

MODULES

- A group of functions, variables and classes saved to a file, which is nothing but module.
- Every Python file (.py) acts as a module.

abcmath.py

```

1) x = 888
2)
3) def add(a,b):
4)   print("The Sum:",a+b)
5)
6) def product(a,b):
7)   print("The Product:",a*b)

```

- abcmath module contains one variable and 2 functions.
- If we want to use members of module in our program then we should import that module.
`import modulename`
- We can access members by using module name.
`modulename.variable`
`modulename.function()`

test.py:

```

1) import abcmath
2) print(abcmath.x)
3) abcmath.add(10,20)
4) abcmath.product(10,20)

```

Output

888
The Sum: 30
The Product: 200

Note: Whenever we are using a module in our program, for that module compiled file will be generated and stored in the hard disk permanently.

Renaming a Module at the time of import (Module Aliasing):

- Eg: `import abcmath as m`
- Here `abcmath` is original module name and `m` is alias name.
- We can access members by using alias name `m`

test.py:

```
1) import abcmath as m
2) print(m.x)
3) m.add(10,20)
4) m.product(10,20)
```

from ... import:

We can import particular members of module by using `from ... import`.
The main advantage of this is we can access members directly without using module name.

```
1) from abcmath import x,add
2) print(x)
3) add(10,20)
4) product(10,20) → NameError: name 'product' is not defined
```

We can import all members of a module as follows `from abcmath import *`

test.py:

```
1) from abcmath import *
2) print(x)
3) add(10,20)
4) product(10,20)
```

Various Possibilities of import:

- 1) `import modulename`
- 2) `import module1,module2,module3`
- 3) `import module1 as m`
- 4) `import module1 as m1,module2 as m2,module3`
- 5) `from module import member`
- 6) `from module import member1,member2,memebr3`
- 7) `from module import memeber1 as x`
- 8) `from module import *`

Member Aliasing:

```
1) from abcmath import x as y,add as sum  
2) print(y)  
3) sum(10,20)
```

Once we defined as alias name,we should use alias name only and we should not use original name

```
1) from abcmath import x as y  
2) print(x) → NameError: name 'x' is not defined
```

Reloading a Module:

By default module will be loaded only once even though we are importing multiple times.

module1.py:

```
print("This is from module1")
```

test.py

```
1) import module1  
2) import module1  
3) import module1  
4) import module1  
5) print("This is test module")
```

Output

This is from module1

This is test module

- In the above program test module will be loaded only once even though we are importing multiple times.
- The problem in this approach is after loading a module if it is updated outside then updated version of module1 is not available to our program.
- We can solve this problem by reloading module explicitly based on our requirement.
- We can reload by using reload() function of imp module.

```
1) import imp  
2) imp.reload(module1)
```

test.py:

```
1) import module1
2) import module1
3) from imp import reload
4) reload(module1)
5) reload(module1)
6) reload(module1)
7) print("This is test module")
```

In the above program module1 will be loaded 4 times in that 1 time by default and 3 times explicitly. In this case output is

- 1) This is from module1
- 2) This is from module1
- 3) This is from module1
- 4) This is from module1
- 5) This is test module

The main advantage of explicit module reloading is we can ensure that updated version is always available to our program.

Finding Members of Module by using dir() Function:

Python provides inbuilt function dir() to list out all members of current module or a specified module.

dir() → To list out all members of current module

dir(moduleName) → To list out all members of specified module

Eg 1: test.py

```
1) x=10
2) y=20
3) def f1():
4)     print("Hello")
5) print(dir()) # To print all members of current module
```

Output

```
['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'f1', 'x', 'y']
```

Eg 2: To display members of particular module

abcmath.py:

```
1) x=888
2)
3) def add(a,b):
4)     print("The Sum:",a+b)
5)
6) def product(a,b):
7)     print("The Product:",a*b)
```

test.py:

```
1) import abcmath
2) print(dir(abcmath))
```

Output

```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'add', 'product', 'x']
```

Note: For every module at the time of execution Python interpreter will add some special properties automatically for internal use.

Eg: __builtins__, __cached__, __doc__, __file__, __loader__, __name__, __package__,
__spec__

Based on our requirement we can access these properties also in our program.

