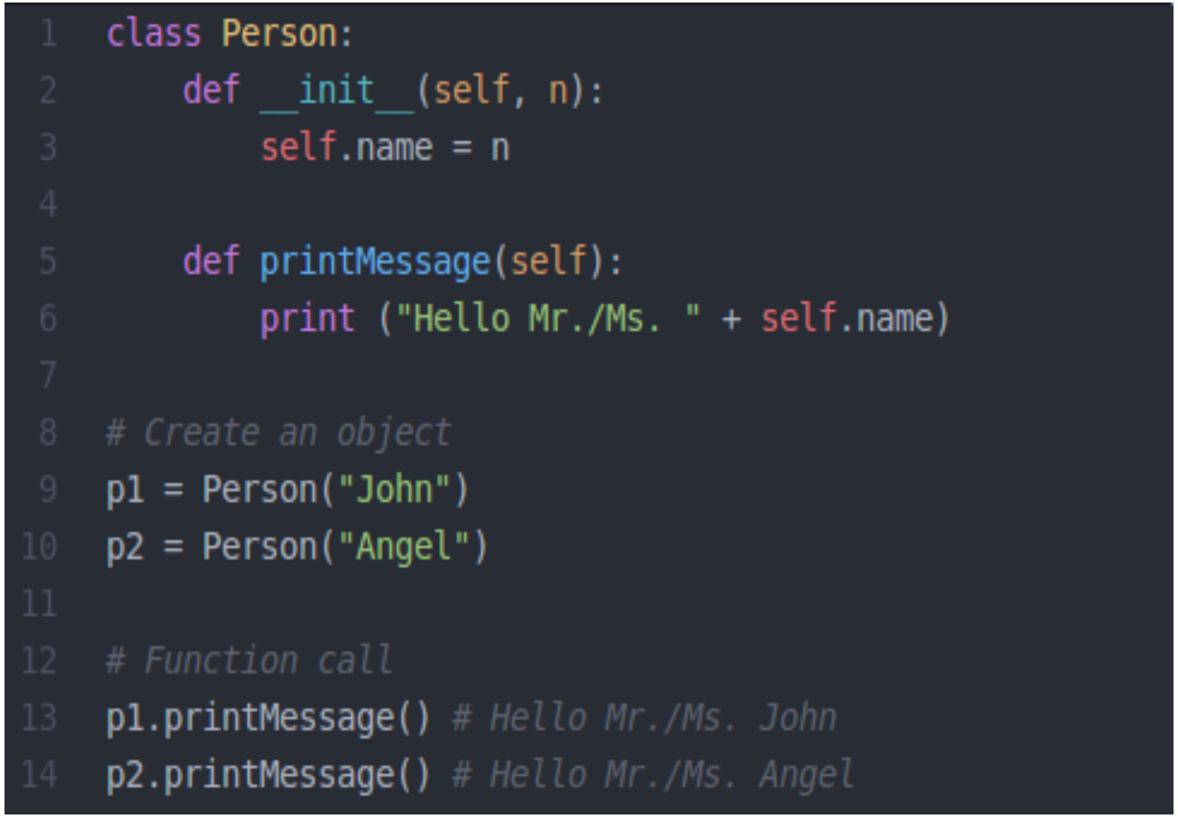
```
# Prints out 0,1,2,3,4

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)
```

Functions



```python
def my_function():
    print ("Hello From My Function!")

def my_function_with_args(username, greeting):
    print ("Hello, %s , From My Function!, I wish you %s" % (username, greeting))

def sum_two_numbers(a, b):
    return a + b

# print a simple greeting
my_function()

#prints - "Hello, John, From My Function!, I wish you a great year!"
my_function_with_args("John", "a great year!")

# after this line x will hold the value 3!
x = sum_two_numbers(1,2)
```

Classes and Objects



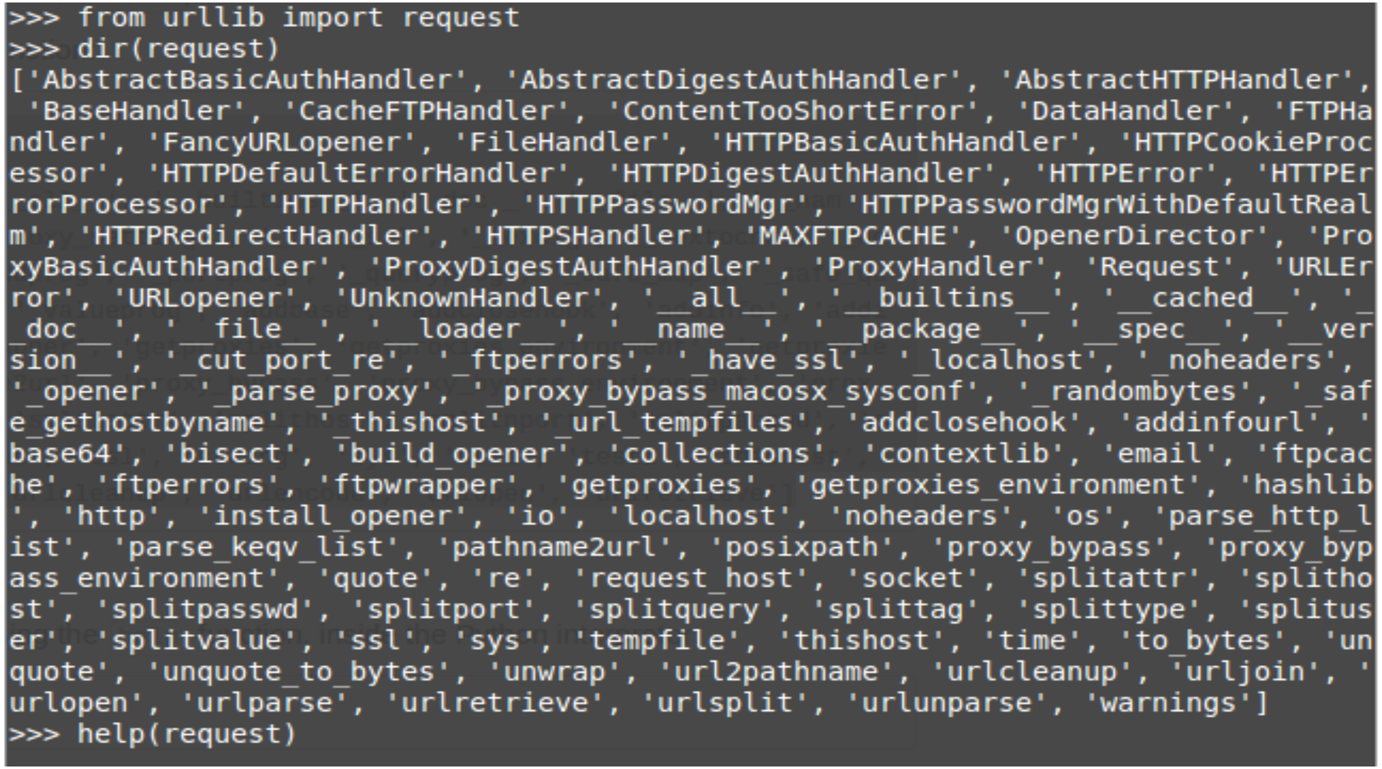
```python
class Person:
    def __init__(self, n):
        self.name = n

    def printMessage(self):
        print ("Hello Mr./Ms. " + self.name)

# Create an object
p1 = Person("John")
p2 = Person("Angel")

# Function call
p1.printMessage() # Hello Mr./Ms. John
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

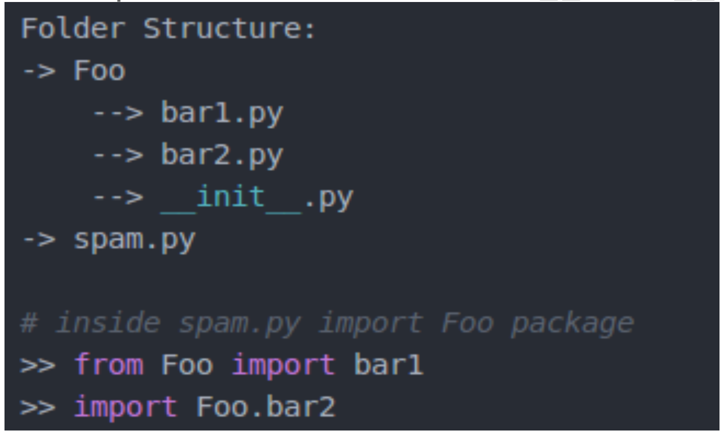
- “*dir*” and “*help*” functions are used to explore.



