第四章 資訊隱藏

語言對實物所表達的常常是代稱，而且這樣的代稱是經年累月的習慣，如「桌子」之所以稱為桌子，我們實在很難找出當初為什麼要用「桌子ㄓㄨㄛ ㄗˇ」，兩個字拼寫在一起。然而在語言的使用上，桌子就是桌子，沒有什麼特別的。

代稱可以說是比擬，一種**抽象**的比擬，這就不難理解為什麼英文稱桌子為「table」，發音與中文大相逕庭，完完全全的不同，可是都表示同一個意含。自然語言是如此，大體上全都反應到我們生活各種的活動，路標常常是大大的圖示，有些提醒司機，有些則提醒路人，主要的目的是讓我們立即明白圖示透露的資訊。

設計物品的設計圖也用了不少圖示，這樣的圓形表示這種元件，那樣的方形則是代表另一種元件，圖形背後所顯示的是我們已經很習慣，並且很自然的做**抽象化**的思考。抽象化的目的在於讓我們集中焦點在同一階層的概念，無須考慮建立此一階層的實體建置，如設計圖中圖形所表示的元件，重點是我們要如何把各種元件組合起來，而非在於元件是用什麼材料構成的。

同樣的，程式設計時利用抽象化利於我們專心在問題解決的途徑，這樣的方式就是把資訊隱藏起來，然後透過我們設計出的介面，作為程式內各種資訊溝通的方式。上一章的函數就是介面的其中之一，我們透過函數名稱的設計，使我們輕易了解函數的功能，然後透過函數呼叫從而利用函數。

當我們把數據及資料逐一封裝進入到函數，我們已經在做資訊隱藏的工作。雖然函數已經可以做很多的事情，畢竟函數並非全能，有些事情仍需借助其他的介面來幫忙，譬如儲存資料，我們能夠利用串列來儲存一長串動物名稱的列表，串列正是一種能夠儲存資料的型態。

複合資料型態的主要功能便是儲存資料，然而這樣的介面限於運用在既有的資料型態上，假如我們要處理向量，而向量是用座標系統中的點來表示，同時我們希望直接對向量進行運算，就跟我們在數學上所用的方法一樣，這要怎麼做呢？

因為在數學上，二維空間的向量是用小括弧圍起來的兩個數字，中間用逗點隔開，又能夠進行兩個向量的相加、相乘或是乘以常數，其結果仍是向量。我們學過的複合資料型態中字串、串列或是序對都是不行的，因為相加與相乘都會造成重複的向量，而非我們預期的結果。

```
>>> "(1, 1)" + "(2, 2)"
'(1, 1)(2, 2)'
>>> [1, 1] + [2, 2]
[1, 1, 2, 2]
>>> (1, 1) + (2, 2)
(1, 1, 2, 2)
```

雖然我們仍是可以用變數來操作，這卻顯得太過繁瑣。

```
>>> x1, y1 = 1, 1
>>> x2, y2 = 2, 2
>>> x3, y3 = x1+x2, y1+y2
>>> print "(", x3, ",", y3, ")"
( 3 , 3 )
```

漏打標點或是任一個變數，許多輸入就得重來。那有沒有其他的型態可以用為向量運算呢？有的，我們可以自行定義新的型態，以符合我們的需求。

作者：張凱慶，Email：kaichingc@gmail.com

第二步：自訂資料型態

自訂資料型態是我們所要引入介面設計的第二步，透過自訂資料型態的介面，我們會希望平面座標的向量會像是如下的方式運作。

```
>>> p1 = Point(1,1)
>>> p2 = Point(2,2)
>>> p3 = p1 + p2
>>> print p3
(3, 3)
```

我們利用類似函數呼叫的方式建立新的資料型態，此時變數p1及p2分別儲存了向量(1,1)及(2,2)，然後我們將p1與p2相加的結果儲存到p3，最後印出p3。我們先來看看p3屬於什麼型態。

```
>>> type(p3)
<type 'instance'>
```

