Python File Handling

Till now, we were taking the input from the console and writing it back to the console to interact with the user.

Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again.

However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling.

In this section of the tutorial, we will learn all about file handling in python including, creating a file, opening a file, closing a file, writing and appending the file, etc.

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.

SN	Access mode	Description
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.
9	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.
10	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.
11	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.
12	ab+	It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

Let's look at the simple example to open a file named "file.txt" (stored in the same directory) in read mode and printing its content on the console.

Example

```
1.  #opens the file file.txt in read mode
2.  fileptr = open("file.txt","r")
3.
4.  if fileptr:
5.      print("file is opened successfully")
```

Output:

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

The 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()

Consider the following example.

Example

```
1.  # opens the file file.txt in read mode
2.  fileptr = open("file.txt","r")
3.
```

```
4.      if fileptr:
5.          print("file is opened successfully")
6.
7.      #closes the opened file
8.      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.

Consider the following example.

Example

```
1.      #open the file.txt in read mode. causes error if no such file exists.
2.      fileptr = open("file.txt","r");
3.
4.      #stores all the data of the file into the variable content
5.      content = fileptr.read(9);
6.
7.      # prints the type of the data stored in the file
8.      print(type(content))
9.
10.     #prints the content of the file
11.     print(content)
12.
13.     #closes the opened file
14.     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

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

#stores all the data of the file into the variable content
content = fileptr.readline();

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

#prints the content of the file
print(content)

#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

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

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

#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

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

```
2.  fileptr = open("file.txt","a");
3.
4.  #appending the content to the file
5.  fileptr.write("Python is the modern day language. It makes things so simple.")
6.
7.
8.  #closing the opened file
9.  fileptr.close();
```

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

File.txt:

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

Example 2

```
1.  #open the file.txt in write mode.
2.  fileptr = open("file.txt","w");
3.
4.  #overwriting the content of the file
5.  fileptr.write("Python is the modern day language. It makes things so simple.")
6.
7.
8.  #closing the opened file
9.  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.

Example

```
1.  #open the file.txt in read mode. causes error if no such file exists.
2.  fileptr = open("file2.txt","x");
3.
4.  print(fileptr)
```

```
5.
6.      if fileptr:
7.          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 files 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

```
1.      with open("file.txt",'r') as f:
2.          content = f.read();
3.          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
2. fileptr = open("file2.txt","r")

3. #initially the filepointer is at 0
4. print("The filepointer is at byte :",fileptr.tell())

5. #reading the content of the file
6. content = fileptr.read();
```

```
7. #after the read operation file pointer modifies. tell() returns the location of the file ptr.
8. print("After reading, the filepointer is at:",fileptr.tell())
9. 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
2.  fileptr = open("file2.txt","r")
3.
4.  #initially the filepointer is at 0
5.  print("The filepointer is at byte :",fileptr.tell())
6.
7.  #changing the file pointer location to 10.
8.  fileptr.seek(10);
9.
10. #tell() returns the location of the fileptr.
11. print("After reading, the filepointer is at:",fileptr.tell())
```

Output:
The filepointer is at byte : 0
After reading, the filepointer is at 10

Python os module

The os module provides us the functions that are involved in file processing operations like renaming, deleting, etc.

Let's look at some of the os module functions.

Renaming the file

The os module provides us the rename() method which is used to rename the specified file to a new name. The syntax to use the rename() method is given below.

rename(?current-name?, ?new-name?)

Example

```
1.  import os;
2.
3.  #rename file2.txt to file3.txt
4.  os.rename("file2.txt","file3.txt")
```

Removing the file

The os module provides us the remove() method which is used to remove the specified file. The syntax to use the remove() method is given below.

remove(?file-name?)

Example

```
1.  import os;
2.
3.  #deleting the file named file3.txt
4.  os.remove("file3.txt")
```

Creating the new directory

The mkdir() method is used to create the directories in the current working directory. The syntax to create the new directory is given below.

mkdir(?directory name?)

Example

```
1.  import os;
2.
3.  #creating a new directory with the name new
4.  os.mkdir("new")
```

Changing the current working directory

The chdir() method is used to change the current working directory to a specified directory.
The syntax to use the chdir() method is given below.

chdir("new-directory")

Example

```
1.  import os;
2.
3.  #changing the current working directory to new
4.
5.  os.chdir("new")
```

The getcwd() method

This method returns the current working directory.
The syntax to use the getcwd() method is given below.

os.getcwd()

Example

```
1.  import os;
2.
3.  #printing the current working directory
4.  print(os.getcwd())
```

Deleting directory

The rmdir() method is used to delete the specified directory.
The syntax to use the rmdir() method is given below.

os.rmdir(?directory name?)

Example

```
1.  import os;
2.
3.  #removing the new directory
4.  os.rmdir("new")
```

Writing python output to the files

In python, there are the requirements to write the output of a python script to a file.
The **check_call()** method of module **subprocess** is used to execute a python script and write the output of that script to a file.

The following example contains two python scripts. The script file1.py executes the script file.py and writes its output to the text file **output.txt**

file.py:

```
1.  temperatures=[10,-20,-289,100]
2.  def c_to_f(c):
3.      if c< -273.15:
4.          return "That temperature doesn't make sense!"
5.      else:
6.          f=c*9/5+32
7.          return f
```

```
8.      for t in temperatures:
9.          print(c_to_f(t))
```

file1.py:

```
1.      import subprocess
2.       with open("output.txt", "wb") as f:
         subprocess.check_call(["python", "file.py"], stdout=f)
```

Output:

```
50
-4
That temperature doesn't make sense!
212
```