

Appendix

D

Python 的類別、物件 與自製模組開發

Python 是一種物件導向程式語言，可以建立類別後再根據類別建立物件。

類別也可以繼承，被繼承的類別稱為父類別 (parent class) 或基底類別 (base class)，繼承的類別稱為子類別 (child class) 或衍生類別 (derived class)，子類別可以繼承父類別中所有共用屬性和方法。

使用 **Spyder** 除了建立檔案，也可以建立專案，然後在專案中再建立目錄和檔案。

一個較大型專案，程式是由許多類別或函式組成，為了程式的分工和維護，可以適度地將程式分割成許多的模組，然後再呼叫並匯入這些模組。

Python 初學特訓班





D.1 類別與物件

較完整的應用程式通常由許多類別組成，Python 是一種物件導向程式語言，可以建立類別後再根據類別建立物件。

D.1.1 建立類別

建立類別

以 **class** 可以建立類別，類別名稱的第一個字元必須使用大寫字元。語法：

```
class 類別名稱():
```

例如：建立類別 **Animal**，其中「**()**」也可以省略，寫成「**class Animal:**」。

```
class Animal():
```

類別的屬性和方法

類別中通常會建立屬性 (**attribute**) 和方法 (**method**)，提供物件使用，類別中的屬性其實就是一般的變數，而方法則是函式，但在類別中不以變數和函式稱呼，而是稱為屬性和方法。定義的方法中第一個參數必須是 **self**，第二個以後的參數則可依實際需要增加或省略。

例如：建立類別 **Animal**，並在類別建立 **name** 屬性和 **sing** 方法。
(<class01.py>)

```
1 class Animal():          # 定義類別
2     name = "小鳥"        # 定義屬性
3     def sing(self):      # 定義方法
4         print("很會唱歌!")  建立物件
```

以類別名稱即可建立物件 (**object**)。語法：

```
類別 物件
```

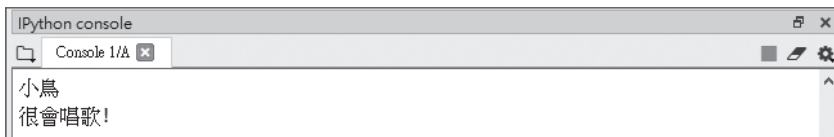
然後以物件執行其屬性和方法。

```
物件.屬性
物件.方法()
```

例如：依據類別 `Animal` 建立物件 `bird`，執行 `name` 屬性和 `sing` 方法。

```
6 bird = Animal() # 建立一個名叫 bird 的 Animal 物件
7 print(bird.name) # 小鳥
8 bird.sing()      # 很會唱歌！
```

執行結果：



```
IPython console
Console 1/A
小鳥
很會唱歌！
```

D.1.2 類別的建構式

建立類別時必須對類別初始化，因此必須建立一個特殊的方法「`__init__`」，這個初始化的方法稱為建構式，建立建構式的語法：

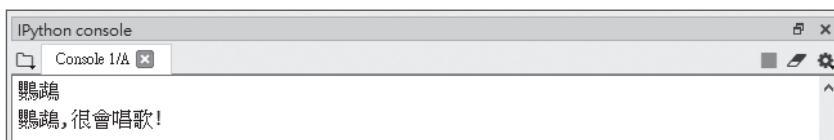
```
def __init__(self[, 參數1, 參數2,...]):
```

建構式必須使用 `__init__()` 函式，參數 `self` 是必須的，同時需要放在最前面，代表建立的物件，其餘的參數是可選擇性的。如此在類別中就可以 `self`. 屬性、`self`. 方法 執行類別的屬性和方法。

例如：建立 `Animal` 類別，並建立 `__init__()` 建構式和 `sing` 方法。(<class02.py>)

```
1 class Animal():          # 定義類別
2     def __init__(self, name):
3         self.name = name  # 定義屬性
4     def sing(self):       # 定義方法
5         print(self.name + ", 很會唱歌!")
6
7 bird = Animal("鸚鵡")    # 建立一個名叫 bird 的 Animal 物件
8 print(bird.name)         # 鸚鵡
9 bird.sing()              # 鸚鵡, 很會唱歌！
```

執行結果：



```
IPython console
Console 1/A
鸚鵡
鸚鵡, 很會唱歌！
```



程式說明

- 4 `def sing(self)` 方法中因為只有一個參數 `self`，因此第 9 列 `bird.sing()` 呼叫時不必傳入任何參數。
- 7 建立 `Animal` 物件時必須傳入一個參數給第 2 列 `__init__()` 中的參數 `name`，在類別中就可以 `self.name` 存取 `name` 屬性。

`__init__()` 建構式既然這麼重要，那為什麼 `<class01.py>` 中並沒有這個建構式呢？那是因為系統預設已隱含建立了一個 `__init__(self)` 的建構式，因為這個預設的建構式只有 `self` 參數，以 `bird = Animal()` 建立物件時就不可以傳入參數。

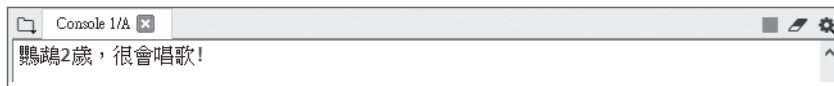
D.1.3 屬性初始值的設定

`<class01.py>` 第 2 列「`name = "小鳥"`」可以設定 `name` 的初始值，但無法在建立物件時就直接初始化，如果將初始化的動作放在 `__init__()` 建構式中，這樣我們就可以在建立物件時，透過參數設定其初始值。

例如：建立物件 `bird`，預設屬性 `name="鸚鵡"`、`age=1`。（`<class03.py>`）

```
1 class Animal():      # 定義類別
2     def __init__(self, name, age):
3         self.name = name # 定義屬性
4         self.age = age
5     def sing(self):    # 定義方法
6         print(self.name + str(self.age) + "歲，很會唱歌!")
7     def grow(self, year): # 定義方法
8         self.age += year
9
10 bird = Animal("鸚鵡", 1) # 建立一個名叫 bird 的 Animal 物件
11 bird.grow(1)           # 長大 1 歲
12 bird.sing()            # 鸚鵡 2 歲，很會唱歌！
```

執行結果：



```
Console 1/A x
鸚鵡2歲，很會唱歌!
```

程式說明

- 10 以 `bird.grow(1)` 將年齡 `age` 增加 1 歲。

D.2 類別封裝

在 `<class03.py>` 程式中，可以 `bird.age` 存取 `age` 屬性，因此就可以設定「`bird.age=-1`」設定 `bird` 年齡為 `-1` 歲，這樣直接從外部設定「年齡 `< 0`」的方式其實並不合理，因此必須對 `age` 屬性作適度的保護。

類別中可以讓外部引用的屬性稱為共用 (public) 屬性、方法稱為共用方法，在 `<class03.py>` 程式中，年齡 `age` 應該以 `bird.grow()` 方法增加，不可以從外部以 `bird.age` 直接設定。

Python 提供私用 (private) 屬性和私用方法，這種私用屬性和私用方法只有類別內部可以使用，類別外部並無法使用，這樣的觀念稱為封裝 (encapsulation)。

在屬性和方法前面加上「`__`」(兩個 `_` 字元)，就成為私用屬性和方法。

例如：類別中建立 `__name`、`__age` 屬性和 `__sing` 方法。(`<class04.py>`)

```
1 class Animal():          # 定義類別
2     def __init__(self, name,age):
3         self.__name = name  # 定義私用屬性
4         self.__age = age
5     def __sing(self):      # 定義私用方法
6         print(self.__name + str(self.__age),end= " 歲，很會唱歌。")
7     def talk(self):        # 定義共用方法
8         self.__sing()      # 使用私用方法
9         print(" 也會模仿人類說話!")
10
11 bird = Animal(" 灰鸚鵡 ",2) # 建立一個名叫 bird 的 Animal 物件
12 bird.talk()                # 灰鸚鵡 2 歲，很會唱歌，也會模仿人類說話！
13
14 bird.__age = -1            # 設定無效
15 bird.talk()                # 灰鸚鵡 2 歲，很會唱歌，也會模仿人類說話！
16 #bird.__sing()             # 執行出現錯誤
```

執行結果：

```
Console 1/A x
灰鸚鵡2歲，很會唱歌，也會模仿人類說話!
灰鸚鵡2歲，很會唱歌，也會模仿人類說話!
```

類別外部並無法使用私用屬性和方法，因此如果在第 14 列執行 `bird.__age = -1` 無效，第 16 列執行 `bird.__sing()` 將會產生錯誤。



D.3 類別繼承

類別也可以繼承，被繼承的類別稱為父類別 (parent class) 或基底類別 (base class)，繼承的類別稱為子類別 (child class) 或衍生類別 (derived class)，子類別可以繼承父類別中所有共用屬性和共用方法，在程式設計時請注意父類別必須放在子類別的前面。

D.3.1 建立子類別

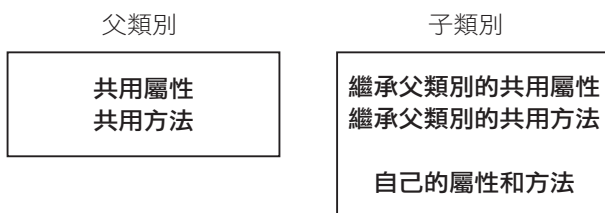
建立子類別的語法：

```
class 類別名稱 ( 父類別 ):
```

例如：建立類別 **Bird** 繼承 **Animal** 類別，其中 **Animal** 是父類別，**Bird** 是子類別。

```
class Bird(Animal):
```

子類別會繼承父類別的所有共用屬性和共用方法，也可以再建立屬於自己的屬性和方法。

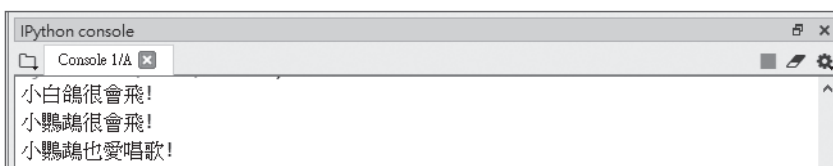


例如：建立父類別 **Animal**，包含 **name**、**fly** 共用屬性和方法，再建立子類別 **Bird** 繼承 **Animal** 類別，並在 **Bird** 類別中建立另一個共用方法 **sing**。(<class05.py>)

```
1 class Animal():          # 定義父類別
2     def __init__(self, name):
3         self.name = name  # 定義共用屬性
4     def fly(self):        # 定義共用方法
5         print(self.name + " 很會飛!")
6
7 class Bird(Animal):      # 定義子類別
8     def __init__(self, name):
9         self.name = name  # 定義共用屬性
10    def sing(self):       # 定義共用方法
11        print(self.name + " 也愛唱歌!")
12
```

```
13 pigeon = Animal("小白鴿") # 建立一個名叫 pigeon 的 Animal 物件
14 pigeon.fly() # 小白鴿很會飛!
15
16 parrot = Bird("小鸚鵡") # 建立一個名叫 parrot 的 Bird 物件
17 parrot.fly() # 小鸚鵡很會飛!
18 parrot.sing() # 小鸚鵡也愛唱歌!
```

執行結果：



程式說明

- 7 Bird 繼承 Animal 類別，也繼承了 name、fly 共用屬性和方法。
- 10~11 建立專屬於 Bird 子類別的方法 sing。
- 13 建立 Animal 父類別物件 pigeon。
- 14 執行 Animal 父類別的 fly 方法。
- 16 建立 Bird 子類別物件 parrot。
- 17 執行繼承 Animal 父類別的 fly 方法。
- 18 執行 Bird 子類別的 sing 方法。

D.3.2 子類別和父類別擁有相同的屬性和方法

有的時候會碰到子類別和父類別擁有相同的屬性和方法，此時子類別會先尋找子類別中是否有此名稱的屬性和方法，如果有找到就使用子類別的的屬性和方法，否則就使用父類別的的屬性和方法。

子類別也可用 `super()` 方法執行父類別的方法。

例如：建立類別 Bird 繼承類別 Animal，子類別 Bird 再以 `super()` 方法覆寫 `__init__()` 和 `fly()` 方法。(class06.py>)

```
1 class Animal(): # 定義父類別
2     def __init__(self,name):
3         self.name = name # 定義共用屬性
4     def fly(self): # 定義共用方法
5         print(self.name + " 很會飛!")
```



```
6
7 class Bird(Animal):      # 定義子類別
8     def __init__(self,name,age):
9         super().__init__(name)
10        self.age = age    # 定義共用屬性
11    def fly(self):        # 定義共用方法
12        print(str(self.age),end=" 歲 ")
13        super().fly()
14
15 pigeon = Animal("小白鴿") # 建立一個名叫 pigeon 的 Animal 物件
16 pigeon.fly()    # 小白鴿很會飛！
17
18 parrot = Bird("小鸚鵡",2) # 建立一個名叫 parrot 的 Bird 物件
19 parrot.fly()    # 2歲小鸚鵡很會飛！
```

執行結果：

```
IPython console
Console 1/A
小白鴿很會飛!
2歲小鸚鵡很會飛!
```

程式說明

- 8~10 `def __init__(self, name,age):` 接收兩個參數，其中 `age` 為年齡，`super().__init__(name)` 執行父類別的 `__init__()` 方法。
- 11~13 `def fly(self):` 以 `super().fly()` 執行父類別的 `fly` 方法。
- 15~16 執行的是 `Animal` 父類別的 `fly` 方法。
- 18~19 執行的是 `Bird` 子類別的 `fly` 方法。

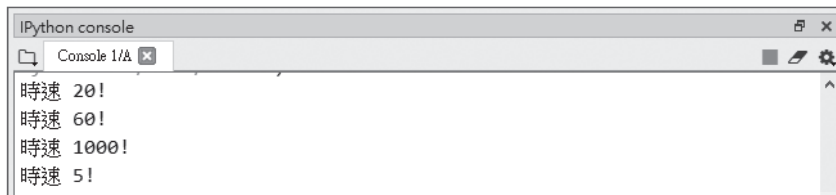
D.4 多型 (polymorphism)

