OS Module in Python with Examples

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 **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 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.

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

import os

cwd = os.getcwd()

print("Current working directory:", cwd)
```

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.

```
# Python program to change the

# current working directory

import os

def current_path():

        print("Current working directory before")

        print(os.getcwd())

        print()

current_path()

os.chdir('../')

current_path()
```

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()

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.

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

# importing os module

import os

directory = "DishaComputer"

parent_dir = "D:/Pycharm projects/"

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

os.mkdir(path)

print("Directory '% s' created" % directory)

directory = "Disha"

parent_dir = "D:/Pycharm projects"

mode = 0o666

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

os.mkdir(path, mode)

print("Directory '% s' created" % directory)
```

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.

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

import os

directory = "Sakshi"

parent_dir = "D:/Pycharm projects/DishaComputer/Authors"

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

```
os.makedirs(path)

print("Directory '% s' created" % directory)

directory = "c"

parent_dir = "D:/Pycharm projects/DishaComputer/a/b"

mode = 0o666

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

os.makedirs(path, mode)

print("Directory '% s' created" % directory)
```

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.

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

import os

path = "/"

dir_list = os.listdir(path)

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

print(dir_list)
```

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()

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.

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

import os

file = 'file1.txt'

location = "D:/Pycharm projects/DishaComputer/Authors/Sakshi/"
```

```
path = os.path.join(location, file)

os.remove(path)
```

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.

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

import os

directory = "Disha"

parent = "D:/Pycharm projects/"

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

os.rmdir(path)
```

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)
```

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:

        filename = 'ABC.txt'

        f = open(filename, 'rU')
```

```
        text = f.read()

        f.close()
except IOError:

        print('Problem reading: ' + filename)
```

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 = "ABC.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")
```

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 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.

```
        import os

        fd = "GFG.txt"

        file = open(fd, 'r')

        text = file.read()

        print(text)

        os.close(file)
```

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')
```

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")
```

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**.

7. os.path.exists(): 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.

```
import os

#importing os module

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

print(result)
```

As in the above code, 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.")
```