

Python程式設計

林奇賦 daky1983@gmail.com

Outline

- 類別

類別

- 類別 (class) 為物件 (object) 設計的模板，Python 裡所有東西都是物件，凡是物件都有屬性 (attribute) 跟方法 (method)。所謂的屬性雷同變數 (variable)，專屬於物件，方法類似函數 (function)，同樣專屬於物件。

定義類別

- 定義類別使用關鍵字 (keyword) **class** ，形式如下

```
class Class_Name:  
    #內容
```

類別

```
class Demo:  
    i = 100  
    def hello(self):  
        print("hello")
```

```
help(Demo)  
print(dir(Demo))
```

內建函數 `help()` 會顯示物件（包括類別）的資訊，`dir()` 則會把物件（包括類別）的所有屬性與方法以串列 (list) 列出

建立物件

```
class Demo:  
    i = 100  
    def hello(self):  
        print("hello")
```

```
d = Demo()  
print(type(Demo))  
print(type(d))  
print(d.i)  
d.hello()
```

Demo() 就是 Demo 類別建立物件的建構子 (constructor) ，這裡利用指派運算子將建構子所建立的實體物件 (instance) 給變數 d ，於是 d 就具有 Demo 型態的物件。

類別 `__init__()`

- 利用建構子 (constructor) 建立的物件被稱為實體 (instance)，實際上建立物件的整個過程是執行 `__init__()` 方法 (method)。自行定義的類別會有預先定義好的 `__init__()`，我們也可以改寫 (override) 成我們自己需要的。
- 改寫方式就是再定義一次，方法的定義與函數 (function) 類似，兩者同樣使用關鍵字 (keyword) **def**

類別 `__init__()`

```
class Demo:
    def __init__(self):
        self.name = "Python"
    def hello(self):
        print("hello",self.name)
```

```
d = Demo()
print(type(Demo))
print(type(d))
print("d.name: %s"%d.name)
d.hello()
```

凡是實體的方法都需要一個特別的參數 -- `self`，`self` 是個預設的保留字 (reserved word)，所指的是實體本身自己，在 `__init__()` 所定義的實體屬性 (attribute) 都需要額外加上 `self`，如第 3 行的 `self.name`。

類別 `__init__()`

```
class Demo:
    def __init__(self):
        self.name = "Python"
    def hello(self):
        print("hello",self.name)
```

```
d = Demo()
print(type(Demo))
print(type(d))
print("d.name: %s"%d.name)
d.hello()
```

凡是實體的方法都需要一個特別的參數 -- `self`，`self` 是個預設的保留字 (reserved word)，所指的是實體本身自己，在 `__init__()` 所定義的實體屬性 (attribute) 都需要額外加上 `self`，如第 3 行的 `self.name`。

設定類別 `__init__()` 參數

```
class Demo:
    def __init__(self,name):
        self.name = name
    def hello(self):
        print("hello",self.name)
```

```
d = Demo("Tom")
print(type(Demo))
print(type(d))
print("d.name: %s"%d.name)
d.hello()
```

凡是實體的方法都需要一個特別的參數 -- `self`，`self` 是個預設的保留字 (reserved word)，所指的是實體本身自己，在 `__init__()` 所定義的實體屬性 (attribute) 都需要額外加上 `self`，如第 3 行的 `self.name`。

類別 `__doc__`

- 類別 (class) 有 `__doc__` 屬性 (attribute) ，這是三引號字串定義的文字，屬於類別的說明文件。

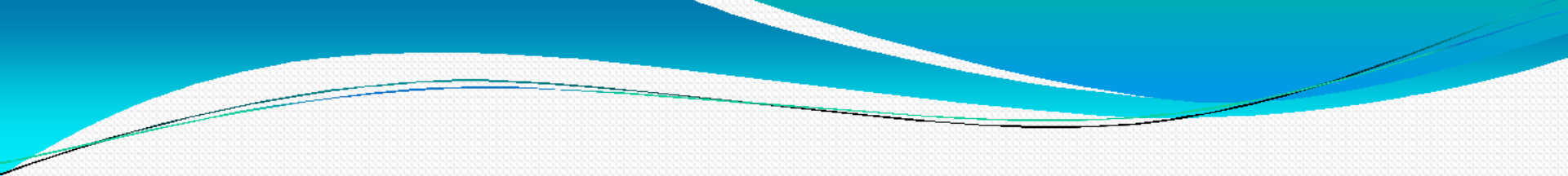
```
class Demo:
    """
    Demo Document:
    hello python
    """
    def __init__(self,name):
        self.name = name
    def hello(self):
        print("hello",self.name)
```

```
d = Demo("Tom")
print(help(d))
```

類別屬性與實體屬性

```
class Demo:
    x=0
    """
    Demo Document:
    hello python
    """
    def __init__(self,i):
        self.i = i
        Demo.x += 1
    def hello(self):
        print("hello",self.i)
```

x 為類別屬性， self.i 則為實體屬性。



```
print("There are", Demo.x, "instances.")
a = Demo(1122)
a.hello()
print("a.x =", a.x)
b = Demo(3344)
b.hello()
print("b.x =", b.x)
c = Demo(5566)
c.hello()
print("c.x =", c.x)
d = Demo(7788)
d.hello()
print("d.x =", d.x)
e = Demo(9900)
e.hello()
print("e.x =", e.x)
print("After all, there are", Demo.x, "instances.")
```

類別屬性與實體屬性

```
class Demo:
```

```
    i=0
```

```
    """
```

```
    Demo Document:
```

```
        hello python
```

```
    """
```

```
    def __init__(self,i):
```

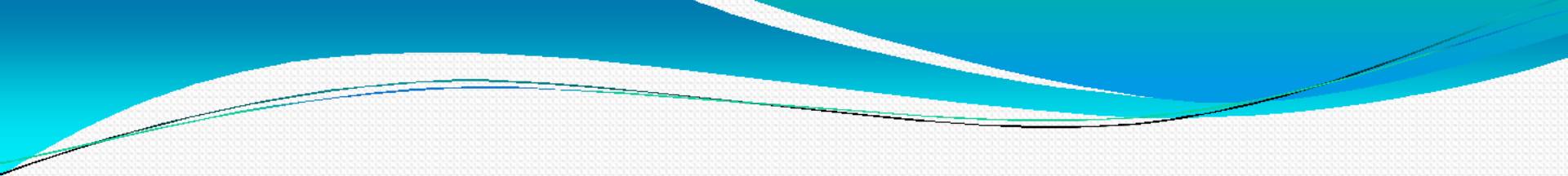
```
        self.i = i
```

```
        Demo.i += 1
```

```
    def hello(self):
```

```
        print("hello",self.i)
```

若是類別屬性與實體屬性的識別字相同，
實體物件只能存取實體屬性



```
print("There are", Demo.i, "instances.")
a = Demo(1122)
a.hello()
print("a.i =", a.i)
b = Demo(3344)
b.hello()
print("b.i =", b.i)
c = Demo(5566)
c.hello()
print("c.i =", c.i)
d = Demo(7788)
d.hello()
print("d.i =", d.i)
e = Demo(9900)
e.hello()
print("e.i =", e.i)
print("After all, there are", Demo.i, "instances.")
```



類別課堂練習

- 試寫一個名為student的類別
 - 其中屬性包含:
 - name, gender, grades
 - 函數包含:
 - avg: 回傳grades list的平均值
 - add(grade): 新增成績到grades list中
 - fcount: 回傳不及格(<60)的總數

- 將課程網頁<http://140.112.31.82/wordpress/?p=216> 的程式碼加入
- 分別將每個學生的成績平均、不及格的的數目印出
- 於類別外寫一個top的函數：
 - 傳入值為學生物件的序列
 - 將平均分數最高的學生回傳

屬性封裝

- Python 類別 (class) 的屬性 (attribute) 權限預設是公開的，因此類別以外的地方也可以存取，例如

```
class Demo:
    x = 0

    def __init__(self, i):
        self.i = i
        Demo.x += 1

    def hello(self):
        print("hello", self.i)
```

```
a = Demo("Tom")
a.hello()
print("hello", a.i)
print()
print("a.i =", a.i)
print("Demo.x =", Demo.x)
```

類別方法

- 類別方法需要一個特別的參數 (parameter) ，習慣上使用 `cls` ，這與實體方法的 `self` 類似，不同的是 `cls` 用來存取類別的屬性 (attribute)

```
@classmethod  
def function_name(cls):  
    return cls.__變數
```



```
class Demo:
```

```
    __x = 0
```

```
    def __init__(self, i):
```

```
        self.__i = i
```

```
        Demo.__x += 1
```

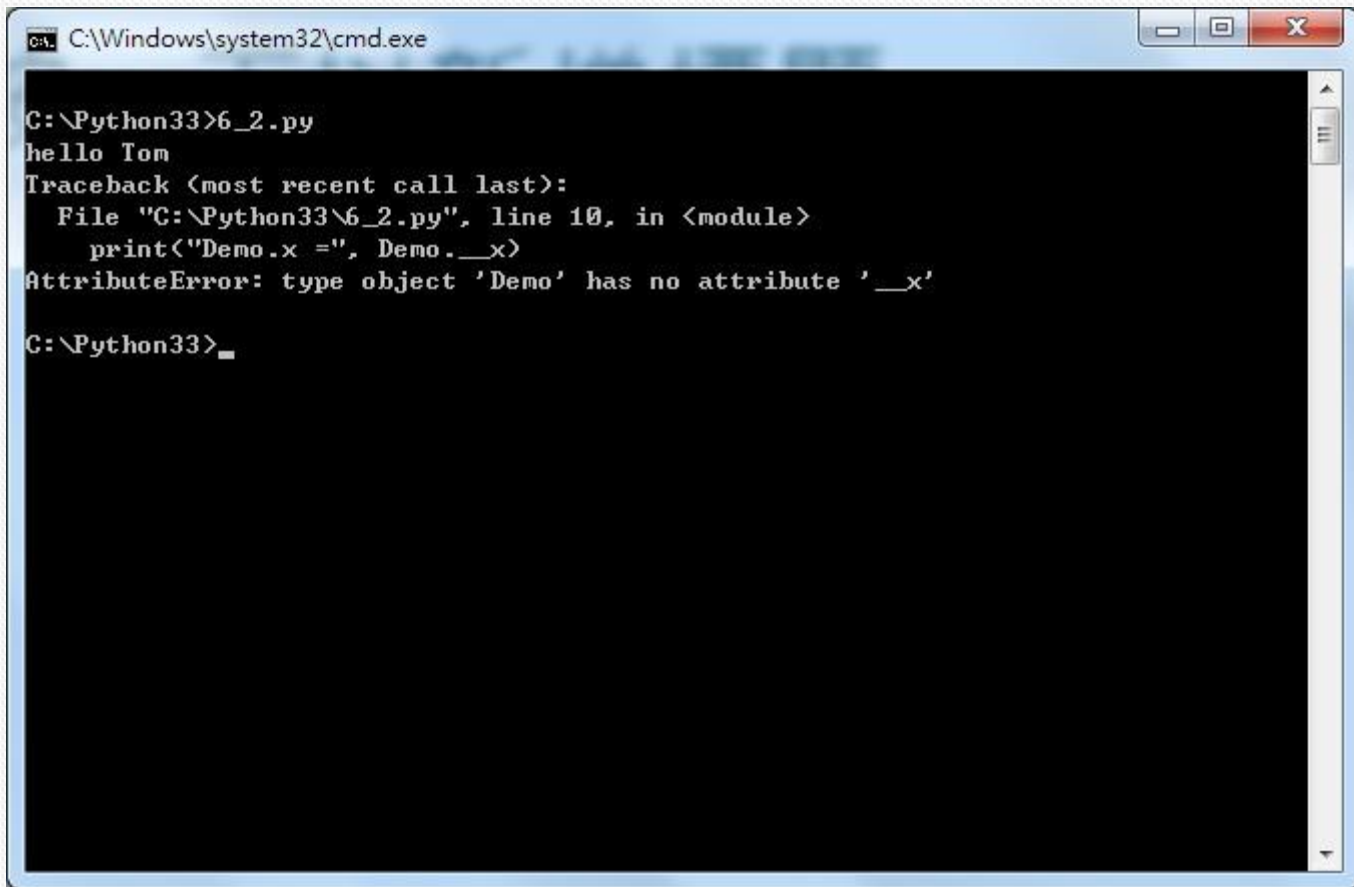
```
    def hello(self):
```

```
        print("hello", self.__i)
```

```
a = Demo("Tom")
```

```
a.hello()
```

```
print("Demo.x =", Demo.__x)
```

```
C:\Windows\system32\cmd.exe

C:\Python33>6_2.py
hello Tom
Traceback (most recent call last):
  File "C:\Python33\6_2.py", line 10, in <module>
    print("Demo.x =", Demo.__x)
AttributeError: type object 'Demo' has no attribute '__x'

C:\Python33>_
```

```
class Demo:
    __x = 0
    def __init__(self, i):
        self.__i = i
        Demo.__x += 1
    def hello(self):
        print("hello", self.__i)

    @classmethod
    def getX(cls):
        return cls.__x
```

```
a = Demo("Tom")
a.hello()
print("Demo.x =", Demo.getX())
```

