Modules

1. Introduction ¶

A group of functions, variables and classes saved to a file, which is nothing but module.

Every Python file (.py) acts as a module.

Eg:

In [2]:

```
x = 888
y = 999
def add(a,b):
    print('The Sum : ',a+b)
def product(a,b):
    print('The product :', a*b)
```

Let me save this code as rgm.py, and itself is a module.

rgm.py module contains two variables and 2 functions.

If we want to use members of module in our program then we should import that module.

Syntax of importing a module

```
import modulename
```

We can access members by using module name.

modulename.variable

modulename.function()

In [1]:

```
import rgm
print(rgm.x)
rgm.add(10,20)
rgm.product(10,20)
```

```
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.This is avaialble at _ _ pycache_ _ file, which is available at current working directory.

2.Advantages of Modules

1. Code Reusability
2. Readability improved
3. Maintainability improved

3. Renaming a module at the time of import (module aliasing):

We can create alias name for a module. This can be done as follows:

import rgm as r

Here, rgm is original module name and r** is alias name. We can access members by using alias name **r.

Eg:

In [2]:

```
import rgm as r
print(r.x)
r.add(10,20)
r.product(10,20)
```

```
888
The Sum :  30
The product : 200
```

In [1]:

```
import rgm as r
print(rgm.x)
rgm.add(10,20)
rgm.product(10,20)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [1], in <cell line: 2>()
      1 import rgm as r
----> 2 print(rgm.x)
      3 rgm.add(10,20)
      4 rgm.product(10,20)

NameError: name 'rgm' is not defined
```

Note:

Once we define alias name for a module, compulsory you should use alias name only. original names by default will be gone related to that particular file.

4. 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.

Eg :

In [1]:

```
from rgm import x,add
print(x)
add(10,20)
product(10,20)
```

```
888
The Sum :  30

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [1], in <cell line: 4>()
      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 rgm import

Eg :

In [1]:

```
from rgm import *
print(x)
add(10,20)
product(10,20)
```

```
888
The Sum :  30
The product : 200
```

5. member aliasing

Similar to module aliasing, member aliasing also possible in python. This can be done as follows:

from rgm import x as y,add as sum

print(y)

sum(10,20)

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

Eg:

In [1]:

```
from rgm import x as y
print(x)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [1], in <cell line: 2>()
      1 from rgm import x as y
----> 2 print(x)

NameError: name 'x' is not defined
```

6. Various possibilties of import

import modulename

import module1,module2,module3

import module1 as m

import module1 as m1,module2 as m2,module

from module import member

from module import member1,member2,memebr3

from module import memeber1 as x

from module import *

7. Reloading a Module

By default, module will be loaded only once eventhough we are importing multiple times.

Demo Program for module reloading:

Assume that we created a module named as module1.py.

In [2]:

```
print('This is from module 1')
```

```
This is from module 1
```

In [5]:

```
%%writefile module1.py
print("This is Module 1")
```

```
Overwriting module1.py
```

In [4]:

```
import module1
import module1
import module1
import module1
import module1
import module1
print('This is Test Module')
```

```
This is Module 1
This is Test Module
```

In [1]:

```
import module1
'''import module1
import module1
import module1
import module1
import module1'''
print('This is Test Module')
```

```
This is Module 1
This is Test Module
```

In the above program test module will be loaded only once eventhough 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.

8. 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: To display members of current module

In [1]:

```
x=10
y=20
def f1():
    print("Hello")
print(dir())
```

```
['In', 'Out', '_', '__', '___', '__builtin__', '__builtins__', '__doc__', '__l
oader__', '__name__', '__package__', '__spec__', '_dh', '_i', '_i1', '_ih', '_
ii', '_iii', '_oh', 'exit', 'f1', 'get_ipython', 'quit', 'x', 'y']
```

Eg 2: To display members of particular module

Consider rgm.py module,

In [2]:

```
import rgm
print(dir(rgm))
```

```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name_
_', '__package__', '__spec__', 'add', 'product', 'x', 'y']
```

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.

9. 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

In [3]:

```
def f1():
    if __name__=='__main__':
        print("The code executed directly as a program")
        print("The value of __name__:",__name__)
    else:
        print("The code executed indirectly as a module from some other program")
        print("The value of __name__:",__name__)
f1()
```

```
The code executed directly as a program
The value of __name__: __main__
```

In [1]:

```
import module1
print("From test we are executing module f1()")
module1.f1()
```

```
This is Module 1
From test we are executing module f1()
The code executed indirectly as a module from some other program
The value of __name__: module1
```

Write a Python program to generate a six digit random number as One Time Password (OTP).

In [2]:

```
from random import *
for i in range(10):
    for x in range(6):
        print(randint(0,9),end='')
    print()
```

```
375176
631462
002901
142503
175605
587189
629871
648681
842331
038407
```

Write aPython program to generate a random password of 6 length.

Within the OTP --

1,3,5 positions are alphabets.

2,4,6 positions are digits.

In [4]:

```
from random import *
for i in range(10):
    for x in range(1,7):
        if x%2 == 1:
            print(chr(randint(65,90)),end='')
        else:
            print(randint(0,9),end='')
    print()
```

```
G0F7Z7
O8M5N2
I7Q2W5
Q9V2M0
J8F4O6
P9O5F9
S7A0M6
A2W3J7
W4L1X7
E8V4G9
```

Working with Directories:

It is very common requirement to perform operations for directories like

1. To know current working directory
2. To create a new directory
3. To remove an existing directory
4. To rename a directory

5. To list contents of the directory

To perform these operations,Python provides inbuilt module os,which contains several functions to perform directory related operations.

To Know Current Working Directory:

In [5]:

```
import os
cwd=os.getcwd()
print("Current Working Directory:",cwd)
```

```
Current Working Directory: C:\Users\pdsin\Python_Parth Sinroza\FCSP-1
```

To create a sub directory in the current working directory:

In [6]:

```
import os
os.mkdir("mysub")
print("mysub directory created in cwd")
```

```
mysub directory created in cwd
```

To create a sub directory in mysub directory:

cwd

```
    |-mysub

        |-mysub2
```

In [8]:

```
import os
os.mkdir("mysub/mysub2")
print("mysub2 created inside mysub")
```

```
mysub2 created inside mysub
```

To remove a directory:

In [9]:

```
import os
os.rmdir("mysub/mysub2")
print("mysub2 directory deleted")
```

```
mysub2 directory deleted
```

To remove a file:

In [12]:

```
import os
os.remove('abc.txt')
```

To rename a directory:

In [10]:

```
import os
os.rename("mysub","newdir")
print("mysub directory renamed to newdir")
```

```
mysub directory renamed to newdir
```

To know contents of directory:

os module provides listdir() to list out the contents of the specified directory. It won't display the contents of sub directory.

Eg:

In [11]:

```
import os
print(os.listdir("."))
```

```
['.ipynb_checkpoints', 'abc.txt', 'abcd.txt', 'big.txt', 'copy.txt', 'data.tx
t', 'database.txt', 'file1.txt', 'file2.txt', 'file3.txt', 'filter.txt', 'filt
er2.txt', 'friends.txt', 'lec1.txt', 'lec2.txt', 'module1.py', 'newdir', 'Prac
tice.ipynb', 'rgm.py', 'testfile', 'Unit-1.ipynb', 'Unit-10.ipynb', 'Unit-1_No
tes.ipynb', 'Unit-2.ipynb', 'Unit-2_Notes.ipynb', 'Unit-3 Notes.ipynb', 'Unit-
3.ipynb', 'Unit-4 Strings.ipynb', 'Unit-4 Tuple.ipynb', 'Unit-4.ipynb', 'Unit-
5 Dictionary.ipynb', 'Unit-5 Lambda Function.ipynb', 'Unit-5 Lists.ipynb', 'Un
it-5 Sets.ipynb', 'Unit-5.ipynb', 'Unit-6 Files.ipynb', 'Unit-6,7.ipynb', 'Uni
t-6.ipynb', 'Unit-7 Module and Directories.ipynb', 'Unit-8.ipynb', 'Unit-8_Pra
ctice.ipynb', '__pycache__']
```

The above program display contents of current working directory but not contents of sub directories.

SYS MODULE

To get the maxsize

In [13]:

```
import sys
print(sys.maxsize)
```

```
9223372036854775807
```

To get version of Python installed

In [14]:

```
import sys
print(sys.version)
```

```
3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit (AMD6
4)]
```

To get path associated

In [15]:

```
import sys
print(sys.path)
```

```
['C:\\Users\\pdsin\\Python_Parth Sinroza\\FCSP-1', 'C:\\Users\\pdsin\\AppData
\\Local\\Programs\\Python\\Python310\\python310.zip', 'C:\\Users\\pdsin\\AppDa
ta\\Local\\Programs\\Python\\Python310\\DLLs', 'C:\\Users\\pdsin\\AppData\\Loc
al\\Programs\\Python\\Python310\\lib', 'C:\\Users\\pdsin\\AppData\\Local\\Prog
rams\\Python\\Python310', '', 'C:\\Users\\pdsin\\AppData\\Local\\Programs\\Pyt
hon\\Python310\\lib\\site-packages', 'C:\\Users\\pdsin\\AppData\\Local\\Progra
ms\\Python\\Python310\\lib\\site-packages\\win32', 'C:\\Users\\pdsin\\AppData
\\Local\\Programs\\Python\\Python310\\lib\\site-packages\\win32\\lib', 'C:\\Us
ers\\pdsin\\AppData\\Local\\Programs\\Python\\Python310\\lib\\site-packages\\P
ythonwin']
```

WAP that defines a function large in a module which will be used to find larger of two values and called from code in another module

In [16]:

```
%%writefile MyModule.py
def large(a,b):
    if a>b:
        return a
    else:
        return b
```

```
Writing MyModule.py
```

In [18]:

```
import MyModule
print('Large(50,100)=',MyModule.large(50,100))
print("Large('Hi','Bi')=",MyModule.large('Hi','Bi'))
```

```
Large(50,100)= 100
Large('Hi','Bi')= Hi
```

WAP that prints square root and cube of a number:

In [2]:

```
import math
def cube(x):
    return x**3

a=100

print('Square root of',a,'is',math.sqrt(a))
print('Cube of',a,'is',cube(a))
```

```
Square root of 100 is 10.0
Cube of 100 is 1000000
```

WAP to display date and time using time module

In [3]:

```
import time
print(time.asctime())
```

```
Tue Jan 24 13:23:00 2023
```

WAP that uses the getpass module to prompt user for a password, without echoing what they type to the console.

In [4]:

```
import getpass

password=getpass.getpass(prompt='Enter Password: ')

if password=="LJU":
    print("Welcome, Correct Password")
else:
    print("Incorrect Password")
```

```
Enter Password: ·····
Welcome, Correct Password
```