16 Modules and Packages

“Organize, and you will be better off...”

Contents

The Main Module

- A module is a .py file containing definitions and statements. So all .py files that we created so far for our programs are modules.

- When we execute a program its module name is **__main__**. This name is available in the variable **__name__**.

```
def display( ) :
    print('You cannot make History if you use Incognito Mode')

def show( ) :
    print('Pizza is a pie chart of how much pizza is left')

print(__name__)
display( )
show( )
```

On execution of this program, we get the following output:

```
__main__
You cannot make History if you use Incognito Mode
Pizza is a pie chart of how much pizza is left
```

Multiple Modules

- There are two reasons why we may want to create a program that contains multiple modules:

  (a) It makes sense to split a big program into multiple .py files, where each .py file acts as a module.

  Benefit - Ease of development and maintenance.

  (b) We may need a set of handy functions in several programs. In such a case instead of copying these functions in different program files, we may keep them in one file and use them in different programs.

  Benefit - Reuse of existing code.

Importing a Module

- To use the definitions and statements in a module in another module, we need to 'import' it into this module.

```
# functions.py
def display( ) :
    print('Earlier rich owned cars, while poor had horses')

def show( ) :
    print('Now everyone has car, while only rich own horses')
```

```
# usefunctions.py
import functions
functions.display( )
functions.show( )
```

When we execute 'usefunctions.py', it runs as a module with name **__main__**.

import functions makes the definitions in 'functions.py' available in 'usefunctions.py'.

- A module can import multiple modules.

```
import math
import random
import functions             # use function.py of previous program
a = 100
b = 200
print(__name__)
print(math.sin(0.5))
prinr(math.cos(0.5))
print(random.random( ))
print(random.randint(30, 45))
functions.display( )
functions.show( )
```

Here **__name__** contains **__main__** indicating that we are executing the main module. **random** and **math** are standard modules. **functions** is a user-defined module.

Variations of *import*

- The **import** statement can be used in multiple forms.

```
import math
import random
```

  is same as

```
import math, random
```

- If we wish, we can import specific names from a module.

```
from math import sin, cos, tan
from functions import display        # imports only display function
from functions import *              # imports all functions
```

- We can rename a module while importing it. We can then use the new name in place of the original module name.

```
import functions as fun
fun.display( )
```

  or even

```
from functions import display as disp
disp( )
```

Search Sequence

- If we import a module called 'myfuncs', following search sequence will be followed:
  - Interpreter will first search for a built-in module called 'myfuncs'.
  - If such a module is not found, then it will search for it in directory list given by the variable **sys.path**.
- The list in the **sys.path** variable contains directory from where the script has been executed, followed by a list of directories as specified in **PYTHONPATH** environment variable.
- We can print the list of directories in **sys.path** using:

```
import sys
for p in sys.path :
    print(p)
```

Same Code, Different Interpretation

- Suppose we have a module called **functions** in 'functions.py'. If this module has functions **display( )** and **main( )**. We want to use this program sometime as an independent script, and at other times as a module from which we can use **display( )** function.

- To achieve this, we need to write the code in this fashion:

```
# functions.py
def display( ) :
    print('Wright Brothers are responsible for 9/11 too')

def main( ) :
    print('If you beat your own record, you win as well as lose')
    print('Internet connects people at a long distance')
    print('Internet disconnects people at a short distance')
    display( )

if __name__ == '__main__' :
    main( )
```

If we run it as an independent program, **if** will be satisfied. As a result, **main( )** will be called. The name of this function need not be **main( )**.

If we import this module in another program, **if** will fail, so **main( )** will not be called. However, the program can call **display( )** independently.

Packages

- The way drives, folders, subfolders help us organize files in an OS, packages help us organize sub-packages and modules.

- A particular directory is treated as a package if it contains a file named __init__.py in it. The directory may contain other sub-packages and modules in it. __init__.py file may be empty or it may contain some initialization code for the package.

- Suppose there is a package called **pkg** containing a module called **mod.py**. If the module contains functions **f1( )** and **f2( )** then the directory structure would be as follows:

  Directory - pkg
  Contents of pkg directory - mod.py and __init__.py
  Contents of mod.py - f1( ) and f2( )

- Program to use **f1( )** and **f2( )** would be as follows:

```
# mod.py
def f1( ) :
    print('Inside function f1')
def f2( ) :
    print('Inside function f2')
```

```
# client.py
import pkg.mod
pkg.mod.f1( )
pkg.mod.f2( )
```

Third-party Packages

- Pythonistas in Python community create software and make it available for other programmers to use. They use **PyPI**—Python Package Index (www.pypi.org) (http://www.pypi.org) to distribute their software. PyPI maintains the list of such third-party Python packages available.

- There are third-party packages available for literally doing everything under the sun.

- You too can register at PyPI and upload your packages there. You should follow the guidelines given at www.pypi.org (http://www.pypi.org) to create the package, build it and upload it to the Python Package Index.

- To use a package available at PyPI we need to first download it and then install it. The installation is done using a package manager utility called **pip**. pip itself is installed when Python is installed.

- Following command shows how to use pip to install a package pykrige that has been downloaded from PyPI.

```
c:\>pip install pykrige
```

P</> Programs

Problem 16.1

Write a Python program that is organized as follows:

Packages:
messages.funny
messages.curt

Modules:
modf1.py, modf2.py, modf3.py in package messages.funny
modc1.py, modc2.py, modc3.py in package messages.curt

Functions:
funf1( ) in module modf1
funf2( ) in module modf2
funf3( ) in module modf3
func1( ) in module modc1
func2( ) in module modc2
func3( ) in module modc3

Use all the functions in a program **client.py**.

Program

Directory structure will be as follows:

```
messages
    __init__.py
    funny
        __init__.py
        modf1.py
        modf2.py
        modf3.py
    curt
        __init__.py
        modc1.py
        modc2.py
        modc3.py
client.py
```

Of these, **messages**, **funny** and **curt** are directories, rest are files. All **__init__.py** files are empty.

```
# modf1.py
def funf1( ) :
    print('The ability to speak several languages is an asset...')
                print('ability to keep your mouth shut in any language is priceless')
```

```
# modf2.py
def funf2( ) :
    print('If you cut off your left arm...')
    print('then your right arm would be left')
```

```
# modf3.py
def funf3( ) :
    print('Alcohol is a solution!')
```

```
# modc1.py
def func1( ) :
  print('Light travels faster than sound...')
  print('People look intelligent, till they open their mouth')
```

```
# modc2.py
def func2( ) :
  print('There is no physical evidence to say that today is Tuesday...')
  print('We have to trust someone who kept the count since first day')
```

```
# modc3.py
def func3( ) :
  print('We spend five days a week pretending to be someone else...')
  print('in order to spend two days being who we are')
```

```
# client.py
import messages.funny.modf1
import messages.funny.modf2
import messages.funny.modf3

import messages.curt.modc1
import messages.curt.modc2
import messages.curt.modc3

messages.funny.modf1.funf1( )
messages.funny.modf2.funf2( )
messages.funny.modf3.funf3( )

messages.curt.modc1.func1( )
messages.curt.modc2.func2( )
messages.curt.modc3.func3( )
```

Tips

- Directory structure is very important. For a directory to qualify as a package, it has to contain a file **__init__.py**.

Problem 16.2

Rewrite the import statements in Program 16.1, such that using functions in different modules becomes convenient.

Program

```
from messages.funny.modf1 import funf1
from messages.funny.modf2 import funf2
from messages.funny.modf3 import funf3

from messages.curt.modc1 import func1
from messages.curt.modc2 import func2
from messages.curt.modc3 import func3

funf1( )
funf2( )
funf3( )

func1( )
func2( )
func3( )
```

Tips

- Benefit - Calls to functions does not need the dotted syntax.
- Limitation - Only the specified function gets imported.

Problem 16.3

Can we rewrite the following imports using * notation?

```
from messages.curt.modc1 import func1
from messages.curt.modc2 import func2
from messages.curt.modc3 import func3

from messages.funny.modf1 import funf1
from messages.funny.modf2 import funf2
from messages.funny.modf3 import funf3
```

Program

We may use the following import statements:

```
# client.py
from messages.curt.modc1 import *
from messages.curt.modc2 import *
from messages.curt.modc3 import *

from messages.funny.modf1 import *
from messages.funny.modf2 import *
from messages.funny.modf3 import *

funf1( )
funf2( )
funf3( )

func1( )
func2( )
func3( )
```

Tips

- Limitation - Since there is only one function in each module, using * is not so useful.
- Also, * is not so popular as it does not indicate which function/class are we importing.

Exercises

[A] Answer the following questions:

(a) Suppose there are three modules **m1.py**, **m2.py**, **m3.py**, containing functions **f1( )**, **f2( )** and **f3( )** respectively. How will you use those functions in your program?

(b) Write a program containing functions **fun1( )**, **fun2( )**, **fun3( )** and some statements. Add suitable code to the program such that you can use it as a module or a normal program.

(c) Suppose a module **mod.py** contains functions **f1( )**, **f2( )** and **f3( )**. Write 4 forms of import statements to use these functions in your program.

[B] Attempt the following questions:

(a) What is the difference between a module and a package?

(b) What is the purpose behind creating multiple packages and modules?

(c) By default, to which module do the statements in a program belong? How do we access the name of this module?

(d) In the following statement what do **a**, **b**, **c**, **x** represent?

```
import a.b.c.x
```

(e) If module **m** contains a function **fun( )**, what is wrong with the following statements?

```
import m
fun( )
```

(f) What are the contents of **PYTHONPATH** variable? How can we access its contents programmatically?

(g) What does the content of **sys.path** signify? What does the order of contents of **sys.path** signify?

(h) Where a list of third-party packages is maintained?

(i) Which tool is commonly used for installing third-party packages?

(j) Do the following import statements serve the same purpose?

```
# version 1
import a, b, c, d

# version 2
import a
import b
import c
import d

# version 3
from a import *
from b import *
from c import *
from d import *
```

[C] State whether the following statements are True or False:

(a) A function can belong to a module and the module can belong to a package.

(b) A package can contain one or more modules in it.

(c) Nested packages are allowed.

(d) Contents of **sys.path** variable cannot be modified.

(e) In the statement **import a.b.c**, **c** cannot be a function.

(f) It is a good idea to use * to import all the functions/classes defined in a module.