第六章 檔案存取與例外處理

如果要進行檔案的存取，我們可以利用內建函數open()，其用法如下。

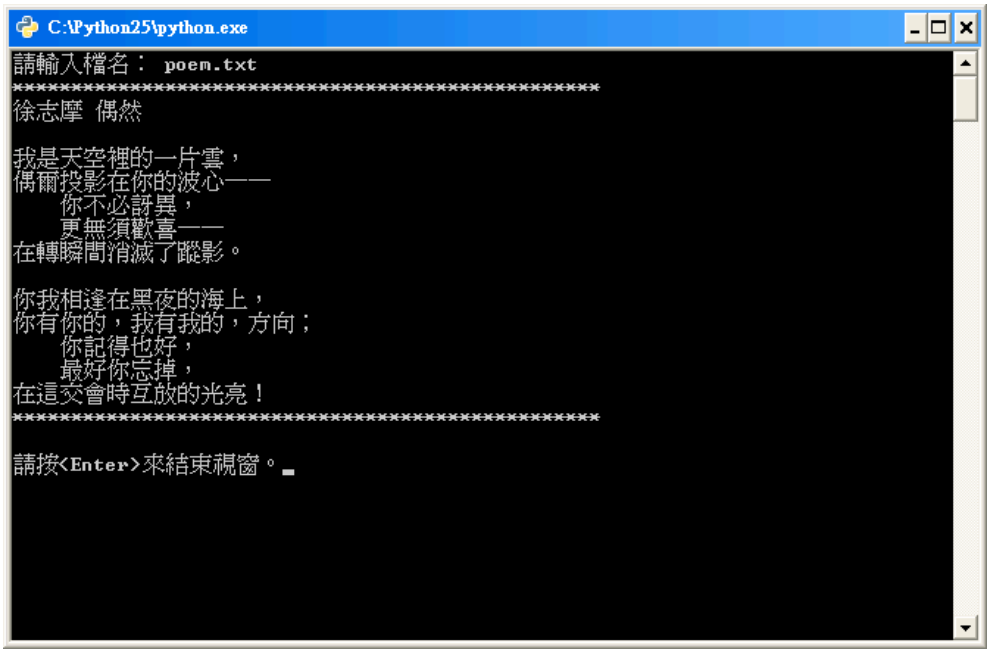
```
open("filename", "mode")
```

函數open需要兩個參數，而且兩個都為字串。第一個參數是檔案名稱，如果程式與程式所要開啟的檔案在相同目錄下，這裡只需要單純的檔名，但如果兩者並不在相同的目錄下，檔名的參數就要連帶包含目錄路徑。第二個參數則是開啟模式，大體上分為三類，“r”為讀取，“w”為覆寫或新建，“a”則為增加。

我們以一個讀取並印出純文字檔的程式作為例子。

```
#-*- coding: UTF-8 -*-

if __name__ == "__main__":
    f = open(raw_input("請輸入檔案名稱： "), "r")
    print "*"*50
    print f.read()
    print "*"*50
    f.close()
    print
    raw_input("請按<Enter>來結束視窗。")
```

先來看看程式的執行方式。



檔案poem.txt中儲存的是徐志摩的《偶然》。我們利用raw_input函數讓使用者輸入檔名，因為輸入的是字串型態，所以這裡直接用為open()函數的第一個參數，而第二個參數設定為“r”，也就是讀取模式。

我們將所開啟的純文字檔案指派到變數f之中，這時候，變數f的型態為file，也就是檔案的意思，然後對該純文字檔案的操作都可以利用變數f進行。接著我們用印出兩個星號行標明所印出的純文字檔案內容。

檔案型態的方法read()將檔案內容轉化為單一的字串，每一行的最後則是跳脫序列的“\n”，因而Python知道那是要印出新行的地方。最後，不要忘了用close方法關閉檔案。

覆寫檔案

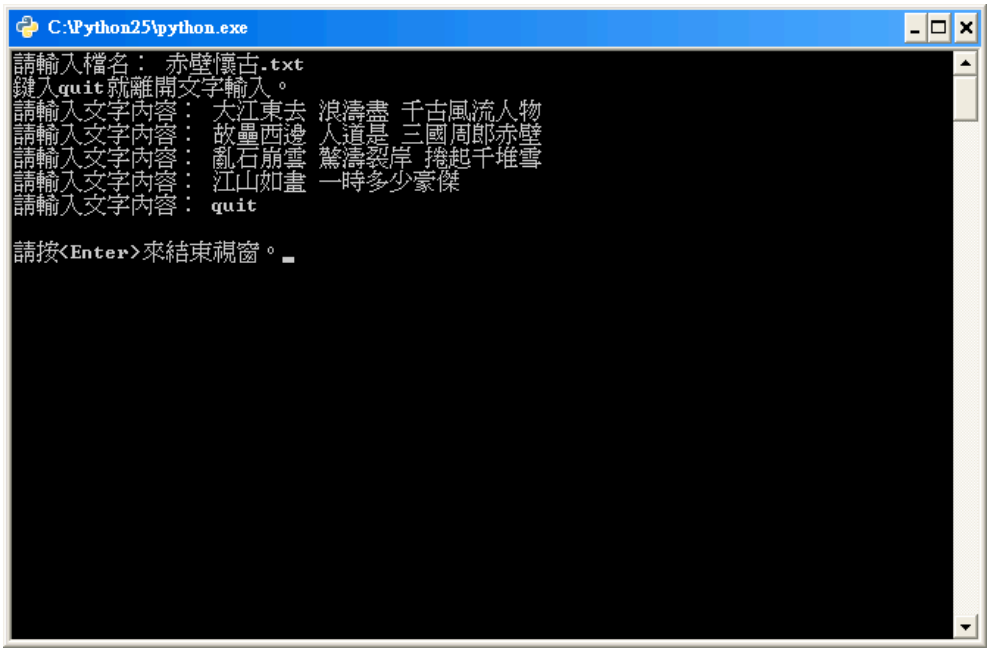
利用這種方式，我們可以寫出簡單的文字編輯器。

```
#-*- coding: UTF-8 -*-

if __name__ == "__main__":
    f = open(raw_input("請輸入檔案名稱： "), "w")

    print "鍵入quit就離開文字輸入。"
    state = True
    while state: #輸入文字的迴圈
        word = raw_input("請輸入文字內容： ")
        if word != "quit":
            f.write(word+"\n")
        else:
            state = False

    f.close()
    print
    raw_input("請按<Enter>來結束視窗。")
```

