(1)Package 套件 就是一個目錄

(2)Module 模組 就是一個.py程式檔案

* 把一個package 想成一個目錄
  目錄底下有很多個檔案，包括一個特別的 "__init__.py" 檔案

* 把一個 module 想成一個檔案
 檔案裡頭可以定義 function，class，和 variable。

注意: 模組只會import一次，你必須Kernel Restart之後，才會重新import

In [ ]:

Module 模組 檔案

只要你建立了一個原始碼檔案 mymodule.py，你就建立了一個模組 mymodule，程式碼檔名就是模組名稱

import mymodule 陳述句會在　目前所在目錄下　尋找 mymodule.py，如果沒找到，則會試著尋找在 sys.path中遞迴地尋找 mymodule.py，如果還是沒有，則會引發 ImportError 例外。

Package 套件 目錄

假設現在你有一些 .py 檔案，別人同樣也有一堆 .py 檔案，你們的檔案現在得放在同一專案中，那麼檔案名稱衝突是有可能發生的，最好是為你們的 .py 檔案分別開設目錄。

使用 Python 時，你可以在開設的目錄中放個 __init__.py 檔案，這樣 Python 就會將這個目錄視為一個套件，而目錄名稱就是套件名稱。

使用 import pack.modu 陳述時，Python 會尋找 pack 目錄，看看裏頭是否有 __init__.py 檔案，然後看看目錄中是否有個 modu.py 檔案。

__init__.py 檔案空白也無所謂，實際上當中也可以寫些程式碼，用來執行這個套件中都會需要的初始工作，不過慣例上，除非你有真正不得已的理由，請保持 __init__.py 檔案空白。在找到模組後，實際上會執行其中頂層範疇中的程式碼，之後，模組中的變數、函式、類別等名稱，可以透過 pack.modu 來取得。

套件

一個專案有很多的模組(.py 檔案)，需要使用套件 (package)，將這些模組分門別類
建立套件就是建立一個目錄，並且在目錄中放一個 __init__.py 檔案，此時Python就會自動將這個目錄視為套件，套件名稱就是目錄名稱。

In [ ]:

模組中的變數、函式與類別，需透過以下方式取用

```
import xx
import xx as zz
from xx import yy
from xx import yy as zz
from xx.yy import zz
```

命名空間 (Namespace)

Python為模組提供命名空間的機制，讓函式名稱不會產生衝突，下面有寫法，習慣上比較推薦第一種，因為第一種除了保證不會與其他名稱衝突外，也可以讓程式讀起來更容易理解，使用的函式是出自哪一個模組中。

```
from x1 import y as z1
from x2 import y as z2
```

In [ ]:

初始化一次
Python 的模組在 import 時只會被載入一次，之後再使用import則不會有任何作用，但是也可以使用 imp.reload 的方式，重新載入。

模組的搜尋路徑

In [1]:
```
import sys
print(sys.path)
```

```
['/Users/clhuang/miniconda3/envs/py36/lib/python36.zip', '/Users/clhuang/miniconda3/envs/py36/lib/python3.6', '/Users/clhuang/miniconda3/envs/py36/lib/python3.6/lib-dynload', '', '/Users/clhuang/miniconda3/envs/py36/lib/python3.6/site-packages', '/Users/clhuang/miniconda3/envs/py36/lib/python3.6/site-packages/IPython/extensions', '/Users/clhuang/.ipython']
```

Python尋找某一模組 xxx.py 時，會依照 sys.path 所定義的路徑，依序搜尋
sys.path 定義的目錄列表如下:
1. 當前目錄下尋找 xxx.py
2. 由環境變數 PYTHONPATH 指定的目錄下尋找
3. 由Python的安裝路徑

In [ ]:

In [ ]:

當你import的時候，python只會在sys.path這個變量（一個list，你可以print出來看）裡面的路徑中找可能匹配的package和module。

而一個package跟一個普通文件夾的區別在於，package的文件夾中多了一個__init__.py文件。換句話說，如果你在某個文件夾中添加了一個__init__.py文件，則python就認為這個文件夾是一個package。

__init__.py文件可以是空的，它只是告訴python當前文件夾是一個package。當然，也可以在裡面添加一些代碼，這些代碼會在import這個包的時候運行。

所以，請確保你要import的文件所在的文件夾有__init__.py文件（除非它在sys.path中某個文件夾下）。

```
parent_package/
 __init__.py            <- EMPTY, NOT NECESSARY in Python 3.3+
 child_package/
      __init__.py       <- STILL REQUIRED if you want to run an initialization
script
      child1.py
      child2.py
      child3.py
```

請在當前目錄下新增一個mydogmodule.py，內容如下

```
class Dog:
    def __init__(self, name, age): #建構子
        self.name = name  #instance variabels實體變數 是public
        self.age = age
    def showMe(self):
        return "我的名字:"+self.name
    def __str__(self): #被print()列印時會執行此程式，等同於java的toString()
        return "姓名:%s, age:%d" %(self.name, self.age)

def hello():
    print('hello()方法來自mymodule.py--你懂得模組概念了!')
```