型態instance的中文為「實例」，凡是由自訂資料型態建立的物件都屬於**實例**型態。我們再來看看Point是屬於哪一種型態。

```
>>> type(Point)
<type 'classobj'>
```

obj是object的簡寫，所以classobj的意思是指由class關鍵字所建立的物件型態。的確，我們要建立自訂資料型態都要由class關鍵字來著手。

```
>>> class new(object):
        attribute=0


>>> new_instance=new()
>>> new_instance.attribute
0
```

如上我們定義了一個new的物件型態，在class底下所被指派的變數被稱為**屬性**，這是說屬性是型態所擁有的預設數值，接著我們用new_instance建立這個new型態的變數，然後用**小數點記號**讀出attribute屬性。

Note

Class本身就是種類之意，許多英文書籍作者習慣稱他所建立新的物件種類為class，因而class除了用作關鍵字，在物件導向程式設計又帶有另一層含意，以致許多中文書籍的作者直稱class為「類別」。這裡，我們要強調由class關鍵字所建立的是新的「型態」，所以我們不用「類別」之詞。

我們也可以在自訂資料型態內定義函數，不過這裡的函數被稱為**方法**，同樣利用def關鍵字。

```
>>> class new(object):
        attribute=0

        def say_hello(self):
                print "Hello, I'm new class."


>>> new_instance=new()
>>> new_instance.attribute
0
>>> new_instance.say_hello()
Hello, I'm new class.
```

在方法名稱的最後同樣要帶有小括弧，小括弧內至少要有self作為參數，這個self如同英文意思「本身」，因為通常我們都會用變數來建立新的型態，self所指的就是用來建立的變數。跟存取屬性一樣，方法要利用小數點記號進行呼叫。

型態的初始化

如果我們想要建立自訂型態的變數，同時能夠賦予一些屬性值給該變數，而非像上面完全一樣的屬性值，這時候可以運用Python特殊設計的__init__()方法。

> **Note**
> 注意，__init__()是前後連續兩個底線圍住init。

譬如Point型態，我們希望具有x屬性及y屬性，分別代表x座標及y座標。

```
>>> class Point(object):
        def __init__(self, x=0, y=0):
                self.x = x
                self.y = y


>>> p1=Point(1,1)
>>> print p1.x
1
>>> print p1
<__main__.Point object at 0x00D2E1D0>
```

> **Note**
> __init__()小括弧中的參數x=0與y=0，這是預設的參數值，假如建立物件時沒有賦予參數，就會將預設值代入。

雖然我們已經利用變數p1建立了Point型態的物件，同時將p1的x及y座標分別設為1，然而我們並沒有定義Point型態的物件如何在print陳述中印出，所以最後Python shell告訴我們的是變數p1在記憶體中的位置。

於是我們需要運用另一個特殊的__str__()方法，使得Point型態的物件可以由print陳述來印出，同時也能夠符合我們所希望的格式。

```
#定義表示向量的型態
class Point(object):
    #初始化
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    #字串輸出
    def __str__(self):
        return "(" + str(self.x) + ", " + str(self.y) + ")"
```

測試結果如下。

```
>>> p1=Point(1,1)
>>> print p1
(1, 1)
>>> p2=Point(2,2)
>>> print p2
(2, 2)
```

