# Day - Three

# Day-3 Agenda

- Read and write operations for files - Interacting with Files
- Switch or Case statements
- Introductions to Functions and examples with application
- Modules
- Packages
- Config parser library
- Database - connection (MySQL)

# Interacting with Files

A **computer file** is a computer resource for recording data discretely in a computer storage device. Just as words can be written to paper, so can information be written to a computer file.

What all can Python open as File?

If your computer considers “something” as a file and can access that as a file, Python can open it.

# Interacting with Files (Cont...)

```
>>> f = open('tutor.zh.utf-8', 'r', encoding='utf-8')
>>> f.name
'tutor.zh.utf-8'
>>> f.encoding
'utf-8'
>>> f.mode
'r'
>>>
```

```
>>> import locale
>>> locale.getpreferredencoding()
'ANSI_X3.4-1968'
>>>
```

```
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
```

# Interacting with Files (Cont …)

```
>>> f = open('japanese.txt', 'r', encoding='utf-8')
>>> f.read()
'あたらしい記事きじを書かこうという気持きもちになるまで長ながい時間じかんがかかった。書かきた
さんあったけれど、息子むすこを産うんだ後あとは書かく時間じかんがあまりなかった。\n\nIt took m
e to find the motivation to write a new article. There were many things I wanted to write abo
n't have much time for it after having given birth to my son.\n'
>>>
>>>
```

```
>>> f = open('japanese.txt', 'r', encoding='utf-8')
>>> f.read(10)
'あたらしい記事きじを'
>>> f.tell()
30
>>> f.seek(300)
300
>>> f.tell()
300
>>> f.read(10)
' to find t'
>>> f.seek(0)
0
>>> f.read(10)
'あたらしい記事きじを'
>>>
```

# Interacting with Files (Cont …)

The seek() method moves to a specific byte position in a file.
The read() method can take an optional parameter, the number of characters to read.

```
>>> f = open('japanese.txt', 'r', encoding='utf-8')
>>> f.read(10)
'あたらしい記事きじを'
>>> f.tell()
30
>>> f.seek(300)
300
>>> f.tell()
300
>>> f.read(10)
' to find t'
>>> f.seek(0)
0
>>> f.read(10)
'あたらしい記事きじを'
>>>
```

# Interacting with Files (Cont …)

```
>>> f.close()
>>>
>>> f
<_io.TextIOWrapper name='japanese.txt' mode='r' encoding='utf-8'>
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
>>>
```

# Interacting with Files (Cont …)

Always try to use "with" for your file operations

```
$ more file_with_demo.py
with open('japanese.txt', 'r', encoding='utf-8') as a_file:
    a_file.seek(340)
    file_str = a_file.read(30)
    print (file_str)
```

```
$ python3 file_with_demo.py
rticle. There were many things
$ _
```

file_with_demo.py

When the with block ends, *Python calls* a_file.close() *automatically.*

# Interacting with Files (Cont...)

Reading Data one Line at a Time - Let us ignore that newline Chars (\n, \r) discussion here. Python takes care of it for us.

file_read_lines.py

```
$ more file_read_lines.py
line_number = 0
with open('men_100meter.txt', 'r', encoding='utf-8') as a_file:
    for line in a_file:
        line_number +=1
        print ("{:>4} {}".format(line_number, line.rstrip()))
        if line_number > 10:
            break
```

```
$ python3 file_read_lines.py
   1 1 9.58 +0.9 Usain Bolt JAM 21.08.86 1 Berlin16.08.2009
   2 2 9.63 +1.5 Usain Bolt JAM 21.08.86 1 London05.08.2012
   3 3 9.69 +0.0 Usain Bolt JAM 21.08.86 1 Beijing 16.08.200
   4 3 9.69 +2.0 Tyson Gay USA 09.08.82 1 Shanghai 20.09.2009
   5 3 9.69 -0.1 Yohan Blake JAM 26.12.89 1 Lausanne 23.08.2012
   6 6 9.71 +0.9 Tyson Gay USA 09.08.82 2 Berlin16.08.2009
   7 7 9.72 +1.7 Usain Bolt JAM 21.08.86 1rA New York City 31.05.2008
   8 7 9.72 +0.2 Asafa Powell JAM 23.11.82 1rA Lausanne 02.09.2008
   9 9 9.74 +1.7 Asafa Powell JAM 23.11.82 1h2 Rieti 09.09.2007
  10 9 9.74 +0.9 Justin Gatlin USA 10.02.82 1 Ad-Dawhah 15.05.2015
  11 11 9.75 +1.1 Yohan Blake JAM 26.12.89 1 Kingston 29.06.2012
```

