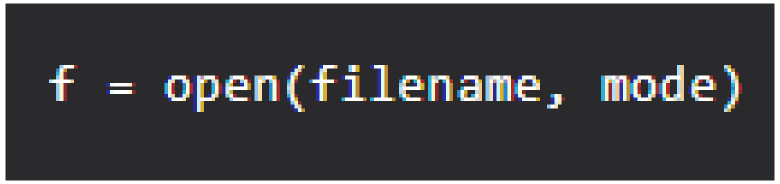
File Handling in Python and Python OS

Python File Handling

- Python too 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, but 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. Let’s start with the reading and writing files.

Working of open() function

- Before performing any operation on the file like reading or writing, first, we have to open that file.
- For this, we should use Python’s inbuilt function open() but at the time of opening, we have to specify the mode, which represents the purpose of the opening file



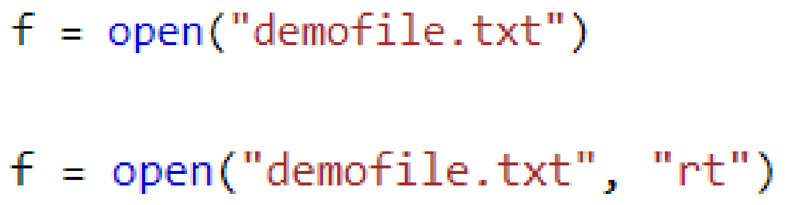
```
f = open(filename, mode)
```

Mode Support

- Where the following mode is supported:
- r: open an existing file for a read operation.
- w: open an existing file for a write operation. If the file already contains some data then it will be overridden but if the file is not present then it creates the file as well.
- a: open an existing file for append operation. It won’t override existing data.
- r+: To read and write data into the file. The previous data in the file will be overridden.
- w+: To write and read data. It will override existing data.
- a+: To append and read data from the file. It won’t override existing data.
- x: Creates the specified file, returns an error if the file exists
- t: Default value. Text mode
- b: Binary mode (e.g. images)

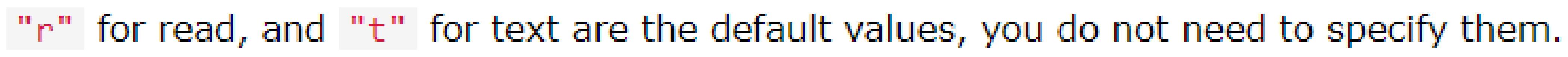
Syntax

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

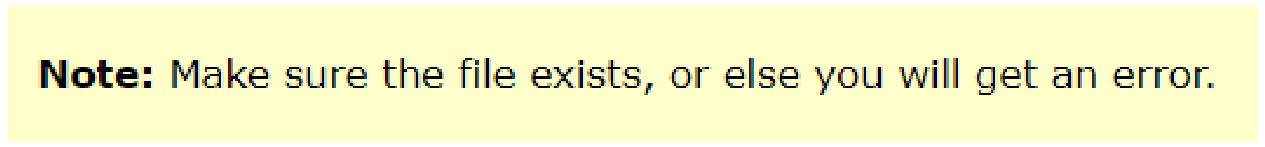


```
f = open("demofile.txt")

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

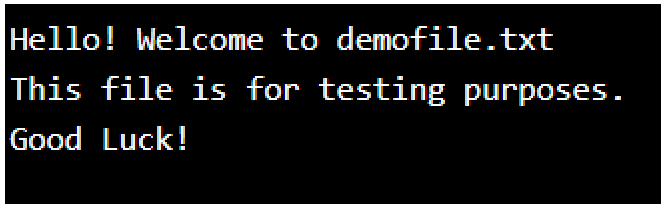


`"r"` for read, and `"t"` for text are the default values, you do not need to specify them.



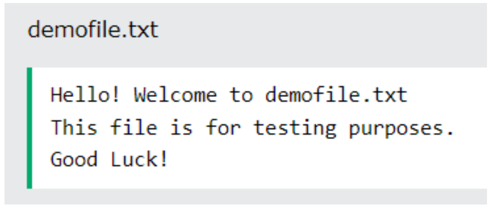
Note: Make sure the file exists, or else you will get an error.

Reading Files



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

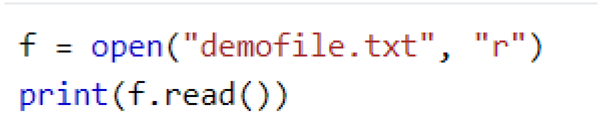
- Open a File on the Server
- Assume we have the following file, located in the same folder as Python:



demofile.txt

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

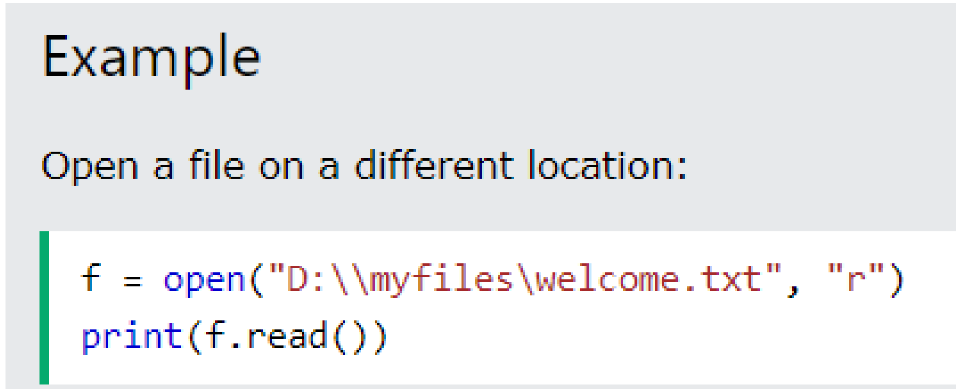
- To open the file, use the built-in open() function.
- The open() function returns a file object, which has a read() method for reading the content of the file:



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

Reading from different location

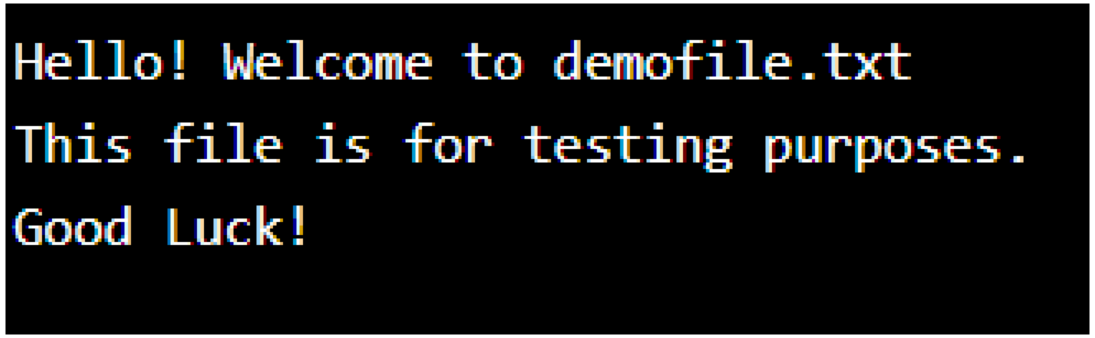
- If the file is located in a different location, you will have to specify the file path, like this:



Example

Open a file on a different location:

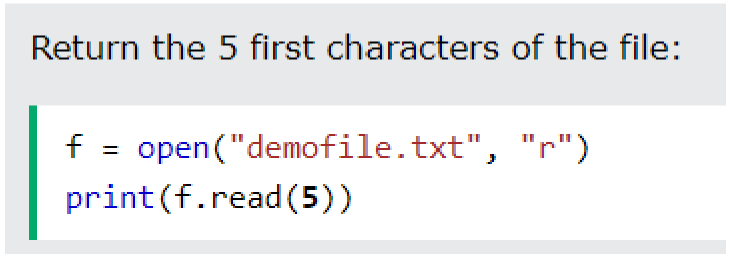
```
f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```



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

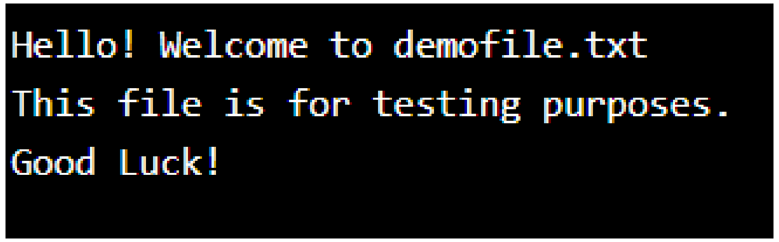
Reading Only Parts of the File

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

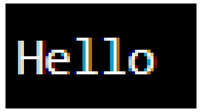


Return the 5 first characters of the file:

