Text File (Module-9)

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. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, like other concepts of Python, this concept here is also easy and short.

Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters, and they form a 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.

File Opening

The key function for working with files in Python is the open() function. The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

r :Read - Default value. Opens a file for reading, error if the file does not exist
a :Append - Opens a file for appending, creates the file if it does not exist
w :Write - Opens a file for writing, creates the file if it does not exist
r+ :To read and write data into the file. This mode does not override the existing data, but you can modify the data starting from the beginning of the file.
w+: To write and read data. It overwrites the previous file if one exists, it will truncate the file to zero length or create a file if it does not exist.
a+: To append and read data from the file. It won't override existing data.
x :Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode
"b" - Binary - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

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

The code above is the same as:

```
f = open("myfile.txt", "rt")
```

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

File Closing

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.

Syntax

The syntax for closing a file in Python is given below -

```
fileobject.close()
```

```
f = open("myfile.txt")
f.close()
```

The with statement

The '**with'** statement was introduced in python 2.5. The 'with' statement is useful in the case of manipulating the files. It is used in the scenario where a pair of statements is to be executed with a block of code in between.

Syntax:

```
with open(<file name>, <access mode>) as <file-object>:
   #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.

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

Reading the content

The open() function returns a file object, which has a read() method for reading the content of the file:

```
f = open("myfile.txt", "r")
print(f.read())
```

Reading Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

Example

Return the 5 first characters of the file:

```
f = open("myfile.txt", "r")
print(f.read(5))
```

Reading Lines

We can return one line by using the readline() method:
By calling readline() two times, you can read the two first lines:

Example

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

Writing to a File

```
f = open("myfile.txt", "w")
f.write("Hello World!")
f.close()

#open and read the file after the writing:
f = open("myfile.txt", "r")
print(f.read())
```

Output:
Hello World!

Delete a File

To delete a file, you must import the OS module, and run its os.remove() function:

```
import os
os.remove("myfile.txt")
```

Check if File exist:
To avoid getting an error, we might want to check if the file exists before you try to delete it:

Example
Check if file exists, *then* delete it:

```
import os
if os.path.exists("myfile.txt"):
  os.remove("myfile.txt")
else:
  print("The file does not exist")
```

seek() method

In Python, seek() function is used to **change the position of the File Handle** to a given specific position. File handle is like a cursor, which defines from where the data has to be read or written in the file.

Syntax: f.seek(offset, from_what), where f is file pointer

Parameters:
Offset: Number of positions to move forward
from_what: It defines point of reference.
Returns: Return the new absolute position.

The reference point is selected by the **from_what** argument. It accepts three values:
0: sets the reference point at the beginning of the file

1: sets the reference point at the current file position

2: sets the reference point at the end of the file

By default from_what argument is set to 0.

Note: Reference point at current position / end of file cannot be set in text mode except when offset is equal to 0.

tell() Method

The tell() method returns the current file position in a file stream.
file.tell()