# Quiz

**What is the output of following code?**

```
#!/usr/bin/python3

f = None
for i in range (5):
    with open("data.txt", "w") as f:
        if i > 2:
            break

print (f.closed)
```

# Interacting with Files (Cont …)

- "Write" mode will overwrite the file. Pass mode='w' to the open()function.
- "Append" mode will add data to the end of the file. Pass mode='a' to the open() function.

```
$ more file_write_demo.py

# write to file
with open('test.log', mode='w', encoding='utf-8') as a_file:
    a_file.write("This is Python Class")

# Verify
with open('test.log', mode='r', encoding='utf-8') as a_file:
    print (a_file.read())

# Let us try append
with open('test.log', mode='a', encoding='utf-8') as a_file:
    a_file.write("This is Seriously Python Class")


# Verify
with open('test.log', mode='r', encoding='utf-8') as a_file:
    print (a_file.read())
```

file_write_demo.py

# Interacting with Files (Cont ...)

```
$ python3 file_write_demo.py
This is Python Class
This is Python ClassThis is Seriously Python Class
$
```

# Quiz

**What's the output of following code?**

```
#!/usr/bin/python3

for i in range (5):
    with open("data.txt", "w") as f:
        f.write("Number: ", i)



with open("data.txt", "w") as fr:
    print (fr.read())
```

1

```
Number: 4
```

2

```
TypeError: write() takes exactly one argument (2 given)
```

3

```
Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
```

4

```
True
```

# Quiz

**What is the output of following code?**

```
$ more data.txt
1
2
$ more t.py
#!/usr/bin/python3

f = open("data.txt", "r")
for i in [100,200]:
    f.readline()

for i in [100,200]:
    print (f.readline())
```

# Task 1

Develop a script “olyimpics_100meter.py” to read file - “men_100meter.txt”. The

script should create dictionary - with keys as “Player Names” with values as

Lists of Speed Seconds.

Example:

```
$ ./olyimpics_100meter.py
{'Usain Bold': [9.58, 9.63, 9.69, 9.72], 'Tyson Gay': [9.69, 9.71, 9.77, 9.77]}
```

# Task 2

Develop a script “python_copy.py” . The script will copy source file (text) into destination file.

Example: python_copy.py source.txt destination.txt

If source.txt do not exists in filesystem - the script should throw an error and make an exit.

If destination.txt file exists in filesystem - the script should throw an error and make an exit.

**Hint - Use “import os” (use os module to verify existance of files)**

# Functions

Problem: Convert Decimal number into Binary Number

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

0 is binary expression of 0
1 is binary expression of 1

So 11 has binary expresion = 1011
```

Division by 2 and collecting remainder at each step is repeated task

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

“function_docstring” is available in:

“functionname.__doc__

# Function (Cont ...)

```
#!/usr/bin/python3

name = "Hari Sadu"

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

hellofunction(name)
#hellofunction()
```

function_hello.py

hellofunction (name) or hellofunction() is called making call to function

# Function (cont ...)

```
#!/usr/bin/python3

name = "Hari Sadu"

def hellofunction(name=None):
    '''This is demonstration to define function.
       The function is not very great'''
    if name:
       print ("Hello " + name )
    else:
      print ("Hello World!")

print (hellofunction.__doc__)
```

```
$ ./function_demo_docstring.py
This is demonstration to define function.
       The function is not very great
$
```

function_demo_dockstring.py

# Function (Cont …)

The return statement is used to exit a function and go back to the place from where it was called

```
return expression
```

The expression is evaluated.  The value is returned by the function.

The absence of return statement returns “None”

# Function (Cont …)

```
#!/usr/bin/python3
def absolute_value(num):
        """This function returns the absolute
        value of the entered number"""

        if num >= 0:
                return num
        else:
                return -num

print(absolute_value(2))

print(absolute_value(-4))

print (absolute_value (0))
```

```
$ ./function_abs.py
2
4
0
```

function_abs.py

# Quiz

**Q 4. What's the output of following program?**

```
#!/usr/bin/python3
def foo():
    try:
        return 1
    finally:
        return 2
k = foo()
print(k)
```

# Quiz

```
#!/usr/bin/python3
import time
def foo():
    try:
        raise Exception("I know python!")
    except:
        time.sleep(10)
        return 1
    finally:
        return 2

k = foo()
print (k)
```

# Task - 3

Problem - Develop a simple calculator “simple_calculator.py”

We have provided above file - you need to modify, update code at “TODO” and “pass” word.

# Functions (Cont ...)