運算子多載

二維空間的向量所能做的計算包括相加、相減、係數積與點積，我們希望自訂的Point型態能夠運用+、-、*等三個符號進行計算，這時候就要用到**運算子多載**。

運算子多載的意思是讓原有用途的運算子，能夠在其他型態中搭載擬定的計算方式。如加號原是用於數字的相加，而在字串或串列則用於連接，這便是運算子多載的一例。同樣的，假設我們要在Point型態加進+、-、*等三個運算子，用作向量的相加、相減、係數積與點積等四種計算，我們則需另外定義__add__()、__sub__()、__mul__()與__rmul__()，這四個特殊的方法。

```
#定義表示向量的型態
class Point(object):
    #初始化
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    #字串輸出
    def __str__(self):
        return "(" + str(self.x) + ", " + str(self.y) + ")"

    #向量加法
    def __add__(self, other):
        return Point(self.x+other.x, self.y+other.y)

    #向量減法
    def __sub__(self, other):
        return Point(self.x-other.x, self.y-other.y)

    #向量的點積
    def __mul__(self, other):
        return self.x*other.x + self.y*other.y

    #向量的係數積
    def __rmul__(self, other):
        return Point(other*self.x, other*self.y)
```

測試如下。

```
>>> ================================ RESTART ================================
>>>
>>> p1=Point(1,1)
>>> p2=Point(2,2)
>>> print p1+p2 #加法
(3, 3)
>>> print p1-p2 #減法
(-1, -1)
>>> print p1*p2 #點積
4
>>> print 6*p1 #係數積
(6, 6)
```

第三個複合資料型態：字典

字串用於儲存文字，串列用於儲存多筆可以索引的資料，Python還有另一種屬於配對的資料型態，這是用大括弧圍起來的**字典**。

```
>>> my_fruit={"banana":"香蕉", "lichee":"荔枝"}
```

字典是屬於key-value互相配對的資料型態，key中文為鑰匙，value為數值，這種配對是由鑰匙來存取所儲存的數值，其比喻如鎖匙的關係，一把鎖只有一隻鑰匙能打開。因此冒號前為key，必須是不可變的資料型態，冒號後為value，可以是任意的資料型態，包括字典本身。

字典因為是配對的資料型態，key就像是串列中的索引值，我們存取資料要透過key，致使資料在字典中的順序變得沒有那麼重要，這是跟串列最主要的不同。

```
>>> fruit_one=my_fruit["banana"]
>>> print fruit_one
香蕉
>>> print my_fruit
{'lichee': '\xaf\xef\xaaK', 'banana': '\xad\xbb\xbf\xbc'}
```

如果我們要在字典中增加資料項目，也是要同時增加key與value。

```
>>> my_fruit["watermelon"]="西瓜"
>>> print my_fruit
{'lichee': '\xaf\xef\xaaK', 'watermelon': '\xa6\xe8\xa5\xca', 'banana': '\xad\xb
b\xbf\xbc'}
```

其實複合資料型態中都有些方法可以運用。

```
>>> my_fruit.keys()
['lichee', 'watermelon', 'banana']
>>> my_fruit.values()
['\xaf\xef\xaaK', '\xa6\xe8\xa5\xca', '\xad\xbb\xbf\xbc']
>>> my_fruit.has_key("banana")
True
>>> my_fruit.has_key("apple")
False
```

keys()讀出my_fruit中所有的key，values()讀出所有的value，而has_key()則帶有參數，判斷參數是否為my_fruit的key之一。

Note

keys()、values()、has_key()只是字典中所能應用的眾多方法的三種，關於字典型態的詳細說明，可參考Python Library Reference中的Mapping Types。

利用字典與所屬的方法很多時候可以帶給我們一些不同的思考，例如上一章中所提計算費伯那西數列採用的遞迴方法，若是以字典作為儲存的資料結構，反而不會拖慢計算速度。

```
#利用字典儲存費伯納西數列
f={0:0,1:1}

#計算費伯納西數列的遞迴函數
def fi_recursion(n):
    if f.has_key(n):
        return f[n]
    else:
        new = fi_recursion(n-1) + fi_recursion(n-2)
        f[n] = new #將求出的值存放入字典中
        return new
```

