Example

# 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

# Open a file

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

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

# Close opend file

fo.close()

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.

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()

This produces the following result −

Read String is : Python is

## 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 opend 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.

## The rename() Method

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* −

import os

# Rename a file from test1.txt to test2.txt

os.rename( "test1.txt", "test2.txt" )

## The *remove()* Method

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")

## 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" )

Open a File on the Server

Assume we have the following file, located in the same folder as Python:

demofile.txt

Hello! Welcome to demofile.txt  
This file is for testing purposes.  
Good Luck!

To open the file, use the built-in open() function.

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

Example

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

Read 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("demofile.txt", "r")  
print(f.read(**5**))

Read Lines

You can return one line by using the readline() method:

Example

Read one line of the file:

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

By calling readline() two times, you can read the two first lines:

Example

Read two lines of the file:

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

By looping through the lines of the file, you can read the whole file, line by line:

Example

Loop through the file line by line:

f = open("demofile.txt", "r")  
for x in f:  
  print(x)

Close Files

It is a good practice to always close the file when you are done with it.

Example

Close the file when you are finish with it:

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

Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Example

Open the file "demofile2.txt" and append content to the file:

f = open("demofile2.txt", "a")  
f.write("Now the file has more content!")  
f.close()  
  
#open and read the file after the appending:  
f = open("demofile2.txt", "r")  
print(f.read())

Example

Open the file "demofile3.txt" and overwrite the content:

f = open("demofile3.txt", "w")  
f.write("Woops! I have deleted the content!")  
f.close()  
  
#open and read the file after the appending:  
f = open("demofile3.txt", "r")  
print(f.read())

**Note:** the "w" method will overwrite the entire file.

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

Example

Create a file called "myfile.txt":

f = open("myfile.txt", "x")

Result: a new empty file is created!

Example

Create a new file if it does not exist:

f = open("myfile.txt", "w")

Delete a File

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

Example

Remove the file "demofile.txt":

import os  
os.remove("demofile.txt")

Check if File exist:

To avoid getting an error, you 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("demofile.txt"):  
  os.remove("demofile.txt")  
else:  
  print("The file does not exist")

Delete Folder

To delete an entire folder, use the os.rmdir() method:

Example

Remove the folder "myfolder":

import os  
os.rmdir("myfolder")

**Note:** You can only remove empty folders.