

# File IO in Python

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

When we want to read from or write to a file we need to open it first.

When we are done, it needs to be closed, so that resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order.

1. Open a file
2. Read or write (perform operation)
3. Close the file

## Open file :

Python has a built-in function `open()` to open a file.

This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
Fd1 = open("Marvellous.txt",'r')
```

Search file in current directory as we provide relative path of file

```
Fd2 = open("/Users/marvellous/Desktop/Today/Marvellous.txt")
```

Search in specific path as we provide absolute path

We can specify the mode while opening a file.

In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file.

We also specify if we want to open the file in text mode or binary mode.

The default is reading in text mode. In this mode, we get strings when reading from the file.

On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

```
f = open("Marvellous.txt")
```

# equivalent to 'r' or 'rt'

```
f = open("Marvellous.txt",'w')
```

# write in text mode

```
f = open("LogoMarvellous.bmp",'r+b')
```

# read and write in binary mode

Unlike other languages, the character 'a' does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings).

Moreover, the default encoding is platform dependent.

In windows, it is 'cp1252' but 'utf-8' in Linux.

So, we must not also rely on the default encoding or else our code will behave differently in different platforms.

Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

```
f = open("Marvellous.txt",mode = 'r',encoding = 'utf-8')
```



## Python File Modes

| Mode | Description   |
|------|---|
| 'r'  | Open a file for reading. (default)  |
| 'w'  | Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.      |
| 'x'  | Open a file for exclusive creation. If the file already exists, the operation fails.                      |
| 'a'  | Open for appending at the end of the file without truncating it. Creates a new file if it does not exist. |
| 't'  | Open in text mode. (default)  |
| 'b'  | Open in binary mode.  |
| '+'  | Open a file for updating (reading and writing)  |

### Close File :

When we are done with operations to the file, we need to properly close the file. Closing a file will free up the resources that were tied with the file and is done using Python `close()` method. Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

```
f = open("Marvellous.txt",encoding = 'utf-8')
```

```
# perform file operations
```

```
f.close()
```

### Read data from file:

Consider below file that we refer for reading

Marvellous.txt

Marvellous Infosystems by Piyush Manohar Khairnar  
Karve Road Pune 411004  
Educating for better tomorrow..

To read a file in Python, we must open the file in reading mode.



There are various methods available for this purpose. We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

```
fd = open("Marvellous.txt",'r',encoding = 'utf-8')
```

```
fd.read(10)  # read the first 10 data
```

Output :

"Marvellous"

```
fd.read(12)  # read the next 12 data
```

Output :

"Infosystems"

```
fd.read()    # read in the rest till end of file
```

Output :

by Piyush Manohar Khairnar

Karve Road Pune 411004

Educating for better tomorrow..

We can change our current file cursor (position) using the seek() method. Similarly, the tell() method returns our current position (in number of bytes).

```
print("Current file position is",fd.tell())  # get the current file position
```

```
fd.seek(0)  # bring file cursor to initial position
```

```
print("Contents of Whole file")
```

```
print(fd.read())
```

## **Writing data into file :**

In order to write into a file in Python, we need to open it in write 'w', append 'a' or exclusive creation 'x' mode.

We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased.

Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

```
fd = open("Marvellous.txt",'w+a',encoding = 'utf-8')
```

```
fd.write("Python : Automation and Machine Learning\n")
```

```
fd.write("Angular : Web Development\n")
```

# File IO methods in Python

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



## Consider below application which demonstrates file IO

```
fd = open("Marvellous.txt",'r')

print("Information about file : ",fd)

print("Contents of Whole file")
print(fd.read())

print("Reading single line from file")
print(fd.readline())

print("Current file position is",fd.tell())    # get the current file position

fd.seek(0)    # bring file cursor to initial position

print("Contents of Whole file")
print(fd.read())

fd.close()