前面不同類別中擁有相同的方法名稱，這樣的觀念稱為多型，但其實多型不一定要有繼承關係，多型的好處是同一個方法名稱卻可以產生不同的功能。

例如：定義 **Bird** 類別繼承 **Animal** 類別以及另一個 **Plane** 類別，這 3 個類別都擁有 **fly** 方法，此外也建立了一個 **fly** 函式。(<class07.py>)

```
1 class Animal():          # 定義父類別
2     def fly(self):        # 定義共用方法
3         print(" 時速 20!")
4
5 class Bird(Animal):       # 定義子類別
6     def fly(self,speed):  # 定義共用方法
7         print(" 時速 " + str(speed) + " !")
8
9 class Plane():            # 定義類別
10    def fly(self):         # 定義共用方法
11        print(" 時速 1000!")
12
13 def fly(speed):           # 定義函式
14    print(" 時速 " + str(speed) + " !")
15
16 animal = Animal() # 建立一個名叫 animal 的 Animal 物件
17 animal.fly()      # 時速 20!
18
19 bird = Bird() # 建立一個名叫 bird 的 Bird 物件
20 bird.fly(60)     # 時速 60!
21
22 plane=Plane() # 建立一個名叫 plane 的 Plane 物件
23 plane.fly()      # 時速 1000!
24
25 fly(5)           # 時速 5
```

執行結果：



```
IPython console
Console 1/A
時速 20!
時速 60!
時速 1000!
時速 5!
```



程式說明

- 16~17 執行的是 **Animal** 類別的 **fly** 方法。
- 19~20 執行的是 **Bird** 類別的 **fly** 方法。
- 21~22 執行的是 **Plane** 類別的 **fly** 方法。
- 23 執行的是第 13~14 列的 **fly** 函式。



如何取得父類別的私用屬性？

基於對私用屬性的保護，類別之外並無法取得類別內的私用屬性，包括它的子類別也無法讀取，如果一定非取得不可，就只能以「**return 私用屬性**」的方式，將私用屬性傳回。

例如：在子類別中以 **super().getEye()** 取得父類別的私用屬性「**self.__eye**」。
(**<getPrivateAttribute.py>**)

```
1  class Father():          # 定義父類別
2      def __init__(self,name):
3          self.name = name  # 定義私用屬性
4          self.__eye=" 黑色 "
5      def getEye(self):     # 定義共用方法傳回私用屬性
6          return self.__eye
7
8  class Child(Father):      # 定義子類別
9      def __init__(self,name,skin):
10         super().__init__(name)
11         self.skin=skin
12         self.fatherEye=super().getEye() # 取得私用屬性
13
14  joe = Child("小華","棕色") # 建立子類別物件 joe
15  print(joe.name+" 眼睛是 "+joe.skin+"，他的父親則是 "+joe.fatherEye)
16  # 執行結果：小華眼睛是棕色，他的父親則是黑色
```

D.5 多重繼承

子類別也可以同時繼承多個父類別，語法：

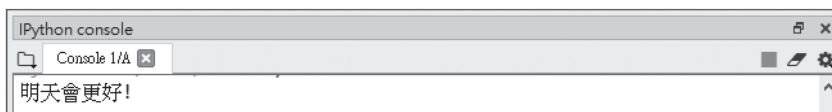
```
class 子類別名稱 ( 父類別 1, 父類別 2, ..., 父類別 n)
```

如果父類別擁有相同名稱的屬性或方法時，就要注意搜尋的順序，是從子類別開始，接著是同一階層父類別由左至右搜尋。

例如：**Child** 子類別同時以「`class Child(Father,Mother)`」繼承 **Father**、**Mother** 類別，並繼承 `say()` 方法。(<class08.py>)

```
1 class Father():          # 定義父類別
2     def say(self):        # 定義共用方法
3         print(" 明天會更好!")
4
5 class Mother():          # 定義父類別
6     def say(self):        # 定義共用方法
7         print(" 包含、尊重!")
8
9 class Child(Father,Mother): # 定義子類別
10     pass
11
12 child = Child() # 建立 child 物件
13 child.say()     # 明天會更好!!
```

執行結果：



程式說明

- 13 `child.say()` 會優先尋找 `child` 的 `say` 方法，如果找不到再尋找 `Father` 的 `say` 方法，最後才尋找 `Mother` 的 `say` 方法。因此本例會執行 `Father` 的 `say` 方法。



D.6 類別應用

範例：計算面積

定義 **Rectangle**、**Triangle** 兩個類別，父類別 **Rectangle** 定義共用屬性 **width**、**height** 和 **area()** 方法計算矩形面積。Triangle 子類別繼承 **Rectangle** 類別並增加一個計算三角形面積的方法 **area2()**。