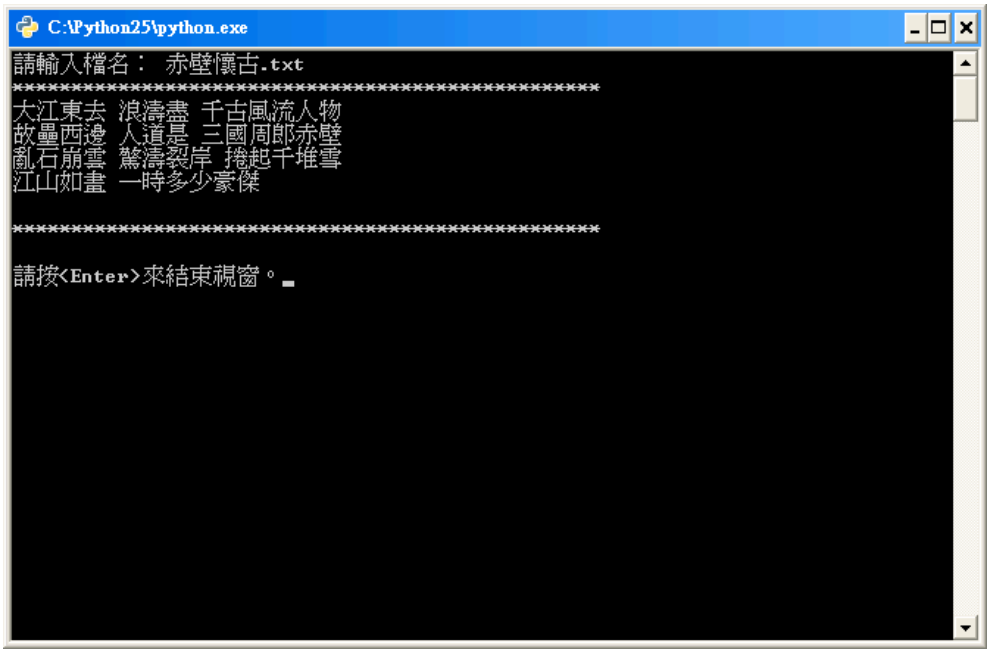
很簡單的，我們將開啟模式改變為“w”，這樣一來就能建立新的檔案或是複寫已經存在的檔案。這裡我們用了個跟主要遊戲迴圈相似的迴圈，「while state」，而變數state的初始值設為True，因此「while state」如同「while True」，這樣可以讓我們一行一行的輸入文字內容。

當然，要有方式離開「while True」的迴圈，不然程式不會終止。只要使用者不是鍵入「quit」，被儲存到變數word的文字內容就會以write方法寫入檔案中，這裡用運算式「word +“\n”」，使輸入完一行就印出新行符號。如果使用者鍵入「quit」，用作迴圈條件檢查的變數state就改變為Fasle，如此迴圈就會結束。

最後仍是要用close方法關閉檔案。我們來試看看這個程式吧！



我們再次利用前面寫過讀取純文字檔案的程式來開啟。



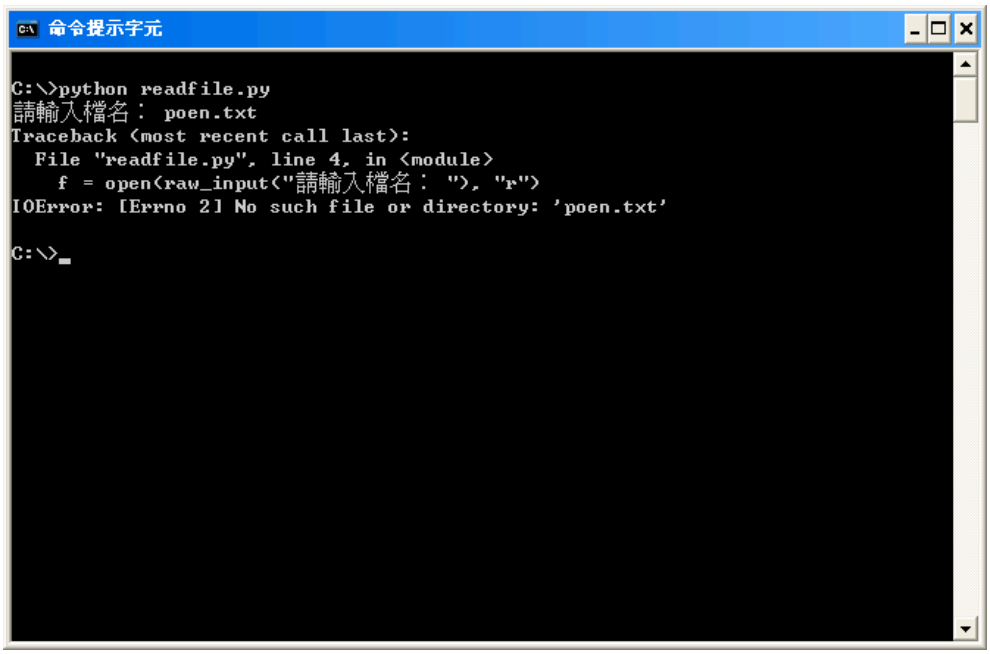
沒錯，編排上正如我們的輸入一樣，雖然讀與寫是兩個不同的程式，這卻已經是一般常見的文字編輯器，如記事本之類軟體的雛型了。

Note

有關檔案物件的詳細資訊，詳情可參考Python Library Reference的File Objects。

問題是.....

假如檔名打錯字呢？



這裡是發生了IOError，因為不存在這個檔案，所以直譯器發生執行錯誤而中斷。我們已經陸陸續續見過不少執行錯誤，包括TypeError、SyntaxError、NameError、IndexError等，尤其當我們的練習越多，因為疏失而遇到的錯誤可能也會相當頻繁。

然而程式是一種經過嚴格定義的正規語言，這是說程式控制電腦的運作，電腦所能理解的是我們所下達的每一個命令，假如命令沒有按照預先定義的方式，電腦就會不知道該做什麼。電腦不知道該做什麼，嚴重的情況會導致當機，或是其他可能會對軟體及硬體的傷害，所以程式設計師有義務將成是規劃完善，盡可能降低錯誤發生的機率。

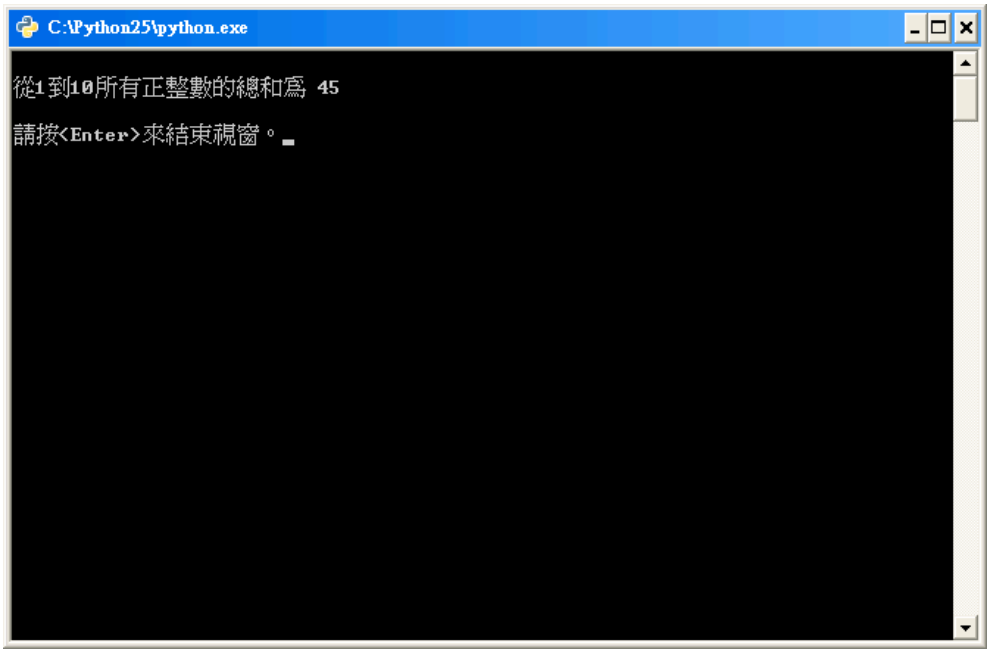
Python利用發起**例外**的方式提醒程式設計師犯錯的地方，同時提供**例外處理**的機制，讓程式執行的過程不會因為執行錯誤而中斷。雖然例外都統稱為執行錯誤，其中SyntaxError稍微特別一點，這叫做**句型錯誤**，通常是漏打冒號或是打錯關鍵字才會發生。

當我們寫程式的經驗越來越豐富，發生句型錯誤的機會也就微乎其微。但是有一個我們必須正視的問題，有一種錯誤，Python直譯器並不會發起例外來告訴我們，我們已經見過一個了，就是在第二章碰到的無窮迴圈。

這種情況則是叫做**語意錯誤**，電腦知道要做什麼，也完成了預定的工作，但是所得到的結果並不正確。再以另一個程式作為例子，譬如我們想要計算從1到10所有正整數的總和。

```
#-*- coding: UTF-8 -*-

if __name__ == "__main__":
    s = 0
    i = 0
    for i < 10:
        s = s + i
        i = i + 1

    print
    print "從1到10所有正整數的總和為", s
    print
    raw_input("請按<Enter>來結束視窗。")
```

程式執行結果如下。



答案很明顯不對，因為從1到10所有正整數的總和是55，而非45，少加了最後一個數10，為什麼呢？因為這一段程式中的條件為「while i < 10」，導致i等於10的時候就會跳出迴圈，所以少加了10。

我們用一個很簡單的例子說明語意錯誤，但是這種錯誤無法透過直譯器察覺，因而當我們測試程式感覺結果有所問題時，我們就得謹慎的逐行檢查程式碼，直到找出並且修正錯誤為止。

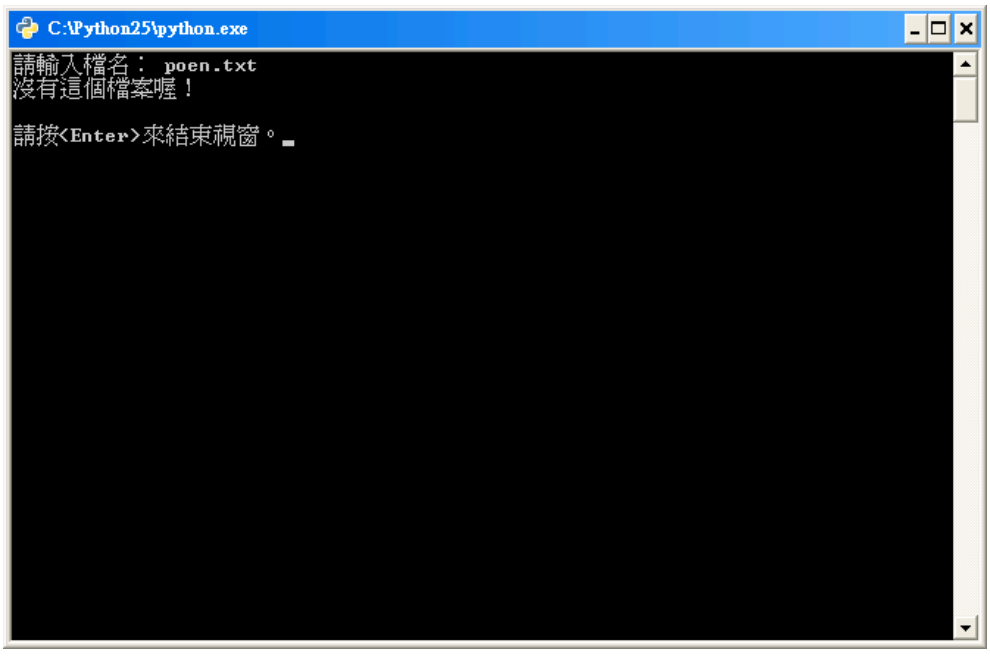
例外處理

譬如要防止讀取純文字檔案的IOError，我們將程式加入例外處理，也就是利用try及except陳述。

```
#-*- coding: UTF-8 -*-

if __name__ == "__main__":
    try:
        f = open(raw_input("請輸入檔案名稱： "), "r")
        print "*"*50
        print f.read()
        print "*"*50
        f.close()
    except IOError:
        print "沒有這個檔案喔！"

    print
    raw_input("請按<Enter>來結束視窗。")
```

程式執行時會先嘗試執行try陳述底下的程式區塊，如果發生錯誤就會跳到except陳述的部份，如果符合IOError，except陳述底下的程式區塊就會執行。這個例子中，我們嘗試開啟使用者輸入的檔名，如果檔名不存在，程式印出「沒有這個檔案喔！」的訊息給使用者。

現在來執行看看。



雖然Python內建的例外有很多，但有時候我們會希望某些情況也能被當作是例外，譬如有一個算成績平均的程式，我們不希望使用者打入負數，或是有成績為零的情況。

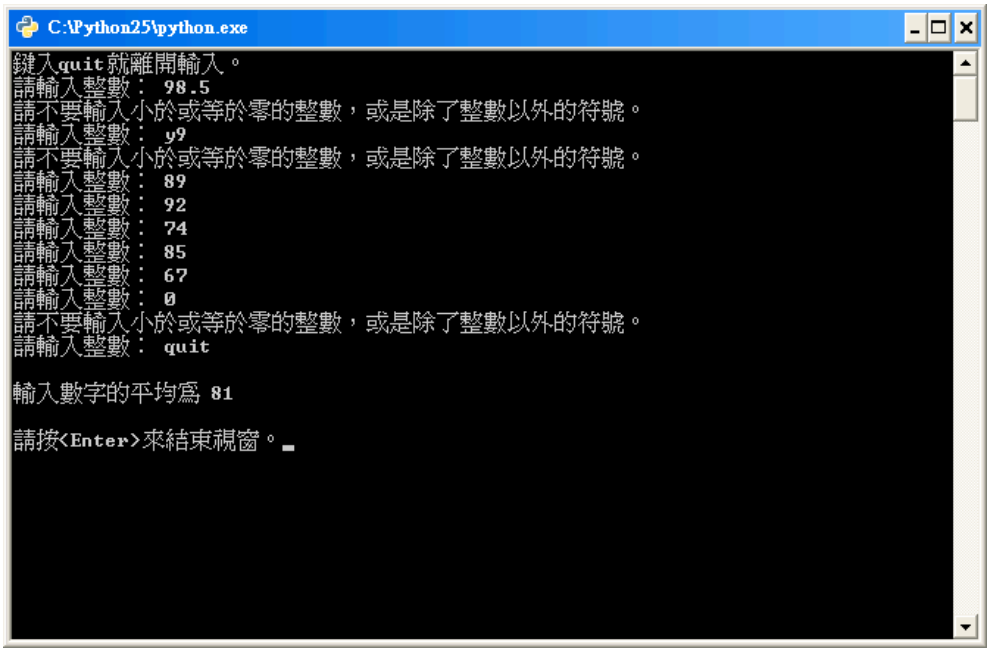
```
#-*- coding: UTF-8 -*-

if __name__ == "__main__":
    count = 0 #累計輸入次數
```

```
total = 0 #加總
state = True
print "鍵入quit就離開輸入。"
while state:
    try:
        r = raw_input("請輸入整數： ")
        if r != "quit":
            n = int(r)
            if n <= 0:
                raise ValueError
            total = total + n
            count = count + 1
        else:
            state = False #離開迴圈的條件設定
    except ValueError:
        print "請不要輸入小於或等於零的整數，或是除了整數以外的符號。"

print
print "輸入數字的平均為", total/count
print
raw_input("請按<Enter>來結束視窗。")
```

變數count用來累計輸入次數，total則用來加總。使用者輸入的內容都會被儲存到變數r之中，其型態為字串，假設r不等於“quit”，先把r轉換為整數，若是不能轉換，就是說使用者按了非數字的按鍵，於是程式會發生ValueError。

如果發生了ValueError，我們印出「請不要輸入小於或等於零的整數，或是除了整數以外的符號。」給使用者，程式不會中斷，迴圈仍會進行。又如果使用者輸入0或是負數，我們利用**raise陳述**自行引起ValueError，同樣印出訊息，然後程式仍然繼續。

直到使用者鍵入quit，變數state轉變為False，迴圈就會結束，然後計算平均同時印出結果。我們來看看這個程式的執行吧！



第四步：套件

Python也能夠利用資料夾來管理模組檔案，這樣的資料夾則被稱為**套件**。資料夾在作業系統中是用路徑表示，如Python在Windows系統中通常安裝在以下的路徑中。

```
C:\Python25>
```

Windows用反斜線表示路徑，UNIX是用斜線，Python程式則是利用小數點表示法。譬如我們有一個package套件，其內包含__init__與greeting模組，__init__.py是必要的，雖然可以只是一個空檔案，因為這是要讓直譯器知道此路徑為套件。然後，我們先在greeting.py放入以下的內容。

```
#-*- coding: UTF-8 -*-

def hello(name):
    print "你好， ", name, "！"
```

我們再用下面的程式存取package套件中的greeting模組。

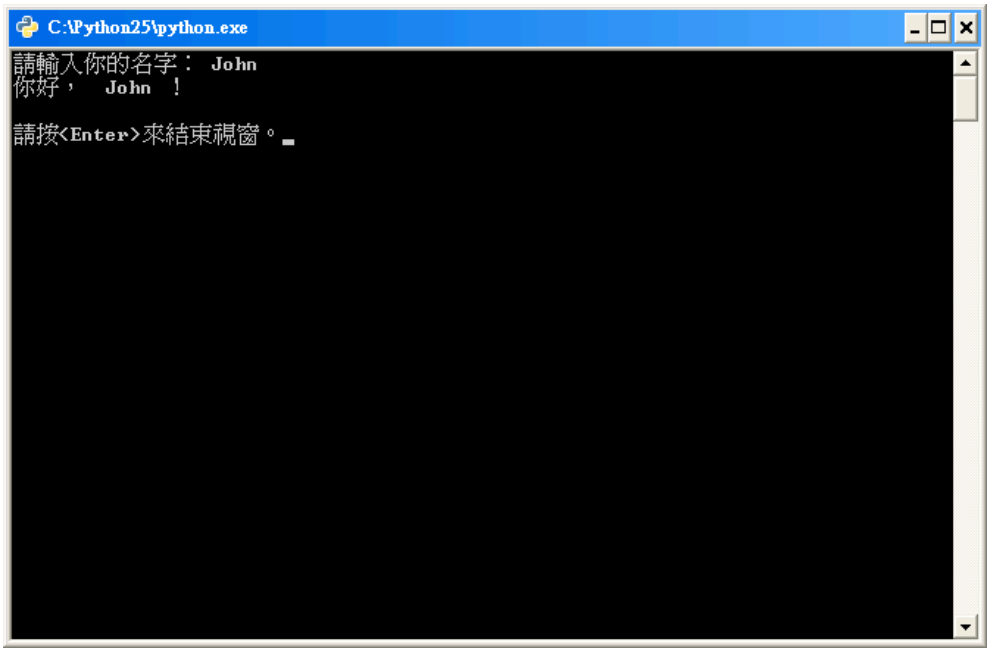
```
#-*- coding: UTF-8 -*-

from package.greeting import *

if __name__ == "__main__":
    hello(raw_input("請輸入你的名字： "))

    print
    raw_input("請按<Enter>來結束視窗。")
```

這裡要注意套件名稱在前面，小數點後面才跟著模組名稱，星號*則表示所有的名稱，包括變數、函數或是型態。

Note

下一章我們開始介紹**標準模組庫**，以及第二篇以後所要介紹的**第三方模組庫**，原始的模組檔案都是以套件的形式組織的。

我們來執行這個程式看看吧！



鬥獸棋的情況

接下來，我們將前兩章所發展的鬥獸棋遊戲也都利用套件管理，將point模組放入vector資料夾，checker模組放入jungle資料夾之中，並且分別在該資料夾之中增加空白的__init__.py檔案。

然後，將引入模組的兩行程式碼更改如下。

```
from vector.point import Point
from jungle.import Checker, Jungle
```

這一章的後半段我們繼續發展這個遊戲，主要目的是要讓棋子能夠在棋盤上移動。

棋子的移動

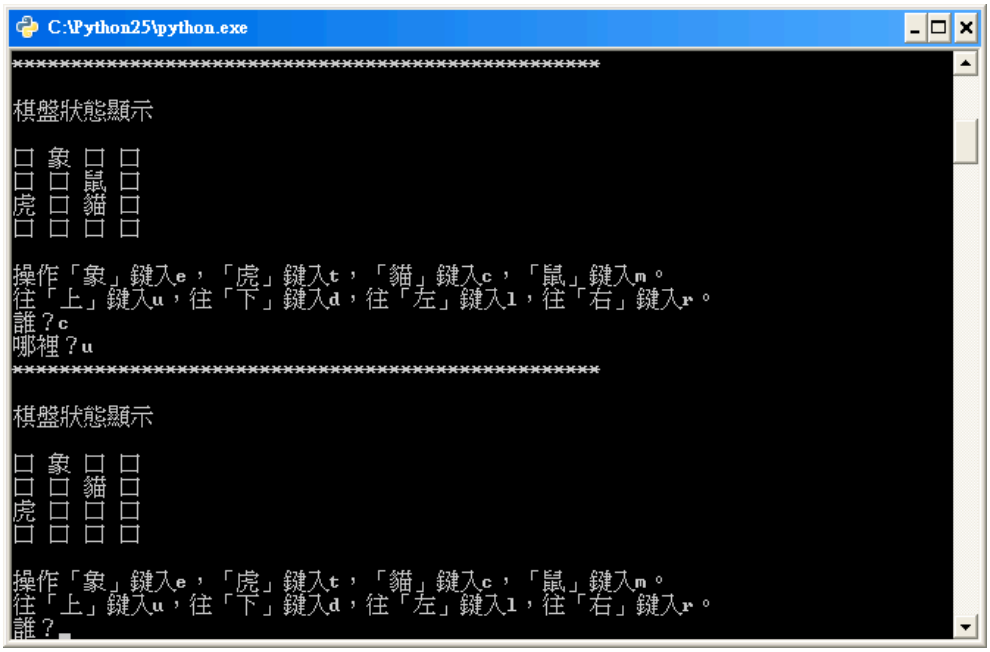
棋子要在棋盤上移動的簡單，只需要改變棋子的座標即可。我們利用另一個函數handle來處理移動的情形，這需要兩個參數，分別是players中的key，以及要往哪個方向移動。

```
def handle(name, direction):
    if direction == "u":
        players[name][1] = players[name][1] - Point(0,1)
    elif direction == "d":
        players[name][1] = players[name][1] + Point(0,1)
    elif direction == "l":
        players[name][1] = players[name][1] - Point(1,0)
    elif direction == "r":
        players[name][1] = players[name][1] + Point(1,0)
    else:
        print "請輸入正確的方向！！"
```

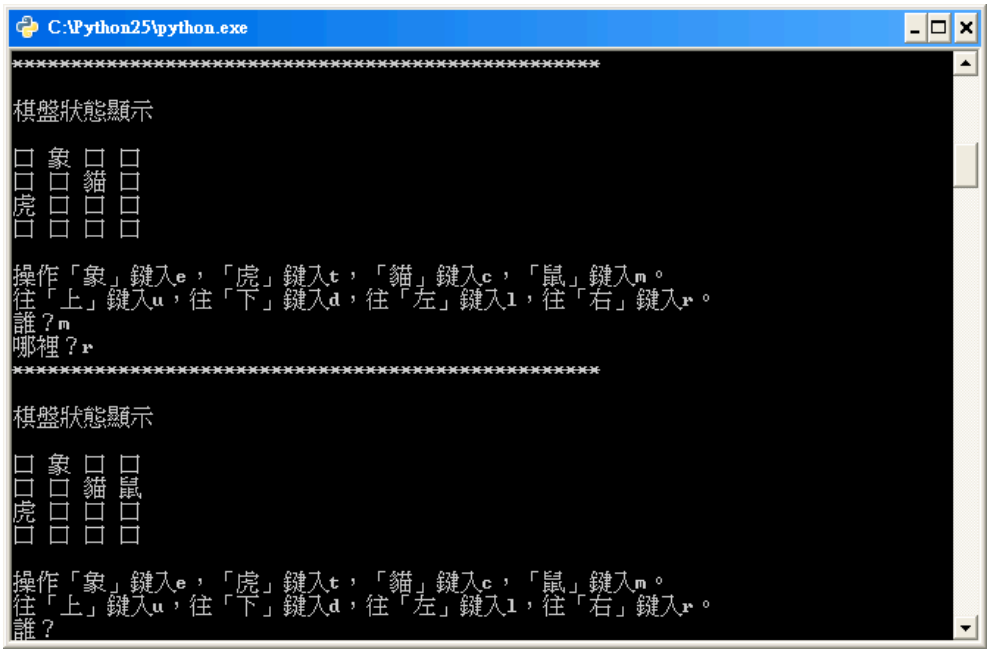
另外要將主要遊戲迴圈中的操作提示與執行棋子互吃方法兩個部份改變如下。

```
print "操作「象」鍵入e，「虎」鍵入t，「貓」鍵入c，「鼠」鍵入m。"
print "往「上」鍵入u，往「下」鍵入d，往「左」鍵入l，往「右」鍵入r。"
handle(raw_input("誰？"), raw_input("哪裡？"))
```

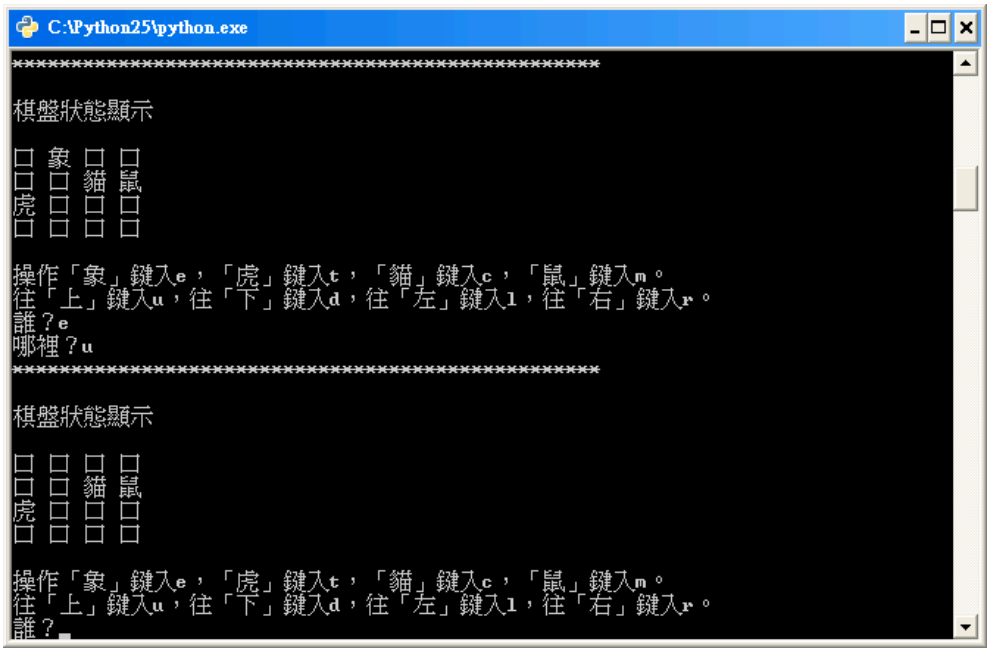
來試看看吧！



咦？「貓」吃掉「鼠」了嗎？



答案是沒有，「貓」與「鼠」共同佔據棋盤裡的同一個位置，而且還有一個很嚴重的問題。



「象」走出棋盤了！

十六個的座標

這兩種問題都構成了語意錯誤，無法用簡單的例外處理來解決。第一個問題是棋子相遇的問題，這比較麻煩，我們稍後再提。第二個問題是「象」走到(0,-1)的位置，事實上這已經在棋盤外面，導致函數status無法印出「象」這個棋子。

我們希望棋子不會也不要走到棋盤外，棋盤的座標有(0, 0)、(0, 1)、(0, 2)、(0, 3)、(1, 0)、(1, 1)、(1, 2)、(1, 3)、(2, 0)、(2, 1)、(2, 2)、(2, 3)、(3, 0)、(3, 1)、(3, 2)、(3, 3)共十六個，因此假

如我們能先行計算棋子走到位置的座標值，然後做一個檢查該座標值是否在這十六個之中，條件為真則移動棋子，條件為假就印出錯誤訊息，而棋子也停留在原地不動。

為此，我們要先建立一個變數囊括這十六個座標值。

```
temp = []
for i in range(4):
    for j in range(4):
        temp.append(Point(i,j))

board = tuple(temp)
```

為了方便起見，我們把這個變數當成Checker型態的一個屬性值。

```
class Checker(Object):
    def __init__(self, square=4):
        self.alive == True #棋子的初始條件

        temp = []
        for i in range(4):
            for j in range(4):
                temp.append(Point(i,j))

        self.board = tuple(temp)
……
```

因為Jungle型態繼承自Checker，所以也要稍做修改。

```
class Jungle(Checker):
    def __init__(self, name, square=4):
        Checker.__init__(self, square=4): #繼承自Checker的__init__()
……
```

然後我們替上下左右的移動各自寫一個函數。

```
def up(name):
    if players[name][1] - Point(0,1) in board:
        players[name][1] = players[name][1] - Point(0,1)
    else:
        print “到邊界囉！”

def down(name):
    if players[name][1] + Point(0,1) in board:
        players[name][1] = players[name][1] + Point(0,1)
    else:
        print “到邊界囉！”

def left(name):
    if players[name][1] - Point(1,0) in board:
        players[name][1] = players[name][1] - Point(1,0)
    else:
        print “到邊界囉！”

def right(name):
    if players[name][1] + Point(1,0) in board:
        players[name][1] = players[name][1] + Point(1,0)
    else:
        print “到邊界囉！”
```

布林函數

我們回到第一個問題，也就是棋子相遇的問題。首先，遊戲開始的時候四隻棋子都有設定位置，「象」、「獅」、「貓」、「鼠」分別在(1, 1)、(1, 2)、(2, 2)、(2, 1)，如果我們讓「象」走到(1, 0)，因為那裡沒有其他的棋子，於是可以直接套用函數up，把「象」的座標由(1, 1)改變成(1, 0)。

但是如果我們要「象」走到(1, 2)，這就發生問題了，「獅」在那裡，若是直接套用函數down，便會發生「象」與「獅」擠在同一個座標，而且「象」無法吃掉「獅」。

因為所有的棋子都儲存在字典players之中，輔助記錄棋子座標的Point型態放在作為value的串列中，如果我們能在移動棋子前先檢查所欲到達的座標有沒有其他棋子，問題似乎就簡單多，有其他棋子能吃就移動過去，不能吃則印出訊息。

這是說我們要由value來找key，但是字典型態的本意是由key找value，因而沒有內建的方法可以讓value找key。沒關係，我們可以自己寫個函數來完成這個任務。

```
def findkey(p):
    j = 0
    for i in players.items():
        if players.items()[j][1][1] == p:
            return players.items()[j][0]
        j = j + 1
    return 0
```

字典的內建方法items()把每一個key-value用序對放在一起，然後全部的序對存到串列之中，所以作為參數的座標如果在players的value之中，我們就能依此回傳key，如果沒有則回傳0。

除了findkey函數，我們還需要一個encounter函數來處理實際相遇的情況。

```
def encounter(first, second):
    if players[first][0].capture(players[second][0]):
        del players[second]
        return True
    else:
        return False
```

這個函數其實就是本來在主要遊戲迴圈中執行棋子互吃方法，然而我們希望簡單一點，用capture方法直接能做條件檢查，於是要將capture方法的定義更改如下。

```
class Jungle(Checker):

……

    def capture(self, other):
        if self.alive and other.alive:
            if other.name in self.food:
                other.dead()
                return True
            else:
                return False
```

如果capture方法為真，那麼刪除該位置存放在players中的棋子，並且回傳True，而capture方法為假則直接回傳False，像這樣最後回傳True/False值的函數被稱為**布林函數**，我們可以直接利用這樣的函數進行條件檢查。