In [ ]:

In [50]:
```
from mydogmodule import Dog
```

In [14]:
```
billy = Dog("Billy", 5)
```

In [15]:
```
billy.showMe()
```

Out[15]:
```
'姓名:Billy, age:5(mydogmodule.py)'
```

In [ ]:
```

```

In [ ]:
```
from mydogmodule import hello
```

In [17]:
```
hello()
```

```
hello()方法來自mymodule.py--你懂得模組概念了!
```

In [ ]:
```

```

我的套件(目錄)、模組(程式檔.py)、類別、方法

```
目錄結構:
mypackage/
  __init__.py
  mymodule.py


__init__.py檔案之內容:

print('當你import這個目錄時，__init__.py被呼叫了')

# from mymodule import Dog # 若缺少一"點"不能運作，相對路徑引入方式relative import
from .mymodule import Dog

dog = Dog('doggy',5)
dogshow = dog.showMe


mymodule.py檔案之內容:
class Dog:
    def __init__(self, name, age): #建構子
        self.name = name  #instance variabels實體變數 是public
        self.age = age
    def showMe(self):
        return "我的名字:"+self.name
    def __str__(self): #被print()列印時會執行此程式，等同於java的toString()
        return "姓名:%s, age:%d" %(self.name, self.age)

def hello():
    print('hello()方法來自mymodule.py--你懂得模組概念了!')


# 相對路徑引入方式relative import
from .mymodule import Dog

dot表示甚麼意思?
```

```
The . is a shortcut that tells it search in current package before rest of the
PYTHONPATH. So, if a same-named module exists somewhere else in your
PYTHONPATH, it won't be loaded.


A single leading dot indicates a relative import, starting with the current
package. Two or more leading dots give a relative import to the parent(s) of
the current package, one level per dot after the first.

from .moduleY import spam
from .moduleY import spam as ham
from . import moduleY
from ..subpackage1 import moduleY
from ..subpackage2.moduleZ import eggs
from ..moduleA import foo
from ...package import bar
from ...sys import path

Reference:
https://www.python.org/dev/peps/pep-0328/#guido-s-decision
```

In [ ]:

In [4]:
```
from mypackage.mymodule import Dog
```

當你import這個目錄時，__init__.py被呼叫了

In [5]:
```
billy = Dog("Billy",5)
billy.showMe()
```

Out[5]: '我的名字:Billy'

In [6]:
```
Dog
```

Out[6]: mypackage.mymodule.Dog

In [7]:
```
from mypackage.mymodule import hello
```

In [8]:
```
hello()
```

hello()方法來自mymodule.py--你懂得模組概念了!

In [9]:
```
hello
```

Out[9]: <function mypackage.mymodule.hello()>

In [ ]:

In [ ]:

"init.py" 幫你做好一些方便的初始化工作

"__init__.py" 幫你先準備好了一些變數或物件，例如：已經宣告初始化一個dog物件給你使用

就像是物件導向程式設計的建構子Constructor

In [47]:
```
from mypackage import dog
```

In [40]:
```
# import mypackage.dog  #這樣寫不行!
```

In [41]:
```
dog
```

Out[41]:
```
<mypackage.mymodule.Dog at 0x10f736320>
```

In [48]:
```
dog.showMe()
```

Out[48]:
```
'我的名字:doggy'
```

In [ ]:

通常我們直接匯入類別 方法，有時候你會只是把模組名稱匯入，方式如下

In [10]:
```
import mypackage
```

In [11]:
```
mypackage
```

Out[11]:
```
<module 'mypackage' from 'G:\\我的雲端硬碟\\10-05-課程\\20-30-Python與R入門與scikitlearn分類與預測程式碼\\10-30-Python簡易入門\\mypackage\\__init__.py'>
```

In [ ]:

In [12]:
```
mypackage.dog
```

Out[12]:
```
<mypackage.mymodule.Dog at 0x21a2ecb9160>
```

In [13]:
```
mypackage.dog.showMe()
```

Out[13]:
```
'我的名字:doggy'
```

In [ ]:

In [14]:
```
mypackage.dogshow
```

Out[14]:
```
<bound method Dog.showMe of <mypackage.mymodule.Dog object at 0x0000021A2ECB9160>>
```

In [15]:
```
mypackage.dogshow()
```

Out[15]:
```
'我的名字:doggy'
```

In [26]:
```
show = mypackage.dogshow
```

In [27]:
```
show
```

Out[27]:
```
<bound method Dog.showMe of <mypackage.mymodule.Dog object at 0x10f736320>>
```

In [28]:
```
show()
```

Out[28]:
```
'我的名字:doggy'
```

In [ ]:

In [ ]:

In [29]:
```
d = mypackage.dog
```

In [30]:
```
d.showMe()
```

Out[30]:
```
'我的名字:doggy'
```

In [ ]:

參考資料:
http://www.codedata.com.tw/python/python-tutorial-the-2nd-class-3-function-module-class-package

In [ ]: