

Programming Fundamental

Lab#11

Contents

File Handling.....	2
Opening a File	2
The file Object Attributes.....	4
The close() Method	5
Reading and Writing Files	6
Write() Method	6
Read() Method	6
Looping over a file object.....	7
File Positions	8
Renaming and Deleting Files.....	9
Directories in Python	10

File Handling

Until now, you 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.

Opening a File

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])
or
with open(file_name [, access_mode]) as file_object:
    print ("file open ", file_object.name)
```

- **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 –

Sr.No	Modes & Description
1	r Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	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.

3	r+ Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4	rb+ Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5	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.
6	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.
7	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.
8	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.
9	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.

10	<p>ab</p> <p>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.</p>
11	<p>a+</p> <p>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.</p>
12	<p>ab+</p> <p>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.</p>

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 –

Sr.No.	Attribute & Description
1	<p>file.closed</p> <p>Returns true if file is closed, false otherwise.</p>
2	<p>file.mode</p> <p>Returns access mode with which file was opened.</p>
3	<p>file.name</p>

Returns name of the file.

Example

```
#!/usr/bin/python

# Open a file
file = open("foo.txt", "wb")
print "Name of the file: ", file.name
print "Closed or not : ", file.closed
print "Opening mode : ", file.mode
```

This produces the following result –

```
Name of the file:  foo.txt
Closed or not :  False
Opening mode :  wb
```

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
file = open("foo.txt", "wb")
print "Name of the file: ", file.name
# Close opened file
file.close()
```

Output

```
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.

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
f = open("foo.txt", "wb")
f.write("Python is a great language.\nYeah its great!!\n");

# Close opened file
f.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.

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

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
f = open("foo.txt", "r+")
str = f.read(10);
print "Read String is : ", str
# Close opened file
f.close()
```

This produces the following result –

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

Looping over a file object

When you want to read – or return – all the lines from a file in a more memory efficient, and fast manner, you can use the loop over method. The advantage to using this method is that the related code is both simple and easy to read.

```
In [15]: file = open("untitled.txt", "r")
         for line in file:
             print line
```

1st line

2nd line

3rd line

4th line

5th line

File Positions

The `tell()` method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The `seek(offset[, from])` method changes the current file position.

The offset argument indicates the number of bytes to be moved.

The from argument specifies the reference position from where the bytes are to be moved.

If *from* is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Example

Let us 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

# Check current position
position = fo.tell();
print "Current file position : ", position

# Reposition pointer at the beginning once again
position = fo.seek(0, 0);
str = fo.read(10);
print "Again read String is : ", str
# Close opened file
fo.close()
```

This produces the following result –

```
Read String is : Python is
Current file position : 10
Again read String is : Python is
```


Renaming and Deleting Files

Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

Rename()

The `rename()` method takes two arguments, the current filename and the new filename.

Syntax

```
os.rename(current_file_name, new_file_name)
```

Example

Following is the example to rename an existing file `test1.txt`

```
#!/usr/bin/python
import os

# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

Remove()

You can use the `remove()` method to delete files by supplying the name of the file to be deleted as the argument.

Syntax

```
os.remove(file_name)
```

Example

Following is the example to delete an existing file `test2.txt` –

```
#!/usr/bin/python
import os
```

```
# Delete file test2.txt
os.remove("text2.txt")
```

Directories in Python

All files are contained within various directories, and Python has no problem handling these too. The **os** module has several methods that help you create, remove, and change directories.

The **mkdir()** Method

You can use the **mkdir()** method of the **os** module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

Syntax

```
os.mkdir("newdir")
```

Example

Following is the example to create a directory test in the current directory

```
#!/usr/bin/python
import os

# Create a directory "test"
os.mkdir("test")
```

The **chdir()** Method

You can use the **chdir()** method to change the current directory. The **chdir()** method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax

```
os.chdir("newdir")
```

Example

Following is the example to go into `"/home/newdir"` directory

```
#!/usr/bin/python
import os

# Changing a directory to "/home/newdir"
os.chdir("/home/newdir")
```

os.path.join() Method

The **os.path.join(dir, name)** method takes the directory/folder name and file name. this function joins the directory path with file name

Following is the example

```
#!/usr/bin/python
import os

folder="files"
name="names.txt"
filename=os.path.join(folder,name)
```

The getcwd() Method

The **getcwd()** method displays the current working directory.

Syntax

```
os.getcwd()
```

Example

Following is the example to give current directory

```
#!/usr/bin/python
import os

# This would give location of the current directory
os.getcwd()
```

The rmdir() Method

The **rmdir()** method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Syntax

```
os.rmdir('dirname')
```

Example

Following is the example to remove "/tmp/test" directory. It is required to give fully qualified name of the directory, otherwise it would search for that directory in the current directory.

```
#!/usr/bin/python
import os

# This would remove "/tmp/test" directory.
os.rmdir( "/tmp/test" )
```

File & Directory Related Methods

There are three important sources, which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows

- **File Object Method:** The *file* object provides functions to manipulate files.
https://www.tutorialspoint.com/python/file_methods.htm
- **OS Object Method:** This provides methods to process files as well as directories.
https://www.tutorialspoint.com/python/os_file_methods.htm