

範例說明

在 Python 裡，Turtle 模組為 Python 的內建模組。在模組中它提供了 turtle 模組所編程的範例程式。在這些程式裡，我們可看出其包含各種領域的相關知識，其涉及幾何學、物理學甚至是各種背景不同的圖學，我們發現這組範例程式即可形成一套完整的教材，能使初學者受益良多。因此在這裏我們將介紹這些程式。

1	簡單繪圖指令 tc_yinyang.py	1
1-1	執行結果	1
1-2	程式碼	1
1-3	程式碼執行解說	4
2	和平標誌圖 tc_peace.py	9
2-1	執行結果	10
2-2	程式碼	10
2-3	程式碼執行解說	12
3	事件驅動畫圖 tc_paint.py	20
3-1	執行結果	21
3-2	程式碼	21
3-3	程式碼執行解說	24
4	兩隻龜 tc_two_canvases.py	30
4-1	執行結果	31

4-2	程式碼	31
4-3	程式碼執行解說	33
5	曲線圖 tc_chaos.py	38
5-1	執行結果	39
5-2	程式碼	39
5-3	程式碼執行解說	41
6	改變顏色 tc_colormixer.py	45
6-1	執行結果	46
6-2	程式碼	46
6-3	程式碼執行解說	48
7	視覺動畫 tc_wikipedia.py	56
7-1	執行結果	56
7-2	程式碼	56
7-3	程式碼執行解說	59
8	時鐘 tc_clock.py	64
8-1	執行結果	65
8-2	程式碼	65
8-3	程式碼執行解說	70
9	三星運動 tc_planet_and_moon.py	80
9-1	執行結果	82

9-2	程式碼.....	82
9-3	程式碼執行解說.....	87
10	畫一顆樹 tc_tree.py.....	96
10-1	執行結果.....	97
10-2	程式碼.....	97
10-3	程式碼執行解說.....	100
11	環繞舞 tc_round_dance.py	108
11-1	執行結果.....	108
11-2	程式碼.....	108
11-3	程式碼執行解說.....	112
12	建立森林 tc_forest.py.....	119
12-1	執行結果.....	120
12-2	程式碼.....	120
12-3	程式碼執行解說.....	124
13	移動河內塔 tc_minimal_hanoi.py.....	134
13-1	執行結果.....	136
13-2	程式碼.....	136
13-3	程式碼執行解說.....	140
14	潘洛斯鋪磚 tc_penrose.py.....	146
14-1	執行結果.....	146

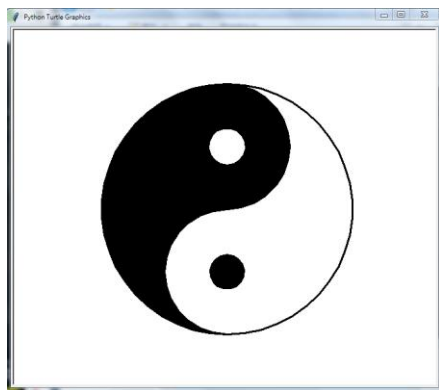
14-2	程式碼.....	147
14-3	程式碼執行解說.....	154
15	結合多邊形！tc_bytedesign.py	166
15-1	執行結果.....	166
15-2	程式碼.....	166
15-3	程式碼執行解說.....	173
16	與電腦 PKtc_nim.py	189
16-1	執行結果.....	192
16-2	程式碼.....	192
16-3	程式碼執行解說.....	201

1 簡單繪圖指令 `tc_yinyang.py`

「陰陽太極圖」是華人的智慧結晶之一，它早已隨著道家哲學思想傳遍全世界。陰陽的概念本身為一種思想理論，它可用來解釋世界萬物的各種邏輯，也用來描述各種相對立又相聯的自然現象，如天地、日夜、男女等，而代表陰陽思想的中國圖案即為太極圖。

在這支程式中其繪製出一張陰陽太極圖，其中只使用了龜模組中最基礎的基本指令，因此對於初學者可以輕易的上手。

1-1 執行結果



1-2 程式碼

```

1  #!/usr/bin/env python3
2  '''龜作圖範例集：
3
4      tdemo_yinyang.py
5
6  一種適合作為初學者繪畫的程式設計實例。
7
8
9  利用簡單的命令畫不同大小及度數的圓圈。
10
11  '''
12
13  from turtle_tc import *
14
15  def 陰(半徑, 顏色 1, 顏色 2):
16      筆寬(3)
17      顏色(黑, 顏色 1)
18      開始填()
19      畫圓(半徑/2., 180)
20      畫圓(半徑, 180)
21      左轉(180)
22      畫圓(-半徑/2., 180)
23      結束填()
24      左轉(90)
25      提筆()

```

```
26     前進 (半徑*0.35)
27     右轉 (90)
28     下筆 ()
29     顏色 (顏色 1, 顏色 2)
30     開始填 ()
31     畫圓 (半徑*0.15)
32     結束填 ()
33     左轉 (90)
34     提筆 ()
35     後退 (半徑*0.35)
36     下筆 ()
37     左轉 (90)
38
39  def 主函數 ():
40     重設 ()
41     陰 (200, 黑, 白)
42     陰 (200, 白, 黑)
43     藏龜 ()
44     return "Done!"
45
46  if __name__ == '__main__':
47     主函數 ()
48     主迴圈 ()
49
```

1-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

☆ *from* turtle_tc *import* * 與 *import* turtle_tc 的差別：

將模組中的函數變數直接導入局部變數，因此呼叫時不需再加上模組名稱 turtle_tc。

範例：

```
>>> import turtle_tc

>>> turtle_tc.前進(10)

>>> from turtle_tc import *

>>> 前進(10)
```

```
if __name__ == '__main__':

    主函數()

    主迴圈()
```


`__name__`變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則`__name__`變數會等於'`__main__`'，程序便會執行其下列動作。

主函數()`被呼叫執行`，其為本程式內部函數。

主迴圈()`被呼叫執行`，其為 `turtle_tc` 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 `mainloop()`，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數()`，其括號為空白`，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

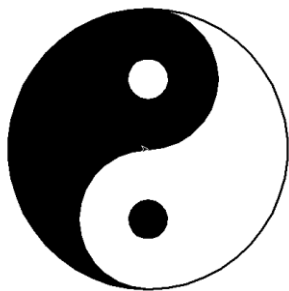
```
def 主函數():  
    重設()  
  
    陰(200, 黑, 白)  
  
    陰(200, 白, 黑)  
  
    藏龜()  
  
    return "Done!"
```

先呼叫 `重設()`使所有類設定回歸初始。（詳細龜模組函數解說可參考 2.4）

之後呼叫 `陰(200, 黑, 白)`，呼叫後會畫完左半部的半圓。 如下圖：



在呼叫一次函數將黑白調換 陰(200, 白, 黑)，即會畫完右半部的半圓，
與左邊顏色相對。如下圖：



最後呼叫 藏龜()，將龜指標隱藏。即為最後執行結果圖。

並回傳字串 "Done!" 。

```
def 陰(半徑, 顏色 1, 顏色 2) :
```

陰(半徑, 顏色 1, 顏色 2)，函數傳入三個變數，「半徑」用來設定太極圖
圓形的 半徑長，「顏色 1」用來設定半圓的填充顏色，「顏色 2」
則是設定中間小 圓的填充顏色。

```
def 陰(半徑, 顏色 1, 顏色 2) :
```

```
    筆寬(3)
```

```
    顏色(黑, 顏色 1)
```

```

開始填 ( )

畫圓 (半徑/2., 180)

畫圓 (半徑, 180)

左轉 (180)

畫圓 (-半徑/2., 180)

結束填 ( )

左轉 (90)

提筆 ( )

前進 (半徑*0.35)

右轉 (90)

下筆 ( )

顏色 (顏色 1, 顏色 2)

開始填 ( )

畫圓 (半徑*0.15)

結束填 ( )

左轉 (90)

提筆 ( )

後退 (半徑*0.35)

下筆 ( )

左轉 (90)

```

筆寬 (3)，設定指標經過(前進、後退等)的線寬度，此處設定為 3。

顏色 (黑, 顏色 1)，設定指標經過的線條顏色為 黑，而當畫圖時所要填充的顏色為 顏色 1。(如只傳入一個顏色，則既為筆色亦填充色)

開始填 ()，因要將圓內部上色，所以先呼叫此函數等待上色。

畫圓(半徑/2., 180)，之後先畫出半圓(圖 1)，半徑為大圓半徑的一半，再畫出大的半圓 畫圓(半徑, 180) (圖 2)。

將龜指標方向左右顛倒 左轉(180)，現在圓心在龜指標的右方，因此小圓半徑須為負數，並畫出半圓 畫圓(-半徑/2., 180)，此時龜指標會回到正中央圓心(圖 3)。

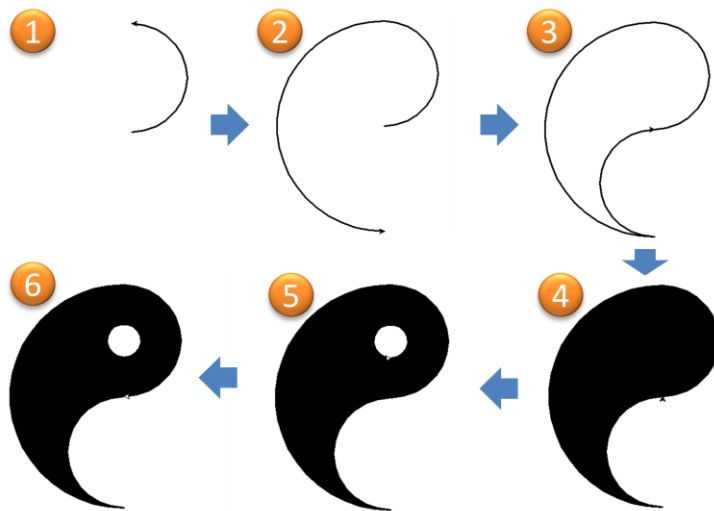
之後呼叫 結束填() 將其填色，並將龜指標方向 左轉(90) (圖 4)，因移動軌跡不需顯現，所以 提筆()，此時往前移動到最小圓的圓心下方，並 右轉(90) 將最小圓的圓心在龜指標方向的左方，且 下筆() 準備畫最小的圓。

由於最小圓顏色與其外部顏色相對，所以重新設定 顏色(顏色 1, 顏色 2)。

因最小的圓中間也需填色，因此呼叫 開始填()。呼叫 畫圓(半徑*0.15) 畫最小圓，呼叫 結束填() 將其填色(圖 5)。

最後 左轉(90)，後退(半徑*0.35)，回到正中央圓心，最後將龜指標 左轉(90)，與一開始龜指標方向左右相反(圖 6)。

圖例(以第一次呼叫為例)：

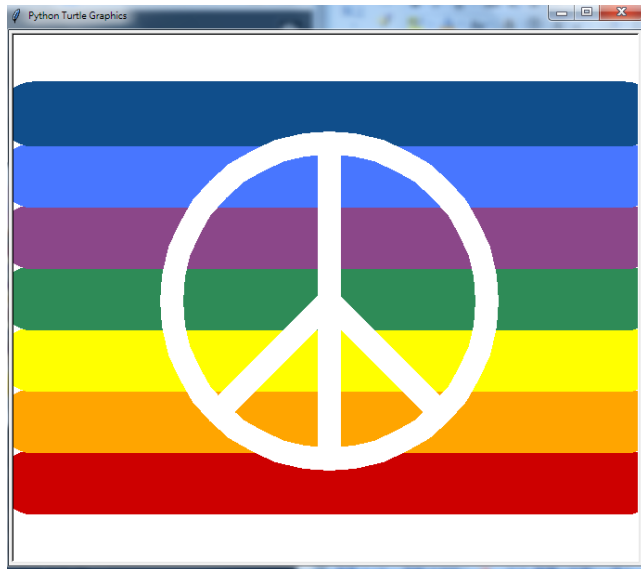


2 和平標誌圖 tc_peace.py

此圖為和平標誌圖，起源於核裁軍運動(Nuclear Disarmament)，為英國畫家 Gerald Holtom 所設計。此標誌的構想主要由海軍旗語信息代碼 N 與 D 字母所組成，也為核裁軍運動的英文首字母，N 的旗語為兩面旗子向左下與右下成 90 度角，D 為兩面旗子向上及向下拿著，兩者及構成圓中間標誌，最後他將這些用圓圍住，即為現在的和平標誌圖。背景的圖為和平旗，也叫彩虹旗，利用彩虹的七個顏色色調組成，流行在義大利，象徵著國際合作運動。

此程式只使用簡單的命令指令，也沒用太多的變數，很適合當作初學者的入門程式。

2-1 執行結果



2-2 程式碼

```

1  #!/usr/bin/env python3
2  '''龜作圖範例集：
3
4      tdemo_peace.py
5
6  一個非常簡單的畫圖範例程序，適合初學者。
7  程式設計實例。
8  除了設定和平顏色們和 for 迴圈，
9  皆只使用命令函數。
10 '''
11
12 from turtle_tc import *
13
14 def 主函數():
15     和平顏色們 = ("red3", 橙, 黃,
16                  "seagreen4", "orchid4",
17                  "royalblue1", "dodgerblue4")
18
19     重設()
20     幕類()
21     提筆()
22     前往(-320, -195)
23     筆寬(70)
24
25     for p 顏色 in 和平顏色們:
26         顏色(p 顏色)
27         下筆()
28         前進(640)
29         提筆()
30         後退(640)
31         左轉(90)
32         前進(66)
33         右轉(90)
34
35     筆寬(25)
36     顏色(白)

```

```

37     前往(0,-170)
38     下筆()
39
40     畫圓(170)
41     左轉(90)
42     前進(340)
43     提筆()
44     左轉(180)
45     前進(170)
46     右轉(45)
47     下筆()
48     前進(170)
49     提筆()
50     後退(170)
51     左轉(90)
52     下筆()
53     前進(170)
54     提筆()
55
56     前往(0,300) # 消失，如果 藏龜() 不可用;-)
57     return "結束!"
58
59 if __name__ == "__main__":
60     主函數()
61     主迴圈()

```

2-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。


```
if __name__ == "__main__":
```

```
    主函數()
```

```
    主迴圈()
```

`__name__` 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則 `__name__` 變數會等於 `'__main__'`，程序便會執行其下列動作。

主函數() 被呼叫執行，其為本程式內部函數。

主迴圈() 被呼叫執行，其為 `turtle_tc` 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 `mainloop()`，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數()，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():
```

```
    和平顏色們 = ("red3", 橙, 黃,  
                  "seagreen4", "orchid4",  
                  "royalblue1", "dodgerblue4")
```

```
重設 ()  
  
幕類 ()  
  
提筆 ()  
  
前往 (-320, -195)  
  
筆寬 (70)  
  
for p 顏色 in 和平顏色們:  
    顏色 (p 顏色)  
  
    下筆 ()  
  
    前進 (640)  
  
    提筆 ()  
  
    後退 (640)  
  
    左轉 (90)  
  
    前進 (66)  
  
    右轉 (90)
```

先設定要繪製的和平圖底色顏色，建立 和平顏色們 顏色陣列。

此時前往左下角，前往 (-320, -195)，並設定 筆寬 (70)。

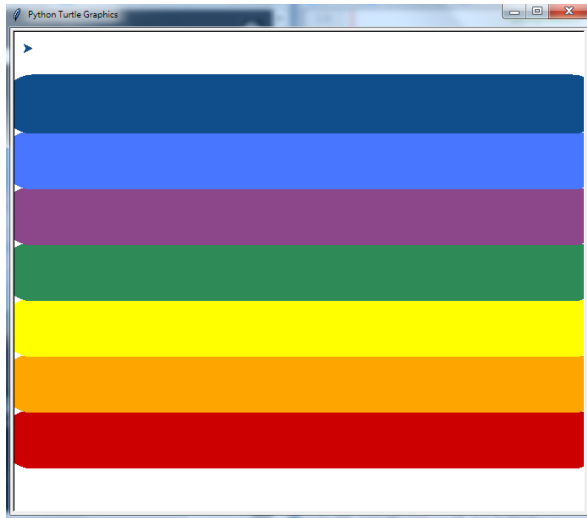
建立 for 迴圈，抓取 和平顏色們 陣列裡的顏色，開始準備畫和平圖背景圖。（for 迴圈詳細解釋請參考下方）

其中從左到右畫線，再提筆回到左邊位置。

左轉往上 前進 (66)，再向 右轉 (90) 回到龜指標鍵頭往右狀態。

重複以上動作，直到 for 迴圈 7 個顏色抓取畫完。

此時圖為：



☆ *for* 迴圈可用來抓取一序列資料的每個元素，並根據資料的元素數量重複執行。

用法如 *for* 值 *in* 序列範圍：，意思即為取得序列範圍的每一個元素，傳進變數 值，執行迴圈程式，直到每個元素都被執行過（總共執行序列範圍的總元素數量次）。

我們將以範例加以說明：

可使用 range 重複執行	
<pre>>>> for i in range(5): ... print(i) ... 0 1 2 3</pre>	<p>此處共執行 5 次，從 0 開始到 5-1 結束，等於 $0 \leq i < 5$。</p> <p>range(5)範圍等於 range(0,5)。</p> <p>若要從 1 開始到 5，則可輸入 range(1,6)。</p>

4	
<pre>>>> for i in range(3,16,3) : ... print(i) ... 3 6 9 12 15</pre>	<p>range(3,16,3)代表從 3 開始到 15，每 3 個數執行一次。</p>
抓取字串	
<pre>>>> a="Python" >>> for i in a : ... print(i) ... P y t h o n</pre>	<p>此處為抓取字串"Python"，由於字串的長度為 5，為 P、y、t、h、o、n，因此 for 迴圈總共執行 5 次。</p>
抓取陣列元素	
<pre>>>> a=["Black","Red",100]</pre>	<p>a 陣列共有 3 個元素，將陣列內</p>

<pre>>>> for i in a : ... print(i) ... Black Red 100</pre>	<p>元素逐個丟進變數 i ，for 迴圈 執行 3 次。</p>
---	---------------------------------------

筆寬 (25)

顏色 (白)

前往 (0,-170)

下筆 ()

畫圓 (170)

左轉 (90)

前進 (340)

提筆 ()

左轉 (180)

前進 (170)

右轉 (45)

下筆 ()

前進 (170)

提筆 ()

後退 (170)

左轉 (90)

下筆 ()

```
前進(170)
```

```
提筆()
```

```
前往(0,300) # 消失，如果 藏龜()不可用;-)
```

```
return "結束!"
```

再來準備畫中間的和平標誌，因此重新設定筆寬及顏色，且 前往

(0,-170)。

到了正中間下方，畫圓(170)，半徑為 170。

左轉(90)，讓龜指標鍵頭往上，並 前進(340)，畫下和平標誌圖的中間直線。

之後 左轉(180)，等於將龜指標鍵頭往下。前進(170)，回到正中間(0,0)。

右轉(45)，將龜指標鍵頭往左下角。前進(170)，到圓邊上，繪製和平標誌圖的左下半徑直線。

再 後退(170) 回到正中間。

左轉(90)，將龜指標鍵頭往右下角。前進(170)，到圓邊上，繪製和平標誌圖的右下半徑直線。

之後 提筆()，不畫出移動軌跡。

並 前往(0,300)，到畫布範圍外，龜指標則不會在畫布上。（也可以選擇使用 藏龜()，則龜指標也不會顯現在畫布上）

最後回傳 "結束!" 字串。（return 用法解釋請參考下方）

★ 函數 (function) 可回傳值給呼叫此函數的指令。如函數中沒有 *return*，則呼叫該函數的指令會自動回傳 *None* 物件。通常 *return* 被用來回傳運算結果，或是用來跳出(結束)函數。

範例：

回傳運算結果	
<pre>>>> def sum(x,y,z): ... return x+y+z,x-y-z ... >>> value1,value2=sum(5,12,7) >>> print(value1,value2) 24 -14</pre>	<p>用來回傳運算結果，而回傳值可回傳不只一值，值與值間用逗號分開即可。</p>
跳出函數	
<pre>>>> def fun(x): ... if type(x)!=int: ... return " 不為整數" ... x= x*100 ... return x ... >>> fun("a")</pre>	<p>因傳入的值不是整數，無法計算，因此回傳"不為整數"值，此時函數下方的程式便不會繼續執行。</p>

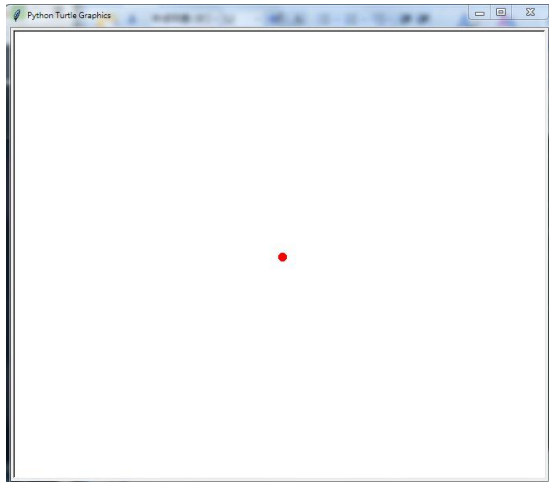
<pre>'不為整數'</pre> <pre>>>> fun(2)</pre> <pre>200</pre>	
無回傳值	
<pre>>>> def talk(name):</pre> <pre>... print(" Hi!",name)</pre> <pre>...</pre> <pre>>>> value = talk ("Sunny")</pre> <pre>Hi! Sunny</pre> <pre>>>> print(value)</pre> <pre>None</pre>	<p>此處由於沒有 return 回傳值， 因此函數自動回傳 None 物件。</p>

3 事件驅動畫圖 tc_paint.py

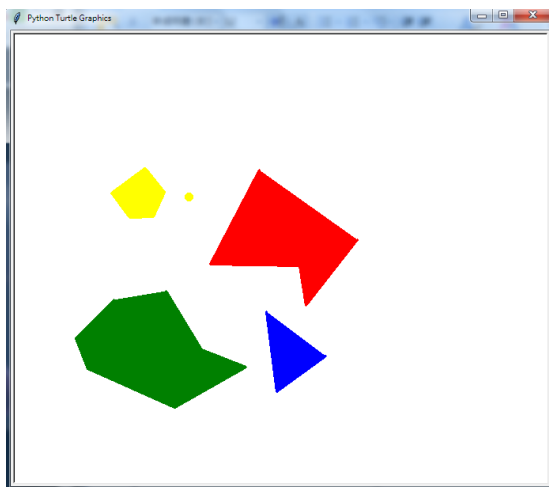
一個簡易的畫圖程式，利用滑鼠點擊事件來控制程式，並根據滑鼠三鍵來區別功能使用。滑鼠左鍵用來移動位置，中鍵用來改變顏色（設定顏色順序為紅、綠、藍、黃）；右鍵則用來切換提筆、下筆，當在下筆狀態時，會開始記錄移動的點座標並畫線，直到再度切換為提筆狀態，如此時記錄點有至少兩個以上，則包含起始點組成的多邊形，會被現在設定的顏色填滿。

3-1 執行結果

開始：



執行：



3-2 程式碼

```
1  #!/usr/bin/env python3
2  '''龜作圖範例集：
3
4      tdemo_paint.py
```

一個簡單的事件驅動畫圖程序

- 滑鼠左鍵移動烏龜
- 滑鼠中鍵更換顏色
- 滑鼠右鍵決定提筆（當 龜指標 移動時不會畫線）

和 下筆（移動會畫線）。

如果提筆時至少有兩個下筆移動點，
則包含起點的多邊形將會被顏色填滿。

點擊進入畫布。

使用三個滑鼠按鈕。

按下 STOP 鈕關閉程序。

```
'''  
from turtle_tc import *  
  
def 切換上下(x=0, y=0):  
    if 筆()["pendown"]:  
        結束填()  
        提筆()  
    else:  
        下筆()  
        開始填()
```

```

31 def 改變顏色(x=0, y=0):
32     global 顏色們
33     顏色們 = 顏色們[1:]+顏色們[:1]
34     顏色(顏色們[0])
35
36 def 主函數():
37     global 顏色們
38     形狀("circle")
39     重設大小模式("user")
40     形狀大小(.5)
41     筆寬(3)
42     顏色們=[紅, 綠, 藍, 黃]
43     顏色(顏色們[0])
44     切換上下()
45     在點擊幕時(前往,1)
46     在點擊幕時(改變顏色,2)
47     在點擊幕時(切換上下,3)
48     return "事件迴圈"
49
50 if __name__ == "__main__":
51     訊息 = 主函數()
52     印(訊息)
53     主迴圈()

```

3-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數。

```
if __name__ == "__main__":  
    訊息 = 主函數()  
    印(訊息)  
    主迴圈()
```

`__name__` 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則 `__name__` 變數會等於 `'__main__'`，程序便會執行其下列動作。

主函數 `()` 被呼叫執行，其為本程式內部函數，將其回傳的資料以 訊息 接收。之後將訊息印出。

主迴圈 `()` 被呼叫執行，其為 turtle_tc 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 `mainloop()`，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數 `()`，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():
    global 顏色們
    形狀("circle")
    重設大小模式("user")
    形狀大小(.5)
    筆寬(3)
    顏色們=[紅, 綠, 藍, 黃]
    顏色(顏色們[0])
    切換上下()
    在點擊幕時(前往,1)
    在點擊幕時(改變顏色,2)
    在點擊幕時(切換上下,3)
    return "事件迴圈"
```

