第貳篇 跨過門檻

第九章 遊戲的圖形顯示

Joel Spolsky提到「如果程式的行為與使用者的期望完全一致，就是一個使用良好的設計介面。」的確，良好的介面設計讓使用者容易操作，不需多費時間學習，然而設計不良的介面令人猜不透程式的想法，初次體驗一旦摸不著頭緒，即便功能強大的程式，也往往被人棄之如敝屣。

當然，這裡所講的介面是使用者與軟體互動溝通的管道。第三章所提的「介面」則是我們自己寫程式時去利用其他已經寫好的程式碼，內建或自訂的函數、型態、模組都是透過「介面」，便於程式的開發。其實道理都是一樣的，優良的使用者介面讓人易於發揮軟體的功能，好的程式介面則可縮短開發時間，便於重複利用。

可是設計使用者介面卻不太簡單，很多時候像是藝術創作，不同的顏色組合、排版格局往往給人不同的感受，另一方面，難以理解的程式介面也常常陷入修改困難，預期效果無法達成，從而拖慢開發的速度。Python語言由廣大的社群支援，標準模組庫也是由經驗純熟的程式設計師反覆測試，然後才放入到Pyhton安裝套件中，供我們直接利用。

除了標準模組庫之外，Python還有許多廣受歡迎的第三方模組庫可供應用，利用Python簡單、易用的特性，讓程式開發的工作更為輕鬆、容易。我們在這一章開始介紹其中一個第三方模組庫----Pygame，這是Python從SDL延伸發展的模組庫，專門為開發遊戲之用。

SDL是Simple Directmedia Layer的縮寫，最初是以C語言撰寫的跨平台多媒體開發函數庫，與DirectX類似，然而其以精簡方式完成許多控制聲音、影像的基礎工作，大幅簡化所需撰寫的程式碼，使開發工作更為容易。

正因如此，高階的繪圖需要與OpenGL結合，OpenGL是一種在電腦工業普遍應用的3D繪圖介面，廣泛應用在遊戲開發、虛擬實境、科學視覺化、電腦輔助設計等。也因為SDL採用LGPL授權，所以商業遊戲也能利用並進行開發，著名如《毀滅戰士3》、《雷神之鎚4》都是利用SDL進行開發的。

由於Pygame是模組庫，因此裡頭包含許多模組，以套件的方式組織，引入時可以只簡單的寫：

```
import pygame
```

然而，我們需要利用小數點記號才能使用其內的模組，包括各式遊戲所需要的控制介面，下表為我們將會用到的模組。

模組名稱	描述
pygame.cursors	載入滑鼠游標的圖示。
pygame.display	控制顯示的視窗。
pygame.draw	基本的形狀繪圖。
pygame.event	事件管理。
pygame.font	載入TrueType字型。
pygame.image	載入圖形檔案。
pygame.key	鍵盤的控制。
pygame.mouse	滑鼠控制。
pygame.surface	圖形與視窗的型態。
pygame.time	管理時間與畫面更新率。

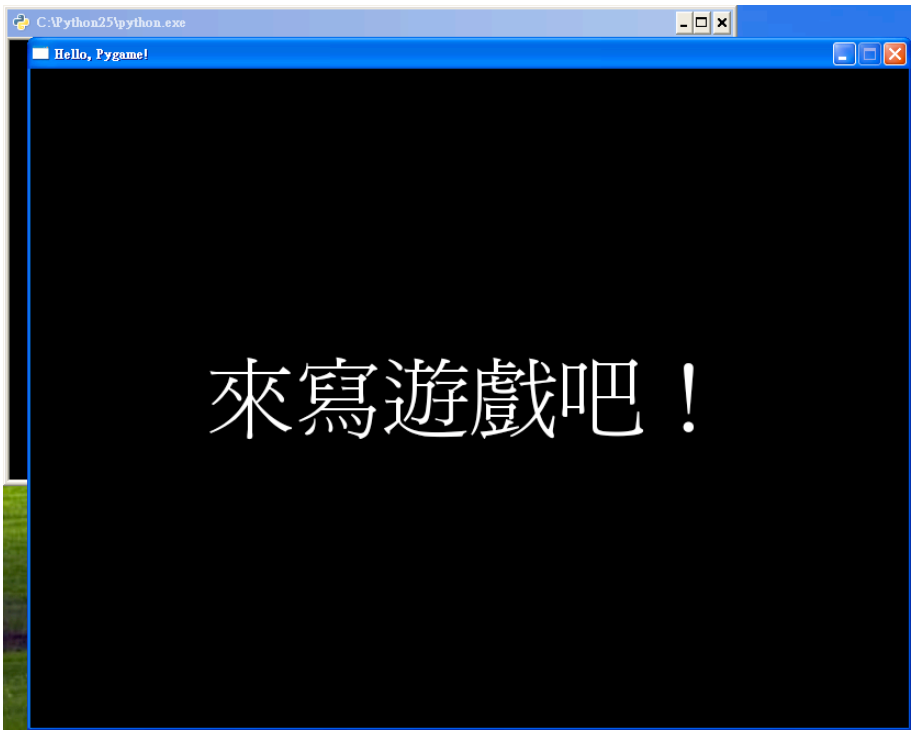
作者：張凱慶，Email：kaichingc@gmail.com

Note

引用Joel Spolsky這句話的原文在User Interface Design for Programmers的Chapter 1: Controlling Your Environment Makes You Happy，繁體中文翻譯在第一章：控制你的環境使你快樂。