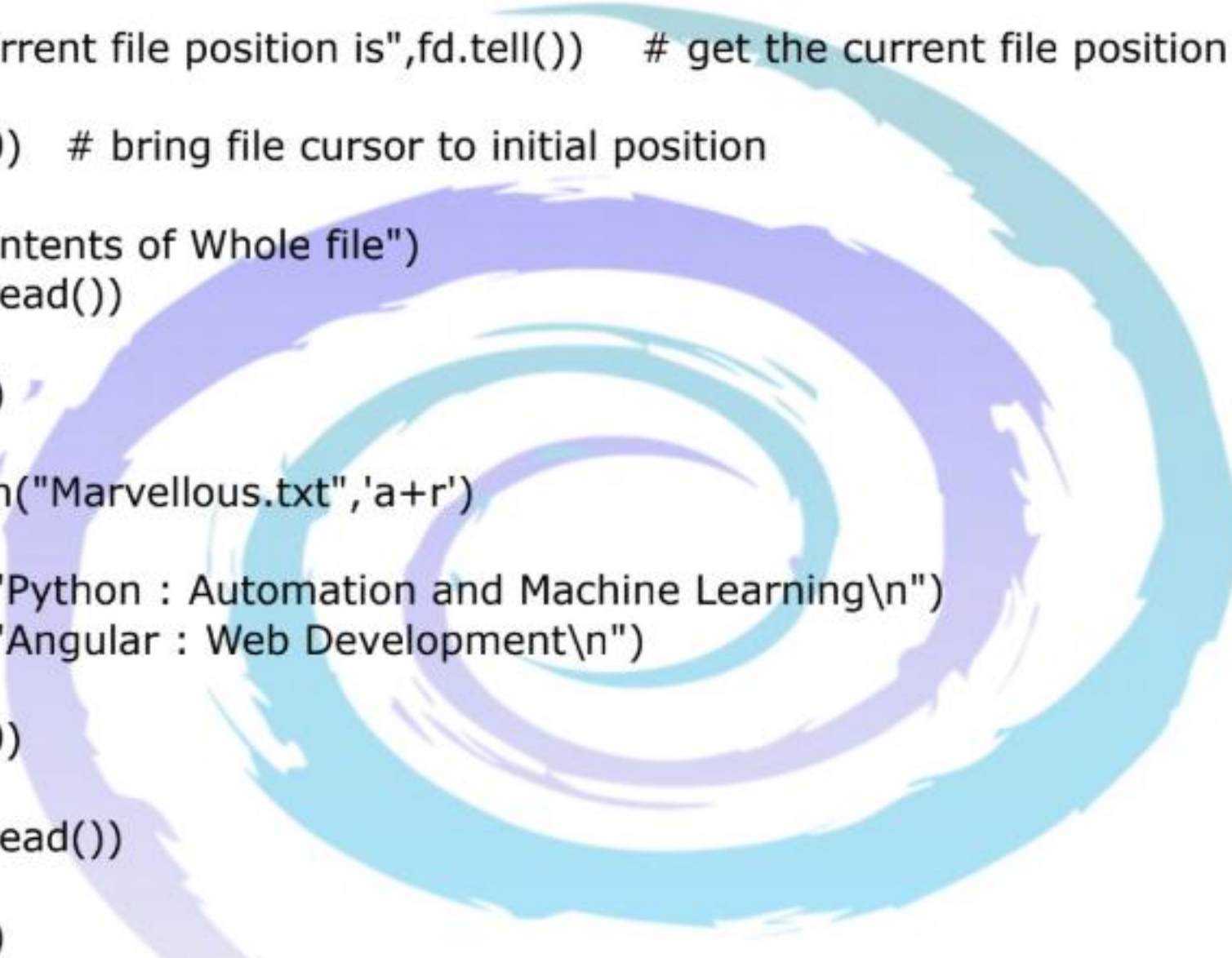
fd = open("Marvellous.txt",'a+r')

fd.write("Python : Automation and Machine Learning\n")
fd.write("Angular : Web Development\n")

fd.seek(0)

print(fd.read())

fd.close()
```



## Output of above application

```
MacBook-Pro-de-MARVELLOUS:Today marvellous$ python
FileIO.py
('Information about file : ', <open file 'Marvellous
s.txt', mode 'r' at 0x10397c5d0>)
Contents of Whole file
Marvellous Infosystems by Piyush Manohar Khairnar
Karve Road Pune 411004
Educating for better tomorrow..Python : Automation
and Machine Learning
Angular : Web Development

Reading single line from file

('Current file position is', 171)
Contents of Whole file
Marvellous Infosystems by Piyush Manohar Khairnar
Karve Road Pune 411004
Educating for better tomorrow..Python : Automation
and Machine Learning
Angular : Web Development
Marvellous Infosystems by Piyush Manohar Khairnar
Karve Road Pune 411004
Educating for better tomorrow..Python : Automation
and Machine Learning
Angular : Web Development
Python : Automation and Machine Learning
Angular : Web Development
```



# PIP

## PIP Install Packages

- PIP is a recursive acronym that stands for “PIP Installs Packages” or “Preferred Installer Program”.
- It’s a command-line utility that allows you to install, reinstall, or uninstall PyPI packages with a simple and straightforward command: `pip`.
- That means it’s a tool that allows us to install and manage additional libraries and dependencies that are not distributed as part of the standard library.
- Package management is so important that `pip` has been included with the Python installer since versions 3.4 for Python 3 and 2.7.9 for Python 2, and it’s used by many Python projects, which makes it an essential tool for every Python developer.

### Install PIP

If you’re using Python 2.7.9 (or greater) or Python 3.4 (or greater), then PIP comes installed with Python by default.

If you’re using an older version of Python, you’ll need to use the installation steps below. Otherwise, skip to the bottom to learn how to start using PIP.

### Install PIP on Windows (Windows 7, Windows 8.1, and Windows 10)

1. Download `get-pip.py` to a folder on your computer (<https://bootstrap.pypa.io/get-pip.py>).
2. Open a command prompt and navigate to the folder containing `get-pip.py`.
3. Run the following command: `python get-pip.py`
4. Pip is now installed.

You can verify that Pip was installed correctly by opening a command prompt and entering the following command:

**`pip -V`**

### How to Install PIP on Linux

If your Linux distro came with Python already installed, you should be able to install PIP using your system’s package manager.

Advanced Package Tool (Python 2.x)

**`sudo apt-get install python-pip`**

Advanced Package Tool (Python 3.x)

**`sudo apt-get install python3-pip`**



# Scheduling in Python

- Schedule is in-process scheduler for periodic jobs that use the builder pattern for configuration.
- Schedule lets you run Python functions (or any other callable) periodically at pre-determined intervals using a simple, human-friendly syntax.
