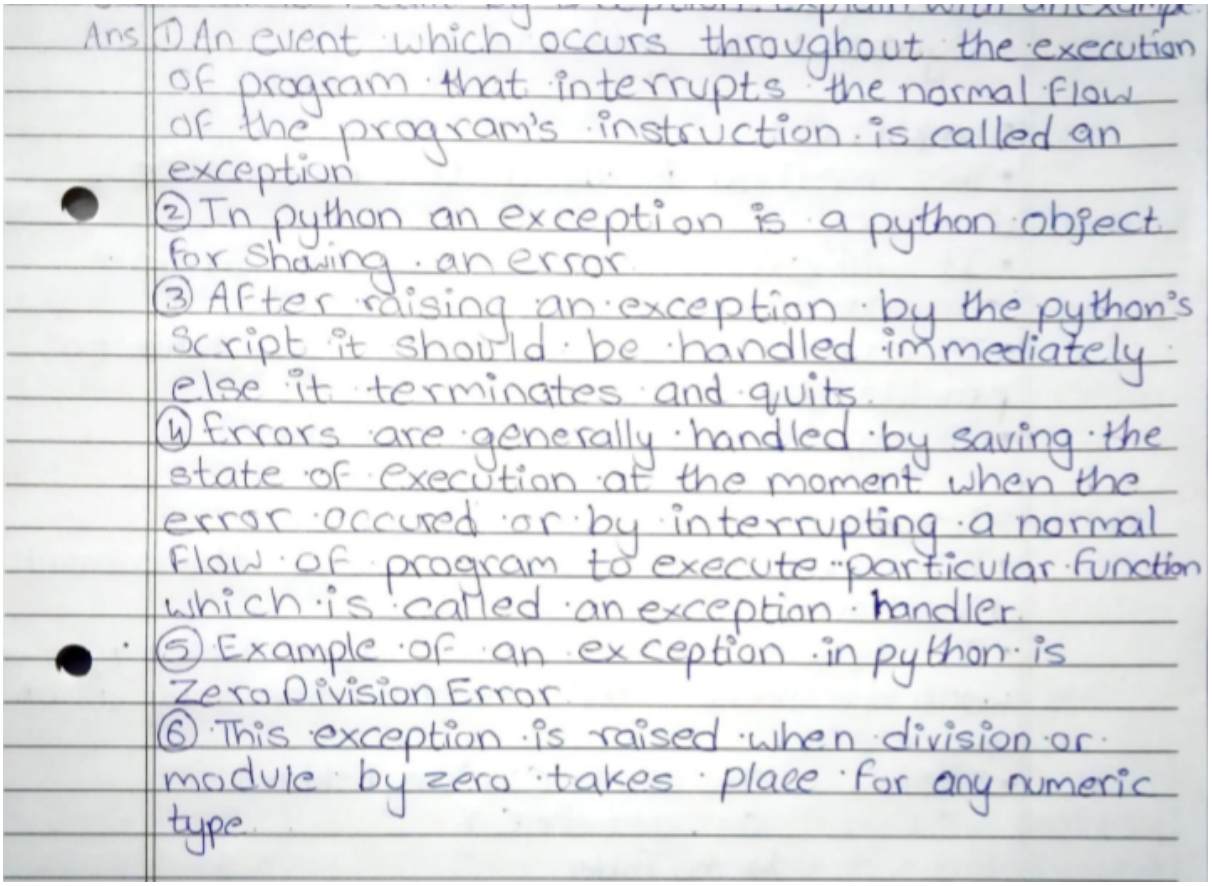
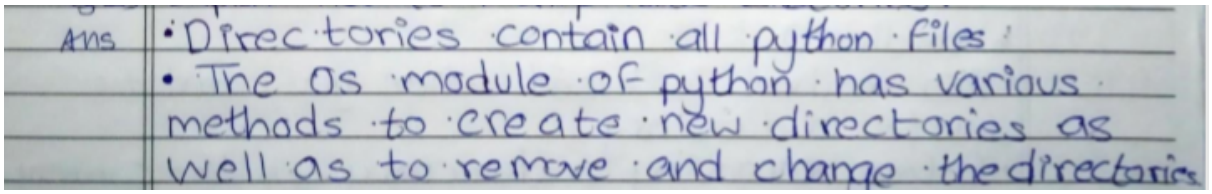
APP

