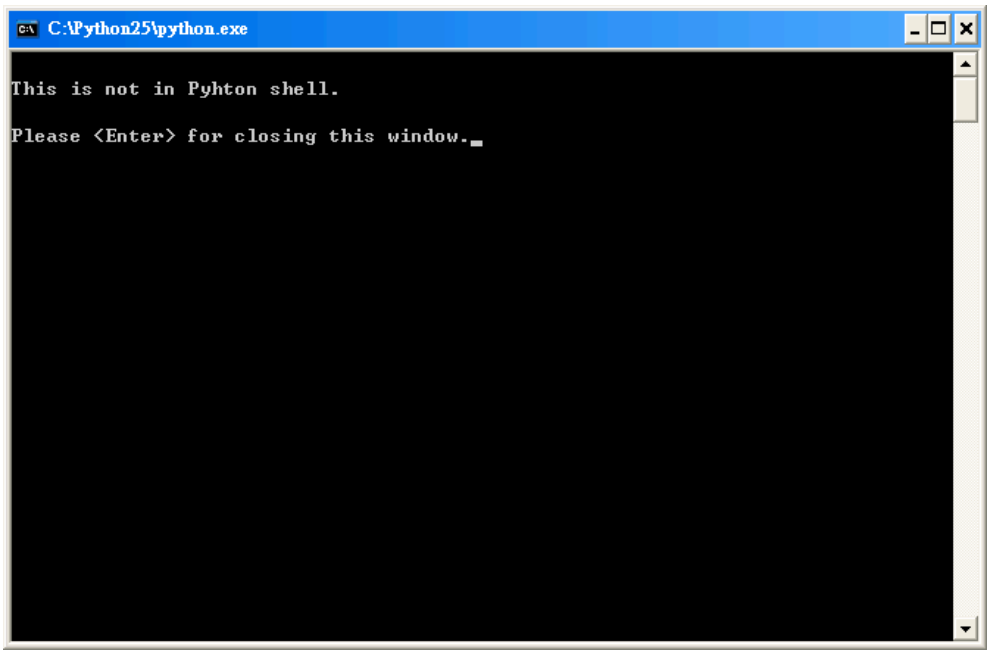
第五章 程式結構

到目前為止我們執行程式都透過IDLE的Python shell，這沒什麼問題，但很多時候我們仍會想要直接執行我們所寫的程式，就像Windows底下的exe執行程式一般。這不難，因為有安裝Python的Windows環境，所有儲存Python程式碼的.py檔案都可以利用滑鼠點兩下直接執行。



程式只要沒有加入圖形使用者介面，預設會從命令列模式來執行，對Windows系統來說就是「命令提示字元」的視窗。大體上在命令列模式下所顯示的方式都跟IDLE的Python shell類似，IDLE則提供了程式編輯的視窗，讓我們可以另外寫程式，然後載入Python shell執行。

Note

什麼是圖形使用者介面？其實舉凡現在的Mac、Windows或是Linux中的KDE、GNOME桌面環境都是建構在圖形使用者介面之下，所以圖形使用者介面可以說是由一個個的視窗組成的環境。我們到第二篇會開始接觸用Python建構的圖形使用者介面，Pygame，這是有關遊戲的設計，而到第三篇會接觸另一種Python的圖形使用者介面，wxPython，利用Python進行應用程式的設計。

如果我們不用IDLE，那我們就需要利用其他如記事本之類的文字編輯器，將程式碼在文字編輯器中輸入完成，接著點擊檔案圖示兩下，Python程式就會被執行，如果有任何的錯誤發生，直譯器會顯示訊息在「命令提示字元」的視窗上。

但是有個問題，嗯，這會是個很嚴重的問題，這個印出英文訊息的原始程式碼如下：

```
print
print “This is not in Python shell.”
print
raw_input(“Please <Enter> for closing this window.”)
```

Note

從這一章開始，我們要離開IDLE寫程式，然後執行在作業系統的環境下進行測試，所有的程式原始碼都以Courier的字體表示，請讀者自行挑選適合的文字編輯器，當然，也可以繼續利用IDLE來編輯。

