39

Day 39: OS Library

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-Module Functions

Here we will discuss some important functions of the Python os module:

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

1. Handling **Current Working Directory**

1.1 Getting the current working directory

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

1.2 Changing the current working directory

You can change Current Working Directory using os.chdir(path).

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

2. Creating a Directory

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

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

2.1 Using os.mkdir()

os.mkdir() method 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.

```
import os

directory = "Rohit"
parent_dir = r"C:\Users\gango\Documents\VS_Code_Files\python_notes\pytho
n_automation"
# if "r" not given, Syntax error will come
path = os.path.join(parent_dir, directory)

os.mkdir(path)
print("Directory '% s' created" % directory)



directory = "Prashant"
parent_dir = r"C:\Users\gango\Documents\VS_Code_Files\python_notes\pytho
n_automation"
mode = 0o666 # read & write permission granted here
path = os.path.join(parent_dir, directory)
```

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

Modes:

Mode	Permission Description	Meaning for Owner/Group/Others	Typical Use Case
0o777	read, write, execute for all	rwx for owner, group, others	Full access for everyone
0o755	owner rwx, others rx	rwx for owner, r-x for group and others	Common for public directories
0o700	owner rwx only	rwx for owner, no access for group/others	Private directory for owner only
0o775	owner rwx, group rwx, others rx	rwx for owner and group, r-x for others	Shared directory among owner and group
0o666	read, write for all (no execute)	rw- for owner, group, others	Not typical for directories; no execute means no access to directory contents
0o644	read, write for owner; read for others	rw- for owner, r-- for group and others	Typical for files, not directories
0o600	read, write for owner only	rw- for owner, no access for group or others	Private file permission, not directory

2.2 **Using os.makedirs()**

os.makedirs() method 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.

```
import os

directory = "Rohit"
parent_dir = r"C:\Users\gango\Documents\VS_Code_Files\python_notes\python_automation\Authors"
```

```
path = os.path.join(parent_dir, directory)
os.makedirs(path)
# If used makedir() Attribute error willc come
print("Directory '% s' created" % directory)



directory = "c"
parent_dir = r"C:\Users\gango\Documents\VS_Code_Files\python_notes\pytho
n_automation\a\b"
mode = 0o666
path = os.path.join(parent_dir, directory)
os.makedirs(path, mode)
print("Directory '% s' created" % directory)
```

3. **Listing out Files and Directories with Python**

os.listdir() method 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.

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

4. Deleting Directory or Files using Python

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

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

4.1 Using os.remove() Method

os.remove() method 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.

```
import os

file = 'new_2.py'
location = r"C:\Users\gango\Documents\VS_Code_Files\python_notes\python_automation"
path = os.path.join(location, file)
os.remove(path)
```

4.2 Using os.rmdir()

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

```
import os

directory = "Rohit"
parent = r"C:\Users\gango\Documents\VS_Code_Files\python_notes\python_automation\Authors"
path = os.path.join(parent, directory)
os.rmdir(path)
```

5. File Permissions and Metadata

Apart from basic file and directory operations, Python's os module provides access to lower-level file system metadata and permission handling- useful for scripting, administration and system-level tasks. Three important methods in this category are:

- **os.chmod()**: Change file or directory permissions
- **os.chown()**: Change file owner and group (Unix only)
- **os.stat()**: Fetch metadata like file size, modification time, permissions, etc.

5.1 os.chmod()

os.chmod() method is used to change the permissions (read, write, execute) of a file or directory. Permissions must be passed in **octal format (e.g., 0o600).**

```
import os

# Set read-write permissions for owner (0o600), Group & Others - No permiss
ion
os.chmod(r"C:\Users\gango\Documents\VS_Code_Files\python_notes\python_
automation\main.py", 0o600)
```

5.2 os.chown()

os.chown() method allows you to change the owner and group ID of a file. This is typically used in system scripts and requires appropriate permissions. It is used in Unix/Linux systems only.

```
import os

# Change owner and group to user ID 1000 and group ID 1000
os.chown("example.txt", 1000, 1000)
```

Explanation:

- Both **uid** and **gid** must be integers.
- Requires root access to change file ownership.
- On unsupported platforms (e.g., Windows), it will raise **AttributeError.**

5.3 os.stat()

os.stat() method is used to retrieve metadata about a file such as its size, permissions and timestamps.

```
import os

path = r"C:\Users\gango\Documents\VS_Code_Files\python_notes\python_aut
```

```
omation\new_1.md"
stats = os.stat(path)

print(os.path.basename(path))
print("Size:", stats.st_size, "bytes") # return size
print("Last modified:", stats.st_mtime) # reurn modified timestamp
print("Permissions:", oct(stats.st_mode)[-3:]) # returns mode
```

Commonly Used Functions

1. **Using os.name function**

```
import os
print(os.name)
# This function gives the name of the operating system dependent modul
e imported.
# The following names have currently been registered: 'posix', 'nt', 'os2',
'ce', 'java' and 'riscos'.
```

Value	Explanation
posix	Refers to POSIX-compliant operating systems such as Linux, macOS, Unix, and other Unix-like systems. Theposixmodule provides standard POSIX system calls and environment functions.
nt	Refers to Windows operating systems (including both Windows NT and modern Windows versions). Thentmodule provides Windows-specific system calls and environment functions.
os2	Refers to IBM OS/2 operating system, a legacy Microsoft/IBM OS that is mostly obsolete now. Python includes support for it but it is rarely used today.
ce	Refers to Windows CE (Compact Edition), a version of Windows for embedded systems and mobile devices. Python's CE support is limited to this environment.
java	Refers to Jython, a Python implementation running on the Java platform. The underlying OS interfaces are Java-based.

Value	Explanation
riscos	Refers to RISC OS, an older operating system for ARM architecture computers, mostly legacy.

2. **Using os.error Function**

   os.error is an alias for built-in OSError exception**.**

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