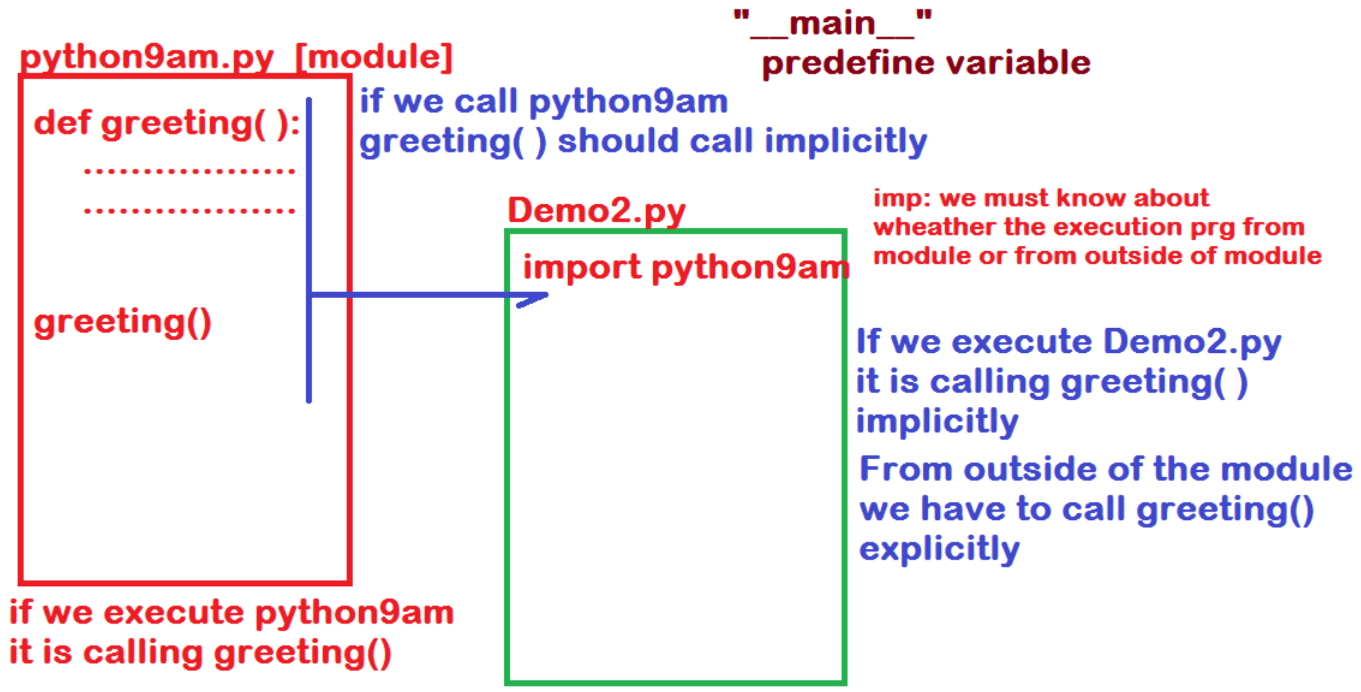
Working with Modules

- In C-language we have some libraries such as stdio.h , conio.h , math.h and string.h .....
- Library is the collection of predefined and user defined function , variable and macros
- Similarly In Java we have packages [java.lang , java.io, java.awt ...]
- In Python we have modules
- Module may contains variables , classes ,object and function
    - Keyword module is having
        - kwlist [predefine variable]
        - iskeyword( ) predefined function
- Modules are classified into 2 types
    - Predefined Modules | built-in modules
        - The Modules which are provided by python software from python software foundation
            - Keyword | time | os | random | sys | gc | re Collections | functools
    - User defined Modules
        - The modules which are defined by us for our application requirements called user defined modules | customized modules
    - Note : Every individual python script file will be considered as a module
- Advantages of Modules are mainly reusability of properties

```
#myinfo.py [module]
print("Hello MyDear ....!!!")
lst=[10,20,30,40,50]


#Demo.py
import myinfo   #user defined
print("type of lst  is : ",type(myinfo.lst))
print("List : ",myinfo.lst)

import keyword   #pre defined module
print("Type of kwlist : ",type(keyword.kwlist) )
print("Keyword : ",keyword.kwlist)

Example 2:
#python9am.py  [module]

def greetings():
   print("Hello ")
   print("MyDear ")
   print("How r u ....")

#calling
greetings()


#Demo2.py
import python9am
```


"__main__"
predefine variable
python9am.py [module]
def greeting( ):
....
....
greeting()
if we call python9am
greeting( ) should call implicitly
Demo2.py
import python9am
imp: we must know about wheather the execution prg from module or from outside of module
If we execute Demo2.py it is calling greeting( ) implicitly
From outside of the module we have to call greeting() explicitly
if we execute python9am it is calling greeting()

print( ) | type( ) | id( ) | del | exit( ) all are from built-in module. __name__ is the predefine variable from built-in module

- If you want use anything from the built-in module then it is not required import just bcoz it is implicitly imported.

- **Default value for "__name__" variable is '__main__'**
- **If the value of "__name__" is "__main__'' then the execution process is started from main module only**
- **If the value of "__name__'' is other than "__main__" then the execution process is not started from main module i.e execution process is started from outside of the module**

```
#python9am.py  [module]

def greetings():
   print("Hello ")
   print("MyDear ")
   print("How r u ....")

#calling
print("Type is : ",type(__name__))  #<class 'str'>
print("value of __name__ is : ",__name__)  #__main__

if __name__=='__main__':
   greetings()

#Demo2.py   From outside of main module
import python9am
python9am.greetings()
```

Note: After importing any module, To access any member from the module then we must use module name along with it's members

```
#sssit.py   [module]

def greetings():
   print("Hello")
   print("My Dear ")
   print("Have a nice Day ....")

lst=[10,20,30,40,50,60]

stu={"sno":101,"sname":"ramesh","scity":"hyd","spin":500090}

#demo3.py   [outside]

import sssit
sssit.greetings()
print("List : ",sssit.lst)
print("Student Data : ")
print(sssit.stu)
```

Note: We avoid using the lengthy module names along with the members of module by using alias for module by using keyword "as"

```
#demo4.py   [outside]
# alias for module
#syn for module alias
#import <modulename> as <aliasname>

import sssit as s

s.greetings()
```

```
print("List : ",s.lst)
print("Student Data : ")
print(s.stu)
```

Based on our application requirements then we can also import the required members from the module rather than importing all the members, thus it can save the memory and increases efficiency by using “from” keyword

If we import the members from the module using “from “ keyword then it is not required to use module name along with its members

```
#demo5.py   [outside]
#from <modulename> import <members>

from sssit import greetings,lst    [required members]
from sssit import *     [All the members from the module]

greetings()
print(lst)
print(stu)
```

- We can also import multiple modules by using a import statement

```
#demo6.py   [outside]
#import <module1>,<module2>,.......

import time,sssit,keyword

sssit.greetings();

print("Keywords ")
for keyword in keyword.kwlist:
   time.sleep(.1)
   print(keyword)
```

Example:

```
#demo67py   [outside]

import time
from sssit import *
from keyword import *

greetings()

print("Keywords ")
for keyword in kwlist:
   time.sleep(.1)
   print(keyword)
```

```
#Dev_Team.py   [Module]

print("Code from ")
print("Dev_Team ")

print("Have a nice Day")
print("Modified")
```

```
#demo8.py   [outside]
import time
import importlib

import Dev_Team
print("Code is done @ team - Member")

time.sleep(30)
importlib.reload( Dev_Team )
```

Note: If you want see the all modules which are provided by python software foundation. Then we have use

```
>>>help('modules')
```

If you want see the members existed in the particular module then we have to use dir( )

```
>>>import time
>>>help(time.time)

>>> import keyword
>>> dir(keyword)
['__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', 'iskeyword', 'kwlist']
>>> help(keyword.kwlist)
Help on list object:



>>> import importlib
>>> dir(importlib)
['_RELOADING', '__all__', '__builtins__', '__cached__', '__doc__', '__file__',
'__import__', '__loader__', '__name__', '__package__', '__path__',
'__spec__', '_bootstrap', '_bootstrap_external', '_imp', '_pack_uint32',
'_unpack_uint32', 'abc', 'find_loader', 'import_module',
'invalidate_caches', 'machinery', 'reload', 'sys', 'types', 'util',
'warnings']
```