- Schedule Library is used to schedule a task at a particular time every day or a particular day of a week.
- We can also set time in 24 hours format that when a task should run.
- Basically, Schedule Library matches your systems time to that of scheduled time set by you.
- Once the scheduled time and system time matches the job function (command function that is scheduled ) is called.

Before using schedule library we have to install it as

Installation of Schedule module

**`pip install schedule`**

## **Classes and methods from schedule library**

### **`schedule.Scheduler class`**

`schedule.every(interval=1)`

Calls every on the default scheduler instance.  
Schedule a new periodic job.

`schedule.run_pending()`

Calls run\_pending on the default scheduler instance.  
Run all jobs that are scheduled to run.

`schedule.run_all(delay_seconds=0)`

Calls run\_all on the default scheduler instance.  
Run all jobs regardless if they are scheduled to run or not.

`schedule.idle_seconds()`

Calls idle\_seconds on the default scheduler instance.

`schedule.next_run()`

Calls next\_run on the default scheduler instance.  
Datetime when the next job should run.

`schedule.cancel_job(job)`

Calls cancel\_job on the default scheduler instance.  
Delete a scheduled job.



## **schedule.Job(interval, scheduler=None) class**

A periodic job as used by Scheduler.

There are two parameters as

interval: A quantity of a certain time unit

scheduler: The Scheduler instance that this job will register itself with once it has been fully configured in Job.do().

Basic methods for Schedule.job

### at(time\_str)

Schedule the job every day at a specific time.

Calling this is only valid for jobs scheduled to run every N day(s).

Parameters: time\_str – A string in XX:YY format.

Returns: The invoked job instance

### do(job\_func, \*args, \*\*kwargs)

Specifies the job\_func that should be called every time the job runs.

Any additional arguments are passed on to job\_func when the job runs.

Parameters: job\_func – The function to be scheduled

Returns: The invoked job instance

### run()

Run the job and immediately reschedule it.

Returns: The return value returned by the job\_func

### to(latest)

Schedule the job to run at an irregular (randomized) interval.

For example, every(A).to(B).seconds executes the job function every N seconds such that  $A \leq N \leq B$ .



# Python Task Scheduler

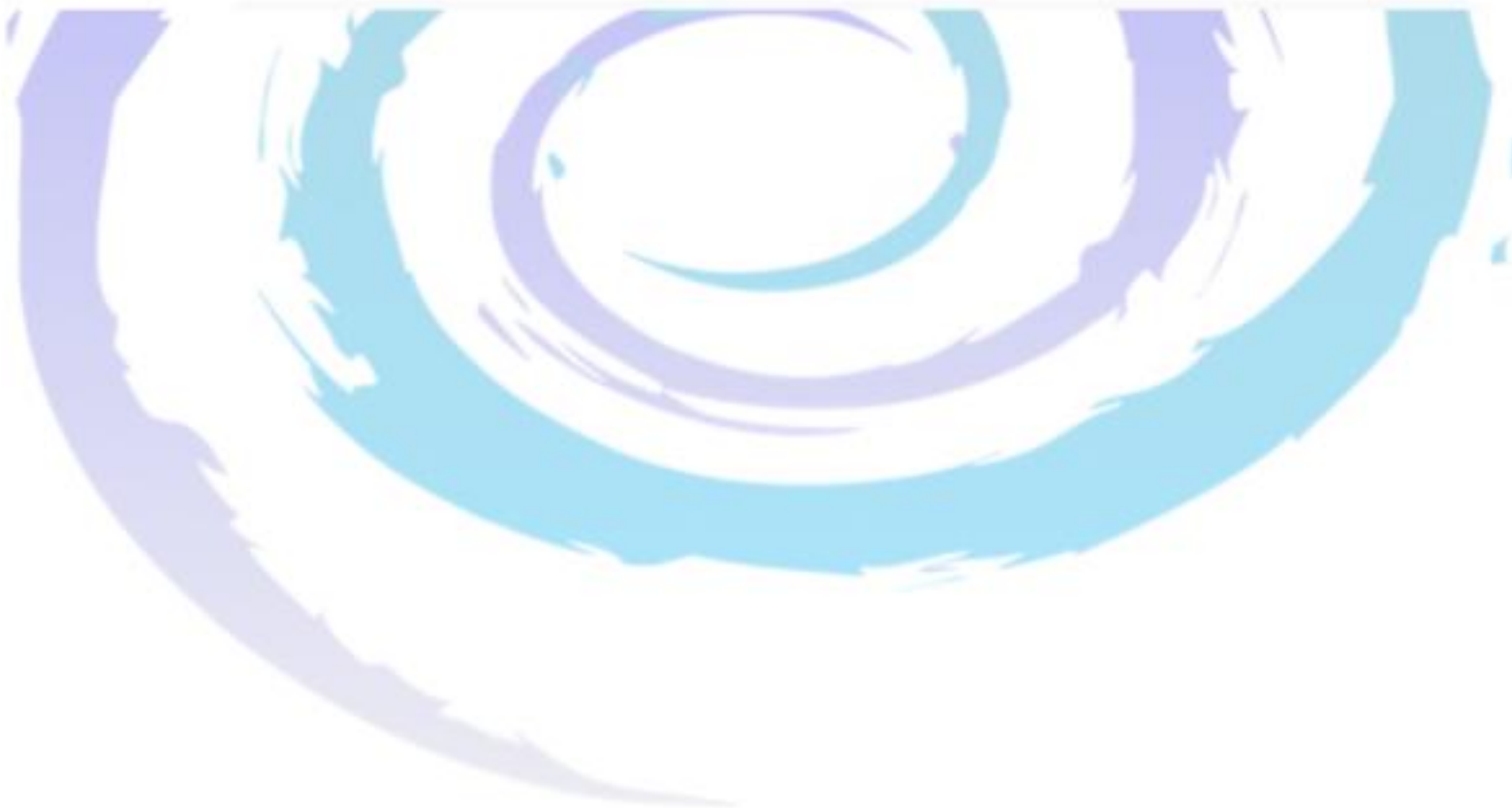
Consider below python application which schedule task after Minute, Hour as well as on specific Day or specific time.

