

# OS Module in Python

The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. The `*os*` and `*os.path*` modules include many functions to interact with the file system.

## Handling the Current Working Directory

Consider the Current **Working Directory(CWD)** as a folder, where the 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 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.

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

**Example:**

```
# Python program to explain os.getcwd()
method

# importing os module
import os

# Get the current working
# directory (CWD)
cwd = os.getcwd()

# Print the current working
# directory (CWD)
print("Current working directory:",
      cwd)
```

### **Output:**

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

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:**

```
# Python program to change the
# current working directory

import os

# Function to Get the current
# working directory
def current_path():
    print("Current working directory
before")
    print(os.getcwd())
    print()

# Driver's code
# Printing CWD before
current_path()

# Changing the CWD
os.chdir('../')

# Printing CWD after
current_path()
```

## Output:

```
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 director.

These are –

- `os.mkdir()`
- `os.makedirs()`

### Using `os.mkdir()`

`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:

```
# Python program to explain os.mkdir()
method
```

```
# importing os module
import os
```

```
# Directory
directory = "GeeksforGeeks"
```

```
# Parent Directory path
parent_dir = "D:/Pycharm projects/"
```

```
# Path
path = os.path.join(parent_dir,
directory)
```

```
# Create the directory
# 'GeeksForGeeks' in
# '/home / User / Documents'
os.mkdir(path)
print("Directory '% s' created" %
directory)
```

```
# Directory
directory = "Geeks"
```

```
# Parent Directory path
parent_dir = "D:/Pycharm projects"
```

```
# mode
mode = 0o666
```

```
# Path
path = os.path.join(parent_dir,
directory)
```

```
# Create the directory
# 'GeeksForGeeks' in
```

```
# '/home / User / Documents'  
# with mode 0o666  
os.mkdir(path, mode)  
print("Directory '% s' created" %  
directory)
```

### **Output:**

```
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.

### **Example:**

```
# Python program to explain os.makedirs() method

# importing os module
import os

# Leaf directory
directory = "Nikhil"

# Parent Directories
parent_dir = "D:/Pycharm
projects/GeeksForGeeks/Authors"

# Path
path = os.path.join(parent_dir, directory)

# Create the directory
# 'Nikhil'
os.makedirs(path)
print("Directory '% s' created" % directory)

# Directory 'GeeksForGeeks' and 'Authors' will
# be created too
# if it does not exists


# Leaf directory
directory = "c"

# Parent Directories
parent_dir = "D:/Pycharm
projects/GeeksforGeeks/a/b"

# mode
mode = 0o666

path = os.path.join(parent_dir, directory)
```

```
# Create the directory 'c'

os.makedirs(path, mode)
print("Directory '% s' created" % directory)


# 'GeeksForGeeks', 'a', and 'b'
# will also be created if
# it does not exists


# If any of the intermediate level
# directory is missing
# os.makedirs() method will
# create them


# os.makedirs() method can be
# used to create a directory tree
```

### **Output:**

```
Directory 'Nikhil' created
```

```
Directory 'c' created
```

## **Listing out Files and Directories with Python**



**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.

### Example:

- Python3

```
# Python program to explain os.listdir()
method

# importing os module
import os

# Get the list of all files and
directories
# in the root directory
path = "/"
dir_list = os.listdir(path)

print("Files and directories in '", path,
      "':")

# print the list
print(dir_list)
```