程式碼：Area.py

```
1 class Rectangle():          # 定義父類別
2     def __init__(self, width,height):
3         self.width = width    # 定義共用屬性
4         self.height = height  # 定義共用屬性
5     def area(self):          # 定義共用方法
6         return self.width * self.height
7
8 class Triangle(Rectangle):   # 定義子類別
9     def area2(self):         # 定義子類別的共用方法
10        return (self.width * self.height)/2
11
12 triangle = Triangle(5,6)    # 建立 triangle 物件
13 print(" 矩形面積 =",triangle.area())    #30
14 print(" 三角形面積 =",triangle.area2()) #15.0
```

程式說明

- 1~6 建立父類別 **Rectangle**。
- 5~6 建立 **area** 方法計算矩形面積。
- 8~10 建立子類別 **Triangle** 繼承 **Rectangle** 類別。
- 9~10 建立 **area2** 方法計算三角形面積。
- 12 **Triangle(5,6)** 建立子類別的物件 **triangle**，並初始化。
- 13~14 計算矩形面積、三角形面積。

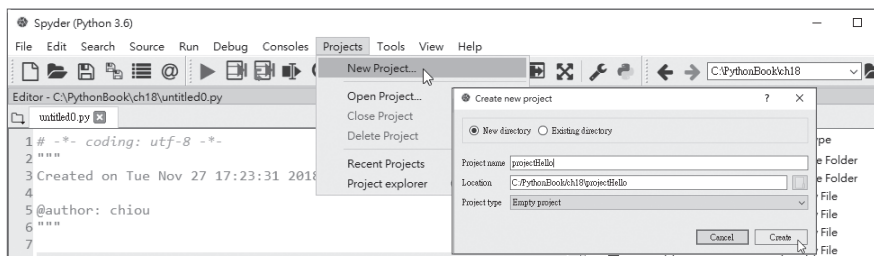
D.7 建立 Python 專案

使用 **Spyder** 除了建立檔案，也可以建立專案，然後在專案中再建立目錄和檔案，我們以實例來說明。

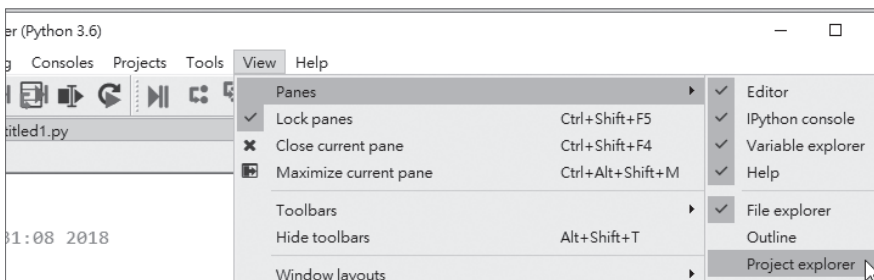
D.7.1 建立新的專案

建立專案

點選功能表 **Projects \ New Project...**，**Project name** 輸入專案名稱，**Location** 設定儲存目錄，然後按 **Create** 鈕即可以建立專案。例如：輸入「projectHello」建立 projectHello 專案。

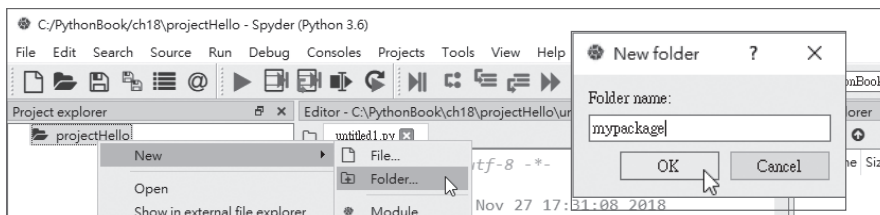


如果專案的 **Project explorer** 未開啟，請核選功能表 **View \ Panes \ Project explorer** 將它開啟。



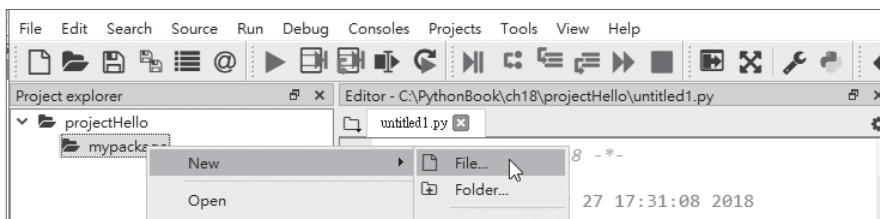
建立目錄

接著在 projectHello 專案建立一個 mypackage 目錄。請在專案名稱上按右鍵，在右鍵功能表中選 **New \ Folder...**，**Folder name** 欄位輸入目錄名稱。例如：輸入「mypackage」。