單獨的print陳述印出一個空白行，接著的print陳述印出英文訊息，再接著印出一個空白行，最後利用內建的raw_input()函數，這種技巧使程式需要接受使用者按下<Enter>鍵，程式才會結束，因而「命令提示字元」視窗不至於立即關閉，使我們得以看見程式執行過程中的印出結果。

如果程式碼都是英文就不會有問題，但如果程式內容加入了中文，問題就會發生。譬如我們將程式內容更改如下。

```
print
print "這不是在Python shell執行的程式。"
print
raw_input("請按<Enter>來結束視窗。")
```

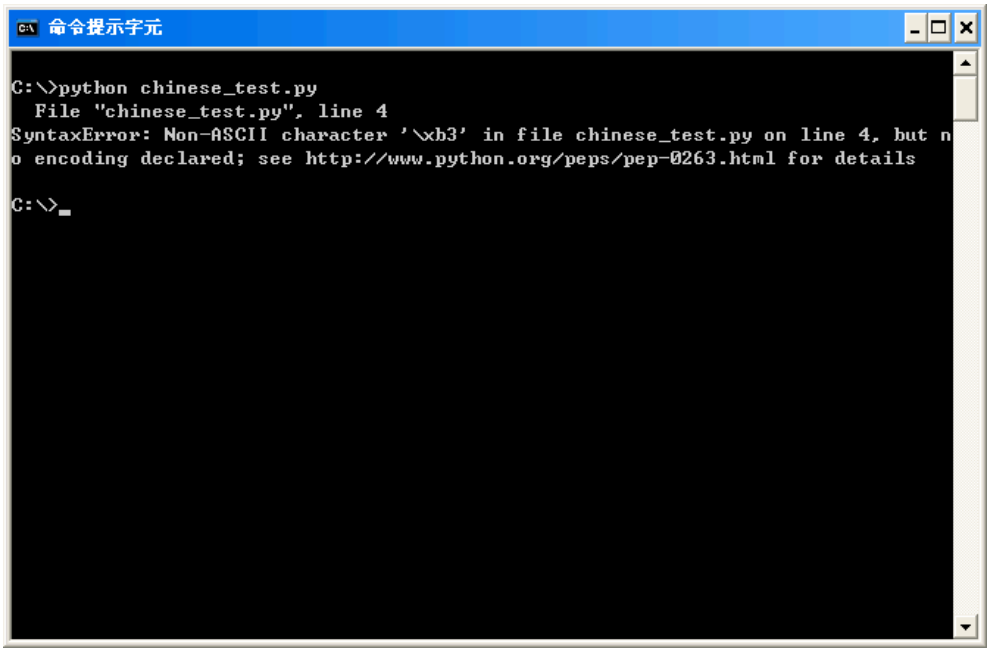
很快的點擊程式的檔案圖示兩次，我們可能會驚呼有個視窗一閃而過，卻沒有任何的「命令提示字元」視窗給我們看到結果，這是為什麼呢？

Note

在MS-Windows作業系統我們可以點擊程式的檔案圖示兩次來執行程式，而在Linux或Mac中，Python大都已經預先裝好了，請開啟終端機，然後移動到程式檔案所存放的目錄下，鍵入「python 檔名.py」的命令，這樣就可以順利執行程式。

編碼問題

我們如果在「命令提示字元」的視窗直接執行這個Python程式，我們可以見到直譯器提供的錯誤訊息。



Note

如果要在Windows環境中如同Unix直接在命令列模式下啟動Python直譯器，須在環境變數Path中加入Python的安裝路徑。

這是由於程式檔案中包含了非ASCII字元，我們在第一章提過，這種非ASCII字元是屬於Big5編碼，直譯器告訴我們並沒有任何的**編碼宣告**，並且建議我們閱讀PEP0263以獲得更多的資訊。

為什麼要做編碼宣告？這是因為Python設計的哲學就是簡單，為了簡單，以致所有的Python程式碼都用英文撰寫，而表達英文二十六個大小寫字母及常用符號，使用ASCII編碼便已足夠，所以Python原生支援的就只有ASCII編碼。然而這是不足的，因為即使英文為全球的通用語言，仍然有很多人無法學習英文，而他們或多或少需要用到軟體，所以如何增進人與軟體之間的互動，這便落到程式設計師的肩膀上來。

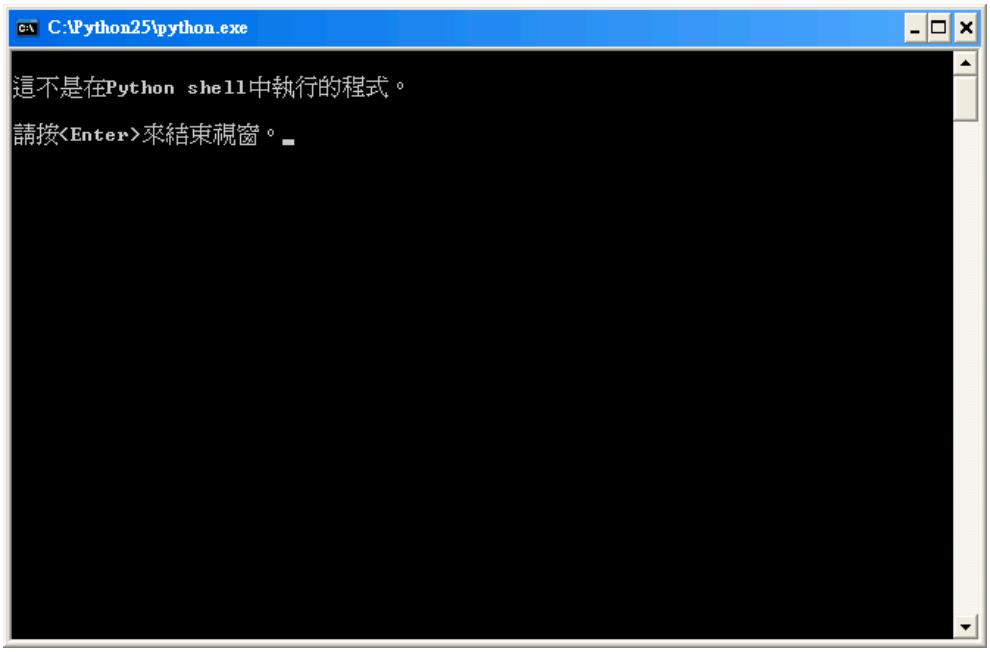
所以在程式裡使用中文，我們需要在程式的第一行或第二行進行編碼宣告，也就是加入如下的陳述。