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



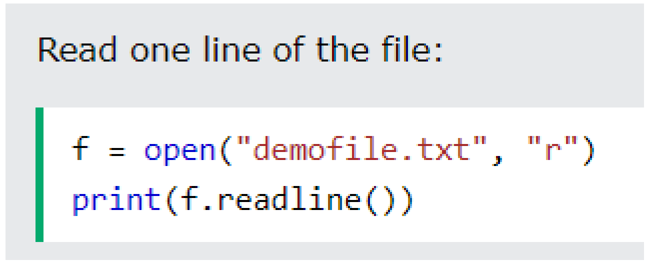
```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```



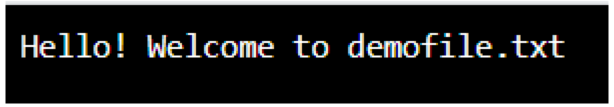
```
Hello
```

Reading Lines

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

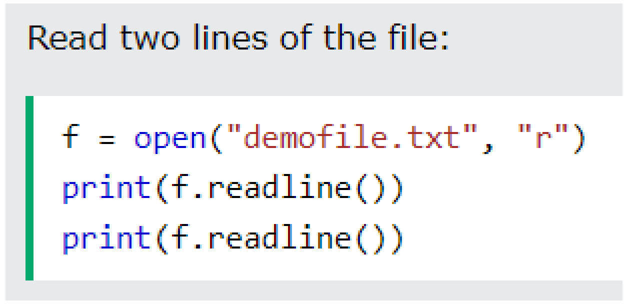

Read one line of the file:

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


```
Hello! Welcome to demofile.txt
```

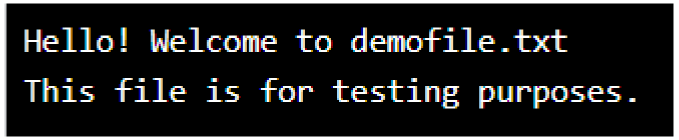
Reading Two Lines

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



Read two lines of the file:

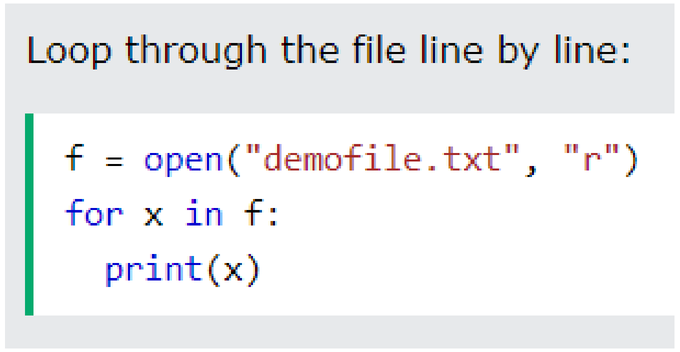
```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```



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

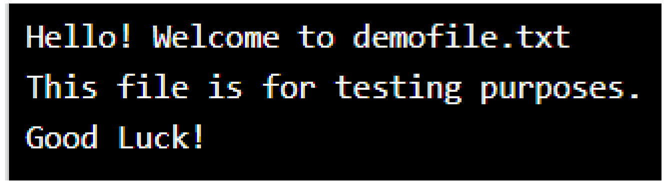
Looping through the Lines

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



Loop through the file line by line:

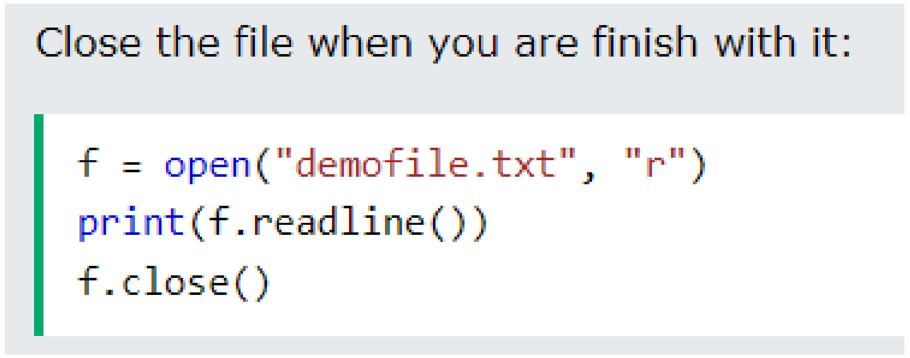
```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```



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

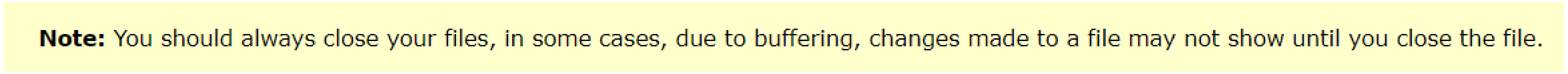
Close Files

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



Close the file when you are finish with it:

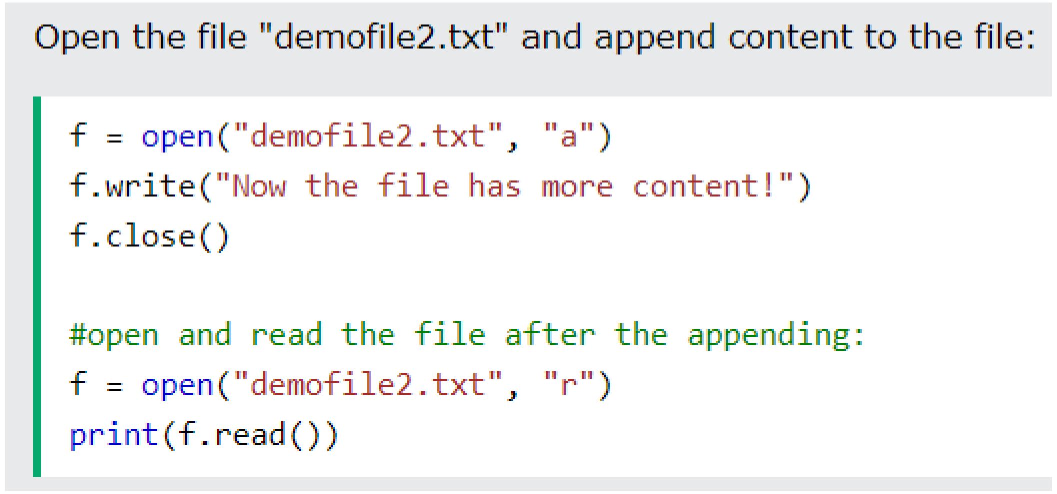
```python
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```



Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

Writing to an Existing File

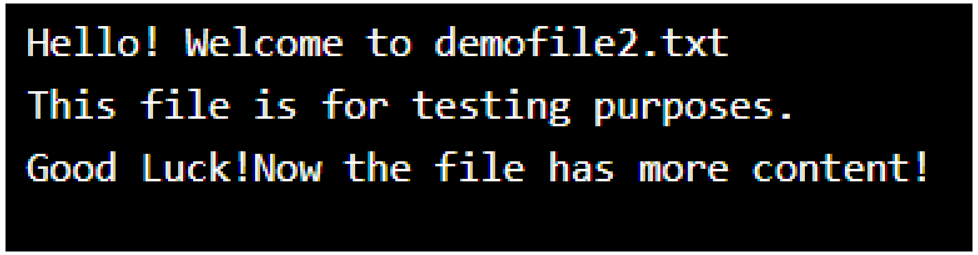
- To write to an existing file, you must add a parameter to the open() function:
- "a" - Append - will append to the end of the file
- "w" - Write - will overwrite any existing content



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

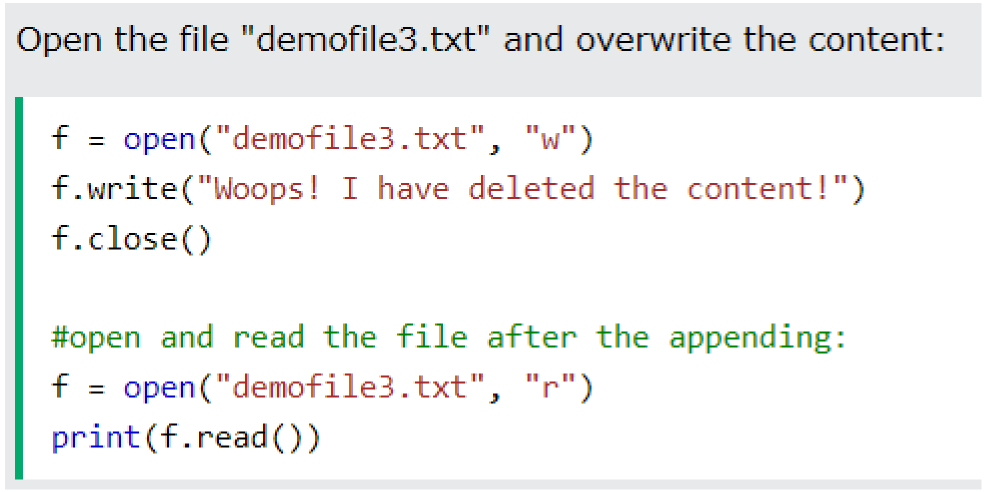
```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```