方法預設也都是公開的，若要定義私有的方法，也就是指能在類別內呼叫的方法，同樣在方法識別字名稱前加上連續兩個底線符號，這樣的方法就變成私有的。

類別繼承

- 若定義了很多類別 (class)，這些類別中又具有相當多相同的屬性 (attribute) 或方法 (method) 定義，這時候，可利用 Python 的繼承 (inheritance) 機制，將共通的屬性及方法提取出來，另行定義父類別 (superclass)，然後將原本提出共通屬性及方法改為繼承 (inherit) 父類別的子類別 (subclass)。

類別繼承

- 繼承的格式如下

```
class SubDemo(Demo):  
    #dosomething
```

- 這是從 **SubDemo** 類別去繼承 **Demo**，注意類別名稱後的小括弧中註明父類別。

```
class Demo:
```

```
    __x = 0
```

```
    def __init__(self, i):
```

```
        self.__i = i
```

```
        Demo.__x += 1
```

```
    def hello(self):
```

```
        print("hello", self.__i)
```

```
    @classmethod
```

```
    def getX(cls):
```

```
        return cls.__x
```

```
    @classmethod
```

```
    def add(cls):
```

```
        Demo.__x +=1
```



```
class subDemo(Demo):
```

```
    pass
```

```
a = Demo("Tom")
```

```
a.hello()
```

```
b = subDemo("John")
```

```
b.hello()
```

```
print("Demo.x =", Demo.getX())
```

內建函數

- 內建函數 (function) **isinstance()** 可以判斷某一個物件是否為某一個類別所建構的實體 (instance)，若真則回傳 **True**，否則回傳 **False**。
- 另一個內建函數 **issubclass()** 則可以判斷某一個類別是否為另一個類別的子類別，同樣的，若真則回傳 **True**，否則回傳 **False**。


```
a = Demo("Tom")  
b = subDemo("John")
```

#instance

```
print(isinstance(a, Demo))  
print(isinstance(a, subDemo))  
print(isinstance(b, Demo))  
print(isinstance(b, subDemo))
```

#issubclass

```
print(issubclass(subDemo, Demo))  
print(issubclass(Demo, subDemo))
```

變數 (variable) b 雖然是由 subDemo 建立的，但是 b 也會是 Demo 的實體，這是由於物件實體的建構過程中，會先建立父類別的部份，因此也會建立屬於 b 的父類別物件實體，使 b 得以運用父類別的屬性及方法。

子類別方法改寫



- 子類別 (subclass) 可依本身特性設定自己的屬性 (attribute) 與方法 (method)，也會從父類別 (superclass) 繼承 (inherit) 屬性與方法。一般來說，沒有設定成私有的屬性及方法都會被繼承，子類別可由父類別公開的方法存取父類別私有的屬性。
- 子類別也可依需要改寫 (override) 父類別的方法，這是說子類別需要用到與父類別具有相同名稱的方法，但是子類別需要的功能有所修改、擴充或增加，因此當子類別裡頭定義與父類別相同名稱的方法時，就會改寫父類別的方法。經過改寫，子類別的方法完全屬於子類別所有。



```
class Demo:
```

```
    __x = 0
```

```
    def __init__(self, i):
```

```
        self.__i = i
```

```
        Demo.__x += 1
```

```
    def hello(self):
```

```
        print("hello", self.__i)
```

```
    @classmethod
```

```
    def getX(cls):
```

```
        return cls.__x
```

```
    @classmethod
```

```
    def add(cls):
```

```
        Demo.__x += 1
```

```
class subDemo(Demo):
```

```
    def __init__(self, i, j):
```

```
        self.__i = i
```

```
        self.__j = j
```

```
    def hello(self):
```


```
        print("hello", self.__i, self.__j)
```

```
a = Demo("Tom")  
a.hello()  
b = subDemo("John","Mary")  
b.hello()  
print("Demo.x =", Demo.getX())
```

- Demo 為父類別，定義四個方法，SubDemo 為子類別，改寫 Demo 的兩個方法，包括 `__init__()` 與 `hello()`。

super()

- 利用內建函數 (function) `super()`，呼叫 (call) 父類別的方法

```
class subDemo(Demo):  
    def __init__(self, i, j="guest"):  
        self.__i = i  
        self.__j = j  
    def hello(self):  
        print("hello", self.__i, self.__j)  
    def superHello(self):  
        super().__init__(self.__i)   
        super().hello()
```

