Python Programming Workshop

Day - 1

Why Python?

- Easy to Learn
  - Syntax
  - Modular
- Large Community - Many Libraries available in Free
- Used in Almost all domains
  - Complex image processing, the Python Imaging Library - http://www.pythonware.com/products/pil/
  - Want to create games? PyGame - http://www.pygame.org/hifi.html
  - If data science is your thing, SciPy - https://www.scipy.org/

Python Installation

- Version - Python 3.6.1 (https://www.python.org/downloads/)
- Windows OS - update PATH environment variable appropriately

Python - Hello

- How to execute/run Python Programs/Scripts?
- Meaning of Python Application - Web, Automation Tools
- Windows vs Linux vs Mac - Terminals/IDE's, IDLE
- Interactive Python

hello.py

```
#!/usr/bin/python3
print ("Hello \"World")
print ('Hello World')
print ("Hello, 'My Dear Friend'")
print ('Hello, "My Dear Friend"')
print ("One", "Two")
```

Python - Hello (cont ...)

hello_multiple.py .

```
#!/usr/bin/python3

print("""
One - Hello
two - Python
three - Data Science
""")

print ("One", "Two", 'One-1', 'Two-2')

print ("I am first line\nI am second line")
```

Task - 1

- Develop script “hello_workshop.py”
- It should print following output

```
Hello to Python Workshop
This is:
    - Python Workshop
    - I am here to explore Python
    - I am suppose to gain skills
```

Comments

- Execution Flow - top to bottom
- Triple - single, double quotes
- Hash (#) - single line comments

comments.py .

```
#!/usr/bin/python3

#this is a comment in Python

print ("Hello World") #This is also a comment in Python

""" This is an example of a multiline
comment that spans multiple lines
...
"""

print ("Let me try triple quotes")
'''
I am also comment
in muliple lines

'''
```

Word on Data Types

- Numbers
  - Int
  - float
- Character Strings
- Boolean - True, False
- List
- Tuple
- Dictionary

- Numbers:
  - 3, 6, -7, 1.25
- Character strings:
  - 'hello', 'The answer is: '
- Lists of objects of any type:
  - [1, 2, 3, 4], ['yes', 'no', 'maybe']
- A special datum meaning nothing:
  - None

Variables

Variables are essentially symbols that stand in for a value you're using in a program. It's kind of container for values.

You can think of a variable as a label that has a name on it, which you tie onto a value:

```
my_string = 'Hello, World!'
my_flt = 45.06
my_bool = 5 > 9 #A Boolean value will return either True or False
my_list = ['item_1', 'item_2', 'item_3', 'item_4']
my_tuple = ('one', 'two', 'three')
my_dict = {'letter': 'g', 'number': 'seven', 'symbol': '&'}
```

```
x = y = z = 0
print(x)
print(y)
print(z)
```

```
j, k, l = "shark", 2.05, 15
print(j)
print(k)
print(l)
```

variables.py .

Mathematics

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

Mathematics (Cont ...)

Arithmetic Operations

- Addition
- Substraction
- Multiplication
- Division
- Modulus
- Exponent
- Floor Division

```
>>> 10 + 20
30
>>> 22 - 11
11
>>> 22 * 4
88
>>> 22/11
2.0
>>> 22 % 10
2
>>> 10 ** 2
100
>>> 22 //2
11
```

Mathematics (Cont ...)

Comparison Operators

- ==
- !=
- >
- <
- >=
- <=

Assignment Operators

- =
- +=
- -=
- *=
- /=
- %=
- **=
- //=

A few Comparison Operators do work on strings

Built-In Functions

- print
- range
- id
- type
- str
- int
- float
- input

```
>>> id(5)
10914496
>>> a=5
>>> id(a)
10914496
>>> b = a
>>> id(b)
10914496
>>> b = 5.0
>>> id(b)
140559854727504
```

Strings

A string is a sequence of characters. A character is simply a symbol. For example, the English language has 26 characters. The same symbols - computer stores in numbers internally (0's and 1's) (encoding/decoding process - ASCII, Unicode)

```
my_string = 'Hello'
print(my_string)

my_string = "Hello"
print(my_string)

my_string = '''Hello'''
print(my_string)

# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
           the world of Python"""
print(my_string)
```

strings_one.py

```
Hello
Hello
Hello
Hello, welcome to
           the world of Python
```

Strings (Cont ...)