### Output:

```
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

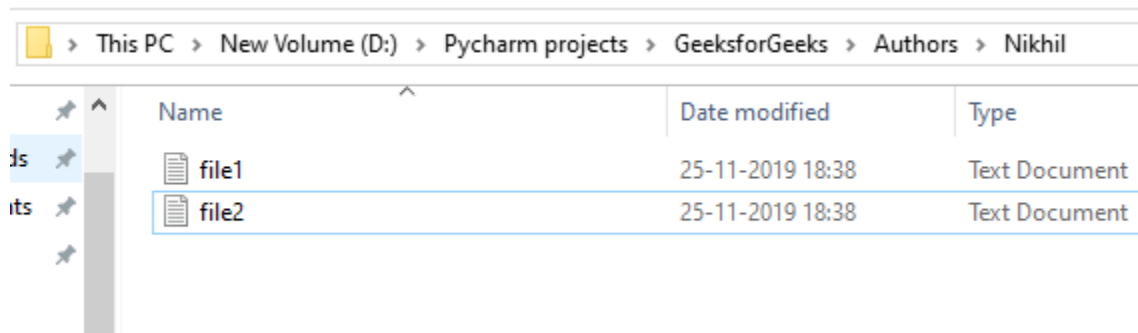
The OS module provides different methods for removing directories and files in Python. These are –

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

### Using `os.remove()`

`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

```
# Python program to explain os.remove() method

# importing os module
import os


# File name
file = 'file1.txt'

# File location
location = "D:/Pycharm
projects/GeeksforGeeks/Authors/Nikhil/"

# Path
path = os.path.join(location, file)

# Remove the file
# 'file.txt'
os.remove(path)
e)
```


**Output:**

 > This PC > New Volume (D:) > Pycharm projects > GeeksforGeeks > Authors > Nikhil				
	Name	Date modified	Type	Size
<div> <div>its</div> <div>file2</div> </div>	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
<div> <div>its</div> <div>Geeks</div> </div>	Geeks	25-11-2019 15:41	File folder
<div> <div>its</div> <div>GeeksforGeeks</div> </div>	GeeksforGeeks	25-11-2019 15:59	File folder
<div> <div>its</div> <div>gfg</div> </div>	gfg	25-11-2019 19:04	File folder

```

# Python program to explain os.rmdir()
method

# importing os module
import os

# Directory name
directory = "Geeks"

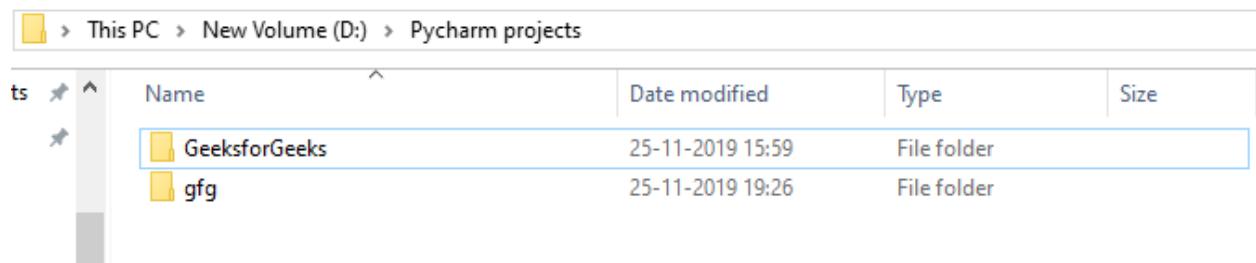
# Parent Directory
parent = "D:/Pycharm projects/"

# Path
path = os.path.join(parent, directory)

# Remove the Directory
# "Geeks"
os.rmdir(path)

```

## Output:



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 Functions

**1. os.name:** 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'.

```
import os

print(os.name)
```

**Output:**

```
posix
```

**Note:** It may give different output on different interpreters, such as 'posix' when you run the code here.

**2. os.error:** 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.

```
import os

try:
    # If the file does not exist,
    # then it would throw an IOError
    filename = 'GFG.txt'
    f = open(filename, 'rU')
    text = f.read()
    f.close()

# Control jumps directly to here if
# any of the above lines throws
IOError.
except IOError:

    # print(os.error) will <class
    'OSError'>
    print('Problem reading: ' +
    filename)

# In any case, the code then continues
with
# the line after the try/except
```

### Output:

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

**3. os.popen():** 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'.

## Syntax:

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

Parameters mode & bufsize are not necessary parameters, if not provided, default 'r' is taken for mode.

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

# popen() is similar to open()
file = open(fd, 'w')
file.write("Hello")
file.close()
file = open(fd, 'r')
text = file.read()
print(text)

# popen() provides a pipe/gateway and accesses the
file directly
file = os.popen(fd, 'w')
file.write("Hello")
# File not closed, shown in next function.
```

## Output:

```
Hello
```

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



**4. os.close():** Close file descriptor fd. A file opened using open(), can be closed by close() only. But files 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.

```
import os

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

### Output:

```
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)
```

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

**5. os.rename():** 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.

```
import os

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

### Output:

```
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'
```

**Understanding the Output:** 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.

**6. os.remove():** Using the Os module we can remove a file in our system using the remove() method. To remove a file we need to pass the name of the file as a parameter.

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

The OS module provides us a layer of abstraction between us and the operating system. When we are working with an os module, we 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**.

**7. os.path.exists():** This method will check whether a file exists or not by passing the name of the file as a parameter. The OS module has a sub-module named PATH by which we can perform many more functions.

```
import os  
#importing os module  
  
result = os.path.exists("file_name") #giving the name of  
the file as a parameter.  
  
print(result)
```

**Output**

```
False
```

As in the above code, if the file does not exist it will give output False. If the file exists it will give us output True.

**8. os.path.getsize():** In this method, 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.

```
import os #importing os module

size = os.path.getsize("filename")

print("Size of the file is", size, "
bytes.")
```

### Output:

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

## os.rename() method

**The OS module** in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality.

**os.rename()** method in Python is used to rename a file or directory.

This method renames a source file/ directory to specified destination file/directory.

**Syntax:** `os.rename(source, destination, *, src_dir_fd = None, dst_dir_fd = None)`

**Parameters:**

**source:** A path-like object representing the file system path. This is the source file path which is to be renamed.

**destination:** A path-like object representing the file system path.

**src\_dir\_fd** (optional): A file descriptor referring to a directory.

**dst\_dir\_fd** (optional): A file descriptor referring to a directory.

**Return Type:** This method does not return any value.

**Code #1:** Use of **os.rename()** method

```
# Python program to explain os.rename() method
```

```
# importing os module
```

```
import os
```

```
# Source file path
```

```
source = 'GeeksforGeeks/file.txt'
```

```
# destination file path
```

```
dest = 'GeekforGeeks/newfile.txt'
```

```
# Now rename the source path

# to destination path

# using os.rename() method

os.rename(source, dest)

print("Source path renamed to destination path
successfully.")
```

**Output:**

```
Source path renamed to destination path successfully.
```

**Code #2:** Handling possible errors

```
# Python program to explain os.rename() method
```

```
# importing os module
```

```
import os
```

```
# Source file path
```

```
source = './GeeksforGeeks/file.txt'
```

```
# destination file path
```

```
dest = './GeeksforGeeks/dir'
```



```
# try renaming the source path
```

```
# to destination path
```

```
# using os.rename() method
```

```
try :
```

```
    os.rename(source, dest)
```

```
    print("Source path renamed to destination path  
successfully.")
```

```
# If Source is a file
```

```
# but destination is a directory
```

```
except IsADirectoryError:
```

```
    print("Source is a file but destination is a  
directory.")
```

```
# If source is a directory
```

```
# but destination is a file
```

```
except NotADirectoryError:
```

```
    print("Source is a directory but destination is a  
file.")
```

```
# For permission related errors
```

```
except PermissionError:

    print("Operation not permitted.")


# For other errors

except OSError as error:

    print(error)
```

### Output:

Source is a file but destination is a directory.

-----X-----X-----X

### Mass renaming -

```
for file in os.listdir(location):
    os.rename(file, new_name)
```