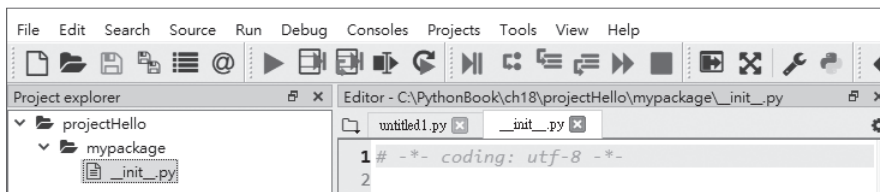


建立模組的 `__init__` 檔

每個模組裡都必須在模組所在的目錄中建立一個 `__init__.py` 檔案，它的目的是告訴 Python 將這個目錄當做模組來對待。`__init__.py` 可以是空的，也可以放一些變數或程式。請在要建立檔案的目錄按右鍵，選擇 **New \ File...**，然後輸入檔案名稱「`__init__.py`」。



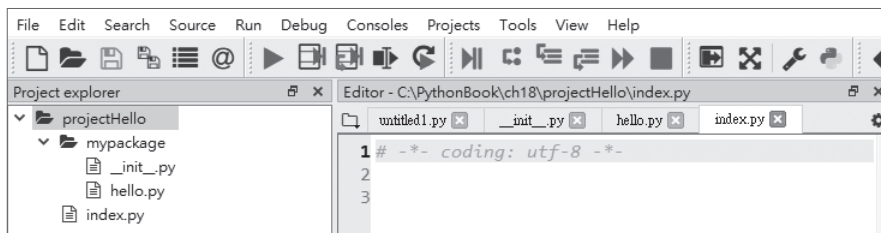
完成後的畫面如下：



建立檔案

類似的操作，再在 `mypackage` 目錄建立 `<Hello.py>`，在 `projectHello` 目錄建立 `<index.py>`。

完成後的畫面如下：



建立模組

<hello.py> 定義 SayHello 自訂函式顯示「Hello」訊息。

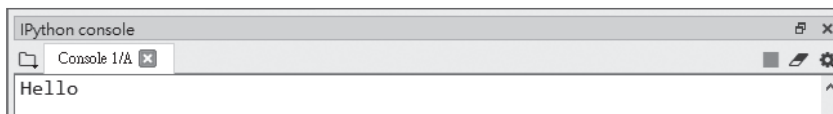
```
程式碼：hello.py  
def sayHello():  
    print("Hello")
```

使用模組

SayHello 自訂函式是在 mypackage 目錄的 <hello.py> 中，必須以 from mypackage.hello import sayHello 匯入該模組。

```
程式碼：index.py  
  
from mypackage.hello import sayHello  
  
sayHello()
```

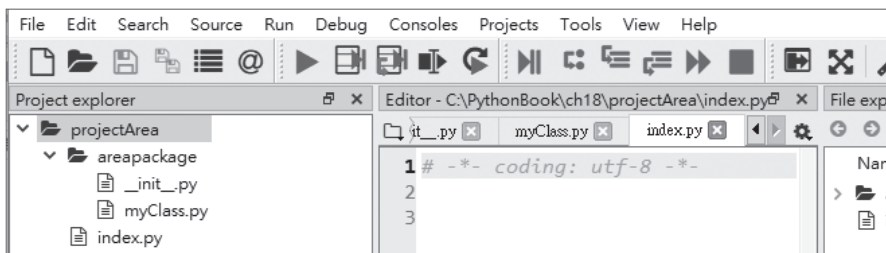
執行結果：



D.7.2 建立含有類別專案

我們也可以將類別加入到專案中，以前面範例「計算面積」為例，我們將它建立成為「projectArea」專案。

請參考前面的操作，建立「projectArea」專案，並建立 areapackage 目錄，在 areapackage 目錄建立 <__init__.py>、<myClass.py> 檔，同時在 projectArea 目錄建立 <index.py> 檔。完成後檔案架構如下圖：



然後加入 <myClass.py> 和 <index.py> 檔的程式碼。

程式碼：myClass.py

```
1 class Rectangle():          # 定義父類別
2     def __init__(self, width,height):
3         self.width = width    # 定義共用屬性
4         self.height = height  # 定義共用屬性
5     def area(self):           # 定義共用方法
6         return self.width * self.height
7
8 class Triangle(Rectangle):    # 定義子類別
9     def area2(self):          # 定義子類別的共用方法
10        return (self.width * self.height)/2
```

程式說明

- 1~6 建立父類別 **Rectangle** 和 **area** 方法計算矩形面積。
- 8~10 建立子類別 **Triangle** 繼承 **Rectangle** 類別，再建立 **area2** 方法計算三角形面積。

程式碼：index.py

```
1 from areapackage.myClass import Rectangle,Triangle
2
3 triangle = Triangle(5,6) #建立 triangle 物件
4 print(" 矩形面積 =",triangle.area())      #30
5 print(" 三角形面積 =",triangle.area2())   #15.0
```

