

Modules in Python

- We know that Functions concept makes us understand How to perform operations and we can re-use within the same program but not able to re-use the functions across the programs.
- To reuse the functions and global variables across the programs, we must use the concept of MODULES.

• Definition of Module:

-
- A Module is a collection of variables (global variables), Functions and Classes.

• Types of Modules:

-
- In Python Programming, we have two types of Modules. They are
 - 1) Pre-defined (or) Built-in Modules
 - 2) Programmer or user or custom-defined modules.

1) Pre-defined (or) Built-in Modules:

-
- These modules are developed by Python Language Developers and they are available in Python Software (APIs) and they are used python programmers for dealing with Universal Requirements.
 - Examples: - math, cmath, functools, sys, calendar, os, e, threading, pickle, random.....etc
 - Out of many pre-defined modules, in python programming one implicit pre-defined module imported to every python program called "built-ins".

2) Programmer or user or custom-defined modules:

-
- These modules are developed by Python Programmers and they are available in Python Project and they are used by other python programmers who are in project development to deal with common requirements.
 - Examples: - aop, mathsinfo, icicietc

Number of approaches to re-use Modules

- To re-use the features (Variable Names, Function Names and Class Names) of module, we have 2 approaches. They are

- 1) By using `import` statement
- 2) By using `from.... import` statement.

1) By using `import` statement:

- 'import' is a keyword.
- The purpose of import statement is that "To refer or access the variable names, function names and class names in current program".
- we can use import statement in 4 ways.

Syntax-1: - `import module name`

- This syntax imports single module.
- Example: - `import math`

Syntax-2: - `import module name1, module ame2....Module name-n`

- This syntax imports multiple modules.
- Example: - `import math, random, os.`

Syntax-3: - `import module name as alias name`

- This syntax imports single module and aliased with another name.
- `import math as m`

Syntax-4: - `import module name1 as alias name, module name2 as alias name.....module name-n as alias name`

- This syntax imports multiple modules and aliased with another names
- Example: - `import math as m, random as rm.`
- Hence after importing all the variable names, Function names and class names by using "import statement" , we must access variable names, Function names and class names w.r.t Module Names or alias names.

```

Module Name.Variable Name
Module Name.Function Name
Module Name.Class Name
(OR)
Alias Name.Variable Name
Alias Name.Function Name
Alias Name.Class Name

```

2) By using from.... import statement.

- Here "from" "import" are the key words.
- The purpose of from.... import statement is that " To refer or access the variable names, function names and class names in current program directly without writing module name".
- we can use from.... import statement in 3 ways.

Syntax-1: - from module name import Variable Names, Function Names, Class Names

- This syntax imports the Variable Names, Function Names, Class Names of a module.
- Example: - from calendar import month

Syntax-2: from module name import Variable Names as alias name, Function Names as alias name, Class Names as alias names.

- This syntax imports the Variable Names, Function Names, Class Names of a module with alias Names.
- Example: - from calendar import month as m

Syntax-3: - from module name import *

- This syntax imports ALL Variable Names, Function Names, Class Names of a module.
- Example: - from calendar import *
- This syntax is not recommended to use bcoz it imports required Features of Module and also import un-interested features also imported and leads more main memory space.
- Hence after importing all the variable names, Function names and class names by using "from import statement", we must access variable names, Function names and class names Directly without using Module Names or alias names.

```

Variable Name
Function Name
Class Name

```

- Hence with "import statement" we can give alias name for module names only but not for Variables Names, Function Names and Class Names. Whereas with "from ... import statement

" we can give alias names for Variables Names, Function Names and Class Names but not for Module Name.

1) Pre-defined (or) Built-in Modules :

Working with math module.

- Python provides Built-in module math.
- This module defines several functions which can be used for mathematical operation/tasks.

```
In [6]: # importing math module.  
import math
```

```
In [8]: # We can find help for any module by using help() function.  
help(math) # This will display the help centre.
```

```
In [10]: # Find the square root of "64" by using math module.  
  
# importing math module.  
import math  
  
# Using sqrt function from math module to find the square root.  
# Accessing function by (modulename.function()).  
math.sqrt(64)
```

Out[10]: 8.0

```
In [20]: # Find the square root and exponential of the specified values using math module.  
  
# importing sqrt and exp from math module.  
from math import sqrt,exp  
  
# Using sqrt function from math module to find the square root.  
# Using exp function from math module to find the exponential.  
# Accessing function by (function()).  
print(sqrt(81))  
print(exp(1))
```

9.0
2.718281828459045

```
In [17]: # Find the square root of "48" by using math module.

# importing math module with alias name.
import math as m

# Using sqrt function from math module to find the square root.
# Accessing function by (aliasname.function()).
m.sqrt(48)
```

Out[17]: 6.928203230275509

```
In [16]: # Find the square root and exponential of the specified values using math module.

# importing sqrt and exp from math module.
from math import *

# Using sqrt function from math module to find the square root.
# Using exp function from math module to find the exponential.
# Accessing function by (function()).
print(sqrt(49))
print(exp(5))
```

7.0

148.4131591025766

Working with random module.

- This module defines several functions to generate random numbers.
- We can use these functions while developing games in cryptography and to generate random numbers on fly for authentication.

random() :

- Returns a random float number between 0 and 1.

```
In [22]: # Find the random number between 0 to 1.

# importing random.
import random

random.random() # Returns a random float number between 0 and 1.
```

Out[22]: 0.3663311454817195

randint() :

- The randint() method returns an integer number selected element from the specified range.

```
In [23]: # Find the random integer number between 10 to 30.

# importing random.
import random

'''Using randint() method returns an integer number selected element from
the specified range.'''
random.randint(10,30)
```

Out[23]: 13

randrange() :

- The randrange() method returns a randomly selected element from the specified range.
- The main difference between randint and randrange is, randrange allows step values but randint not allowed any step value.

```
In [24]: # Find the random integer number between 100 to 130 by step value 5.

# importing random.
import random

'''Using randrange() method returns a randomly selected element
from the specified range with step value.'''
random.randrange(100,130,5)
```

Out[24]: 125

choice() :

- The choice() method returns a randomly selected element from the specified sequence.
- The sequence can be a string, a range, a list, a tuple or any other kind of sequence.

```
In [26]: # Find the random name from the given list.

# User given list.
lst=["Hari","Raju","Kiran"]

'''Using choice() method returns a randomly selected element
from the specified sequence..'''
random.choice(lst)
```

Out[26]: 'Raju'

Development of Programmer-Defined Module :

- To develop Programmer-Defined Modules, we must use the following steps.

Step-1: Define Variables (Global variables)

Step-2: Define Functions

Step-3: Define Classes

- After developing step-1, step-2 and step-3, we must save on some file name with an extension .py (FileName.py) and it is treated as module name.
- When a file name treated as a module name, internally Python execution environment creates a folder automatically on the name of **pycache** and it contains module name on the name of "filename.cpython-310.pyc".

Examples:

```

-----
pycache                                <-----Folder Name
-----
aop.cpython-310.pyc                  <-----Module Name
mathsinfo.cpython-310.pyc<-----Module Name
icici.cpython-310.pyc<-----Module Name
-----

```

- In order to work with user defined modules, that particular module should be available in your working directory then only it will work, otherwise we will get "ModuleNotFoundError".

What is working directory ?

- Wherever you are present working in the folder / directory (Jupyter notebbok) is available in the the system, that particular folder is known as working directory.

To check the present working directory, there are two options.

- 1) By importing "os" module and print "os.getcwd()"
- 2) By entering the "pwd".

Option - 1

```

In [2]: # To check the present working directory by option - 1.
        # import the "os" module.
        import os

        # Write the "os.getcwd()" then hit shift+enter.
        os.getcwd()

```

Out[2]: 'C:\\Users\\Administrator\\PYTHON'

Option - 2

```

In [7]: # To check the present working directory by option - 2.
        # Write the "pwd" then hit shift+enter.

```

```
In [8]: pwd
```

```
Out[8]: 'C:\\Users\\Administrator\\PYTHON'
```

Changing the present working directory.

- To change the working directory.
- import the "os" module.

```
import os
```

-Write the "os.chdir(path)" then hit shift+enter. os.chdir(path)

- Then it will change the working directory.

To work with python module in jupyter notebook there are two options.

1) My filename.py (module file) should be in the same folder / directory.

- if .py file is not there in the same directory, then bring your filename.py (module file) in your working directory by copy / paste.

2) Change your directory where that .py file is available.

- To change the working directory.
- import the "os" module.

```
import os
```

-Write the "os.chdir(path)" then hit shift+enter.