測試結果如下。

```
>>> ================================ RESTART ================================
>>>
>>> fi_recursion(46)
1836311903
>>> print f
{0: 0, 1: 1, 2: 1, 3: 2, 4: 3, 5: 5, 6: 8, 7: 13, 8: 21, 9: 34, 10: 55, 11: 89,
12: 144, 13: 233, 14: 377, 15: 610, 16: 987, 17: 1597, 18: 2584, 19: 4181, 20: 6
765, 21: 10946, 22: 17711, 23: 28657, 24: 46368, 25: 75025, 26: 121393, 27: 1964
18, 28: 317811, 29: 514229, 30: 832040, 31: 1346269, 32: 2178309, 33: 3524578, 3
4: 5702887, 35: 9227465, 36: 14930352, 37: 24157817, 38: 39088169, 39: 63245986,
 40: 102334155, 41: 165580141, 42: 267914296, 43: 433494437, 44: 701408733, 45:
1134903170, 46: 1836311903}
```

Note

字串、串列也都有眾多方法可供應用，關於字串型態的詳細說明，可參考Python Library Reference中的String Methods，而串列型態的詳細說明，則可參考Mutable Sequence Types。

來玩鬥獸棋

電影《九品芝麻官》中皇帝在妓院因為怕被協理大臣發現，躲進床下後撞見豹頭與包龍星，皇帝機智的用鬥獸棋來解決當下的困境：「有沒有玩過鬥獸棋？象吃老虎，老虎吃貓，貓吃老鼠，老鼠可以吃象。老鼠，你懂怎麼做吧？」

鬥獸棋是個有趣的遊戲，雙方各有象、獅、虎、豹、狼、狗、貓、鼠等八隻棋子，依序大小，大可以吃小，同類也可以互吃，最小的鼠則可以吃最大的象，棋盤中又有小河、陷阱等特殊的地形，只要佔據對手的獸穴即可得勝。

我們來想想怎麼用Python來寫這個遊戲，遊戲規則有點複雜，不怕，我們試作一個簡化的版本當作練習。暫不考慮棋盤與棋子的移動，我們把焦點放在大小互吃的關係上，首先，我們需要一個型態來表示「棋子」，對了，我們自己可以設計這個型態。

```
#設定棋子的型態
class checker(object):
    def __init__(self):
        self.alive = True #棋子的初始條件

    def dead(self):
        self.alive = False #棋子被吃掉
```

棋子仍在，屬性alive為真，若是被吃掉，運用dead方法使alive轉為假，藉此我們可以判斷這一隻棋子是否仍存活。

```
>>> ================================ RESTART ================================
>>>
>>> a=checker()
>>> a.alive
True
>>> a.dead()
>>> a.alive
False
```

我們利用這個型態來操縱棋子的生死，接下來，便是將動物的特性放到棋子之中。

老鼠的兒子會打洞

雖然我們可以繼續修改checker型態，使其符合鬥獸棋中各種動物的特性，然而checker型態像是棋類遊戲中的基本單元，幾乎各種棋類遊戲都可適用。於是我們不妨設計另一個型態來專門囊括鬥獸棋的遊戲規則，同時讓這個型態**繼承**checker型態。

我們需要設計另一個__init__()方法，增加辨識動物種類的屬性，還要讓每種動物曉得自己的食物是哪種動物，另外要有__str__()方法，用來印出狀態，最後，「吃」的方法也要一併設計出來。

```
#...省略class Checker(object)...
#鬥獸棋的遊戲型態
class Jungle(Checker):
    def __init__(self, name):
        Checker.__init__(self) #繼承自checker的__init__()

        animal = {"E":"象", "T":"虎", "C":"貓", "M":"鼠"} #棋子種類
        food = {"E":["象", "虎", "貓"],"T":["虎", "貓", "鼠"],"C":["貓", "鼠"], "M":["象", "鼠"]} #食物鏈關係

        if animal.has_key(name):
            self.name = animal[name]
            self.food = food[name]
        else:
            print "沒有這種動物喔！"
            self.name = False

    def __str__(self):
        if self.alive and self.name:
            return "我是「%s」。" %self.name
        else:
            return "「%s」被吃掉了。" %self.name

    #棋子互吃的方法
    def capture(self, other):
        if self.name == other.name:
            print self.name, "不能自己吃自己！"
        else:
            if self.alive and other.alive:
                if other.name in self.food:
                    print self.name, "吃", other.name
                    other.dead()
                else:
                    print self.name, "不能吃", other.name
            elif not self.alive:
                print self.name, "已經死掉了！"
            elif not other.alive:
                print other.name, "已經死掉了！"
            else:
                print self.name, "和", other.name, "都已經死了！"
```

Note

所以從new、Point到現在的Checker型態，其後小括弧中的object，這也是繼承自預設的object型態。

jungle型態後的小括弧中的checker，便表示jungle繼承自checker，這是說checker裡設計的方法在jungle也能施用，但是jungle的__init__()會覆載checker的__init__()，這是因為方法名稱相同。於是我們要額外放入checker.__init__(self)的陳述，使得jungle的__init__()能繼承checker的__init__()。

我們使jungle的初始化就帶有一個參數name，好讓我們可以指定動物的種類，以及該種動物能吃哪些其他的動物，於是我們需要兩筆額外的資料，一個儲存動物名稱，另一個則是儲存食物種類。

鬥獸棋共有八種動物，我們簡單一點只採其中四種。那要用什麼型態作為儲存這兩筆額外資廖的結構呢？沒錯，儲存資料的型態就是字典，配對的方式使我們只檢查一個字母就能將兩比資料分別設定為屬性。

當然，假如輸入的字母不存在於animal或food中，我們必須顯示錯誤的訊息。

這裡的__str__()方法有點不一樣，我們採用**字串格式輸出**，百分比符號%除了用作取餘數的運算子外，也用作格式輸出的符號。%s表示格式的為字串，而self.name的格式也為字串，這種方式的好處是變數能安插在字串中，當作字串的一部分。

Note

關於字串格式輸出的詳細說明，參考Python Library Reference的String Formatting Operation。

最後一個capture則是「吃」的方法，這裡要考慮的情況比較多，主要考慮吃的動物以及被吃得動物各自的生存狀況，當然，我們不會希望已經被吞進肚子裡的動物還能張大嘴吃其他的動物，我們也不希望動物自己吃自己。於是我們利用了許多巢狀結構做條件檢查，符合條件的情況下，只要other是self的食物，我們便呼叫dead()方法，於是other的生存狀態由真變為假。如果不是，當然，我們也要提供訊息。

我們來玩玩看吧！

```
>>> ================================ RESTART ================================
>>>
>>> e, t, c, m = Jungle("E"), Jungle("T"), Jungle("C"), Jungle("M")
>>> t.capture(c)
虎 吃 貓
>>> e.capture(t)
象 吃 虎
>>> m.capture(e)
鼠 吃 象
>>> print e, t, c, m
「象」被吃掉了。 「虎」被吃掉了。 「貓」被吃掉了。 我是「鼠」。
```

Note

「繼承」是物件導向程式設計中一個重要也是核心的觀念，動詞原文為inherit。雖然inherit在英文的意思泛指從什麼得到什麼，用作中文「繼承」說得通，也能用作「遺傳」。然而中文的「繼承」隱含某物不再，另物將起的意思，譬如「我繼承某某的精神」，雖然某某不見得已死，未來將要付出努力的卻是我而非某某。因而這裡的意思中文用「遺傳」比較恰當，子代會從親代遺傳某些生物特性，子代與親代也會並存一段時間，這就沒有某物不再的意含了。然而這裡我們仍沿用程式設計常用的「繼承」一詞，但仍提出意見以免讀者混淆。

主要遊戲迴圈

一個一個的設變數，然後一個吃一個，然後另一個再吃一個，有點麻煩，不是嗎？正因最後活著的動物是贏家，我們可以想的簡單一點，遊戲開始的時候有四隻動物，結束的條件只剩下一隻動物，所以我們可以用迴圈來輔助記錄現存的動物隻數，到只存活一隻時便跳出迴圈，這也代表了遊戲的結束。