```
1 import schedule
2 import time
3 import datetime
4
5 def fun_Minute():
6     print("Current time is ")
7     print(datetime.datetime.now())
8     print("Scheduler executed after Minute")
9
10 def fun_Hour():
11     print("Current time is ")
12     print(datetime.datetime.now())
13     print("Scheduler executed after Hour")
14
15 def fun_Day():
16     print("Current time is ")
17     print(datetime.datetime.now())
18     print("Scheduler executed after Day")
19
20 def fun_Afternoon():
21     print("Current time is ")
22     print(datetime.datetime.now())
23     print("Scheduler executed at 12")
24
25 def main():
26     print("Marvellous Infosystems : Python Automation & Machine Learning")
27
28     print("Python Job Scheduler")
29     print(datetime.datetime.now())
30
31     schedule.every(1).minutes.do(fun_Minute)
32
33     schedule.every().hour.do(fun_Hour)
34
35     schedule.every().day.at("00:00").do(fun_Afternoon)
36
37     schedule.every().sunday.do(fun_Day)
38
39     schedule.every().saturday.at("18:30").do(fun_Day)
40
41     while True:
42         schedule.run_pending()
43         time.sleep(1)
44
45 if __name__ == "__main__":
46     main()
47
```



## Output of above script

```
MacBook-Pro-de-MARVELLOUS: Desktop marvellous$ python sch.py
Marvellous Infosystems : Python Automation & Machine Learning
Python Job Scheduler
2019-04-20 13:39:17.501788
Current time is
2019-04-20 13:40:17.653276
Scheduler executed after Minute
Current time is
2019-04-20 13:41:17.782483
Scheduler executed after Minute
Current time is
2019-04-20 13:42:17.899424
Scheduler executed after Minute
Current time is
2019-04-20 13:43:18.072133
Scheduler executed after Minute
```





# Python Programming

## Assignment : 15

1. Write a program which accepts file name from user and check whether that file exists in current directory or not.

Input : Demo.txt

Check whether Demo.txt exists or not.

2. Write a program which accept file name from user and open that file and display the contents of that file on screen.

Input : Demo.txt

Display contents of Demo.txt on console.

3. Write a program which accept file name from user and create new file named as Demo.txt and copy all contents from existing file into new file. Accept file name through command line arguments.

Input : ABC.txt

Create new file as Demo.txt and copy contents of ABC.txt in Demo.txt

4. Write a program which accept two file names from user and compare contents of both the files. If both the files contains same contents then display success otherwise display failure. Accept names of both the files from command line.

Input : Demo.txt Hello.txt

Compare contents of Demo.txt and Hello.txt

5. Accept file name and one string from user and return the frequency of that string from file.

Input : Demo.txt

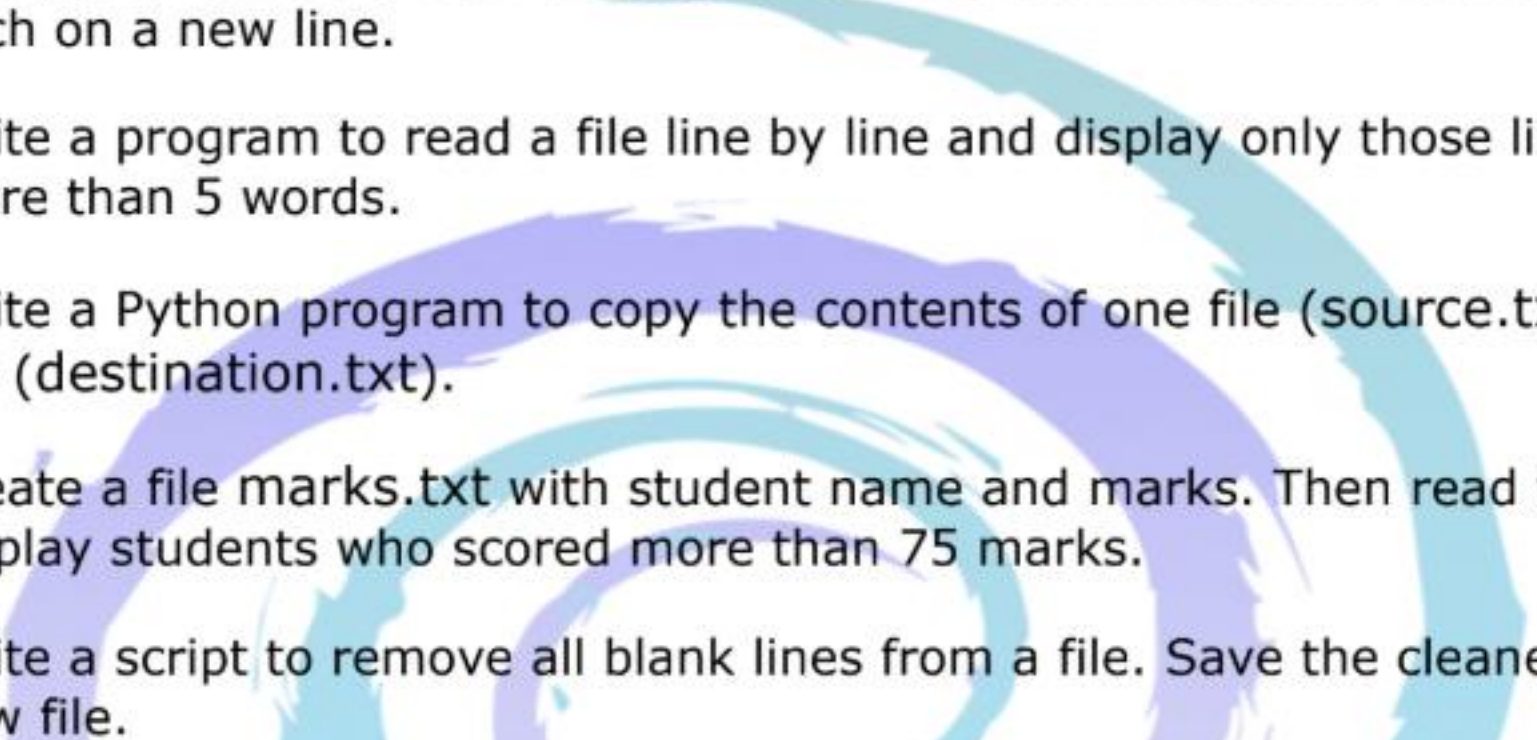
Marvellous

Search "Marvellous" in Demo.txt



# Python Programming

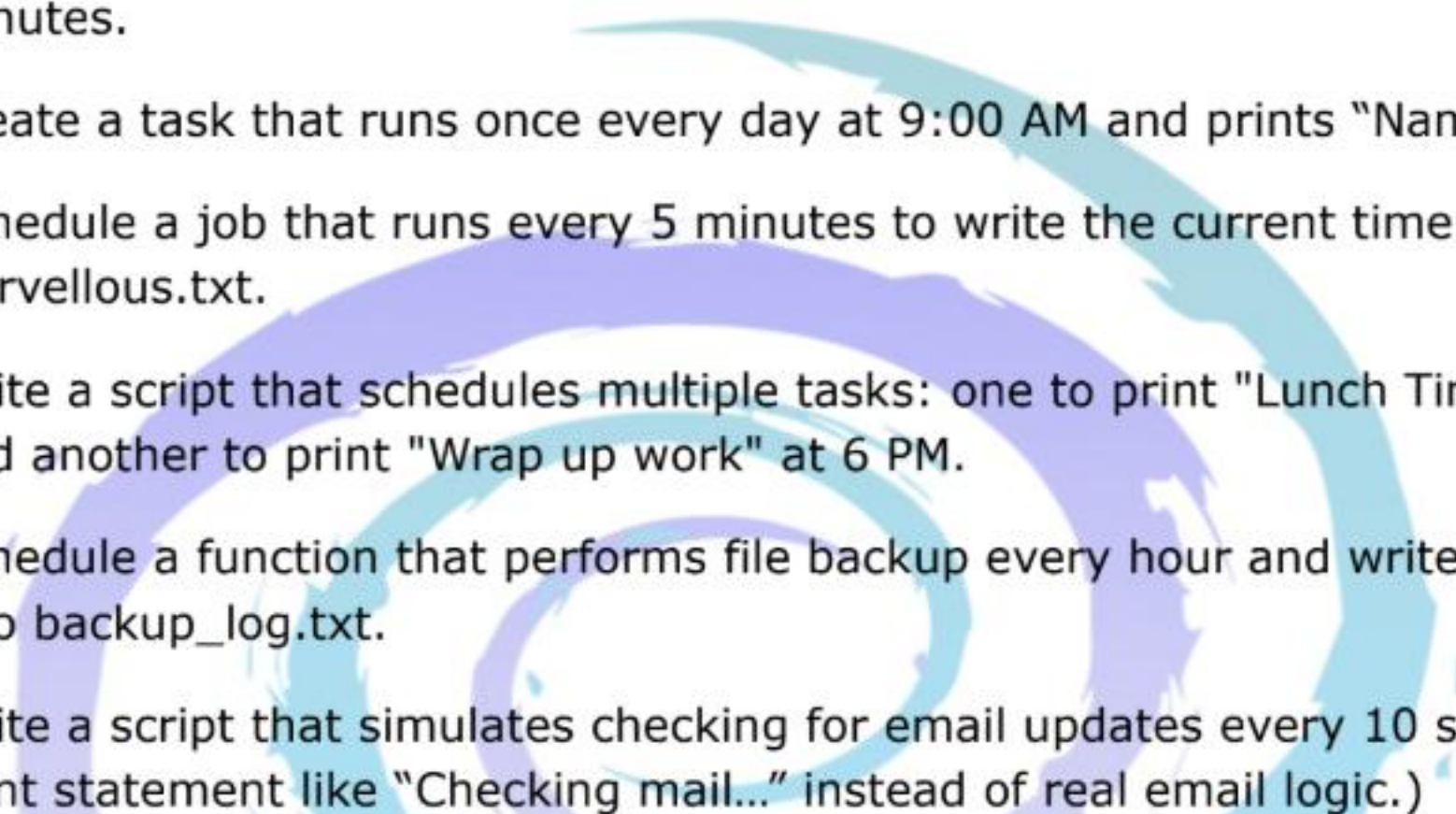
## Assignment : 16

1. Write a Python program to create a text file named student.txt and write the names of 5 students into it.
  2. Write a program to read and display the contents of a file data.txt.