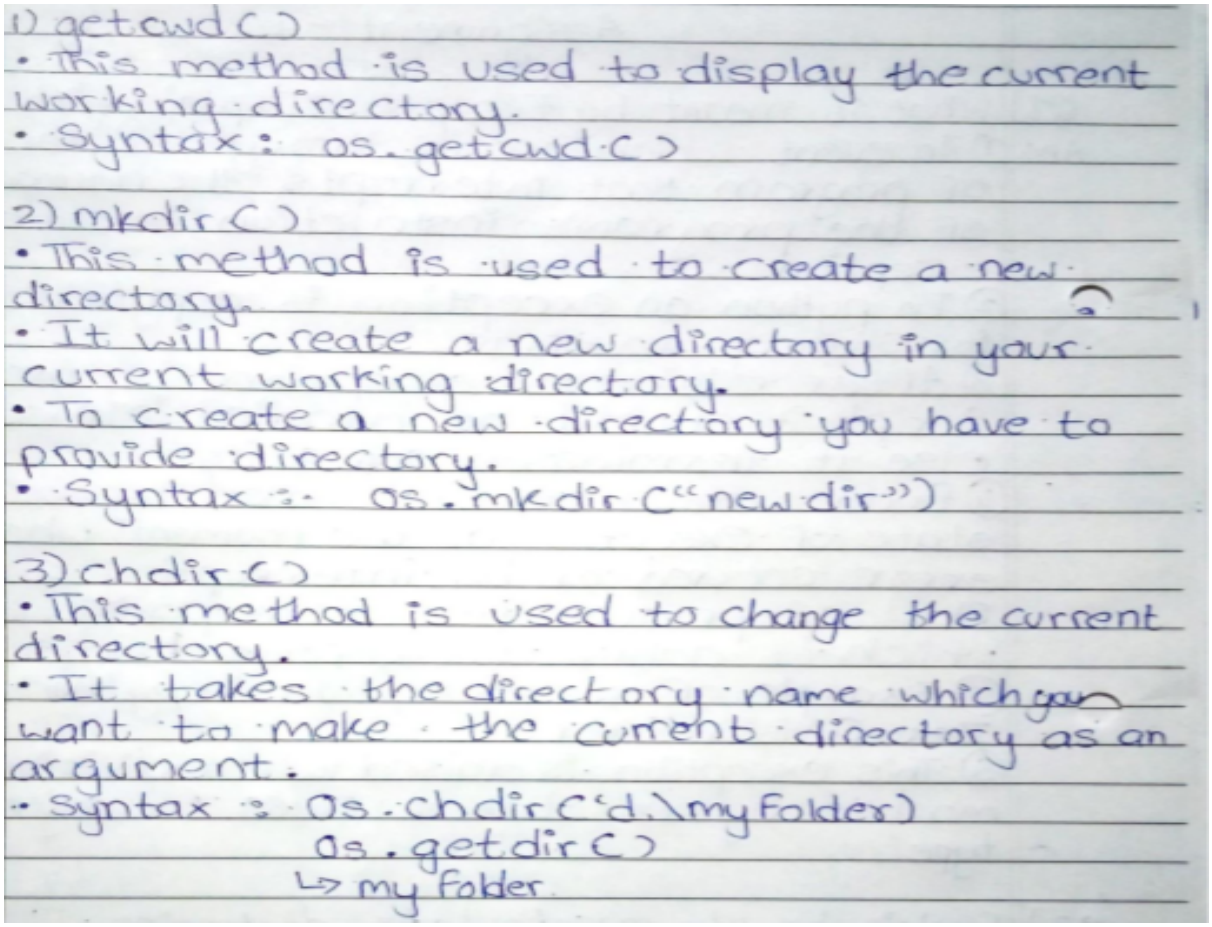
1. What is meant by Exception? Explain with an example.

An event that occurs during the execution of a program that disrupts the normal flow of instructions is called an exception.


Ans ① An event which occurs throughout the execution of program that interrupts the normal flow of the program's instruction is called an exception
② In python an exception is a python object for showing an error
③ After raising an exception by the python's script it should be handled immediately else it terminates and quits.
④ Errors are generally handled by saving the state of execution at the moment when the error occured or by interrupting a normal flow of program to execute particular function which is called an exception handler.
⑤ Example of an exception in python is ZeroDivisionError
⑥ This exception is raised when division or module by zero takes place for any numeric type.

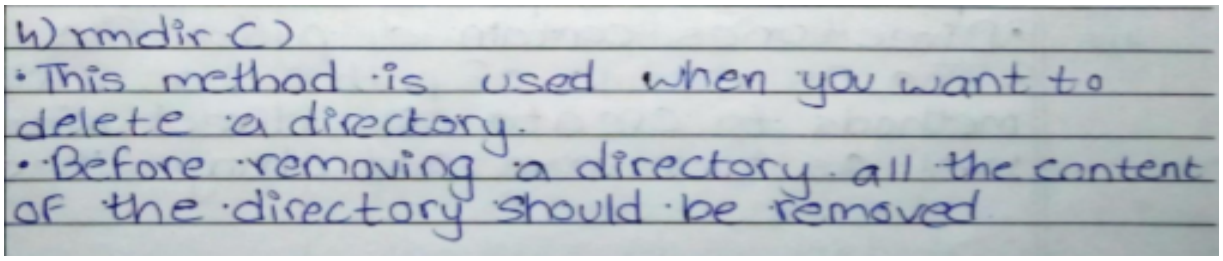
2. Explain how to manipulate directories.

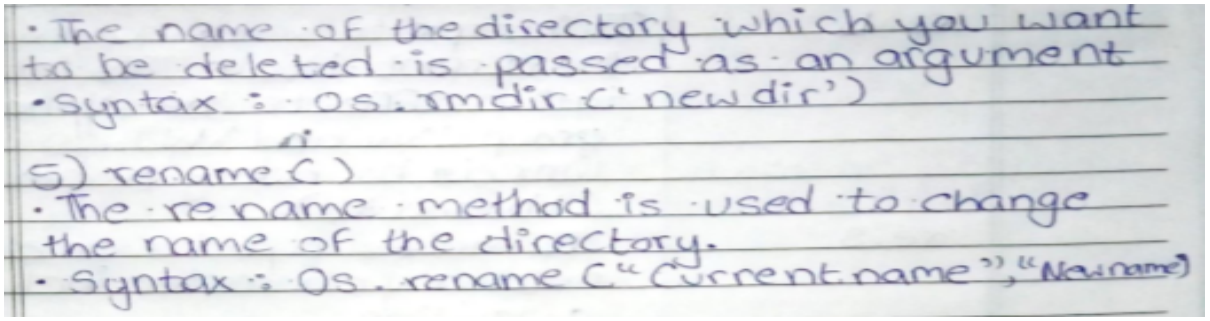

Ans
• Directories contain all python files.
• The os module of python has various methods to create new directories as well as to remove and change the directories