Eg: test.py

```
1) print(__builtins__)
2) print(__cached__)
3) print(__doc__)
4) print(__file__)
5) print(__loader__)
6) print(__name__)
7) print(__package__)
8) print(__spec__)
```

Output

```
<module 'builtins' (built-in)>
```

```
None
```

```
None
```

test.py

```
1) <_frozen_importlib_external.SourceFileLoader object at 0x00572170>
2) __main__
3) None
4) None
```

The Special Variable `__name__`:

- For every Python program, a special variable `__name__` will be added internally.
- This variable stores information regarding whether the program is executed as an individual program or as a module.
- If the program executed as an individual program then the value of this variable is `__main__`
- If the program executed as a module from some other program then the value of this variable is the name of module where it is defined.
- Hence by using this `__name__` variable we can identify whether the program executed directly or as a module.

Demo program:

module1.py:

```
1) def f1():
2)     if __name__=='__main__':
3)         print("The code executed as a program")
4)     else:
5)         print("The code executed as a module from some other program")
6) f1()
```

test.py:

```
1) import module1
2) module1.f1()
```

D:\Python_classes>py module1.py

The code executed as a program

D:\Python_classes>py test.py

The code executed as a module `from` some other program

The code executed as a module `from` some other program

Working with math Module:

- Python provides inbuilt module math.
- This module defines several functions which can be used for mathematical operations.
- The main important functions are
 - 1) `sqrt(x)`
 - 2) `ceil(x)`
 - 3) `floor(x)`
 - 4) `fabs(x)`
 - 5) `log(x)`
 - 6) `sin(x)`
 - 7) `tan(x)`
 - 8)

```
1) from math import *
2) print(sqrt(4))
3) print(ceil(10.1))
4) print(floor(10.1))
5) print(fabs(-10.6))
6) print(fabs(10.6))
```

Output

2.0
11
10
10.6
10.6

Note: We can find help for any module by using `help()` function

Eg:

```
import math
help(math)
```

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.

1) `random()` Function:

This function always generate some float value between 0 and 1 (not inclusive)
 $0 < x < 1$

```
1) from random import *
2) for i in range(10):
3)     print(random())
```

Output

0.4572685609302056
0.6584325233197768
0.15444034016553587
0.18351427005232201
0.1330257265904884
0.9291139798071045
0.6586741197891783
0.8901649834019002
0.25540891083913053
0.7290504335962871

2) randint() Function:

To generate random integer between two given numbers(inclusive)

```
1) from random import *
2) for i in range(10):
3)     print(randint(1,100)) # generate random int value between 1 and 100(inclusive)
```

Output

51
44
39
70
49
74
52
10
40
8

3) uniform() Function:

It returns random float values between 2 given numbers (not inclusive)

```
1) from random import *
2) for i in range(10):
3)     print(uniform(1,10))
```

Output

9.787695398230332

6.81102218793548

8.068672144377329

8.567976357239834

6.363511674803802

2.176137584071641

4.822867939432386

6.0801725149678445

7.508457735544763

1.9982221862917555

random() → in between 0 and 1 (not inclusive)

randint(x,y) → in between x and y (inclusive)

uniform(x,y) → in between x and y (not inclusive)

4) randrange ([start], stop, [step])

- Returns a random number from range
- start \leq x < stop
- start argument is optional and default value is 0
- step argument is optional and default value is 1
- randrange(10) → generates a number from 0 to 9
- randrange(1,11) → generates a number from 1 to 10
- randrange(1,11,2) → generates a number from 1,3,5,7,9

```
1) from random import *
2) for i in range(10):
3)     print(randrange(10))
```

Output: 9

4

0

2

9

4

8

9

5

9

```
1) from random import *
2) for i in range(10):
3)     print(randrange(1,11))
```

Output: 2

```
2  
8  
10  
3  
5  
9  
1  
6  
3
```

```
1) from random import *  
2) for i in range(10):  
3)     print(randrange(1,11,2))
```

Output: 1

```
3  
9  
5  
7  
1  
1  
1  
7  
3
```

5) choice() Function:

- It won't return random number.
- It will return a random object from the given list or tuple.

```
1) from random import *  
2) list=["Sunny","Bunny","Chinny","Vinny","pinny"]  
3) for i in range(10):  
4)     print(choice(list))
```

Output

```
Bunny  
pinny  
Bunny  
Sunny  
Bunny  
pinny  
pinny  
Vinny  
Bunny  
Sunny
```