```
#!/usr/bin/python3

def zero():
    print ("You typed zero")

def sqr():
    print ("n is a perfect square")

def even():
    print ("n is an even number")

def prime():
    print ("n is a prime number")

options = {     0 : zero,
                1 : sqr,
                4 : sqr,
                9 : sqr,
                2 : even,
                3 : prime,
                5 : prime,
                7 : prime,
}
options.get(1)()
options.get(7)()
```

```
$ ./switch_python.py
n is a perfect square
n is a prime number
```

switch_python.py

This is also called dispatch table

# Quiz

Which of the following is a features of DocString?

a) Provide a convenient way of associating documentation with Python modules, functions, classes, and methods

b) All functions should have a docstring

c) Docstrings can be accessed by the __doc__ attribute on objects

d) All of the mentioned

# Quiz

**What is the output of following code?**

```
#!/usr/bin/python3
def printMax(a, b):
    if a > b:
        print(a, 'is maximum')
    elif a == b:
        print(a, 'is equal to', b)
    else:
        print(b, 'is maximum')
printMax(3, 4)
```

# Quiz

**What is the output of following script?**

```
#!/usr/bin/python3
def func(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is', c)

func(3, 7)
func(25, c = 24)
func(c = 50, a = 100)
```

# Functions (Cont ...)

## Recursion

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

d = decToBin(12)

print (d)
```

```
$ ./function_dec2binary.py
01100
```

function_dec2binary.py

# Functions (Cont ...)

## Arbitrary Number of Arguments

```
$ more function_arbitrary_arguments.py
def arithmetic_mean(first, *values):
    """ This function calculates the arithmetic mean of a non-empty
        arbitrary number of numerical values """

    return (first + sum(values)) / (1 + len(values))

print(arithmetic_mean(45,32,89,78))
print(arithmetic_mean(8989.8,78787.78,3453,78778.73))
print(arithmetic_mean(45,32))
print(arithmetic_mean(45))
```

```
$ python3 function_arbitrary_arguments.py
61.0
42502.3275
38.5
45.0
```

function_arbitrary_arguments.py

# Function (Cont ...)

## Arbitrary Number of Keyword Parameters

function_arbitrary_keyword.py

```
$ more function_arbitrary_keyword.py
def f(**kwargs):
    print(kwargs)

f()
f(de="German",en="English",fr="French")

def f2(a,b,x,y):
    print(a,b,x,y)


d = {'a':'append', 'b':'block','x':'extract','y':'yes'}
f2(**d)
```

```
$ python3 function_arbitrary_keyword.py
{}
{'de': 'German', 'en': 'English', 'fr': 'French'}
append block extract yes
```

# Modules (Cont … )

sys module = installed with Python package provides lots of useful functionality

```
#!/usr/bin/python3

import sys

print (sys.argv)

# Iterate via Loop

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

```
$ ./sys_arg.py one 1 two
['./sys_arg.py', 'one', '1', 'two']
script name:  ./sys_arg.py
1  : argument:  one
2  : argument:  1
3  : argument:  two
$
```

sys.exit() is important function call

**sys_arg.py**

# Task - 4

Problem - Modify “simple_calculator.py” to handle calculations. It should take input of operator and numbers via command line argument.

Example:

$simple_calculator.py + 15 20

35

$simple_calculator.py - 20 10

10

# Module vs Script

Modules can be used as script. Inside the script - __name__ is nothing but __main__ space.

```
#!/usr/bin/python3

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

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

if __name__ == '__main__':
    print ("I am script - calling goodbye via script")
    goodbye()
```

```
$ ./module_vs_script.py
I am script - calling goodbye via script
Good Bye Dear!
$
```

**module_vs_script.py**

# Module (Cont …)

```
#!/usr/bin/python3

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

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

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

**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% /
```

**new_monitor.py**

# 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 a package 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% /
>>>
```

# Configuration Files

- Any software, tool - “configuration” or “config” file is required to store
  - Parameters
  - Initial Settings
- Those tools can be - OS, Software Applications, Servers, User Applications etc

INI files are text files that store key/value pairs that are grouped in sections - these format files  initially used by Windows OS/Microsoft folks.