  3. Write a Python script to count the number of lines, words, and characters in a given file.
  4. Accept 10 numbers from the user and write them into a file named numbers.txt, each on a new line.
  5. Write a program to read a file line by line and display only those lines that contain more than 5 words.
  6. Write a Python program to copy the contents of one file (source.txt) into another file (destination.txt).
  7. Create a file marks.txt with student name and marks. Then read the file and display students who scored more than 75 marks.
  8. Write a script to remove all blank lines from a file. Save the cleaned output to a new file.
- 



# Python Programming

## Assignment : 17

1. Write a Python script that prints "Jay Ganesh..." every 2 seconds. Use the `schedule.every(2).seconds.do(...)` function.
  2. Schedule a task that displays the current date and time every minute using the `datetime` module.
  3. Write a program that schedules a function to print "Do Coding..!" every 30 minutes.
  4. Create a task that runs once every day at 9:00 AM and prints "Namskar..."
  5. Schedule a job that runs every 5 minutes to write the current time to a file `Marvellous.txt`.
  6. Write a script that schedules multiple tasks: one to print "Lunch Time!" at 1 PM, and another to print "Wrap up work" at 6 PM.
  7. Schedule a function that performs file backup every hour and writes a log entry into `backup_log.txt`.
  8. Write a script that simulates checking for email updates every 10 seconds. (Use a print statement like "Checking mail..." instead of real email logic.)
- 



# Python Programming

## Assignment : 18

1. Write a program which accepts file name from user and check whether that file exists in current directory or not.

Input : Demo.txt

Check whether Demo.txt exists or not.

2. Write a program which accept file name from user and open that file and display the contents of that file on screen.

Input : Demo.txt

Display contents of Demo.txt on console.

3. Write a program which accept file name from user and create new file named as Demo.txt and copy all contents from existing file into new file. Accept file name through command line arguments.

Input : ABC.txt

Create new file as Demo.txt and copy contents of ABC.txt in Demo.txt

4. Write a program which accept two file names from user and compare contents of both the files. If both the files contains same contents then display success otherwise display failure. Accept names of both the files from command line.

Input : Demo.txt Hello.txt

Compare contents of Demo.txt and Hello.txt

5. Accept file name and one string from user and return the frequency of that string from file.

Input : Demo.txt

Marvellous

Search "Marvellous" in Demo.txt



# Python Programming

## Automation Assignment : 19

Please follow below rules while designing automation script as

- Accept input through command line or through file.
- Display any message in log file instead of console.
- For separate task define separate function.
- For robustness handle every expected exception.
- Perform validations before taking any action.