宣告 顏色們 為全域變數（Global，詳細解釋請參考下方）。

設置龜指標形狀為圓形 形狀("circle")，並 重設大小模式("user")，之後設置其形狀大小(.5) 並且 筆寬(3)，之後設定顏色陣列 顏色們=[紅, 綠, 藍, 黃]，並且將顏色設定為陣列第一個 顏色(顏色們[0])。

此時呼叫 切換上下() 函數，將 龜指標 從一開始的下筆狀態切換為提筆。

之後根據滑鼠點擊螢幕的左中右鍵呼叫不同的函數，函數的第二個參數代表滑鼠左中右鍵。

在點擊幕時 (前往,1) ，1 代表左鍵，當按下左鍵便呼叫前往函數，並傳入
點擊坐標位置，即等於呼叫 前往(x,y)。

在點擊幕時 (改變顏色,2) ，2 代表中鍵。

在點擊幕時 (切換上下,3) ，3 代表右鍵。

最後回傳 "事件迴圈" 字串。（*return* 用法解釋請參考下方）

★ *global* 為全域變數，可以使變數能夠在函數內或類別內都可以直接存取。而一般我們宣告的變數皆為區域變數，其所取用值的作用域只在其建立的範圍內。

例如：

區域變數	
<pre>>>> x=1 >>> def fun(): ... x=2 ... print(x) ... >>> fun() 2 >>> x 1</pre>	因在 fun() 函數內的 x 為區域變數，所以其值的範圍作用也只在此函數內部，不影響外部變數的 x 值。
<pre>>>> x=1 >>> def fun():</pre>	而當其取用變數時，若在內部函數範圍查找不到，則往外部範圍往外尋

<pre>... print(x) ... >>> fun() 1</pre>	找，取得外部的 x 值。
全域變數	
<pre>>>> x=1 >>> def fun(): ... global x ... x=2 ... print(x) ... >>> fun() 2 >>> x 2</pre>	因在 fun() 函數內的 x 宣告為全域變數，所以其取用值的範圍作用域在程式全域，函數內部的 x 值改變，外部的 x 值。

★ 函數 (function) 可回傳值給呼叫此函數的指令。如函數中沒有 *return*，則呼叫該函數的指令會自動回傳 *None* 物件。通常 *return* 被用來回傳運算結果，或是用來跳出(結束)函數。

範例：

回傳運算結果	
>>> def sum(x,y,z):	用來回傳運算結果，而回傳值

<pre>... return x+y+z,x-y-z ... >>> value1,value2=sum(5,12,7) >>> print(value1,value2) 24 -14</pre>	<p>可回傳不只一值，值與值間用逗號分開即可。</p>
<p>跳出函數</p>	
<pre>>>> def fun(x): ... if type(x)!=int: ... return " 不為整數" ... x= x*100 ... return x ... >>> fun("a") '不為整數' >>> fun(2) 200</pre>	<p>因傳入的值不是整數，無法計算，因此回傳"不為整數"值，此時函數下方的程式便不會繼續執行。</p>
<p>無回傳值</p>	
<pre>>>> def talk(name): ... print(" Hi!",name) ... >>> value = talk ("Sunny")</pre>	<p>此處由於沒有 return 回傳值，因此函數自動回傳 None 物件。</p>

Hi! Sunny	
>>> print(value)	
None	

```
def 切換上下(x=0, y=0):
    if 筆() ["pendown"]:
        結束填()
        提筆()
    else:
        下筆()
        開始填()
```

傳入參數 x、y，用來切換下筆、提筆狀態，並且進行填色。

當現在如果處於下筆狀態 `if 筆() ["pendown"]:`，則呼叫 `結束填()`，進行填色，並將 龜指標 切換為提筆狀態 `提筆()`。

而如果目前為提筆狀態，則切換為下筆狀態 `下筆()`，並呼叫 `開始填()`，開始記錄移動座標點。

```
def 改變顏色(x=0, y=0):
    global 顏色們
    顏色們 = 顏色們[1:]+顏色們[:1]
    顏色(顏色們[0])
```

傳入參數 x 、 y ，用來改變 龜指標 顏色，將顏色切換成陣列下一個顏色。

將顏色們陣列第一個元素移到最後一個，顏色們 = 顏色們[1:] + 顏色們[:1]，。

顏色們[1:] 代表陣列第一個到最後一個，顏色們[:1] 而代表從第零個到第一個（不包含第一個）。

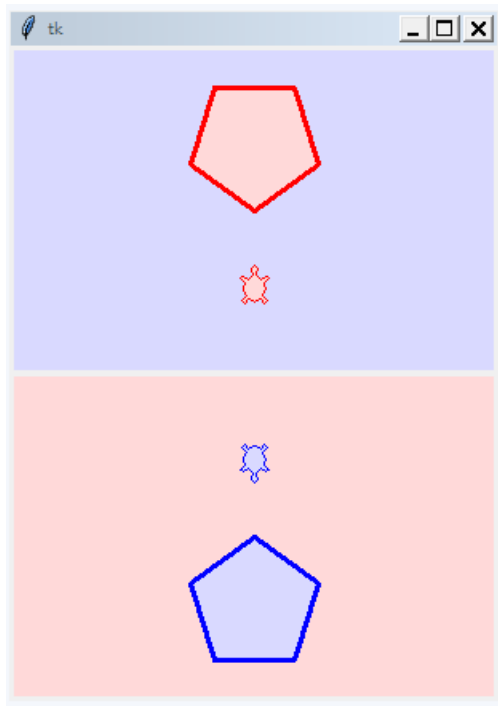
而將顏色切換為第零個，顏色(顏色們[0])。

4 兩隻龜 tc_two_canvases.py

利用 TK 建立兩個畫布導入龜幕類，並建立兩個原龜類（RawTurtle）物件在龜幕類上，在此兩個龜幕類上畫正五邊形。

此程式創建原龜類，原龜類為龜類的父類，原龜類跟龜類擁有一樣的方法屬性，但原龜類的創建需要指定一個畫布或龜幕類參數，讓其知道在哪裡繪圖，而龜類的龜幕類則會自動產生（如果它沒有設置的話）。

4-1 執行結果



4-2 程式碼

```
1  '''turtledemo.two_canvases
2
3  利用 TurtleScreen【龜幕類】跟 RawTurtle【原生龜類】
4  在兩個不同的畫布上繪製。
5  兩個畫布需要分開關閉，
6  除非按 STOP 按鈕。
7  '''
8
9  from turtle_tc import *; from turtle_tc import 龜幕類, 原龜類, TK
10
```

```
11 def 主函數():
12     根 = TK.Tk()
13     畫布1 = TK.Canvas(根, width=300, height=200, bg="#ddffff")
14     畫布2 = TK.Canvas(根, width=300, height=200, bg="#ffeeee")
15     畫布1.pack()
16     畫布2.pack()
17
18     螢幕1 = 龜幕類(畫布1)
19     螢幕1.背景色(0.85, 0.85, 1)
20     螢幕2 = 龜幕類(畫布2)
21     螢幕2.背景色(1, 0.85, 0.85)
22
23     紅龜 = 原龜類(螢幕1)
24     藍龜 = 原龜類(螢幕2)
25
26     紅龜.顏色(紅, (1, 0.85, 0.85))
27     紅龜.筆寬(3)
28     藍龜.顏色(藍, (0.85, 0.85, 1))
29     藍龜.筆寬(3)
30
31     for t in 紅龜, 藍龜:
32         t.形狀(龜形)
33         t.左轉(36)
34
35     藍龜.左轉(180)
36
```

```

37     for t in 紅龜, 藍龜:
38         t.開始填()
39     for i in 範圍(5):
40         for t in 紅龜, 藍龜:
41             t.前進(50)
42             t.左轉(72)
43     for t in 紅龜, 藍龜:
44         t.結束填()
45         t.左轉(54)
46         t.提筆()
47         t.後退(50)
48
49     return "主迴圈"
50
51
52 if __name__ == '__main__':
53     主函數()
54     TK.mainloop() # 保持窗口打開，直到使用者關閉

```

4-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```
if __name__ == '__main__':  
    主函數()  
  
    TK.mainloop() # 保持窗口打開，直到使用者關閉
```

__name__ 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則__name__ 變數會等於 '__main__'，程序便會執行其下列動作。

主函數() 被呼叫執行，其為本程式內部函數。

mainloop () 被呼叫執行，其為 TK 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```


主函數()，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

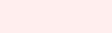
```
def 主函數():  
  
    根 = TK.Tk()  
  
    畫布 1 = TK.Canvas(根, width=300, height=200, bg="#ddffff")  
  
    畫布 2 = TK.Canvas(根, width=300, height=200, bg="#ffeeee")  
  
    畫布 1.pack()  
  
    畫布 2.pack()
```

設置 TK.Tk() 為 根。：

創建兩個畫布，寬為 300、高為 200，並設定其背景色。

將兩個畫布 `pack()`，自動配置元件位置，由上到下自動排序。

`#ddffff`： 淡藍色

`#ffeeee`： 淡紅色

def 主函數():

```
螢幕 1 = 龜幕類(畫布 1)
```

```
螢幕 1.背景色(0.85, 0.85, 1)
```

```
螢幕 2 = 龜幕類(畫布 2)
```

```
螢幕 2.背景色(1, 0.85, 0.85)
```

建立 螢幕 1 跟 螢幕 2，分別為 畫布 1 跟 畫布 2。

設置 螢幕 1 背景色為淡藍色(0.85, 0.85, 1)。

設置 螢幕 2 背景色為淡紅色(1, 0.85, 0.85)。

def 主函數():

```
紅龜 = 原龜類(螢幕 1)
```

```
藍龜 = 原龜類(螢幕 2)
```

```
紅龜.顏色(紅, (1, 0.85, 0.85))
```

```
紅龜.筆寬(3)
```

```
藍龜.顏色(藍, (0.85, 0.85, 1))
```

```
藍龜.筆寬(3)
```

創建紅龜，其為原龜類，作圖在 螢幕 1 上。

創建藍龜，其為原龜類，作圖在 螢幕 2 上。

設置紅龜筆色為紅色，填色為淡紅色(1, 0.85, 0.85)。

設定紅龜畫出來的寬度為 3。

設置藍龜筆色為藍色，填色為淡藍色(1, 0.85, 0.85)。

設定藍龜畫出來的寬度為 3。

```
def 主函數():
```

```
    for t in 紅龜, 藍龜:
```

```
        t.形狀(龜形)
```

```
        t.左轉(36)
```

```
    藍龜.左轉(180)
```

利用 for 迴圈抓取紅龜、藍龜，執行下列動作。

設定其龜指標形狀為 龜形。

將龜指標向左轉 36 度，準備畫五邊形。

for 迴圈結束。

藍龜向左轉 180 度，與紅龜運動軌跡相反。


```
def 主函數():  
  
    for t in 紅龜, 藍龜:  
        t.開始填()  
  
    for i in 範圍(5):  
        for t in 紅龜, 藍龜:  
            t.前進(50)  
            t.左轉(72)  
  
    for t in 紅龜, 藍龜:  
        t.結束填()  
        t.左轉(54)  
        t.提筆()  
        t.後退(50)  
  
    return "主迴圈"
```

利用 for 迴圈抓取紅龜、藍龜，執行下列動作。

將其呼叫 開始填，開始記錄座標，準備開始填色。

結束 for 迴圈。

利用 for 迴圈重複執行 5 次，數字範圍為 0~4，準備開始畫正五邊形。

裡面再建立一個 for 迴圈抓取紅龜、藍龜。

將其前進 50，畫出正五邊形的邊。

左轉 72 度，因正五邊形的角為 108 度。

裡面 for 迴圈結束。

for 迴圈結束，紅龜與藍龜的正五邊形畫完。

創建 for 迴圈抓取紅龜、藍龜，執行下列動作。

呼叫 結束填，根據設定的填色填滿五邊形。

將其左轉 54 度。（此時紅龜指標向上，藍龜指標向下）

提筆，龜指標移動實部畫出軌跡。

最後龜指標後退 50。

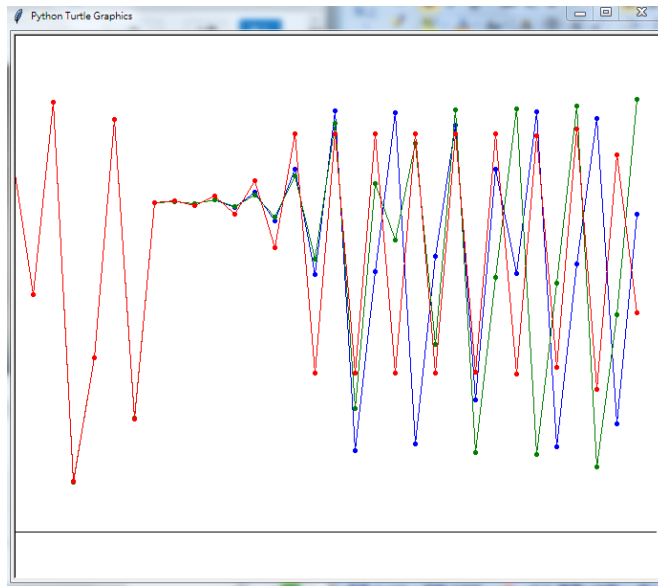
for 迴圈結束。

回傳字串"主迴圈"。

5 曲線圖 tc_chaos.py

此程式介紹如何利用龜作圖繪製曲線圖，可藉由設座標系統設定畫布座標，並根據座標繪製。在此程式設定了三個方程式，其三個方程式展開後會發現皆為同一個方程式，但由於這三個方程式的算式優先順序不同，因此在計算中會造成結果的差異，因此將三個結果持續丟入三個方程式，會造成結果差異越來越大。在此程式三個方程式的曲線圖畫出，可輕易看出結算結果差異。

5-1 執行結果



5-2 程式碼

```
1  # 文件：tdemo_chaos.py
2  # 作者：Gregor Lingl
3  # 日期：2009-06-24
4
5  # 混亂的示範
6
7  from turtle_tc import *
8
9  N = 80
10
11  def f(x):
12      return 3.9*x*(1-x)
13
```

```

14 def g(x):
15     return 3.9*(x-x**2)
16
17 def h(x):
18     return 3.9*x-3.9*x*x
19
20 def 跳至(x, y):
21     提筆(); 前往(x, y)
22
23 def 直線(x1, y1, x2, y2):
24     跳至(x1, y1)
25     下筆()
26     前往(x2, y2)
27
28 def 座標系統():
29     直線(-1, 0, N+1, 0)
30     直線(0, -0.1, 0, 1.1)
31
32 def 畫(函數, 開始, 顏色):
33     筆色(顏色)
34     x = 開始
35     跳至(0, x)
36     下筆()
37     點(5)
38     for i in 範圍(N):

```

```

39         x=函數(x)
40         前往(i+1,x)
41         點(5)
42
43     def 主函數():
44         重設()
45         設座標系統(-1.0,-0.1, N+1, 1.1)
46         速度(0)
47         藏龜()
48         座標系統()
49         畫(f, 0.35, 藍)
50         畫(g, 0.35, 綠)
51         畫(h, 0.35, 紅)
52         # 現在放大：
53         for s in 範圍(100):
54             設座標系統(0.5*s,-0.1, N+1, 1.1)
55         return "結束!"
56
57 if __name__ == "__main__":
58     主函數()
59     主迴圈()

```

5-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```
if __name__ == "__main__":  
    主函數()  
    主迴圈()
```

__name__ 變數通常用來偵測本程式是否作為獨立的主程式被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則 __name__ 變數會等於 '__main__'，程序便會執行其下列動作。

主函數() 被呼叫執行，其為本程式內部函數。

主迴圈() 被呼叫執行，其為 turtle_tc 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 mainloop()，等候使用者進一步使用滑鼠鍵盤來控制。

```
N = 80
```

宣告在最外部，所以皆可以存取此數。

```
def 主函數():
```

主函數()，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():  
    重設()  
    設座標系統(-1.0, -0.1, N+1, 1.1)  
    速度(0)  
    藏龜()  
    座標系統()  
    畫(f, 0.35, 藍)  
    畫(g, 0.35, 綠)  
    畫(h, 0.35, 紅)  
    # 現在放大：  
    for s in 範圍(100):
```

```
    設座標系統(0.5*s,-0.1, N+1, 1.1)
    return "結束!"
```

設座標系統(-1.0,-0.1, N+1, 1.1)設置幕的座標系統左下角為(-1,-0.1)而右上
上角為(N+1,1.1)。(N 為 80)

設置 速度(0)，最快速即不會有動畫產生。

隱藏龜指標，藏龜()。

呼叫 座標系統() 函數，便會繪製 xy 座標軸。

之後呼叫函數 畫()，開始畫圖。

畫(f, 0.35, 藍)，第一個為參數連結函數 f (要繪製的方程式)，第二
個參數為起始點，第三個參數為繪製顏色。

畫(g, 0.35, 綠)，與上者類似，繪製第二個方程式。

畫(h, 0.35, 紅)，與上者類似，繪製第三個方程式。

繪製完三個方程式後，將後部分有差異的區間放大顯示。

設置 for 迴圈，重複 100 次(0~99)，從 設座標系統(-1.0,-0.1, 81, 1.1)到 設
座標系統(49.5,-0.1, 81, 1.1)逐漸放大。

最後回傳 "事件迴圈" 字串。

```
def 座標系統():
    直線(-1, 0, N+1, 0)
    直線(0, -0.1, 0, 1.1)
```

函數 座標系統()，無傳入參數。用來繪製 xy 座標軸。

呼叫直線(-1, 0, N+1, 0)，繪製 y=0 座標軸，從座標(-1,0)到座標(81, 0)。

呼叫直線(0, -0.1, 0, 1.1)，繪製 x=0 座標軸，從座標(0, -0.1)到座標
(0, 1.1)。

```
def 直線(x1, y1, x2, y2):
```

```
跳至 (x1, y1)
```

```
下筆 ()
```

```
前往 (x2, y2)
```

函數，傳入參數 $x1, y1, x2, y2$ ，繪製從 $(x1, y1)$ 到 $(x2, y2)$ 的直線。

呼叫函數 跳至 $(x1, y1)$ ，將位置移往 $(x1, y1)$ 。

開始準備畫線，因此下筆 $()$ 。

開始畫線，前往 $(x2, y2)$ 。

```
def 跳至 (x, y):
```

```
    提筆 (); 前往 (x, y)
```

函數 跳至 (x, y) ，傳入 x, y 參數，將位置移往座標 (x, y) 且不畫出移動軌跡。

因不畫出軌跡，所以 提筆 $()$ 。前往 (x, y) ，到位置 (x, y) 。

```
def 畫 (函數, 開始, 顏色):
```

```
    筆色 (顏色)
```

```
    x = 開始
```

```
    跳至 (0, x)
```

```
    下筆 ()
```

```
    點 (5)
```

```
    for i in 範圍 (N):
```

```
        x=函數 (x)
```

```
        前往 (i+1, x)
```

```
        點 (5)
```

函數 畫 $()$ ，傳入參數 函數, 開始, 顏色，根據指定函數及顏色繪製方程式。

設定繪製顏色 筆色 $(顏色)$ ，並設定開始位置，

呼叫函數 跳至 $(0, x)$ ，到位置 $(0, x)$ 。

開始準備畫線，因此下筆 $()$ 。先在開始點繪製直徑 5 的圓點，點 (5) 。

設置 for 迴圈，迴圈範圍為 0~79，也為 i 值大小。

將 x 丟到函數，得到下一個 x，即新的位置。

前往 (i+1, x)，到新的位置，並在新的位置繪製圓點。

再到重複動作，直到迴圈結束。

```
def f(x):  
    return 3.9*x*(1-x)  
  
def g(x):  
    return 3.9*(x-x**2)  
  
def h(x):  
    return 3.9*x-3.9*x*x
```

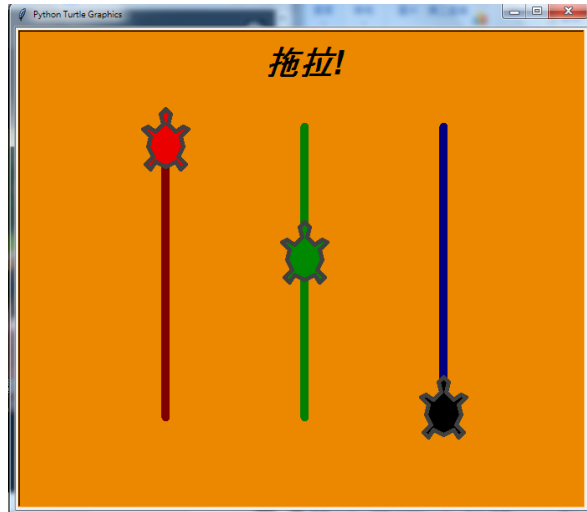
三個函數，傳入參數 x，三個方程式展開皆一樣，但根據計算優先順序，會造成結果有些微差異。

6 改變顏色 tc_colormixer.py

根據座標位置改變其背景顏色，創造一個新的類別繼承龜類，再分別以三個新類別物件代表 RPG 設置，使用者可藉由滑鼠的拖曳事件改變背景顏色設置。

此程式實作了如何繼承父類別創建新的類別，及將滑鼠事件連結函數，可作為一個簡單的基礎教學。

6-1 執行結果



6-2 程式碼

```
1  # COLORMIX
2
3  from turtle_tc import *; from turtle_tc import 幕類, 龜類, 主迴圈
4
5  class 顏色龜類(龜類):
6
7      def __init__(我, x, y):
8          龜類.__init__(我)
9          我.形狀(龜形)
10         我.重設大小模式("user")
11         我.形狀大小(3,3,5)
12         我.筆粗(10)
13         我._color = [0,0,0]
```

```

14         我.x = x
15         我._color[x] = y
16         我.顏色(我._color)
17         我.速度(0)
18         我.左轉(90)
19         我.提筆()
20         我.前往(x, 0)
21         我.下筆()
22         我.設 y 座標(1)
23         我.提筆()
24         我.設 y 座標(y)
25         我.筆色("gray25")
26         我.在拖曳時(我.平移)
27
28     def 平移(我, x, y):
29         我.設 y 座標(max(0, min(y, 1)))
30         我._color[我.x] = 我.y 座標()
31         我.填色(我._color)
32         設背景的顏色()
33
34     def 設背景的顏色():
35         幕.背景色(紅龜.y 座標(), 綠龜.y 座標(), 藍龜.y 座標())
36
37     def 主函數():
38         global 幕, 紅龜, 綠龜, 藍龜

```

```

39     幕 = 幕類()
40     幕.延遲(0)
41     幕.設座標系統(-1, -0.3, 3, 1.3)
42
43     紅龜 = 顏色龜類(0, .5)
44     綠龜 = 顏色龜類(1, .5)
45     藍龜 = 顏色龜類(2, .5)
46     設背景的顏色()
47
48     寫手 = 龜類()
49     寫手.藏龜()
50     寫手.提筆()
51     寫手.前往(1,1.15)
52     寫手.寫("拖拉!",align="center",font=("Arial",30,("bold","italic")))
53     return "事件迴圈"
54
55 if __name__ == "__main__":
56     訊息 = 主函數()
57     印(訊息)
58     主迴圈()

```

6-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```
if __name__ == "__main__":  
    訊息 = 主函數()  
    印(訊息)  
    主迴圈()
```

__name__ 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則__name__ 變數會等於'__main__'，程序便會執行其下列動作。

主函數() 被呼叫執行，其為本程式內部函數，將其回傳的資料以 訊息 接收。之後將訊息印出。

主迴圈() 被呼叫執行，其為 turtle_tc 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 mainloop()，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數()，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():  
    global 幕, 紅龜, 綠龜, 藍龜  
    幕 = 幕類()  
    幕.延遲(0)
```

```
幕.設座標系統(-1, -0.3, 3, 1.3)
```

```
紅龜 = 顏色龜類(0, .5)
```

```
綠龜 = 顏色龜類(1, .5)
```

```
藍龜 = 顏色龜類(2, .5)
```

```
設背景的颜色()
```

```
寫手 = 龜類()
```

```
寫手.藏龜()
```

```
寫手.提筆()
```

```
寫手.前往(1,1.15)
```

```
寫手.寫("拖拉!",align="center",font=("Arial",30,("bold","italic")))
```

```
return "事件迴圈"
```

宣告全域變數。

宣告幕為幕類 幕 = 幕類()，幕.延遲(0) 設置幕的更新延遲時間為 0，

幕.設座標系統(-1, -0.3, 3, 1.3)設置幕的座標系統左下角為(-1,-0.3)而右上角為(3,1.3)。

宣告三個龜為顏色龜類，並傳入其顏色龜位置，同等於顏色設定。

紅龜 = 顏色龜類(0, .5)，第一個參數代表 x 座標(也代表 RPG 第 0 個參數)，第二個參數代表 y 座標(也代表 RPG 第 0 個參數的數字大小)。

綠龜 = 顏色龜類(1, .5)，與紅龜類似。

藍龜 = 顏色龜類(2, .5)，與紅龜類似。

呼叫函數 `set_bg_color()`，根據目前設定改變背景顏色。(初始設定為 `[0.5,0.5,0.5]`)

新宣告一個新的龜類，隱藏其龜指標並移動時不顯現軌跡，讓其前往到 `(1,1.15)` 位置，寫下資訊。

寫手.寫 `write("拖拉!",align="center",font=("Arial",30,("bold","italic")))`

之後回傳"事件迴圈"訊息。

```
class 顏色龜類(龜類):
```

宣告一個顏色龜類，設定其形狀、顏色、移動方式。其可用來控制背景顏色。

因其繼承了龜類，因此包含了龜類的屬性、方法。

★ *Class* (類) 與 *Inheritance* (繼承) 基本介紹，使用者自行定義一個物件的屬性(attribute)及方法(method)。

例如：

類別例子：

```
class 學生類:

    def __init__(我,名字,學號):

        我.名字 = 名字

        我.學號 = 學號

        我.學分 = 0

    def 通過(我,學分數):

        我.學分 +=學分數
```

創建一個學生類，裡面有學生類的屬性(我、名字、學號)以及方法(通過、改名)。

<pre> return 我.學分 def 改名(我, 新名): 我.名字= 新名 </pre>	
應用：	
<pre> 新生_1=學生類("王大明",22111) 新生_2=學生類("陳小生",22112) 新生_1.改名("王曉明") 新生_2.通過(3) for i in [新生_1,新生_2]: print("%s\t%d\t%d"%(i.名字,i.學 號,i.學分)) </pre>	<p>在這裡利用學生類創建了兩個學生，並將其之一改名，另一個通過3學分，再將其兩個屬性全部印出。</p> <p>印出如下：</p> <pre> 3 王曉明 22111 0 陳小生 22112 3 </pre>

Inheritance 介紹，子類可繼承父類，獲得其屬性、方法。在 python 中所有類都繼承於 Object 類，所以

<pre> class 學生類: pass </pre>	同等於	<pre> class 學生類(object): pass </pre>
-----------------------------------	-----	---

這裡沿用之前 類 的範例

<pre> class 新學生類(學生類): def __init__(我,名字,學號,年齡): 學生類.__init__(我,名字,學 號) </pre>	<p>這裡繼承了父類別【學生類】，產生新的子類別【新學生類】，其將父類別增加新的屬性(年</p>
--	--

我.年齡 = 年齡	齡)。
新生=新學生類("陳小可",22111,15) 新生.通過(6) 新生.改名("陳可可") print("%s\t%d\t%d\t%d"%(新生.名字, 新生.學號,新生.年齡,新生.學分))	創建一個新生，並呼叫其父類別的方法。 印出如下： 6 陳可可 22111 15 6

```

class 顏色龜類(龜類):

    def __init__(我, x, y):
        龜類.__init__(我)
        我.形狀(龜形)
        我.重設大小模式("user")
        我.形狀大小(3,3,5)
        我.筆粗(10)
        我._color = [0,0,0]
        我.x = x
        我._color[x] = y
        我.顏色(我._color)
        我.速度(0)
        我.左轉(90)
        我.提筆()
        我.前往(x,0)
        我.下筆()

```

```
我.設 y 座標(1)

我.提筆()

我.設 y 座標(y)

我.筆色("gray25")

我.在拖曳時(我.平移)
```

設定顏色龜類的初始設定，傳入 x、y 參數。

龜類.__init__(我)，繼承龜類的初始設定。

並設定其形狀、模式、形狀大小、筆粗。

我._color = [0,0,0]，此處設定了其屬性 _color 初始皆為 0。

我._color[x] = y，並設定其第 x 個的參數等於 y。

並根據其設定改變龜指標顏色，我.顏色(我._color)。

將速度設為 0，則不會有動作產生。因其一開始為向右，因此向左轉 90°。並 我.提筆()，不印出移動軌跡。前往位置(x,0)，我.下筆()，印出移動軌跡。

之後從 (x,0)座標到 (x,1)座標，畫出顏色龜類 y 座標移動範圍的一條線。

再 我.提筆()，回到傳入的(x,y)座標，我.設 y 座標(y)。

之後更改筆色 我.筆色("gray25")。

我.在拖曳時(我.平移)，最後設定當滑鼠拖曳顏色龜類時，連結方法 平移()
()，傳入參數為拖曳時的 x、y 座標。

```

class 顏色龜類(龜類):

    def 平移(我, x, y):

        我.設 y 座標(max(0,min(y,1)))

        我._color[我.x] = 我.y 座標()

        我.填色(我._color)

        設背景的颜色()

```

用來移動顏色龜類的 y 座標，傳入 x,y 座標參數。

```
我.設 y 座標(max(0,min(y,1)))
```

移動其 y 座標位置，其移動的 y 座標必須 $0 \leq y \leq 1$ ，如超過則為 1、較小則為 0。

我._color[我.x] = 我.y 座標()，並根據其 xy 座標改變其顏色設定，代表 RPG 第 x 個參數設為目前 y 座標。（我.x 不等於傳入參數 x，傳入參數 x 在此方法無使用）

將其龜指標內部填色，我.填色(我._color)。

且呼叫 設背景的颜色()，根據新的座標改使其背景顏色。

```

def 設背景的颜色():

    幕.背景色(紅龜.y 座標(), 綠龜.y 座標(), 藍龜.y 座標())

```

函數 def 設背景的颜色()，無傳入參數。

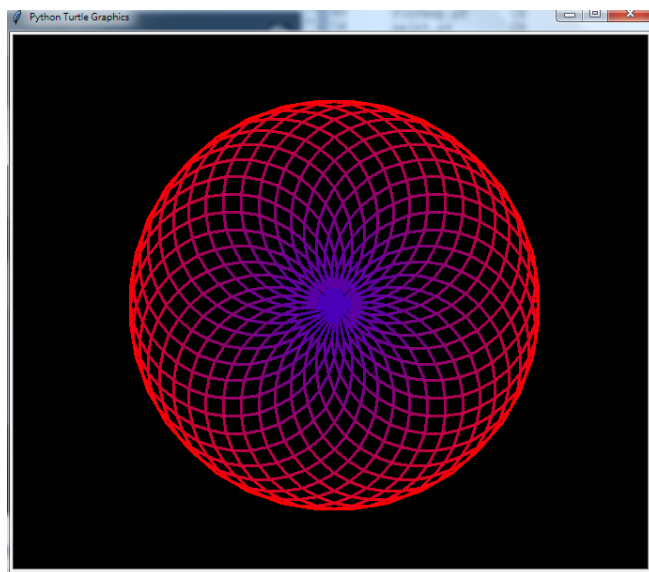
```
幕.背景色(紅龜.y 座標(), 綠龜.y 座標(), 藍龜.y 座標())
```

根據三個顏色龜類的 y 座標位置設定背景顏色：

7 視覺動畫 tc_wikipedia.py

此程式利用迴圈複製龜類，並進行動作。由中間向外逐漸右轉前進，並且逐漸改變顏色，最後會行成一個大圓，之後利用回復暫存區逐漸回復動作。在動作時由於我們將其每 36 步才更新螢幕一次，等於 36 個龜類皆往前一步時才進行更新，因此在視覺上像是 36 個龜類在同步平行進行動作，在螢幕上像是製作了一個小型簡短的視覺動畫。

7-1 執行結果



7-2 程式碼

```

1  '''龜作圖範例集：
2
3      tdemo_wikipedia3.py
4
5  本例是以 Turtle graphics 的維基百科文章為靈感。
6  (可參考維基百科網址)
7
8
9  首先先複製第一隻龜 p【畫筆】
10 創造另外 ne-1【線條數-1】(此例為 35) 隻龜。
11 之後讓全部龜平行運行。
12
13
14 最後將全部龜 undo()【回復()】
15 '''
16 from turtle_tc import *; from turtle_tc import 幕類, 龜類, 主迴圈
17 from time import clock, sleep
18
19 def 建立圖(畫筆, 線條數, sz):
20     龜列表 = [畫筆]
21     # 建立 線條數-1 個的其他海龜
22     for i in 範圍(1, 線條數):
23         新畫筆 = 畫筆.複製()
24         新畫筆.右轉(360.0/線條數)
25         龜列表.append(新畫筆)

```

```
26     畫筆 = 新畫筆
27     for i in 範圍(線條數):
28         c = abs(線條數/2.0-i)/(線條數*.7)
29         # 讓 36 個 海龜做下一步
30         # 平行:
31         for t in 龜列表:
32             t.右轉(360./線條數)
33             t.筆色(1-c,0,c)
34             t.前進(sz)
35
36 def 主函數():
37     螢幕 = 幕類()
38     螢幕.背景色(黑)
39     畫筆=龜類()
40     畫筆.速度(0)
41     畫筆.藏龜()
42     畫筆.筆色(紅)
43     畫筆.筆粗(3)
44
45     螢幕.追蹤(36,0)
46
47     at = clock()
48     建立圖(畫筆, 36, 19)
49     et = clock()
50     費時 = et-at
51
```

```

52     sleep(1)
53
54     at = clock()
55     while any([t.回復暫存區的個數() for t in 螢幕.龜群()]):
56         for t in 螢幕.龜群():
57             t.回復()
58     et = clock()
59     return "執行時間: %.3f 秒" % (費時+et-at)
60
61
62 if __name__ == '__main__':
63     訊息 = 主函數()
64     印(訊息)
65     主迴圈()

```

7-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```

if __name__ == '__main__':
    訊息 = 主函數()
    印(訊息)

```

主迴圈()

__name__ 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則__name__ 變數會等於'__main__'，程序便會執行其下列動作。

主函數()被呼叫執行，其為本程式內部函數，將其回傳的資料以 訊息 接收。之後將訊息印出。

主迴圈()被呼叫執行，其為 turtle_tc 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 mainloop()，等候使用者進一步使用滑鼠鍵盤來控制。

def 主函數():

主函數()，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

def 主函數():

螢幕 = 幕類()

螢幕.背景色(黑)

畫筆=龜類()

畫筆.速度(0)

畫筆.藏龜()

畫筆.筆色(紅)

畫筆.筆粗(3)


```

螢幕.追蹤(36,0)

at = clock()

建立圖(畫筆, 36, 19)

et = clock()

費時 = et-at

sleep(1)

at = clock()

while any([t.回復暫存區的個數() for t in 螢幕.龜群()]):

    for t in 螢幕.龜群():

        t.回復()

et = clock()

return "執行時間: %.3f 秒" % (費時+et-at)

```

宣告一個 幕類()，設置背景色為黑色。

並創建一個 龜類()。設置速度為0，即為最快速。

將 畫筆 指標隱藏。並設置其筆色為紅色。筆粗為3。

螢幕.追蹤(36,0)，設定螢幕更新為36步更新一次，延遲時間0秒。即

代表每36個畫筆行動時，便更新畫布一次。

用 at 記錄目前處理器時間。

呼叫 建立圖(畫筆, 36, 19)，開始繪圖。

再用 et 記錄目前處理器時間。

兩者相減可得到繪圖所費時間。

停止 1 秒不動作。

再用 at 記錄下目前處理器時間。

當螢幕上的任一個畫筆尚有動作存於回復暫存器中時，用 while 迴圈進行下列動作。

利用 for 迴圈抓取螢幕上的龜類，再利用得到的龜類抓取其回復暫存區的個數放進陣列裡，最後利用 any 得知還有沒有動作存於暫存器，沒有則回傳 False。

用 for 迴圈抓取螢幕上的所有龜群，並每個回復一個動作狀態。

重複以上步驟直到動作都回復完。

因此圖形會像倒帶一樣回到中間。

再用 et 記錄目前處理器時間。

之後回傳繪圖及回復動作所需時間。

```
def 建立圖(畫筆, 線條數, sz):  
    龜列表 = [畫筆]  
    # 建立 線條數-1 個的其他海龜  
    for i in 範圍(1, 線條數):  
        新畫筆 = 畫筆.複製()  
        新畫筆.右轉(360.0/線條數)  
        龜列表.append(新畫筆)  
    畫筆 = 新畫筆
```

```

for i in 範圍(線條數):

    c = abs(線條數/2.0-i)/(線條數*.7)

    # 讓 36 個 海龜做下一步

    # 平行:

    for t in 龜列表:

        t.右轉(360./線條數)

        t.筆色(1-c,0,c)

        t.前進(sz)

```

函數 建立圖(畫筆, 線條數, sz), 傳入參數 畫筆, 線條數, sz, 畫筆為第一隻可拿來複製, 線條數為總共需要的畫筆數量, sz 為每次前進的單數大小。

建立龜列表儲存畫筆及待會複製的畫筆們。

用 for 迴圈建立複製畫筆, 迴圈共會執行 35 次(線條數為 36), 即 1~35。

將畫筆複製, 並將其右轉(360.0/線條數)。之後將它增加到龜列表。

再將新畫筆取代畫筆。這樣下一次複製又會將再 右轉(360.0/線條數)。

重複執行, 直到 for 迴圈結束。

再創造新的 for 迴圈, 迴圈共會執行 36 次(線條數為 36), 即 0~35。

計算 c 數字, 作為顏色每次改變的大小變化, 使其顏色逐漸由藍色到紅色。

在裡面再建立 for 迴圈, 抓取 龜列表 內的畫筆, 因其有 36 個畫筆, 所以總共執行 36 次。

將畫筆 右轉(360.0/線條數) ，改變顏色，向指標方向 前進(sz)。

36 個畫筆都做完後，再回到外部迴圈，再計算一次 c，重新進入內部 for 迴圈。

直到外部迴圈 36 次結束。

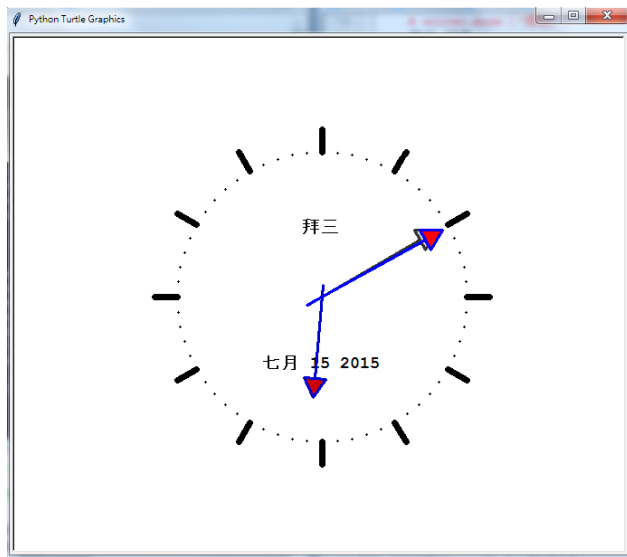
8 時鐘 tc_clock.py

時鐘是最早的發明之一，有了時鐘人類才可以更方便的計算時間。

自古以來人類發明了許多計算時間的設備，像沙漏、擺鐘等等，但這些對於時間的計算都不夠精細。如今我們運用電子時鐘可以增高時間計算的精準性。

在此程式我們運用龜作圖畫出時鐘鐘面，利用抓取電腦時間來得到現在的日期時間，並將之顯現。

8-1 執行結果



8-2 程式碼

```
1  #!/usr/bin/env python3
2  #  - * - 編碼：CP1252  - * -
3  '''龜作圖範例集：
4
5      tdemo_clock.py
6
7  時鐘程序，顯示日期
8  和時間
9
10     -----
11     按 STOP 退出程序！
12     -----
13  '''
14  from turtle_tc import *
```

```

14 from datetime import datetime
15
16 def 跳(距離, 角度=0):
17     提筆()
18     右轉(角度)
19     前進(距離)
20     左轉(角度)
21     下筆()
22
23 def 指針(長度, 尺寸大小):
24     前進(長度*1.15)
25     右轉(90)
26     前進(尺寸大小/2.0)
27     左轉(120)
28     前進(尺寸大小)
29     左轉(120)
30     前進(尺寸大小)
31     左轉(120)
32     前進(尺寸大小/2.0)
33
34 def 作指針形狀(名, 長度, 尺寸大小):
35     重設()
36     跳(-長度*0.15)
37     開始多邊形()
38     指針(長度, 尺寸大小)

```

```

39     結束多邊形 ()
40     指針形式 = 取多邊形 ()
41     登記形狀 (名, 指針形式)
42
43 def 鐘面 (半徑):
44     重設 ()
45     筆粗 (7)
46     for i in 範圍 (60):
47         跳 (半徑)
48         if i % 5 == 0:
49             前進 (25)
50             跳 (-半徑-25)
51         else:
52             點 (3)
53             跳 (-半徑)
54         右轉 (6)
55
56 def 設立 ():
57     global 秒針, 分針, 時針, 寫手
58     模式 (角度從北開始順時針)
59     作指針形狀 ("秒針", 125, 25)
60     作指針形狀 ("分針", 130, 25)
61     作指針形狀 ("時針", 90, 25)
62     鐘面 (160)
63     秒針 = 龜類 ()

```

```

64     秒針.形狀("秒針")
65     秒針.顏色("gray20", "gray80")
66     分針 = 龜類()
67     分針.形狀("分針")
68     分針.顏色("blue1", "red1")
69     時針 = 龜類()
70     時針.形狀("時針")
71     時針.顏色("blue3", "red3")
72     for 指針 in 秒針, 分針, 時針:
73         指針.重設大小模式("user")
74         指針.形狀大小(1, 1, 3)
75         指針.速度(0)
76     藏龜()
77     寫手 = 龜類()
78     # writer.mode("標誌")
79     寫手.藏龜()
80     寫手.提筆()
81     寫手.後退(85)
82
83     def 星期標籤(t):
84         星期標籤 = ["拜一", "拜二", "拜三",
85                     "拜四", "拜五", "拜六", "拜日"]
86         return 星期標籤[t.weekday()]
87
88     def 年月日期(z):

```



```

89     月份標籤 = ["一月", "二月", "三月", "四月", "五月", "六月",
90                 "七月", "八月", "九月", "十月", "十一月", "十二月"]
91     年 = z.year
92     月 = 月份標籤[z.month - 1]
93     日 = z.day
94     return "%s %d %d" % (月, 日, 年)
95
96 def 滴答():
97     t = datetime.today()
98     秒數 = t.second + t.microsecond*0.000001
99     分數 = t.minute + 秒數/60.0
100    時數 = t.hour + 分數/60.0
101    try:
102        追蹤(假) # Terminator 可能發生在這裡
103        寫手.清除()
104        寫手.回家()
105        寫手.前進(65)
106        寫手.寫(星期標籤(t),
107                align="center", font=("Courier", 14, "bold"))
108        寫手.後退(150)
109        寫手.寫(年月日期(t),
110                align="center", font=("Courier", 14, "bold"))
111        寫手.前進(85)
112        追蹤(真)
113        秒針.設頭向(6*秒數) # 或這裡

```

```

114         分針.設頭向(6*分數)
115         時針.設頭向(30*時數)
116         追蹤(真)
117         在計時後(滴答, 100)
118     except Terminator:
119         pass # 使用者按下 STOP
120
121 def 主函數():
122     追蹤(假)
123     設立()
124     追蹤(真)
125     滴答()
126     return "事件迴圈"
127
128 if __name__ == "__main__":
129     模式(角度從北開始順時針)
130     訊息 = 主函數()
131     印(訊息)
132     主迴圈()

```

8-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```
if __name__ == "__main__":  
    模式 (角度從北開始順時針)  
  
    訊息 = 主函數()  
  
    印 (訊息)  
  
    主迴圈()
```

__name__ 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則__name__ 變數會等於'__main__'，程序便會執行其下列動作。

設定角度單位從北開始順時針計算。

主函數() 被呼叫執行，其為本程式內部函數，將其回傳的資料以 訊息 接收。之後將訊息印出。

主迴圈() 被呼叫執行，其為 turtle_tc 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 mainloop()，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數()，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():
```

```
追蹤(假)

設立()

追蹤(真)

滴答()

return "事件迴圈"
```

宣告全域變數。

追蹤(假)，關閉龜指標繪製。

呼叫函數 設立()，製作秒針、分針、時針形狀大小。

追蹤(真)，開啟龜指標動畫繪製。

呼叫函數 滴答()，繪製時鐘動畫。

最後回傳 "事件迴圈" 字串。

```
def 設立():

    global 秒針, 分針, 時針, 寫手

    模式(角度從北開始順時針)

    作指針形狀("秒針", 125, 25)

    作指針形狀("分針", 130, 25)

    作指針形狀("時針", 90, 25)

    鐘面(160)

    秒針 = 龜類()

    秒針.形狀("秒針")

    秒針.顏色("gray20", "gray80")

    分針 = 龜類()

    分針.形狀("分針")
```

```

分針.顏色("blue1", "red1")

時針 = 龜類()

時針.形狀("時針")

時針.顏色("blue3", "red3")

for 指針 in 秒針, 分針, 時針:

    指針.重設大小模式("user")

    指針.形狀大小(1, 1, 3)

    指針.速度(0)

藏龜()

寫手 = 龜類()

# writer.mode("標誌")

寫手.藏龜()

寫手.提筆()

寫手.後退(85)

```

函數 設立(), 無傳入參數, 建立時鐘鐘面及指針形狀。

宣告三個指針及寫手為全域變數。

設定 角度從北開始順時針 計算。

呼叫 作指針形狀("秒針", 125, 25), 製作秒針的指針形狀。

呼叫 作指針形狀("分針", 130, 25), 製作分針的指針形狀。

呼叫 作指針形狀("時針", 90, 25), 製作時針的指針形狀。

呼叫 鐘面(160), 製作時鐘面。

建立三個龜類，形狀分別為秒針、分針、時針，且設定其顏色，秒針為深灰及淺灰色、分針及時針為藍色及紅色。

用 for 迴圈抓取三個指針，設定其形狀大小及速度。

指針.形狀大小(1, 1, 3)，設定其延展因子為 1:1(垂直:指標方向)，輪廓寬度為 3。

速度設為 0，即最快速(移動時無動畫產生)。

藏龜()，隱藏內建的龜指標。

製造新的龜類名為 寫手，並隱藏其指標、提筆(移動時不出現軌跡)，且後退 85。

def 作指針形狀(名, 長度, 尺寸大小):

重設()

跳(-長度*0.15)

開始多邊形()

指針(長度, 尺寸大小)

結束多邊形()

指針形式 = 取多邊形()

登記形狀(名, 指針形式)

函數，傳入參數，根據傳入的大小製作指針並將之形狀登記。

呼叫函數 跳(-長度*0.15)，往後退幾步。

開始多邊形()，開始記錄多邊形。

呼叫函數 指針(長度, 尺寸大小)，開始根據傳入參數畫出指針形狀。

結束多邊形 ()，結束記錄多邊形。

指針形式 = 取多邊形 ()，將剛記錄的多邊形取出。

登記形狀 (名, 指針形式)，並且將形狀登記取名。

```
def 跳 (距離, 角度=0):
```

```
    提筆 ()
```

```
    右轉 (角度)
```

```
    前進 (距離)
```

```
    左轉 (角度)
```

```
    下筆 ()
```

函數，傳入參數 距離、 角度=0，當無傳入角度參數則預設為 0。其等

於前進到某距離，但不要顯現前進軌跡

提筆 ()，移動時不畫線。

根據 角度 右轉。並前進 距離。

再左轉 角度，回到原本方向。

下筆 ()，移動時會畫出移動軌跡。

```
def 指針 (長度, 尺寸大小):
```

```
    前進 (長度*1.15)
```

```
    右轉 (90)
```

```
    前進 (尺寸大小/2.0)
```

```
    左轉 (120)
```

```
    前進 (尺寸大小)
```

```
    左轉 (120)
```

```
前進(尺寸大小)

左轉(120)

前進(尺寸大小/2.0)
```

函數 指針(長度, 尺寸大小)，傳入參數 長度、尺寸大小。根據長度及尺寸大小畫出指針。

先前進 $1.15 * \text{長度}$ ，因之前後退了 $0.15 * \text{長度}$ ，畫出指針線。

左轉 90 度準備畫指針頭。前進一半的 尺寸大小，再 左轉(120)，前進(尺寸大小)，再 左轉(120)，前進(尺寸大小)，最後左轉(120)，前進(尺寸大小/2.0)，畫出最後一邊剩下的一半尺寸大小。其等於畫出一個正三角形，邊長為 尺寸大小。

```
def 鐘面(半徑):

    重設()

    筆粗(7)

    for i in 範圍(60):

        跳(半徑)

        if i % 5 == 0:

            前進(25)

            跳(-半徑-25)

        else:

            點(3)

            跳(-半徑)

    右轉(6)
```

函數 鐘面(半徑)，傳入參數 半徑，在螢幕上製造時鐘鐘面。

重設()，將龜指標回歸原位。

設置筆粗度為 7。

建立 for 迴圈，重複執行 60 次(0~59)，因為鐘面共有 60 個點加線。

呼叫 跳(半徑)，將位置移到要畫線的地方。

if i % 5 == 0，如果 i 除以 5 餘數等於 0，等於為整點，所以 前進(25)，

畫線，之後 跳(-半徑-25)，等於後退回圓心位置。

else:，如果不是整點位置，那就畫圓點，點(3)。再 跳(-半徑)，回到圓

心位置。

之後右轉 6 度，因鐘面總共 60 個點，所以 $360^\circ/60=6^\circ$ 。

重複以上 for 迴圈動作直到 60 個點線畫完，鐘面完成。

def 滴答():

```
t = datetime.today()
```

```
秒數 = t.second + t.microsecond*0.000001
```

```
分數 = t.minute + 秒數/60.0
```

```
時數 = t.hour + 分數/60.0
```

try:

```
追蹤(假) # Terminator 可能發生在這裡
```

```
寫手.清除()
```

```
寫手.回家()
```

```
寫手.前進(65)
```

```
寫手.寫(星期標籤(t),
```

```
align="center", font=("Courier", 14, "bold"))
```

```

寫手.後退(150)

寫手.寫(年月日期(t),

            align="center", font=("Courier", 14, "bold"))

寫手.前進(85)

追蹤(真)

秒針.設頭向(6*秒數) # 或這裡

分針.設頭向(6*分數)

時針.設頭向(30*時數)

追蹤(真)

在計時後(滴答, 100)

except Terminator:

    pass # 使用者按下 STOP