```
Hello! Welcome to demofile2.txt
This file is for testing purposes.
Good Luck!Now the file has more content!
```

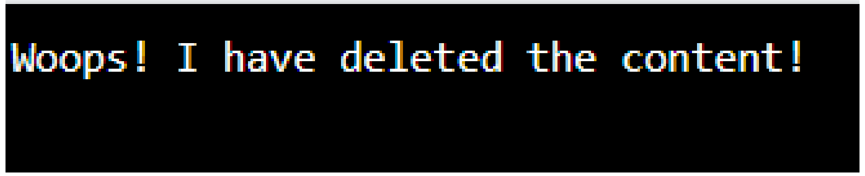
Writing to an Existing File



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

```python
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())
```

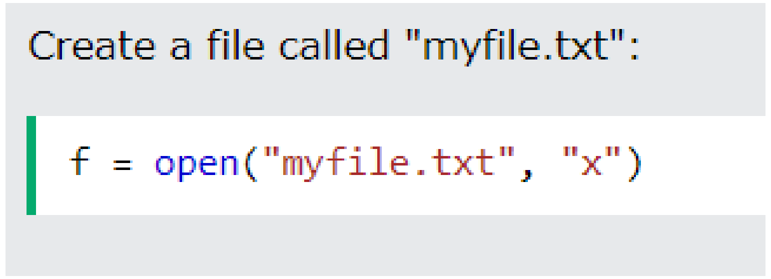


```
Woops! I have deleted the content!
```

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

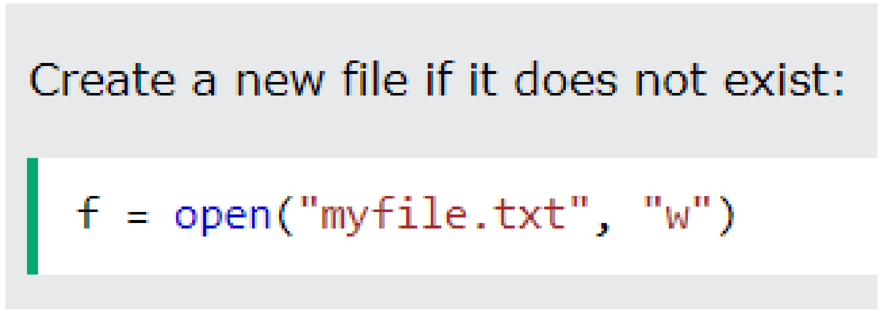
Creating a New File

- 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



Create a file called "myfile.txt":

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



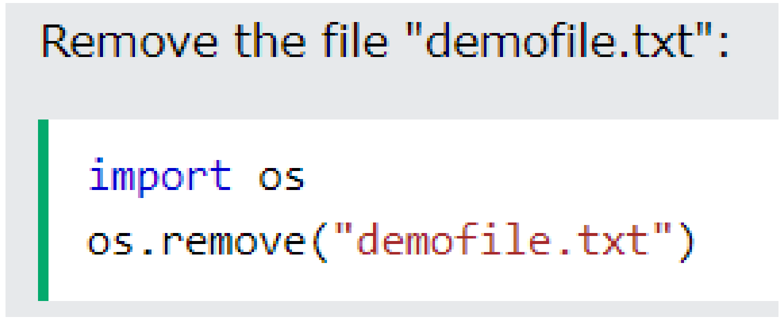
Create a new file if it does not exist:

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

Result: a new empty file is created!

Deleting A File

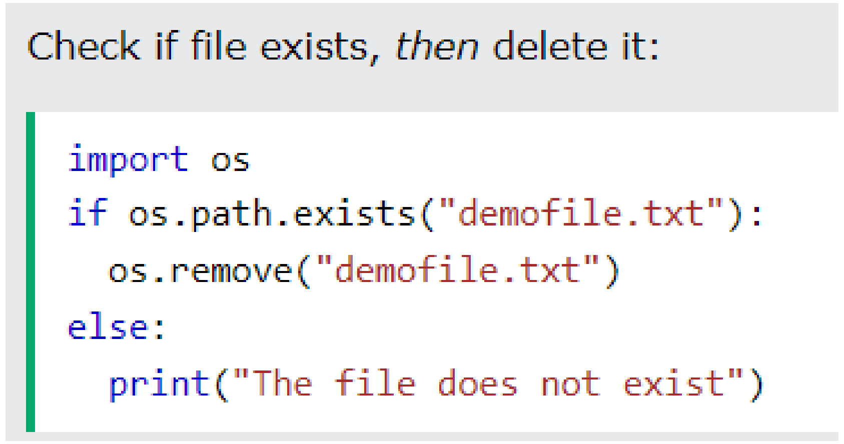
- To delete a file, you must import the OS module, and run its os.remove() function:


Remove the file "demofile.txt":

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

Checking if a file exist

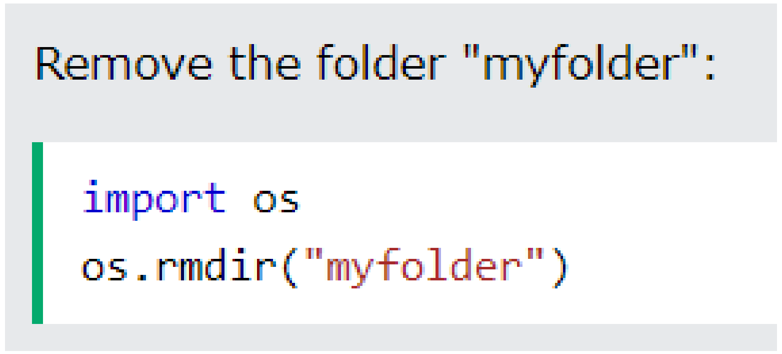
- To avoid getting an error, you might want to check if the file exists before you try to delete it:


Check if file exists, *then* delete it:

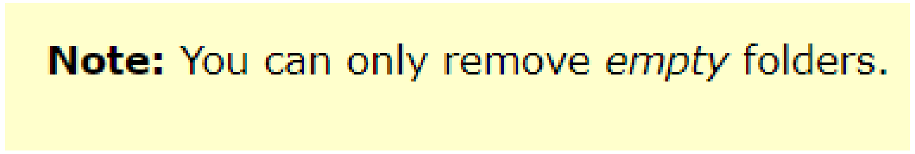
```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Deleting Folders

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


Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```

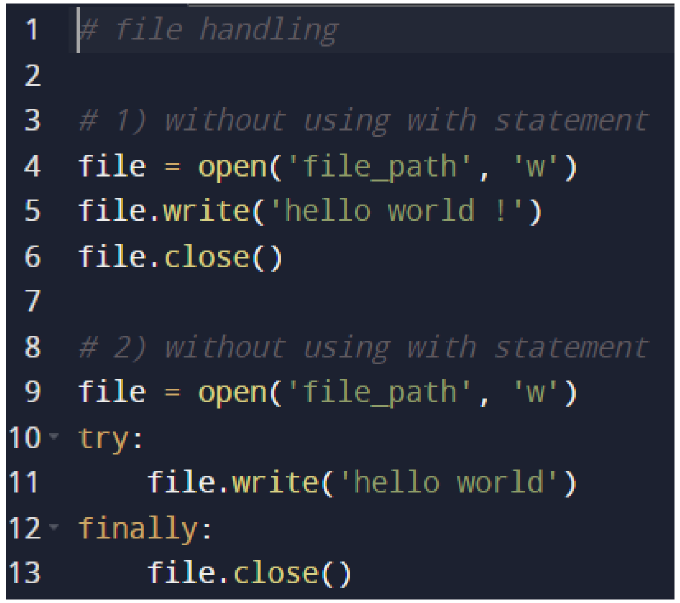

Note: You can only remove *empty* folders.