- Create user defined modules to store the functionality.

1. Design automation script which accept directory name and file extension from user. Display all files with that extension.

Usage : `DirectoryFileSearch.py "Demo" ".txt"`

Demo is name of directory and .txt is the extension that we want to search.

2. Design automation script which accept directory name and two file extensions from user. Rename all files with first file extension with the second file extension.

Usage : `DirectoryRename.py "Demo" ".txt" ".doc"`

Demo is name of directory and .txt is the extension that we want to search and rename with .doc.

After execution this script each .txt file gets renamed as .doc.

3. Design automation script which accept two directory names. Copy all files from first directory into second directory. Second directory should be created at run time.

Usage : `DirectoryCopy.py "Demo" "Temp"`

Demo is name of directory which is existing and contains files in it. We have to create new Directory as Temp and copy all files from Demo to Temp.

4. Design automation script which accept two directory names and one file extension. Copy all files with the specified extension from first directory into second directory. Second directory should be created at run time.

Usage : `DirectoryCopyExt.py "Demo" "Temp" ".exe"`

Demo is name of directory which is existing and contains files in it. We have to create new Directory as Temp and copy all files with extension .exe from Demo to Temp.



# Python Programming

## Automation Assignment : 20

Please follow below rules while designing automation script as

- Accept input through command line or through file.
- Display any message in log file instead of console.
- For separate task define separate function.
- For robustness handle every expected exception.
- Perform validations before taking any action.
- Create user defined modules to store the functionality.

1. Design automation script which accept directory name and display checksum of all files.

Usage : DirectoryChecksum.py "Demo"

Demo is name of directory.

2. Design automation script which accept directory name and write names of duplicate files from that directory into log file named as Log.txt. Log.txt file should be created into current directory.

Usage : DirectoryDusplicate.py "Demo"

Demo is name of directory.

3. Design automation script which accept directory name and delete all duplicate files from that directory. Write names of duplicate files from that directory into log file named as Log.txt. Log.txt file should be created into current directory.

Usage : DirectoryDusplicateRemoval.py "Demo"

Demo is name of directory.

4. Design automation script which accept directory name and delete all duplicate files from that directory. Write names of duplicate files from that directory into log file named as Log.txt. Log.txt file should be created into current directory. Display execution time required for the script.

Usage : DirectoryDusplicateRemoval.py "Demo"

Demo is name of directory.



# Python Programming

## Automation Assignment : 21

Please follow below rules while designing automation script as