- Subsets of strings can be taken using the slice operator ([ ] and [:])
  - with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

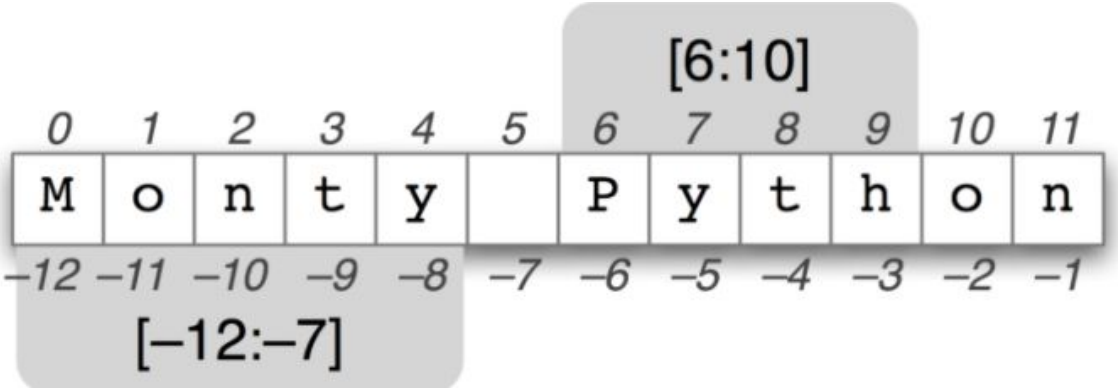
OPERATIONS

- Concatenation +
- Repetition with the help of asterisk *

strings_demo.py

```
#!/usr/bin/python3

str = 'Hello World!'

print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])      # Prints string starting from 3rd character
print (str * 2)      # Prints string two times
print (str + "TEST") # Prints concatenated string
```

Strings (Cont...)



strings_len.py

```
#!/usr/bin/python3

s = 'Hello World'
print (len(s))
print (type(s))

s = 'Monty Python'

print(s[6:10])
print(s[-12:-7])
```

```
11
<class 'str'>
Pyth
Monty
```

Strings - Important Methods

- s.lower(), s.upper() -- returns the lowercase or uppercase version of the string
- s.strip() -- returns a string with whitespace removed from the start and end
- s.isalpha()/s.isdigit()/s.isspace()... -- tests if all the string chars are in the various character classes
- s.startswith('other'), s.endswith('other') -- tests if the string starts or ends with the given other string
- s.find('other') -- searches for the given other string (not a regular expression) within s, and returns the first index where it begins or -1 if not found
- s.replace('old', 'new') -- returns a string where all occurrences of 'old' have been replaced by 'new'
- s.split('delim') -- returns a list of substrings separated by the given delimiter. T
- s.join(list) -- opposite of split()

Strings Quiz

- What is the output when following statements is executed ?
  - >>>"a"+"bc"
  - >>>"abcd"[2:]
- What arithmetic operators cannot be used with strings ?
  - +, -, *, **
- What is the output when following code is executed ?
  - >>>print r"\nhello"
  - print '\x97\x98'
- What is the output when following code is executed ?
  - >>>str1="helloworld"
  - >>>str1[::-1]
- What is the output of the following?
  - print("xyyzxyzxzxyy".count('yy'))
  - print("xyyzxyzxzxyy".count('yy', 1))
  - print("xyyzxyzxzxyy".count('xyy', 0, 100))

String Formatting - Text Displaying

```
# implicit order (default one)
default_order = "{}, {} and {}".format('Hari','Sadu','Naukari')
print('\n--- Default Order ---')
print(default_order)

# order using positional argument
positional_order = "{1}, {0} and {2}".format('Hari','Sadu','Naukari')
print('\n--- Positional Order ---')
print(positional_order)

# order using keyword argument
keyword_order = "{s}, {n} and {h}".format(h='Hari',s='Sadu',n='Naukari')
print('\n--- Keyword Order ---')
print(keyword_order)
```

```
--- Default Order ---
Hari, Sadu and Naukari

--- Positional Order ---
Sadu, Hari and Naukari

--- Keyword Order ---
Sadu, Naukari and Hari
```

String Format (Cont ...)

```
# formatting integers
#'Binary representation of 12 is 1100'
print("Binary representation of {0} is {0:b}".format(12))

 # formatting floats
#'Exponent representation: 1.566345e+03'
print("Exponent representation: {0:e}".format(1566.345))

# round off
#'One third is: 0.333'
print( "One third is: {0:.3f}".format(1/3))

# string alignment
#'|butter    |  bread   |       ham|'
print( "|{:<10}|{:^10}|{:>10}|".format('butter','bread','ham'))
```

```
Binary representation of 12 is 1100
Exponent representation: 1.566345e+03
One third is: 0.000
|butter    |  bread   |       ham|
```

Interaction with Script

input () - built in method is used to capture user / keyboard input

```
#!/usr/bin/python3
print ("Your Name:>")
name = input()

print("Thank you for interacting with Python Interpretor: {}".format(name))
```

```
Your Name:>
Hari Sadu
Thank you for interacting with Python Interpretor: Hari Sadu
```