With Keyword

- In Python, with statement is used in exception handling to make the code cleaner and much more readable.
- It simplifies the management of common resources like file streams.
- Observe the following code example on how the use of with statement makes code cleaner.

with statement in Python

- In Python, with statement is used in exception handling to make the code cleaner and much more readable.
- It simplifies the management of common resources like file streams. Observe the following code example on how the use of with statement makes code cleaner.



```python
# file handling

# 1) without using with statement
file = open('file_path', 'w')
file.write('hello world !')
file.close()

# 2) without using with statement
file = open('file_path', 'w')
try:
    file.write('hello world')
finally:
    file.close()
```

Advantages of with

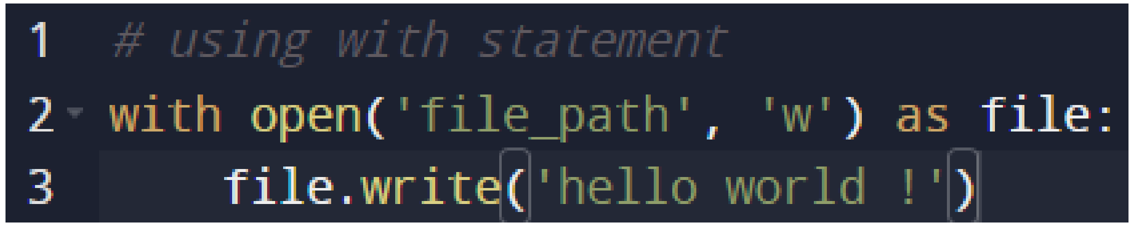
Notice that unlike the first two implementations, there is no need to call file.close() when using with statement.

The with statement itself ensures proper acquisition and release of resources. An exception during the file.write() call in the first implementation can prevent the file from closing properly which may introduce several bugs in the code, i.e. many changes in files do not go into effect until the file is properly closed.

The second approach in the above example takes care of all the exceptions but using the with statement makes the code compact and much more readable. Thus, with statement helps avoiding bugs and leaks by ensuring that a resource is properly released when the code using the resource is completely executed. The with statement is popularly used with file streams, as shown above and with Locks, sockets, subprocesses and telnets etc.

Supporting the “with” statement in user defined objects

There is nothing special in open() which makes it usable with the with statement and the same functionality can be provided in user defined objects. Supporting with statement in your objects will ensure that you never leave any resource open. To use with statement in user defined objects you only need to add the methods __enter__() and __exit__() in the object methods. Consider the following example for further clarification.



```python
# using with statement
with open('file_path', 'w') as file:
    file.write('hello world !')
```

Using the with open() syntax in Python for file I/O operations offers several advantages:

- Automatic resource management
- Cleaner code
- Improved error handling
- Support for context managers
- Scoping

- Provides safer and more efficient file handling in Python, with built-in support for resource management, error handling, and scoping.
- It promotes cleaner and more readable code while reducing the risk of common programming errors related to file operations.

Python OS

Python os Module

- Python has a built-in os module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.

- The os module has the following set of methods and constants.

Python-OS-Module Functions

- Handling the Current Working Directory
- Creating a Directory
- Listing out Files and Directories with Python
- Deleting Directory or Files using Python

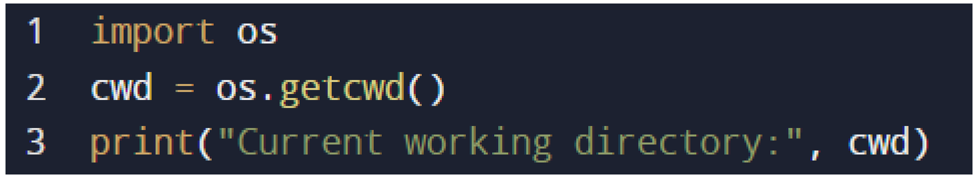
Handling the Current Working Directory

- Consider Current Working Directory(CWD) as a folder, where Python is operating. Whenever the files are called only by their name, Python assumes that it starts in the CWD which means that name-only reference will be successful only if the file is in the Python's CWD.
- Note: The folder where the Python script is running is known as the Current Directory. This is not the path where the Python script is located.

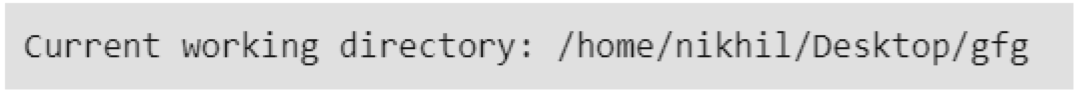
Getting the Current working directory

To get the location of the current working directory os.getcwd() is used.

Example: This code uses the ‘os' module to get and print the current working directory (CWD) of the Python script. It retrieves the CWD using the ‘os.getcwd()' and then prints it to the console.



```python
import os
cwd = os.getcwd()
print("Current working directory:", cwd)
```



```
Current working directory: /home/nikhil/Desktop/gfg
```

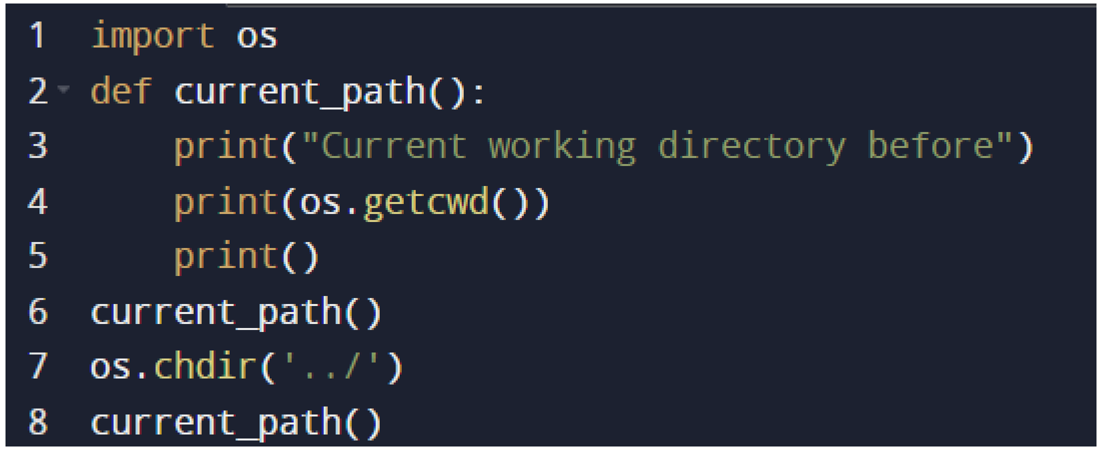
Changing the Current working directory

To change the current working directory(CWD) os.chdir() method is used. This method changes the CWD to a specified path. It only takes a single argument as a new directory path.

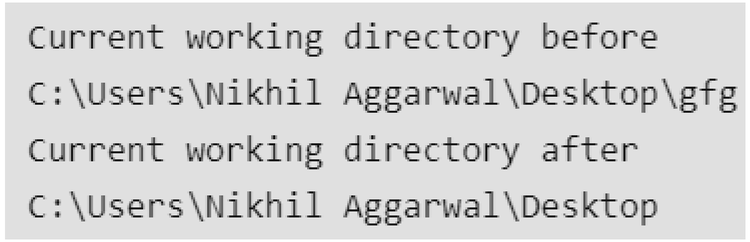
Note: The current working directory is the folder in which the Python script is operating.

Example: The code checks and displays the current working directory (CWD) twice: before and after changing the directory up one level using os.chdir('../’).

It provides a simple example of how to work with the current working directory in Python.



```python
import os
def current_path():
    print("Current working directory before")
    print(os.getcwd())
    print()
current_path()
os.chdir('../')
current_path()
```



```
Current working directory before
C:\Users\Nikhil Aggarwal\Desktop\gfg
Current working directory after
C:\Users\Nikhil Aggarwal\Desktop
```

Creating a Directory

- There are different methods available in the OS module for creating a directory. These are –

os.mkdir()

os.makedirs()

Using os.mkdir()

