

In [ ]:

## Python - OS Module

It is possible to automatically perform many operating system tasks. The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

You first need to import the os module to interact with the underlying operating system. So, import it using the import os statement before using its functions.

## 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. Getting the Current working directory

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

In [6]: pwd

Out[6]: 'D:\\2024\_25 PYTHON1\\T3\\T3-PYTHON1'

```
In [8]: #The getcwd() function confirms returns the current working  
import os  
os.getcwd()
```

Out[8]: 'D:\\2024\_25 PYTHON1\\T3\\T3-PYTHON1'

## 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.

In order to set the current directory to the parent directory use `".."` as the argument in the `chdir()` function.

```
In [14]: import os
os.chdir("D:\\202324python\\T2-PYTHON1") # changing current
os.getcwd()
```

```
Out[14]: 'D:\\202324python\\T2-PYTHON1'
```

## Creating a Directory

`os.mkdir()`

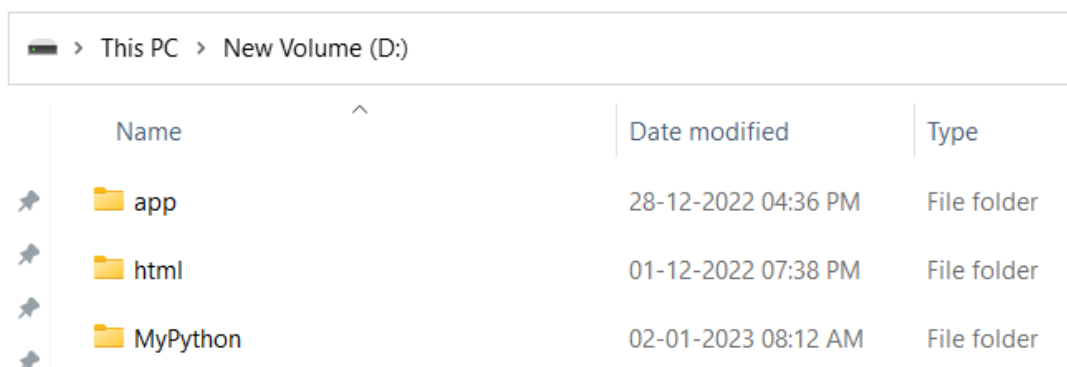
`os.makedirs()`

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.

By default, if you don't specify the whole path in the `mkdir()` function, it will create the specified directory in the current working directory or drive.

```
In [23]: import os
os.mkdir(r"d:\MyPython")
```



The screenshot shows a Windows File Explorer window with the address bar set to 'This PC > New Volume (D:)'. The main area displays a table of files and folders. There are three folders listed: 'app', 'html', and 'MyPython'. Each folder has a star icon to its left. The 'app' folder was created on 28-12-2022 at 04:36 PM. The 'html' folder was created on 01-12-2022 at 07:38 PM. The 'MyPython' folder was created on 02-01-2023 at 08:12 AM. All are listed as 'File folder' type.

Name	Date modified	Type
app	28-12-2022 04:36 PM	File folder
html	01-12-2022 07:38 PM	File folder
MyPython	02-01-2023 08:12 AM	File folder

```
In [27]: import os
os.chdir(r"d:\MyPython") # changing current workign director
os.getcwd()
```

Out[27]: 'd:\\MyPython'

```
In [29]: import os
os.mkdir("vishal")
```

📁 > This PC > New Volume (D:) > MyPython >				
	Name	Date modified	Type	Size
📁	vishal	02-01-2023 08:17 AM	File folder	

## 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.

```
In [31]: import os
os.listdir("c:")
```

```
Out[31]: ['$Recycle.Bin',
'$SysReset',
'$WinREAgent',
'Config.Msi',
'Documents and Settings',
'DumpStack.log',
'DumpStack.log.tmp',
'hiberfil.sys',
'hp',
'M1130MFP_M1210MFP_Full_Solution',
'msdia80.dll',
'OneDriveTemp',
'pagefile.sys',
'PerfLogs',
'Program Files',
'Program Files (x86)',
'ProgramData',
'Recovery',
'swapfile.sys',
'SWSetup',
'System Volume Information',
'System.sav',
'Users',
'Windows',
'Windows.old']
```

## 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()`

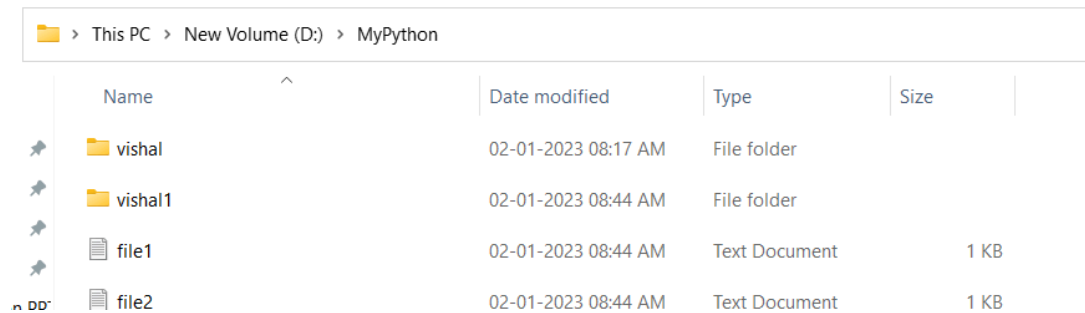
`os.remove()` method in Python is used to remove or delete a file path.

This method can not remove or delete a directory. If the specified path is a directory then `OSError` will be raised by the method.

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

```
In [37]: import os
os.chdir(r"D:\MyPython")
os.mkdir("vishal1")
f=open("file1.txt","w")
f.write("hello")
f.close()
f=open("file2.txt","w")
f.write("hello1")
f.close()
```



Name	Date modified	Type	Size
vishal	02-01-2023 08:17 AM	File folder	
vishal1	02-01-2023 08:44 AM	File folder	
file1	02-01-2023 08:44 AM	Text Document	1 KB
file2	02-01-2023 08:44 AM	Text Document	1 KB

```
In [39]: import os
os.chdir(r"D:\MyPython")
print(os.getcwd())
os.remove(r"D:\MyPython\file2.txt")
```

D:\MyPython

```
In [ ]: import os

# File name
file = 'file2.txt'

# File location
location = "D:\MyPython"

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

# Remove the file
# 'file2.txt'
os.remove(path)
```

This PC > New Volume (D:) > MyPython				
	Name	Date modified	Type	Size
	vishal	02-01-2023 08:17 AM	File folder	
	vishal1	02-01-2023 08:44 AM	File folder	
	file1	02-01-2023 08:44 AM	Text Document	1 KB

In [38]: **import** os

```
# File name
file = 'file2.txt'

# File location
location = "D:\MyPython"

# Path
path = os.path.join(location, file)
print(path)
# Remove the file
# 'file2.txt'
os.remove(path)
```

D:\MyPython\file2.txt

```
-----
-----
FileNotFoundError                                Traceback (most
recent call last)
~\AppData\Local\Temp\ipykernel_22760\724587826.py in <modu
le>
      12 # Remove the file
      13 # 'file2.txt'
----> 14 os.remove(path)
```

**FileNotFoundError:** [WinError 2] The system cannot find the file specified: 'D:\\MyPython\\file2.txt'

```
In [45]: import os

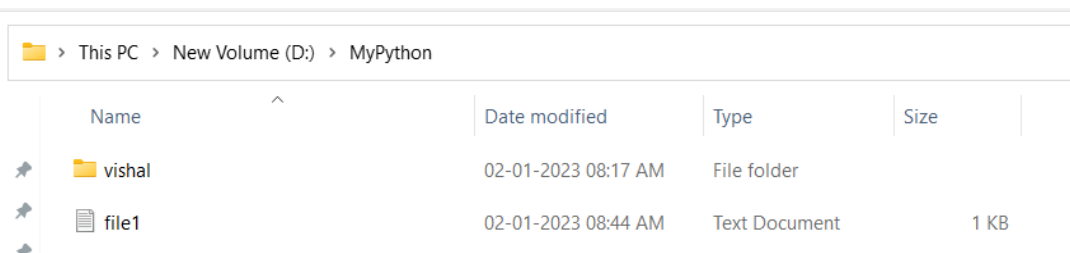
# Directory name
directory = "vishal1"

# Parent Directory
parent = "D:\MyPython"

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

# Remove the Directory

os.rmdir(path)
```