類別多重繼承

- 設計類別 (class) 時，父類別 (superclass) 可以有多個，這是說子類別 (subclass) 能夠繼承 (inherit) 多個父類別，使子類別可以有各種特性。
- 這裡須注意一點，當子類別繼承 (inheritance) 超過一個來源的時候，會以寫在最左邊的父類別優先繼承，這是說，多個父類別如果有相同名稱的屬性 (attribute) 與方法 (method)，例如 `__init__()`、`__str__()` 等，就會以最左邊的父類別優先。

#多重繼承

class Demo:

 __x = 0

 def __init__(self, i):

 self.__i = i

 Demo.__x += 1

 def hello(self):

 print("hello", self.__i)

 @classmethod

 def getX(cls):

 return cls.__x

 @classmethod

 def add(cls):

 Demo.__x += 1

class Demo2:

 def __init__(self, i):

 self.__i = i

 def reverseString(self, string):

 reverse=""

 for i in range(len(string)-1, -1, -1):

 reverse += string[i]

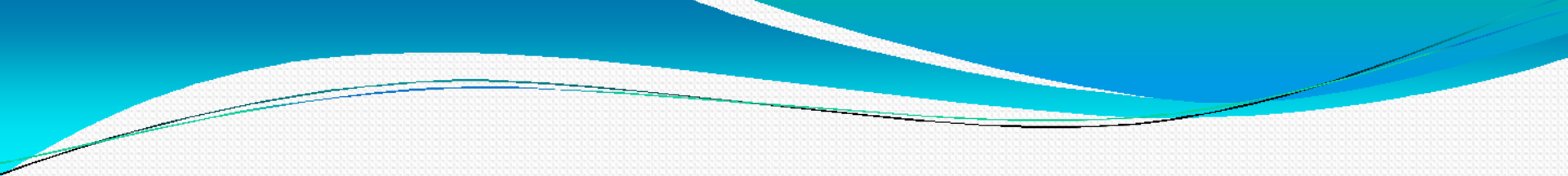
 return reverse

```
class subDemo(Demo2,Demo):
    def __init__(self, i, j="guest"):
        super().__init__(i)
        self.__i = i
        self.__j = j
    def hello(self):
        print("hello", self.__i,self.__j)
    def superHello(self):
        super().__init__(self.__j)
        super().hello()
```

```
a = subDemo("Tom")
print(a.reverseString("Tom"))
print("Demo.x =", Demo.getX())
```

類別 `__del__()`

- 建構子 (constructor) 用來建立物件 (object)，當物件不需要被使用時，直譯器 (interpreter) 會主動替物件呼叫 `__del__()` 方法 (method)，這是物件自動銷毀的方法，也就是從記憶體中釋放空間的步驟，被稱為解構子 (destructor)，當然，我們也可以改寫 (override) 這個方法。



```
class Demo:
    def __init__(self, i):
        self.i = i

    def __str__(self):
        return str(self.i)

    def __del__(self):
        print("del called: " + self.__str__())

    def hello(self):
        print("hello " + self.__str__())

a = Demo("Tommy")
a.hello()
```

- 我們只有使用變數 (variable) a 一個名稱，利用建構子 Demo() 建立物件後呼叫 hello()，然後重新呼叫 Demo() 建立另一個 Demo 型態的物件，我們可以看到直譯器主動呼叫 `__del__()`，印出 "del called" 的訊息。

最後程式結束執行前，直譯器同樣主動呼叫最後建立物件解構子，完全釋放所使用的記憶體空間。

類別多型

- 多型 (polymorphism) 是物件導向程式語言 (object-oriented programming language) 的一項主要特性，使物件 (object) 的使用更具彈性。簡單來說，多型可使物件的型態具有通用的效力，例如以下程式

```
class Animal:
    def __init__(self, name): # Constructor of the class
        self.name = name
    def talk(self): # Abstract method, defined by convention only
        raise NotImplementedError("Subclass must implement abstract
method")
class Cat(Animal):
    def talk(self):
        return 'Meow!'
class Dog(Animal):
    def talk(self):
        return 'Woof! Woof!'

animals = [Cat('Missy'), Cat('Garfield'), Dog('Lassie')]
for animal in animals:
    print(animal.name + ': ' + animal.talk())
```

其他的例子

```
d1 = "12345"
```

```
d2 = [1, 2, 3, 4, "5"]
```

```
print(d1.count("4"))
```

```
print(d2.count("4"))
```


其他的例子

- d1 為字串 (string) ， d2 為串列 (list) ， 兩者皆屬於序列 (sequence) 的複合資料型態 (compound data type) ， 有通用的 count() 方法，可計算某元素 (element) 累計出現的次數。

多型的應用很多，例如串列中可接受不同型態的物件當元素，或是方法可用不同型態的參數等。