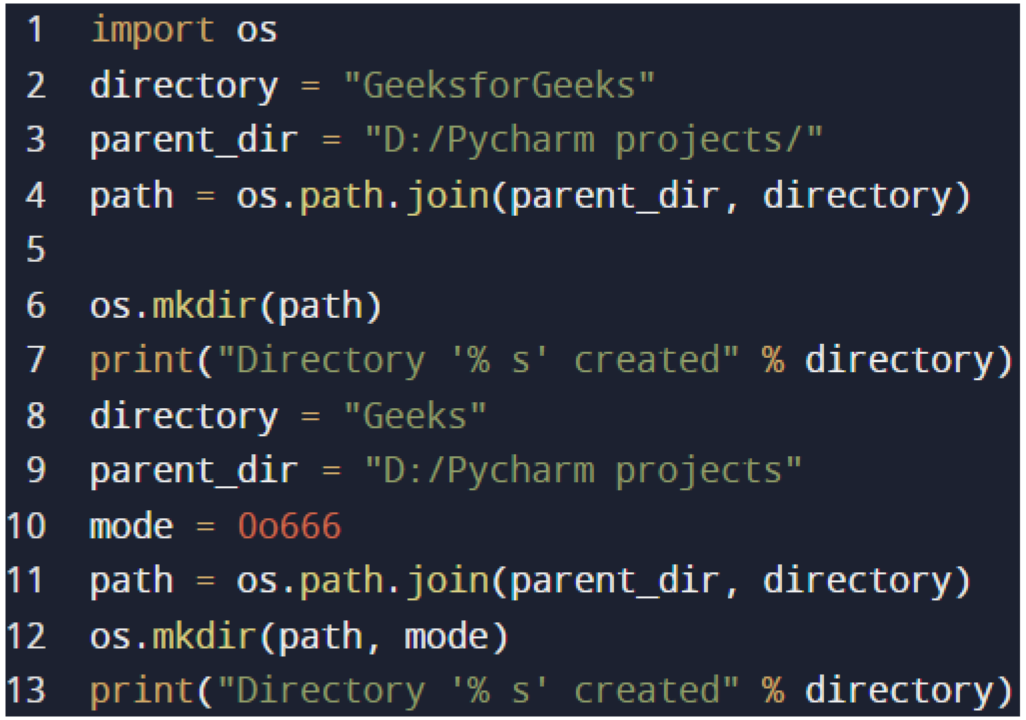
- By using os.mkdir() method in Python is used to create a directory named path with the specified numeric mode.
- This method raises FileExistsError if the directory to be created already exists.
- Example: This code creates two directories: “GeeksforGeeks” within the “D:/Pycharm projects/” directory and “Geeks” within the “D:/Pycharm projects” directory.

Using os.mkdir() example

The first directory is created using the os.mkdir() method without specifying the mode.

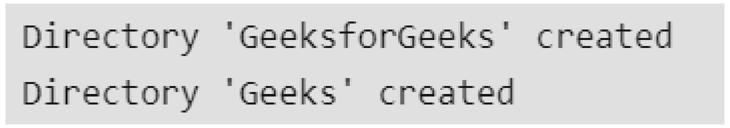
The second directory is created using the same method, but a specific mode (0o666) is provided, which grants read and write permissions.

The code then prints messages to indicate that the directories have been created.



```python
import os
directory = "GeeksforGeeks"
parent_dir = "D:/Pycharm projects/"
path = os.path.join(parent_dir, directory)

os.mkdir(path)
print("Directory '% s' created" % directory)
directory = "Geeks"
parent_dir = "D:/Pycharm projects"
mode = 0o666
path = os.path.join(parent_dir, directory)
os.mkdir(path, mode)
print("Directory '% s' created" % directory)
```



```
Directory 'GeeksforGeeks' created
Directory 'Geeks' created
```

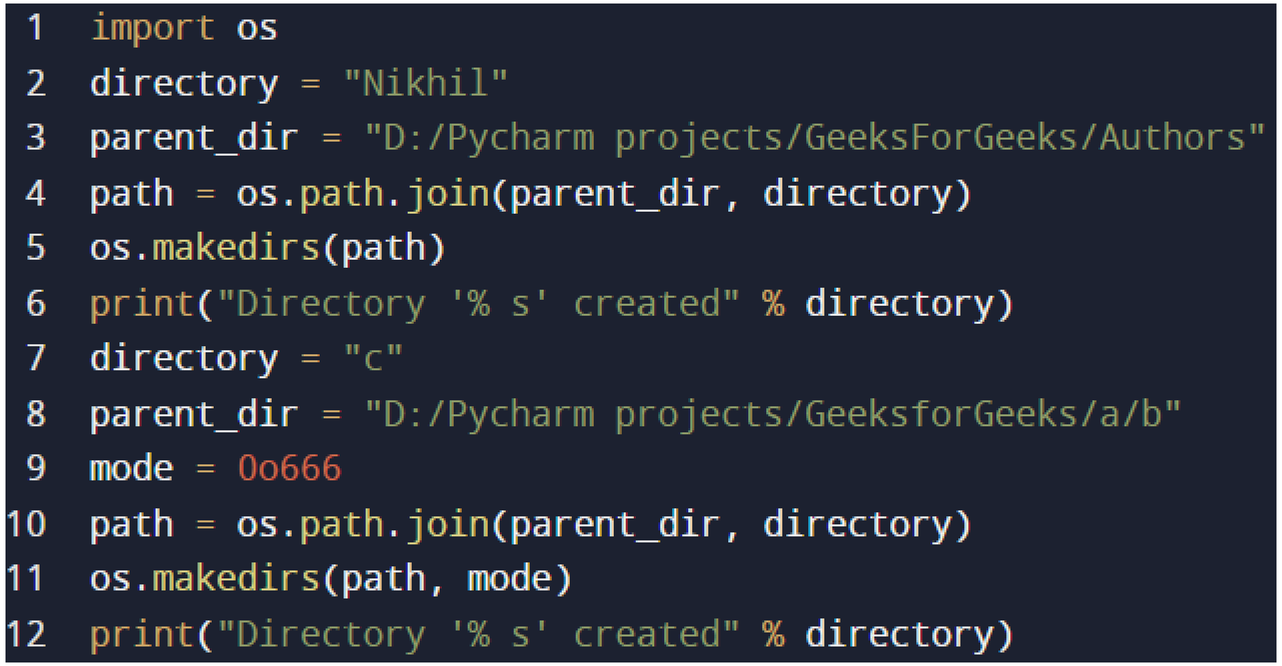
Using os.makedirs()

- os.makedirs() method in Python is used to create a directory recursively. That means while making leaf directory if any intermediate-level directory is missing, os.makedirs() method will create them all.

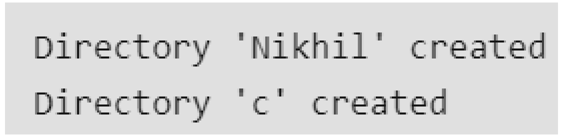
Using os.makedirs() example

Example: This code creates two directories, “Nikhil” and “c”, within different parent directories. It uses the os.makedirs function to ensure that parent directories are created if they don’t exist.

It also sets the permissions for the “c” directory. The code prints messages to confirm the creation of these directories



```python
import os
directory = "Nikhil"
parent_dir = "D:/Pycharm projects/GeeksForGeeks/Authors"
path = os.path.join(parent_dir, directory)
os.makedirs(path)
print("Directory '% s' created" % directory)
directory = "c"
parent_dir = "D:/Pycharm projects/GeeksforGeeks/a/b"
mode = 0o666
path = os.path.join(parent_dir, directory)
os.makedirs(path, mode)
print("Directory '% s' created" % directory)
```



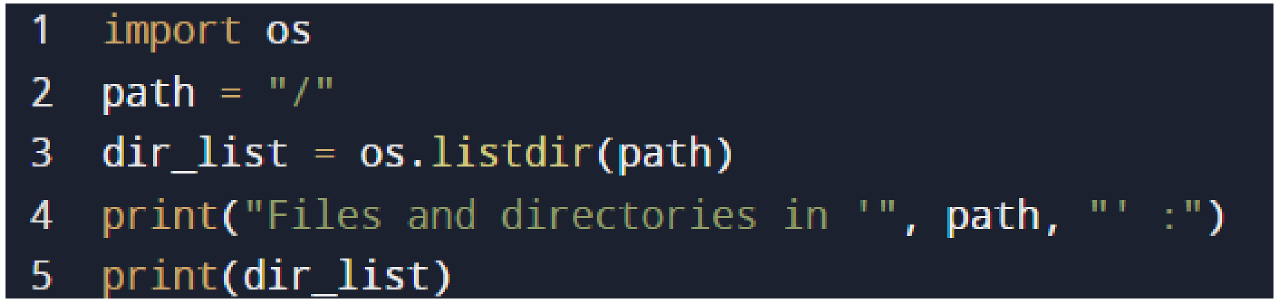
```
Directory 'Nikhil' created
Directory 'c' created
```

Listing out Files and Directories with Python

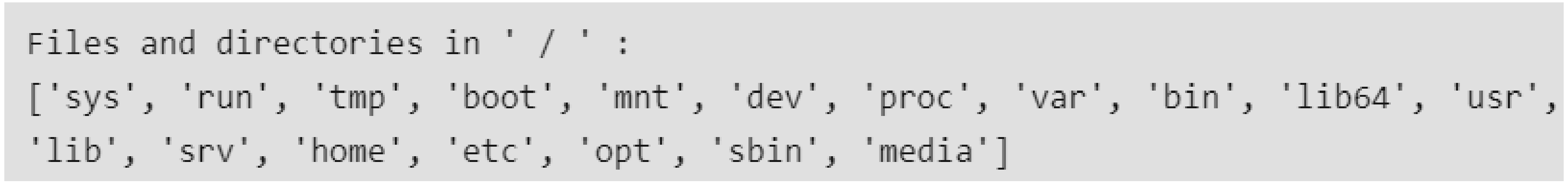
- There is os.listdir() method in Python is used to get the list of all files and directories in the specified directory. If we don’t specify any directory, then the list of files and directories in the current working directory will be returned.

Listing out Files and Directories with Python example

Example: This code lists all the files and directories in the root directory (“/”). It uses the os.listdir function to get the list of files and directories in the specified path and then prints the results.



```python
import os
path = "/"
dir_list = os.listdir(path)
print("Files and directories in '", path, "' :")
print(dir_list)
```



```
Files and directories in ' / ' :
['sys', 'run', 'tmp', 'boot', 'mnt', 'dev', 'proc', 'var', 'bin', 'lib64', 'usr',
'lib', 'srv', 'home', 'etc', 'opt', 'sbin', 'media']
```

Deleting Directory or Files using Python

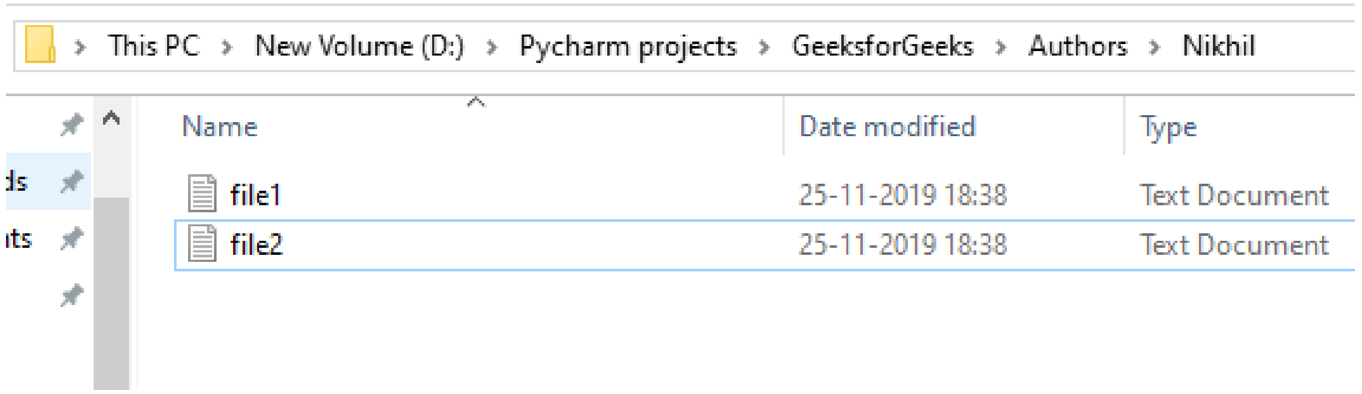
- OS module proves different methods for removing directories and files in Python. These are –

- Using os.remove()
- Using os.rmdir()

Using os.remove() Method

- os.remove() method in Python is used to remove or delete a file path. This method can not remove or delete a directory. If the specified path is a directory then OSError will be raised by the method.

- Example: Suppose the file contained in the folder are:

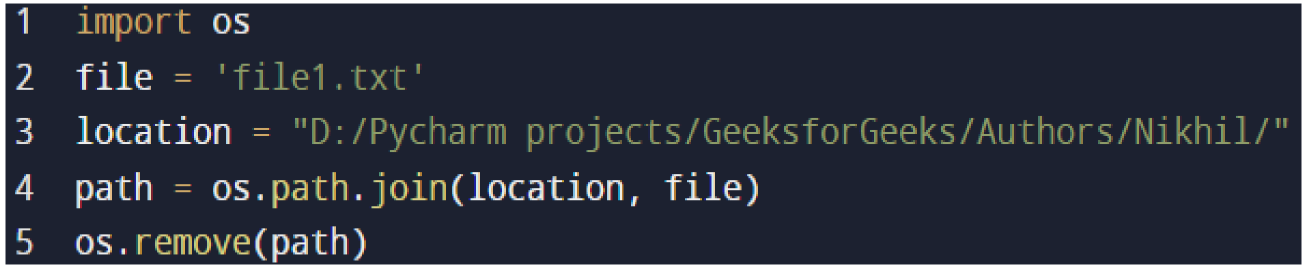


This PC › New Volume (D:) › Pycharm projects › GeeksforGeeks › Authors › Nikhil

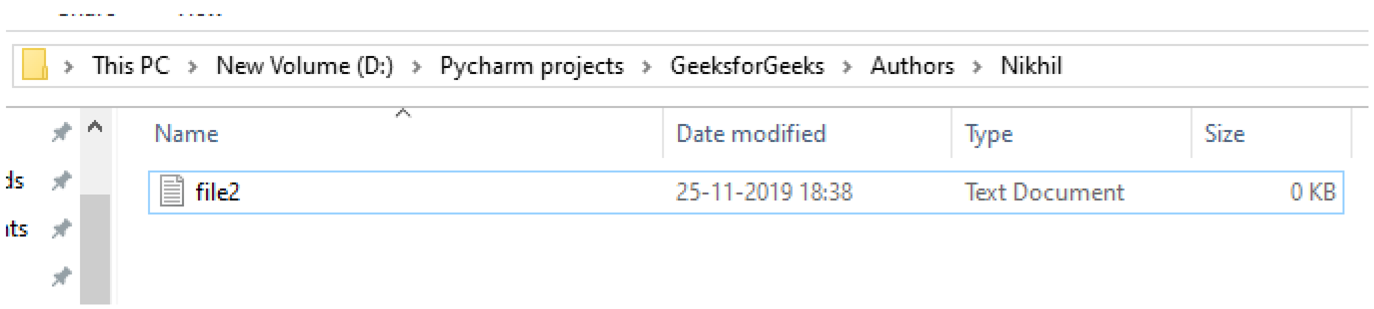
Name	Date modified	Type
file1	25-11-2019 18:38	Text Document
file2	25-11-2019 18:38	Text Document

Using os.remove() Method Example

This code removes a file named “file1.txt” from the specified location “D:/Pycharm projects/GeeksforGeeks/Authors/Nikhil/”. It uses the os.remove function to delete the file at the specified path.



```python
import os
file = 'file1.txt'
location = "D:/Pycharm projects/GeeksforGeeks/Authors/Nikhil/"
path = os.path.join(location, file)
os.remove(path)
```



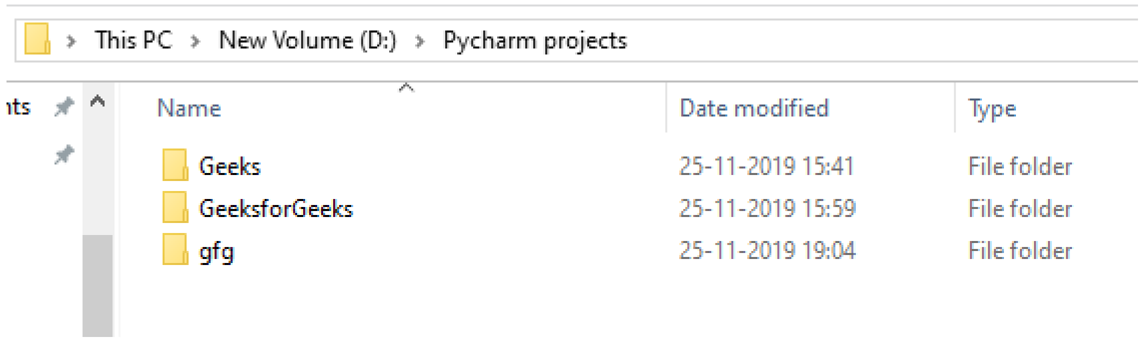
This PC › New Volume (D:) › Pycharm projects › GeeksforGeeks › Authors › Nikhil

Name	Date modified	Type	Size
file2	25-11-2019 18:38	Text Document	0 KB

Using os.rmdir()

- os.rmdir() method in Python is used to remove or delete an empty directory. OSError will be raised if the specified path is not an empty directory.

