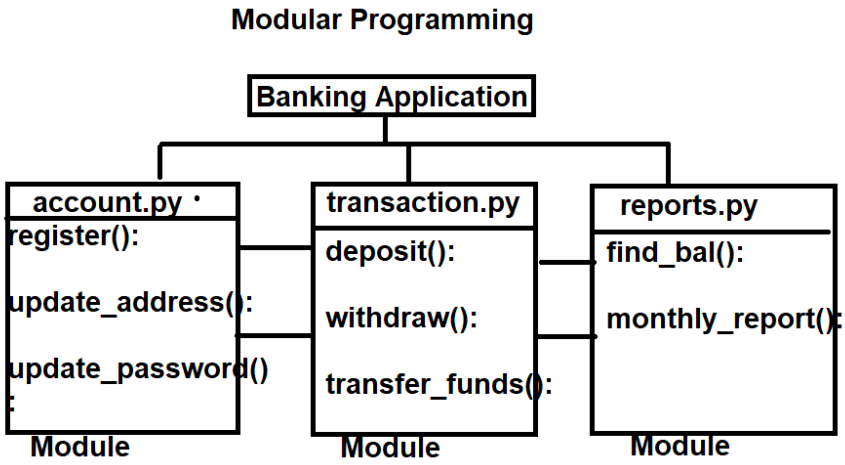
Modules and Packages

What is module?

A module is python program. Modules in python are saved with extension .py.
A module is collection of variables, functions and classes.
Modules allow reusability between programs.



Module provides,

1. Reusability between programs
2. Easy to understand
3. Modularity (Dividing application functionality into number of programs).

Modules are two types.

1. Predefined modules
2. User defined modules

Predefined modules

Existing modules are called predefined modules; these modules are also called libraries.
Example: os module, calendar module, datetime module,…

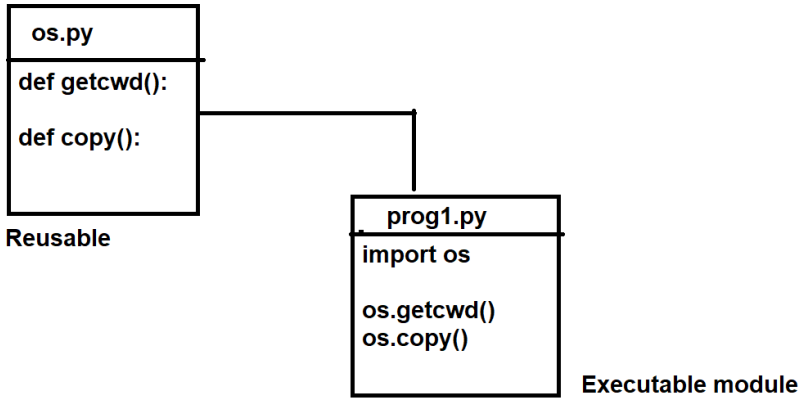
User defined modules

Modules developed by programmer are called user defined modules or application specific modules.
Creating module is nothing but creating python program.

Modules are two types

1. Executable module or Program
2. Non executable module or reusable module or program

Executable module is having executable statements, this module can run. Non executable module does not have executable statements, this module functionality is used inside another module.



import keyword or import statement

importing or using the content of one module inside another module.

Syntax1: import <module-name>
Syntax2: import <module-name> as <alias-name>
Syntax3: from <module-name> import <identifiers>
Syntax4: from <module-name> import *
Syntax5: from <module-name> import <identifier> as <alias-name>,<identifier> as <alias-name>,..

Syntax1: import <module-name>

This syntax imports the entire module. Importing is not including file, it is process of verifying module name. If module name is exists PVM creating namespace with module name along with path.

module1.py	**module2.py**

def fun1():
  print("inside fun1 of module1")
def fun2():
  print("inside fun2 of module1")
def fun3():
	import module1
	
	print(globals())
	module1.fun1()
	module1.fun2()

print("inside fun3 of module1")	module1.fun3() **Output** inside fun1 of module1 inside fun2 of module1 inside fun3 of module1

mathmodule.py	**Test3.py**

def isEven(num):
  return num%2==0
def isPrime(num):
  c=0
  for i in range(1,num+1):
  if num%i==0:
  c=c+1
  return c==2
 	
def factorial(num):
  if num==0:
  return 1
  else:
  return num*factorial(num-1)
	import mathmodule
	
	res1=mathmodule.isEven(4)
	res2=mathmodule.isEven(9)
	res3=mathmodule.isPrime(5)
	res4=mathmodule.factorial(5)
	print(res1,res2,res3,res4)
	
	Output:
	True False True 120

Syntax-2:
Import <module-name> as <alias-name>

Import module with alias name or alternative name. if module name is very big, it is imported with alias name

usersmodule.py	**test4.py**

def user_register(u,p):
  if u in users:
  print(f'{u} is exists')
  else:
  users[u]=p
  print(f'{u} is registered')
	import usersmodule as users
	
	print(globals())
	while True:
	  print("1. User Register")
	  print("2. Login")

```
def login(u,p):
    if u in users and users[u]==p:
        print(f'{u} welcome')
    else:
        print("invalid user name or
password")
```

```
    print("3. Exit")
    opt=int(input("Enter Your
Option "))
    if opt==1:
        user=input("UserName ")
        pwd=input("Password ")
users.user_register(user,pwd)
    elif opt==2:
        user=input("UserName ")
        pwd=input("Password ")
        users.login(user,pwd)
    elif opt==3:
        break
```

Output
1. User Register
2. Login
3. Exit
Enter Your Option 3

Syntax3: from module-name import <identifier-name>,<identifier-name>,…

Identifiers are variable-name, function names or class names. This avoids using of module name

mathmodule.py

```
def isEven(num):
    return num%2==0
def isPrime(num):
    c=0
    for i in range(1,num+1):
        if num%i==0:
            c=c+1
    return c==2
```

Test5.py

```
from mathmodule import
isEven,isPrime


res1=isEven(5)
print(res1)
res2=isPrime(9)
print(res2)
```

```
def factorial(num):
    if num==0:
        return 1
    else:
        return num*factorial(num-1)
```

Syntax4: from module-name import *
This syntax allows using entire module content, without using module name.

mathmodule.py

```
def isEven(num):
    return num%2==0
def isPrime(num):
    c=0
    for i in range(1,num+1):
        if num%i==0:
            c=c+1
    return c==2

def factorial(num):
    if num==0:
        return 1
    else:
        return num*factorial(num-1)
```

Test6.py

```
from mathmodule import *

res1=isEven(5)
print(res1)
res2=isPrime(9)
print(res2)
```

Syntax-5
from module-name import <identifier-name> as <alias-name>,<identifier-name> as <alias-name>,…

This is used for importing identifiers of module with another name or alias name, this avoid duplication of identifiers with current module.

mathmodule.py

```
def isEven(num):
    return num%2==0
def isPrime(num):
```

Test7.py

```
from mathmodule import isEven as even,isPrime as prime
```

c=0
for i in range(1,num+1):
if num%i==0:
c=c+1
return c==2
 	
def factorial(num):
if num==0:
return 1
else:
return num*factorial(num-1)
	res1=even(5)
	print(res1)
	res2=prime(3)
	print(res2)

How to use executable module as a reusable module?