程式說明

- 1~3 必須匯入 **Triangle** 類別才能建立 **Triangle** 類別物件。
- 4~5 計算矩形面積、三角形面積。

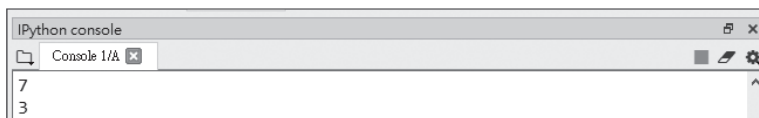
D.8 打造自己的模組

一個較大型專案，程式是由許多類別或函式組成，為了程式的分工和維護，可以適度地將程式分割成許多的模組，然後再匯入並呼叫這些模組。

D.8.1 準備工作

下列程式包含計算兩數相加、兩數相減的兩個函式，可以直接呼叫 `add`、`sub` 函式執行兩數相加、相減的運算。

```
1 # chD-1.py
2 def add(n1,n2):
3     return n1+n2
4
5 def sub(n1,n2):
6     return n1-n2
7
8 print(add(5,2)) # 7
9 print(sub(5,2)) # 3
```



有時為了程式的分工和維護，我們會將程式分割成模組。

D.8.2 打造自己的模組

首先我們將 `add`、`sub` 兩個函式建立成一個獨立的模組，模組名稱為 `<calculate.py>`。

```
1 # calculate.py
2 def add(n1,n2):
3     return n1+n2
4
5 def sub(n1,n2):
6     return n1-n2
```



D.8.3 匯入自己建立的模組

可以使用下列不同的 `import` 方法匯入並呼叫模組內的函式。

import 模組名稱

以 `import` 匯入自己建立的模組後，即可以呼叫使用這些模組內的函式。

引用自己建立的模組語法：

```
import 模組名稱
```

例如：匯入 `calculate` 模組。

```
import calculate
```

這種方式呼叫函式時，必須加上模組名稱，語法：

```
模組名稱.函式名稱
```

範例：匯入 `<calculate.py>` 模組並呼叫模組內的 `add`、`sub` 函式。

```
# chD-2.py
import calculate # 匯入 calculate 模組

print(calculate.add(5,2)) # 7
print(calculate.sub(5,2)) # 3
```

匯入模組內函式

每次使用模組內的函式都要輸入模組名稱非常麻煩，下列 `import` 的方法可改善此種情況，語法為：

```
from 模組名稱 import 函式名稱 1[, 函式名稱 2, ..., 函式名稱 n]
```

這種方式呼叫函式時，可以省略模組名稱，直接以函式名稱呼叫。

範例：匯入 `<calculate.py>` 模組內的 `add`、`sub` 函式，並呼叫模組內的 `add`、`sub` 函式。

```
1 # chD-3.py
2 from calculate import add,sub
3
4 print(add(5,2)) # 7
5 print(sub(5,2)) # 3
```

第 2 列以 `from calculate import add, sub` 同時匯入 `add`、`sub` 函式，第 4~5 列執行時就可以直接以 `add`、`sub` 呼叫函式。

但請注意：下列程式第 2 列並未 `import sub` 函式，因此第 5 列呼叫 `sub` 函式時，將會出現「`NameError: name 'sub' is not defined`」的錯誤。

```
1 # chD-4.py
2 from calculate import add
3
4 print(add(5,2)) # 7
5 print(sub(5,2)) # NameError: name 'sub' is not defined
```

匯入模組內所有函式

如果要匯入模組內所有函式，語法如下：

```
from 模組名稱 import *
```

範例：以 `import *` 匯入 `<calculate.py>` 模組內的所有函式。

```
1 # chD-5.py
2 from calculate import *
3
4 print(add(5,2)) # 7
5 print(sub(5,2)) # 3
```

這種方法雖然方便，卻隱藏著極大風險：因為每一個模組擁有眾多函式，若兩個模組具有相同名稱的函式，由於未輸入模組名稱，使用函式時將會造成錯誤。

使用 `as` 指定函式別名

如果不同模組中的函式名稱相同，或是函式名稱太長，也可以自行指定函式的別名。語法為：

```
from 模組名稱 import 函式名稱 as 函式別名
```

這樣一來，使用函式時就可用「函式別名」呼叫。例如：以別名 `a` 替代 `add` 函式。

```
1 # chD-6.py
2 from calculate import add as a
3
4 print(a(5,2)) # 7
```



使用 **as** 指定模組別名

如果模組的名稱太長，也可以將模組另取一個簡短的別名。語法為：

```
import 模組名稱 as 別名
```

這樣一來，使用函式時使用「別名.函式名稱」呼叫，就可避免輸入較長的模組名稱。例如：以別名 **cal** 替代 **calculate** 模組。

```
1 # chD-7.py
2 import calculate as cal # 匯入 calculate 模組，並取別名為 cal
3
4 print(cal.add(5,2)) # 7
5 print(cal.sub(5,2)) # 3
```