This PC > New Volume (D:) > MyPython				
	Name	Date modified	Type	Size
	vishal	02-01-2023 08:17 AM	File folder	
	file1	02-01-2023 08:44 AM	Text Document	1 KB

## 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'.

```
In [46]: import os

print(os.name)
```

nt

## 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.

In [51]: `import os`  
`os.getcwd()`  
`os.chdir("d:\\MyPython")`  
`os.rename("file1.txt", 'New.txt')`

📁 > This PC > New Volume (D:) > MyPython

	Name	Date modified	Type	Size
📁	vishal	02-01-2023 08:17 AM	File folder	
📄	New	02-01-2023 08:44 AM	Text Document	1 KB

In [52]: `import os`  
`size = os.path.getsize("New.txt")`  
`print("Size of the file is", size, " bytes.")`

Size of the file is 5 bytes.

In [53]: `import sys`  
`print(sys.maxsize)`  
`print(sys.path)`  
`print(sys.version)`

9223372036854775807

['C:\\Users\\VISHAL\\Documents', 'C:\\Users\\VISHAL\\anaconda3\\python39.zip', 'C:\\Users\\VISHAL\\anaconda3\\DLLs', 'C:\\Users\\VISHAL\\anaconda3\\lib', 'C:\\Users\\VISHAL\\anaconda3', '', 'C:\\Users\\VISHAL\\anaconda3\\lib\\site-packages', 'C:\\Users\\VISHAL\\anaconda3\\lib\\site-packages\\win32', 'C:\\Users\\VISHAL\\anaconda3\\lib\\site-packages\\win32\\lib', 'C:\\Users\\VISHAL\\anaconda3\\lib\\site-packages\\Pythonwin', 'C:\\Users\\VISHAL\\anaconda3\\lib\\site-packages\\IPython\\extensions', 'C:\\Users\\VISHAL\\.ipython']

3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)]

In [ ]:

AVHARYA  
VISHAL



# Module\_VHA

January 8, 2024

## 1 Module

A Python module is the highest level of program organization, packing code, and data for reuse and easy interoperability at the functional level and for providing self-contained namespaces to prevent variables from clashing across different programs.

Modules are files containing Python code that define functions, classes, and variables. Runnable code can also be included in a module. These pieces should be shared, so Python allows a module to “bring in” and use attributes from other modules to take advantage of work that has been done, maximizing code reusability. This process of associating the attributes from other modules with your module is called importing.

## 2 INTRODUCTION

- A module is a file containing Python definitions and statements.
- A module is a file containing group of variables, methods, function and classes etc.
- They are executed only the first time the module name is encountered in an import statement.
- The file name is the module name with the suffix .py appended.
- Ex:- mymodule.py ##### Type of Modules:-
- User-defined Modules Ex:- array, math, sys
- Built-in Modules

## 3 When and Why use Module

Assume that you are building a very large project, it will be very difficult to manage all logic within one single file so If you want to separate your similar logic to a separate file, you can use module.

It will not only separate your logics but also help you to debug your code easily as you know which logic is defined in which module. When a module is developed, it can be reused in any program that needs that module.

**Database »»»»> db.py**

**Calculation »»»»> cal.py**

**Searching »»»»> search.py**

## 4 We will cover the following topics in this chapter:

The import statement

The from..import statement

The from..import \* statement

Executing modules as scripts

The module search path

The dir() function

## 5 How to use Module

## 6 import statement is used to import modules.

Syntax:-

```
import module_name
```

```
import module_name as alias_name
```

```
from module_name import
```

```
from module_name import f_name as alias_f_name
```

```
from module_name import *
```

- from module\_name import class\_name1, class\_name2,....., class\_nameN
- from module\_name import var\_name1, var\_name2,....., var\_nameN
- from module\_name import function\_name1, function\_name2,....., function\_nameN
- from module\_name import var\_name, f\_name, class\_name, method\_name.....,

## 7 The import statement

Any Python source file can be used as a module by executing an import statement in another Python source file.

Syntax:

```
import module1[, module2[,... moduleN]
```

When the interpreter finds an import statement, it imports the module if it is present in the search path. The search path refers to a list of directories that the interpreter searches before importing a module.

## 8 import module\_name

This does not enter the names of the functions defined in module directly in the current symbol table; it only enters the module name there.