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

UTF-8是unicode編碼的一種，其囊括多數的中文字編碼，我們之前的程式中加入這個陳述。

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

print
print "這不是在Python shell執行的程式。"
print
raw_input("請按<Enter>來結束視窗。")
```

接著來執行看看。



這樣問題就解決了嗎？大體上，如果我們一直使用的是中文的Windows系統，這樣問題其實已經完滿解決了，然而如果我們有時要將程式檔案移轉到Mac或Linux下使用，卻會發現程式執行的結果是一堆亂碼。

這仍然是編碼的問題，有關純文字檔案的編碼上，中文的Windows系統所用的是Big5編碼，Mac或Linux則多半是採用unicode編碼，所以執行上雖然沒有發生錯誤，Mac或Linux卻會硬將Big5編碼找unicode對應，導致執行結果會是一堆亂碼。

因此如果程式有跨平台需求，我們強烈建議程式檔案分別用Big5及unicode兩種編碼來儲存。

Note

我們需要認識的是Python的.py檔案，也就是儲存程式碼的檔案，大多是屬於純文字檔案。Unicode的優點是廣泛支援各國語言文字的編碼方式，因此對Mac或Linux而言，不論在世界何處使用純文字檔案交流，對當地的語言文字都有一定的支援。繁體中文的Windows環境長期以來一直都是用Big5編碼，導致純文字檔案的利用限於安裝繁體中文的Windows系統，不然就得轉換編碼的方式。

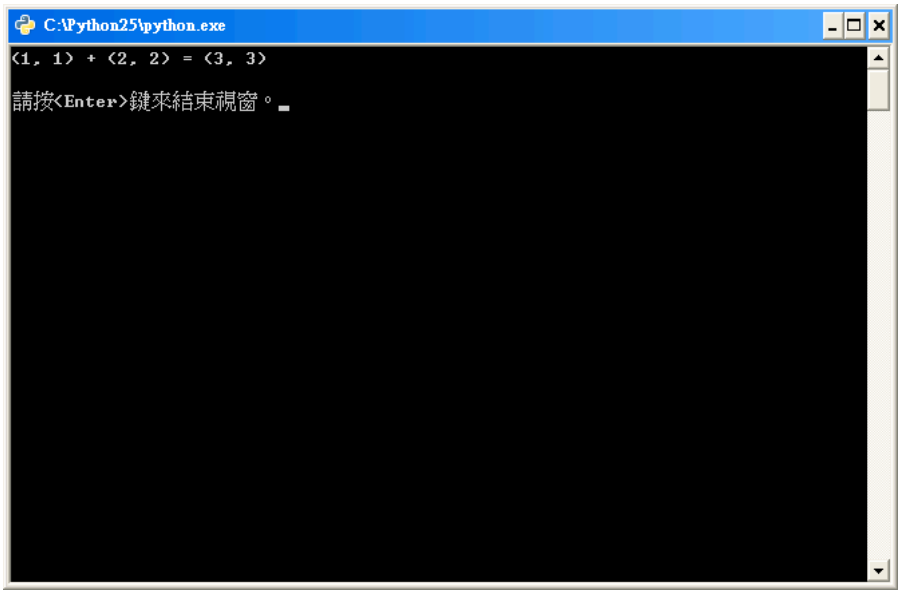
第三步：模組

我們在第三章中提過所有的Python檔案都是**模組**，其他的程式就能夠利用模組中所定義的函數及型態，我們現在就來看看要如何利用。

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

import point

a = point.Point(1,1)
b = point.Point(2,2)
print a, “+”, b, “=”, a+b
print
raw_input(“請按<Enter>來結束視窗。”)
```

利用**import陳述**就能利用其他模組中所定義的型態及函數。假設上一章中我們所寫的Point型態是儲存在point.py的檔案中，而我們要在目前的程式檔案引入point.py中Point型態的定義，簡單的作法便是第三行的「import point」，point即是我們所要引入的模組名稱，import陳述中不需要加上副檔名。

第五行和第六行利用引入的point模組定義新的變數，注意，這裡的寫法要用到小數點記號，小數點前是模組名稱，小數點後則是我們所要用的型態定義。執行這個程式我們可以得到如下的結果。



其實只要**名稱空間**不相衝突，也就是引入模組的程式中若是沒有其他的point定義，我們可以直接引入型態定義（或是函數定義）的名稱，這樣就不需要用到小數點記號。

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

```
from point import Point

a = Point(1,1)
b = Point(2,2)
print a, “+”, b, “=”, a+b
print
raw_input(“請按<Enter>來結束視窗。”)
```

關鍵字from搭配import陳述使用，這樣我們就可以直接引入Point的名稱了。

> **Note**
> 這裡import陳述的用法，被引入的模組與欲引入的程式必須儲存在相同的資料夾下，不然Python會找不到模組而發生錯誤。

模組與程式

上一章的鬥獸棋遊戲中，我們將其程式碼儲存到checker.py，可以直接點兩下執行嗎？答案是不行的，因為我們只定義了兩個型態與一個函數，程式的最後並沒有放入呼叫main函數，因此Python直譯器讀取完檔案的中所有的定義，卻不知道該如何去執行。

雖然我們可以像第三章一般，將main()函數放到程式最後沒有縮排的地方，但是這會產生一個不良的影響，就是當我們在其他的程式把checker.py當成模組，引入到程式中的時候，正因main()函數放在沒有縮排的地方，導致執行新的程式都會先去執行作為模組的checker.py。

嗯，這不會是我們寫程式的初衷，尤其是我們希望可以反覆的利用其他模組內所定義的型態與函數，而非反覆去執行其他模組所編寫的程式內容。可不是所有的Python檔案都是模組嗎？要如何讓Python直譯器知道什麼時候這是個執行程式，什麼時候被當作其他程式引入的模組，而非執行程式呢？

有個很簡單的方式，利用內建特殊的變數「__name__」。

```
if __name__ == “__main__”:
    main()
```

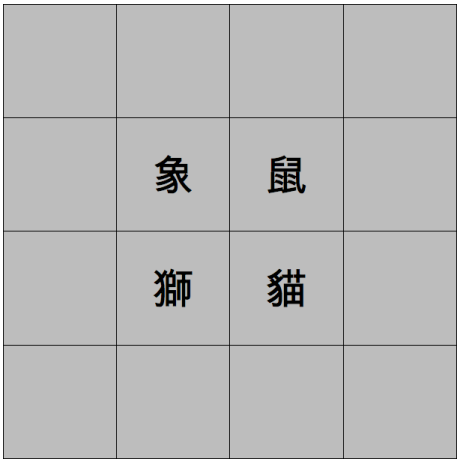
被執行的Python檔案，內建特殊的變數「__name__」會被儲存為字串「__main__」，因此藉由這個條件判斷，Python直譯器就能區別是否為執行程式，並且其他Python程式引入後不會被強迫執行。

我們在checker.py的最後加入上面兩行程式碼。

鬥獸棋的棋盤

我們在上一章中的鬥獸棋遊戲中，每一輪迴的開始顯示四種動物棋子的生存狀態，雖說是用一個for迴圈，實際上是印出四個字串，藉以顯示不同種類的動物棋子是否存活。棋類遊戲都有棋盤，我們現在來設想如何把棋盤呈現出來。

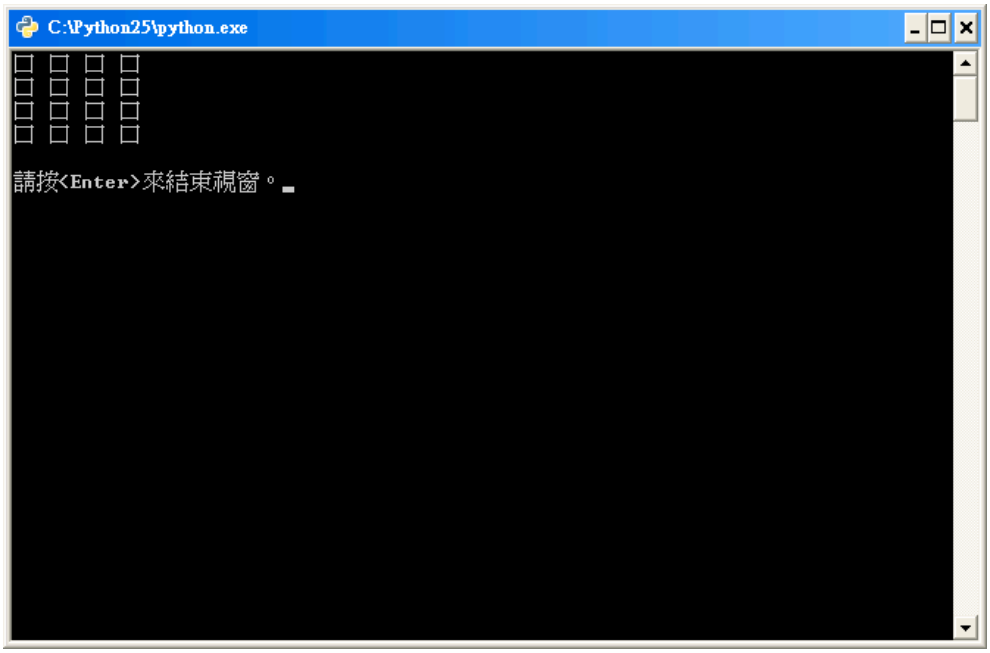
假設我們要用的是4×4的棋盤，這是說長寬各四格，然後把四個動物棋子放在中央的四格。



