

Syntax2: import module-name as alias-name

This syntax allows importing module with alternative name or another name.

Why module should import with another name?

1. To overcome duplication
2. To shorter imported module name

import math as x # To avoid duplication import numpy as np # To shorter module name math=100 print(math) x.sqrt(9) a=np.array([1,2,3])	Output 100
---	----------------------

Syntax3: from module-name import identifier,identifier,...

This syntax allows importing content of module as part of current module name space

module1.py	module2.py
x=100 y=200 def fun1(): print("fun1") def fun2(): print("fun2") def fun3(): print("fun3")	from module1 import x,y print(x) print(y) import module1 print(module1.x) print(module1.y)

Users.py	Main.py
registered_users={} def signup(uname,pwd): if uname in registered_users: print(f'{uname} exists') else: registered_users[uname]=pwd print("User Registered") def signin(uname,pwd): if uname in registered_users and pwd==registered_users[uname]: print(f'{uname} Welcome') else: print("Invalid username or password")	from users import signup,signin while True: print("1.Signup") print("2.Signin") print("3.Exit") opt=int(input("Enter Your Option:")) if opt==1: user=input("UserName :") pwd=input("Password") signup(user,pwd) elif opt==2: user=input("UserName :") pwd=input("Password :") signin(user,pwd) elif opt==3: break

	2.Signin 3.Exit Enter Your Option:1 UserName :nit Passwordn123 User Registered 1.Signup 2.Signin 3.Exit Enter Your Option:2 UserName :nit Password :n123 nit Welcome
--	--

Syntax4: from module-name import *

This syntax allows importing all the names or identifiers as a part of current module.

Umath.py	Main1.py
<pre>def isprime(num): c=0 for i in range(1,num+1): if num%i==0: c+=1 return c==2 def ispal(num): rev=0 num1=num while num>0: r=num%10 rev=(rev*10)+r num=num//10 return rev==num1 def iseiven(num): return num%2==0 def isodd(num): return num%2!=0</pre>	<pre>from umath import * r1=isprime(5) print(r1) r2=iseven(7) print(r2) r3=isodd(9) print(r3) r4=ispal(121) print(r4)</pre> <p>Output</p> <pre>True False True True</pre>

Syntax5: from module-name import identifier as alias-name

Main2.py
<pre>from umath import isprime as prime from umath import iseiven as even isprime=100 iseiven=200 r1=prime(5) r2=even(4)</pre>

```
print(isprime)
print(iseven)
print(r1)
print(r2)
```

Output

```
100
200
True
True
```

Syntax6: import module-name,module-name,...

"import" keyword can be used import more than one module by separating with comma

Main3.py

```
import math,umath
print(math.sqrt(9))
print(umath.isprime(9))
```

Output

```
3.0
False
```

How to set module path?

There are two methods of configuring module PYTHONPATH

1. Using os environment variable path
2. using program (OR) within program configuring path

Using os environment variable PYTHONPATH

Setting the PYTHONPATH environment variable: This method permanently adds the specified path to the PYTHONPATH environment variable, which Python checks for module paths. This change is effective for all Python sessions.

On Windows:

Search for "environment variables" in the Start Menu and select "Edit the system environment variables."

Click on "Environment Variables."

In the "System variables" section, find the variable named "PYTHONPATH" (create it if it doesn't exist).

Click "Edit" and add the path to your module, separated by semicolons from existing paths.

Click "OK" on all windows to save the changes.

Using sys.path.append(): This method temporarily adds the specified path to the sys.path list, which contains the directories Python searches for modules. This change is only effective for the current session.

Create module with name module1.py and save inside d: drive within a folder named folder3

module1.py (d:\folder3)

```
def fun1():
    Print("fun1")
def fun2():
    Print("fun2")
def fun3():
    Print("fun3")
```

Main4.py (d:\fsp5pm)

```
import sys
sys.path.append("d:\\folder3")
import module1

module1.fun1()
module1.fun2()
```

Packages

What is package?

Package is collection of modules (.py files) (OR) python programs.

Technically package is nothing but folder, which consist of .py files.

Packing is nothing but grouping set of related programs and referring with one name.

numpy → package
sys.py → module
os.py

calendar.py
pandas → package

Advantage of packages

1. Grouping and referring with one name
2. Maintaining of modules becomes easy
3. Hiding modules
4. Creating app installer or compressing in single file (.whl)