第一個例子

下面是第一個例子在MS-Windows的顯示結果。



顯示結果出現兩個視窗，底下的視窗標題列為「C:\Python25\python.exe」，這也是上一篇中我們執行程式的命令提示字元的視窗，因為Python的預設是由命令列模式執行，所以會先出現命令提示字元的視窗，然後程式繼續執行，到直譯器看到要由Pygame建立新的視窗時，新的視窗就會出現。

因此執行的時候會在兩個模式下，其一為Python預設的命令列模式，其二就是Pygame建立的圖形模式。這個例子只有顯示「來寫遊戲吧！」的句子，程式碼如下。

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

import pygame
from pygame.locals import *
import os
from sys import exit

size = (800, 600)
black = (0, 0, 0)
white = (255, 255, 255)
title = "Hello, Pygame!"
chinese_message = "來寫遊戲吧！"
message = unicode(chinese_message, "big5")

def run():
    pygame.init()
```

```
    screen = pygame.display.set_mode(size, 0, 32)
    pygame.display.set_caption(title)

    font = pygame.font.Font(os.environ['SYSTEMROOT'] +
                                            "\\Fonts\\mingliu.ttc", 80)
    text = font.render(message, True, white)

    x = (size[0]-text.get_width()) / 2
    y = (size[1]-text.get_height()) / 2

    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                exit()

        screen.fill(black)
        screen.blit(text, (x, y))

        pygame.display.update()

if __name__ == "__main__":
    run()
```

程式碼中有很多新的東西，主要是關於圖形顯示的函數，別急，我們會慢慢來解釋。首先，我們先看到引入模組的部份。

```
import pygame
from pygame.locals import *
import os
from sys import exit
```

第一行及第二行是引入Pygame模組庫的部份，第三行及第四行則是引入標準模組庫的os與sys，其中sys模組只用到exit函數。接下來的六行程式碼則是一些用為常數的數值指派到變數裡，這是用英文的變數名稱代替數值，使程式容易了解。

座標與顏色

變數size的型態為序對，用來儲存指定視窗大小的數值。

```
size = (800, 600)
```

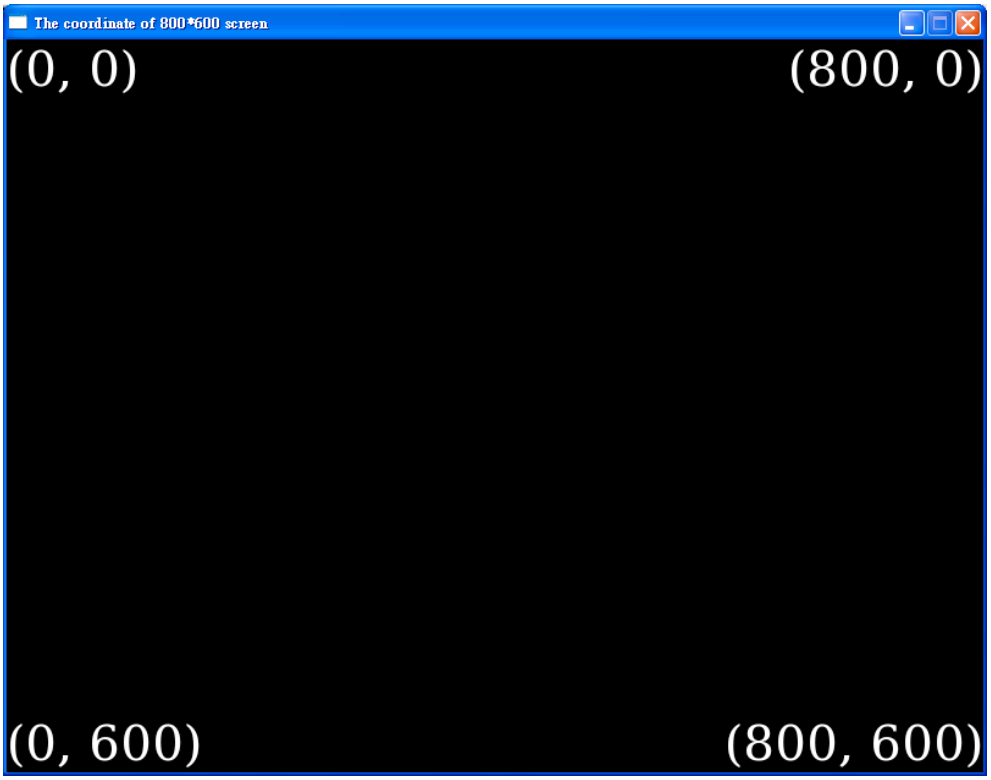
我們準備將視窗大小設定為800×600，被指派的size變數可以作為待會真正設置視窗函數的參數。這裡要注意一點，800×600雖說是視窗設定的解析度，但同時代表視窗的座標。第六章中的座標是每一個文字，「象」、「虎」、「貓」、「鼠」及「口」的位置，這裡所表示的視窗座標則是每一個畫素的位置。

以下的程式顯示四個邊角的座標值。

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

import pygame
from pygame.locals import *
from sys import exit

size = (800, 600)
p1 = (0, 0)
p2 = (800, 0)
p3 = (0, 600)
p4 = (800, 600)
title = "The coordinate of 800*600 screen"
black = (0, 0, 0)
```

```
white = (255, 255, 255)

def run():
    pygame.init()

    screen = pygame.display.set_mode(size, 0, 32)
    pygame.display.set_caption(title)

    font = pygame.font.SysFont("times", 40)
    text1 = font.render(str(p1), True, white)
    text2 = font.render(str(p2), True, white)
    text3 = font.render(str(p3), True, white)
    text4 = font.render(str(p4), True, white)

    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                exit()

        screen.fill(black)

        screen.blit(text1, (0, 0))
        screen.blit(text2, (p2[0]-text2.get_width(), 0))
        screen.blit(text3, (0, p3[1]-text3.get_height()))
        screen.blit(text4, (p4[0]-text4.get_width(), \
                                            p4[1]-text4.get_height()))

        pygame.display.update()

if __name__ == "__main__":
    run()
```