```

函數 滴答()，無傳入參數。製造時鐘。

`t = datetime.today()`，抓取今天的日期時間放入 `t`。

`秒數 = t.second + t.microsecond*0.000001`，抓取秒數及毫秒，並將毫秒

換算成秒數單位，因在顯示上差距可能在點間距上。

`分數 = t.minute + 秒數/60.0`，抓取分數，再加上秒數換算成分數，因分

數在顯示上差距可能在點間距上。

`時數 = t.hour + 分數/60.0`，抓取時數，再加上分數換算成時數，因時

數在顯示上差距可能在點間距上。

用 `try` 指令，當發生意外時則跳到 `except` 執行。

設置 追蹤(假)，關閉龜指標動畫繪製。

清除 寫手 所畫的圖。並回到原點座標，且指標方向回到初始狀態。(此處由於角度從北開始計算，所以此處為向上)

寫手.前進(85)。

呼叫函數 星期標籤(t)，傳入 t，得到星期資訊並寫下。

寫手.後退(150)

呼叫函數 年月日期(t)，傳入 t，得到日期資訊並寫下。

寫手.前進(85)，回到圓心位置。

設置 追蹤(假)，開啟龜指標動畫繪製。

根據秒數*6° 得到角度，即秒針該指向位置。

根據分數*6° 得到角度，即分針該指向位置。

根據時數*6°*5 得到角度，因時數間距為 5，即時針該指向位置。

在計時後(滴答, 100)，過 100 毫秒後，呼叫 滴答() 函數。

```
def 星期標籤(t):  
    星期標籤 = ["拜一", "拜二", "拜三",  
                "拜四", "拜五", "拜六", "拜日"]  
    return 星期標籤[t.weekday()]
```

函數 星期標籤(t)，傳入參數 t。根據 t 得到今天星期資訊。

設置標籤陣列，存有星期字串。

根據 t.weekday() 回傳星期字串。

(如 t.weekday() 為 0，則為"拜一")

```
def 年月日期(z):
    月份標籤 = ["一月", "二月", "三月", "四月", "五月", "六月",
                "七月", "八月", "九月", "十月", "十一月", "十二月"]

    年 = z.year

    月 = 月份標籤[z.month - 1]

    日 = z.day

    return "%s %d %d" % (月, 日, 年)
```

函數 年月日期(z)，傳入參數 z。根據 z 得到今天日期資訊。

設置標籤陣列，存有月份字串。

根據 z.month-1 回傳星期字串。

(如 z.month-1 為 0，則為"一月")

最後回傳 日月年資訊。

9 三星運動 tc_planet_and_moon.py

這支程式運用 牛頓重力定律 來模擬太陽、地球、月亮的三星運動，不需求解複雜的微分方程式，僅用「步進」方式，就可做出「三星互繞」的軌跡動畫，十分有趣。在程式中，每個星體都會相互受到彼此重力的影響，也會受到本身與太陽的位置導致本身星體明、暗顏色位置變化。

以重力定律的公式為起點，做些簡單的推導如下：

$$\vec{F} = m\vec{a} = \frac{GMm}{|\vec{r}|^3} \cdot \vec{r}$$

當某星體 m 受到 其他星體 M 的重力吸引時，重力強度與質量乘積成正比，與距離平方成反比，其中， \vec{r} 代表起點為 m ，終點為 M 的位移(距離)向量，通常可表示為： $\vec{r} = \vec{s}_M - \vec{s}_m$ ， G 為重力常數 數值約為 $6.67 \times 10^{-11} \text{ N} \cdot (\text{m/kg})^2$ ，當某星體 m 受到 其他星體 M 的重力吸引時， m 會產生一個加速度：

$$\vec{a} = \frac{GM}{|\vec{r}|^3} \cdot \vec{r}$$

如果 m 受到不只一個星體的重力，就累加來自所有其他星體的重力效果。

$$\vec{a}_i = \frac{GM_i}{|\vec{r}_i|^3} \cdot \vec{r}_i$$

$$\vec{a} = \sum_{\forall i} \vec{a}_i$$

需用到所有星體的質量 M_i

算出本星體的加速度之後，再套用 牛頓運動學 公式如下：

$$d\vec{v} = \vec{a} \cdot dt$$

$$d\vec{s} = \vec{v} \cdot dt$$

$$\vec{v}(t + dt) = \vec{v}(t) + d\vec{v}$$

$$\vec{s}(t + dt) = \vec{s}(t) + d\vec{s}$$

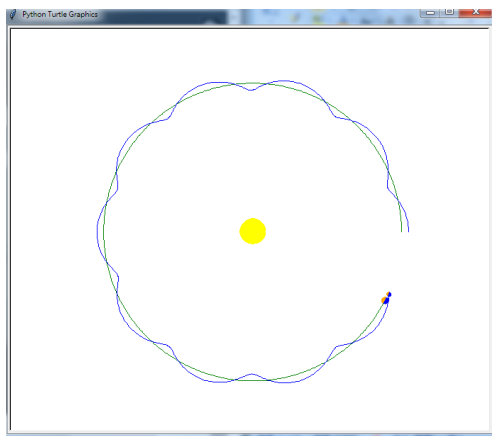
在 $t=0$ 時，指定 初位置 以及 初速度：

$$\vec{s}(0) = \vec{s}_0$$

$$\vec{v}(0) = \vec{v}_0$$

不斷套用以上的(遞迴)公式(3), (4)，所有星體的位置 $\vec{s}(t)$ ，就可以個別完全得知。最後，運用做圖指令，隨時間的前進，在個別位置 $\vec{s}(t)$ 上，畫出每一星體，就可做出動畫了。如果 m 受到不只一個星體的重力，就累加它們。在此需用到所有星體的質量 M_i ，不斷套用以上的(遞迴)公式，所有星體的位置就可以個別完全得知。最後，運用做圖指令，隨時間的前進，在個別位置上，畫出每一星體，就可做出動畫了。

9-1 執行結果



9-2 程式碼

```
1 #!/usr/bin/env python3
2 '''龜作圖範例集：
3
4     tdemo_planets_and_moon.py
5
```

重力系統模擬使用費曼物理學講義的進似法在龜作圖上。

(Feynman-lectures,p.9-8)

例如：沉重的太陽，較輕的地球，最輕的月亮。

地球有一個圓行的軌道運行，月亮按照穩定的軌道環繞地球。

你可以藉由按下滑鼠左鍵暫時保持畫布上的滾動運動。

'''

from turtle_tc import *; from turtle_tc import 形狀類, 龜類, 主迴圈, 二維向量類 as 向量類

from time import sleep

G 常數 = 8

class 重力系統類(object):

def __init__(我):

我.行星們 = []

我.t = 0

我.dt = 0.01

def 起始化(我):

for 星 in 我.行星們:

```

31         星.起始化()
32     def 開始(我):
33         for i in 範圍(10000):
34             我.t += 我.dt
35             for 星 in 我.行星們:
36                 星.步進()
37
38 class 星類(龜類):
39     def __init__(我, m, x, v, 重力系統, 形狀):
40         龜類.__init__(我, shape=形狀)
41         我.提筆()
42         我.m = m
43         我.設位置(x)
44         我.v = v
45         重力系統.行星們.append(我)
46         我.重力系統 = 重力系統
47         我.重設大小模式("user")
48         我.下筆()
49     def 起始化(我):
50         dt = 我.重力系統.dt
51         我.a = 我.加速度()
52         我.v = 我.v + 0.5*dt*我.a
53     def 加速度(我):
54         a = 向量類(0,0)
55         for 行星 in 我.重力系統.行星們:

```



```

56         if 行星 != 我:
57             v = 行星.位置() - 我.位置()
58             a += (G 常數 * 行星.m / abs(v) ** 3) * v
59         return a
60     def 步進(我):
61         dt = 我.重力系統.dt
62         我.設位置(我.位置() + dt * 我.v)
63         if 我.重力系統.行星們.index(我) != 0:
64             我.設頭向(我.朝向(我.重力系統.行星們[0]))
65         我.a = 我.加速度()
66         我.v = 我.v + dt * 我.a
67
68     # 創建複合黃色/藍色烏龜形狀的行星
69
70     def 主函數():
71         龜 = 龜類()
72         龜.重設()
73         龜.取幕().追蹤(0, 0)
74         龜.藏龜()
75         龜.提筆()
76         龜.前進(6)
77         龜.左轉(90)
78         龜.開始多邊形()
79         龜.畫圓(6, 180)
80         龜.結束多邊形()

```

```

81  形狀一 = 龜.取多邊形 ()
82  龜.開始多邊形 ()
83  龜.畫圓 (6,180)
84  龜.結束多邊形 ()
85  形狀二 = 龜.取多邊形 ()
86
87  行星形狀 = 形狀類 ("compound")
88  行星形狀.加成員 (形狀一,橙)
89  行星形狀.加成員 (形狀二,藍)
90  龜.取幕 ().登記形狀 ("行星", 行星形狀)
91  龜.取幕 ().追蹤 (1,0)
92
93  # 設立 重力系統
94  重力系統 = 重力系統類 ()
95  日 = 星類 (1000000, 向量類 (0,0), 向量類 (0,-2.5), 重力系統, "circle")
96  日.顏色 (黃)
97  日.形狀大小 (1.8)
98  日.提筆 ()
99  地球 = 星類 (12500, 向量類 (210,0), 向量類 (0,195), 重力系統, "行星")
100  地球.筆色 (綠)
101  地球.形狀大小 (0.8)
102  月球 = 星類 (1, 向量類 (220,0), 向量類 (0,295), 重力系統, "行星")
103  月球.筆色 (藍)
104  月球.形狀大小 (0.5)
105  重力系統.起始化 ()

```

```
106     重力系統.開始()  
107     return "結束!"  
108  
109 if __name__ == '__main__':  
110     主函數()  
111     主迴圈()  
112
```

9-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```
G 常數 = 8
```

在外部設定的變數 G 常數。因其宣告在最外部，因此在此程式中可任意存取此數。

G常數是牛頓重力常數。作者令其為 8，是一個隨便設定的常數，並沒有照真正的物理量來設定。（原數值為 $6.67e-11$ ）

```
if __name__ == '__main__':  
    主函數()  
    主迴圈()
```

`__name__` 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則 `__name__` 變數會等於 `'__main__'`，程序便會執行其下列動作。

主函數 `()` 被呼叫執行，其為本程式內部函數。

主迴圈 `()` 被呼叫執行，其為 `turtle_tc` 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 `mainloop()`，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數 `()`，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():
```

```
    龜 = 龜類()

    龜.重設()

    龜.取幕().追蹤(0,0)

    龜.藏龜()

    龜.提筆()

    龜.前進(6)

    龜.左轉(90)

    龜.開始多邊形()

    龜.畫圓(6, 180)

    龜.結束多邊形()

    形狀一 = 龜.取多邊形()
```

```
龜.開始多邊形()  
  
龜.畫圓(6,180)  
  
龜.結束多邊形()  
  
形狀二 = 龜.取多邊形()  
  
行星形狀 = 形狀類("compound")  
  
行星形狀.加成員(形狀一,橙)  
  
行星形狀.加成員(形狀二,藍)  
  
龜.取幕().登記形狀("行星", 行星形狀)  
  
龜.取幕().追蹤(1,0)
```

建立一個龜類為龜。

設其幕類為設置 追蹤(0,0) ，設置為 0 則為不更新螢幕。將之龜指標隱藏並提筆(移動不畫出軌跡)。

龜.前進(6) ，並 龜.左轉(90) ， 龜.開始多邊形() ，開始記錄其畫出的多邊形。

龜.畫圓(6, 180) ，畫半徑為 6 的半圓。

龜.結束多邊形() ，結束記錄其畫出的多邊形。

形狀一 = 龜.取多邊形() ，取得剛畫出的多邊形放入 形狀一。

龜.開始多邊形() ，再開始記錄其畫出的多邊形。

龜.畫圓(6, 180) ，畫半徑為 6 的另一半半圓。

龜.結束多邊形() ，結束記錄其畫出的多邊形。

形狀二 = 龜.取多邊形() ，取得剛畫出的多邊形放入 形狀二。

設置行星形狀為組合形狀。並將形狀一及形狀二加入，一個為橙色一個為藍色。

將其形狀登記為"行星"。

龜.取幕().追蹤(1,0)，開始追蹤龜動態。

```
# 設立 重力系統

重力系統 = 重力系統類()

日 = 星類(1000000, 向量類(0,0), 向量類(0,-2.5), 重力系統, "circle")

日.顏色(黃)

日.形狀大小(1.8)

日.提筆()

地球 = 星類(12500, 向量類(210,0), 向量類(0,195), 重力系統, "行星")

地球.筆色(綠)

地球.形狀大小(0.8)

月球 = 星類(1, 向量類(220,0), 向量類(0,295), 重力系統, "行星")

月球.筆色(藍)

月球.形狀大小(0.5)

重力系統.起始化()

重力系統.開始()

return "結束!"
```

建立一個 重力系統。

建立 日為 星類，將其加入重力系統，形狀為圓形，顏色為黃色、大小為 1.8。

建立 地球 為 星類，將其加入重力系統，其形狀為"行星"，移動顏色為綠色、大小為 0.8。

建立 月球 為 星類，將其加入重力系統，其形狀為"行星"，移動顏色為藍色、大小為 0.5。

重力系統.起始化()，將重力系統初始化。

重力系統.開始()，並讓系統運作開始。

最後回傳 "事件迴圈" 字串。

```
class 重力系統類(object):
```

建立一個 class，為重力系統或稱星系，其存儲行星並設定全部行星資訊。

在這個 重力系統類 中，主要的資料成員是 行星們，用來存放行星名稱。

主要的方法為 開始()，用來讓行星開始運轉。一旦呼叫了 開始()，行星們內的每個行星成員，都會做 步進() 步進動作，而且重複這種動作 10000 次

```
class 重力系統類(object):
```

```
    def __init__(我):
```

```
        我.行星們 = []
```

```
        我.t = 0
```

```
        我.dt = 0.01
```

重力系統初始設定，無傳入參數，設置一個空的陣列用來存行星。

且設定t（星系的絕對時間，起始為0）與dt（時間間隔，每次星體步進的微量時間）。

```
class 重力系統類(object):  
  
    def 起始化(我):  
  
        for 星 in 我.行星們:  
  
            星.起始化()
```

方法 起始化(我)，無傳入參數。將重力系統內全部行星起始化。

用 for 迴圈抓取 我.行星們 陣列內的行星，並將其全部 星.起始化()。

```
class 重力系統類(object):  
  
    def 開始(我):  
  
        for i in 範圍(10000):  
  
            我.t += 我.dt  
  
            for 星 in 我.行星們:  
  
                星.步進()
```

方法 開始(我)，無傳入參數。設定重力系統內全部 星類 開始運作。太陽系正式運轉的指令。

利用 for 迴圈重複執行 10000 次(0~9999)。

每執行一次時間 t 增加 dt。

再藉由 for 迴圈抓取 行星們 內部 星類，讓每個行星運作。重複執行動作，直到 for 迴圈結束。

```
class 星類(龜類):
```


建立一個 class，繼承龜類的屬性方法，用來製造太陽、地球、月亮等星體，設置星類的動作規律及軌跡，並設定星體形狀及所屬星系。

```
class 星類(龜類):  
  
    def __init__(我, m, x, v, 重力系統, 形狀):  
  
        龜類.__init__(我, shape=形狀)  
  
        我.提筆()  
  
        我.m = m  
  
        我.設位置(x)  
  
        我.v = v  
  
        重力系統.行星們.append(我)  
  
        我.重力系統 = 重力系統  
  
        我.重設大小模式("user")  
  
        我.下筆()
```

星類初始設定，傳入參數 `m`, `x`, `v`, `重力系統`, `形狀`，`m` 為質量，`x` 為初始位置，`v` 為速度向量，`重力系統` 為要加入的 `重力系統類`，`形狀` 為行星的龜形名稱。

其繼承龜類，龜形狀為傳入參數 `形狀`。

設定星類本身參數。（`m`、`v`）

`我.設位置(x)`，設定星類本身位置。

`重力系統.行星們.append(我)`，將自身加入重力系統的行星們陣列裡。

設置 `我.下筆()`，移動時會畫出移動軌跡。

```
class 星類(龜類):
```

```
def 起始化(我):  
  
    dt = 我.重力系統.dt  
  
    我.a = 我.加速度()  
  
    我.v = 我.v + 0.5*dt*我.a
```

方法 起始化()，設定時間間距、計算加速度及速度。

```
class 星類(龜類):  
  
    def 加速度(我):  
  
        a = 向量類(0,0)  
  
        for 行星 in 我.重力系統.行星們:  
  
            if 行星 != 我:  
  
                v = 行星.位置()-我.位置()  
  
                a += (G 常數*行星.m/abs(v)**3)*v  
  
        return a
```

方法 加速度()，計算加速度。算出星體目前所受的其他星體重力因而造成的加速度，運用了牛頓力學重力公式 $a = GM/r^{**2}$ ，在此為二維向量的形式。

宣告加速度向量為 (0,0)。

用 for 迴圈抓取重力系統內所有 行星們 陣列的 星類，並執行以下動作。

如果抓取的行星不是本身星類，則會遭受影響，因此需要計算。

將 行星 位置扣掉自身位置得到速度，並利用重力公式算出加速度，

但由於星體會受到「所有」其他星體的作用，故有「累加」的動作。

所有行星都執行過後，for 迴圈結束。

將算出的加速度 a 回傳。

```
class 星類(龜類):  
  
    def 步進(我):  
  
        dt = 我.重力系統.dt  
  
        我.設位置(我.位置() + dt*我.v)  
  
        if 我.重力系統.行星們.index(我) != 0:  
  
            我.設頭向(我.朝向(我.重力系統.行星們[0]))  
  
        我.a = 我.加速度()  
  
        我.v = 我.v + dt*我.a
```

方法 `步進()`，讓 `星類` 開始運作。運用運動學原理，位置、速度以及加速度之關係，計算出下一刻的位置，然後運用龜作圖，畫出圖形，呈現出動畫效果。

運動學原理：

速度的改變 = 加速度 * 微量時間	$dv = a * dt$
位置的改變 = 速度 * 微量時間	$ds = v * dt$
下一刻的速度 = 現在的速度 + 速度的改變	$v = v + dv$
下一刻的位置 = 現在的位置 + 位置	$s = s + ds$

的改變	
-----	--

$dt = \text{我.重力系統}.dt$ ，抓取時間間隔，星體步進的微量時間。

並設定位置為原本位置加上微量時間乘上速度。

之後查看是否本身星體為太陽（太陽處在行星們的位置 0），如不是的話，則進行動作，將星體的頭部（亮的那面）朝向太陽（行星們[0]）。

呼叫方法 `加速度()`，重新計算加速度。

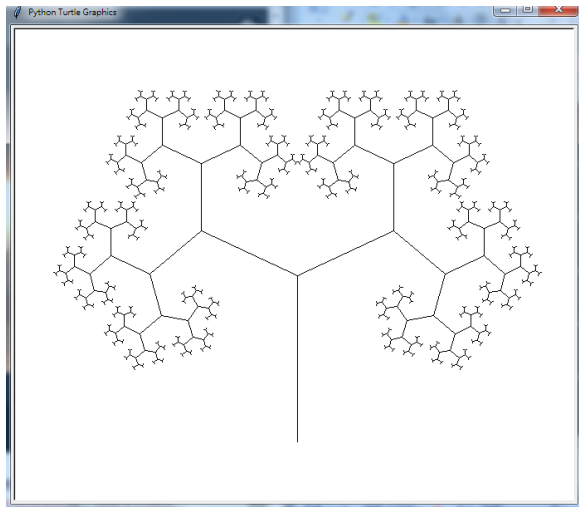
$\text{我}.v = \text{我}.v + dt * \text{我}.a$ ，並將速度加上微量時間乘上加速度，得到新的速度。

10 畫一顆樹 `tc_tree.py`

利用龜作圖畫一顆樹，樹中固定分為兩個分支，每次分支角度也固定，再固定以同樣的倍率縮短分支長度，持續到其長度小於等於 3 為止，因此此樹為完全對稱。當樹畫完時，印出花費所需時間及螢幕所有的龜群數量。

此程式利用遞迴函數製作樹，藉由反覆複製龜類來製造分支，所複製得龜類位置、屬性會完全一樣，每遇到一個分支點便複製一個新的龜類。

10-1 執行結果



10-2 程式碼

```
1  #!/usr/bin/env python3
2  '''龜作圖範例集：
3
4      tdemo_tree.py
5
6  顯示'廣度優先樹'。
7  相對於經典的 Logo 樹繪圖程式，其為'深度優先演算法'。
8
9
10  用途：
11  （1） 一個樹的生成器，
12  邊產生邊畫圖，
13  其生成器持續產生 無。
14
15  （2） 複製龜：
```

```

15 在每個分支點複製當前的龜。
16 所以最終有 1024 隻龜。
17 '''
18 from turtle_tc import *; from turtle_tc import 龜類, 主迴圈
19 from time import clock
20
21 def 樹(p 列表, l, a, f):
22     '''p 列表 是筆的清單
23     l 為分支長度
24     a 是 2 個分支之間的角度
25     f 是由一層分支到另一層分支的縮短因子
26     '''
27     if l > 3:
28         列表 = []
29         for p in p 列表:
30             p.前進(l)
31             q = p.複製()
32             p.左轉(a)
33             q.右轉(a)
34             列表.append(p)
35             列表.append(q)
36         for x in 樹(列表, l*f, a, f):
37             yield 無
38
39 def 製造樹():

```

```

40     p = 龜類()
41     p.設回復暫存區(無)
42     p.藏龜()
43     p.速度(0)
44     p.取幕().追蹤(30,0)
45     p.左轉(90)
46     p.提筆()
47     p.前進(-210)
48     p.下筆()
49     t = 樹([p], 200, 65, 0.6375)
50     for x in t:
51         pass
52     印(len(p.取幕().龜群()))
53
54 def 主函數():
55     時間一=clock()
56     製造樹()
57     時間二=clock()
58     return "結束: %.2f 秒" % (時間二-時間一)
59
60 if __name__ == "__main__":
61     訊息 = 主函數()
62     印(訊息)
63     主迴圈()

```

10-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```
if __name__ == "__main__":  
    訊息 = 主函數()  
    印(訊息)  
    主迴圈()
```

__name__ 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則__name__變數會等於'__main__'，程序便會執行其下列動作。

主函數()被呼叫執行，其為本程式內部函數，將其回傳的資料以 訊息 接收。之後將訊息印出。

主迴圈()被呼叫執行，其為 turtle_tc 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 mainloop()，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數()，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():
```



```

時間一=clock()

製造樹()

時間二=clock()

return "結束: %.2f 秒" % (時間二-時間一)

```

用 時間一 記錄下目前處理器時間。

呼叫函數 製造樹()，開始畫樹。

再用 時間二 記錄下目前處理器時間。

之後回傳繪圖所需時間（將兩時間相減）。

```

def 製造樹():

    p = 龜類()

    p.設回復暫存區(無)

    p.藏龜()

    p.速度(0)

    p.取幕().追蹤(30,0)

    p.左轉(90)

    p.提筆()

    p.前進(-210)

    p.下筆()

    t = 樹([p], 200, 65, 0.6375)

    for x in t:

        pass

    印(len(p.取幕().龜群()))

```

函數 製造樹()，無傳入參數。

創造一個龜類。

設回復暫存區(無)，將其回復暫存區關閉。

並將其指標隱藏，設置速度為 0（最快速，指標無移動動作，直接到結數位置）。

取幕()，抓取其螢幕，並設置螢幕 追蹤(30,0)，即螢幕每 30 次動作更新一次、延遲時間 0 毫秒。

設置左轉 90 度，此時龜指標方向往上。提筆()，移動時不畫出軌跡。

並 前進(-210)（等於 後退(210)）。

下筆()，移動時畫出軌跡。

呼叫樹，傳入龜類、長度、角度及縮短因子。

因其函數 樹() 內含產生器 *yield*，因此藉由 for 迴圈重複接收回傳的數值，並 *pass* 持續執行。(yield、pass 解釋可參考下方)

藉由 p.取幕()，抓取其螢幕，並再根據其螢幕抓取 龜群()，抓取其螢幕上全部龜類，再藉由 len 計算其總共有多少隻龜類在螢幕上並印出。

★ *yield* 與 *return* 類似，差別在於其會持續執行函數。當執行函數

遇到 *yield*，它會先回傳，但此函數並未結束，他會等待下次呼叫再持續執行。因此當呼叫該函數時，其會回傳成為一個產生器物件 (generator object)，所以需要持續接受使其生成。其也

可藉由 send 指令傳送參數到產生器，其使用 send 指令會先傳送參數，再持續執行函數程式到下一個 yield 指令。

範例：

利用 for 接收產生器數值	
<pre>def 計算(): 數字 =1 yield 數字 數字+=1 yield 數字 數字+=1 yield 數字 for i in 計算(): print(i)</pre>	<p>其印出結果為：</p> <p>1</p> <p>2</p> <p>3</p> <p>因其 yield 為三次，因此 for 迴圈重複三次，將其產生結果印出。</p>
利用 next 方法(函數在上方)	
<pre>g=計算() print(g) print(next(g)) print(next(g)) print(next(g)) print(next(g))</pre>	<p>其印出結果為：</p> <p><generator object 計算 at 0x023FA698></p> <p>1</p> <p>2</p> <p>3</p> <p>Traceback (most recent call last):</p> <p>File "<stdin>", line 1, in</p>

	<p><module></p> <p>StopIteration</p> <p>呼叫函數 計算，回傳為 g，g 為一個產生器物件（generator object）。</p> <p>因其 yield 為三次，因此最後一次已超出範圍，所以印出 StopIteration。</p>
<p>可藉由 send 傳送數值</p>	
<pre>def 計算 2 (數字) : 數字 +=1 yield 數字 數字 +=1 數字 =yield 數字 yield 數字 g=計算 2 (2) print (next (g)) print (next (g)) print (g.send(100))</pre>	<p>其印出結果為：</p> <p>3</p> <p>4</p> <p>100</p> <p>呼叫函數 計算 2，傳入數字 2，回傳為 g。</p> <p>第一次 yield 為 3，因其前面已加 1。</p> <p>第二次 yield 為 4，因其前面又加 1。</p>

	<p>因其先傳入參數 100 到上一次</p> <p><i>yield</i> 指令的地方，因此數字改變</p> <p>變為 100，再持續指行到下一個</p> <p><i>yield</i>，回傳 100。</p>
--	--

★ *pass* 指令為空語句，其不作任何事，通常用來佔位。可用在不作任何事的迴圈或函數。

範例：

佔位語句	
<pre>a=10 if a == 10: pass else: print(a)</pre>	<p>其無印出結果。</p> <p>用來佔位用，因當其 a 等於 10，不作任何事。</p>
函數使用	
<pre>def 計算(數字): pass</pre>	<p>可用在尚未撰寫的函數上，用來佔位。</p>

```
def 樹(p列表, l, a, f):
    if l > 3:
```

```

列表 = []

for p in p列表:

    p.前進(l)

    q = p.複製()

    p.左轉(a)

    q.右轉(a)

    列表.append(p)

    列表.append(q)

for x in 樹(列表, l*f, a, f):

    yield 無

```

函數 樹，傳入參數 p 列表, l, a, f，畫樹。p 列表 為龜類清單、l 為此

分支長度、a 為分支角度 及 f 為下一個分支長度縮短因子。

如果 l 大於 3（即長度還有 3 以上），則開始執行：

設置列表為空。

利用 for 迴圈抓取 p 列表內龜類。

將其 p.前進(l) 即畫出樹枝。

因要畫出分支，所以將 p.複製() 設為 q。

並一個向 左轉(a)、一個向 右轉(a)，代表兩分支夾角為 $2*a$ 度。

並將其兩個都加入到 列表。

for 迴圈結束。

再藉由 for 迴圈執行樹(列表, $l*f$, a, f) 產生下一個分支並抓取其回傳

的值為 x，藉由 $l*f$ 產生下一層分支長度。

產生 無 回傳。

for 迴圈結束。

此函數為遞迴函數，藉由反覆呼叫本身函數來創造分支，開始時傳入

p 列表 參數內只有一個龜類（樹幹），之後遞迴開始每次複製增加一

個龜類(分支固定分為兩個)，分支角度固定為 a (65)，而每一層分

支的長度固定乘上 f (0.6375)，持續減少，直到長度小於或等於 3，

則停止遞迴。所印出得龜群數量等於最後分支數量的兩倍，因為最後

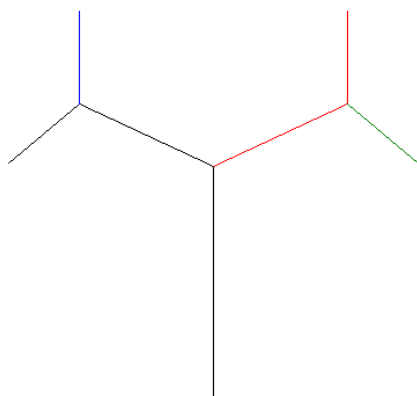
複製的分支長度會小於 3，所以不會長出。

範例：

不同顏色的樹枝代表不同的龜類所畫，此處總共有四個龜類所畫的線，

但龜群數量有八個，因其每個分支末端都會再複製一個龜類，只是未

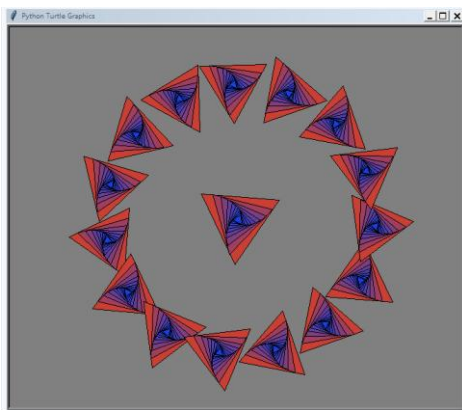
長出。



11 環繞舞 tc_round_dance.py

環繞舞（round dance），一種現代舞蹈中環繞著舞池以圓形逆時鐘旋轉的團體舞。舞者圍繞著中間連結成圓圈跳舞的民族舞蹈。在領舞的帶領下，隨著音樂與領舞一樣的步伐、節奏跳舞。

11-1 執行結果



11-2 程式碼

```
1 '''龜作圖範例集：
2
3     tc_demo_round_dance.py
4
5 (需要烏龜模組的版本 1.1，包含於 Python 3.1)
6
7
8 跳舞龜是由一系列的遞減尺寸三角型所組成的複合形狀。
```


9

10

11

12 龜沿著圓周軌道遊行，而本身反方向旋轉，只有正中央的例外。

13 在例子中吸引力的對稱提高是否會造成斷裂？

14

15

16

17 按下任意鍵停止動畫。

18

19 技術上：

20 演示複合形狀的使用、形狀的變換以及龜的複製。

21 動畫是根據 `update()` 【更新()】來控制。

22

23 '''

24

25 `from turtle_tc import *`

26

27 `def 停止():`

28 `global 正在跑`

29 正在跑 = 假

30

31 `def 主函數():`

32 `global 正在跑`

33 清除幕()

```

34     背景色 (灰色)
35     追蹤 (假)
36     形狀 (三角形)
37     f = 0.793402
38     φ = 9.064678
39     s = 5
40     c = 1
41     # 創建複合形狀
42     sh = 形狀類 ("compound")
43     for i in 範圍 (10):
44         形狀大小 (s)
45         p =取形狀多邊形 ()
46         s *= f
47         c *= f
48         傾斜 (-φ)
49         sh.加成員 (p, (c, 0.25, 1-c), 黑)
50     登記形狀 ("多重三角形", sh)
51     # 創建舞者
52     形狀大小 (1)
53     形狀 ("多重三角形")
54     提筆 ()
55     設位置 (0, -200)
56     舞者們 = []
57     for i in 範圍 (180):
58         前進 (7)

```

```
59     傾斜(-4)
60     左轉(2)
61     更新()
62     if i % 12 == 0:
63         舞者們.append(複製())
64     回家()
65     # 舞蹈
66     正在跑 = 真
67     在按著鍵時(停止)
68     聽()
69     cs = 1
70     while 正在跑:
71         ta = -4
72         for 舞者 in 舞者們:
73             舞者.前進(7)
74             舞者.左轉(2)
75             舞者.傾斜(ta)
76             ta = -4 if ta > 0 else 2
77         if cs < 180:
78             右轉(4)
79             形狀大小(cs)
80             cs *= 1.005
81         更新()
82     return "結束!"
83
```

```
84 if __name__ == '__main__':  
85     印(主函數())  
86     主迴圈()
```

11-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```
if __name__ == '__main__':  
    印(主函數())  
    主迴圈()
```

__name__ 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則__name__變數會等於 '__main__'，程序便會執行其下列動作。

主函數()被呼叫執行，其為本程式內部函數，將其回傳的資料以 訊息 接收。之後將訊息印出。

主迴圈()被呼叫執行，其為 turtle_tc 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 mainloop()，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數()_{，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。}

```
def 主函數():
```

```
    global 正在跑
```

```
    清除幕()
```

```
    背景色(灰色)
```

```
    追蹤(假)
```

```
    形狀(三角形)
```

設置全域變數 正在跑。

清除幕，將螢幕畫面清除。

設置螢幕背景顏色為灰色。

並設置追蹤為假，即代表畫面不更新。

之後將龜指標的形狀設為三角形。

```
def 主函數():
```

```
    f = 0.793402
```

```
    φ = 9.064678
```

```
    s = 5
```

```
    c = 1
```

設置變數。

f 為三角形逐漸縮小及變色的倍率。

φ 為三角形逐漸旋轉的角度。

s 為三角形初始的形狀大小。

c 為三角形的顏色變色數字。

```
def 主函數():  
  
    # 創建複合形狀  
  
    sh = 形狀類("compound")  
  
    for i in 範圍(10):  
  
        形狀大小(s)  
  
        p = 取形狀多邊形()  
  
        s *= f  
  
        c *= f  
  
        傾斜(-φ)  
  
        sh.加成員(p, (c, 0.25, 1-c), 黑)  
  
    登記形狀("多重三角形", sh)
```

創建形狀，其形狀為複合形狀。

因其形狀為三角形內包含眾多三角形。總共十個三角形。

所以設置 for 迴圈，重複執行 10 次，數字範圍為 0~9。

形狀大小為 s ，亦及 5。

將龜指標（三角形形狀）的頂點位置座標記錄為 p 。

將 s 乘上 f 作為下一個三角形形狀大小位置。

將 c 乘上 f 作為三角形的顏色變化參數。

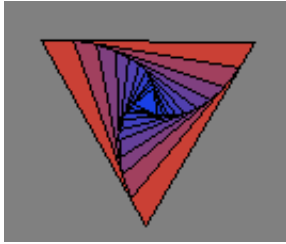
並將龜指標傾斜 $-\varphi$ 度，作為下一個三角形的傾斜角度。

（傾斜的方向根據模式所決定，此處為角度從東開始逆時針）

將之前記錄的形狀位置加入到 sh，並設置其顏色及其框線為黑色。

重複執行以上動作，直到迴圈結束，最後形狀如下，共有十個三角形。

並將此形狀 sh 登記，形狀名字為"多重三角形"。



```
def 主函數():  
  
    # 創建舞者  
  
    形狀大小(1)  
  
    形狀("多重三角形")  
  
    提筆()  
  
    設位置(0, -200)  
  
    舞者們 = []  
  
    for i in 範圍(180):  
  
        前進(7)  
  
        傾斜(-4)  
  
        左轉(2)  
  
        更新()  
  
        if i % 12 == 0:  
  
            舞者們.append(複製())  
  
    回家()
```

將龜指標形狀設為 1。並設置其形狀為"多重三角形"。

提筆，龜指標移動時不畫出軌跡。

設定龜指標其位置為 (0, -200)。

並設定列表 舞者們，用來記錄跳舞的舞者（多重三角形）。

創建 for 迴圈，重複執行 180 次，數字範圍為 0~179。

龜指標前進 7。

並將其旋轉-4 度。（運動方向不變）

角度左轉 2 度。（運動方向改變）

將畫面更新，圖像顯現。

如果其 i 可整除於 12，則將龜指標複製存入 舞者們。

重複執行以上動作，直到迴圈結束。總共會複製 16 個龜類，圍成一個圓圈。

將龜指標回到原位座標(0,0)。

```
def 主函數():
```

```
    # 舞蹈
```

```
    正在跑 = 真
```

```
    在按著鍵時(停止)
```

```
    聽()
```

```
    cs = 1
```

```
    while 正在跑:
```



```

    ta = -4

    for 舞者 in 舞者們:

        舞者.前進(7)

        舞者.左轉(2)

        舞者.傾斜(ta)

        ta = -4 if ta > 0 else 2

    if cs < 180:

        右轉(4)

        形狀大小(cs)

        cs *= 1.005

    更新()

    return "結束!"

```

此時開始準備讓各個舞者開始跳舞。

設置正在跑變數為真，代表控制程式的運行。

當其使用者壓下鍵盤，則呼叫函數，將正在跑變數改為假。

設定 聽，監聽鍵盤事件。

設置變數 cs 為 1，用來設置中心點的龜指標形狀。

創建 while 迴圈，當 正在跑 為真時，持續進行下列動作。

將 ta 變數設為-4，用來設定傾斜角度。

建立 for 迴圈重複執行，抓取 舞者們 內的舞者，讓其旋轉並前進圍成一個圈。

將舞者前進 7。

並將舞者左轉 2 度，來控制其可圍成一個圓。（運動方向改變）

將其形狀傾斜 t_a 度。（運動方向不變）

（此時 t_a 變數如果為正，則逆時鐘旋轉，否則相反。）

設置 t_a 變數，如果此時數字大於 0 則為 -4，否則為 2。

每個舞者都執行後，for 迴圈結束。

如果 c_s 變數小於 180，則執行下列動作。

將原點的龜指標右轉 4 度。

設定形狀大小為 c_s 。

並將 c_s 乘上 1.005，作為龜指標形狀放大的參數。

結束如果 c_s 變數小於 180 的動作。

最後將畫面更新，將剛所作的動作顯現。

迴圈動作結束並重複執行，此迴圈會讓原點舞者旋轉逐漸放大，周遭

舞者旋轉並依循一個圓形軌基逆時鐘前進。

回傳字串 "結束!"。

```
def 停止():
```

```
    global 正在跑
```

```
    正在跑 = 假
```

函數 停止，無傳入參數，用來將變數 正在跑 設為 假。

抓取全域變數 正在跑。

並將變數 正在跑 設為 假，代表其程式停止執行。

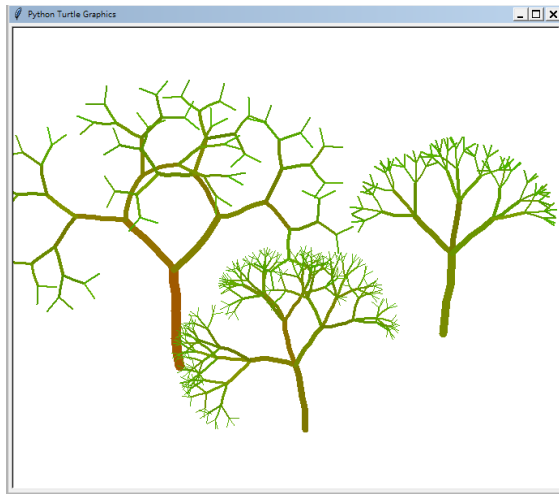
12 建立森林 tc_forest.py

利用龜作圖畫三顆樹，其程式跟 tc_tree.py 有點類似，但又增加了不少元素。每顆樹中分支數量為固定，但分支傾斜角度跟長度為隨機，顏色也有些微的變化。樹中顏色固定從底部往上為棕色（紅色與綠色，紅色比率較高）到綠色，而三顆樹的深度為固定，從左到右的樹固定深度為 7、6、5，。

此程式利用遞迴函數製作樹，藉由複製龜類和上一個數據（分支距離、分支角度）及隨機變數來製造分支，所複製得龜類位置、屬性會完全一樣，每遇到一個分支點便複製新的龜類，其複製數量與分支數量一致。程式樹作圖為廣度優先，為了實施廣度優先，因此利用生成器每顆樹輪流製造分支，等到三個生成器沒有可生成後，則代表結束。

廣度優先與深度優先相對應，顧名思義，廣度優先又名橫向優先為在作圖時從根部到頂部，一層一層往上作；而深度優先則相反，它為先將一分支從底部作到頂部在逐漸完成其它分支。

12-1 執行結果



12-2 程式碼

```
1  #!/usr/bin/env python3
2  '''turtlegraphics, 範例集:
3
4      tdemo_forest.py
5
6  顯示一座森林，其中有三顆廣度優先樹。
7
8  進一步說明可參考 tree.py。
9
10  這個範例為改寫 Erich Neuwirth 撰寫的一個 Logo 程序'breadth-first'。
11  詳情請看下列網址：
12  http://homepage.univie.ac.at/erich.neuwirth/
13  '''
14  from turtle_tc import *; from turtle_tc import 龜類, 色模式, 追蹤, 主迴圈
```

```

15 from random import randrange
16 from time import clock
17
18 def 隨機符號(n):
19     return randrange(-n,n+1)
20
21 def 隨機化( 分支列表, 角度距離, 邊距離 ):
22     return [ (角度+隨機符號(角度距離),
23              尺寸因子*1.01**隨機符號(邊距離))
24              for 角度, 尺寸因子 in 分支列表 ]
25
26 def 隨機前進( t, 距離, 部分, 角度距離 ):
27     for i in 範圍(部分):
28         t.左轉(隨機符號(角度距離))
29         t.前進( (1.0 * 距離)/部分 )
30
31 def 樹(t 列表, 尺寸, 水平, 寬度因子, 分支列表們, 角度距離=10, 邊距離=5):
32     # 列出使用的龜列表和分支清單
33     # 每個龜之一！
34     if 水平 > 0:
35         列表 = []
36         brs = []
37         for t, 分支列表 in list(zip(t 列表,分支列表們)):
38             t.筆粗( 尺寸 * 寬度因子 )
39             t.筆色( 255 - (180 - 11 * 水平 + 隨機符號(15)),

```

```

40             180 - 11 * 水平 + 隨機符號(15),
41             0 )
42         t.下筆()
43         隨機前進(t, 尺寸, 水平, 角度距離)
44         yield 1
45         for 角度, 尺寸因子 in 分支列表:
46             t.左轉(角度)
47             列表.append(t.複製())
48             brs.append(隨機化(分支列表, 角度距離, 邊距離))
49             t.右轉(角度)
50         for x in 樹(列表, 尺寸*尺寸因子, 水平-1, 寬度因子, brs,
51             角度距離, 邊距離):
52             yield 無
53
54
55 def 開始(t, x, y):
56     色模式(255)
57     t.重設()
58     t.速度(0)
59     t.藏龜()
60     t.左轉(90)
61     t.提筆()
62     t.設位置(x, y)
63     t.下筆()
64

```

```

65 def 做它 1 (水平, 筆):
66     筆.藏龜()
67     開始(筆, 20, -208)
68     t = 樹([筆], 80, 水平, 0.1, [[ (45,0.69), (0,0.65), (-45,0.71) ]])
69     return t
70
71 def 做它 2 (水平, 筆):
72     筆.藏龜()
73     開始(筆, -135, -130)
74     t = 樹([筆], 120, 水平, 0.1, [[ (45,0.69), (-45,0.71) ]])
75     return t
76
77 def 做它 3 (水平, 筆):
78     筆.藏龜()
79     開始(筆, 190, -90)
80     t = 樹([筆], 100, 水平, 0.1, [[ (45,0.7), (0,0.72), (-45,0.65) ]])
81     return t
82
83 # 這裡有 3 棵樹生成器:
84 def 主函數():
85     p = 龜類()
86     p.藏龜()
87     追蹤(75,0)
88     u = 做它 1 (6, 龜類(undobuffersize=1))
89     s = 做它 2 (7, 龜類(undobuffersize=1))

```

```

90     t = 做它 3 (5, 龜類 (undobuffersize=1))
91     a = clock ()
92     while 真:
93         做完了 = 0
94         for b in u,s,t:
95             try:
96                 b.__next__ ()
97             except:
98                 做完了 += 1
99         if 做完了 == 3:
100             break
101
102     追蹤 (1,10)
103     b = clock ()
104     return "執行時間: %.2f 秒。" % (b-a)
105
106 if __name__ == '__main__':
107     主函數 ()
108     主迴圈 ()

```

12-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。


```
if __name__ == '__main__':  
    主函數()  
    主迴圈()
```

__name__ 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則__name__ 變數會等於 '__main__'，程序便會執行其下列動作。

主函數() 被呼叫執行，其為本程式內部函數。

主迴圈() 被呼叫執行，其為 turtle_tc 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 mainloop()，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數()，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():  
    p = 龜類()  
    p.藏龜()  
    追蹤(75,0)  
    u = 做它 1(6, 龜類(undobuffersize=1))  
    s = 做它 2(7, 龜類(undobuffersize=1))  
    t = 做它 3(5, 龜類(undobuffersize=1))
```

創建一個龜類名為 p，並將其龜指標隱藏。

設置畫面每 75 步更新一次，延遲值 0。

呼叫函數 做它 1，製造第一顆樹，其水平深度為 6，並傳入龜類，設置其的回覆緩存區大小為 1。

呼叫函數 做它 2，製造第二顆樹，其水平深度為 7，並傳入龜類，設置其的回覆緩存區大小為 1。

呼叫函數 做它 3，製造第三顆樹，其水平深度為 5，並傳入龜類，設置其的回覆緩存區大小為 1。

```
def 主函數():  
  
    a = clock()  
  
    while 真:  
        做完了 = 0  
  
        for b in u,s,t:  
            try:  
                b.__next__()  
  
            except:  
                做完了 += 1  
  
        if 做完了 == 3:  
            break  
  
    追蹤(1,10)  
  
    b = clock()  
  
    return "執行時間: %.2f 秒。" % (b-a)
```

a = clock()，取得當前處理器的時間存入 a。

藉由 *while* 迴圈重複執行以下動作，因其條件式為真，所以會持續執行，直到執行跳出為止。因為要實施廣度優先樹，在這裡會抓取三個樹的生成，利用生成器每個樹輪流製造分支。

設置 做完了 參數為 0。

藉由 for 迴圈抓取回傳的樹生成器 u、s、t，執行下列動作，。

嘗試執行 `b.__next__()`，產生樹的生成。

如果出錯了則代表生成器結束了，做完了加 1。

如果做完了等於 3，則代表三棵樹都生成完了。

執行 *break*，跳出 *while* 迴圈。

設置 追蹤 (1,10)，代表畫面每 1 步更新一次，延遲值為 10。

`b = clock()`，取得當前處理器的時間存入 a。

回傳字串，利用 `b-a` 得到產生樹的執行時間。

```
def 做它 1 (水平, 筆):  
    筆.藏龜()  
    開始(筆, 20, -208)  
    t = 樹([筆], 80, 水平, 0.1, [[ (45,0.69), (0,0.65), (-45,0.71) ]])  
    return t
```

函數 做它 1，傳入參數水平、筆，製造中間的樹。

將傳進來的筆（龜類）隱藏龜指標。

呼叫函數 開始，傳入筆跟畫樹開始座標，設定筆的初始設定跟座標 (20, -208)。

呼叫函數 樹，傳入筆、80（樹的尺寸因子）、水平（樹的深度）、0.1（寬度因子）跟[[(45,0.69), (0,0.65), (-45,0.71)]]（樹的分支角度跟尺寸因子）。所回傳的生成器命為 t。

回傳生成器 t。

```
def 做它 2 (水平, 筆):  
    筆.藏龜()  
    開始(筆, -135, -130)  
    t = 樹([筆], 120, 水平, 0.1, [[ (45,0.69), (-45,0.71) ]])  
    return t
```

函數 做它 2，傳入參數水平、筆，製造左邊的樹。

將傳進來的筆（龜類）隱藏龜指標。

呼叫函數 開始，傳入筆跟畫樹開始座標，設定筆的初始設定跟座標 (-135, -130)。

呼叫函數 樹，傳入筆、120（樹的尺寸因子）、水平（樹的深度）、0.1（寬度因子）跟[[(45,0.69), (-45,0.71)]]（樹的分支角度跟尺寸因子）。所回傳的生成器命為 t。

回傳生成器 t。

```
def 做它 3 (水平, 筆):
```

```
筆.藏龜()
```

```
開始(筆, 190, -90)
```

```
t = 樹([筆], 100, 水平, 0.1, [[(45,0.7), (0,0.72), (-45,0.65)]])
```

```
return t
```

函數 做它 3，傳入參數水平、筆，製造右邊的樹。

將傳進來的筆（龜類）隱藏龜指標。

呼叫函數 開始，傳入筆跟畫樹開始座標，設定筆的初始設定跟座標 (190, -90)。

呼叫函數 樹，傳入筆、100（樹的尺寸因子）、水平（樹的深度）、0.1（寬度因子）跟 [[(45,0.7), (0,0.72), (-45,0.65)]]（樹的分支角度跟尺寸因子）。所回傳的生成器命為 t。

回傳生成器 t。

```
def 開始(t, x, y):
```

```
    色模式(255)
```

```
    t.重設()
```

```
    t.速度(0)
```

```
    t.藏龜()
```

```
    t.左轉(90)
```

```
    t.提筆()
```

```
    t.設位置(x, y)
```

```
    t.下筆()
```

函數 開始，傳入參數 `t`（龜類）、`x`（座標 `X`）、`y`（座標 `y`），設置筆的初始設定跟座標。

設置色模式為 255，顏色數字範圍 0~255。

重置 `t` 跟刪除 `t` 所畫出的軌跡。

設置 `t` 速度為 0，最快速，移動時直接跳至最後結果。

隱藏 `t` 龜指標。將 `t` 左轉 90 度（改變移動方向，此時龜指標方向向上）。

將 `t.penup()`，移動時不畫出軌跡。

設定 `t` 的座標為 `(x,y)`。

將 `t.pendown()`，移動時畫出軌跡。

```
def 樹(t 列表, 尺寸, 水平, 寬度因子, 分支列表們, 角度距離=10, 邊距離=5):
```

```
    # 列出使用的龜列表和分支清單
```

```
    # 每個龜之一！
```

```
    if 水平 > 0:
```

```
        列表 = []
```

```
        brs = []
```

```
        for t, 分支列表 in list(zip(t 列表, 分支列表們)):
```

```
            t.筆粗( 尺寸 * 寬度因子 )
```

```
            t.筆色( 255 - (180 - 11 * 水平 + 隨機符號(15)),
```

```
                    180 - 11 * 水平 + 隨機符號(15),
```

```
                    0 )
```

```
            t.下筆()
```

```
            隨機前進(t, 尺寸, 水平, 角度距離 )
```

```
yield 1
```

```
for 角度, 尺寸因子 in 分支列表:
```

```
    t.左轉(角度)
```

```
    列表.append(t.複製())
```

```
    brs.append(隨機化(分支列表, 角度距離, 邊距離))
```

```
    t.右轉(角度)
```

```
for x in 樹(列表, 尺寸*尺寸因子, 水平-1, 寬度因子, brs,
```

```
    角度距離, 邊距離):
```

```
yield 無
```

函數 樹，傳入參數 t 列表（存著龜類的表單）、尺寸（樹的尺寸因子）、水平（樹的深度）、寬度因子（樹的寬度因子）、分支列表們（樹的分支角度跟尺寸因子）、角度距離跟邊距離，回傳隨機亂數 -n~n。

如果水平大於零，代表還有樹的分支還沒作，則執行下列動作。

宣告列表為空的清單（list，用來存新的龜類）。

宣告 brs 為空的清單（list，用來存新的分支座標）。

藉用 for 迴圈重複執行，此處藉由 `list(zip(t 列表, 分支列表們))` 來作成新的清單，其抓取龜類跟對應的分支列表。（zip 用法可參閱下方）

設置 t（龜類）筆粗為 尺寸 * 寬度因子。

設置 t 顏色，主要為紅色與綠色，當水平度越高代表越下面，則紅色數字越高；當水平度越低代表越上面，綠色數字越高。（所以顏色會從咖啡色到綠色，將顏色扣掉隨機變數，讓顏色變化多樣化）

將 `t`.`下筆()`，移動時會畫出軌跡。

呼叫函數，讓樹畫出樹支。

產生 1 回傳。

在裡面再藉由 `for` 迴圈抓取分支列表拆成 角度 跟 尺寸因子 重複執行。

將 `t` 左轉 角度 度。

複製 `t` 並存入 列表。代表複製一個新的龜類來畫分支。

把函數 `隨機化` 所回傳的表單加入到 `brs`。(新龜類的分支列表)

將 `t` 右轉 角度 度。回到原本方向。

新的龜類及分支列表都作完後，第二個 `for` 迴圈結束。

當分支都畫完且也製造新的分支龜類及列表後，`for` 迴圈結束。

將新的龜類列表及分支列表等丟入函數 `樹`，並水平-1。以及為了讓函數遞迴，藉由 `for` 迴圈進行遞迴迴圈，並生產 無。

★ `zip` 可將傳入的參數們按照相對應的位置打包成一個元組。

範例：

```
t 列表 = [筆]
```

```
分支列表 = [[ (45,0.7), (0,0.72), (-45,0.65) ]]
```

```
a = zip(t 列表,分支列表們)
```

```
print(a)
```

```
>>> [(筆, [(45, 0.7), (0, 0.72), (-45, 0.65)])]
```

```
def 隨機符號(n):
```



```
return randrange(-n,n+1)
```

函數 隨機符號，傳入參數 n ，回傳隨機亂數 $-n \sim n$ 。

`randrange(-n,n+1)`，根據傳入參數決定亂數範圍，此處代表隨機產生亂數，其範圍在 $-n$ 到 n 中（ $n+1$ 不包含在裡面）。

```
def 隨機前進( t, 距離, 部分, 角度距離 ):
```

```
    for i in 範圍(部分):
```

```
        t.左轉(隨機符號(角度距離))
```

```
        t.前進( (1.0 * 距離)/部分 )
```

函數 隨機前進，傳入參數 t （龜類）、距離、部分（水平）跟角度距離，隨機決定左轉角度並前進。

藉由 `for` 迴圈重複執行，重複 部分 次，執行下列動作。

根據隨機符號函數回傳的數字，決定左轉的角度。

t 前進距離 $(1.0 * \text{距離}) / \text{部分}$ 。

`for` 迴圈結束。

```
def 隨機化( 分支列表, 角度距離, 邊距離 ):
```

```
    return [ (角度+隨機符號(角度距離),
```

```
              尺寸因子*1.01**隨機符號(邊距離))
```

```
            for 角度, 尺寸因子 in 分支列表 ]
```

函數 隨機化，傳入參數分支列表、角度距離、邊距離，來創造新的分支角度跟距離。

藉由 for 迴圈抓取分支列表內的角度跟尺寸因子，藉由這些數值來創造新分支的分支列表。

將角度加上隨機變數。

（藉由函數 `隨機符號` 所產生，範圍在-角度距離~+角度距離）

將尺寸因子乘上 1.01 的隨機平方。

（藉由函數 `隨機符號` 所產生，範圍在-邊距離~+邊距離）

將兩個參數為元組並加入到清單裡。

回傳表單。

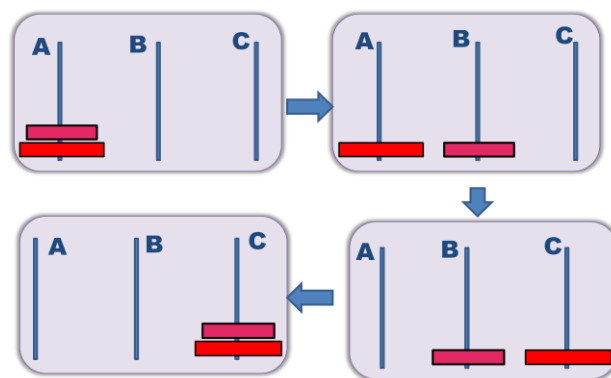
13 移動河內塔 `tc_minimal_hanoi.py`

河內塔問題（Tower of Hanoi）是由法國數學家 愛德華·盧卡斯（François Édouard Anatole Lucas）所提出，傳說在古印度伯那爾斯（Benares），有一座被稱為世界之中心的神廟裡，有三根柱子，一根柱子上串有 64 塊由上到下逐次變大的金盤。寺廟的僧侶依照古老的預言，必須按照規則，將 64 塊金盤移至三根柱子其中一根，其中規則為一次只能移動一個盤子及大盤子不能移到小盤子上方，而當有一天僧侶將 64 塊金盤都移到另一根柱子上時，神廟將化為灰燼，而世界末日也將會降臨。不過此傳說眾說紛紜，也有的說法是 盧卡斯 自己

編造了這個傳說，也有說其故事背景是在越南的河內等等。但先不論此故事真假與否，在其規則的建立上就是個有趣的數學問題，其解法也相當有趣。現在有 3 根柱子為 A、B、C，而 A 柱有 N 個盤子，要將其移到 C 柱，其解法便是將 A 柱最上方的 N-1 個盤子先移到 B 柱，再將剩下的最大盤子移到 C 柱，最後便把剩下的 N-1 的盤子移到 C 柱，而其 N-1 的移動方法則是根據上述移法來移動，若其 N-1 為 1 時，則可直接移動 B 柱。

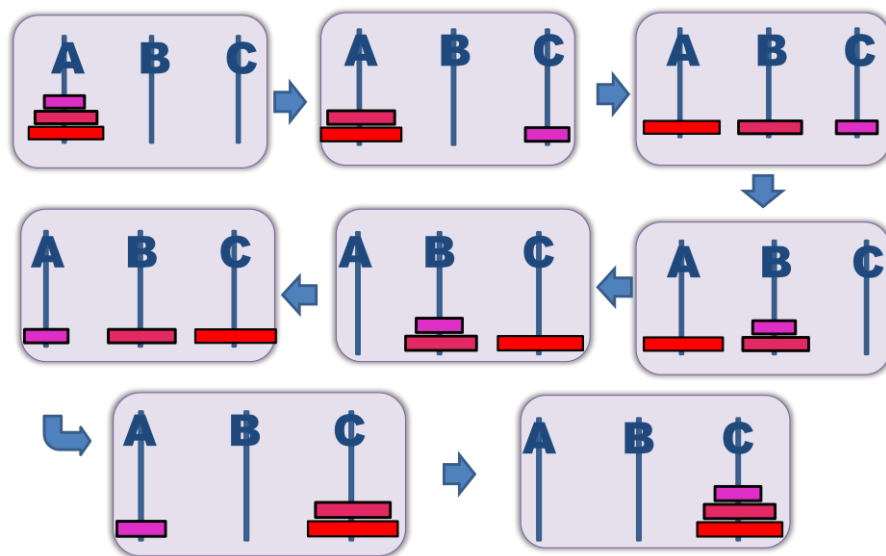
範例：

a. 假設 N 為 2



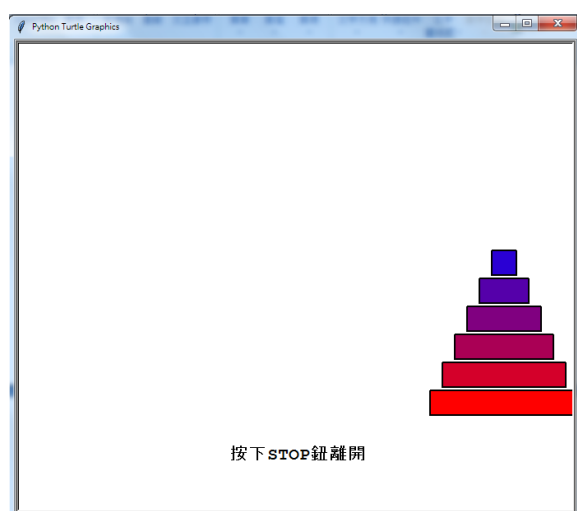
總共經過了 $2^2-1=3$ 步步驟

b. 假設 N 為 3



總共經過了 $2^3-1=7$ 步步驟

13-1 執行結果



13-2 程式碼

```

1  '''龜作圖範例集：
2
3      tdemo_minimal_hanoi.py
4

```

```

5 一個迷你的河內塔動畫：
6 將六個圓盤從左邊桿轉移到右邊桿。
7
8
9 一個簡單且精美的河內塔物件實例，
10 由內置類型表中衍生實現。
11
12
13 盤類為龜的形狀 "方形" 作出，
14 再用 形狀大小() 延伸至矩形。
15 -----
16         要退出，請按 STOP 按鈕
17 -----
18 '''
19 from turtle_tc import *
20
21 class 盤類(龜類)：
22     def __init__(我, n)：
23         龜類.__init__(我, shape=方形, visible=假)
24         我.提筆()
25         我.形狀大小(1.5, n*1.5, 2) # 方形 -> 矩形
26         我.填色(n/6., 0, 1-n/6.)
27         我.顯龜()
28
29 class 塔類(list)：

```

```

30     "河內塔，繼承內置類型 list"
31     def __init__(我, x):
32         "建造一個空的塔。x 為塔的座標 x"
33         我.x = x
34     def 推入(我, 盤):
35         盤.設 x 座標(我.x)
36         盤.設 y 座標(-150+34*len(我))
37         我.append(盤)
38     def 拋出(我):
39         盤 = list.pop(我)
40         盤.設 y 座標(150)
41         return 盤
42
43 def 河內(n, 從_, 伴隨_, 去_):
44     if n > 0:
45         河內(n-1, 從_, 去_, 伴隨_)
46         去_.推入(從_.拋出())
47         河內(n-1, 伴隨_, 從_, 去_)
48
49 def 玩():
50     在按鍵時(無, 空白鍵)
51     清除()
52     try:
53         河內(6, 塔一, 塔二, 塔三)
54         寫("按下 STOP 鈕離開",
55             align="center", font=("Courier", 16, "bold"))

```

```

56     except Terminator:
57         pass # turtledemo 使用者按 STOP
58
59 def 主函數():
60     global 塔一, 塔二, 塔三
61     藏龜(); 提筆(); 前往(0, -225) # 作家龜
62     塔一 = 塔類(-250)
63     塔二 = 塔類(0)
64     塔三 = 塔類(250)
65     # 建造塔的六個盤子
66     for i in 範圍(6, 0, -1):
67         塔一.推入(盤類(i))
68     # 準備使用者介面;-)
69     寫("請按下 空白鍵 來開始遊戲",
70         align="center", font=("Courier", 16, "bold"))
71     在按鍵時(玩, 空白鍵)
72     聽()
73     return "事件迴圈"
74
75 if __name__=="__main__":
76     訊息 = 主函數()
77     印(訊息)
78     主迴圈()

```

13-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```
if __name__ == "__main__":  
    訊息 = 主函數()  
    印(訊息)  
    主迴圈()
```

__name__ 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則 __name__ 變數會等於 '__main__'，程序便會執行其下列動作。

主函數() 被呼叫執行，其為本程式內部函數，將其回傳的資料以 訊息 接收。之後將訊息印出。

主迴圈() 被呼叫執行，其為 turtle_tc 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 mainloop()，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數()，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():
```



```

global 塔一, 塔二, 塔三

藏龜(); 提筆(); 前往(0, -225)    # 作家龜

塔一 = 塔類(-250)

塔二 = 塔類(0)

塔三 = 塔類(250)

# 建造塔的六個盤子

for i in 範圍(6,0,-1):

    塔一.推入(盤類(i))

# 準備使用者介面;-)

寫("請按下 空白鍵 來開始遊戲",

    align="center", font=("Courier", 16, "bold"))

在按鍵時(玩, 空白鍵)

聽()

return "事件迴圈"

```

宣告三個塔為全域變數。

在 python 中一個指令的結尾通常是以換行號作辨別，但如果要一行輸入多個指令，就可以使用分號作區隔。由於龜模組中本身便會內建一個龜物件，因此需 藏龜()，不需顯現。其中因為盤子的移動不需畫出軌跡，所以 提筆()。將龜指標 前往(0, -225)，準備寫字。宣告三個塔為 塔類() 類別，塔座標分別在 X 作標-250、0、250。

之後利用 for 迴圈建造 6 個 盤類()，都將其推疊在第一個塔上 塔一.推入(盤類(i))，由於要先從大到小推疊，因此 for 迴圈的 i 從 6~1。

並 寫("請按下 空白鍵 來開始遊戲", align="center", font=("Courier", 16, "bold"))，等待使用者按下空白鍵。利用 聽() 監聽使用者鼠標和鍵盤事件，如使用者按下空白鍵，則呼叫 玩() 函數，在按鍵時(玩, 空白鍵)。

最後回傳 "事件迴圈" 字串。

```
def 玩():  
    在按鍵時(無, 空白鍵)  
  
    清除()  
  
    try:  
        河內(6, 塔一, 塔二, 塔三)  
  
        寫("按下 STOP 鈕離開",  
            align="center", font=("Courier", 16, "bold"))  
  
    except Terminator:  
        pass # turtledemo 使用者按 STOP
```

無傳入參數，由於避免使用者繼續按空白鍵重複呼叫 玩()，因此設定在按鍵時(無, 空白鍵) 按空白鍵不會呼叫函數。

清除() 將原本下方內建龜類所寫的信息清除。

嘗試開始移動盤類，如有錯誤則跳到 *except Terminator*。

呼叫 河內(6, 塔一, 塔二, 塔三)，將六個盤從塔一移到塔三，中間過渡為塔二。

當盤子都移完時，寫("按下 STOP 鈕離開", align="center", font=("Courier", 16, "bold"))。

```

class 盤類(龜類):

    def __init__(我, n):

        龜類.__init__(我, shape=方形, visible=假)

        我.提筆()

        我.形狀大小(1.5, n*1.5, 2) # 方形 -> 矩形

        我.填色(n/6., 0, 1-n/6.)

        我.顯龜()

```

建造 `class 盤類(龜類)`，繼承龜類。

其中 `__init__` 函數就像是此類別的建構子(constructor)，每當呼叫此類別創造物件(Object)時，程序被會自動呼叫此函數執行。此函數通常為類別的基本或初始設定。建造類別的第一個參數，將會保留為物件自己，也就是 `我 (self)`。而當函數中呼叫此類別的屬性(變數)時，都需加上 `我 (self)`。為建造盤類物件時，所傳入的參數。

`龜類.__init__(我, shape=方形, visible=假)`，由於繼承 龜類，因此本身具有龜類特性，設定其形狀為方形。

因盤類的移動軌跡不需被畫出，所以 `我.提筆()`。

設定 `我.形狀大小(1.5, n*1.5, 2)`，將形狀由方型延伸為矩形，第一個數字為 龜指標 方向的垂直方向延展，因此保持 1.5。第二個數字為 龜指標 方向的延展，根據所傳入的 `n` 決定。最後設定其輪廓的線寬為 2。

我.填色(n/6., 0, 1-n/6.) 設定其形狀的內部填色，將顏色設定藉由傳入 n 的數字大到小(6~1)，由紅到藍。（範例：如傳入的 n 為 6，則顏色為(1,0,0)即紅色。）

由於盤類要顯現，所以 我.顯龜()。

```
class 塔類(list):
```

建造 class 塔類(list)，繼承內置類型 list(串列)。

```
class 塔類(list):

    def __init__(我, x):

        "建造一個空的塔. x 為塔的座標 x"

        我.x = x
```

設定所傳入的參數為塔類的 x，即塔類的 X 座標。

```
class 塔類(list):

    def 推入(我, 盤):

        盤.設 x 座標(我.x)

        盤.設 y 座標(-150+34*len(我))

        我.append(盤)
```

當有 盤 要被推入塔類，即把 盤 的 x 座標設為塔的 x 座標。

並根據塔目前有幾個盤子乘上 34 再加上基底 y 座標-150。

最後將此 盤 加入到塔 list 中。

```
class 塔類(list):

    def 拋出(我):

        盤 = list.pop(我)
```

```
    盤.設 y 座標(150)  
  
    return 盤
```

當有 盤 要被拋出塔類，即用 list 的 pop 把 盤 丟出。

(pop 為後入先出，即最後加入的會最早出去。因此會拋出最上面的盤。)

再把 盤 的 y 座標設為 150。即把 盤 往上移。

最後將拋出的 盤 回傳。

```
def 河內(n, 從_, 伴隨_, 去_) :  
    if n > 0 :  
        河內(n-1, 從_, 去_, 伴隨_)  
        去_.推入(從_.拋出())  
        河內(n-1, 伴隨_, 從_, 去_)
```

函數 `def 河內(n, 從_, 伴隨_, 去_)`，傳入參數 `n` 為要移動的盤子總數，`從_` 為要移動的盤子所在的初始柱子，`伴隨_` 為移動盤子的中間柱子，`去_` 為目標柱子。此函數為遞迴函數，即函數呼叫函數本身。

當移動的盤子等於一個時，即 `n` 等於 1，則 `n-1` 等於 0，呼叫本身函數時，不執行任何動作。等於直接移動盤子。

而當移動的盤子大於一個時，則需要呼叫本身函數，遞迴移動。

`if n > 0`，當 `n` 大於 0，即有盤子要移才執行下列指令。

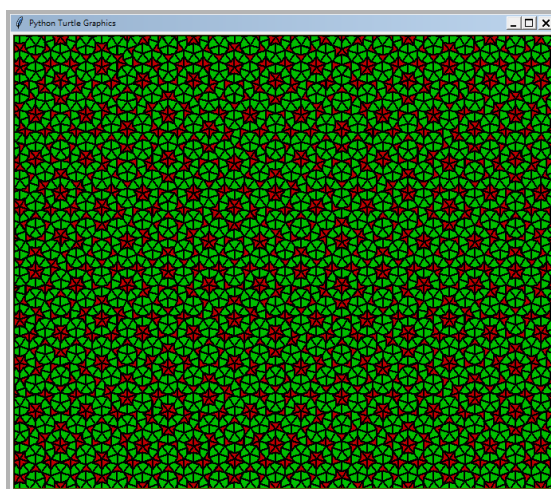
按照之前移動河內塔的解法，先將 `n-1` 的的盤類移動到中間柱子 `伴隨_`，`河內(n-1, 從_, 去_, 伴隨_)`。

之後把最後的第 n 個盤類從初始柱子移到目標柱子，即將它從 `從_` 拋出() 然後接收到盤類後，再將它 `去_` 推入。(`去_` 推入(`從_` 拋出()))
最後再把移動到中間柱子的 $n-1$ 個盤類移到目標柱子，河內($n-1$, 伴隨_
`從_`, `去_`)，即全部移完。

14 潘洛斯鋪磚 tc_penrose.py

1974 年數學家羅傑·潘洛斯發明了潘洛斯鋪磚(Penrose Tiling)，發現它能以兩種磚片（風箏形狀、飛鏢形狀）非週期性的鋪滿整個平面。在程式裡先畫出風箏及飛鏢的形狀及設定其大小，之後計算出風箏及飛鏢的位置並記錄，最後根據存取的數據將指標蓋章在畫布上。

14-1 執行結果



14-2 程式碼

```
1  #!/usr/bin/env python3
2  '''xturtle ，範例集：
3
4      xtx_kites_and_darts.py
5
6  建造兩個非週期性的潘洛斯鋪磚 (penrose-tilings) ，
7  由風箏跟飛鏢組成，再經過六個步驟的膨脹函數。
8
9
10 出發點為包含五個風箏的圖案 "日"
11 和 包含五個飛鏢的圖案 "星"
12
13
14 欲了解更多信息，請參閱：
15  http://en.wikipedia.org/wiki/Penrose\_tiling
16  -----
17  '''
18  from turtle_tc import *
19  from math import cos, pi
20  from time import clock, sleep
21
22  f = (5**0.5-1)/2.0  # ( SQRT ( 5 ) -1 ) / 2 - 黃金比例
23  d = 2 * cos(3*pi/10)
24
```

```

25  def 風箏(l):
26      fl = f * l
27      左轉(36)
28      前進(l)
29      右轉(108)
30      前進(fl)
31      右轉(36)
32      前進(fl)
33      右轉(108)
34      前進(l)
35      右轉(144)
36
37  def 飛鏢(l):
38      fl = f * l
39      左轉(36)
40      前進(l)
41      右轉(144)
42      前進(fl)
43      左轉(36)
44      前進(fl)
45      右轉(144)
46      前進(l)
47      右轉(144)
48
49  def 膨脹風箏(l, n):

```



```

50     if n == 0:
51         px, py = 位置()
52         h, x, y = int(頭向()), round(px,3), round(py,3)
53         瓦片字典[(h,x,y)] = 真
54         return
55     fl = f * l
56     左轉(36)
57     膨脹飛鏢(fl, n-1)
58     前進(l)
59     右轉(144)
60     膨脹風箏(fl, n-1)
61     左轉(18)
62     前進(l*d)
63     右轉(162)
64     膨脹風箏(fl, n-1)
65     左轉(36)
66     前進(l)
67     右轉(180)
68     膨脹飛鏢(fl, n-1)
69     左轉(36)
70
71 def 膨脹飛鏢(l, n):
72     if n == 0:
73         px, py = 位置()
74         h, x, y = int(頭向()), round(px,3), round(py,3)

```

```

75         瓦片字典[(h,x,y)] = 假
76         return
77     fl = f * l
78     膨脹風箏(fl, n-1)
79     左轉(36)
80     前進(l)
81     右轉(180)
82     膨脹飛鏢(fl, n-1)
83     左轉(54)
84     前進(l*d)
85     右轉(126)
86     膨脹飛鏢(fl, n-1)
87     前進(l)
88     右轉(144)
89
90 def 畫(l, n, th=2):
91     清除()
92     l = l * f**n
93     形狀大小(l/100.0, l/100.0, th)
94     for k in 瓦片字典:
95         h, x, y = k
96         設位置(x, y)
97         設頭向(h)
98     if 瓦片字典[k]:
99         形狀("風箏")

```

```

100         顏色(黑, (0, 0.75, 0))
101     else:
102         形狀("飛鏢")
103         顏色(黑, (0.75, 0, 0))
104         蓋章()
105
106 def 日(l, n):
107     for i in 範圍(5):
108         膨脹風箏(l, n)
109         左轉(72)
110
111 def 星(l, n):
112     for i in 範圍(5):
113         膨脹飛鏢(l, n)
114         左轉(72)
115
116 def 製造形狀們():
117     追蹤(0)
118     開始多邊形()
119     風箏(100)
120     結束多邊形()
121     登記形狀("風箏", 取多邊形())
122     開始多邊形()
123     飛鏢(100)
124     結束多邊形()

```

```

125     登記形狀("飛鏢", 取多邊形())
126     追蹤(1)
127
128 def 開始():
129     重設()
130     藏龜()
131     提筆()
132     製造形狀們()
133     重設大小模式("user")
134
135 def 測試(l=200, n=4, 函數=日, 開始位置=(0,0), th=2):
136     global 瓦片字典
137     前往(開始位置)
138     設頭向(0)
139     瓦片字典 = {}
140     時間一 = clock()
141     追蹤(0)
142     函數(l, n)
143     時間二 = clock()
144     畫(l, n, th)
145     追蹤(1)
146     時間三 = clock()
147     印("計算時間:  %7.4f s" % (時間二 - 時間一))
148     印("繪圖時間:  %7.4f s" % (時間三 - 時間二))
149     印("總和時間:  %7.4f s" % (時間三 - 時間一))