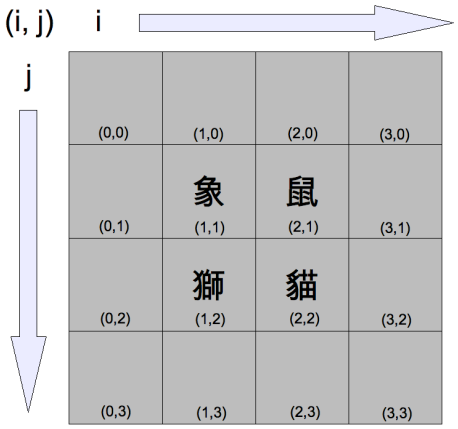
如果每一格都用一個「口」字來表示，我們就可以簡單的用兩個for迴圈印出來。

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

#印出棋盤的函數
def status(square):
    for j in range(square):
        for i in range(square):
            print "口",
        print

if __name__ == "__main__":
    status(4)
    print
    raw_input("請按<Enter>來結束視窗。")
```

我們將上述的程式碼先儲存到board.py中，測試如下。



如此一來，棋盤便可在命令列下顯示出來，我們便可以棋盤代替四種動物棋子的生存狀態。然後首先我們需要認識像是這樣的雙重迴圈，所作的動作便是印出表格，實際上表格的每一欄都可以用i與j的座標來表示。



對，就是一個座標系統，原點為左上方的第一格，j表示縱軸，i表示橫軸，這可能跟我們習慣的平面座標系統不太相同，不必擔心，我們只需要注意座標的對應方式。

把棋子放入棋盤

我們繼續來擴展board.py，當然，我們的目的是希望以這樣的顯示方式來取代生存狀態的顯示。但是如何顯示座標系統呢？對了，我們已經定義Point型態。

```
players = {"e":[Jungle("E",4), Point(1,1)], "t":[Jungle("T",4), Point(1,2)], \
           "c":[Jungle("C",4), Point(2,2)], "m":[Jungle("M",4), Point(2,1)]}
```

我們要給players每個key加入一個Point的value，與Jungle合起來放在串列中，因為key只能對應一個value，而我們又希望這個value可以被我們調整，所以用串列而非其他不可變的資料型態。

相同的，函數status我們也要稍作修改，

```
#印出棋盤的函數
def status(square):
    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
```

如果players存在key為“e”，接著才去找i及j相對的座標值，那我們要不要考慮Jungle的屬性alive呢？不，我們設想的更簡單一點，當一隻動物棋子被吃掉時，我們直接從players中移除掉這隻棋子。

這樣一來便能直接以players的長度大於1，主要遊戲迴圈就持續進行，而當players的長度等於1時，主要遊戲迴圈也就隨之結束。

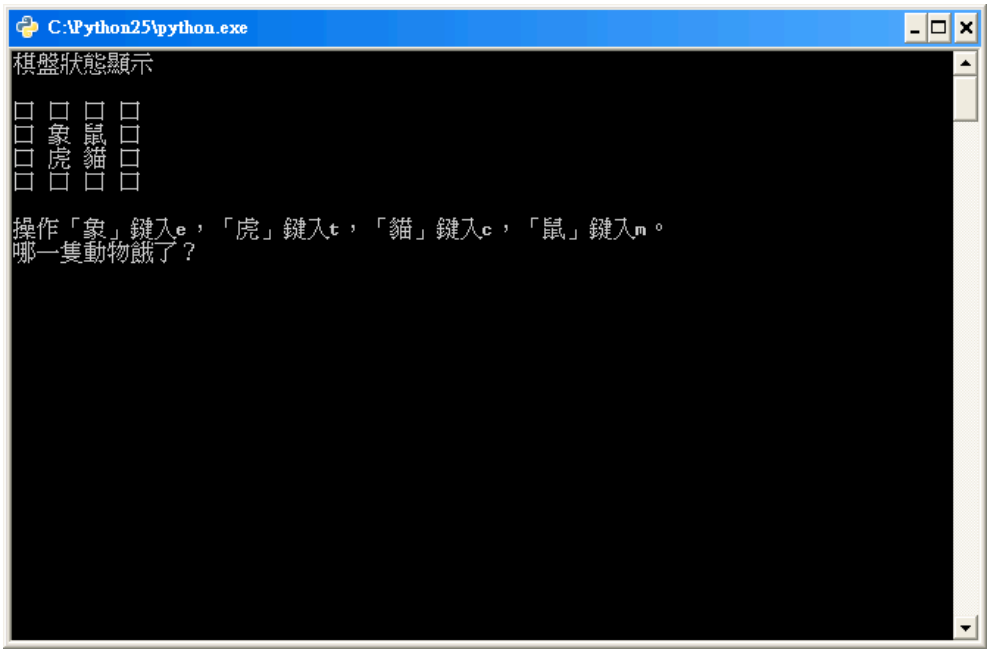
另外，我們將players移出主要遊戲迴圈，使其成為**全域變數**，如此程式中的各個函數都能進行利用。

我們將所有的程式碼稍做修改，列出如下。

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

from point import Point
from 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):
    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

def main():
```

```
        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。"
            first = raw_input("哪一隻動物餓了？ ") #這是會被儲存為self變數
            second = raw_input("要吃哪一隻？ ") #這是會被儲存為other變數

            #執行「棋子互吃的方法」，如果鍵入非預定的字母，會直接跳過進行下一輪
            if first in players.keys() and second in players.keys():
                players[first][0].capture(players[second][0])
                if not players[second][0].alive:
                    del players[second]

            #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__":
        main()
        print
        raw_input("請按<Enter>來結束視窗。")
```