結果如下。



原點與我們鬥獸棋棋盤的座標相同，兩者都在左上角，x座標往右遞增，y座標則是往下遞增。程式內容不必擔心，因為沒有用到新的東西，所以當解釋完第一個例子時，也能夠理解這個程式。

變數black與white的型態也是序對。

```
black = (0, 0, 0)
white = (255, 255, 255)
```

這是用來表示顏色的RGB值，亦即紅綠藍三種顏色，分別用0到255的整數值表示色階，當三者都為0時就是黑色，而三者都為255時則為白色。常見顏色的RGB值如下表。

顏色名稱	紅色（R）	綠色（G）	藍色（B）	序對值	實際顏色
黑色	0	0	0	(0, 0, 0)	
湛藍	0	0	128	(0, 0, 128)	
綠色	0	128	0	(0, 128, 0)	
褐紅	128	0	0	(128, 0, 0)	
青綠	0	128	128	(0, 128, 128)	
紫色	128	0	128	(128, 0, 128)	
橄欖	128	128	0	(128, 128, 0)	
灰色	128	128	128	(128, 128, 128)	
紅色	255	0	0	(255, 0, 0)	
萊姆綠	0	255	0	(0, 255, 0)	
藍色	0	0	255	(0, 0, 255)	
銀灰	192	192	192	(192, 192, 192)	
紫紅	255	0	255	(255, 0, 255)	
黃色	255	255	0	(255, 255, 0)	
碧綠	0	255	255	(0, 255, 255)	
白色	255	255	255	(255, 255, 255)	

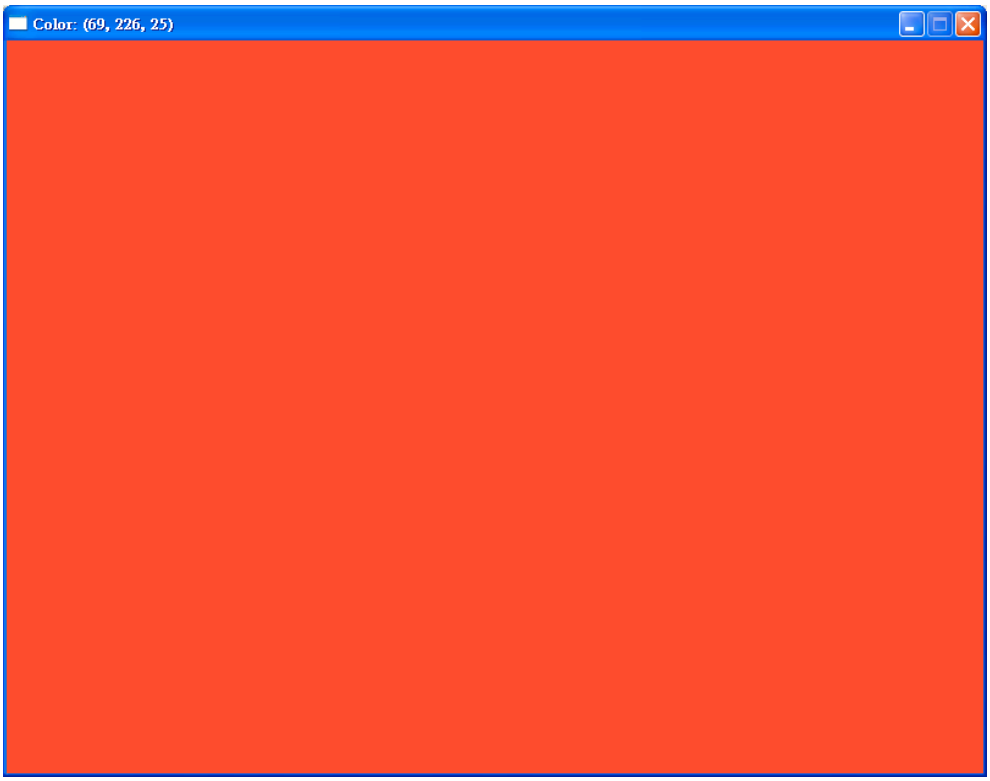
雖然很多顏色都有名稱，電腦實際卻可以演算出256×256×256=16777216種顏色，約一千六百多萬種顏色，然而並不是每一種顏色都有名稱，況且許多顏色的RGB值極為相近，我們未必能分辨的出來，因而知道所需的RGB值就已足夠。

以下的程式每秒隨機展示一種顏色。

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

import pygame
from pygame.locals import *
from sys import exit
from random import randint
from time import sleep

size = (800, 600)

def run():
    pygame.init()

    screen = pygame.display.set_mode(size, 0, 32)

    while True:
        for event in pygame.event.get():
```

```
            if event.type == QUIT:
                exit()

        color = (randint(0, 255), randint(0, 255), randint(0, 255))
        screen.fill(color)
        pygame.display.set_caption("Color: " + str(color))
        sleep(1)

        pygame.display.update()

if __name__ == "__main__":
    run()
```

標題列會顯示目前顏色的RGB值，以下是某一秒的結果。



變數title用為標題列的文字。

```
title = "Hello, Pygame!"
chinese_message = "來寫遊戲吧！"
message = unicode(chinese_message, "big5")
```

變數chinese_message是「來寫遊戲吧！」的文字字串，因為中文字串非ASCII字元，所以用內建unicode函數轉換編碼，然後儲存到變數message之中。

建立顯示的視窗

接下來，我們看到第一個例子中所定義的run函數，這是程式實際執行的函數。

```
def run():
    pygame.init()
```

首先要先對Pygame模組庫初始化，使其內的模組都能夠被利用。

然後建立一個screen變數。

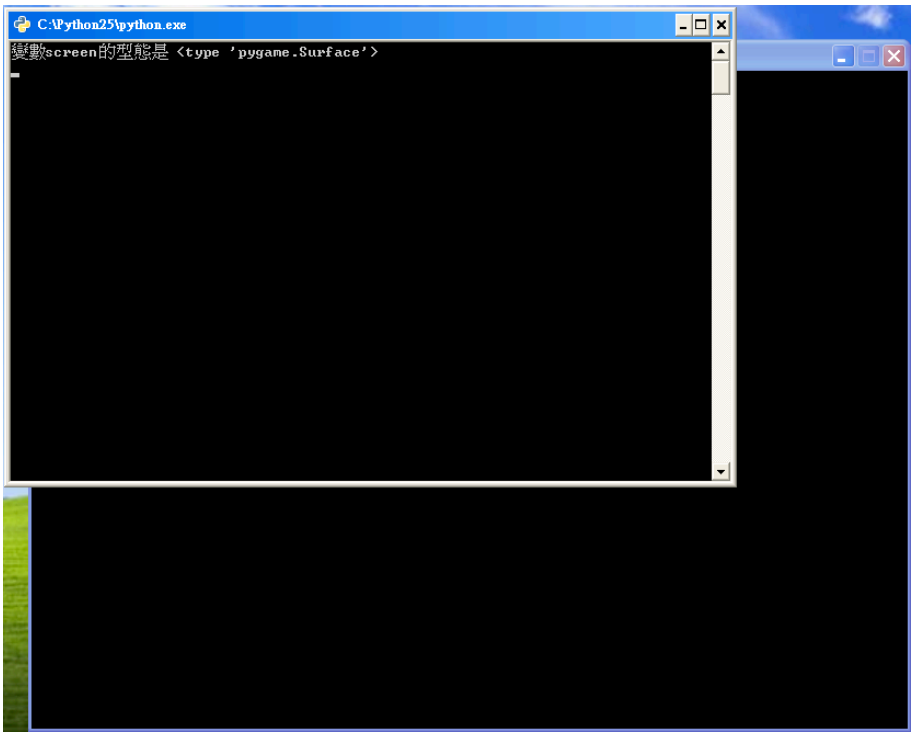
```
screen = pygame.display.set_mode(size, 0, 32)
```

這是利用display模組中的set_mode()函數，需要三個參數，第一個參數指定視窗大小，我們已經將(800, 600)的數值指派到變數size之中，第二個是有關顯示的特殊設定，我們暫時不會用到，因此設為0，第三個則是色彩位元數的設定，這裡我們設定為32位元。

如此我們就能得到一個顯示的視窗了，變數screen是什麼型態呢？我們可以加入下面這一行程式來看結果。