```

```

150     nk = len([x for x in 瓦片字典 if 瓦片字典[x]])
151     nd = len([x for x in 瓦片字典 if not 瓦片字典[x]])
152     印("%d 個風箏 和 %d 個飛鏢 = %d 片" % (nk, nd, nk+nd))
153
154 def 展示(函數=日):
155     開始()
156     for i in 範圍(8):
157         時間一 = clock()
158         測試(300, i, 函數)
159         時間二 = clock()
160         t = 時間二 - 時間一
161         if t < 2:
162             sleep(2 - t)
163
164 def 主函數():
165     # 標題 ("潘洛斯瓦片風箏和飛鏢。 " )
166     模式(角度從北開始順時針)
167     背景色(0.3, 0.3, 0)
168     展示(日)
169     sleep(2)
170     展示(星)
171     筆色(黑)
172     前往(0, -200)
173     筆色(0.7, 0.7, 1)
174     寫("請等待...",
175         align="center", font=('Arial Black', 36, 'bold'))

```

```

176     測試(600, 8, 開始位置=(70, 117))
177     return "結束"
178
179 if __name__ == "__main__":
180     訊息 = 主函數()
181     主迴圈()

```

14-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```

f = (5**0.5-1)/2.0    # ( SQRT ( 5 ) -1 ) / 2 - 黃金比例
d = 2 * cos(3*pi/10)

```

設定常數。

```

if __name__ == "__main__":
    訊息 = 主函數()
    主迴圈()

```

__name__ 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則__name__ 變數會等於'__maim__'，程序便會執行其下列動作。

主函數()被呼叫執行，其為本程式內部函數，將其回傳的資料以 訊息 接收。

主迴圈()被呼叫執行，其為 turtle_tc 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 mainloop()，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數()，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():
```

```
    # 標題 ("潘洛斯瓦片風箏和飛鏢。 ")
```

```
    模式(角度從北開始順時針)
```

```
    背景色(0.3, 0.3, 0)
```

```
    展示(日)
```

```
    sleep(2)
```

```
    展示(星)
```

```
    筆色(黑)
```

```
    前往(0,-200)
```

```
    筆色(0.7,0.7,1)
```

```
    寫("請等待...",
```

```
        align="center", font=('Arial Black', 36, 'bold'))
```

```
    測試(600, 8, 開始位置=(70, 117))
```

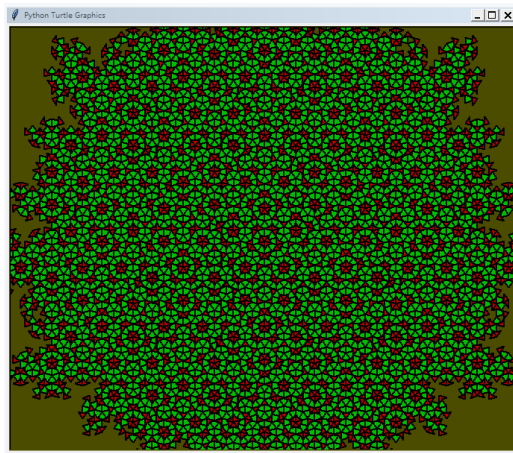
```
    return "結束"
```

一開始設定其模式，讓其角度的設定是從北開始順時針計算。

設定其 背景色 `(0.3, 0.3, 0)` 。

之後呼叫 `展示(日)`，即先畫出出發點為五個風箏往外擴張的圖案。

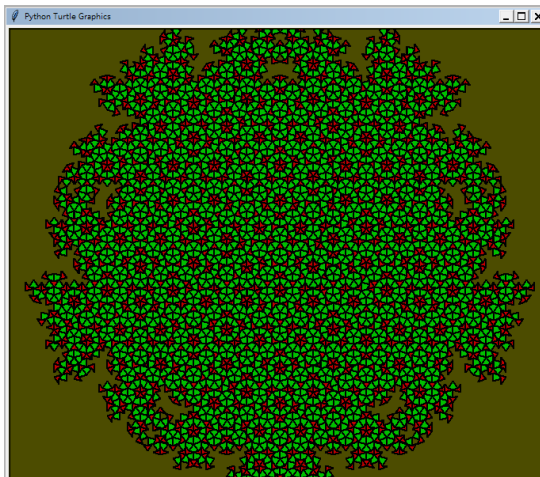
`sleep(2)`，休息 2 秒。



執行完後，圖為：

此時停止 2 秒，`sleep(2)`。

再呼叫 `展示(星)`，畫出出發點為五個飛鏢往外擴張的圖案。



執行完後，圖為：

前往 `(0,-200)`，將龜指標移到位置 `(0,-200)`。

並將筆色改為 `(0.7,0.7,1)`，寫下 "請等待..."。

最後再呼叫 `測試(600, 8, 開始位置=(70, 117))`，即為結果圖。

```
def 展示(函數=日):
```



```

開始()

for i in 範圍(8):

    時間一 = clock()

    測試(300, i, 函數)

    時間二 = clock()

    t = 時間二 - 時間一

    if t < 2:

        sleep(2 - t)

```

函數 展示(函數=日)，傳入參數 函數，如無傳入參數則默認為 日。畫出風箏及飛鏢擴張圖。

呼叫 開始()，清空及初始設定。

之後使用 for 迴圈，執行 8 次 (0~7)。

時間一 = clock()，抓取程式執行的實際時間。

呼叫 測試(300, i, 函數)，開始畫圖。

時間二 = clock()，抓取第一次呼叫後的到現在的執行時間。

將兩個時間相減，得到呼叫 測試 後的執行時間。如果沒有超過兩秒，則休息到兩秒再繼續執行 sleep(2 - t)。

```

def 開始():

    重設()

    藏龜()

    提筆()

    製造形狀們()

```

重設大小模式 ("user")

函數 開始 ()，無傳入參數。此函數目的為設定畫圖前的初始狀態。

重設 ()，將螢幕清空，所有 龜類 回到初始位置及設定。

藏龜 ()，將龜指標隱藏，不要顯現。並且 提筆 ()，使其在移動時不會畫出軌跡。

之後呼叫 製造形狀們 ()，來製造將使用到的形狀。

重設大小模式 ("user")，並且將龜指標形狀的尺寸設定改為 "user" 定義。

def 製造形狀們 () :

 追蹤 (0)

 開始多邊形 ()

 風箏 (100)

 結束多邊形 ()

 登記形狀 ("風箏", 取多邊形 ())

 開始多邊形 ()

 飛鏢 (100)

 結束多邊形 ()

 登記形狀 ("飛鏢", 取多邊形 ())

 追蹤 (1)

函數 製造形狀們 ()，無傳入參數。此函數目的為製造此程式會使用到的兩個龜形狀。

設定 追蹤 (0)，即設定其不會更新螢幕畫面。

並 開始多邊形 `()`，開始記錄多邊形頂點。呼叫 風箏 `(100)`，製造風箏形狀的多邊行。結束多邊形 `()`，結束記錄多邊形頂點。

登記形狀 `("風箏", 取多邊形())`，之後將此形狀登記，取名為 `"風箏"`。

製造完風箏形狀後，便開始製造飛鏢形狀。

一樣呼叫 開始多邊形 `()`，然後呼叫 飛鏢 `(100)`，製造飛鏢形狀，最後 結束多邊形 `()`。

登記形狀 `("飛鏢", 取多邊形())`，將此形狀登記，取名為 `"飛鏢"`。

最後將螢幕更新改回 追蹤 `(1)`，每執行一次更新一次。



```
def 風箏(l):
```

```
    fl = f * l
```

```
    左轉(36)
```

```
    前進(l)
```

```
    右轉(108)
```

```
    前進(fl)
```

```
    右轉(36)
```

```
    前進(fl)
```

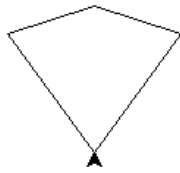
```
    右轉(108)
```

```
    前進(l)
```

```
    右轉(144)
```

函數 風箏 `(l)`，傳入參數 `l`，目的為畫出為大小係數為 `l` 的風箏。

畫出圖為：(1 為 100)



def 飛鏢(1) :

 f1 = f * 1

 左轉(36)

 前進(1)

 右轉(144)

 前進(f1)

 左轉(36)

 前進(f1)

 右轉(144)

 前進(1)

 右轉(144)

函數 飛鏢(1)，傳入參數 1，目的為畫出為大小係數為 1 的飛鏢。

畫出圖為：(1 為 100)



def 測試(l=200, n=4, 函數=日, 開始位置=(0,0), th=2) :

global 瓦片字典

 前往(開始位置)

 設頭向(0)

 瓦片字典 = {}

```

時間一 = clock()

追蹤(0)

函數(l, n)

時間二 = clock()

畫(l, n, th)

追蹤(1)

時間三 = clock()

印("計算時間:  %7.4f s" % (時間二 - 時間一))

印("繪圖時間:  %7.4f s" % (時間三 - 時間二))

印("總和時間:  %7.4f s" % (時間三 - 時間一))

nk = len([x for x in 瓦片字典 if 瓦片字典[x]])

nd = len([x for x in 瓦片字典 if not 瓦片字典[x]])

印("%d 個風箏 和 %d 個飛鏢 = %d 片" % (nk, nd, nk+nd))

```

函數 測試 (l=200, n=4, 函數=日, 開始位置=(0,0), th=2) , 傳入參數 l、n、函

數、開始位置、th , 架構出潘洛斯鋪磚圖並印出執行時間及個數。

設定瓦片字典為 *global*。將龜指標位置移到 開始位置 , 並設定其角度 0° (方向朝上)。宣告瓦片字典種類為 dict。

設定 追蹤(0) , 即不更新螢幕畫布動作。

時間一 = clock() , 抓取程式執行時間。呼叫 函數(l, n) , 開始計算潘洛斯鋪磚圖。時間二 = clock() , 抓取執行時間。

時呼叫 畫(l, n, th) , 開始畫潘洛斯鋪磚圖。

追蹤(1) , 每一動作更新一次, 即顯現剛繪圖資訊。

時間三 = clock()，抓取執行時間。

將 時間一、 時間二 相減，得到呼叫 函數(l, n) 後的執行時間。

並將之印出，印("計算時間: %7.4f s" % (時間二 - 時間一))。7.4f 為印出

至少 7 位數空間，保留小數點 4 位。

將 時間二、 時間三 相減，得到呼叫 畫(l, n, th) 後的執行時間。

並將之印出，印("繪圖時間: %7.4f s" % (時間三 - 時間二))。

最後將 時間一、 時間三 相減，得到全部執行時間。

並將之印出，印("總和時間: %7.4f s" % (時間三 - 時間一))。

```
def 日(l, n):  
    for i in 範圍(5):  
        膨脹風箏(l, n)  
        左轉(72)
```

函數 日(l, n)，傳入參數 l、n，創造風箏散開圖。

for 迴圈重複執行 5 次 (0~4)。

呼叫 膨脹風箏(l, n)，龜指標左轉 72°(因為 360°/5=72°)。

```
def 膨脹風箏(l, n):  
    if n == 0:  
        px, py = 位置()  
        h, x, y = int(頭向()), round(px, 3), round(py, 3)  
        瓦片字典[(h, x, y)] = 真  
        return  
    fl = f * l
```

```

左轉 (36)

膨脹飛鏢 (fl, n-1)

前進 (1)

右轉 (144)

膨脹風箏 (fl, n-1)

左轉 (18)

前進 (1*d)

右轉 (162)

膨脹風箏 (fl, n-1)

左轉 (36)

前進 (1)

右轉 (180)

膨脹飛鏢 (fl, n-1)

左轉 (36)

```

函數 膨脹風箏(l, n)，傳入參數 l、n，製造風箏並讓風箏散開。

如果 n 為 0，則代表遞迴已經作完，存取現在龜指標位置。

並存取其方向及座標位置(四捨五入後)，存到 瓦片字典 設為真代表為風箏，之後根據這存進去的數據畫圖。

之後根據算式計算其它風箏跟飛鏢位置，並遞迴計算。

```

def 星(l,n):

    for i in 範圍(5):

        膨脹飛鏢(l, n)

        左轉 (72)

```

傳入參數 l 、 n ，飛鏢散開圖。

for 迴圈重複執行 5 次 (0~4)。

呼叫 膨脹飛鏢(l , n)，龜指標左轉 72° (因為 $360^\circ/5=72^\circ$)。

```
def 膨脹飛鏢(l, n):  
  
    if n == 0:  
  
        px, py = 位置()  
  
        h, x, y = int(頭向()), round(px,3), round(py,3)  
  
        瓦片字典[(h,x,y)] = 假  
  
        return  
  
    fl = f * l  
  
    膨脹風箏(fl, n-1)  
  
    左轉(36)  
  
    前進(l)  
  
    右轉(180)  
  
    膨脹飛鏢(fl, n-1)  
  
    左轉(54)  
  
    前進(l*d)  
  
    右轉(126)  
  
    膨脹飛鏢(fl, n-1)  
  
    前進(l)  
  
    右轉(144)
```

函數 膨脹飛鏢(l , n)，傳入參數 l 、 n ，製造飛鏢並讓飛鏢散開。

如果 n 為 0，則代表遞迴已經作完，存取現在龜指標位置。

並存取其方向及座標位置(四捨五入後)，存到 瓦片字典 設為假代表為飛鏢，之後根據這存進去的數據畫圖。

之後根據算式計算其它風箏跟飛鏢位置，並遞迴計算。

```
def 畫(l, n, th=2):  
    清除()  
  
    l = l * f**n  
  
    形狀大小(l/100.0, l/100.0, th)  
  
    for k in 瓦片字典:  
        h, x, y = k  
  
        設位置(x, y)  
  
        設頭向(h)  
  
        if 瓦片字典[k]:  
            形狀("風箏")  
  
            顏色(黑, (0, 0.75, 0))  
  
        else:  
            形狀("飛鏢")  
  
            顏色(黑, (0.75, 0, 0))  
  
    蓋章()
```

函數 畫(l, n, th=2)，傳入參數 l、n、th，利用之前 瓦片字典 存的方向位置及形狀畫圖。

清除()，清除龜指標所畫的圖。

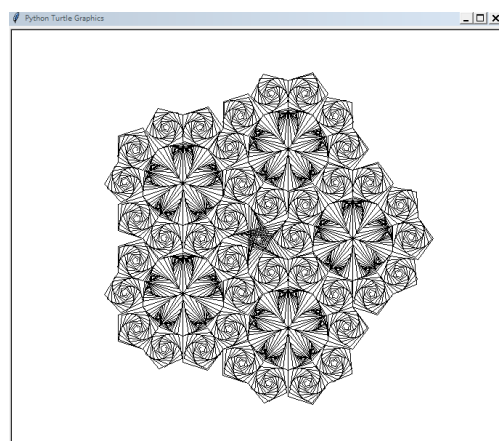
計算並設定形狀大小。

建立 for 迴圈，抓取瓦片字典裡的數據，設定位置及方向，根據存的真假分辨風箏及飛鏢，將龜指標 蓋章() 在畫布上。

15 結合多邊形！tc_bytedesign.py

這次的程式是改編自 PythonCard 龜作圖的範例集。它是基於一篇在 BYTE 雜誌上的文章。用 LOGO 解決問題，利用龜作圖繪圖設計。程式中結合了三邊形、五邊形及星形加上多邊形螺旋線，利用這三種多邊形作結合，來繪出結果。

15-1 執行結果



15-2 程式碼

```
1 #!/usr/bin/env python3
2 '''龜作圖範例集：
```

3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

```
tdemo_bytedesign.py
```

改編自 PythonCard 龜作圖的範例集。

它是基於在 BYTE 雜誌上的一篇文章，

用 LOGO 解決問題：

使用龜作圖重繪設計

1982 年 11 月，118 - 134 頁。

由於該指令，

```
t.delay(0)
```

在第 152 行，將動畫延遲值設為 0。

因此動畫運行每條線都會盡可能快。

```
'''
```

```
import math
```

```
from turtle_tc import *; from turtle_tc import 龜類, 主迴圈
```

```
from time import clock
```

```

29 # 包裝任何額外的繪圖程序
30 # 都需要加以了解
31 class 設計師類(龜類):
32
33     def 設計(我, 家的位置, 比例尺度):
34         我.提筆()
35         for i in 範圍(5):
36             我.前進(64.65 * 比例尺度)
37             我.下筆()
38             我.輪子(我.位置(), 比例尺度)
39             我.提筆()
40             我.後退(64.65 * 比例尺度)
41             我.右轉(72)
42             我.提筆()
43             我.前往(家的位置)
44             我.右轉(36)
45             我.前進(24.5 * 比例尺度)
46             我.右轉(198)
47             我.下筆()
48             我.中央塊(46 * 比例尺度, 143.4, 比例尺度)
49             我.取幕().追蹤(真)
50
51     def 輪子(我, 開始位置, 比例尺度):
52         我.右轉(54)
53         for i in 範圍(4):

```

```

54         我.五之塊(開始位置, 比例尺度)
55     我.下筆()
56     我.左轉(36)
57     for i in 範圍(5):
58         我.三之塊(開始位置, 比例尺度)
59     我.左轉(36)
60     for i in 範圍(5):
61         我.下筆()
62         我.右轉(72)
63         我.前進(28 * 比例尺度)
64         我.提筆()
65         我.後退(28 * 比例尺度)
66     我.左轉(54)
67     我.取幕().更新()
68
69     def 三之塊(我, 開始位置, 比例尺度):
70         舊頭向 = 我.頭向()
71         我.下筆()
72         我.後退(2.5 * 比例尺度)
73         我.右三邊形(31.5 * 比例尺度, 比例尺度)
74         我.提筆()
75         我.前往(開始位置)
76         我.設頭向(舊頭向)
77         我.下筆()
78         我.後退(2.5 * 比例尺度)

```

```

79         我.左三邊形(31.5 * 比例尺度, 比例尺度)
80         我.提筆()
81         我.前往(開始位置)
82         我.設頭向(舊頭向)
83         我.左轉(72)
84         我.取幕().更新()
85
86     def 五之塊(我, 開始位置, 比例尺度):
87         舊頭向 = 我.頭向()
88         我.提筆()
89         我.前進(29 * 比例尺度)
90         我.下筆()
91         for i in 範圍(5):
92             我.前進(18 * 比例尺度)
93             我.右轉(72)
94         我.右五邊形(18 * 比例尺度, 75, 比例尺度)
95         我.提筆()
96         我.前往(開始位置)
97         我.設頭向(舊頭向)
98         我.前進(29 * 比例尺度)
99         我.下筆()
100        for i in 範圍(5):
101            我.前進(18 * 比例尺度)
102            我.右轉(72)
103        我.左五邊形(18 * 比例尺度, 75, 比例尺度)

```

```

104         我.提筆()
105         我.前往(開始位置)
106         我.設頭向(舊頭向)
107         我.左轉(72)
108         我.取幕().更新()
109
110     def 左五邊形(我, 邊, 角度, 比例尺度):
111         if 邊 < (2 * 比例尺度): return
112         我.前進(邊)
113         我.左轉(角度)
114         我.左五邊形(邊 - (.38 * 比例尺度), 角度, 比例尺度)
115
116     def 右五邊形(我, 邊, 角度, 比例尺度):
117         if 邊 < (2 * 比例尺度): return
118         我.前進(邊)
119         我.右轉(角度)
120         我.右五邊形(邊 - (.38 * 比例尺度), 角度, 比例尺度)
121
122     def 右三邊形(我, 邊, 比例尺度):
123         if 邊 < (4 * 比例尺度): return
124         我.前進(邊)
125         我.右轉(111)
126         我.前進(邊 / 1.78)
127         我.右轉(111)
128         我.前進(邊 / 1.3)

```

```

129         我.右轉(146)
130         我.右三邊形(邊 * .75, 比例尺度)
131
132     def 左三邊形(我, 邊, 比例尺度):
133         if 邊 < (4 * 比例尺度): return
134         我.前進(邊)
135         我.左轉(111)
136         我.前進(邊 / 1.78)
137         我.左轉(111)
138         我.前進(邊 / 1.3)
139         我.左轉(146)
140         我.左三邊形(邊 * .75, 比例尺度)
141
142     def 中央塊(我, s, a, 比例尺度):
143         我.前進(s); 我.左轉(a)
144         if s < (7.5 * 比例尺度):
145             return
146         我.中央塊(s - (1.2 * 比例尺度), a, 比例尺度)
147
148     def 主函數():
149         t = 設計師類()
150         t.速度(0)
151         t.藏龜()
152         t.取幕().延遲(0)
153         t.取幕().追蹤(0)

```



```

154     時間一 = clock()
155     t.設計(t.位置(), 2)
156     時間二 = clock()
157     return "執行時間: %.2f 秒。" % (時間一-時間二)
158
159 if __name__ == '__main__':
160     訊息 = 主函數()
161     印(訊息)
162     主迴圈()

```

15-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```

if __name__ == '__main__':
    訊息 = 主函數()
    印(訊息)
    主迴圈()

```

__name__ 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則__name__變數會等於 '__main__'，程序便會執行其下列動作。

主函數() 被呼叫執行，其為本程式內部函數。將回傳的字串存進訊息。

將訊息印出。

主迴圈 `()` 被呼叫執行，呼叫後進入使用者圖形介面(GUI)的主迴圈

`mainloop()`，等候使用者進一步使用滑鼠鍵盤來控制。

```
def 主函數():
```

主函數`()`，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():  
  
    t = 設計師類()  
  
    t.速度(0)  
  
    t.藏龜()  
  
    t.取幕().延遲(0)  
  
    t.取幕().追蹤(0)  
  
    時間一 = clock()  
  
    t.設計(t.位置(), 2)  
  
    時間二 = clock()  
  
    return "執行時間: %.2f 秒。" % (時間一-時間二)
```

宣告一個設計師類名為 `t`。因其繼承龜類，所以也擁有龜類的屬性方法。

將 `t` 速度設為 `0`，亦及最快速度，其移動時直接移到最終結果。

`t.藏龜()`，將龜指標隱藏。

取 `t` 的螢幕並設置其更新延遲值為 `0`。

取 `t` 的螢幕並設置其停止更新。（也代表每 `0` 步更新一次螢幕）

時間一 = clock()，抓取現在處理器時間。

呼叫設計師類的方法 t.設計，傳入 t 的位置及比例尺。

最後回傳包含執行時間的字串，其利用時間一剪掉時間二。

```
class 設計師類(龜類):
```

定義 class，名為設計師類，其繼承龜類。

```
class 設計師類(龜類):
```

```
    def 設計(我, 家的位置, 比例尺度):
```

```
        我.提筆()
```

```
        for i in 範圍(5):
```

```
            我.前進(64.65 * 比例尺度)
```

```
            我.下筆()
```

```
            我.輪子(我.位置(), 比例尺度)
```

```
            我.提筆()
```

```
            我.後退(64.65 * 比例尺度)
```

```
            我.右轉(72)
```

```
        我.提筆()
```

```
        我.前往(家的位置)
```

```
        我.右轉(36)
```

```
        我.前進(24.5 * 比例尺度)
```

```
        我.右轉(198)
```

```
        我.下筆()
```

```
        我.中央塊(46 * 比例尺度, 143.4, 比例尺度)
```

```
        我.取幕().追蹤(真)
```

方法 設計，傳入參數為家的位置、比例尺度，。

設置提筆，移動時不畫出軌跡。

利用 for 迴圈重複執行下列動作，因為總共有五個輪子，所以重複五次，抓取數字為 0~4。

前進 64.65 * 比例尺度。

並設置下筆 ()，移動時會畫出軌跡。

呼叫 我.輪子，開始畫輪子，傳入現在位置跟比例尺度。

設置提筆，移動時不畫出軌跡。

後退 64.65 * 比例尺度。回到原位。

右轉 72 度（因為總共有五個大小一致的輪子，因此 $360^\circ/5=72^\circ$ ）

執行五次後，迴圈結束。（圖可參考下方）

設置提筆，移動時不畫出軌跡。

前往到 家的位置，即原點。

右轉 36 度，此時頭向為 324 度。（角度由東逆時鐘計算）

前進 24.5 * 比例尺度。此時龜指標到中間星星部分的頂端附近。

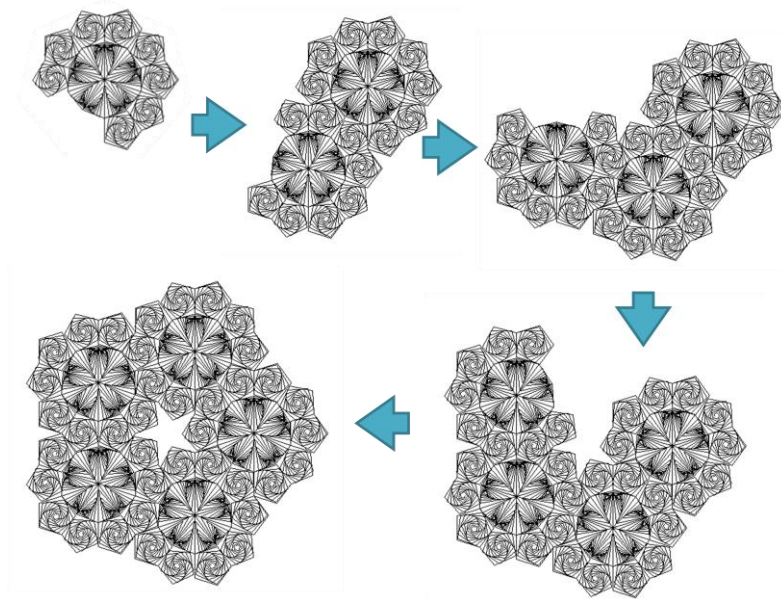
再右轉 198 度，此時頭向為 126 度。

設置下筆 ()，移動時會畫出軌跡。

呼叫中央塊 ()，傳入參數，開始畫中間星星的區塊。

繪圖結束後則取幕 () . 追蹤 (真)，將畫面更新。

五次迴圈：



```
class 設計師類 (龜類) :
```

```
def 輪子 (我, 開始位置, 比例尺度) :
```

```
    我.右轉 (54)
```

```
    for i in 範圍 (4) :
```

```
        我.五之塊 (開始位置, 比例尺度)
```

```
    我.下筆 ()
```

```
    我.左轉 (36)
```

```
    for i in 範圍 (5) :
```

```
        我.三之塊 (開始位置, 比例尺度)
```

```
    我.左轉 (36)
```

```
    for i in 範圍 (5) :
```

```
        我.下筆 ()
```

```
        我.右轉 (72)
```

```
我.前進(28 * 比例尺度)

我.提筆()

我.後退(28 * 比例尺度)

我.左轉(54)

我.取幕().更新()
```