```
>>> from urllib import request
>>> dir(request)
['AbstractBasicAuthHandler', 'AbstractDigestAuthHandler', 'AbstractHTTPHandler',
 'BaseHandler', 'CacheFTPHandler', 'ContentTooShortError', 'DataHandler', 'FTPHa
ndler', 'FancyURLopener', 'FileHandler', 'HTTPBasicAuthHandler', 'HTTPCookieProc
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.



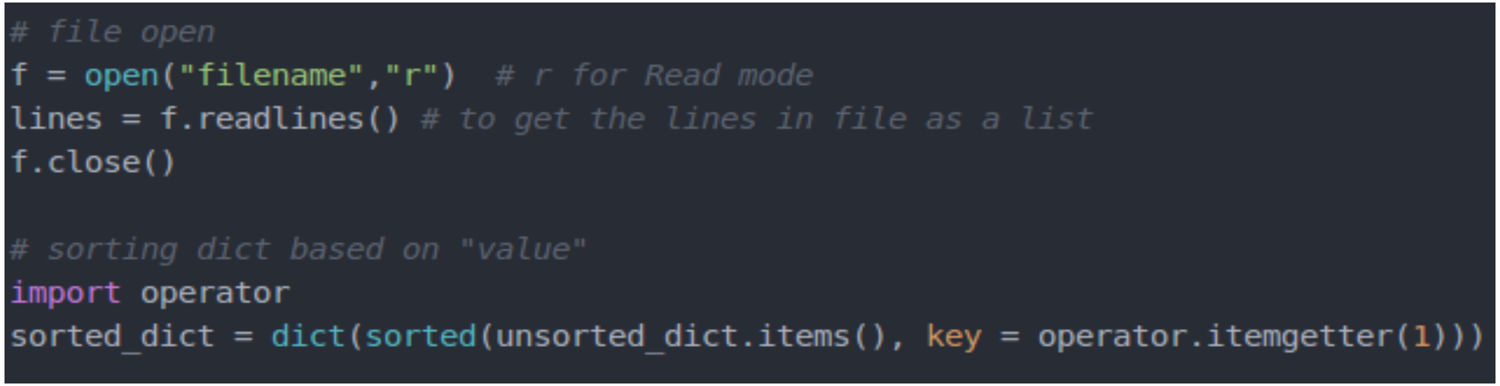
```
Folder Structure:
-> Foo
    --> bar1.py
    --> bar2.py
    --> __init__.py
-> spam.py

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

Some Packages I Have USed:

- ➔ Scikit-Learn - Machine Learning Tools
- ➔ Scrapy - Web Scrapping
- ➔ Nltk - Natural Language Processing Tools
- ➔ Gensim - Topic Modelling Library
- ➔ Numpy, Scipy - Mathematical and Scientific Library
- ➔ Cherrypy - lightweight Apache-server-like python version
- ➔ Tweepy - Twitter API
- ➔ Twokenize - Tweets Tokenizer Tool.
- ➔ Etc…

Time To Code ... !

- ➔ Open the folder **"/Py Script Files/Ex1/"**.
- ➔ 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), list slicing, files io, sorting, etc to build the solution.
- ➔ Hint (File IO and Sorting):



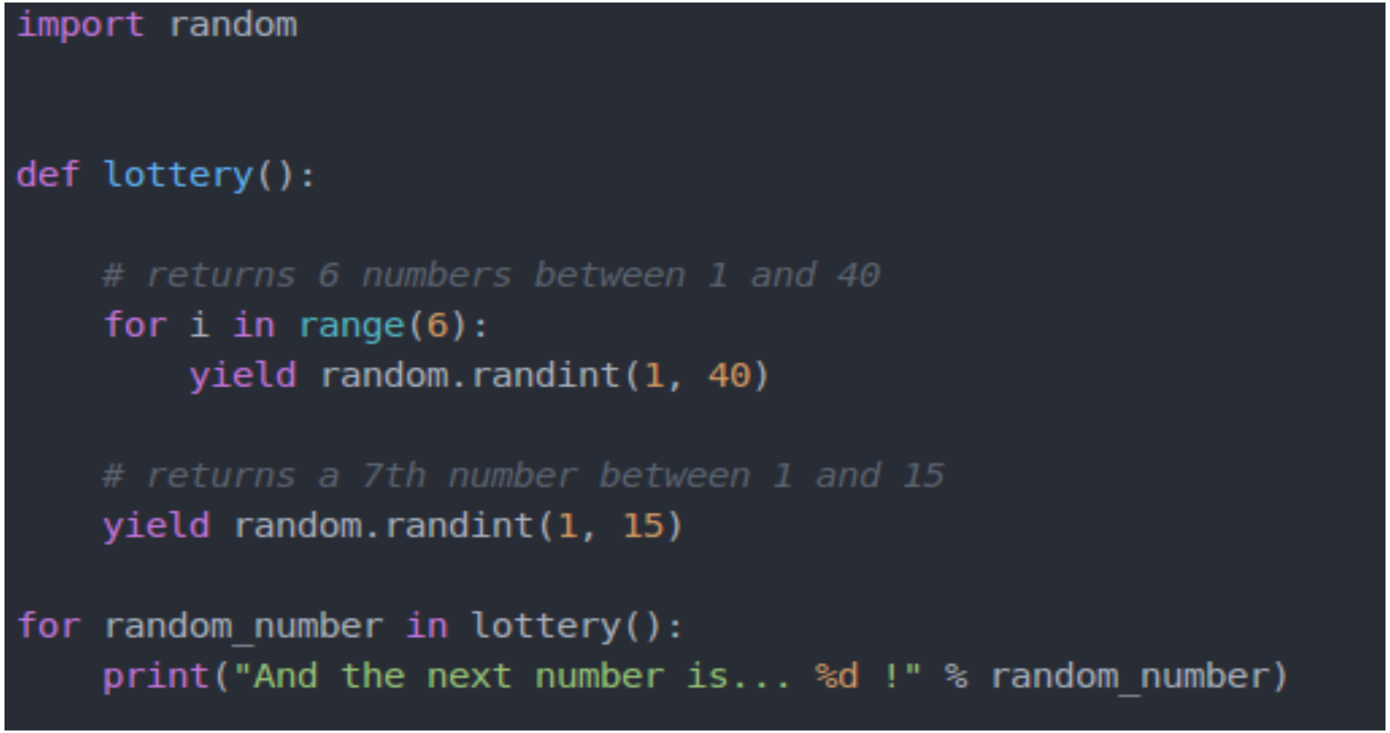
```python
# file open
f = open("filename","r")  # r for Read mode
lines = f.readlines() # to get the lines in file as a list
f.close()