```
print unicode("變數screen的型態是","big5"), type(screen)
```

存檔後重新執行程式，因為print陳述只能在命令列上運作，所以我們要移回「命令提示字元」的視窗。



Surface是Pygame中用來控制顯示的型態，有關顯示顯示方面的操作都要透過Surface型態。建立完screen變數的下一行，set_caption()函數是用來設定標題列的顯示文字。

```
pygame.display.set_caption(title)
```

設定字型

我們在處理一般的打字工作時，文書處理軟體通常會自動載入系統預設的字型，除非美觀或其他需求，我們自己不需要另外設定字型，然而Pygame如同Python預設只認識ASCII字元，凡是中文文字或是全形符號都需要先轉換為Unicode字元，程式才不會印出亂碼。

所以我們如果要在surface物件上印出中文，就需要先建立變數font來指定載入的字型。

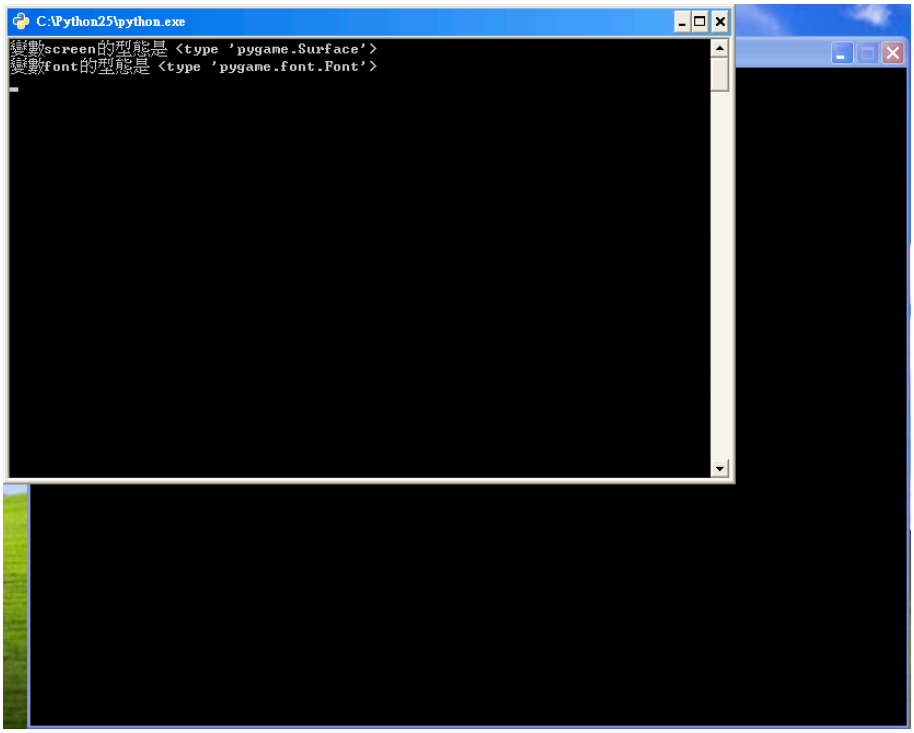
```
font = pygame.font.Font(os.environ['SYSTEMROOT'] +
                                        "\\Fonts\\mingliu.ttc", 80)
```

font模組是專門處理字型的模組，函數Font()則是用來載入字型，需要兩個參數，第一個參數型態為字串，其為字型名稱。我們用了一個技巧，os.environ['SYSTEMROOT']為系統路徑，我們接上Fonts目錄的mingliu.ttc字型檔案名稱，這是細明體的字型。第二個參數則是設定字型大小。

變數font的型態是什麼呢？我們一樣加上印出type()函數的程式碼。