1) getcwd()
• This method is used to display the current working directory.
• Syntax: os.getcwd()

2) mkdir()
• This method is used to create a new directory.
• It will create a new directory in your current working directory.
• To create a new directory you have to provide directory.
• Syntax: os.mkdir("new dir")

3) chdir()
• This method is used to change the current directory.
• It takes the directory name which you want to make the current directory as an argument.
• Syntax: os.chdir('d:\myFolder)
os.getdir()
↳ my folder


4) rmdir()
• This method is used when you want to delete a directory.
• Before removing a directory all the content of the directory should be removed

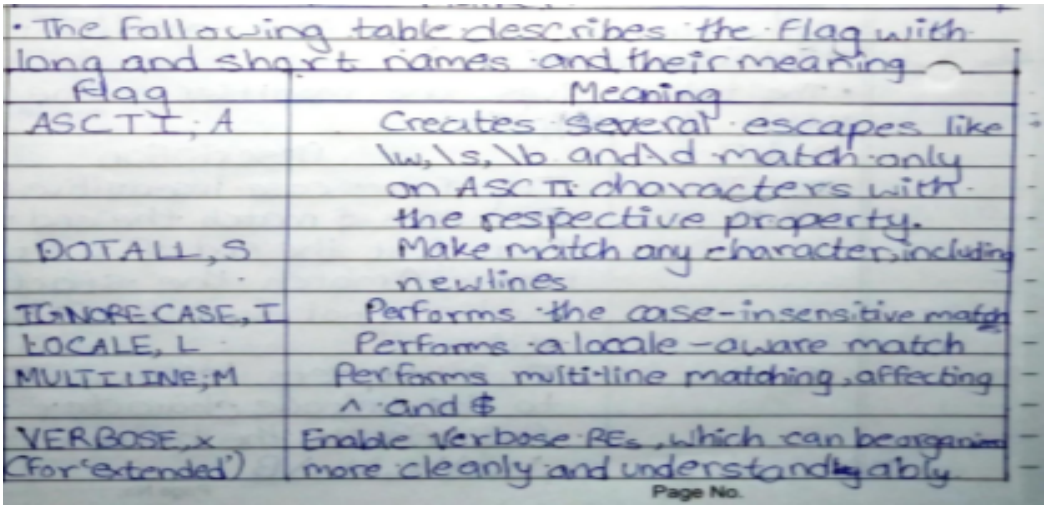

• The name of the directory which you want to be deleted is passed as an argument
• Syntax: os.rmdir('new dir')

5) rename()
• The rename method is used to change the name of the directory.
• Syntax: os.rename("Current name","New name")

3. What are regular expression modifiers or flags?

Ans
- Regular expression literals may include an optional modifier to control various aspects of matching.
- The modifiers are specified as an optional flag.
- You can provide multiple modifiers using exclusive OR (|)
- The table gives the modifier name and their description.

Modifier	Description
re.I	It performs case-insensitive matching
re.M	It makes $ match the end of a line (not just the end of a string) and makes ^ match the start of any line (not just the start of the string)
re.U	It interprets letters according to the unicode character set This flag affects the behaviour of \w, \W, \b, \B
re.L	It interprets words according to the current locale This interpretation affects the alphabetic group (\w and \W), as well as word boundary behaviour (\b and \B)
re.S	It makes a period or dot to match any character; including a newline
re.X	It permits "cuter" regular expression syntax. It ignores white space (except inside a set [] or when escaped by a backslash) and treats unescaped # as a comment marker.



• The following table describes the Flag with long and short names and their meaning.

Flag	Meaning
ASCII, A	Creates several escapes like \w, \s, \b and \d match only on ASCII characters with the respective property.
DOTALL, S	Make match any character, including newlines
IGNORECASE, I	Performs the case-insensitive match
LOCALE, L	Performs a locale-aware match
MULTILINE, M	Performs multi-line matching, affecting ^ and $
VERBOSE, x (for 'extended')	Enable verbose REs, which can be organised more cleanly and understandably

Page No.

4. Explain the iterables and iterators with examples.

Difference Between Iterable and Iterator in Python

In this article, we will discuss the difference between iterable and iterator in Python. But before we do that, let us first know a bit more about them individually.

What is an Iterable?

An Iterable is basically an object that any user can iterate over. We can generate an iterator when we pass the object to the iter() method.

What is an Iterator?

An Iterator is also an object that helps a user in iterating over another object (that is iterable). We use the __next__() method for iterating. This method helps iterators return the next item available from the object.

Remember that every iterator is basically iterable, but vice versa is not always true. For instance, a list can be iterable but not an iterator. We can create an iterator from an iterator using the iter() function. To make it happen, the classes of any object require either an __iter__ method, which basically returns an iterator or a method __getitem__ that has sequential indexes beginning with 0.

When we execute a loop, we use the inter() statement calls on the object that loops over. In case this call becomes successful, an iterator object defining the __next__() will return. It accesses the elements present in an object- one at a time.