# Configuration Files (Cont ...)

sample_configuration.ini

Let us look at sample configuration file:

```
; This is sample configuration ini file
# The file was created during Python Course at Enterprise XYZ Inc.

[session]
title: The Python Programming Course
trainer: Hari Sadu
email: harry@example.com
version: 2.0

[ematter]
slides: 300

[code]
repository: http://github.com
```

Lines starting with semi-colon (;) or hash (#) are treated as comments. That is not visible when accessing the contents of the configuration file programmatically.

# ConfigParser (Cont ...)

config-parser-one.py

```
#!/usr/bin/python3
from configparser import ConfigParser

parser = ConfigParser()
parser.read('sample_configuration.ini')

print(parser.get('session', 'trainer'))
```

```
$ ./config-parser-one.py
Hari Sadu
```

# ConfigParser (Cont ...)

config-parser-two.py

```
#!/usr/bin/python3
from configparser import ConfigParser

parser = ConfigParser()
parser.read('sample_configuration.ini')

for section_name in parser.sections():
    print('Section:', section_name)
    print('  Options:', parser.options(section_name))
    for name, value in parser.items(section_name):
        print('  {} = {}'.format(name, value))
    print()
```

```
$ ./config-parser-two.py
Section: session
  Options: ['title', 'trainer', 'email', 'version']
  title = The Python Programming Course
  trainer = Hari Sadu
  email = harry@example.com
  version = 2.0

Section: ematter
  Options: ['slides']
  slides = 300

Section: code
  Options: ['repository']
  repository = http://github.com
```

# ConfigParser (Cont ...)

config-parser-three.py

```
#!/usr/bin/python3
from configparser import ConfigParser

parser = ConfigParser()
parser.read('sample_configuration.ini')

for candidate in ['session', 'class', 'city', 'ematter']:
    print('{:<12}: {}'.format(
        candidate, parser.has_section(candidate)))
```

```
$ ./config-parser-three.py
session     : True
class       : False
city        : False
ematter     : True
```

# ConfigParser (Cont …)

All section and option names are treated as strings, but option values can be strings, integers, floating point numbers, or Booleans. There are a range of possible Boolean values that are converted true or false. The following example file includes one of each.

```
from configparser import ConfigParser

try:
   parser = ConfigParser()
   parser.read('sample_configuration.ini')
except configparser.ParsingError as err:
   print('Could not parse:', err)

version = parser.getfloat('session', 'version')
print (type(version))

# similarly we can have methods:
#    getboolean
#    getint
```

config-parser-four.py

```
$ ./config-parser-four.py
<class 'float'>
```

# ConfigParser (Cont …)

You can also save/modify configuration files

```
#!/usr/bin/python3

import configparser
import sys

parser = configparser.ConfigParser()

parser.add_section('bug_tracker')
parser.set('bug_tracker', 'url', 'http://example.com/bugs')
parser.set('bug_tracker', 'username', 'dhellmann')
parser.set('bug_tracker', 'password', 'secret')
#parser.write(sys.stdout)

f = open('myconfig.ini', 'w')
parser.write(f)
f.close()
```

```
$ python3 config-parser-five.py
$ more myconfig.ini
[bug_tracker]
url = http://example.com/bugs
username = dhellmann
password = secret
```

# Quiz

**What is the output of following code?**

```
import configparser
import sys

parser = configparser.ConfigParser()

parser.add_section('name')
parser.set('first', 'Hari')
parser.set('last', 'Sadu')
parser.write(sys.stdout)
```

# Task - 5

Develop a script “create-mysql.py” which help create “mysql.ini configuration file using ConfigParser

```
$ more mysql.ini
[mysql]
user=root
password=welcome123
host=localhost
database=pythoncourse
```

# Databases

What is Database?