```
print unicode("變數font的型態是", "big5"), type(font)
```

重新執行這個程式，我們看到「命令提示字元」的視窗。

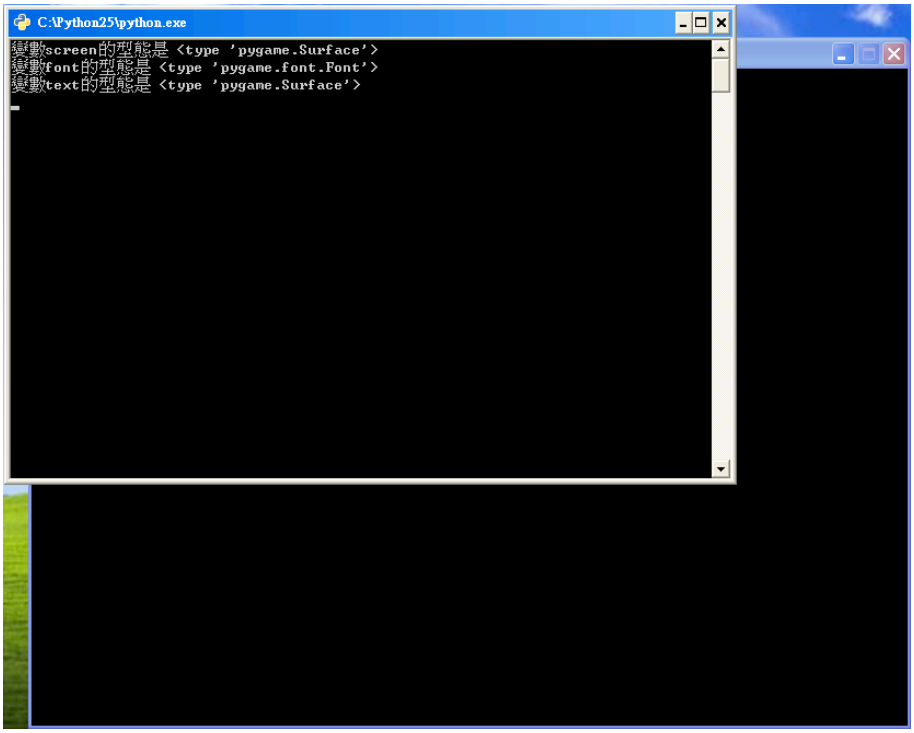


型態為Font，這是Pygame中處理文字顯示的型態。接下來，我們還需要指定顯示的文字及顏色。

```
text = font.render(message, True, white)
```

Font型態的render方法就是將文字著色於Surface物件之上，需要三個參數，第一個就是字串，我們已經把所要印出的文字儲存到變數message之中，第二個設為True，使文字不會印成斜體字，第三個參數則是指定顏色。

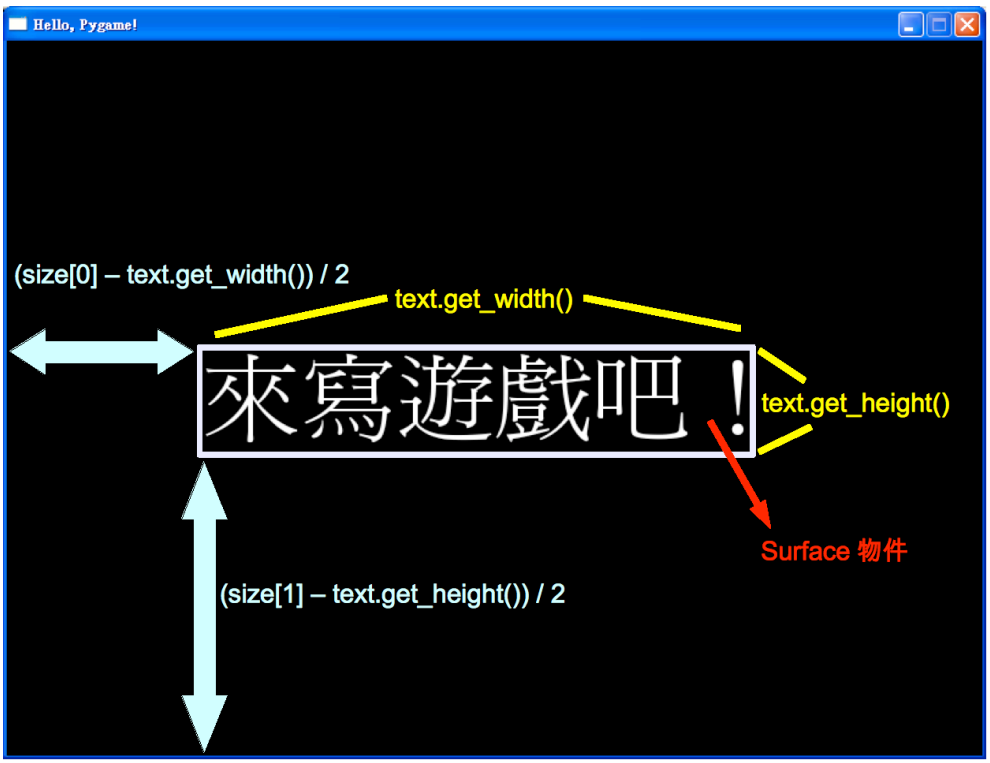
如此一來，變數text的型態會是Font還是Surface呢？我們永遠都可以請教Python直譯器。



型態轉變回Surface，這樣才能在Pygame的視窗中顯示出來。接下來的x、y計算出繪出文字的起始座標。

```
x = (size[0]-text.get_width()) / 2
y = (size[1]-text.get_height()) / 2
```

因為Surface型態是一種長方形的物件，而get_width()及get_height()方法分別取得該物件的寬及高，視窗的寬減去物件的寬，然後除以二得到x座標，高的情況也類似，如下圖。



事件處理

接下來我們看到遊戲的主要迴圈。

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()

    screen.fill(black)
    screen.blit(text, (x, y))

    pygame.display.update()
```

這裡有個重要的觀念，傳統的命令列模式只接受使用者從鍵盤輸入，當使用者按下鍵盤上的按鍵時，凡是可見字元螢幕立即印出該字元的符號，然而我們已經習以為常的圖形介面卻很不一樣。

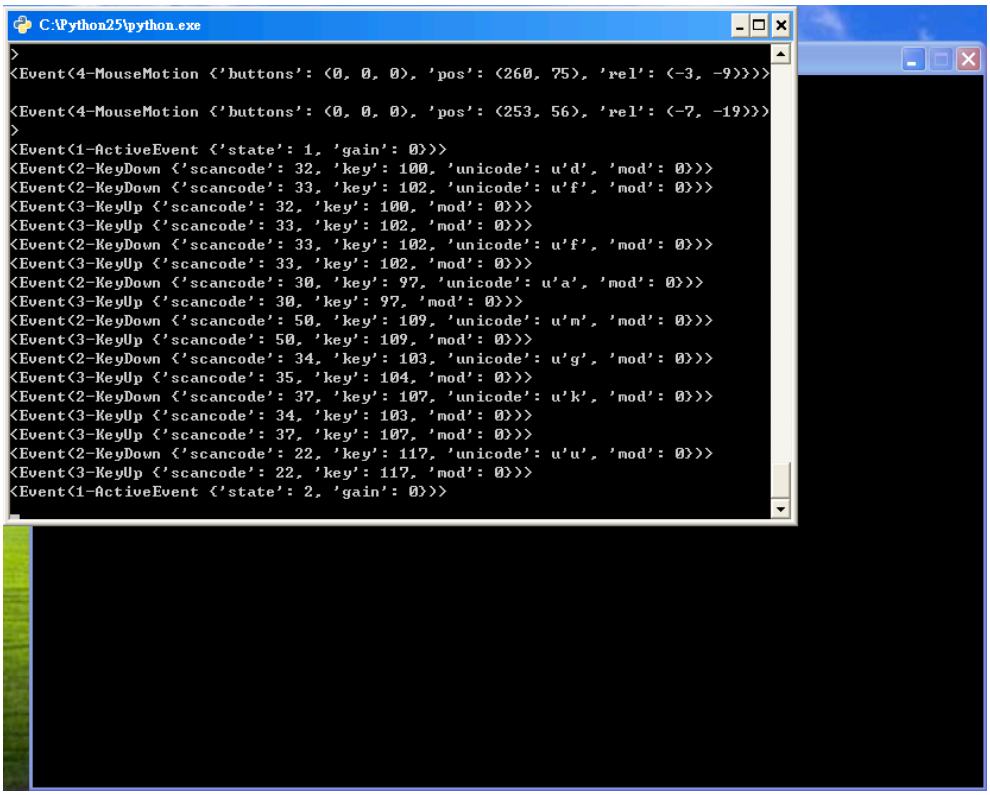
如何的不一樣呢？鍵盤的輸入若非快捷鍵的組合，大多只能在特定的區域，例如搜尋引擎的文字輸入方塊，除了這個區域，其他則要利用滑鼠點擊選取等等，滑鼠成了圖形使用者介面最常使用的輸入方式之一。

於是電腦要瞭解使用者的輸入，並與之適當的互動，如我們在搜尋引擎內輸入完關鍵字後，點擊「搜尋」的按鈕，電腦除了預先就掌握按鈕的位置，也要在我們按下滑鼠左鍵的時候，同時知道我們的動作，然後資料才會回傳到搜尋引擎的伺服器，經過運算，接著再把結果下載回我們的電腦，螢幕上的網頁視窗顯示出所找到的網址。

輸入文字也就是按下鍵盤的某些按鍵，滑鼠移動到按鈕上，點擊該按鈕，這一連串的動作對圖形使用者介面而言，電腦都當成一個一個的**事件**在處理。我們先在「while True:」陳述下加入以下的兩行程式。

```
test = pygame.event.wait()
print test
```

重新執行程式，我們看到「命令提示字元」的視窗。



我們可以發現當Pygame的視窗在最頂層時，滑鼠移動或按下鍵盤「命令提示字元」就不斷的印出新的東西來，這些就是Pygame正在捕捉到的事件。某方面來說，使用者利用輸入裝置所做的任何事情，電腦都會當成事件記錄下來。

第一個例子中我們只處理一個事件，就是當事件的type屬性為QUIT的時候，程式執行exit()函數結束Pygame的視窗，這也就是我們需要給程式提供一個結束執行的途徑，不然while True迴圈永遠不會結束。

程式執行依賴這七行程式碼，底下我們將視窗的Surface物件利用fill()方法，充滿我們指定的黑色，接著再用blit()方法，將文字的Surface物件轉換到視窗的Surface物件上，第二個參數則是所放位置的起始座標。

