File Handling

Python Section-2

- Python supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files.
- Python treats file differently as text or binary and this is important.
- Each line of code includes a sequence of characters and they form text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character.
- It ends the current line and tells the interpreter a new one has begun. Let’s start with Reading and Writing files.

Opening a file

- Python provides the open() function which accepts two arguments, file name and access mode in which the file is accessed,
- The function returns a file object which can be used to perform various operations like reading, writing, etc.
- The syntax to use the open() function is given below.

  file object = open(<file-name>, <access-mode>, <buffering>)

- The files can be accessed using various modes like read, write, or append. The following are the details about the access mode to open a file.

1. r : It opens the file to read-only. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.
2. rb : It opens the file to read only in binary format. The file pointer exists at the beginning of the file.
3. r+ : It opens the file to read and write both. The file pointer exists at the beginning of the file.
4. rb+ : It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.
5. w: It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
6. wb : It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
7. w+ : It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. It creates a new file if no file exists. The file pointer exists at the beginning of the file.

8. wb+ : It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file.

- a : It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists with the same name.
- ab : It opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name.
- a+ : It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.
- ab+ : It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

Ex: open () method

```
  opens the file file.txt in read mode
   fileptr = open("file.txt","r")
   if fileptr:
       print("file is opened successfully")
```

```
Output: <class '_io.TextIOWrapper'>
file is opened successfully
```

Ex: close () method

- Once all the operations are done on the file, we must close it through our python script using the close() method. Any unwritten information gets destroyed once the close() method is called on a file object.
- We can perform any operation on the file externally in the file system is the file is opened in python, hence it is good practice to close the file once all the operations are done.
- The syntax to use the close() method is given below.

```
fileobject.close()
```

Examples

1. opens the file file.txt in read mode

```
fileptr = open("file.txt","r")
if fileptr:
    print("file is opened successfully")
```

2. closes the opened file

```
fileptr.close()
```

Reading the file

- To read a file using the python script, the python provides us the read() method. The read() method reads a string from the file. It can read the data in the text as well as binary format.
- The syntax of the read() method is given below.

```
fileobj.read(<count>)
```

Here, the count is the number of bytes to be read from the file starting from the beginning of the file. If the count is not specified, then it may read the content of the file until the end.

Example:

1. open the file.txt in read mode. causes error if no such file exists.
   fileptr = open("file.txt","r");

2. stores all the data of the file into the variable content
   content = fileptr.read(9);

3. prints the type of the data stored in the file
   print(type(content))

4. prints the content of the file
   print(content)

5. closes the opened file
   fileptr.close()

Output:

```
<class 'str'>
Hi, I am
```

Read Lines of the file

- Python facilitates us to read the file line by line by using a function readline(). The readline() method reads the lines of the file from the beginning, i.e., if we use the readline() method two times, then we can get the first two lines of the file.
- Consider the following example which contains a function readline() that reads the first line of our file **"file.txt"** containing three lines.
- Example

1. open the file.txt in read mode. causes error if no such file exists.

   ```
   fileptr = open("file.txt","r");
   ```

2. stores all the data of the file into the variable content

   ```
   content = fileptr.readline();
   ```

3.prints the type of the data stored in the file

print(type(content))

4. prints the content of the file

print(content)

5. closes the opened file

fileptr.close()

Output:

```
<class 'str'>
Hi, I am the file and being used as
```

Looping through the file

- By looping through the lines of the file, we can read the whole file.
- Example

1.open the file.txt in read mode. causes an error if no such file exists.

```
fileptr = open("file.txt","r");
```

2. running a for loop

```
for i in fileptr:
      print(i) # i contains each line of the file
```

Output:

```
Hi, I am the file and being used as an example to read a file in python.
```

Writing the file

- To write some text to a file, we need to open the file using the open method with one of the following access modes.
- **a:** It will append the existing file. The file pointer is at the end of the file. It creates a new file if no file exists.
- **w:** It will overwrite the file if any file exists. The file pointer is at the beginning of the file.

Consider the following example.

Example 1

1. open the file.txt in append mode. Creates a new file if no such file exists.

```
fileptr = open("file.txt","a");
```

2. appending the content to the file

```
fileptr.write("Python is the modern day language. It makes things so si
mple.")
```

3. closing the opened file

```
fileptr.close();
```

Now, we can see that the content of the file is modified.

Ex : File.txt:

Hi, I am the file **and** being used as
an example to read a
file **in** python.
Python **is** the modern day language. It makes things so simple.

Example 2

1. open the file.txt in write mode.
   ```
   fileptr = open("file.txt","w");
   ```
2. overwriting the content of the file
   ```
   fileptr.write("Python is the modern day language. It makes things so
   simple.")
   ```
3. closing the opened file
   ```
   fileptr.close();
   ```

Now, we can check that all the previously written content of the file is overwritten with the new text we have passed.

File.txt:

Python **is** the modern day language. It makes things so simple.

Creating a new file

- The new file can be created by using one of the following access modes with the function open(). **x:** it creates a new file with the specified name. It causes an error a file exists with the same name.
- **a:** It creates a new file with the specified name if no such file exists. It appends the content to the file if the file already exists with the specified name.
- **w:** It creates a new file with the specified name if no such file exists. It overwrites the existing file.

Consider the following example.

1. open the file.txt in read mode. causes error if no such file exists.

```
fileptr = open("file2.txt","x");
print(fileptr)
if fileptr:
    print("File created successfully");
```

Output:

```
File created successfully
```

Using with statement with files

- The with statement was introduced in python 2.5. The with statement is useful in the case of manipulating the files. The with statement is used in the scenario where a pair of statements is to be executed with a block of code in between.
- The syntax to open a file using with statement is given below.
- with open(<file name>, <access mode>) as

  <file-pointer>:

  #statement suite
- The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.
- It is always suggestible to use the with statement in the case of file s because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file. It doesn't let the file to be corrupted.

Consider the following example.

Example

```
with open("file.txt",'r') as f:
    content = f.read();
    print(content)
```

Output:

```
Python is the modern day language. It makes things so simple.
```

File Pointer positions

- Python provides the tell() method which is used to print the byte number at which the file pointer exists. Consider the following example.

Example

1. open the file file2.txt in read mode
   fileptr = open("file2.txt","r")
2. initially the filepointer is at 0
   print("The filepointer is at byte :",fileptr.tell())
3. reading the content of the file
   content = fileptr.read();
4. after the read operation file pointer modifies. tell() returns the location of the fileptr.
   print("After reading, the filepointer is at:",fileptr.tell())

Output:

The filepointer is at byte : 0 After reading, the filepointer is at 26

Modifying file pointer position

- In the real world applications, sometimes we need to change the file pointer location externally since we may need to read or write the content at various locations.
- For this purpose, the python provides us the seek() method which enables us to modify the file pointer position externally.
- The syntax to use the seek() method is given below.

  <file-ptr>.seek(offset[, **from**)
- The seek() method accepts two parameters:
- **offset:** It refers to the new position of the file pointer within the file.
- **from:** It indicates the reference position from where the bytes are to be moved. If it is set to 0, the beginning of the file is used as the reference position. If it is set to 1, the current position of the file pointer is used as the reference position. If it is set to 2, the end of the file pointer is used as the reference position.

Consider the following example.

Example

1. open the file file2.txt in read mode

   fileptr = open("file2.txt","r")

2. initially the filepointer is at 0

   print("The filepointer is at byte :",fileptr.tell())

3. changing the file pointer location to 10.

   fileptr.seek(10);

4. tell() returns the location of the fileptr.

   print("After reading, the filepointer is at:",fileptr.tell())

   Output:

   The filepointer is at byte : 0 After reading, the filepointer is at