Chapter 06

使用類別

Python的類別機制是C++ 以及Modula-3的綜合體，當我們在使用Python的類別時並不用去宣告該類別的型態，也不用去宣告這個類別是否為public或private，Python所有的類別與其包含的成員都是public的，對於類別（Class）來說最重要的一些特性在Python程式語言裡面都有完全的保留，如最重要的單一繼承與多重繼承，一個衍生／子類別（Derived class）可以覆載（Override）其所有基礎類別（base class）的任何方法（method），方法（method）也可以呼叫一個基礎類別（base class）的同名方法。

Python的類別裡，所有的函式成員（member functions）都是virtual的，這裡也沒有所謂的建構函數（Constructors）或是解構函數（Destructors）的存在。

Python的類別宣告、建立一個實例（Instance）及繼承都是很快速及簡單，在您閱讀完本章節後您將學會底下知識：

- 建立一個類別。
- 進階的物件導向技巧使用。
- 了解類別變數與實例變數的差異。
- 如何動態的新增與刪除類別實例的方法成員。
- 如何動態的新增與刪除類別實例的屬性。

Python 的字串也是物件

當我們建立一個字串變數，其實它就是字串物件，從**dir()**函數可以看到它繼承了許多物件的特性，**dir()**函數列出我們定義的simple字串物件可以使用的方法（method）成員。

```
>>> simple = "an object"
>>> dir(simple)
['__add__', '__class__', '__contains__', '__delattr__',
'__doc__', '__eq__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__getslice__', '__gt__', '__hash__',
'__init__', '__le__', '__len__', '__lt__', '__mod__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
```

```
'__rmod__', '__rmul__', '__setattr__', '__str__', 'capitalize',
'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs',
'find', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith','strip','swapcase','title','translate','upper',
'zfill']
```

6.1 定義類別

在Python程式語言裡要定義一個類別必須使用**class**敘述句，然後在敘述句後面接著類別的名稱，並不用去定義是否為public或private等，也不用去定義資料型態，如下定義：

class 敘述句用法：

```
class ClassName:
    <statements>
<next statements>
```

如底下範例我們宣告一個簡單的物件，名稱為myFirstObj，在物件裡面跟函數一樣都可以使用備註說明，我們可以使用**__doc__**屬性（attribute）來呼叫該物件的備註說明，如果我們直接呼叫物件的名稱將會得到該物件的名稱與識別碼，如<class __main__.myFirstObj at 0x01FF7E10>。

```
>>> class myFirstObj:
    """ 建立一個簡單的物件。
    在這裡的是物件的備註說明，跟函數一樣。
    """

>>> myFirstObj
<class __main__.myFirstObj at 0x01FF7E10>
```

```
>>> print(myFirstObj.__doc__)
 建立一個簡單的物件。
   在這裡的是物件的備註說明，跟函數一樣。
```

6.1.1 使用類別

定義好類別後我們必須建立一個物件的實例（Instance）才能使用物件，如下範例建立一個物件的實例，名稱為“o”，我們可以直接呼叫“o”或是利用**type()**函數來知道它是物件的實例。

```
>>> o = myFirstObj()
>>> o
<__main__.myFirstObj instance at 0x01FF8F58>

>>> type(o)
<type 'instance'>
```

Python 3.0差異在於instance變成object。

```
>>> o
<__main__.myFirstObj object at 0x01019B90>
>>> type(o)
<class '__main__.myFirstObj'>
```

我們可以透過dir()函數來了解目前在物件內的方法（method）成員。

```
>>> dir(o)
['__doc__', '__module__']
```

接著我們可以嘗試宣告『o.mylist = []』，表示定義一個mylist成員，然後『[]』符號代表這個mylist成員是序列型態，定義完後再嘗試使用**dir()**函數可以發現mylist名稱被加入清單中。

```
>>> o.mylist = []
>>> dir(o)
['__doc__', '__module__', 'mylist']
```

接著我們來看剛才所宣告的mylist有什麼作用，如Figure 6-1畫面，沒錯剛才的mylist繼承了序列物件的所有（method）方法成員。

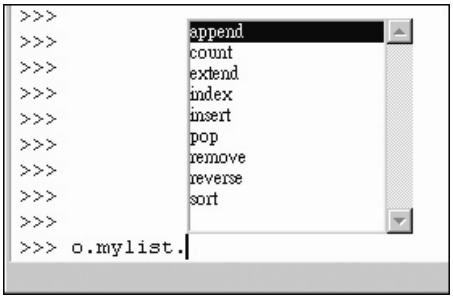

Figure 6-1 mylist 成員可以使用的方法（method）成員。

接著我們可以嘗試使用mylist裡面的方法，底下範例是使用extend()函數：

```
>>> o.mylist.extend(["a", "b", "c"])
>>> o.mylist
['a', 'b', 'c']
```

6.1.2 定義類別內的方法（method）

類別內的方法（method）宣告方式跟函數一樣，您可以想像類別方法就像是將函數放進類別裡面，差異性在於函數的引數部份必須要加入**self**敘述句，**self**敘述句就像其他程式語言裡面的**this**。

底下範例定義了一個方法成員，名稱為func，在func雖然定義了**self**與*name*引數，但是下面在呼叫該方法只需要傳值給*name*引數就可以，**self**引數是不用理會的，我們也無法傳值給它，但是在定義類別方法成員時是一定要使用的。

程式碼：ch601.py

```
#!/usr/bin/env python
#coding=utf-8
```

```
# 定義 myFirstObj 類別
class myFirstObj:
    """ 建立一個簡單的物件。
    在這裡的是物件的備註說明，跟函數一樣。
    """
    count = 100
    def func(self, name):
        return "Hello", name

# 建立 instance
o = myFirstObj()

# 呼叫類別內的 attribute
print(o.count)

# 呼叫類別內的 method
print(o.func("s"))

# 使用 type 查看 o.func 的型態
print(type(o.func))
```

程式輸出結果為：

```
100
('Hello', 's')
<type 'method'>
```

如果在定義func函數時沒有給予self敘述句則會發生**TypeError**錯誤訊息，如下：

```
Traceback (most recent call last):
  File "D: \code\ch601.py", line 20, in <module>
    print o.func()
TypeError: func() takes no arguments (1 given)
```

6.1.3 定義類別內的__init__()方法

Python程式語言裡的類別有一個特別方法（method）稱為__init__()，當建立一個實例（Instance）的同時類別就會去執行這個函數，其實這個就像其他程式語言的類別所定義的建構函數（Constructors），在範例ch602裡當宣告n = operator(2)後就會進入operator類別裡的__init__()方法，如下：

```
def __init__(self, x=None):
    self.x = x
    self.total = 100
    print("run __init__()")
```

另外在這類別裡有一個實例的全域變數為self.total = 100，當執行plus()方法與reducr()方法時就會針對這個全域變數進行遞增與遞減。

程式碼：ch602.py

```
#!/usr/bin/env python
#coding=utf-8

# 定義 operator 類別
class operator:
    # 這個特殊的__init__()函數會在建立實例 Instance 時就會執行
    def __init__(self, x=None):
        #在這裡所宣告的 attributes 前面如果加上 self 就表示全域變數
        self.x = x
        self.total = 100
        print("run __init__()")

    # 這個函數會將 self.total 變數遞增
    def plus(self):
        self.total += self.x
        return self.total

    # 這個函數會將 self.total 變數遞增
```

```
    def reduce(self):
        self.total -= self.x
        return self.total

# 建立 Instance, 傳入 2 表示一次遞增 2 或遞減 2
n = operator(2)
print("run plus method", n.plus())
print("run reduce method", n.reduce())
print("run reduce method", n.reduce())
```

程式輸出結果為：

```
run __init__()
run plus method 102
run reduce method 100
run reduce method 98
```

6.1.4 定義類別內的屬性（attributes）

在類別裡面有所謂的類別變數（class variable）與實例變數（Instance variable），通常定義在__init__()建構函數（Constructors）裡面的稱為實例變數，而在類別的__init__()建構函數與方法成員之外的變數稱為類別變數，實例變數（Instance variable）除了必須使用**self**敘述句外也可以使用該類別的名稱，假設類別的名稱為cos，然後變數名稱為x，那麼實例全域變數可以設定為self.x或是cos.x，如下範例定義類別稱為operator，所以實例全域變數名稱也可以寫成operator.x與operator.total。

程式碼：ch603.py

```
#!/usr/bin/env python
#coding=utf-8
```

```
# 定義 operator 類別
class operator:
    def __init__(self, x=None):
        operator.x = x
        operator.total = 100

    # 這個函數會將 self.total 變數遞增
    def plus(self):
        operator.total += operator.x
        return "increase,", operator.total

    # 這個函數會將 self.total 變數遞增
    def reduce(self):
        operator.total -= operator.x
        return "decrease, ", operator.total

# 建立 Instance, 傳入 2 表示一次遞增 2 或遞減 2
n = operator(2)
print(n.plus())
print(n.reduce())
print(n.reduce())
print(n.reduce())
```

程式輸出結果為：

```
('increase,', 102)
('decrease, ', 100)
('decrease, ', 98)
('decrease, ', 96)
```

6.1.5 類別變數（class variable）與實例變數

類別變數（class variable）與實例變數（Instance variable）定義如下範例，在範例結果中我們可以清楚的知道這兩種變數之間的差異，類別變數會在建立的實例（Instance）之間共存如範例中的a與b，而實例變數並不會共存，如範例中的c與d。

程式碼：ch604.py

```
#!/usr/bin/env python
#coding=utf-8

class lists:
    # 類別變數
    class_variable = []
    def __init__(self):
        # self.keywords 是實例變數
        self.instance_variable = []
# 建立 instance a 與 b
a = lists()
b = lists()

# 給類別變數值
a.class_variable.extend([1,2,3,4,5])

# 呼叫 instance a 與 b
print("call a", a.class_variable)
print("call b", b.class_variable)

# 建立 instance c 與 d
c = lists()
d = lists()
```

```
# 給實例變數值
c.instance_variable.extend(["a","b","c","d","e"])

# 呼叫 instance c 與 d
print("call c", c.instance_variable)
print("call d", d.instance_variable)
```

程式輸出結果為：

```
call a [1, 2, 3, 4, 5]
call b [1, 2, 3, 4, 5]
call c ['a', 'b', 'c', 'd', 'e']
call d []
```

6.2 動態新增類別實例方法成員

當我們在建立一個類別物件時都必須先定義好該類別的屬性、方法成員等，但是您有沒有想過，當我們定義完的物件如果在執行期間（run-time）能不能也能動態的加入新的方法成員或是屬性到物件裡？就好像當我們定義好的物件實例在執行期間時發現本身所定義的方法成員不足夠時，然後看到其他的物件有一個不錯的方法成員，那麼我們是不是可以讓自己的類別物件去學習該物件內的某個方法成員。

如範例程式碼ch605，一開始我們定義了類別A，然後也定義一個方法成員，名稱為sort，在執行期間遇到x序列，我們想要對這個序列排序，但是在我們所定義的類別A裡面的sort()方法函數裡面並沒有定義任何程式，接著我們知道在python本身有一個內置的函數稱為 sorted()這個函數是在做排序的，所以我們決定學習它，接著重新嘗試呼叫a.sort()，我們發現原本的sort學習了sorted()的功能，另外，我們發現所定義的類別A並沒有總合運算的功能，但是我們知道在Python本身也有一個內置的sum()函數，雖然我們之前定義的類別A裡面沒有sum這個名稱，可以學習嗎？答案是可以的。

程式碼：ch605.py

```
#!/usr/bin/env python
#coding=utf-8

# 定義一個類別, 名稱為 A
class A:
    # 定義一個方法成員, 名稱為 sort
    def sort(self):
        pass

# 建立一個實例(instance)
a = A()

# 呼叫方法成員 a.sort()
print(a.sort())

"""
在執行期間遇到 x 序列, 我們想要對這個序列排序,
但是在我們所定義的類別 A 裡面的 sort()方法函數
裡面並沒有定義任何程式.
"""
x = [5,2,3,1,4]

"""
接著我們知道在 python 本身有一個內置的函數稱為 sorted()
這個函數是在做排序的, 所以我們決定學習它.
"""
a.sort = sorted

"""
接著重新嘗試呼叫 a.sort(), 我們發現原本的 sort 學習了 sorted()的功能
"""
print(a.sort(x))
```

```
"""
另外，我們發現所定義的類別 A 並沒有總合運算的功能，
但是我們知道在 Python 本身也有一個內置的 sum()函數，
但是我們之前定義的類別 A 裡面沒有 sum 這個名稱，可以學習嗎?
答案是可以的.
"""
a.sum = sum

"""
學習 sum()函數後我們進行呼叫
"""
print(a.sum(x))
```