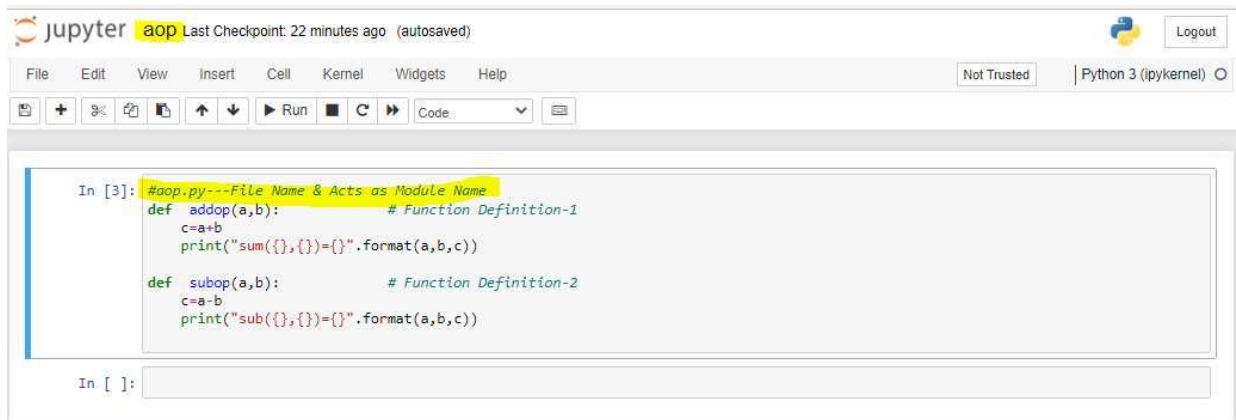
```
os.chdir(path)
```

- Then it will change the working directory.

Examples : -

In this example I am going with option - 1.

- Here I am creating / defining a functions (addop, subop) with file name (aop) with in PYTHON folder.



```

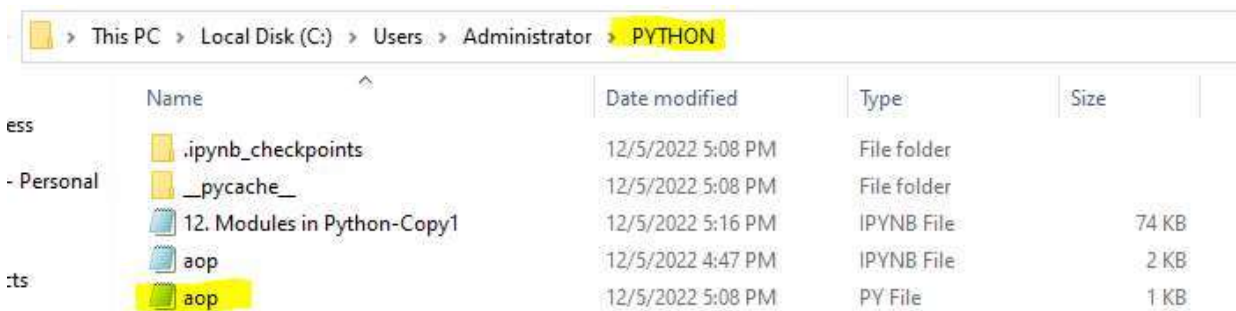
In [3]: #aop.py---File Name & Acts as Module Name
def addop(a,b):      # Function Definition-1
    c=a+b
    print("sum({},{})={}".format(a,b,c))

def subop(a,b):      # Function Definition-2
    c=a-b
    print("sub({},{})={}".format(a,b,c))

In [ ]:

```

- Then downloading this file with an extension .py (FileName.py) and it is treated as module name.
- After that, I copied this file from the downloads and pasted in the PYTHON folder.



Name	Date modified	Type	Size
.ipynb_checkpoints	12/5/2022 5:08 PM	File folder	
__pycache__	12/5/2022 5:08 PM	File folder	
12. Modules in Python-Copy1	12/5/2022 5:16 PM	IPYNB File	74 KB
aop	12/5/2022 4:47 PM	IPYNB File	2 KB
aop.py	12/5/2022 5:08 PM	PY File	1 KB

- I am creating another file as "Modules in Python" in the same "PYTHON" folder and importing the "aop" module in this file and calling the "aop" functions from this file, for executing the "aop" program.

```

In [6]: # import the "aop" file (module)
import aop      # import module name

# calling the "aop" functions in "Modules in Python" file.
# calling addop function.
aop.addop(20,10)      # Function call

# calling subop function.
aop.subop(20,10)      # Function call

sum(20,10)=30
sub(20,10)=10

```

```
In [7]: # import the "aop" file (module)
import aop as a          # import module name as alias name.

# calling the "aop" functions in "Modules in Python" file.
# calling addop function.
a.addop(300,110)          # Function call

# calling subop function.
a.subop(200,150)          # Function call

sum(300,110)=410
sub(200,150)=50
```

```
In [9]: # import the "aop" file (module)
from aop import addop,subop  # from module name import Function Names.

# calling the "aop" functions in "Modules in Python" file.
# calling addop function.
addop(30,11)              # Function call

# calling subop function.
subop(20,15)              # Function call

sum(30,11)=41
sub(20,15)=5
```

```
In [10]: # import the "aop" file (module).
# from module name import Function Names as alias names.
from aop import addop as a,subop as s

# calling the "aop" functions in "Modules in Python" file.
# calling addop function.
a(310,111)                # Function call

# calling subop function.
s(210,151)                # Function call

sum(310,111)=421
sub(210,151)=59
```

```
In [11]: # import the "aop" file (module).
# from module name import *
from aop import *

# calling the "aop" functions in "Modules in Python" file.
# calling addop function.
addop(10,11)              # Function call

# calling subop function.
subop(110,151)            # Function call

sum(10,11)=21
sub(110,151)=-41
```