- When 2 modules having same function name then This import module is good approach to use.

Ex:- import cal

How to access Methods, Functions, Variable and Classes ?

Using the module name you can access the functions.

Syntax:- module\_name.function\_name()

```
[6]: %%writefile cal.py
a = 50
def name():
    print("From Module cal")
def add(a,b):
    return a+b
def sub(a,b):
    return a-b
```

Overwriting cal.py

```
[7]: import cal # Importing Cal Module

print("cal Module's variable:", cal.a) # Accessing Cal Module's Variable

cal.name() #Accessing Cal Module's Function

a = cal.add(10,20) # Accessing Cal Module's Function
print(a)

b = cal.sub(20, 10) # Accessing Cal Module's Function
print(b)
```

```
cal Module's variable: 50
From Module cal
30
10
```

```
[8]: import cal # Importing Cal Module

print("cal Module's variable:", cal.a) # Accessing Cal Module's Variable

cal.name() # Accessing Cal Module's Function

add = cal.add # Accessing and Assigning Module's Function to Variable
a = add(10, 20) # Accessing Cal Module's Function
print(a)

b = cal.sub(20, 10) # Accessing Cal Module's Function
```

```
print(b)
```

```
cal Module's variable: 50
From Module cal
30
10
```

## 9 import module\_name as alias\_name

This does not enter the names of the functions defined in module directly in the current symbol table; it only enters the module name there. If the module name is followed by as, then the name following as is bound directly to the imported module.

Ex:- import cal as c

How to access Methods, Functions, Variable and Classes ?

Using the alias\_name you can access the functions

```
[9]: import cal as c # Importing Cal Module

print("cal Module's variable:", c.a) # Accessing Cal Module's Variable

c.name() # Accessing Cal Module's Function

a = c.add(10,20) # Accessing Cal Module's Function
print(a)

b = c.sub(20, 10) # Accessing Cal Module's Function
print(b)
```

```
cal Module's variable: 50
From Module cal
30
10
```

## 10 The from...import statement

You can import specific attributes from a module into the current namespace using Python's from statement.

Syntax:

```
from modname import name1[, name2[, ... nameN]]
```

## 11 from module\_name import function\_name

There is a variant of the import statement that imports names from a module directly into the importing module's symbol table.

Ex:- from cal import add, sub

How to access Methods, Functions, Variable and Classes ?

You can access the functions directly by it's name.

## 12 from module\_name import f\_name as a\_name

Ex:- from cal import add as s

How to access Methods, Functions, Variable and Classes ? You can access the functions directly by it's alias name.

Ex:- s(10, 20)

```
[10]: from cal import a, name, add as s, sub # Importing Cal Module

print("cal Module's variable:", a) # Accessing Cal Module's Variable

name() # Accessing Cal Module's Function

add = s(10,20) # Accessing Cal Module's Function
print(add)

b = sub(20, 10) # Accessing Cal Module's Function
print(b)
```

cal Module's variable: 50

From Module cal

30

10

## 13 The from...import \* statement

All the names in a module can be imported into the current namespace using the following import statement.

Syntax:

from modname import \*

A module can be imported into the current namespace using this statement, but the information should be used sparingly.

## 14 from module\_name import \*

This imports all names except those beginning with an underscore (\_).

Ex:- from cal import \*

How to access Methods, Functions, Variable and Classes ?

You can access the functions directly by it's name.

```
[11]: from cal import * # Importing Cal Module

print("cal Module's variable:", a) # Accessing Cal Module's Variable

name() # Accessing Cal Module's Function

a = add(10,20) # Accessing Cal Module's Function
print(a)

b = sub(20, 10) # Accessing Cal Module's Function
print(b)
```

```
cal Module's variable: 50
From Module cal
30
10
```

```
[12]: %%writefile first.py
a=50
def name():
    print("in")
```

Writing first.py

```
[14]: %%writefile second.py
a=40
def sname():
    print("out")
```

Writing second.py

```
[15]: #Importing Two Modules and accessing their members
# Both Modules has same name variable and function
import first # Importing first Module
import second # Importing second Module

print(first.a) # Accessing first Module's Variable
first.name() # Accessing first Module's Function

print(second.a) # Accessing Second Module's Variable
second.sname() # Accessing Second Module's Function
```

```
50
in
40
out
```

```
[17]: %%writefile first1.py
a=50
def name():
    print("in")
```