最後，我們把findkey與encounter兩個函數加入handle函數之中。

```
def handle(name, direction):
    if direction == "u":
        if findkey(players[name][1] - Point(0,1)):
            if encounter(name, findkey(players[name][1] - Point(0,1))):
                up(name)
        else:
            up(name)
    elif direction == "d":
```

```
        if findkey(players[name][1] + Point(0,1)):
            if encounter(name, findkey(players[name][1] + Point(0,1))):
                down(name)
        else:
            down(name)
    elif direction == "r":
        if findkey(players[name][1] + Point(1,0)):
            if encounter(name, findkey(players[name][1] + Point(1,0))):
                right(name)
        else:
            right(name)
    elif direction == "l":
        if findkey(players[name][1] - Point(1,0)):
            if encounter(name, findkey(players[name][1] - Point(1,0))):
                left(name)
        else:
             left(name)
    else:
        print "請輸入正確的方向！！"
```

還有一個問題

問題全部解決了嗎？不，還有一個問題。原本程式要求使用者依序輸入哪兩隻棋子，實際輸入的是棋子的key值，然後檢查players是否存在所輸入的key值，如果players中沒有相對的key值，遊戲直接跳到下一輪。

現在要求的是使用者輸入一隻棋子的key值與移動的方向，函數handle利用if...elif...else陳述巧妙的防止使用者輸入錯誤方向，我們並沒有對使用者輸入的key值做檢查，於是使用者按錯按鍵，譬如原本想操作「象」，卻鍵入了r，直譯器便會引發KeyError。

對，這是一個例外，因此加入例外處理就能輕鬆解決這個問題。我們把所有程式碼列出如下。

```
#-*- coding: UTF-8 -*-

"""簡化版鬥獸棋遊戲。"""

from vector.point import Point
from jungle.checker import Checker, Jungle

#初始條件設定
#參與遊戲的動物棋子
players = {"e":[Jungle("E"),Point(1,1)], "t":[Jungle("T"),Point(1,2)], "c":
[Jungle("C"),Point(2,2)], "m":[Jungle("M"),Point(2,1)]}

#印出棋盤的函數
def status(square):
    """以4×4的方格顯示棋盤。"""

    print "棋盤狀態顯示"
    print
    for j in range(square):
        for i in range(square):
            if players.has_key("e") and i == players["e"][1].x \
                                    and j == players["e"][1].y:
                print "象",
            elif players.has_key("t") and i == players["t"][1].x \
                                      and j == players["t"][1].y:
                print "虎",
```

```
            elif players.has_key("c") and i == players["c"][1].x \
                                        and j == players["c"][1].y:
                print "貓",
            elif players.has_key("m") and i == players["m"][1].x \
                                        and j == players["m"][1].y:
                print "鼠",
            else:
                print "口",
        print

#往上移動
def up(name):
    if players[name][1] - Point(0,1) in players[name][0].board:
        players[name][1] = players[name][1] - Point(0,1)
    else:
        print "超過邊界囉！"

#往下移動
def down(name):
    if players[name][1] + Point(0,1) in players[name][0].board:
        players[name][1] = players[name][1] + Point(0,1)
    else:
        print "超過邊界囉！"

#往左移動
def left(name):
    if players[name][1] - Point(1,0) in players[name][0].board:
        players[name][1] = players[name][1] - Point(1,0)
    else:
        print "超過邊界囉！"

#往右移動
def right(name):
    if players[name][1] + Point(1,0) in players[name][0].board:
        players[name][1] = players[name][1] + Point(1,0)
    else:
        print "超過邊界囉！"

#從字典型態變數players的value找相對應的key
def findkey(p):
    j = 0
    for i in players.items():
        if players.items()[j][1][1] == p:
            return players.items()[j][0]
        j = j + 1
    return 0

#處理兩棋子的相遇情況
def encounter(first, second):
    if players[first][0].capture(players[second][0]):
        del players[second]
        return True
    else:
        return False

#控制遊戲的函數
def handle(name, direction):
    try:
        if direction == "u":
            if findkey(players[name][1] - Point(0,1)):
```

```
                if encounter(name, findkey(players[name][1] - Point(0,1))):
                    up(name)
            else:
                up(name)
        elif direction == "d":
            if findkey(players[name][1] + Point(0,1)):
                if encounter(name, findkey(players[name][1] + Point(0,1))):
                    down(name)
            else:
                down(name)
        elif direction == "l":
            if findkey(players[name][1] - Point(1,0)):
                if encounter(name, findkey(players[name][1] - Point(1,0))):
                    left(name)
            else:
                left(name)
        elif direction == "r":
            if findkey(players[name][1] + Point(1,0)):
                if encounter(name, findkey(players[name][1] + Point(1,0))):
                    right(name)
            else:
                right(name)
        else:
            print "請輸入正確的方向！！"
    except KeyError:
        print "請輸入正確的棋子！！"

def run():
    while len(players) > 1:
        status(4)

        #操作提示
        print
        print "操作「象」鍵入e，「虎」鍵入t，「貓」鍵入c，「鼠」鍵入m。"
        print "往「上」鍵入u，往「下」鍵入d，往「左」鍵入l，往「右」鍵入r。"
        handle(raw_input("誰？"), raw_input("哪裡？"))

        #印出間隔線
        print "*"*50
        print

    #印出遊戲勝利者
    for winner in players.values():
        if winner[0].alive == True:
            print winner[0].name, "是最後的存活者！"

if __name__ == "__main__":
    run()
    print
    raw_input("請按<Enter>來結束視窗。")
```