- Example: Suppose the directories are



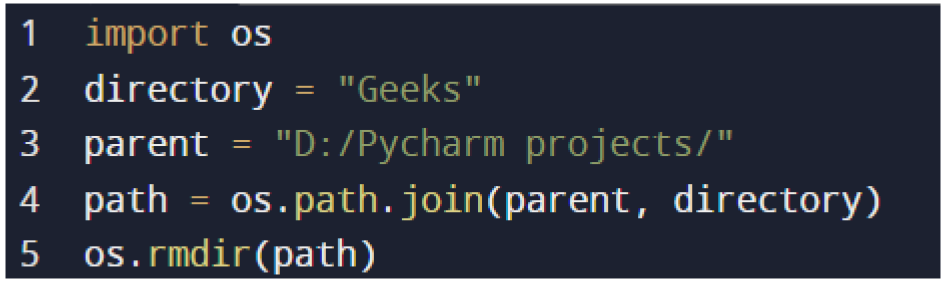
This PC › New Volume (D:) › Pycharm projects

Name	Date modified	Type
Geeks	25-11-2019 15:41	File folder
GeeksforGeeks	25-11-2019 15:59	File folder
gfg	25-11-2019 19:04	File folder

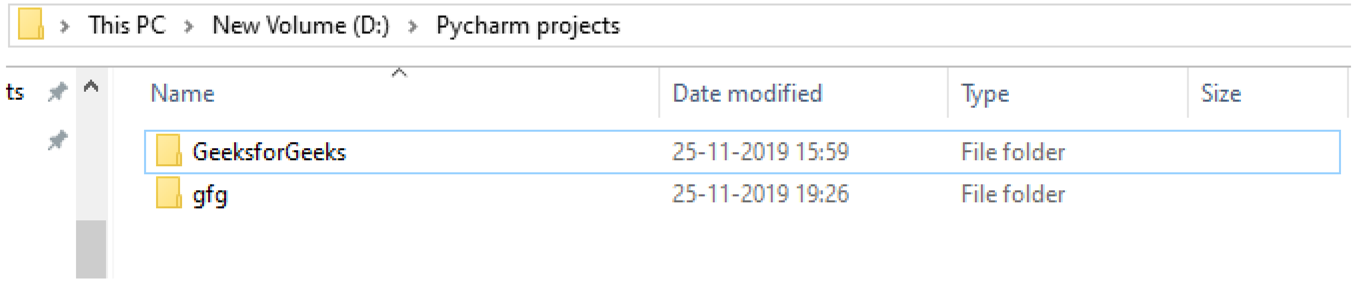
Using os.rmdir() example

This code attempts to remove a directory named “Geeks” located at “D:/Pycharm projects/”.

It uses the os.rmdir function to delete the directory. If the directory is empty, it will be removed. If it contains files or subdirectories, you may encounter an error.



```python
import os
directory = "Geeks"
parent = "D:/Pycharm projects/"
path = os.path.join(parent, directory)
os.rmdir(path)
```



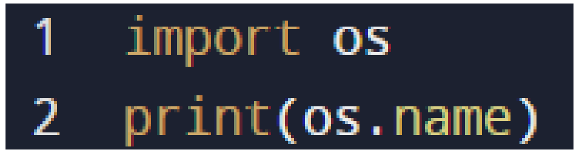
```
This PC > New Volume (D:) > Pycharm projects

Name                Date modified       Type          Size
GeeksforGeeks       25-11-2019 15:59    File folder
gfg                 25-11-2019 19:26    File folder
```

Commonly Used Python OS Functions

Using os.name function

- This function gives the name of the operating system dependent module imported. The following names have currently been registered: ‘posix’, ‘nt’, ‘os2’, ‘ce’, ‘java’ and ‘riscos’.
- Note: It may give different output on different interpreters.



```python
import os
print(os.name)
```

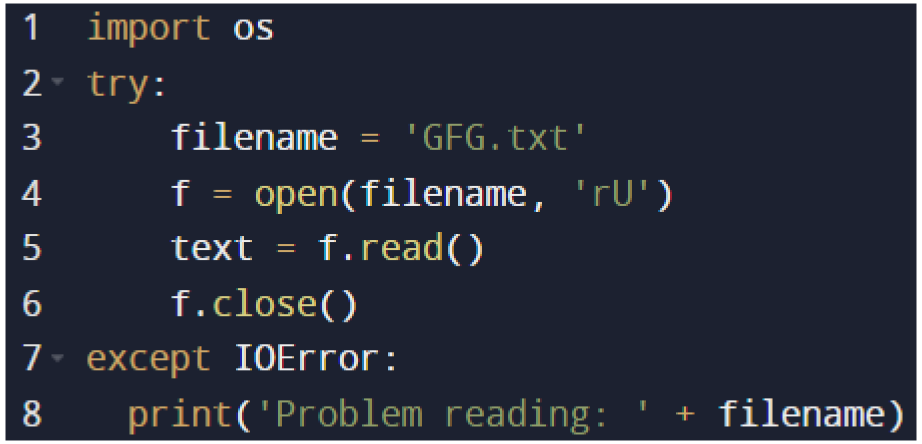
Using os.error Function

- All functions in this module raise OSError in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system. os.error is an alias for built-in OSError exception.

Using os.error Function Example

This code reads the contents of a file named ‘GFG.txt’. It uses a ‘try…except‘ block to handle potential errors, particularly the ‘IOError‘ that may occur if there’s a problem reading the file.

If an error occurs, it will print a message saying, “Problem reading: GFG.txt.”

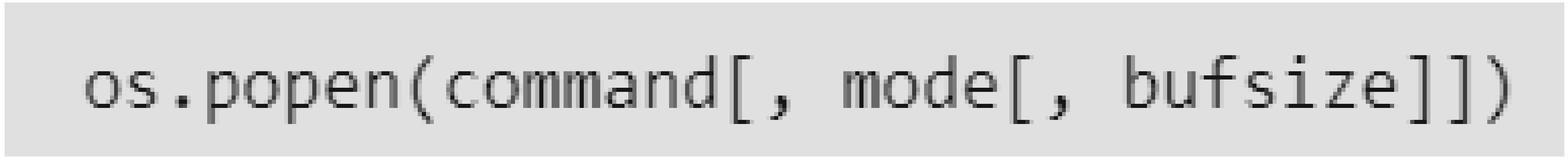


```python
import os
try:
        filename = 'GFG.txt'
        f = open(filename, 'rU')
        text = f.read()
        f.close()
except IOError:
    print('Problem reading: ' + filename)
```

```
Problem reading: GFG.txt
```

Using os.popen() Function

- This method opens a pipe to or from command. The return value can be read or written depending on whether the mode is ‘r’ or ‘w’.
- Parameters mode & bufsize are not necessary parameters, if not provided, default ‘r’ is taken for mode.



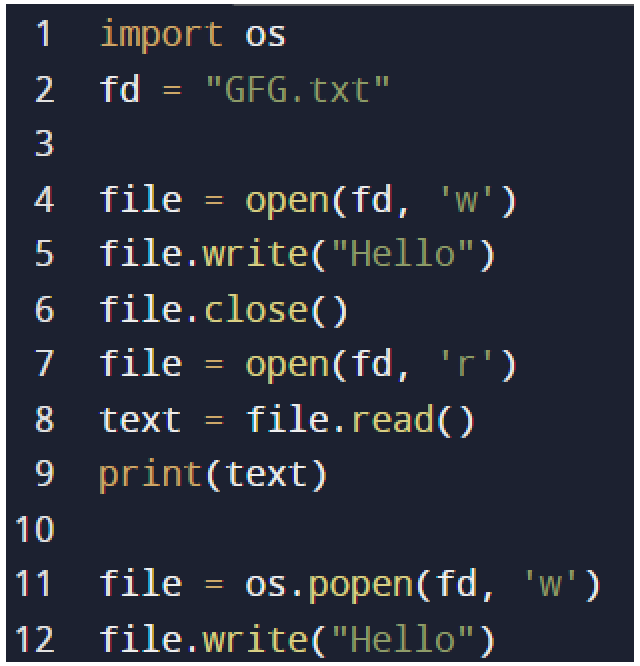
```
os.popen(command[, mode[, bufsize]])
```

Using os.popen() Function Example

This code opens a file named ‘GFG.txt’ in write mode, writes “Hello” to it, and then reads and prints its contents.

The use of os.popen is not recommended, and standard file operations are used for these tasks.

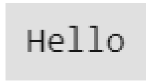
Note: Output for popen() will not be shown, there would be direct changes into the file.



```python
import os
fd = "GFG.txt"

file = open(fd, 'w')
file.write("Hello")
file.close()
file = open(fd, 'r')
text = file.read()
print(text)

file = os.popen(fd, 'w')
file.write("Hello")
```



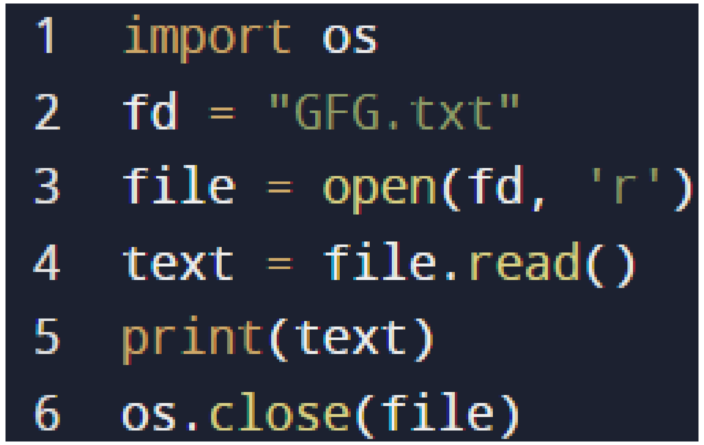
```
Hello
```

Using os.close() Function

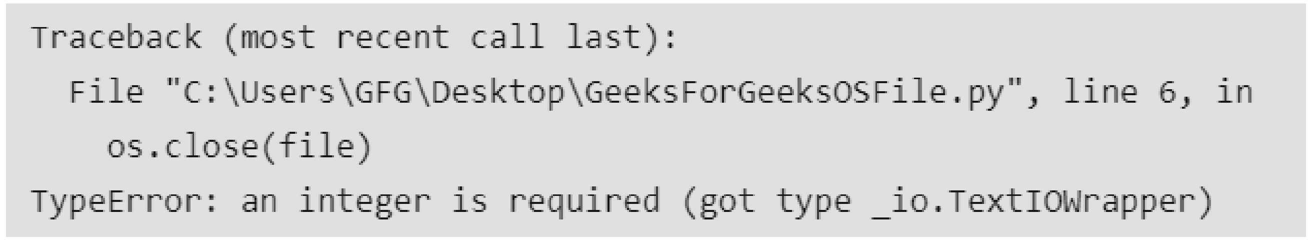
- Close file descriptor fd. A file opened using open(), can be closed by close()only.
- But file opened through os.popen(), can be closed with close() or os.close().
- If we try closing a file opened with open(), using os.close(), Python would throw TypeError.

Using os.close() Function example

Note: The same error may not be thrown, due to the non-existent file or permission privilege.



```python
import os
fd = "GFG.txt"
file = open(fd, 'r')
text = file.read()
print(text)
os.close(file)
```



```
Traceback (most recent call last):
  File "C:\Users\GFG\Desktop\GeeksForGeeksOSFile.py", line 6, in
    os.close(file)
TypeError: an integer is required (got type _io.TextIOWrapper)
```

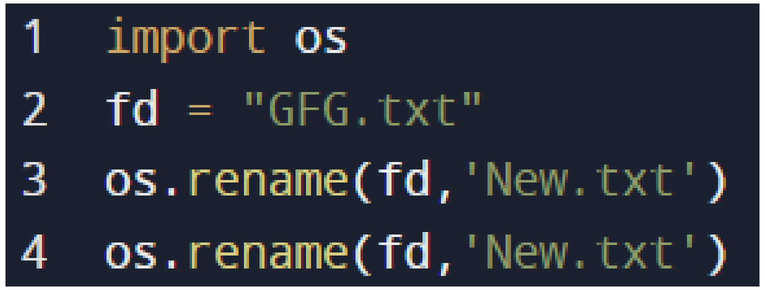
Using os.rename() Function

A file old.txt can be renamed to new.txt, using the function os.rename().

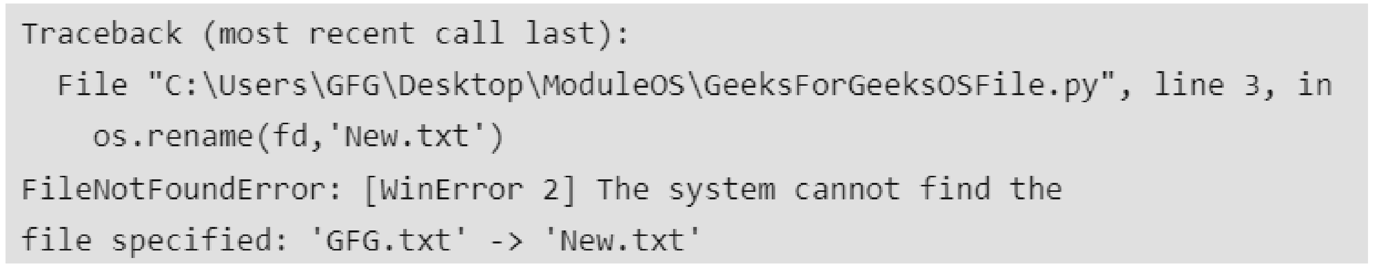
The name of the file changes only if, the file exists and the user has sufficient privilege permission to change the file.

A file name "GFG.txt" exists, thus when os.rename() is used the first time, the file gets renamed.

Upon calling the function os.rename() second time, file "New.txt" exists and not "GFG.txt" thus Python throws FileNotFoundError.



```python
import os
fd = "GFG.txt"
os.rename(fd,'New.txt')
os.rename(fd,'New.txt')
```



```
Traceback (most recent call last):
  File "C:\Users\GFG\Desktop\ModuleOS\GeeksForGeeksOSFile.py", line 3, in
    os.rename(fd,'New.txt')
FileNotFoundError: [WinError 2] The system cannot find the
file specified: 'GFG.txt' -> 'New.txt'
```

Using os.remove() Function Example

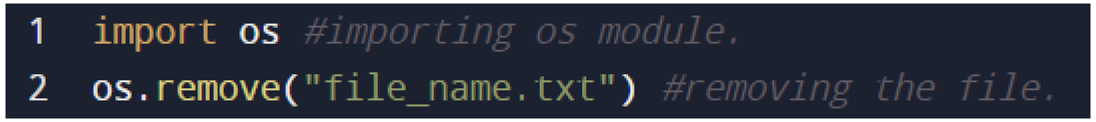
Using the OS module we can remove a file in our system using the os.remove() method.

To remove a file we need to pass the name of the file as a parameter.

The OS module provides us a layer of abstraction between us and the operating system.

When we are working with os module always specify the absolute path depending upon the operating system the code can run on any os but we need to change the path exactly.

If you try to remove a file that does not exist you will get FileNotFoundError.



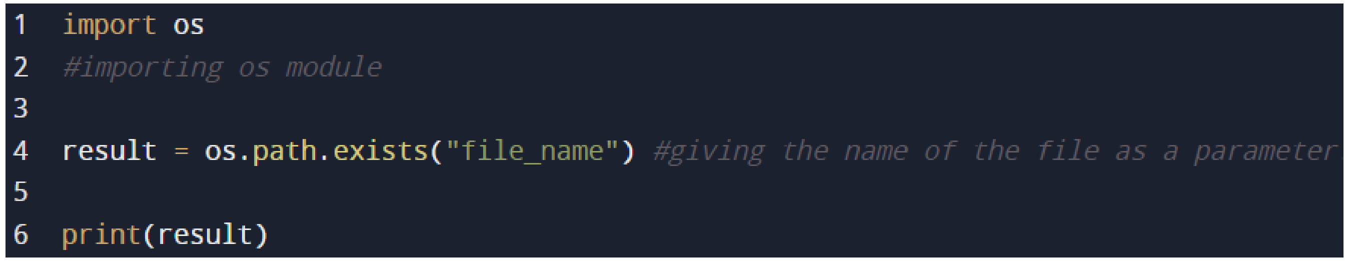
```python
import os #importing os module.
os.remove("file_name.txt") #removing the file.
```

Using os.path.exists() Function Example

This method will check whether a file exists or not by passing the name of the file as a parameter. OS module has a sub-module named PATH by using which we can perform many more functions.

As in the code, the file does not exist it will give output False.

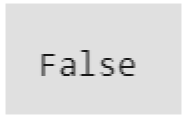
If the file exists it will give us output True.



```python
import os
#importing os module

result = os.path.exists("file_name") #giving the name of the file as a parameter

print(result)
```

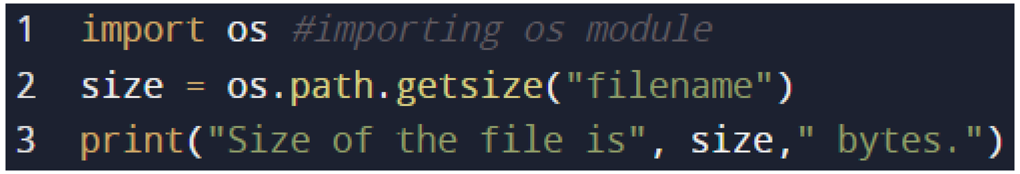


```
False
```

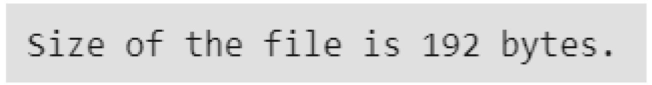
Using os.path.getsize() Function Example

In os.path.getsize() function, python will give us the size of the file in bytes.

To use this method we need to pass the name of the file as a parameter.



```python
import os #importing os module
size = os.path.getsize("filename")
print("Size of the file is", size," bytes.")
```



```
Size of the file is 192 bytes.
```