執行結果如下。



Docstring

程式中的註釋已經可以提供許多資訊，Python還有另一種方式，可以讓使用者直接在程式碼中撰寫程式的文件，主旨用於說明模組、型態或函數的用途，這是以建立獨立存在的字串來進行的，其被稱為docstring。

單引號或雙引號字串都可用為docstring，不過Python還提供了另一種字串，連續用三個單引號或是雙引號圍起來的**三引號字串**，這在印出時會保留字串內原始編排格式，因此若有多行撰寫的需求，會是一種方便的選擇。

```
"""
Python是一種容易使用、學習的語言……

   Python is fun! Enjoying it!

隨著我們的進度一路走到這裡，你是否有同感呢？
"""
```

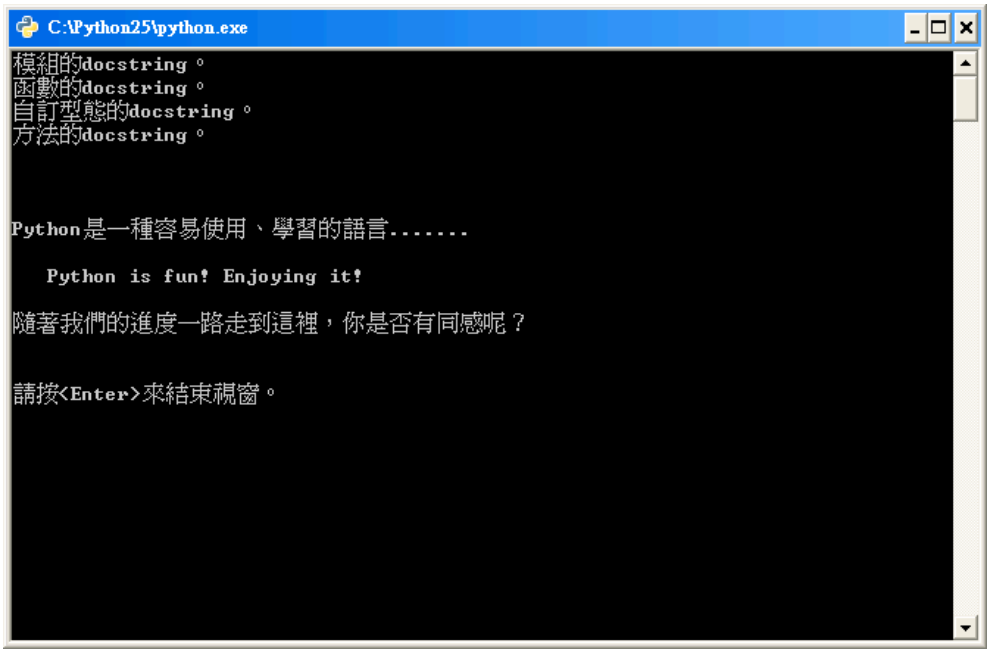
Docstring的寫法主要分為兩種，其一為單行的方式，內容簡單扼要的交代模組、型態或函數的功能，另一種則是多行，首行與單行的方式類似，作為整體的摘要，下方空一行，然後接續的以例子來說明功能。我們以單行的方式為例，說明docstring 在程式裡的位置，如以下的程式。

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

"""模組的docstring。"""

temp="""
Python是一種容易使用、學習的語言.......

   Python is fun! Enjoying it!

隨著我們的進度一路走到這裡，你是否有同感呢？
"""

def testfunction():
    """函數的docstring。"""
    pass

class testclass(object):
    """自訂型態的docstring。"""

    def testmethod():
        """方法的docstring。"""

        pass
```

內建的特殊屬性__doc__，可用來存取docstring，我們以另一個程式來印出上面的docstring。

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

import testfile

if __name__ == "__main__":
    print testfile.__doc__
    print testfile.testfunction.__doc__
    t = testfile.testclass()
    print t.__doc__
    print t.testmethod.__doc__
    print
    print
    print testfile.temp
    print
    raw_input("請按<Enter>來結束視窗。")
```

執行結果如下。



Note

有關docstring的慣例，詳情可參考PEP257。

風格指南

寫程式只要合乎語法，同時程式也能完成預期的工作，要用如何的風格鋪陳程式碼，這其實是很自由的，每個人的喜好與習慣或多或少有所不同。雖說如此，撰寫Python程式仍希望以易讀為優先考量，這樣程式才跟著容易被理解。

可讀性不但讓人容易理解，同時也容易進行維護，尤其開發工作往往由團隊來進行，也可能有後來才加入的成員，若能將程式碼的撰寫風格包持一致，成員間的溝通才不會造成困難。因此我們綜合一些Python程式的撰寫習慣，對程式的風格提供一些建議。

我們先來看看程式的布局。

```
(1)編碼宣告                (1)#-*- coding: UTF-8 -*-
(2)模組的docstring         (2)"""這是模組的說明文件。"""
(3)引入模組                (3)import something
(4)全域變數的建立          (4)variable = value
(5)型態定義                (5)class type:
(6)函數定義                (6)def function:
(7)程式執行的主體          (7)if __name__ == "__main__":
```

型態與函數的定義內也都應該包含docstring，位置就放在緊接著class及def關鍵字的下一行。其他應該注意的包括適當加入註解，慎選自行定義的名稱，還有要統一縮排的方式，tab鍵可作參考，然而我們建議四格會比較恰當。

大體上到目前為止，我們所有的程式都依循這樣的布局，以供參考。

Note

如果是UNIX的使用者，第一部份除了編碼宣告，建議加入UNIX系統所屬的啟動行，「#!/usr/bin/env python」，這可以讓程式在UNIX系統下只以名稱來執行。

Note

有關Python程式碼風格指南的官方建議，詳情可參考PEP8。

Be Pythonic

我們可以依據所處理的問題，透過函數、型態到模組等不的介面設計，從而建立一些有用的程式，當探索Python的世界越深，我們同時也都應該越來越Pythonic。

Pythonic，形容詞，意旨像是具有Python語言經驗的人。每一種受歡迎的語言背後都有龐大的社群支持，社群中不乏經驗老手，他們對語言的熱忱與執著，正是推動該語言有所進步的動力。

因而社群裡存在對呈現語言的共識，這些共識久而久之形成程式的慣例寫法，隨著語言實務經驗的增加，我們同時也應逐步熟悉這些習慣，熟悉了社群中息以為常的方式，不論往後自己撰寫程式或是閱讀別人寫的程式，不會因為習慣的不同而造成一時間難以接受。

所以，Be Pythonic，隨著進一步體會與了解Python語言的特性，同時發揮這些特性的優點，這會帶給我們寫程式更多的樂趣。