# 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

Generators

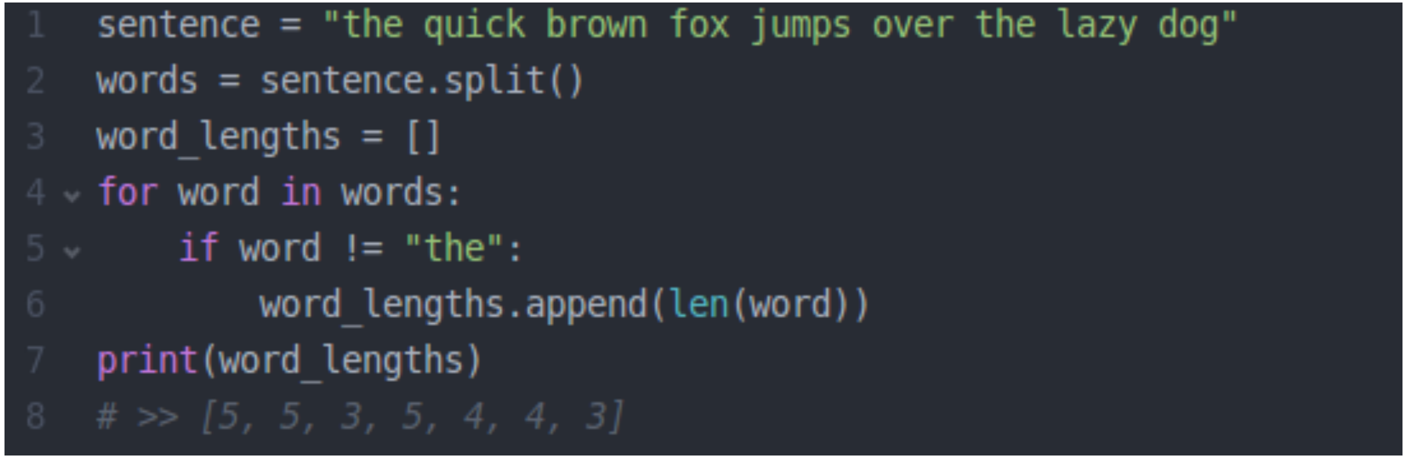
Simple functions which return an iterable set of items, one at a time, in a special way.



```
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)
```