Language: SQL is a standard language for storing, manipulating and retrieving data in databases

Examples - MySQL, PostgreSQL, Oracle, SQLite

Reference - https://dev.mysql.com/doc/connector-python/en/

# MySQL

Let us focus on MySQL

```
$ mysql -u root -h localhost -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

# MySQL

```
For server side help, type 'help contents'

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)

mysql> use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| engine_cost               |
| event                     |
| func                      |
| general_log               |
```

# MySQL

```
mysql> select Host, User from user;
+-----------+------------------+
| Host      | User             |
+-----------+------------------+
| localhost | debian-sys-maint |
| localhost | mysql.session    |
| localhost | mysql.sys        |
| localhost | root             |
+-----------+------------------+
4 rows in set (0.00 sec)

mysql> select * from time_zone;
Empty set (0.00 sec)

mysql> desc time_zone;
+------------------+------------------+------+-----+---------+----------------+
| Field            | Type             | Null | Key | Default | Extra          |
+------------------+------------------+------+-----+---------+----------------+
| Time_zone_id     | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| Use_leap_seconds | enum('Y','N')    | NO   |     | N       |                |
+------------------+------------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql>
```

# MySQL

**Python Library/Connector Required to connect, retrieve, update data with MySQL database**

- MySQL Connector/Python - from Official MySQL Site
- https://pypi.python.org/pypi/mysqlclient (mysqlclient)
- https://github.com/farcepest/moist - Moist
- PyMySQL

# MySQL

```
mysql> create database pythoncourse;
Query OK, 1 row affected (0.00 sec)

mysql> use pythoncourse;
Database changed
mysql>
```

```
mysql> CREATE TABLE presidents (
    -> id INT(11) NOT NULL AUTO_INCREMENT,
    -> name VARCHAR(45) NOT NULL,
    -> age INT(11) NOT NULL,
    -> PRIMARY KEY (id)
    -> ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> desc presidents;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| name  | varchar(45) | NO   |     | NULL    |                |
| age   | int(11)     | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

# MySQL

```
mysql> INSERT INTO presidents (id, name, age) VALUES (1, 'Donalt T', 74);
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO presidents (name, age) VALUES ('Barack O', 54);
Query OK, 1 row affected (0.06 sec)

mysql> select * from presidents;
+----+----------+-----+
| id | name     | age |
+----+----------+-----+
|  1 | Donalt T |  74 |
|  2 | Barack O |  54 |
+----+----------+-----+
2 rows in set (0.00 sec)
```

```
mysql> select * from presidents ORDER BY age;
+----+----------+-----+
| id | name     | age |
+----+----------+-----+
|  2 | Barack O |  54 |
|  1 | Donalt T |  74 |
+----+----------+-----+
2 rows in set (0.00 sec)
```

# MySQL

```
$ 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.
>>> import mysql.connector
>>> cnx = mysql.connector.connect(user='root', password='welcome123', host='localhost', database='mysql')
>>> cursor = cnx.cursor()
>>> cursor.execute('SELECT VERSION()')
>>> data = cursor.fetchone()
>>> data
('5.7.19-0ubuntu0.16.04.1',)
>>> print ("Database version : {} ".format( data))
Database version : ('5.7.19-0ubuntu0.16.04.1',)
>>> cnx.close()
```

# LAB Assignment

Q1. Develop a script (use functions) - to find the standard deviation of numbers/stock prices.

Q. 2 Develop “twiki_analysis.py” script. The script takes one argument for username. The script prints “User do not exists if the user is not present in the log file”. If User is present then it’s activity is displayed on the script (maximum possible activity - topics viewed, from which IP activity was conducted”. The script will read “TWiki_Application.log” file

Requested to use - dictionary for cache purposes

# LAB Assignment

**Q. 3 Create a package called "myfinance". This package should have module called "csvmodule" (myfinance/csvmodule.py). You should be able perform operation like (you can enter any csv file as an argument to your script):**

```
from myfinance import csvmodule
numbers = csvmodule.countdigits("2017-18-statement.csv")
print ("The report has {} numbers".format(numbers))
```

**Q. 4 Read the Data from "AMD.txt". Develop script "high_closing_price.py" to find the high closing price of stack. The script should be generic - it should take argument as scrip/stock symbol (here pass AMD as argument)**

# LAB Assignment

Q. 5 You need to create database table by “firstname_lastname” after connecting to database. We have provided “todo_schema.sql” in “assignment directory. Use that file to create tables described below. Deliver the screen-shots of your work.

**project**

| Column | Type | Description |
|---|---|---|
| name | text | Project Name |
| description | text | Project Description |
| deadline | date | Due Date |

**task**

| Column | Type | Description |
|---|---|---|
| id | number | Uniq Task Identifier |
| priority | integer | Priority of the task |
| details | text | Task Description |
| status | text | Status |
| deadline | date | Due Date |
| completed_on | date | Completion Date |
| project | text | Task Belongs to Project |

# Thank you