BCA – 502: Python Programming

Rahul Kumar Singh

In today's Class we have discussed on File Operation in Python.

File Handling:-

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

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 −

- raw_input
- input

The raw_input Function

The raw_input([prompt]) function reads one line from standard input and returns it as a string (removing the trailing newline).

Example:-

```
#!/usr/bin/python

str = raw_input("Enter your input: ")

print "Received input is : ", str
```

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello Python!", its output is like this –

```
Enter your input: Hello Python

Received input is :  Hello Python
```

The input Function:-

The input([prompt]) function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

Example:-

```
#!/usr/bin/python

str = input("Enter your input: ")

print "Received input is : ", str
```

Output:-

This would produce the following result against the entered input –

```
Enter your input: [x*5 for x in range(2,10,2)]
Recieved input is :  [10, 20, 30, 40]
```

Opening and Closing Files:-

Until now, we have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a file object.

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])
```

Here are parameter details –

file_name – The file_name argument is a string value that contains the name of the file that you want to access.

access_mode – The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).

Here is a list of the different modes of opening a file –

r - Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.

rb - Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.

rb+ - Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.

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.

wb - Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

w+ - Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

wb+ - Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

a - Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

ab - Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

a+ - Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

ab+ - Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

buffering – If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

The file Object Attributes:-

Once a file is opened and you have one file object, you can get various information related to that file.

Here is a list of all attributes related to file object –

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.softspace - Returns false if space explicitly required with print, true otherwise.

Example

```
#!/usr/bin/python

# Open a file

fo = open("foo.txt", "wb")

print "Name of the file: ", fo.name

print "Closed or not : ", fo.closed

print "Opening mode : ", fo.mode

print "Softspace flag : ", fo.softspace
```

This produces the following result –

```
Name of the file:  foo.txt

Closed or not :  False

Opening mode :  wb

Softspace flag :  0
```

The close() Method:-

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax:-

```
fileObject.close()
```

Example:-

```
#!/usr/bin/python

# Open a file

fo = open("foo.txt", "wb")

print "Name of the file: ", fo.name
```

```
# Close opend file
fo.close()
```

Output:-

This produces the following result –

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

Reading and Writing Files:-

The file object provides a set of access methods to make our lives easier. We would see how to use read() and write() methods to read and write files.

The write() Method

The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

The write() method does not add a newline character ('\n') to the end of the string –

Syntax

```
fileObject.write(string)
```

Here, passed parameter is the content to be written into the opened file.

Example

```
#!/usr/bin/python
# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n")
# Close opend file
fo.close()
```

The above method would create foo.txt file and would write given content in that file and finally it would close that file. If you would open this file, it would have following content.

Output:-

```
Python is a great language.
Yeah its great!!
```

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.

Example:-

Let's take a file foo.txt, which we created above.

```
#!/usr/bin/python

# Open a file

fo = open("foo.txt", "r+")

str = fo.read(10);

print "Read String is : ", str

# Close opend file

fo.close()
```

Output:-

This produces the following result –

```
Read String is :  Python is
```