OS Unit in Python

Gaurav Kr. suman

MIT13

OS Unit 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.

Following are some functions in OS module:

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.getcwd(): Function os.getcwd(), returns the Current Working Directory(CWD) of the file used to execute the code, can vary from system to system.

```
import os
print(os.getcwd())
# To print absolute path on your system
# os.path.abspath('.')

# To print files and directories in the current directory
# on your system
# os.listdir('.')
```

Output:

```
C:\Users\GKS\Desktop\ModuleOS
```

Note: In case of GKG interpreter, directory used is \root.

3. 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 = 'GKS.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: GKS.txt
```

File Object Manipulation

4. os.popen(): This method opens a pipe to or from command. The return value can be read or written depending on whether 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.

5. 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)
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\GFG\Desktop\gauravOSFile.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 non-existent of file or permission privilege.

6. 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 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\gauravOSFile.py", line 3, in
    os.rename(fd,'New.txt')
FileNotFoundError: [WinError 2] The system cannot find the
file specified: 'GFG.txt' -> 'New.txt'
```