同樣的，我們直接拿一個main()函數來寫，方便等一下的測試。整個main()函數會分成三個主要部份，分別是「初始條件設定」、「主要遊戲迴圈」以及「印出結果（遊戲勝利者）」。先來看看「初始條件設定」部份的程式碼。

```
#...省略class Checker(object)...
#...省略class Jungle(object)...
#執行遊戲的函數
def main():

    #初始條件設定
    #參與遊戲的動物棋子
    players = {"e":Jungle("E"), "t":Jungle("T"), "c":Jungle("C"), "m":Jungle("M")}
    #總存活數的設定
    lives = len(players)
```

我們仍是用字典作為參與遊戲棋子的儲存結構，分別以英文的小寫字母為key，這方便進行遊戲時我們按鍵操作之用。內建函數len(players)回傳players的長度到變數lives之中，長度為4，也就是遊戲開始的時候分別有象、虎、貓、鼠四隻動物棋子。

```
#...省略class Checker(object)...
#...省略class Jungle(object)...
#...省略def main()...
#...省略初始條件設定...

    #主要遊戲迴圈
    while lives > 1:
        #印出每隻動物的存活訊息
        for player in sorted(players.values()):
            print player

        #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].capture(players[second])
            if not players[second].alive:
                lives = lives - 1

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

主要遊戲迴圈就是當變數lives大於1時所進行的迴圈，只要任何一隻動物被吃掉，lives就會遞減。我們將由遊戲迴圈分成四個部份，第一個部份印出所有動物的狀況，players.values()會將字典players中所有的value轉換成一個串列，內建的sorted()函數則會將這個串列排序，使得每一輪印出的順序都一樣。

第二個部份則是印出操作提示，先印出一個空白行，然後印出提示我們操作那一隻動物按哪一個小寫字母的按鍵，接下來便是給我們按下按鍵，變數first用為Jungle型態定義中的self，而變數second則用為other。

第三個部份執行「棋子互吃的方法」，也就是Jungle型態定義中的capture(self)方法，我們先做一個條件檢查，如果鍵入的小寫字母是四種動物之一，也就是在字典players裡的key之中，才會進行互吃的方法，有動物棋子被吃，變數lives就會遞減。

如果不是，跳過直接印出間隔線，也就是第四個部份，這就到了迴圈的最後，然後就是下一輪的開始。

```
#...省略class Checker(object)...
#...省略class Jungle(object)...
#...省略def main()...
#...省略初始條件設定...
#...省略主要遊戲迴圈...

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

當lives等於1時，主要遊戲迴圈隨之結束，然後我們用另一個迴圈找出遊戲的勝利者，當然，最後只會有一隻動物屬性alive為真，於是我們印出結果。

來玩玩看吧！「象」能吃「鼠」嗎？

```
>>> ================================ RESTART ================================
>>>
>>> main()
我是「貓」。
我是「鼠」。
我是「虎」。
我是「象」。

操作「象」鍵入e，「虎」鍵入t，「貓」鍵入c，「鼠」鍵入m。
哪一隻動物餓了？ e
要吃哪一隻？ m
象 不能吃 鼠
**************************************************
```

答案符合遊戲規則。那「鼠」能吃「鼠」嗎？

```
我是「貓」。
我是「鼠」。
我是「虎」。
我是「象」。

操作「象」鍵入e，「虎」鍵入t，「貓」鍵入c，「鼠」鍵入m。
哪一隻動物餓了？ m
要吃哪一隻？ m
鼠 不能自己吃自己！
**************************************************
```

很好，仍是符合遊戲規則。我們接下來讓「虎」吃「貓」，貓就不能捉老鼠，「象」吃「虎」，最後，符合規則就是「鼠」吃掉「象」。

好不好玩呢？當然，這場生存遊戲是由我們自行控制，不久的將來，我們再來試看看讓電腦自己跑出結果來。