最後一行程式是利用display模組中的update()函數更新視窗畫面，假如少了這一行程式，我們會看不到文字內容。

動畫效果

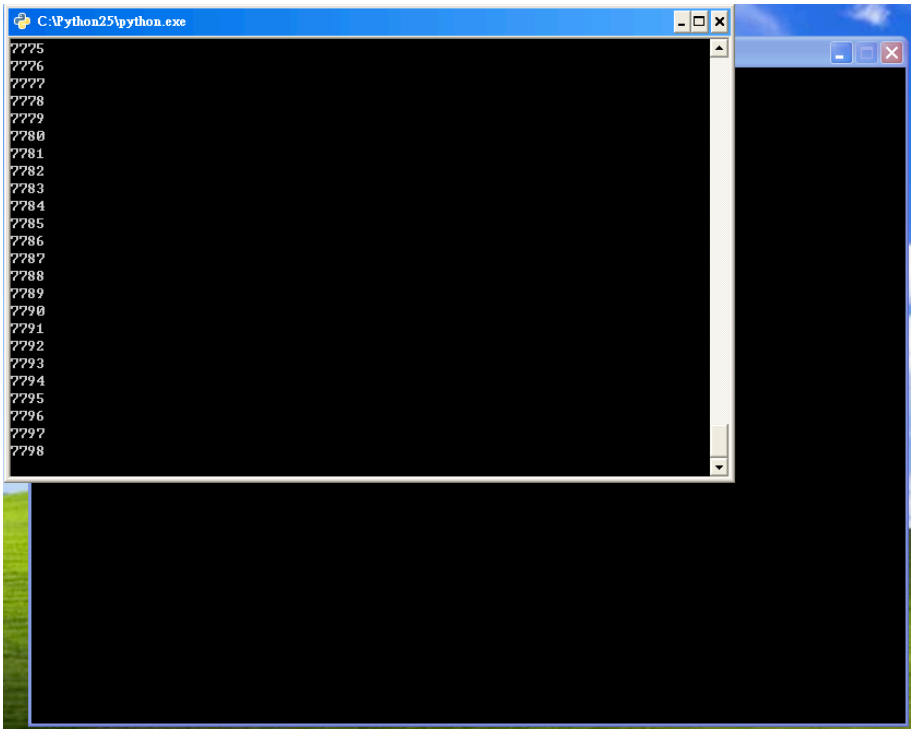
假如我們在while True迴圈之前加入以下的陳述。

```
i = 0
```

然後在while True迴圈加入以下兩行陳述。

```
print i
i += 1
```

重新執行程式，我們看到「命令提示字元」的視窗。



不斷的印出遞增的變數i，而Pygame視窗依舊不變。這是告訴我們要維持螢幕的顯示，實際上在電腦裡就是利用類似while True迴圈的方式，非常短的時間之內，用非常快的速度在螢幕上輸出光點。所以使用者如果沒有任何的操作，電腦就會不斷的重複在相同的位置輸出光點。

我們把程式在螢幕的輸出稱之為**繪圖**。重新看到遊戲的主要迴圈，程式只做了兩個繪圖工作，第一項繪圖是讓視窗充滿黑色，第二項繪圖則是把字串放置到視窗中央，程式就是不停的利用while True迴圈做這兩件繪圖工作。

假如隨著while True迴圈逐次改變第二項繪圖的座標位置，其實也就製造了動畫效果。我們試著讓文字左右移動，到視窗的邊緣便往返方向回來，首先，在while True迴圈之前我們要先建立一個新的變數。

```
dx = 1
```

然後，在while True迴圈之內加入三行程式。

```
x += dx
if (x + text.get_width()) > size[0] or x < 0:
    dx *= -1
```

這樣一來，每一次的重新繪圖文字就會位移一點，到了視窗邊緣，也就是x座標為0或是x座標加上文字寬度為800時，位移量dx的正負號相反，使文字往返方向移動。我們把所有改過的程式碼都放在一起。

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

import os
import pygame
from pygame.locals import *
from sys import exit

size = (800, 600)
title = "Hello, Pygame!"
chinese_message = "來寫遊戲吧！"
message = unicode(chinese_message, "big5")
black = (0, 0, 0)
white = (255, 255, 255)

def run():
```

```
    pygame.init()

    screen = pygame.display.set_mode(size, 0, 32)
    print unicode("變數screen的型態是", "big5"), type(screen)
    pygame.display.set_caption(title)

    font = pygame.font.Font(os.environ['SYSTEMROOT'] + \
                                                "\\Fonts\\mingliu.ttc", 80)
    print unicode("變數font的型態是", "big5"), type(font)
    text = font.render(message, True, white)
    print unicode("變數text的型態是", "big5"), type(text)

    dx = 1
    x = (size[0]-text.get_width()) / 2
    y = (size[1]-text.get_height()) / 2

    i = 0
    while True:
        test = pygame.event.wait()
        print test

        print i
        i += 1

        for event in pygame.event.get():
            if event.type == QUIT:
                exit()

        screen.fill(black)

        x += dx
        if (x+text.get_width()) > size[0] or x < 0:
            dx *= -1

        screen.blit(text, (x, y))

        pygame.display.update()

if __name__ == "__main__":
    run()
```

因為event模組中的wait()會等待事件的發生，所以只有當我們按下鍵盤或是移動滑鼠時，文字才會隨之移動。