Unit 6 :File I/O Handling and Exception Handling

I/O OPERATIONS

- **Reading keyboard input:** Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are :

- **input (prompt):** The *input([prompt])* function allows user input. It takes one argument. The syntax is as follows

Syntax:

```
input(prompt)
```

Example

```
x = input("Enter your name: ")
print("hello",x)
```

Output:

```
Enter your name: ABC
hello ABC
```

- **input():** The *input()* function always evaluate the input provided by user and return same type data**.**

Syntax:

x=input()

Example:

```
x=int(input())
print(type(x))
```

Output:

```
14
<class 'int'>
```

➢ Printing to the screen:

- In Python, you can display output to the screen using the print() function.
- The print() function prints the specified message to the screen, or other standard output device.
- The message can be a string, or any other object, the object will be converted into a string before written to the screen.

Syntax:

print(object(s), sep=separator, end=end, file=file, flush=flush)

Parameter values:

- **Object(s)** : It can be any object but will be converted to string before primted.
- **sep='seperator'** : Optional. Specify how to separate the objects, if there are more than one. Default is ' '.
- **end='end':** Optional. Specify what to print at the end. Default is '\n'(line feed).
- **file:** Optional. An object with a write method, Default is sys.stdout
- **flush:** Optional. A Boolean, specifying if the output is flushed(True) or buffered(False). Default is False.

Example

```
print("hello","how are you?",sep="---")
```

Output:

```
hello---how are you?
```

- To make the output more attractive formatting is used. This can be done by using the str.format() method.

Example

```
a=10
b=20
print('Value of a is {} and b is {}'.format(a,b))
```

Output:

```
Value of a is 10 and b is 20
```

File Handling

What is a file?

- 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:
  - Open a file
  - Read or write (perform operation)
  - Close the file

How to open a 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.
- 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.

The open function:

- Before you can read or write a file, you have to open it using Python's built-in **open()** function. This function creates a file object, which would be utilized to call other support methods associated with it.

Syntax

file object = open(file_name [, access_mode][, buffering])

Opening File in different Modes:

In Python, you can open a file in different modes using the **open()** function. Modes determine how the file will be treated when it is opened, whether for reading, writing, appending, or in binary mode.

- **Read Mode ('r'):** This is the default mode when you open a file. It allows you to read the contents of the file. If the file does not exist, it will raise a FileNotFoundError.
- **Write Mode ('w'):** This mode allows you to write to a file. If the file does not exist, it will be created. If the file already exists, its contents will be overwritten.
- **Append Mode ('a'):** This mode allows you to append content to the end of a file. If the file does not exist, it will be created.
- **r+** :Opens a file for both reading and writing. The file pointer placed at the beginning

Example:

```
# For different modes of opening a file
f=open("D:\python programs\Test.txt",'r')
        # opening file in r reading mode
print(f.read())

f=open("Test.txt",'w')
        # Opening file in w writing mode
```

Output:

```
Hello
```

Accessing file contents using standard library functions

- In Python, you can access file contents using standard library function
- Once a file is opened and you have one file object, you can get various information related to that file.

Attribute & Description:

- **file.closed** -Returns true if file is closed, false otherwise.
- **file.mode** -Returns access mode with which file was opened.
- **file.name** -Returns name of the file.
- **file.encoding** -Returns encoding of the file.

Example:

```
fo = open("F:\\2019-2020\\PYTH prog\\README.txt","w)
print("Name of the file: ",fo.name)
print("Closed or not : ",fo.closed)
print("Opening mode : ",fo.mode)
print("File encoding : ",fo.encoding)
```

Output:

```
Name of the file: F:\2019-2020\PYTH prog\README.txt
Closed or not : False
Opening mode : w
File encoding : cp1252
```

Reading data from files

The read() Method: The read() method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax :

fileObject.read([count])

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

The readline() Method-

- The readline() method output the entire line whereas readline(n) outputs at most n bytes of a single line of a file.
- Once the end of line is reached, we get empty string on further reading

Syntax:

file_object.readline(size)

The readlines() Method-

- The readlines() method in Python is used to read all the lines from a file and return them as a list of strings. Each element of the list represents a line from the file

Syntax:

file_object.readlines(sizehint)

Writing a file

There are two ways to write in a file.

write(string): Inserts the string str1 in a single line in the text file.

Syntax:

File_object.write(str1)

Example:

```
fo = open("README.txt","w+")

fo.write("Hello World\n")
```

```
fo = open("README.txt","r")
print(fo.read())
```

Output:

Hello World

The writelines() method:

- The writelines() method in Python is used to write a list of strings to a file.
- It takes a list of strings as an argument and writes each string as a separate line in the file.

Example:

```
L=["str1\n","str2\n","str3\n"]
fo = open("README.txt","w+")
fo.writelines(L)


fo = open("README.txt","r")
print(fo.read())
```

Output:

```
str1
str2
str3
```

Closing a 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 filed and is done using python close() method.

Example:

```
# Program closing a file
fo = open("README.txt")
print("Name of the file: ",fo.name)
fo.close()
```

Output:

```
Name of the file: README.txt
```

Renaming a File

- Renaming a file in python is done with the help of rename() method. To rename a file in python the OS module needs to be imported.
- OS module allows us to perform operating system dependent operations such as making a folder, listing contents of a folder etc.
- The rename() method takes two arguments, the current filename and the new filename.

Syntax :

```
os.rename(current_filename, new_filename)
```

Deleting a File

- We can use the remove() method to delete files by supplying the name of the file to be deleted as the argument.
- To remove a file, the OS module need to be imported.

Syntax :

```
os.remove(filename)
```

Directories in Python

- If there is large number of files to handle in python program, we can arrange the code within different directories to make things more manageable.

- A directory or folder is a collection of files and sub directories. Python has the OS module, which provides us with many useful methods to work with directories.

Directory Related Standard Functions:

➢ Create New Directory:

- 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

➢ Get Current Directory

- We can get the present working directory using the getcwd() method.
- This method returns the current working directory in the form of a string.

➢ Changing Directory:

- We can change the current working directory using the **chdir()** method.
- The new path that we want to change to must be supplied as a string to this method. We can use both forward slash (/) or the backward slash (\) to separate path elements.

➢ List Directories and Files:

- All files and sub directories inside a directory can be known using the **listdir()** method.
- This method takes in a path and returns a list of sub directories and files in that path. If no path is specified, it returns from the current working directory.

➢ Removing Directory:

- A file can be removed (deleted) using the remove() method.
- Similarly, the **rmdir()** method removes an empty directory.

Exception Handling

➢ Introduction

When we execute a python program there may be a few uncertain conditions which occur, known as errors. There are three types of errors:

- Compile Time Error- Occurs at the time of compilation due to violation of syntax rules like missing of a colon.
- Run Time Error- Occurs during the runtime of a program due to wrong input submitted to the program by user.
- Logical Errors- Occurs due to wrong logic written in the program.

- Errors occurs at runtime are known as exception.
- An exception is also called as runtime error that can halt the execution of the program.
- When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, our program will crash.
- For handling exception in python, the exception handler block needs to be written which consists of set of statements that need to be executed according to raised exception.

➢ **‘try : except’** statement-

- In Python, exceptions can be handled using a try statement.
- A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.
- It is up to us, what operations we perform once we have caught the exception.

Syntax:

```
try:
        # Code that may raise an exception
except ExceptionType:
        # Code to handle the exception
```

Example:

```
try:
        x = 5 / 0
# Division by zero, raises ZeroDivisionError

except ZeroDivisionError:
        print("Error: Division by zero occurred!")
```

➢ raise statement:

- In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise.
- We can also optionally pass in value to the exception to clarify why that exception was raised.

Example:

```
while True:
try:
age=int(input("Enter your age for election: "))
if age<18:
raise Exception
else:
```

```
print("You are eligible for election")
break
except Exception:
print("This value is too small, try again")
```

Output:

```
Enter your age for election: 15
This value is too small, try again
Enter your age for election: 20
You are eligible for election
```

User Defined Exception

- Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.
- However, sometimes you may need to create custom exceptions that serve your purpose.
- In Python, users can define such exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from Exception class. Most of the built-in exceptions are also derived from this class.
- User can also create and raise his/her own exception known as user defined exception.

Example:

Q)Program to raise user defined exception if age is less than 18

```
#define python user defined exception
class error(Exception):
   """Base class for other exceptions"""  #empty class


pass
class AgeSmallException(error):
"""Raised when the input value is too small""" #empty class
pass
```

```
# main program
while True:
        try:
                age=int(input("Enter your age for election: "))
if age<18:
        raise AgeSmallException
else:
        print("You are eligible for election")
        break
except AgeSmallException:
        print("This value is too small , try again")
        print()
```

Output:

Enter your age for election: 15

This value is too small , try again

Enter your age for election: 20

You are eligible for election