Modules:

---->> If we need to reuse the code in python then we can define functions , but if we need to reuse number of functions then we have to go for modules concept.

---->> A module is nothing but a file with extention of .py , which may contains methods , variables & also classes.

---->> In Python , Every python file itself is known as a module .

---->> Modules of python .py files that concists of python code.

Any python file can be referenced as a Module.

---->> In order to use the properties of one module into another module we use the "import" statement.

syntax : import required_moduleName

Modules are three types

1) Standard / predefined / built-in modules
2) User defined Modules.
3) 3rd party modules.

1) Predefined modules

These modules are already defined and kept in python software.

So when we install python then automatically these standard modules will install in our Machine.

For example,

math, calender, os , sys, json , datetime, ...

2) User defined Modules.

These modules are defined by users as per their requirement .

So here User defined module is nothing but the .py file which contains methods, variables, and also classes.

For example: demo.py, sample.py , add.py(), subtract.py

3) 3rd party modules.

These modules are already defined by some other people and kept in internet.

So we can download and install in our machines by using "pip" (Python Installer Package).

---->> PIP is a package management system used to install and manage software packages written in python like pymysql , cx_oracle.

------>> In python, modules are accessed by using "import" statement.

------>> When our current file needed to use the code which is already existed in other files we can import that file(modules)

------>> When python import a module called as "Mymodule" for example , the interpreter will first search for a built-in module called Mymodule.

----->> If a built-in module is not found , the python interpretor will then search for a file named Mymodule.py in a list of directories that it receives from the sys.path variables.

We can import modules in 3 diffrent ways ,

1) import <module_name>

```
import math

print(math.pow(2,4))
```

---->> It is not recomended way in real time programs. Because when importing module name directly then this module contains all members like functions,variables anc classes.

--->> So if we want to use only one member of this module then remaining all members are loading not good.

----->> here every time we need to use module name when we want to use member of this module.

---->> to overcome this problem we can use another way of importing.

2) from <module_name> import *

```
from math import *

print(pow(2,4))   # 16
```

---->> here we can import all members of required module at a time. we can use these members directly without module name.

3) from <module_name> import *

```
from math import pow
print(pow(2,4))  # 16
```

---->> here we can directly importing required module name only what we want.

---->> we can use this member directly.

Example:

module1.py file

```
x = 100
def f1():
    print("in f1 of mod1")
def f2():
    print("in f2 of mod1")
```

module2.py file

```
import module1.py
print(module1.x)
module1.f1()
module1.f2()
print("end")
```

output:

```
100
in f1 of mod1
in f2 of mod2
end
```

Note: Whenever we import one module into another module then imported module file will be generated and stored that file into computer hard disc premenently.

After generating the compiled file for python module with out sharing the .py file of that module, we can import that module into other module.

Example:

mod3.py

```
def add(a,b):
    return a+b

def sub(a,b):
    return a-b

def abs(a):
    if a >= 0:
        return a
    else:
        return (-a)
```

mod4.py

```
import  mod3
sum = mod3.add(30,20)
print(sum)
sub = mod3.sub(30,20)
print(sub)
pr = mod3.abs(-10)
print(pr)
```

output:

```
50
10
10
```

Renaming a module at the time of importing :

When ever we import a module , to access the properties of that module then compulsary we have to use modulename.propertyname

--->> If we rename a module at the time of importing, we can access the propterties of that module by using alicename.propertyName

For Example:

We can import the "math" module by using alias name , which is providing all mathematicl properties .

```
import math as m

print("the value of pi is " , m.pi)
```

Output: the value of pi is 3.14159265359

from ... import ... :

Inorder to import the specific properties of one module into another module we use

Syntax like bellow,

```
from <module_name> import <property_name>
```

Example:

mod6.py

```
from math import pi,e

print("the value of pi is",pi)

print("the value of e is",e)
```

Output:

the value of pi is 3.14159265354

the value of e is 2.17182

Note:

Whenever we import the properties of one module into another module by using

"from ... import ... " then we can access the properties of that module directly without using module name or alias name .

Example:

```
from math import  pi

print(pi)
```

import statement

import is used to import all the properties of the module at a time.

Example:

```
from math import *

print("the value of pi is",pi)

print("the value of e is",e)
```

Output:

'the value of pi is', 3.14159265358793

'the value of e is', 2.718281828459045

How to import a module multiple times in another module:

reload()

When the module is imported into a script, the code in the top-level portion of a module is executed only once.

Therefore, if you want to reexecute the top-level code in a module, you can use the reload() function. The reload() function imports a previously imported module again. The syntax of the reload() function is like below,

syntax:

```
import importlib

importlib.reload(module_name)
```

Here, module_name is the name of the module you want to reload and not the string containing the module name. For example, to reload hello module, do the following -

```
reload(hello)
```

For example:

```
import mod1
# print(x)  # NameError: name 'x' is not defined
print(mod1.x)
mod1.f1()
print(mod1.add(10,20))
print(mod1.add(50,20))
print()
import importlib
importlib.reload(mod1)
```

Note:

---->For every module by default some properties are added automatically like bellow.

[__builtins__ , __doc__ , __file__ , __package__ , __name__]

Example:

```
x = 1000
def f1():
    ''' this is f1() documentation'''
    print("in f1 of mods")
    print(__builtins__)
    print(f1.__doc__)
    print(__file__)
    print(__package__)
    print(__name__)
f1()
print(x)
```

Output:

```
in f1 of mods
```

<module 'builtins' (built-in)>

this is f1() documentation

D:\Python_Batch\module2.py

None

__main__

1000

dir() function:

dir() function is used to find / represent the properties of the current module or specified module.

print(dir())

Output:

['__annotations__', '__builtins__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'argv', 'f1', 'x']

help()

Use the help() function to know the methods and properties of a module. For example, call the help("math") to know about the math module. If you already imported a module, then provide its name, e.g. help(math).

>>> import math

>>> help(math)

Working with math module:

>>> math.pow(2,3)

8.0

>>> math.pi

3.141592653589793

>>> math.sqrt(16)

4.0

>>> math.factorial(5)

```
120
>>> math.ceil(10.5)
11
>>> math.ceil(10.6)
11
>>> math.ceil(10.2)
11
>>> math.floor(10.2)
10
```