Interact with Script (Cont ...)

```
#!/usr/bin/python3
'''Illustrate input and print.'''

applicant = input("Enter the applicant's name: ")
interviewer = input("Enter the interviewer's name: ")
time = input("Enter the appointment time: ")


print(interviewer, "will interview", applicant, "at", time)
```

```
Enter the applicant's name: John
Enter the interviewer's name: Hari
Enter the appointment time: 12:00
Hari will interview John at 12:00
```

interview.py

```
#!/usr/bin/python3

'''Conversion of strings to int before addition'''

xString = input("Enter a number: ")
x = int(xString)
yString = input("Enter a second number: ")
y = int(yString)
print('The sum of ', x, ' and ', y, ' is ', x+y, '.', sep='')
```

```
Enter a number: 23
Enter a second number: 78
The sum of 23 and 78 is 101.
```

addition.py

Making Decisions

```
#!/usr/bin/python3
# If the number is positive, we print an appropriate message

num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")

num = -1
if num > 0:
    print(num, "is a positive number.")
print("This is also always printed.")
```

```
3 is a positive number.
This is always printed.
This is also always printed.
```

if_demo.py

Making Decisions (Cont...)

```
#!/usr/bin/python3
# Program checks if the number is positive or negative
# And displays an appropriate message

num = 3

# Try these two variations as well.
# num = -5
# num = 0

if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

```
Positive or Zero
```

if_else_demo.py

Making Decisions (Cont...)

```
#!/usr/bin/python3
# In this program,
# we check if the number is positive or
# negative or zero and
# display an appropriate message

num = 3.4

# Try these two variations as well:
# num = 0
# num = -4.5

if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

```
Positive number
```

if_elif_demo.py

Python Lists

A list contains items separated by commas and enclosed within square brackets ([])

```
#!/usr/bin/python3

list     = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']


print (list )              # Prints complete list
print (list[0]  )          # Prints first element of the list
print (list[1:3] )         # Prints elements starting from 2nd till 3rd
print (list[2:] )          # Prints elements starting from 3rd element
print (tinylist * 2 )      # Prints list two times
print (list + tinylist )   # Prints concatenated lists

print (type(list))         # type
print(len(list))           # length
```

List is sequence

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
<class 'list'>
5
```

lists_demo.py

for - Loop

```
#!/usr/bin/python3

languages = ["C", "Java", "Perl", "Python", "Node.js", "JavaScript"]

for x in languages:
     print (x)
```

```
C
Java
Perl
Python
Node.js
JavaScript
```

for_demo.py

for - loop (Cont...)

```
#!/usr/bin/python3

edibles = ["ham", "spam","eggs","nuts"]
for food in edibles:
    if food == "spam":
        print("No more spam please!")
        break
    print("Great, delicious " + food)
else:
    print("I am so glad: No spam!")
print("Finally, I finished stuffing myself")
```

```
Great, delicious ham
No more spam please!
Finally, I finished stuffing myself
```

for_demo_adv.py

for_demo_adv_nospam.py

for - loop (Cont...)

```
#!/usr/bin/python3

edibles = ["ham", "spam","eggs","nuts"]
for food in edibles:
    if food == "spam":
        print("No more spam please!")
        break
    print("Great, delicious " + food)
else:
    print("I am so glad: No spam!")
print("Finally, I finished stuffing myself")
```

```
Great, delicious ham
No more spam please!
Finally, I finished stuffing myself
```

for_demo_adv.py

for_demo_adv_nospam.py

while - loop

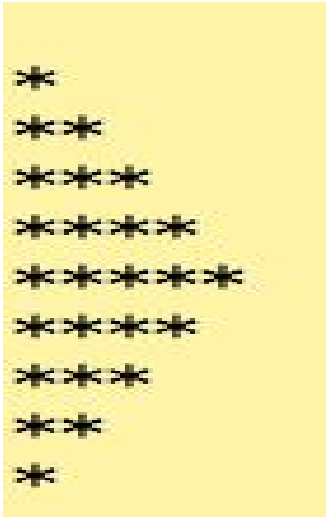
```
#!/usr/bin/python3

count = 0
while (count < 9):
   print ('The count is:', count)
   count = count + 1

print ("Good bye!")
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

while_demo.py

Task - 2

Write a script to print pattern like below. (try to use task_three.py)



Task - 3

Write a program - to count “even” and “odd” numbers which are less than 50.

(Try to use task_four.py)

Revisit Lists

- Comprehensions
- Lambda Functions
- Map
- Filters

```
>>> squares = []
>>> for x in range(10):
...     sq = x ** 2
...     squares.append(sq)
...
>>>
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
```

Old Way

```
>>> new_style = [x ** 2 for x in range(10)]
>>> new_style
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
```

New Way

```
>>> for x in [1, 3, 4, 2, 5, 6]:
...     for y in [2, 3, 5, 4, 10]:
...         if x != y:
...             print ("(", x, y, ")")
...
( 1 2 )
( 1 3 )
( 1 5 )
( 1 4 )
```

Old Way

New Way

```
>>> [(x, y) for x in [1, 3, 4, 2, 5, 6]  for y in [2, 3, 5, 4, 10] if x != y]
[(1, 2), (1, 3), (1, 5), (1, 4), (1, 10), (3, 2), (3, 5), (3, 4), (3, 10), (4, 2), (4, 3), (
 4), (2, 10), (5, 2), (5, 3), (5, 4), (5, 10), (6, 2), (6, 3), (6, 5), (6, 4), (6, 10)]
>>>
```

Handling Errors (cont...)