D.8.4 將自建의專案存成多個模組

前面的「projectArea」專案，其實已經將 **Rectangle**、**Triangle** 等類別存在 **areapackage** 目錄的 **<myClass.py>** 模組檔案中，主程式 **<index.py>** 要建立 **Rectangle**、**Triangle** 等類別物件就必須以「**from areapackage.myClass import Rectangle, Triangle**」匯入該類別。

當一個模組內包含太多類別時，可以將該模組再拆成更多的模組，如果拆開後不同類別的模組間有繼承關係，則子類別的模組中必須要匯入父類別，否則執行時會出現錯誤。

範例：模組匯入另一個模組

建立「projectArea2」專案，並在 **projectArea2** 建立 **areapackage2** 目錄，在 **areapackage2** 目錄建立 **<__init__.py>**、**<Rectangle.py>**、**<Triangle.py>** 檔，同時在 **projectArea2** 目錄建立 **<index.py>** 檔。

程式碼：Rectangle.py

```
1 class Rectangle(): # 定義父類別
2     def __init__(self, width,height):
3         self.width = width # 定義共用屬性
4         self.height = height # 定義共用屬性
5     def area(self): # 定義共用方法
6         return self.width * self.height
```

程式說明

- 1~6 建立父類別 `Rectangle` 和 `area` 方法計算矩形面積。

程式碼：Triangle.py

```
1 from areapackage2.Rectangle import Rectangle
2
3 class Triangle(Rectangle): # 定義子類別
4     def area2(self):        # 定義子類別的共用方法
5         return (self.width * self.height) / 2
```

程式說明

- 1 子類別必須匯入 `Rectangle` 父類別。

程式碼：index.py

```
1 from areapackage2.Rectangle import Rectangle
2 from areapackage2.Triangle import Triangle
3
4 triangle = Triangle(5,6) # 建立 triangle 物件
5 print(" 矩形面積 =", triangle.area()) #30
6 print(" 三角形面積 =", triangle.area2()) #15.0
```

程式說明

- 1~2 必須匯入 `Rectangle`、`Triangle` 類別才能建立 `Rectangle`、`Triangle` 類別物件。

D.8.5 在別的方案使用自己的模組

在「projectArea2」方案中，主程式 <index.py> 檔和 areapackage2 目錄建立 <__init__.py>、<Rectangle.py>、<Triangle.py> 檔都是在同一個方案，因此執行時不會有問題。

現在我們獨立建立一個 <CallModule.py>，這個檔案不在「projectArea2」方案中，例如：<C:\PythonBook\chD>。執行後當然會產生錯誤，因為它找不到相關模組。

程式碼：CallModule.py

```
# CallModule.py
from areapackage2.Rectangle import Rectangle
from areapackage2.Triangle import Triangle
```



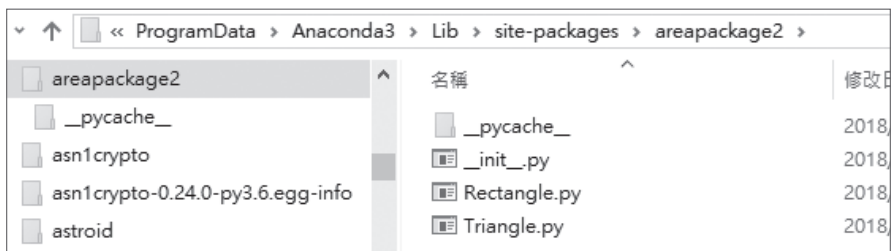
```
triangle = Triangle(5,6) # 建立 triangle 物件
print(" 矩形面積=",triangle.area())      #30
print(" 三角形面積=",triangle.area2())   #15.0
```

執行結果：

```
C:\PythonBook\ch18>python callmodule.py
Traceback (most recent call last):
  File "callmodule.py", line 2, in <module>
    from areapackage2.Rectangle import Rectangle
ModuleNotFoundError: No module named 'areapackage2'
```

那不同的專案怎麼使用自建的模組呢？以使用「projectArea2」專案 areapackage2 目錄中的 <Rectangle.py>、<Triangle.py> 模組為例，其實只要將包含模組的這個 areapackage2 目錄全部複製到 Anaconda3 中的 Lib 目錄下即可。例如：筆者路徑為 <C:\ProgramData\Anaconda3\Lib>，Python 執行時會到 <C:\ProgramData\Anaconda3\Lib> 目錄及它的子目錄搜尋指定的模組。

由於 Lib 目錄放置的是 Python 內建的模組，使用者以「pip install 模組」安裝時模組檔案是放在 <C:\ProgramData\Anaconda3\Lib\site-packages> 目錄，因此建議將 areapackage2 目錄全部複製到 <C:\ProgramData\Anaconda3\Lib\site-packages> 目錄。



複製完成後重新執行 <CallModule.py>，如下：

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

C:\PythonBook\ch18>python CallModule.py
矩形面積= 30
三角形面積= 15.0

C:\PythonBook\ch18>_
```