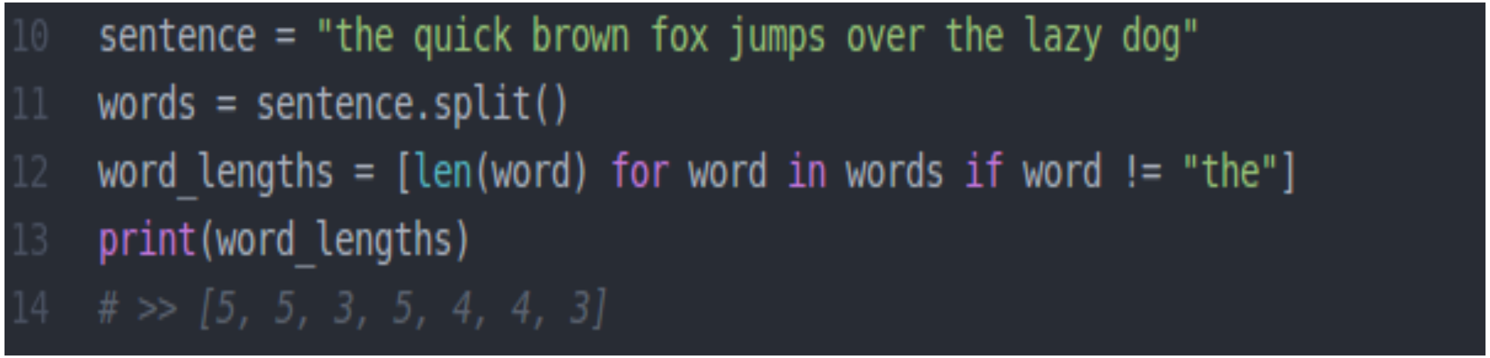
List Comprehensions

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



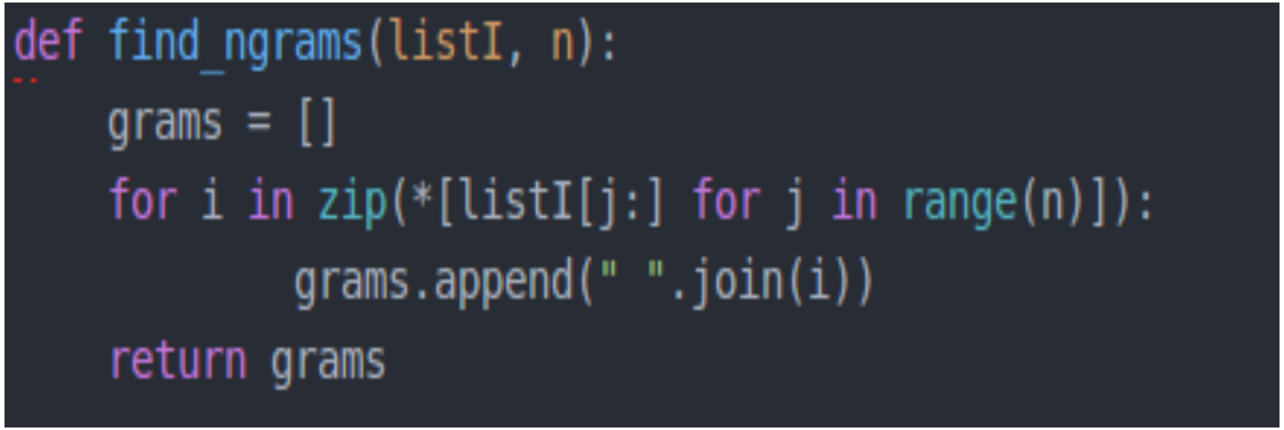
```python
sentence = "the quick brown fox jumps over the lazy dog"
words = sentence.split()
word_lengths = []
for word in words:
    if word != "the":
        word_lengths.append(len(word))
print(word_lengths)
# >> [5, 5, 3, 5, 4, 4, 3]
```

Much Clean and Simple Version



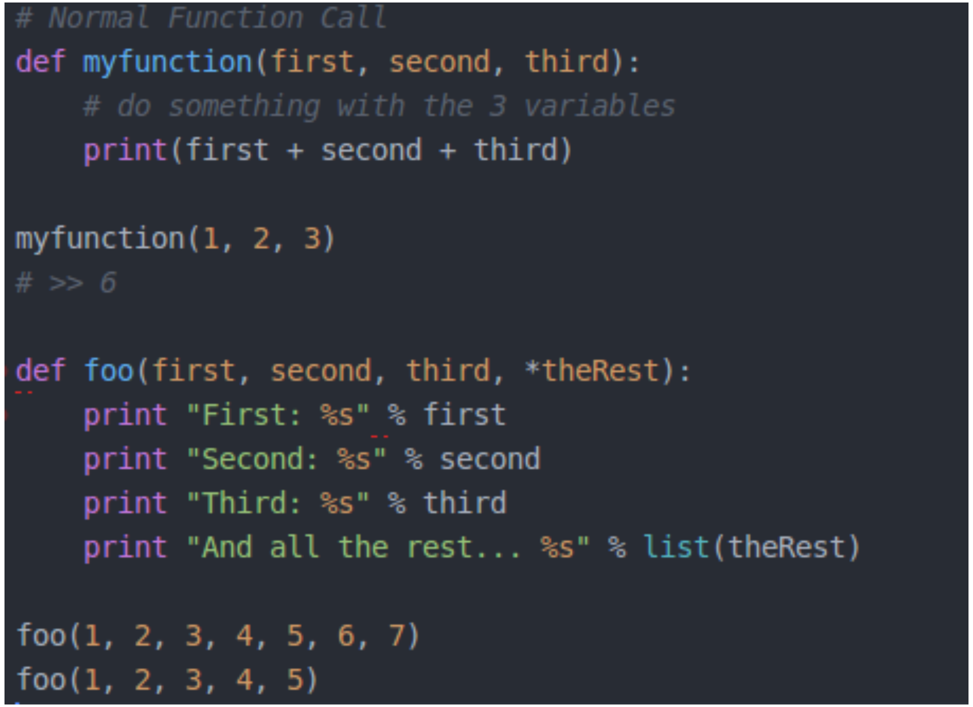
```python
sentence = "the quick brown fox jumps over the lazy dog"
words = sentence.split()
word_lengths = [len(word) for word in words if word != "the"]
print(word_lengths)
# >> [5, 5, 3, 5, 4, 4, 3]
```

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

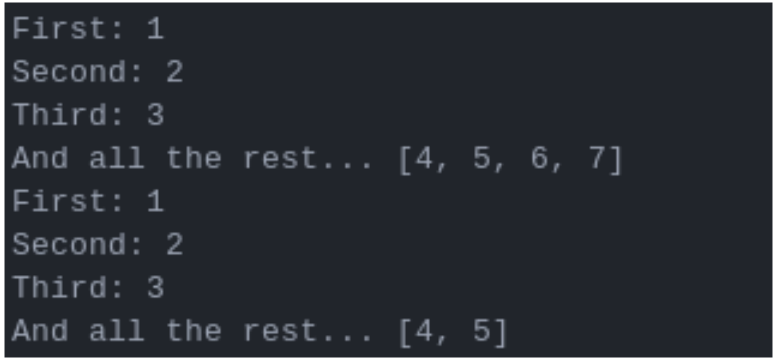


```
def find_ngrams(listI, n):
    grams = []
    for i in zip(*[listI[j:] for j in range(n)]):
            grams.append(" ".join(i))
    return grams
```

Multiple Function Arguments

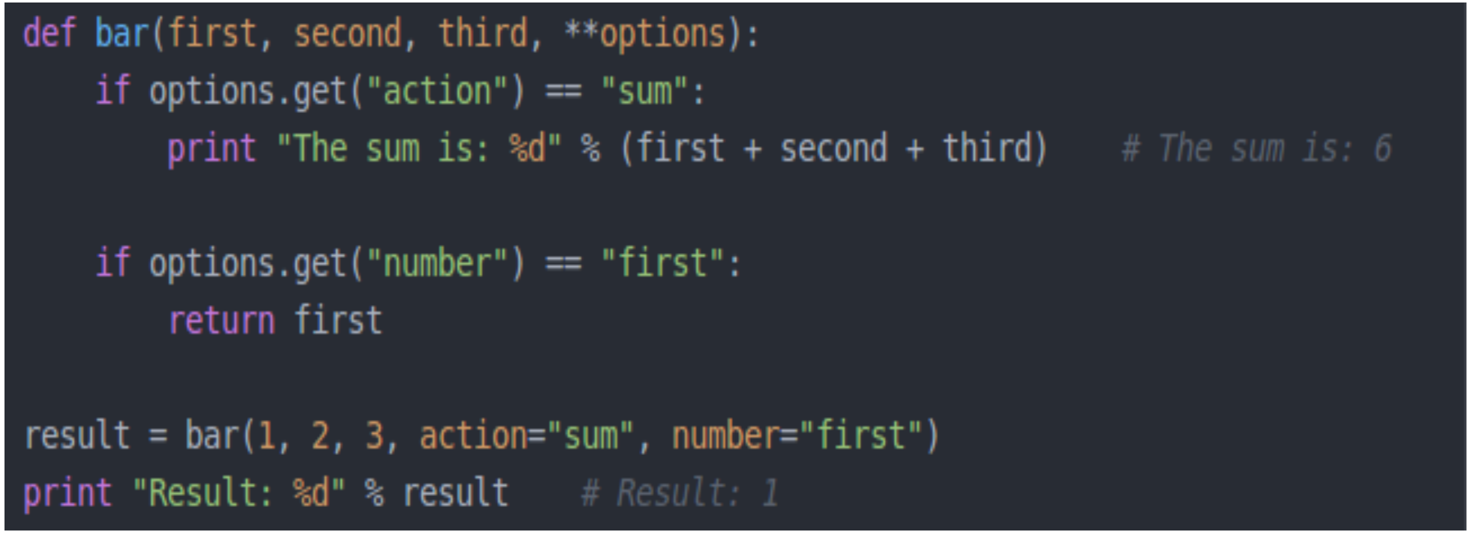


```python
# Normal Function Call
def myfunction(first, second, third):
    # do something with the 3 variables
    print(first + second + third)

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)
```



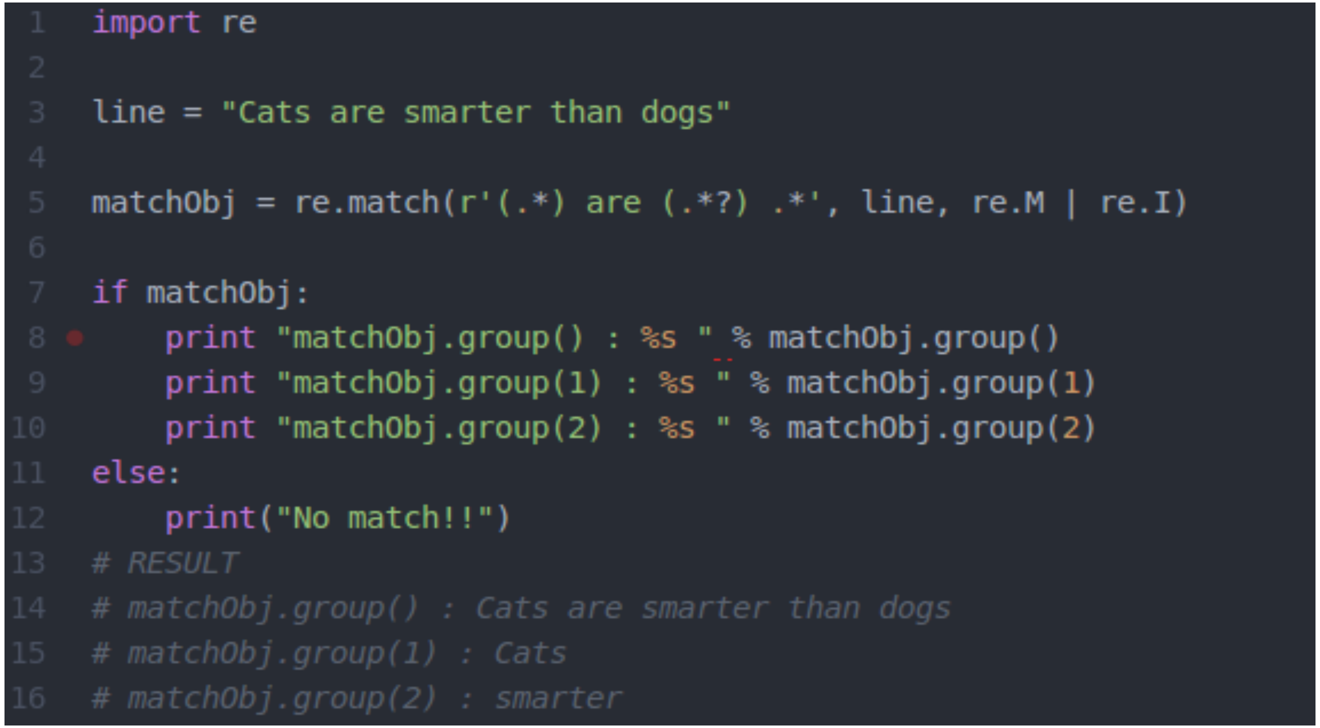
```
First: 1
Second: 2
Third: 3
And all the rest... [4, 5, 6, 7]
First: 1
Second: 2
Third: 3
And all the rest... [4, 5]
```

More ...



```python
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
```

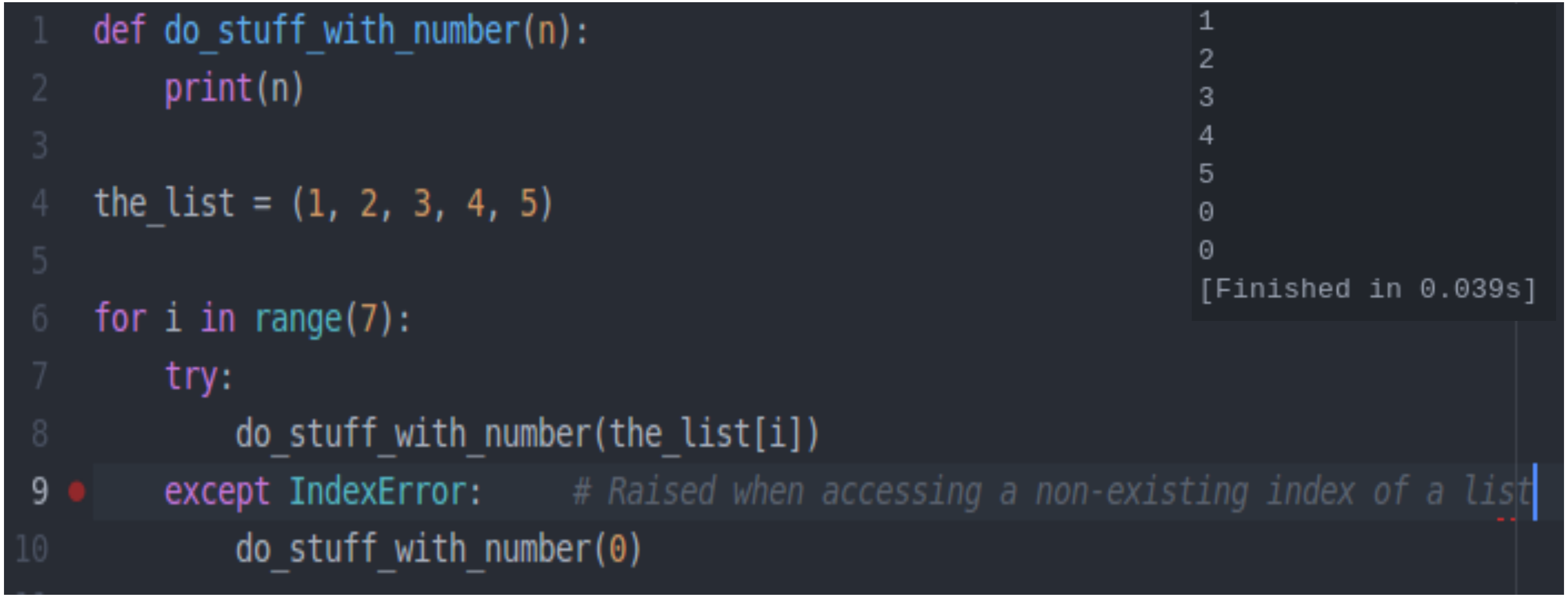
Regular Expressions



```python
import re

line = "Cats are smarter than dogs"

matchObj = re.match(r'(.*) are (.*?) .*', line, re.M | re.I)

if matchObj:
    print "matchObj.group() : %s " % matchObj.group()
    print "matchObj.group(1) : %s " % matchObj.group(1)
    print "matchObj.group(2) : %s " % matchObj.group(2)
else:
    print("No match!!")
# RESULT
# matchObj.group() : Cats are smarter than dogs
# matchObj.group(1) : Cats
# matchObj.group(2) : smarter
```

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

Exception Handling

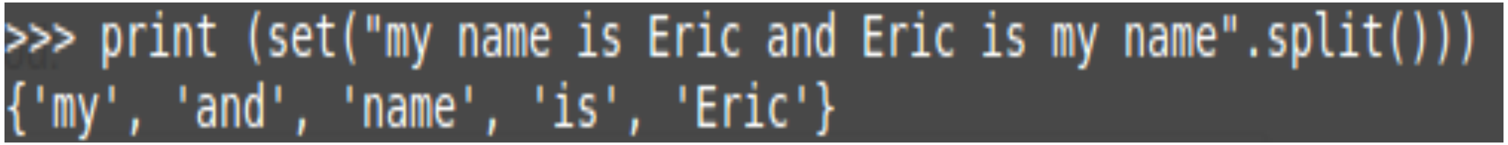


```
def do_stuff_with_number(n):
    print(n)

the_list = (1, 2, 3, 4, 5)

for i in range(7):
    try:
        do_stuff_with_number(the_list[i])
    except IndexError:     # Raised when accessing a non-existing index of a list
        do_stuff_with_number(0)
```

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

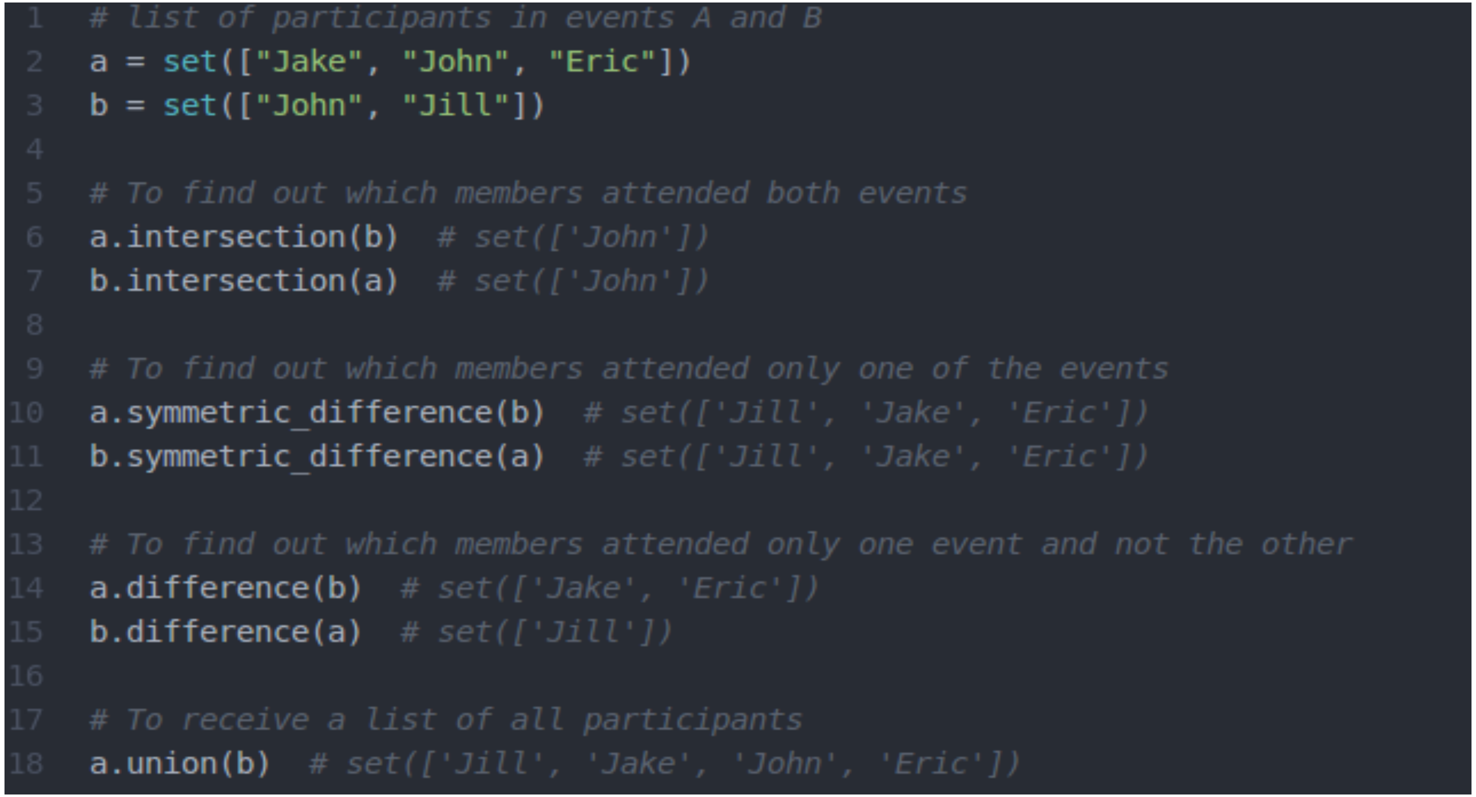
Sets

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



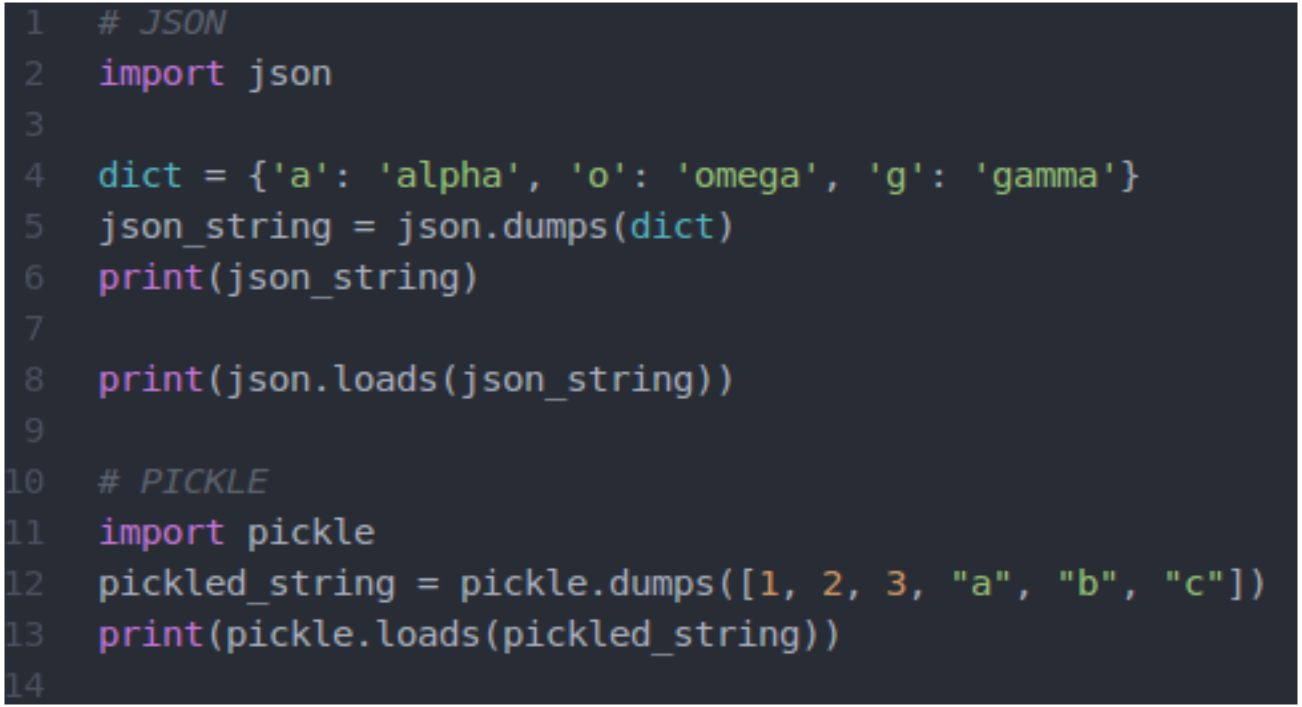
```
>>> print (set("my name is Eric and Eric is my name".split()))
{'my', 'and', 'name', 'is', 'Eric'}
```

EXAMPLE:



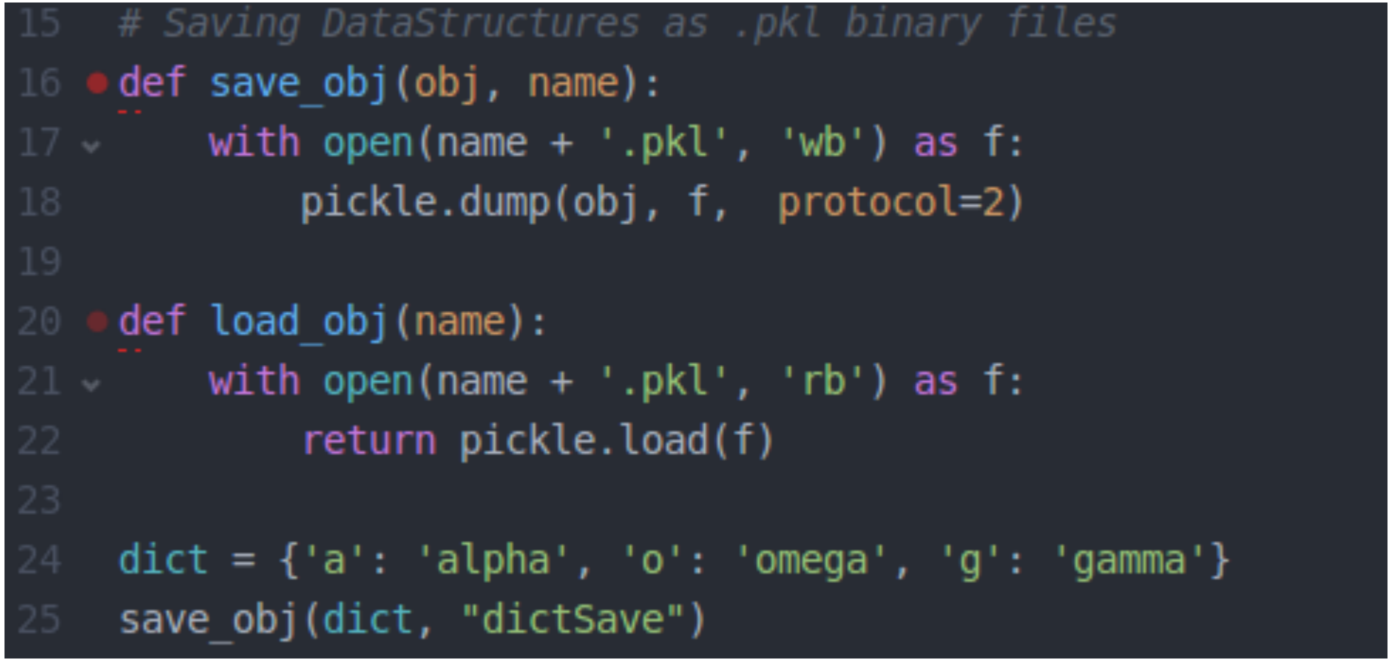
```python
# list of participants in events A and B
a = set(["Jake", "John", "Eric"])
b = set(["John", "Jill"])

# 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'])
```

Serialization



```python
# 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))

```

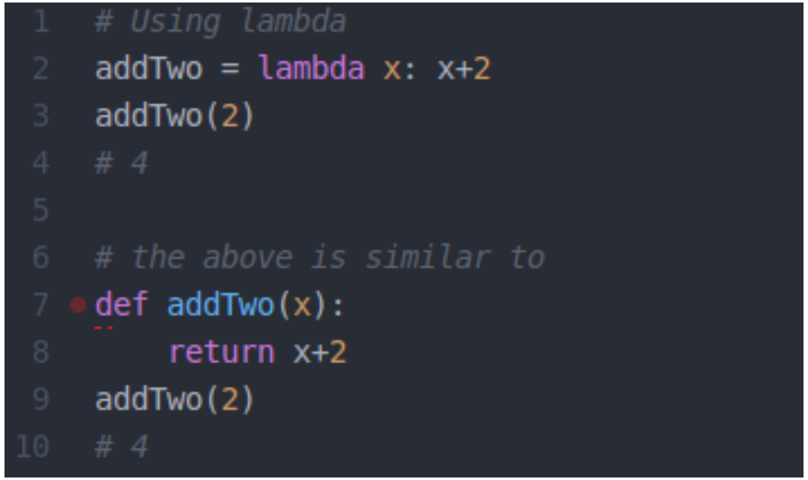
Saving Objects and Data Structures - Persistence



```python
# Saving DataStructures as .pkl binary files
def save_obj(obj, name):
    with open(name + '.pkl', 'wb') as f:
        pickle.dump(obj, f,  protocol=2)

def load_obj(name):
    with open(name + '.pkl', 'rb') as f:
        return pickle.load(f)

dict = {'a': 'alpha', 'o': 'omega', 'g': 'gamma'}
save_obj(dict, "dictSave")
```

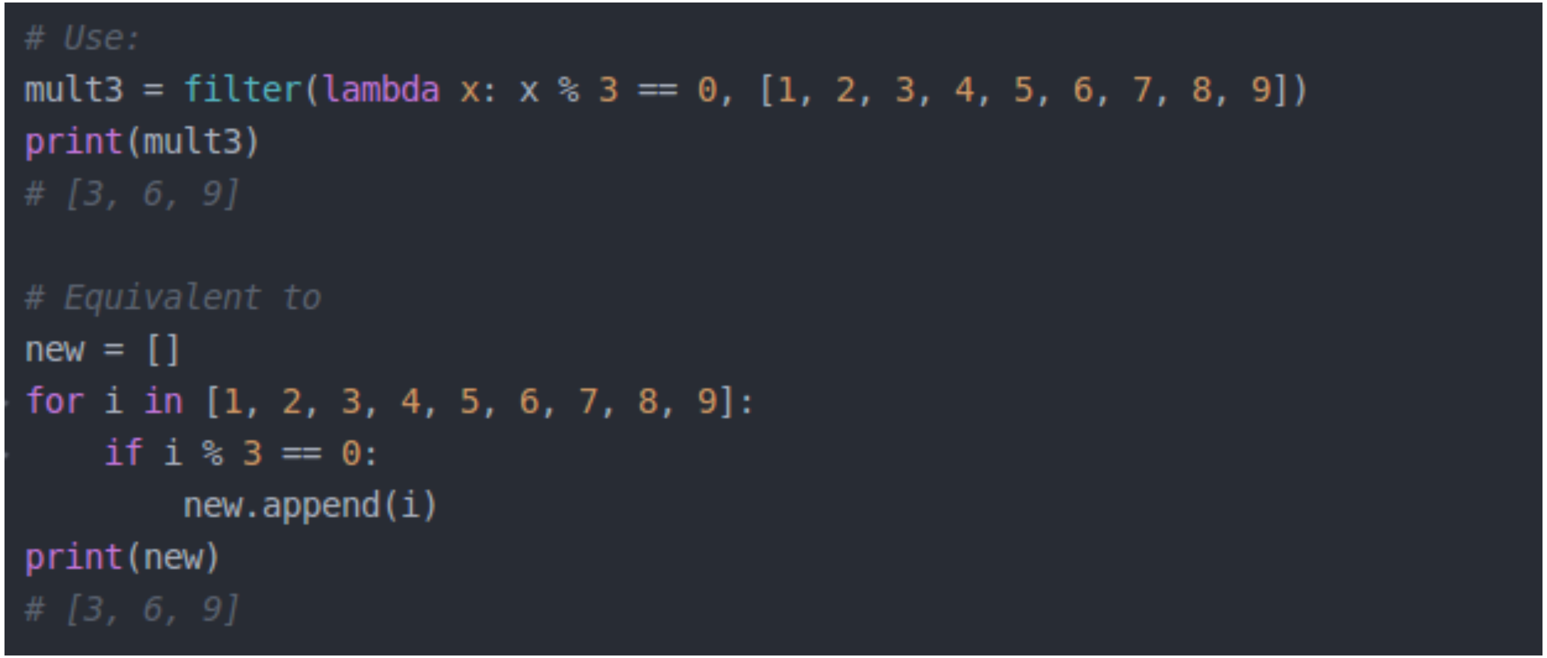
Lambda Operator

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



```python
# 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



```python
# 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]
```

Time to Code ... !

- ➔ Open the folder **“/Py Script Files/Ex2/”**.
- ➔ 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

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, let’s see how to download images from Google Image Search using Google Images API.
- ➔ To the code …

Hands-On Session :
Parser for Social Network Text data

Some Pointers to Read Up...

- http://stackoverflow.com/questions/3217222/beginner-python-practice/3226704#3226704
- **Python Google Code University:** (Python User Community) https://groups.google.com/forum/?fromgroups#!forum/python-gcu-forum (Apart from Stack Overflow !)
- http://coursera.org/course/interactivepython
- http://www.informit.com/articles/article.aspx?p=1849069

Thank You!

Satish Palaniappan

Contact:

Email:	tpsatish95@gmail.com
Phone:	+91 9488515784
Github:	tpsatish95
LinkedIn:	https://in.linkedin.com/in/satishpalaniappan