Writing first1.py

```
[18]: %%writefile second1.py
a=40
def sname():
    print("out")
```

Writing second1.py

```
[20]: import first1 as f # Importing first Module
import second1 as s # Importing second Module

print(f.a) # Accessing first Module's Variable
f.name() # Accessing first Module's Function

print(s.a) # Accessing Second Module's Variable
s.sname() # Accessing Second Module's Function
```

50  
in  
40  
out

```
[21]: from first1 import name, a # Importing first Module

print(a) # Accessing first Module's Variable
name() # Accessing first Module's Function

from second1 import sname, a # Importing second Module

print(a) # Accessing Second Module's Variable
sname() # Accessing Second Module's Function
```

50  
in  
40  
out

```
[26]: %%writefile firstclass.py
class Myclass:
    def name(self):
        print("Name Method from first Module")

class Myschool:
```

```
def show(self):
    print("Show Method from first Module")
```

Overwriting firstclass.py

```
[27]: %%writefile secondclass.py
class Mycollege:
    def disp(self):
        print("Disp Method from Second Module")
```

Overwriting secondclass.py

```
[28]: import firstclass # Importing first Module

c = firstclass.Myclass() # Creating Myclass Object
c.name()

s = firstclass.Myschool() # Creating Myschool Object
s.show()
```

Name Method from first Module  
Show Method from first Module

```
[29]: import firstclass # Importing first Module
import secondclass # Importing second Module

c = firstclass.Myclass() # Creating Myclass Object - first Module
c.name()

s = firstclass.Myschool() # Creating Myschool Object - first Module
s.show()

cl = secondclass.Mycollege() # Creating Myschool Object - second Module
cl.disp()
```

Name Method from first Module  
Show Method from first Module  
Disp Method from Second Module

```
[31]: from firstclass import Myclass, Myschool # Importing first Module
c = Myclass() # Creating Myclass Object - first Module
c.name()

s = Myschool() # Creating Myschool Object - first Module
s.show()
from second1 import sname, a # Importing second Module

print(a) # Accessing Second Module's Variable
name() # Accessing Second Module's Function
```



Name Method from first Module  
Show Method from first Module  
40  
out

## 15 The dir() function

Directory() returns a sorted list of strings containing the names defined in a module. The list consists the names of all the variables, and functions defined in a module.

```
[2]: print(dir())
```

```
['In', 'Myclass', 'Myschool', 'Out', '_', '__', '___', '__builtin__',  
 '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__',  
 '_dh', '_i', '_i1', '_i2', '_ih', '_ii', '_iii', '_oh', 'a', 'c', 'exit',  
 'get_ipython', 'name', 'quit', 's']
```

```
[3]: print(dir(Myschool))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',  
 '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',  
 '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__',  
 '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',  
 '__str__', '__subclasshook__', '__weakref__', 'show']
```

## 16 What is Module in Python?

- The module is a simple Python file that contains collections of functions and global variables and with having a .py extension file. It is an executable file and to organize all the modules we have the concept called Package in Python.

Examples of modules: - Datetime - Regex - Random etc.

Example: Save the code in a file called demo\_module.py

## 17 What is Package in Python?

- The package is a simple directory having collections of modules. This directory contains Python modules and also having init.py file by which the interpreter interprets it as a Package. The package is simply a namespace. The package also contains sub-packages inside it.

Examples of Packages:

- Numpy
- Pandas

## 18 What is Library in Python¶

- The library is having a collection of related functionality of codes that allows you to perform many tasks without writing your code. It is a reusable chunk of code that we can use by importing it into our program, we can just use it by importing that library and calling the method of that library with a period(.). However, it is often assumed that while a package is a collection of modules, a library is a collection of packages.

Examples of Libraries:

- Matplotlib
- Pytorch
- Pygame
- Seaborn etc.

In the sector of software development, terminology like “module,” “package,” “library,” and “framework” is frequently used to describe various schemes for classifying and reusing code. However, the precise meanings of these phrases might change depending on the context and programming language used. Here is the definition with a real-life example:

- Module = Your fingers
- Package= Your hands
- Library = Building your home
- Framework = Buying a home

1:

1: