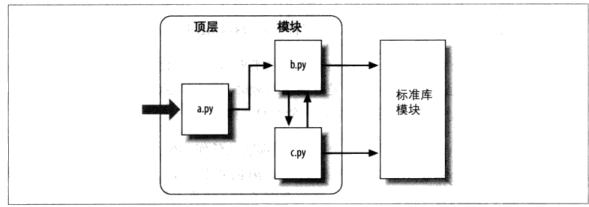
Python中模块对应最高级别的程序组织单元，一般对应Python程序文件.py，也包括用外部语言C/Java/C#的扩展。

Python程序架构：

简单的例子：



模块搜索路径：

1. 程序的主目录

这个目录先被自动搜索，其文件会覆盖其他路径中同样名称的模块。

1. PYTHONPATH目录（如果进行了设置）

需要手动设置。在系统变量中设置PYTHONPATH，需要重启电脑。

1. 标准链接库目录

自动搜索。

1. 任何.pth文件的内容（如果存在）

需要手动设置。要将.pth文件放置在Python的安装路径下，对应的目录为C:\Program Files\Python下，才能识别到.pth文件中的搜索路径。

例：程序module\_test/module\_test1.py

import sys  
import sum # PYTHONPATH  
import sub # PYTHONPATH  
import multiply # .pth文件  
import divide # .pth文件  
print("path: ", sys.path)  
print(sum.sum1(1, 2))  
print(sub.sub1(1, 2))  
print(multiply.multiply1(2, 3))  
print(divide.divide1(5, 2))

from和import均为赋值语句，遵循赋值操作的规则。

模块的导入只发生一次。

例：程序module\_test/small.py

x = 1  
y = [1, 2]  
  
print("x id in small.py", id(x))  
print("y id in small.py", id(y))

程序module\_test/module\_test2.py

# 将small.py模块赋值给small变量  
import small  
  
# 将small.py模块中的x,y赋值给本地的x,y  
from small import x, y  
print(type(small))  
print()  
print("x id in module\_test2.py", id(x))  
print("y id in module\_test2.py", id(y))  
print("original x", x)  
print("original y", y)  
x = 42 # 不变对象  
print("x id ", id(x))  
print("small.x = ", small.x)  
print("small.x id ", id(small.x))  
y[0] = 42 # 可变对象  
print("small.y = ", small.y)

输出为：

x id in small.py 1347972576

y id in small.py 802220131272

<class 'module'>

x id in module\_test2.py 1347972576

y id in module\_test2.py 802220131272

original x 1

original y [1, 2]

x id 1347973888

small.x = 1

small.x id 1347972576

small.y = [42, 2]

从输出可以看出，import将small.py模块对象赋值给small变量，small.x和small.y与small.py中的x，y引用同一个对象。from语句将small.py中的x，y赋值给本地的x，y。在本地修改了x，其对应的id发生了变化，通过修改y[0]，会导致small.y改变。from和import遵循赋值操作的规则，类似函数的调用。

模块命名空间：由模块文件生成命名空间

1. 模块语句会在首次导入时执行。模块第一次导入时，Python会建立空的模块对象，并逐一执行该模块文件内的语句，依照文件从头到尾的顺序。
2. 顶层的赋值语句会创建模块属性。导入时，文件顶层（不在def或class之内）赋值变量的语句（例如=、def、import、from），会建立模块对象的属性，赋值的变量名会存储在模块的命名空间内。
3. 模块是一个独立的作用域（本地变量就是全局变量）

例：程序module\_test/module1.py

print('starting to load...')  
import sys  
name = 42  
  
  
def func():  
pass  
  
  
class glass:  
pass  
  
  
print("done loading...")

程序module\_test/module\_test3.py

import module1  
print(module1.sys)  
print(module1.name)  
print(module1.func)  
print(module1.glass)

输出为：

starting to load...

done loading...

<module 'sys' (built-in)>

42

<function func at 0x0000009F3EF0DAE8>

<class 'module1.glass'>

模块的删除

程序module\_test/module\_test4.py

import small  
print("module id", id(small)) # module id 202670467256  
print("delete module")  
del small  
# print(small.y) # Error，name 'small' is not defined  
  
import small # 不会重新执行导入  
print("module id", id(small)) # module id 202670467256

输出为：

x id in small.py 1347972576

y id in small.py 202671306056

module id 202670467256

delete module

module id 202670467256

从输出可以看出，通过del可以删除模块对象，del small不能再使用，可以通过import再次导入small，但并未重新执行导入small.py模块的操作。

在python中，del是解除变量对对象的引用，而并未删除对象。del small是解除了small变量对模块对象的引用，再次使用import small，又将small重新引用了模块对象，所以两次id(small)的结果是相同的。

如果在一个py文件中导入不同模块的同名函数，如果调用该函数，会发现后导入的覆盖了之前导入的。

例：程序udacity\_test/udacity\_test7.py