Files in Python

Gerardo Carmona

Opening a File

- open() returns a file object, and is most commonly used with two arguments: open(filename, mode).

```
>>> f = open('workfile', 'w')
>>> print f
```

Modes

Modes	Description
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.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
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.

Create a File

- You can name it anything you like, in this example we will name it "newfile.txt".

```
>>> file = open("newfile.txt", "w")
>>> file.write("hello world in the
new file\n")
>>> file.write("and another line\n")
>>> file.close()
```

- If we now look in the newfile.txt, we can see the text that we wrote:

```
$ cat newfile.txt
hello world in the new file
and another line
```

Read a Text File

- To read a file, we can use different methods.
- If you want to return a string containing all characters in the file:

```
>>> file = open('newfile.txt', 'r')
>>> print file.read()
>>> file.close()
```

- This reads the first 5 characters of data and returns it as a string.

```
>>> file = open('newfile.txt', 'r')
>>> print file.read(5)
>>> file.close()
```

Read a Text File

- The readline() function will read from a file line by line (rather than pulling the entire file in at once). Basically, it will read a single line from the file and return a string containing characters up to \n.

```
>>> file = open('newfile.txt', 'r')
>>> print file.readline()
>>> file.close()
```

- readlines() returns the complete lines as a list of strings each separated by \n

```
>>> file = open('newfile.txt', 'r')
>>> print file.readlines()
>>> file.close()
```

- Output:

```
['hello world in the new file\n', 'and another line\n']
```

Looping over a file object

- For reading lines from a file, you can loop over the file object. This is memory efficient, fast, and leads to simple code.

```
>>> file = open('newfile.txt', 'r')
>>> for line in file:
>>>     print line,
```

Write to a File

- The write method takes one parameter, which is the string to be written. To start a new line after writing the data, add a \n character to the end.

```
>>> f = open("hello.txt","w")
>>> f.write("Hello World")
>>> f.close()
```

- Also, you can use:

```
>>> f = open("hello.txt", "w")
>>> lines_of_text = ["a line of
text\n", "another line of text\n",
"a third line\n"]
>>> f.writelines(lines_of_text)
>>> f.close()
```

Close a File

- When you're done with a file, call f.close() to close it and free up any system resources taken up by the open file.
- After calling f.close(), attempts to use the file object will automatically fail.

```
>>> f.close()
```

More About Files

- 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:

Attribute	Description
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.

An Advanced Example

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

File System

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

- Following is the example to rename an existing file *test1.txt*:

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

Remove

- Following is the example to delete an existing file *test2.txt*:

```
>>> 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.

Create a Directory

- 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.

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

Change Directory

- 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.

```
>>> import os
>>>
>>> #Changing a directory to "/home/test"
>>> os.chdir("/home/test")
```

Get Current Directory

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

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

Remove Directory

- 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.

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