- Accept input through command line or through file.
- Display any message in log file instead of console.
- For separate task define separate function.
- For robustness handle every expected exception.
- Perform validations before taking any action.
- Create user defined modules to store the functionality.

1.Design automation script which display information of running processes as its name, PID, Username.

Usage : ProcInfo.py

2.Design automation script which accept process name and display information of that process if it is running.

Usage : ProcInfo.py Notepad

3. Design automation script which accept directory name from user and create log file in that directory which contains information of running processes as its name, PID, Username.

Usage : ProcInfoLog.py Demo

Demo is name of Directory.

4. Design automation script which accept directory name and mail id from user and create log file in that directory which contains information of running processes as its name, PID, Username. After creating log file send that log file to the specified mail.

Usage : ProcInfoLog.py Demo Marvellousinfosystem@gmail.com

Demo is name of Directory.  
marvellousinfosystem@gmail.com is the mail id.



# Python Programming

## Automation Assignment : 22

Please follow below rules while designing automation script as

- Accept input through command line or through file.
- Display any message in log file instead of console.
- For separate task define separate function.
- For robustness handle every expected exception.
- Perform validations before taking any action.
- Create user defined modules to store the functionality.

Design automation script which performs following task.

Accept Directory name from user and delete all duplicate files from the specified directory by considering the checksum of files.

Create one Directory named as Marvellous and inside that directory create log file which maintains all names of duplicate files which are deleted.

Name of that log file should contains the date and time at which that file gets created.

Accept duration in minutes from user and perform task of duplicate file removal after the specific time interval.

Accept Mail id from user and send the attachment of the log file.

Mail body should contains statistics about the operation of duplicate file removal.

Mail body should contains below things :

Starting time of scanning

Total number of files scanned

Total number of duplicate files found

Consider below command line options for the gives script

DuplicateFileRemoval.py E:/Data/Demo 50 marvellousinfosystem@gmail.com

- DuplicateFileRemoval.py

Name of python automation script

- E:/Data/Demo

Absolute path of directory which may contains duplicate files

- 50

Time interval of script in minutes

- marvellousinfosystem@gmail.com

Mail ID of the receiver

Note :

For every separate task write separate function.

Write all user defined functions in one user defined module.

Use proper validation techniques.

Provide Help and usage option for script.

Create one Readme file which contains description of our script, details of command line options.