程式輸出結果為：

```
None
[1, 2, 3, 4, 5]
15
```

我們針對“動態新稱方法”來做更詳細的介紹，底下可以看到定義了一個類別B，然後這個類別B本身並沒有NewMethod方法成員，後來我們透過**new**模組內的**instancemethod()**函數來新增NewMethod方法成員到實例b裡面，另外Python 3.0已經移除new模組。

```
>>> class B:
...     def foo(self):
...         print "foo"
...
>>> b = B()
>>> def NewMethod(self):
...         print("New method!")
...
>>> import new
```

```
>>> b.NewMethod = new.instancemethod(NewMethod, b, B)
>>> b.NewMethod
<bound method B.NewMethod of <__main__.B instance at 0x01FE9760>>
>>> b.NewMethod()
New method!
```

6.3 動態刪除類別實例方法成員

這章節介紹如何將動態新增的類別實例刪除，我們可以使用**del**敘述句來刪除，如下範例。

定義類別，名稱為B。

```
>>> class B:
...     def __init__(self):
...         self.total = 10
...     def foo(self):
...         print("foo")
...
```

定義NewMethod函數。

```
>>> def NewMethod(self):
...         print("New method!")
```

建立實例b。

```
>>> b = B()
```

將NewMethod函數加入實例b

```
>>> import new
>>> b.NewMethod = new.instancemethod(NewMethod, b, B)
>>> b.NewMethod()
New method!
```

使用del敘述句刪除NewMethod函數。

```
>>> del b.NewMethod
```

刪除後再呼叫時就會出現AttributeError錯誤訊息，因為實例內已經沒有NewMethod函數。

```
>>> b.NewMethod()
Traceback (most recent call last):
  File "<input>", line 1, in <module>
AttributeError: B instance has no attribute 'NewMethod'
```

實例本身的變數也是可以刪除的，下面我們刪除了原本在類別內定義的b.total實例變數。

```
>>> del b.total
```

重新呼叫就會出現AttributeError錯誤訊息，因為實例b裡面已經沒有該變數名稱。

```
>>> print(b.total)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
AttributeError: B instance has no attribute 'total'
```

那麼原本類別裡面定義的foo()方法成員可以刪除嗎？答案是不行的，會發生SyntaxError錯誤訊息。

```
>>> del b.foo()
  File "<input>", line 1
SyntaxError: can't delete function call (<input>, line 1)
>>>
```

摘要

- 當我們在使用 Python 的類別時並不用去宣告該類別的型態，也不用去宣告這個類別是否為 public 或 private，Python 所有的類別與其包含的成員都是 public 的。
- 在 Python 程式語言裡要定義一個類別必須使用 class 敘述句。
- 我們可以使用__doc__屬性（attribute）來呼叫該物件的備註說明。
- 定義好類別後我們必須建立一個物件的實例（Instance）才能使用物件。
- 類別內的方法（method）宣告方式跟函數一樣，您可以想像類別方法就像是將函數放進類別裡面，差異性在於函數的引數部份必須要加入 self 敘述句，self 敘述句就像其他程式語言裡面的 this。
- Python 程式語言裡的類別有一個特別方法（method）稱為__init__()，當建立一個實例（Instance）的同時類別就會去執行這個函數。
- 在類別裡面有所謂的類別變數（class variable）與實例變數（Instance variable），通常定義在__init__()建構函數（Constructors）裡面的稱為實例變數，而在類別的__init__()建構函數與方法成員之外的變數稱為類別變數。
- 類別變數會在建立的實例（Instance）之間共存，而實例變數並不會共存。
- 我們可以透過 new 模組內的 instancemethod()函數來新增方法成員到實例類別裡面。
- 使用 del 敘述句來刪除實例本身的變數，但是本身已定義好的方法成員是不能刪除的。

練習

6.1 底下（1）為學生類別，請使用new模組與動態新稱方法成員的觀念改變printDetails函數，並且新增一個email屬性到實例類別student1裡，並輸出printDetails函數結果與下面一樣：

```
>>> student1.printDetails("abc@gmail.com")
姓名: John
課程: ['C201']
email: abc@gmail.com
```

（1）

```
>>> class Student:
...     """ 學生類別 """
...
...     def __init__(self, name ="Huang", courses =[]):
...         self.name = name
...         self.courses = courses
...         print("建立實例: " + name)
...
...     def printDetails(self):
...         print("姓名: ", self.name)
...         print("課程: ", self.courses)
...
...     def enroll(self, course):
...         self.courses.append(course)
...
>>> student1 = Student("John", ["C201"])
```

6.2 請建立一個類別，並定義一個全域變數命名為value，接著再定義一個方法成員命名為pow(x)，並將value值做x次方。