Two Types of Errors - [1] Syntax Error [2] Exceptions

Syntax Errors - Correct the Syntax!

Exceptions - Handle Exceptions, depends on the Logic of Application. Depends on Data of Application

IOError : If the file cannot be opened.

ImportError : If python cannot find the module

ValueError Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value

KeyboardInterrupt Raised when the user hits the interrupt key (normally Control-C or Delete)

EOFError Raised when one of the built-in functions (input() or raw_input()) hits an end-of-file condition (EOF) without reading any data

Handling Errors(Cont..)

```
>>> try:
...     d = 1/0
... except:
...     print ("I could not divide 1 by 0")
...
I could not divide 1 by 0
>>>
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> try:
...     d = 1/0
... except ZeroDivisionError:
...    print("Don't try to divide by 0")
...
Don't try to divide by 0
>>> try:
...     d = 10/2
... except ZeroDivisionError:
...    print("Don't try to divide by 0")
... else:
...    print("I am in else")
...
I am in else
>>> print(d)
5.0
>>>
```

```
#!/usr/bin/python3

my_lines = []
flag = True
while(flag):
    try:
        line = input()
        line.rstrip()
        my_lines.append(line)
    except EOFError:
        flag = False

print ("Let us print list of lines")

for l in my_lines:
    print (l)
```

handle_eof.py

Task - 4

Develop a script to take two numbers as input. Divide first number by second number. It should handle division by zero. The interaction of the script should look something similar to following

$divide.py

Enter First Number:10

Enter Second Number:0

You tried to divide 10 by zero

$*divide.py*

Enter First Number:10

Enter Second Number:2

The division is: 5.0

Task - 5

Problem: Write a script to find the factors of any given number

Task - Name the script as - *number_factors.py* . The script takes number using command line argument. The script creates the list of factors using remainder function and *for* loop.

Sample Input

$ number_factors.py 100

Sample Output

The factors of 100 are: [1, 2, 5, 10, 20, 25, 50, 100]

Day - 2

Functions

Problem: Convert decimal number into Binary Number

```
11 = 5 * 2 + 1
 5 = 2 * 2 + 1
 2 = 2 * 1 + 0
 1 = 2 * 0 + 1

0 's binary expression is - 1
1 's binary expression is - 1

so 11 's binary expression is - 1011
```

```
def functionname( parameters ):
      "function_docstring"
      python statements - logic/code of function
      return [expression]
```

Functions

```
#!/usr/bin/python3

name = "Aegis"

def hellofunction(name=None):
    '''hello function'''
    if name:
       print ("Hello " + name )
    else:
      print ("Hello World!")

#hellofunction(name)
hellofunction()
```

function_hello.py

```
#!/usr/bin/python3
def decToBin(n):
    if n==0:
         return '0'
    else:
        return decToBin(int(n/2)) + str(n%2)

d = decToBin(11)

print (d)
```

function_dec2binary.py

Task - 6

Complete “simple_calculator.py” script

Task - 7

Work on - palindrome_io.py - Take word as argument and verify if it's palindrom or not.

Interacting with Files

Open - function in Python. This helps with all file operations

```
file_object = open ("filename", "mode")

Mode is one of the following:
   r  : Read mode which is used when the file is only being read
   w  : Write mode which is used to edit and write new information to the
        file (any existing files with the same name will be erased when this mode is activated)
   a  : Appending mode, which is used to add new data to the end of the file;
       that is new information is automatically amended to the end
   r+ : Special read and write mode, which is used to handle both actions when working with a file
```

- Read the file - all data, line by line, in blocks
- Write the File
- Append to the file

f = open("filename",mode = 'r',encoding = 'utf-8')

Interacting with Files (Cont...)

simple_file_creation.py

```
#!/usr/bin/python3

f = open ('simple_data.txt', 'w')

f.write("Hello to writing data into file")
f.write("Line ------2")
f.write("Line ------3")
f.close()
```

simple_data.txt

```
Hello to writing data into fileLine ------2Line ------3
```

simple_file_read.py

```
#!/usr/bin/python3

f = open('simple_data.txt', 'r')

print (f.read())
f.close()
```

Interacting with Files (Cont ...)

You need to use close() - to close the file properly, else you may corrupt data.

```
try:
   f = open("sample.txt",encoding = 'utf-8')
   # perform file operations
   # again perform file operations
finally:
   f.close()
```

Python has “with” - using “with” can help to ignore explicit “close()” calls.

```
with open("sample.txt",encoding = 'utf-8') as f:
   # perform file operations
   # again perform file operations
```

Interacting with Files (Cont...)

```
#!/usr/bin/python3

def do_stuff_that_fails():
     print("I actually failed to work")

def do_stuff_when_it_doesnt_work():
     print("Tried but it did not work")

try:
    with open('my_file_which_do_not_exist') as f:
        read_data = f.read()
        print (read_data)

        do_stuff_that_fails()
except (IOError, OSError) as e:
    do_stuff_when_it_doesnt_work()
```

good_deal.py

Interacting with Files (Cont ...)

close(): Close an open file. It has no effect if the file is already closed.
detach(): Separate the underlying binary buffer from the TextIOBase and return it.
fileno(): Return an integer number (file descriptor) of the file.
flush(): Flush the write buffer of the file stream.
read(n): Read atmost n characters form the file. Reads till end of file if it is negative or None.
readable(): Returns True if the file stream can be read from.
readline(n=-1): Read and return one line from the file. Reads in at most n bytes if specified.
readlines(n=-1): Read, return a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset,from=SEEK_SET): Change the file position to offset bytes, in reference to from (start, current, end).
seekable(): Returns True if the file stream supports random access.
tell(): Returns the current file location.
truncate(size=None): Resize the file stream to size bytes. If size is not specified, resize to current location.
writable(): Returns True if the file stream can be written to.
write(s): Write string s to the file and return the number of characters written.
writelines(lines): Write a list of lines to the file.

Task - 8

Read File - [1] Number of lines [2] Number of words

Copy File - Read “A.txt” file and Copy the content to “B.txt” using Python Programming

Read TWiki Log file - print activity of particular user

Modules

- Modules are Python files with the .py extension.
- These files implement a set of functions and can have python statements.
- The modules can be imported from other modules using the import command.

The modules are best way to share your work, your tools with others.

Have a look at:

- https://docs.python.org/3/library/
- https://docs.python.org/2/library/

Modules (cont...)

```
#!/usr/bin/python3

def helloworld():
   print ("Hello World!")

def goodbye():
   print ("Good Bye Dear!")
```

hello_module.py

```
#!/usr/bin/python3
from hello_module import goodbye

print ("------I")
print (goodbye)

print ("------II")
goodbye()
```

test_hello_module.py

```
------I
<function goodbye at 0x7f538a412ea0>
------II
Good Bye Dear!
```

Modules (Cont...)

```
>>> import hello_module
>>> help(hello_module)
Help on module hello_module:

NAME
    hello_module

FUNCTIONS
    goodbye()

    helloworld()

FILE
    /datascience/hello_module.py


>>> dir(hello_module)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name
orld']
>>>
>>> from hello_module import goodbye
>>> help(goodbye)
Help on function goodbye in module hello_module:

goodbye()

>>> goodbye()
Good Bye Dear!
>>>
```

Modules (Cont ...)

```
>>> import sys
>>> sys.version
'3.5.2 (default, Nov 17 2016, 17:05:23) \n[GCC 5.4.0 20160609]'
>>> sys.version_info
sys.version_info(major=3, minor=5, micro=2, releaselevel='final', serial=0)
>>>
```

```
#!/usr/bin/python3
# import module sys to get the type of exception
import sys

randomList = ['a', 0, 2]

for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!",sys.exc_info()[0],"occured.")
        print("Next entry.")
        print()
print("The reciprocal of",entry,"is",r)
```

```
The entry is a
Oops! <class 'ValueError'> occured.
Next entry.

The entry is 0
Oops! <class 'ZeroDivisionError'> occured.
Next entry.

The entry is 2
The reciprocal of 2 is 0.5
```

one_more_exception.py

Module (Cont ...)

```
import subprocess
def disk (partition="/"):
     info = subprocess.call(["df", partition])
```

monitor.py

```
#!/usr/bin/python3

import subprocess
def disk (partition="/"):
     info = subprocess.call(["df", partition])

if __name__ == '__main__':
    import sys
    disk(sys.argv[1])
```

new_monitor.py

```
root@d5fc7cce17b6:/datascience# ./new_monitor.py /home
Filesystem     1K-blocks     Used Available Use% Mounted on
none            61890340 24866440  33856976  43% /
root@d5fc7cce17b6:/datascience#
root@d5fc7cce17b6:/datascience#
root@d5fc7cce17b6:/datascience#
root@d5fc7cce17b6:/datascience# python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from new_monitor import disk
>>> disk("/home")
Filesystem     1K-blocks     Used Available Use% Mounted on
none            61890340 24866440  33856976  43% /
```

Modules (Cont ...)

- Where to write/develop module files? Which directory
  - Role of PYTHONPATH,
  - Role of sys.path
  - HOME directory - ~/.local/lib/pythonX.X/site-packages

A few Libraries/Modules - sys

- sys.argv - Command line Arguments
- sys.path - directories where interpretor searches for modules/libraries

Sys.exit, sys.input, sys.out, sys.err etc

```
#!/usr/bin/python3

import sys

print (type(sys.argv))
print (sys.argv)

# Let us iterate through iterated via a for loop:

for i in range(len(sys.argv)):
    if i == 0:
        print ("script name: {}".format(sys.argv[0]))
    else:
        print ("{}. argument: {}".format(i,sys.argv[i]))
```

A few Libraries/Modules - logging

What's Logging? How does it help in case Application Crash? Servers Crash?

There are different importance levels - [1] debug, [2] info, [3] warning, [4] error and [5] critical

```
#!/usr/bin/python3
import logging

LOG_FILENAME = 'logging_example.out'
logging.basicConfig(
    filename=LOG_FILENAME,
    level=logging.DEBUG,
)

logging.debug('This message should go to the log file')
```

A few Libraries/Modules - logging (Cont...)

```
try:
    open('/path/to/does/not/exist', 'rb')
except (SystemExit, KeyboardInterrupt):
    raise
except Exception, e:
    logger.error('Failed to open file', exc_info=True)



ERROR:__main__:Failed to open file
Traceback (most recent call last):
  File "example.py", line 6, in <module>
    open('/path/to/does/not/exist', 'rb')
IOError: [Errno 2] No such file or directory: '/path/to/does/not/exist'
```

Dictionaries

Also known as:

- *Associate Array*
- *Map*
- *Hash Map*
- *Unordered Map*

Dictionaries (Cont …)

```
#!/usr/bin/python3

d1 = {}
print ("----------I")
print (type(d1))

d2 = {'one': 1, 'two':2}
print ("----------II")
print (d2)
print ("----------III")
print (type(d2))

d3 = dict(one=2, three=4)
print ("----------IV")
print (d3)
print (type(d3))

d4 = dict([(1, 2), (3, 4)])
print ("----------V")
print (d4)

d5 = dict({1:2, 3:4})
print ("----------V")
print (d5)
```

```
----------I
<class 'dict'>
----------II
{'one': 1, 'two': 2}
----------III
<class 'dict'>
----------IV
{'one': 2, 'three': 4}
<class 'dict'>
----------V
{1: 2, 3: 4}
----------V
{1: 2, 3: 4}
```

dictionaries.py

Dictionaries (Cont …)

- d.clear()
- d.copy()
- del k[d]
- dict.fromkeys(seq[, value])
- iteration/accessing elements of dictionaries
- for key in my_dictionary:
- for key, value in my_dictionary.items():
- for value in my_dictionary.values():
- iter(d)
- len(d)
- d.keys()
- d.values()
- d.items()

Task - 9

- Create a List - "A", elements are integers. List may have repeated elements
- Create new "B", which has same element from "A" but elements of "B" are not repeated elements
- Develop a function - to take generic list and return sorted, unique element list. Let us assume the numbers are integers

Task - 10

Develop A Phone-Book using Dictionaries. Functions similar to following:

$dictionaries_maps.py
2
Hari 9850111111
Sadu 9850222222
Enter Names for which you would like to query numbers
John
Hari
Sadu

Printing Phone Book for Queried Names
For John Number not found
Hari=9850111111
Sadu=9850222222

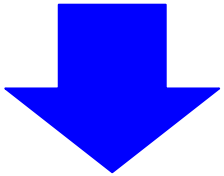
Task - 11

Create Dictionary - Take input as Integer, say it's "N". The keys are list of integers upto "N". The values for these keys are factors of the keys.

How about using dictionaries to improve the speed of program calculations?

Caching or Memoization

```
def some_function (arg1, arg2, arg3,..., argN):
        result = some computation involving arg1, arg2, arg3, ... argN
        return result
```



```
catche = {}
def some_function_with_catching(arg1, arg2, arg3, ..., argN):
        key = str(arg1) + str(arg2) + str(arg3) + .... + str(argN)
        if key in cache:
              return catche[key]
        else:
             result = same computation involving arg1, arg2, arg3, ..., argN
             catche[key]=result
             return result
```

Fibonacci Numbers

The Fibonacci Sequence is the series of numbers:

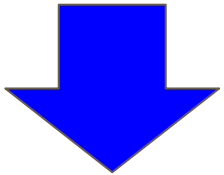
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

The next number is found by adding up the two numbers before it.

- The 2 is found by adding the two numbers before it (1+1)
- The 3 is found by adding the two numbers before it (1+2),
- And the 5 is (2+3),
- and so on!

Caching or Memoization - Fibonacci Numbers

```
def fib(n):
    return n if n < 2 else fib(n-2) + fib(n-1)
```



```
__fib_cache = {}
def fib(n):
    if n in __fib_cache:
        return __fib_cache[n]
    else:
        __fib_cache[n] = n if n < 2 else fib(n-2) + fib(n-1)
        return __fib_cache[n]
```

Regular Expressions

Let us look at some regular expression examples: Define some rules

- Write a letter "a" at least once
- Append to this the letter "k" exactly three times
- Append to this the letter "c" any even number of times
- Optionally, write the letter "d" at the end

Examples of such strings are:

- aaaakkkccccd
- aakkkcc

Regular Expression (Cont …)

There are infinitely many such strings which satisfy above rules.

Regular Expressions are merely a shorthand way of expressing these sets of rules

- Regex are text matching patterns described with a formal syntax
- The patterns which are executed on text as input to produce either matching subset or modified version of original text
- Regular Expression is kind of programming language itself
- "re" module provides this functionality in Python Programming

Your friendship with re will always add advantage to your skills if you ever need to deal with Text Processing in your project.

Regular Expression (Cont ...)

simple_rule.py .

```
#!/usr/bin/python3
import re

text = 'aaaabbbbbccccccd'
pattern = 'a+kkk(cc)+d?'
match = re.match(pattern, text)

if match:
   print ("I am satisfying the Rules")
else:
   print ("Nope - The rules are not satisfied")
```

```
Nope - The rules are not satisfied
```

Regular Expressions (Cont ...)

```
#!/usr/bin/python3
import re

pattern = 'Hello'
text = 'Hello Python Folks, How are you?'

match = re.search(pattern, text)

s = match.start()
e = match.end()

print('Found "{}\n" in "{}\n" from {} to {} ("{}")'.format(match.re.pattern, match.string, s, e, text[s:e]))
```

```
Found "Hello
" in "Hello Python Folks, How are you?
" from 0 to 5 ("Hello")
```

simple_match.py

Regular Expression (Cont …)

Python supports compilation of pattern - thats efficient to compile the pattern and use it. The compile() function converts an expression string into a RegexObject.

```
#!/usr/bin/python3
import re

pattern = 'Hello'
re_pattern = re.compile(pattern)
text = 'Hello Python Programmers, How are you?'

print (type(pattern))

print('Seeking "{}"'.format(re_pattern.pattern))
match = re.search(re_pattern, text)
print (type(match))

s = match.start()
e = match.end()

print('Found "{}"\nin "{}"\nfrom {} to {} ("{}")'.format(match.re.pattern, match.string, s, e, text[s:e]))
```

simple_compiled.py

```
<class 'str'>
Seeking "Hello"
<class '_sre.SRE_Match'>
Found "Hello"
in "Hello Python Programmers, How are you?"
from 0 to 5 ("Hello")
```

Regular Expression (Cont ...)

```
#!/usr/bin/python3
import re

text = 'abbaaabbbbaaaaa'

pattern = 'ab'

for match in re.findall(pattern, text):
    print (type(match) )
    print ('Found "%s"' % match)
```

```
<class '_sre.SRE_Match'>
Found "ab" at 0:2
<class '_sre.SRE_Match'>
Found "ab" at 5:7
```

find_all.py

```
<class 'str'>
Found "ab"
<class 'str'>
Found "ab"
```

find_iter.py

```
#!/usr/bin/python3

import re

text = 'abbaaabbbbaaaaa'

pattern = 'ab'

for match in re.finditer(pattern, text):
    print (type(match))
    s = match.start()
    e = match.end()
    print ('Found "%s" at %d:%d' % (text[s:e], s, e))
```

Regular Expression (Cont ...)

A Few Rules: Commonly Used RegEx symbols

symbol	Meaning	Example Pattern	Example Matches
*	Matches Preceding Char, Subexpression, or bracked char 0 or more times	*ab*	aaaaaa, aaabbbb, bbbb
+	Matches Preceding Char, Subexpression, or bracked char 1 or more times	a+b+	aaaaab, aaabbbb, abbbb
[]	Matches any char within bracket	[A-Z]*	APPLE, CAPITAL,
()	A groupd subexpression	*(ab)*	aaabaab, abaaab
{m, n}	Matches the preceding character, subexpression, or bracketed chars between m and n times	a{2,3}b{2,3}	
[^]	Matches any single character that is not in the brackets	[^A-Z]*	aaple
		Matches any char, or subexpression, separated by	
.	Matches any single charector	b.d	bed, bzd, b$d
^	Beging of line	^a	apple, an,
|An Escape Char			
$	Used for end of line char	[A-Z]*[a-z]*$	ABCabc, zzzyz, Bob

Matching Codes

Code	Meaning
\d	a digit
\D	a non-digit
\s	whitespace (tab, space, newline, etc.)
\S	non-whitespace
\w	alphanumeric
\W	non-alphanumeric

Regular Expression (Cont …)

Mostly used functions from re module,

- compile(pattern, flags=0) – it compiles a regular expression pattern into a regular expression object, which can be used for matching using the match and search methods.
- match(pattern, string, flags=0) – if zero or more characters from the beginning of the string match, it returns a Match object, otherwise, it returns None.
- search(pattern, string, flags=0) – similar to match(), but it scans all the string, not only it's beginning.
- sub(pattern, repl, string, count=0, flags=0) - Return the string obtained by replacing the leftmost non-overlapping occurrences of the pattern in string by the replacement repl. repl can be either a string or a callable; if a string, backslash escapes in it are processed. If it is a callable, it's passed the match object and must return a replacement string to be used

Regular Expression (Cont ...)

```
#!/usr/bin/python3
import re

line = 'The   fox jumped   over    the        log'
pattern = re.compile('\s+')

line = re.sub(pattern, '_', line)
print (line)
```

```
The_fox_jumped_over_the_log
```

How about following expression?

```
re.sub('\s{2,}', ' ', line)
```

remove_multiple_spaces.py

Task - 12

- Remove starting spaces in line
- Remove ending spaces in line
- How about removing all digits from line?

Develop Regular Expression to extract numbers (float, integers) from given text string. The numbers can be in any of the following format

'10.5', '-10.5', '- 10.5', '+ .2e10', ' 1.01e-2', ' 1.01', '-.2', ' 456', ' .123'

Class

```
#!/usr/bin/python3
class MyClass(object):
    variable = "myvalue"

    def function(self):
        print("This is a message inside the class.")


myobj = MyClass()

print (type(myobj))
print (myobj.variable)


yourobj =  MyClass()
print (yourobj.variable)

yourobj.variable = "yourvalue"

print (yourobj.variable)
yourobj.function()
```

```
<class '__main__.MyClass'>
myvalue
myvalue
yourvalue
This is a message inside the class.
```

my_class.py

Class (Cont ...)

- Classes are essentially a template to create your objects.
- init is the constructor for a class.
  - The self parameter refers to the instance of the object

```
#!/usr/bin/python3
class MyClass:
    variable = 'myvalue'
    def __init__(self, value = None):
        if value:
           self.variable = value


    def function(self):
        print("This is a message inside the class.")

    def  __repr__ (self):
         return "I am representation"

myobj = MyClass("Hello")

print (type(myobj))
print (myobj.variable)

print (myobj)
```

```
<class '__main__.MyClass'>
Hello
I am representation
```

my_advance_class.py

Class (Cont …)

```
class Person(object):
    def __init__(self, name, age):
          self.name = name
          self.age  = age

    def __repr__(self):
        return "Name: %s , Age: %d \n"  % (self.name, self.age)
```

Person.py

```
#!/usr/bin/python3
from  Person import  Person


def byAge(Person):
   return Person.age

p1  = Person("Doland Trump", 70)
p2  = Person("Barack Obama", 55)
p3  = Person("G Bush", 62)
p4  = Person("Bill Clinton", 54)
p5  = Person("Ronald Reagan", 77)



presidents  = [p1, p2, p3, p4, p5]

print (presidents)

sorted_presidents = sorted(presidents,  key=byAge)

print  (sorted_presidents)
```

sort_person.py

```
[Name: Doland Trump , Age: 70
, Name: Barack Obama , Age: 55
, Name: G Bush , Age: 62
, Name: Bill Clinton , Age: 54
, Name: Ronald Reagan , Age: 77
]
[Name: Bill Clinton , Age: 54
, Name: Barack Obama , Age: 55
, Name: G Bush , Age: 62
, Name: Doland Trump , Age: 70
, Name: Ronald Reagan , Age: 77
]
```

Task - 13

- Create Class - "City"
- Attributes
  a. Population
  b. Country
  c. Name
- Develop Script - to sort Cities (City instances) by Population
- Add GDP to City Class
  a. sort by GDP

Packages

- Help structure Python's module namespace by using "dotted module names".
  - The module name *package_a.mod_b* designates a submodule named *mod_b* in named *package_a*.
- Use of dotted module names saves the developers of multi-module packages from having to worry about each other's global variable names.

Let us assume we have the following directory structure. Here, hello.py & monotor.py are same modules described in *Module* section, and init.py is an empty file:

Packages (Cont …)

```
mypackage
|-- __init__.py
|-- hello_module.py
`-- monitor.py

0 directories, 3 files
```

init.py helps Python to treats this directory (*/mypackage*) as package directory.

```
>>> from mypackage import hello
>>> from mypackage import monitor
>>> dir(monitor)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__
s']
>>> dir(hello)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__
orld']
>>> hello.helloworld()
Hello World!
>>> monitor.disk("/home")
Filesystem     1K-blocks     Used Available Use% Mounted on
none            61890340 24866440  33856976  43% /
>>>
```

Task - 13

Q1. Create Package - which has modules to

- List factors of a given number
- Calculate HCF of given numbers
- Calculate the Factorial

Thank you