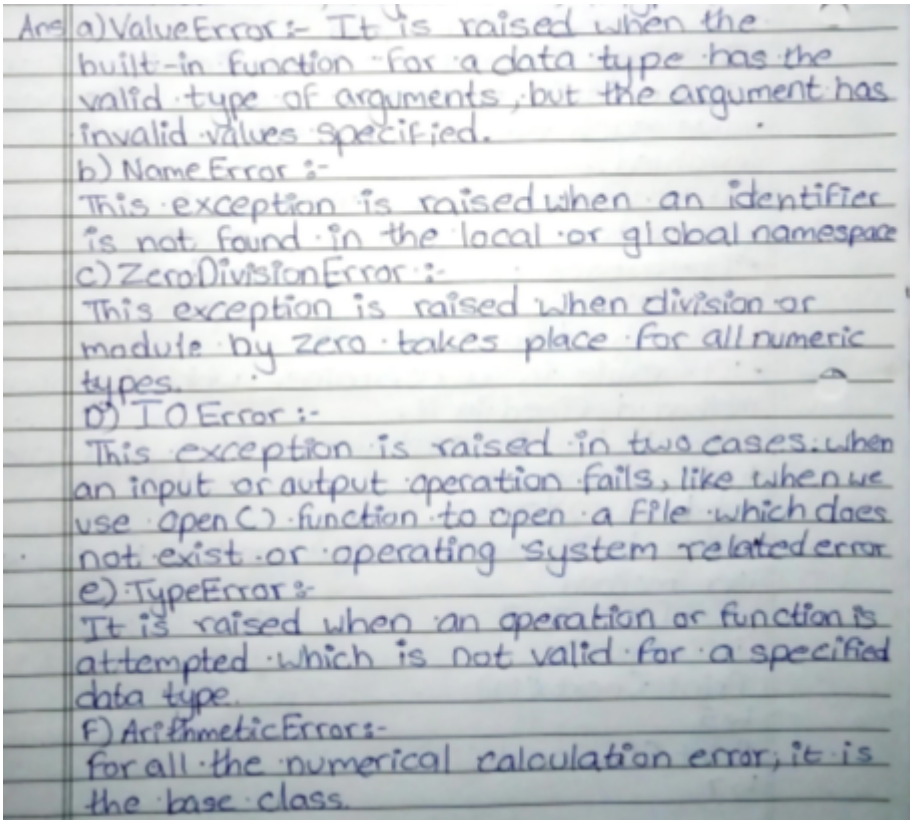
If there are no further elements, then the method __next__() will raise an exception to **StopIteration**. As soon as the fort loop catches this exception, it will immediately terminate.

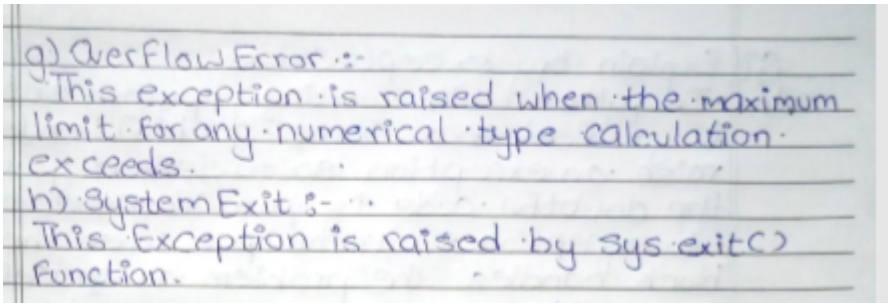
Difference Between Iterable and Iterator in Python

Here is a list of the differences between Iterable and Iterator in Python.

Parameters	Iterable	Iterator
Meaning and Definition	An Iterable is basically an object that any user can iterate over.	An Iterator is also an object that helps a user in iterating over another object (that is iterable).
Method Used	We can generate an iterator when we pass the object to the iter() method.	We use the __next__() method for iterating. This method helps iterators return the next item available from the object.
Inter-Relation	Every iterator is basically iterable.	Not every iterable is an iterator.

5. Explain in brief following exception:
a. ValueError b.NameError c. ZeroDivisionError d.IOError e. TypeError f.ArithmeticError g. OverflowError h.SystemExit I. TypeError

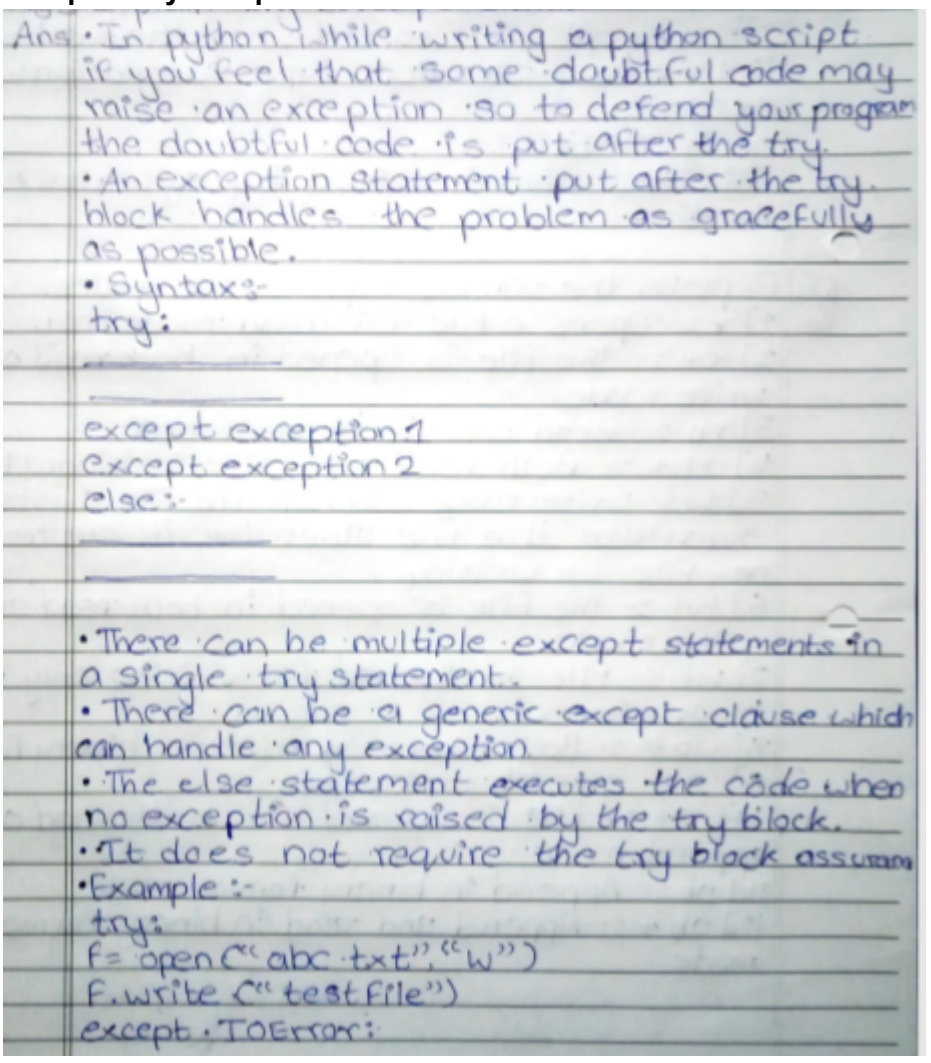

Ans a) ValueError :- It is raised when the built-in function for a data type has the valid type of arguments, but the argument has invalid values specified.
b) NameError :-
This exception is raised when an identifier is not found in the local or global namespace
c) ZeroDivisionError :-
This exception is raised when division or module by zero takes place for all numeric types.
d) IOError :-
This exception is raised in two cases. when an input or output operation fails, like when we use open() function to open a file which does not exist or operating system related error
e) TypeError :-
It is raised when an operation or function is attempted which is not valid for a specified data type.
f) ArithmeticError :-
for all the numerical calculation error, it is the base class.

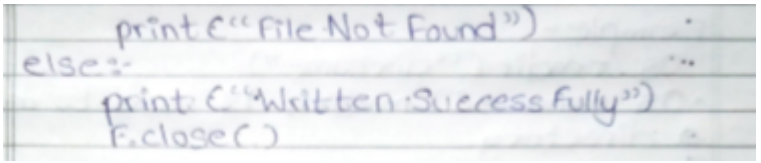

g) OverflowError :-
This exception is raised when the maximum limit for any numerical type calculation exceeds.
h) SystemExit :-
This Exception is raised by sys.exit() function.

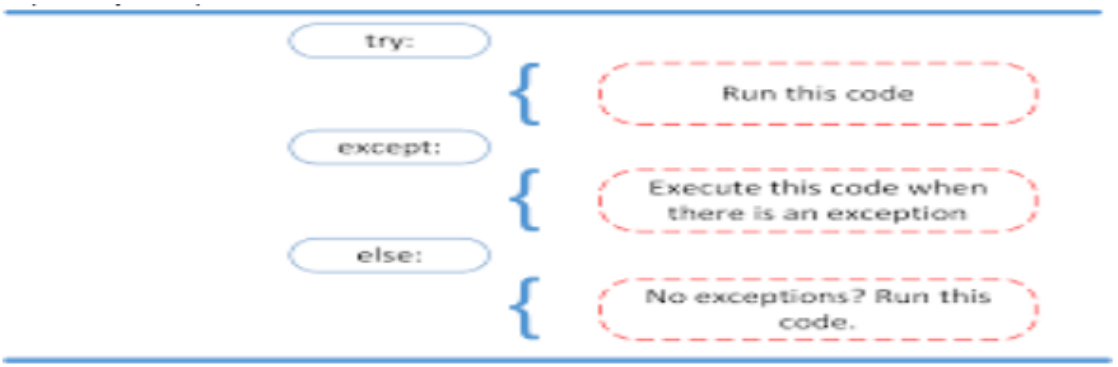
6. Explain the various types of file modes .

Mode	Description
r	Open a file for reading.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
r+	opens for reading and writing (cannot truncate a file)
w+	For writing and reading (can truncate a file)
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.
b	Open in binary mode.
x	Open for exclusive creation, failing if the file already

7. Explain try except clause







8. Explain the OS module of python.

Python - OS Module

It is possible to automatically perform many operating system tasks. The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

You first need to import the `os` module to interact with the underlying operating system. So, import it using the `import os` statement before using its functions.

Getting Current Working Directory

The `getcwd()` function confirms returns the current working directory.

Example: Get Current Working Directory

```
>>> import os
>>> os.getcwd()
'C:\\Python37'
```

Creating a Directory

We can create a new directory using the `os.mkdir()` function, as shown below.

Example: Create a Physical Directory

```
>>> import os
>>> os.mkdir("C:\MyPythonProject")
```

A new directory corresponding to the path in the string argument of the function will be created. If you open the `C:\` drive, then you will see the `MyPythonProject` folder has been created.

By default, if you don't specify the whole path in the `mkdir()` function, it will create the specified directory in the current working directory or drive. The following will create `MyPythonProject` in the `C:\Python37` directory.

Example: Create a Physical Directory

```
>>> import os
>>> os.getcwd()
'C:\Python37'
>>> os.mkdir("MyPythonProject")
```

Changing the Current Working Directory

We must first change the current working directory to a newly created one before doing any operations in it. This is done using the `chdir()` function. The following change current working directory to `C:\MyPythonProject`.

Example: Change Working Directory

```
>>> import os
>>> os.chdir("C:\MyPythonProject") # changing current workign directory
>>> os.getcwd()
'C:\MyPythonProject'
```

You can change the current working directory to a drive. The following makes the `C:\` drive as the current working directory.

Example: Change Directory to Drive

```
>>> os.chdir("C:\\")
>>> os.getcwd()
'C:\\'
```

In order to set the current directory to the parent directory use `".."` as the argument in the `chdir()` function.

Example: Change CWD to Parent

```
>>> os.chdir("C:\\MyPythonProject")
>>> os.getcwd()
'C:\\MyPythonProject'
>>> os.chdir("..")
>>> os.getcwd()
'C:\\'
```

Removing a Directory

The `rmdir()` function in the OS module removes the specified directory either with an absolute or relative path. Note that, for a directory to be removed, it should be empty.

Example: Remove Directory

```
>>> import os
>>> os.rmdir("C:\\MyPythonProject")
```

However, you can not remove the current working directory. To remove it, you must change the current working directory, as shown below.

Example: Remove Directory

```
>>> import os
>>> os.getcwd()
'C:\\MyPythonProject'
>>> os.rmdir("C:\\MyPythonProject")
PermissionError: [WinError 32] The process cannot access the file because it is being used by another process: 'd:\\MyPythonProject
>>> os.chdir("..")
>>> os.rmdir("MyPythonProject")
```

Above, the `MyPythonProject` will not be removed because it is the current directory. We changed the current working directory to the parent directory using `os.chdir("..")` and then remove it using the `rmdir()` function.

List Files and Sub-directories

The `listdir()` function returns the list of all files and directories in the specified directory.

Example: List Directories

```
>>> import os
>>> os.listdir("c:\python37")
['DLLs', 'Doc', 'fantasy-1.py', 'fantasy.db', 'fantasy.py', 'frame.py',
'gridexample.py', 'include', 'Lib', 'libs', 'LICENSE.txt', 'listbox.py', 'NEWS.txt',
'place.py', 'players.db', 'python.exe', 'python3.dll', 'python36.dll', 'pythonw.exe',
'sclst.py', 'Scripts', 'tcl', 'test.py', 'Tools', 'tooltip.py', 'vcruntime140.dll',
'virat.jpg', 'virat.py']
```

If we don't specify any directory, then list of files and directories in the current working directory will be returned.

Example: List Directories of CWD

```
>>> import os
>>>os.listdir()
['.config', '.dotnet', 'python']
```

9. Explain how to create a directory , how to change a directory and how to remove a directory in python.

Making a New Directory in Python

In Python, we can make a new directory using the `mkdir()` method.

This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.

```
os.mkdir('test')

os.listdir()
['test']
```

Changing Directory in Python

In Python, we can change the current working directory by using the `chdir()` method.

The new path that we want to change into must be supplied as a string to this method. And we can use both the forward-slash `/` or the backward-slash `\` to separate the path elements.

Let's see an example,

```
import os

# change directory
os.chdir('C:\\Python33')

print(os.getcwd())

Output: C:\Python33
```

Here, we have used the `chdir()` method to change the current working directory and passed a new path as a string to `chdir()`.

Renaming a Directory or a File

The `rename()` method can rename a directory or a file.

For renaming any directory or file, `rename()` takes in two basic arguments:

- the old name as the first argument
- the new name as the second argument.

Let's see an example,

```
import os

os.listdir()
['test']

# rename a directory
os.rename('test','new_one')

os.listdir()
['new_one']
```

10. Write a python program to write the data into the file.

```
# Python program to demonstrate
# writing to file

# Opening a file
file1 = open('myfile.txt', 'w')
L = ["This is Delhi \n", "This is Paris \n", "This is London \n"]
s = "Hello\n"

# Writing a string to file
file1.write(s)

# Writing multiple strings
# at a time
file1.writelines(L)

# Closing file
file1.close()

# Checking if the data is
# written to file or not
file1 = open('myfile.txt', 'r')
print(file1.read())
file1.close()
```

11. Explain using match function

In Python, match() is a method of the module re

Syntax

Syntax of match()

```
re.match(pattern, string):
```

This method finds match if it occurs at start of the string. For example, calling match() on the string 'TP Tutorials Point TP' and looking for a pattern 'TP' will match. However, if we look for only Tutorials, the pattern will not match. Let's check the code.

Example

```
import re
result = re.match(r'TP', 'TP Tutorials Point TP')
print result
```

Output

```
<_sre.SRE_Match object at 0x0000000005478648>
```

Above, it shows that pattern match has been found. To print the matching string we use method group. Use "r" at the start of the pattern string, it designates a python raw string.

Example

```
import re
result = re.match(r'TP', 'TP Tutorials Point TP')
print result.group(0)
```

Output

```
TP
```

Let's now find 'Tutorials' in the given string. Here we see that string is not starting with 'TP' so it should return no match. Let's see what we get –

Example

```
import re
result = re.match(r'Tutorials', 'TP Tutorials Point TP')
print result
```

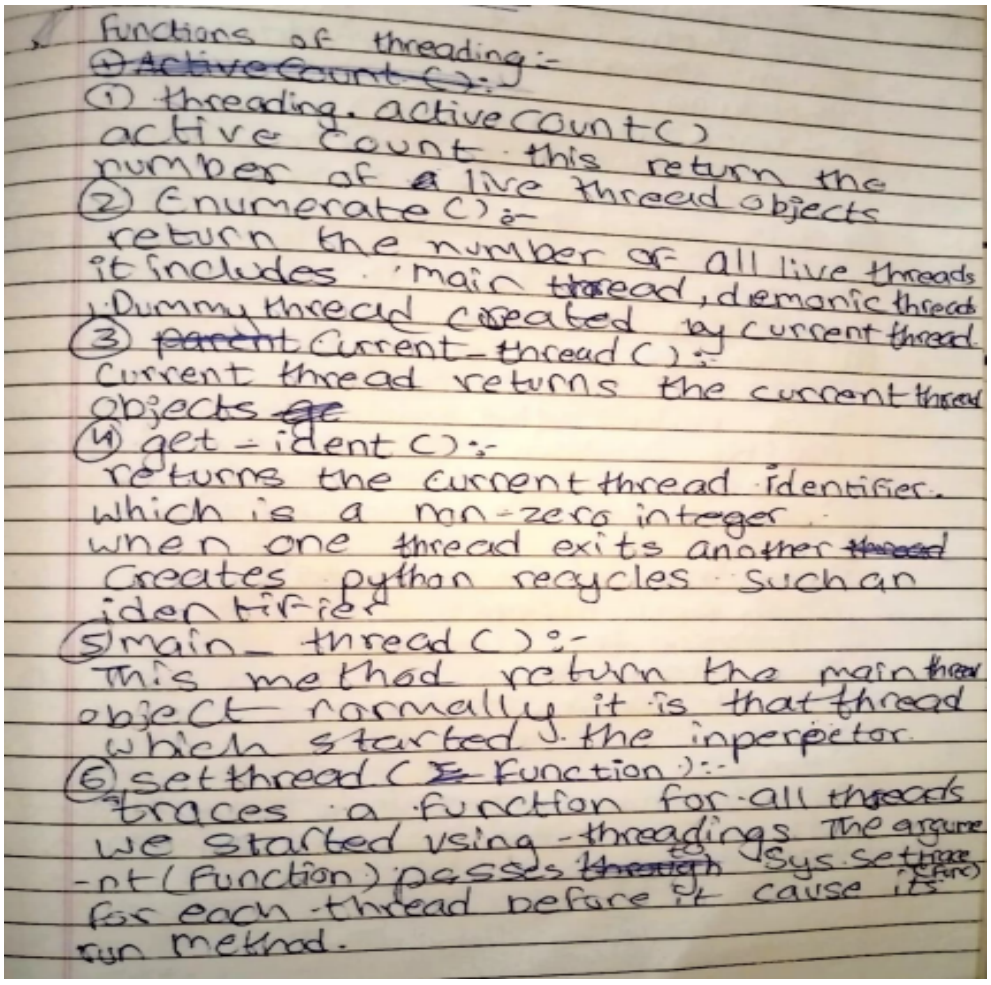
Output

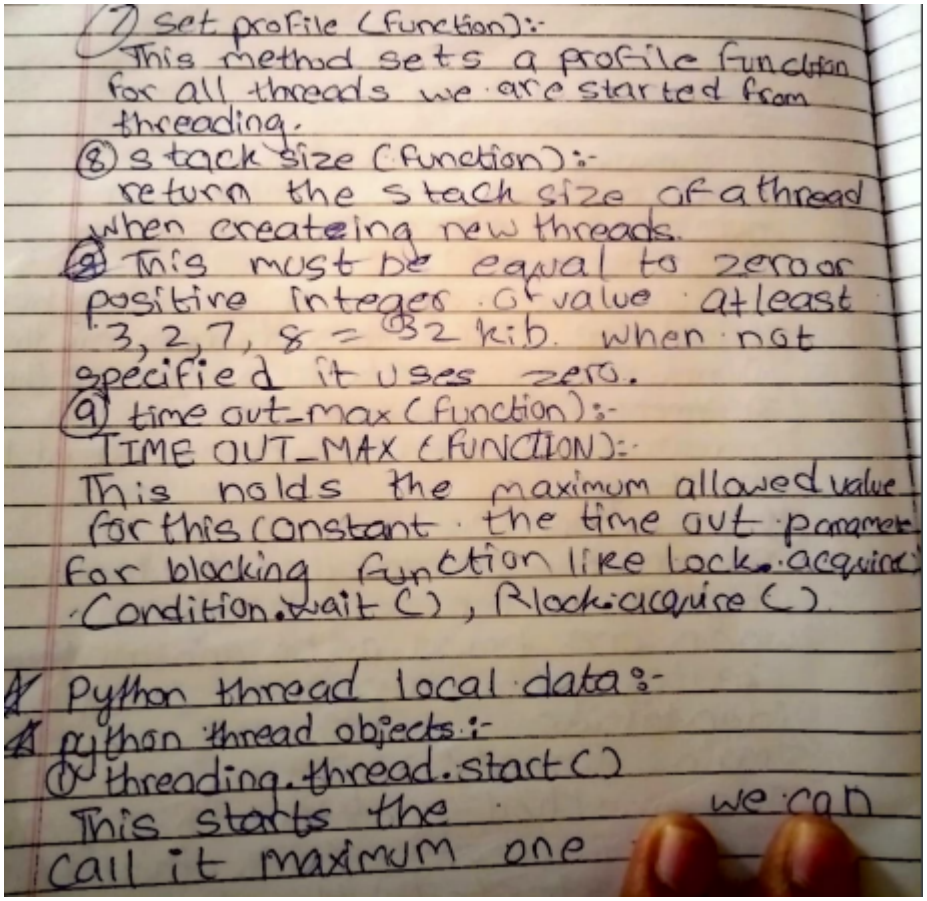
```
None
```

12. Differentiate between thread and process.

Process	Thread
A process is the instance of a program executed by one or many threads.	A thread is a basic unit of CPU utilisation, which consists of its own thread ID, a program counter, a register and a stack.
A process may contain multiple threads depending on the operating system.	A thread is the smallest unit of execution within a process.
In a multiprocessing environment, multiple processes do not share resources such as memory with each other.	Multiple threads of a given process running concurrently can share resources such as memory with each other.
Processes take more time for context switching.	Threads take less time for context switching.
Processes take more time for creation and termination.	Threads take lesser time for creation and termination.
Processes consume more resources.	Threads consume fewer resources.
No other process can execute until the first process gets unblocked.	Another thread in the same task can run while one thread is blocked and waiting.
Process communication is complex.	Thread communication is much easier and efficient.

13. Explain different methods of thread.





14. Explain the thread synchronization process.

Ans:The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads. A new lock is created by calling the Lock() method, which returns the new lock.

The acquire(blocking) method of the new lock object is used to force threads to run synchronously. The optional blocking parameter enables you to control whether the thread waits to acquire the lock.

If blocking is set to 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired. If blocking is set to 1, the thread blocks and wait for the lock to be released.

The release() method of the new lock object is used to release the lock when it is no longer required.

15. Explain the difference between user-defined thread and kernel level thread.

S. No.	Parameters	User Level Thread	Kernel Level Thread
1.	Implemented by	User threads are implemented by users.	Kernel threads are implemented by Operating System (OS).
2.	Recognize	Operating System doesn't recognize user level threads.	Kernel threads are recognized by Operating System.
3.	Implementation	Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
4.	Context switch time	Context switch time is less.	Context switch time is more.
5.	Hardware support	Context switch requires no hardware support.	Hardware support is needed.
6.	Blocking operation	If one user level thread performs blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread can continue execution.
7.	Multithreading	Multithread applications cannot take advantage of multiprocessing.	Kernels can be multithreaded.
8.	Creation and Management	User level threads can be created and managed more quickly.	Kernel level threads take more time to create and manage.
9.	Operating System	Any operating system can support user-level threads.	Kernel level threads are operating system-specific.
10.	Thread Management	The thread library contains the code for thread creation, message passing, thread scheduling, data transfer and thread destroying	The application code does not contain thread management code. It is merely an API to the kernel mode. The Windows operating system makes use of this feature.
11.	Example	Example: Java thread, POSIX threads.	Example: Window Solaris.
12.	Advantages	• User Level Threads are simple and quick to create. • Can run on any operating system • They perform better than kernel threads since they don't need to make system calls to create threads. • In user level threads, switching between threads does not need kernel mode privileges.	• Scheduling of multiple threads that belong to same process on different processors is possible in kernel level threads. • Multithreading can be there for kernel routines. • When a thread at the kernel level is halted, the kernel can schedule another thread for the same process.
13.	Disadvantages	• Multithreaded applications on user-level threads cannot benefit from multiprocessing. • If a single user-level thread performs a blocking operation, the entire process is halted.	• Transferring control within a process from one thread to another necessitates a mode switch to kernel mode. • Kernel level threads takes more time to create and manage than user level threads.

16. Short note on multi-threading

Multithreading refers to **concurrently executing multiple threads by rapidly switching the control of the CPU between threads** (called context switching). The Python Global Interpreter Lock limits one thread to run at a time even if the machine contains multiple processors.
Python supports 2 modules for multithreading: __thread module: It provides a low-level implementation for threading and is obsolete. threading module: It provides a high-level implementation for multithreading and is the current standard

Multithreading is running multiple tasks within a process. It is of two types, namely **user level threads and kernel level threads**. It is economical, responsive, scalable, efficient, and allows resource sharing. There are three models in multithreading: Many to many model, Many to one model, and one to one model.

17. Explain the objects of threading in python.

The threading module exposes all the methods of the thread module and provides some additional methods − threading. activeCount() − Returns the number of thread objects that are active. threading. currentThread() − Returns the number of thread objects
in the caller's thread control

18. Explain timedelta() with an example

19. Explain the time module in python

20. Explain date time module of python

21. What is database connectivity in python, Explain

22. How to execute different types of SQL statements in python, Explain with an example.

23. Explain exception in database connectivity in python.

24. Write a python program to create a table in database using python.

25. Explain the steps to install mySQL connector and explain each and every method.

26. Differentiate between Socket connect() and socket bind()

27. Explain network connectivity.

28. What is socket module? Explain

29. How to send email in python? Explain with an example.

30. Explain URL with example.

31. How to create server-client program? Explain with an example.

32. What is GUI? write advantages and disadvantages.

33. Write a short note on GUI library.

34. What is grid layout? Give Suitable example

35. What are the different types of messagebox available in message widgets?

36. Explain the basic OOPs concept in Python.

37. Explain types of inheritance in python.

38. Short note on Data Abstraction.

39. Differentiate between a Class and Object

40. Explain how to create a class in python.

41. Differentiate between Abstract class and interface.