方法 輪子，傳入參數 開始位置、比例尺度，利用三之塊和五之塊方法來結合製造輪子圖案。

右轉 54 度。

利用 for 迴圈重複執行，數字範圍 0~3，共執行 4 次。

呼叫方法 五之塊，傳入開始位置跟比例尺。

執行完 4 次後，for 迴圈結束，此時圖為〈圖一〉。

設置 下筆()，讓龜指標移動時會畫出軌跡。

並左轉 36 度，準備開始畫三之塊。

利用 for 迴圈重複執行，數字範圍 0~4，共執行 5 次。

呼叫方法 三之塊，傳入開始位置跟比例尺。

執行完 5 次後，for 迴圈結束，此時圖為〈圖二〉。

之後再左轉 36 度。準備開始畫中間往外的五條線，如〈圖三〉紅線所示。

再利用 for 迴圈重複執行，數字範圍 0~4，共執行 5 次。

設置 下筆()，讓龜指標移動時會畫出軌跡。

右轉 72 度，前進 28 * 比例尺度，畫出線。

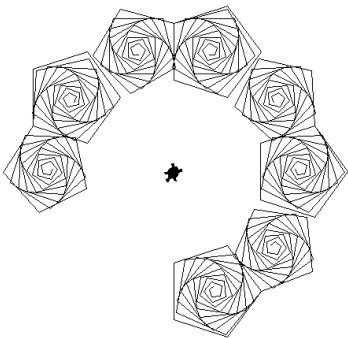
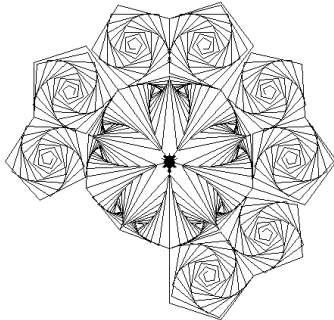
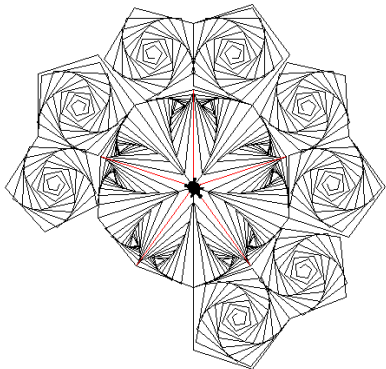
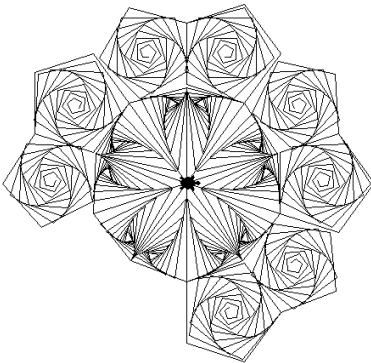
設置 提筆 ()，讓龜指標移動時不會畫出軌跡。

再後退 28 * 比例尺度，回到剛剛位置。

重複執行五次，則畫完五條線了。

此時圖也畫完了，最後將龜指標左轉 54 度，回到最一開始傳入的指標方向。

並抓取自身所在的螢幕，將其更新顯現出所畫的圖。此時狀態如<圖四>。

圖一：	圖二：
	
圖三：	圖四：
	

```
class 設計師類(龜類):
```

```
def 三之塊(我, 開始位置, 比例尺度):
```

```
    舊頭向 = 我.頭向()
```

```
    我.下筆()
```

```
    我.後退(2.5 * 比例尺度)
```

```
    我.右三邊形(31.5 * 比例尺度, 比例尺度)
```

```
    我.提筆()
```

```
    我.前往(開始位置)
```

```
    我.設頭向(舊頭向)
```

```
    我.下筆()
```

```
    我.後退(2.5 * 比例尺度)
```

```
    我.左三邊形(31.5 * 比例尺度, 比例尺度)
```

```
    我.提筆()
```

```
    我.前往(開始位置)
```

```
    我.設頭向(舊頭向)
```

```
    我.左轉(72)
```

```
    我.取幕().更新()
```

方法 三之塊，傳入參數開始位置、比例尺度，主要將右三邊形與左三邊形方法的圖案作結合。

將現在的頭向角度記錄下來，命名為舊頭向。

設置 下筆()，移動時會畫出軌跡。

後退 2.5 * 比例尺度。

呼叫方法 右三邊形，傳入參數 31.5 * 比例尺度（邊）、比例尺度，畫出

往右轉的三邊形。

設置 提筆()，移動時不會畫出軌跡。

將位置移回原本傳入的開始位置，將頭向改回 舊頭向。

再設置 下筆()，移動時會畫出軌跡。

並準備開始畫另一邊的三邊形，因此步驟與之前一樣。

先後退 2.5 * 比例尺度。

再呼叫方法 左三邊形，傳入參數 31.5 * 比例尺度（邊）、比例尺度，畫

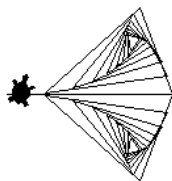
出往左轉的另一個三邊形。

最後設置 提筆()，移動時不會畫出軌跡。並回到原本開始位置，將頭

向改回 舊頭向。

設置左轉 72 度。

並抓取自身所在的螢幕，將其更新顯現出所畫的圖。



```
class 設計師類(龜類):
```

```
    def 五之塊(我, 開始位置, 比例尺度):
```

```
        舊頭向 = 我.頭向()
```

```
        我.提筆()
```

```

我.前進(29 * 比例尺度)

我.下筆()

for i in 範圍(5):

    我.前進(18 * 比例尺度)

    我.右轉(72)

我.右五邊形(18 * 比例尺度, 75, 比例尺度)

我.提筆()

我.前往(開始位置)

我.設頭向(舊頭向)

我.前進(29 * 比例尺度)

我.下筆()

for i in 範圍(5):

    我.前進(18 * 比例尺度)

    我.右轉(72)

我.左五邊形(18 * 比例尺度, 75, 比例尺度)

我.提筆()

我.前往(開始位置)

我.設頭向(舊頭向)

我.左轉(72)

我.取幕().更新()

```

方法 五之塊，傳入參數開始位置、比例尺度，主要將右五邊形與左五邊形方法的圖案作結合。

將現在的頭向角度記錄下來，命名為舊頭向。

設置 提筆()，移動時不會畫出軌跡。

前進 29 * 比例尺度。

設置 下筆 ()，移動時會畫出軌跡。

利用 for 迴圈重複執行下列動作，重複數字範圍為 0~4，總共 5 次。

前進 29 * 比例尺度。

右轉 72 度。

重複五次後 for 迴圈結束，可得到一個正五邊形，如〈圖一〉。

此時呼叫右五邊形函數，畫出正五邊形內的漩渦圖案，如〈圖二〉。

之後 提筆 ()，讓移動座標時不會畫出軌跡。

將位置移回原本傳入的開始位置，並將頭向改回 舊頭向。

並準備開始畫另一邊的五邊形，因此步驟與之前相似。

前進 29 * 比例尺度。

設置 下筆 ()，移動時會畫出軌跡，準備開始作圖。

再一次利用 for 迴圈重複執行下列動作，重複數字範圍為 0~4，總共 5 次。

前進 29 * 比例尺度。

右轉 72 度。

重複五次後 for 迴圈結束，龜指標會沿著剛剛畫正五邊形的路，回到

〈圖一〉龜指標的位置，如〈圖三〉。

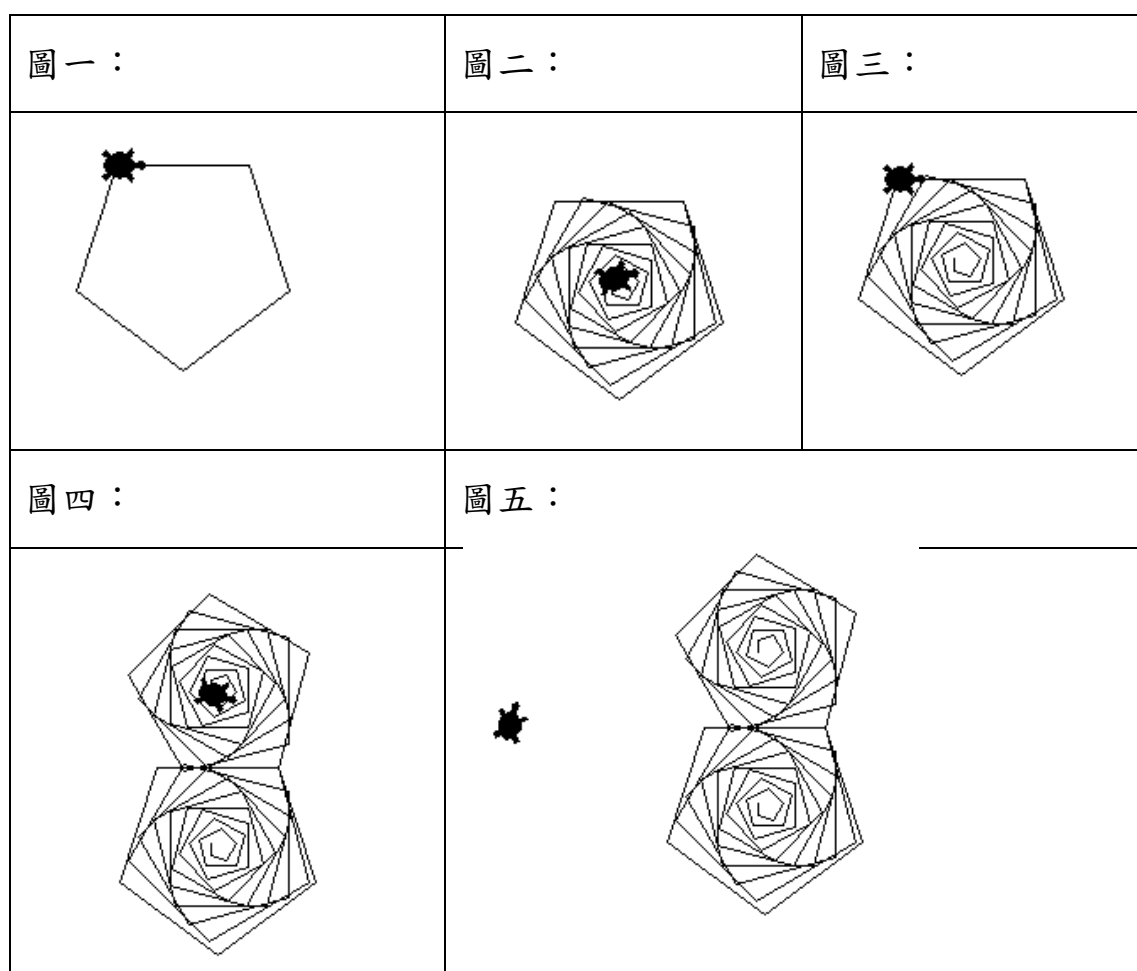
之後呼叫左五邊形函數，可畫出另一邊五邊形，圖案也就完成了，如

<圖四>。

最後設置 提筆(`penup`)，移動時不會畫出軌跡。並回到原本開始位置，將頭向改回 舊頭向。

設置左轉 72 度，等待下一個動作。

並抓取自身所在的螢幕，將其更新顯現出所畫的圖。此時狀態如<圖五>。



`class` 設計師類(龜類)：

```

def 左五邊形(我, 邊, 角度, 比例尺度):

    if 邊 < (2 * 比例尺度): return

    我.前進(邊)

    我.左轉(角度)

    我.左五邊形(邊 - (.38 * 比例尺度), 角度, 比例尺度)

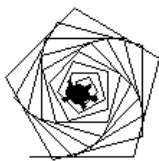
```

方法 左五邊形，傳入參數 邊、角度、比例尺度。此為遞迴函數，此圖案可與 右五邊形 方法圖案相對應，此圖為一個趨近於正五邊形的圖案，但一邊傾斜角度及邊長持續遞減，直到邊長小於 $2 * \text{比例尺度}$ 。圖可參考下方。

當邊小於 $2 * \text{比例尺度}$ ，則 return 亦即結束執行。

前進 邊 距離。左轉 角度 度數。

將邊減掉 $0.38 * \text{比例尺度}$ ，傳入本身函數，繼續遞迴。



```

class 設計師類(龜類):

    def 右五邊形(我, 邊, 角度, 比例尺度):

        if 邊 < (2 * 比例尺度): return

        我.前進(邊)

        我.右轉(角度)

        我.右五邊形(邊 - (.38 * 比例尺度), 角度, 比例尺度)

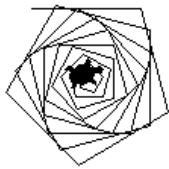
```

方法 右五邊形，傳入參數 邊、比例尺度。此為遞迴函數，此圖案可與 左五邊形 方法圖案相對應，指令也大致相同，主要差別在於右五邊形轉彎為右轉。圖可參考下方。

當邊小於 $2 * \text{比例尺度}$ ，則 return 亦即結束執行。

前進 邊 距離。右轉 角度 度數。

將邊減掉 $0.38 * \text{比例尺度}$ ，傳入本身函數，繼續遞迴。



```
class 設計師類(龜類):
```

```
    def 右三邊形(我, 邊, 比例尺度):
```

```
        if 邊 < (4 * 比例尺度): return
```

```
        我.前進(邊)
```

```
        我.右轉(111)
```

```
        我.前進(邊 / 1.78)
```

```
        我.右轉(111)
```

```
        我.前進(邊 / 1.3)
```

```
        我.右轉(146)
```

```
        我.右三邊形(邊 * .75, 比例尺度)
```

方法 右三邊形，傳入參數 邊、比例尺度。此為遞迴函數，此圖案可與

左三邊形 方法圖案相對應，此圖為一個三邊形的圖案，但邊傾斜角度

及邊長持續遞減，直到邊長小於 $4 * \text{比例尺度}$ 。圖可參考下方。

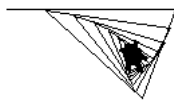
當邊小於 $4 * \text{比例尺度}$ ，則 return 亦即結束執行。

前進 邊 距離。右轉 111 度。

再前進 邊 / 1.78 距離。右轉 111 度。

最後再前進 邊 / 1.3 距離。右轉 146 度。

將邊乘上 0.75，傳入本身函數，繼續遞迴。



```
class 設計師類(龜類):
```

```
    def 左三邊形(我, 邊, 比例尺度):
```

```
        if 邊 < (4 * 比例尺度): return
```

```
        我.前進(邊)
```

```
        我.左轉(111)
```

```
        我.前進(邊 / 1.78)
```

```
        我.左轉(111)
```

```
        我.前進(邊 / 1.3)
```

```
        我.左轉(146)
```

```
        我.左三邊形(邊 * .75, 比例尺度)
```

方法 左三邊形，傳入參數 邊、比例尺度。此為遞迴函數，此圖案可與

右三邊形 方法圖案相對應，指令也大致相同，主要差別在於左三邊形

轉彎為左轉。圖可參考下方。

當邊小於 $4 * \text{比例尺度}$ ，則 return 亦即結束執行。

前進 邊 距離。左轉 111 度。

再前進 邊 / 1.78 距離。左轉 111 度。

最後再前進 邊 / 1.3 距離。左轉 146 度。

將邊乘上 0.75，傳入本身函數，繼續遞迴。



```
class 設計師類(龜類):
```

```
    def 中央塊(我, s, a, 比例尺度):
```

```
        我.前進(s); 我.左轉(a)
```

```
        if s < (7.5 * 比例尺度):
```

```
            return
```

```
        我.中央塊(s - (1.2 * 比例尺度), a, 比例尺度)
```

方法 中央塊，傳入參數 s、a、比例尺度。此為遞迴函數，畫一個趨近

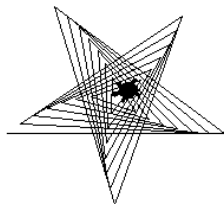
於正星形的圖案，但邊傾斜角度及邊長持續遞減，直到邊長小於 $7.5 * \text{比例尺度}$

比例尺度。其圖可參考下方。

前進距離 s，並向左轉 a 角度。

如果 s 小於 $7.5 \times$ 比例尺度 則 return 亦即結束執行。

不然則將 s 減掉 $1.2 \times$ 比例尺度，傳入本身函數，繼續遞迴。



16 與電腦 PKtc_nim.py

捻遊戲已流傳很久，據說來自於中國，藉由被販賣到美洲的勞工外傳。其遊戲主要利用石頭或硬幣遊玩，最為常見的玩法為將硬幣分為三列，分別為三、四、五個硬幣，兩人輪流取硬幣，每次只能取一個或以上的硬幣在同列上，玩到最後，拿到最後一個硬幣的人就贏了。

到了如今其遊戲的規則也漸漸多樣化，每列的數量、取硬幣的數量及最後輸贏的決定等等，而有一名哈佛大學數學系副教授 Chales Leonard Bouton 提出了分析和證明，利用二進位法，對有隨機列數，而每列有隨機個數，如何達到必勝法則。

在其中，局面上被拿過後剩下的殘局只分為兩種：安全的局面 (safe)、不安全的局面 (unsafe)，當自方取完後殘局處於安全的局面下

時，則對方不管怎麼拿都會處於不安全的局面，之後自方只要在適當的某列取下適當的數量則又會處於安全的局面，這樣玩到最後則可獲得勝利。而如果自方留下不安全的局面，則對方可在適當的某列取下適當的數量，成為安全局面，則持續下去則會輸掉比賽。因此最重要的則是在於看哪一方能將狀況處在自方拿取後會留下安全局面的贏。

（如果目前比賽是最後取的人則輸，則相反，哪一方能將狀況處在自方拿取後會留下不安全局面的贏）。而此時我們在這介紹其必勝的二

進位算法：

$3 = 011$ $4 = 100$ $5 = 101$ $+= 212$ 此時為不安全的 殘局	如為 3,4,5 個，則換成二進位 相加三位無皆為偶數(0,2)則為不 安全的殘局
$1 = 001$ $4 = 100$ $5 = 101$ $+= 202$ 此時為不安全的 殘局	此時從第一列取走 2 個，變 為 1,4,5 個，換成二進位相加三位 皆為偶數則為安全的殘局

在此程式中我們利用互斥或(詳細介紹可參閱下方)達到目的，因數字相加為奇數時，則互斥或也會為奇數，因此可藉由此得知是否為安全的局面。

互斥或：

$a \wedge b \wedge c$	$1 \wedge 1 = 0$	$1 \wedge 0 = 1$	$0 \wedge 1 = 1$	$0 \wedge 0 = 0$
0	0	1	1	0
1	1	0	0	1

相加：

$a+b+c$	$1+1=2$	$1+0=1$	$0+1=1$	$0+0=0$
0	2	1	1	0
1	3	2	2	1

在此程式中我們利用 MVC 模式將遊戲架構系統分為三部份：模型（Model）、視圖（View）、控制（Controller）。MVC 主要目的為實作一個動態的程式設計，利用 MVC 分別寫成一個類別。其功能區分為

模型（Model）：負責控制事件，對事件進行處理。

視圖（View）：負責圖形介面設計，顯示資訊。

控制（Controller）：負責程式功能。

由於我們在此需要揀棒子，所以還要建造一個類別 棒子類，繼承龜類，擁有龜類的屬性，可將其轉換顏色(區別已拿取、可拿取)、監聽事件(滑鼠點擊)等等。

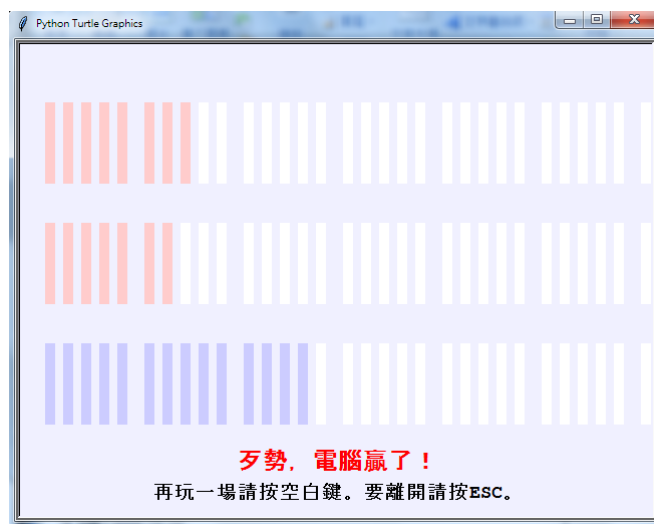
我們可以藉由這個程式學習到如何製作一個遊戲。

★ 互斥或，利用位元運算得到結果，逐位元進行比較運算，如下

A^B	0	1
0	0	1
1	1	0

範例: $6^8 \Rightarrow 0110^1000 = 1110 \Rightarrow 14$

16-1 執行結果



16-2 程式碼

```

1  '''龜作圖範例集：
2
3      tdemo_nim.py
4
5  與電腦玩 nim 遊戲。
6  誰拿走最後一根的贏。
7
8  實現了 MVC 模式 (Model-View-Controller) 的設計。
9

```

```

10  '''
11
12
13  import turtle_tc as turtle; from turtle_tc import *
14  import random
15  import time
16
17  幕寬常數 = 640
18  幕高常數 = 480
19
20  最小棒子數常數 = 7
21  最大棒子數常數 = 31
22
23  垂直單位常數 = 幕高常數 // 12
24  水平單位常數 = 幕寬常數 // ((最大棒子數常數 // 5) * 11 + (最大棒子數常數 % 5) * 2)
25
26  S 顏色常數 = (63, 63, 31)
27  H 顏色常數 = (255, 204, 204)
28  顏色常數 = (204, 204, 255)
29
30  def 隨機列():
31      return random.randint(最小棒子數常數, 最大棒子數常數)
32
33  def 電腦揀棒子(狀態):
34      互斥或 = 狀態[0] ^ 狀態[1] ^ 狀態[2]

```

```

35     if 互斥或 == 0:
36         return 隨機移動(狀態)
37     for z in 範圍(3):
38         s = 狀態[z] ^ 互斥或
39         if s <= 狀態[z]:
40             移動 = (z, s)
41         return 移動
42
43 def 隨機移動(狀態):
44     m = max(狀態)
45     while 真:
46         z = random.randint(0,2)
47         if 狀態[z] > (m > 1):
48             break
49     亂數 = random.randint(m > 1, 狀態[z]-1)
50     return z, 亂數
51
52
53 class 拾遊戲模型類(object):
54     def __init__(我, 遊戲):
55         我.遊戲 = 遊戲
56
57     def 設立(我):
58         if 我.遊戲.狀態 not in [拾遊戲類.被創造狀態, 拾遊戲類.遊戲結束狀態]:
59             return

```

```
60     我.棒子們 = [隨機列(), 隨機列(), 隨機列()]
61     我.玩家 = 0
62     我.贏家 = 無
63     我.遊戲.視圖.設立()
64     我.遊戲.狀態 = 捻遊戲類.正在跑狀態
65
66     def 移動(我, 列, 行):
67         最大分裂數 = 我.棒子們[列]
68         我.棒子們[列] = 行
69         我.遊戲.視圖.通知移動(列, 行, 最大分裂數, 我.玩家)
70     if 我.遊戲結束():
71         我.遊戲.狀態 = 捻遊戲類.遊戲結束狀態
72         我.贏家 = 我.玩家
73         我.遊戲.視圖.通知結束()
74     elif 我.玩家 == 0:
75         我.玩家 = 1
76         列, 行 = 電腦揀棒子(我.棒子們)
77         我.移動(列, 行)
78         我.玩家 = 0
79
80     def 遊戲結束(我):
81         return 我.棒子們 == [0, 0, 0]
82
83     def 通知移動(我, 列, 行):
84         if 我.棒子們[列] <= 行:
85             return
```

```

86         我.移動(列, 行)
87
88
89 class 棒子類(turtle.龜類):
90     def __init__(我, 列, 行, 遊戲):
91         turtle.龜類.__init__(我, visible=假)
92         我.列 = 列
93         我.行 = 行
94         我.遊戲 = 遊戲
95         x, y = 我.座標們(列, 行)
96         我.形狀(方形)
97         我.形狀大小(垂直單位常數/10.0, 水平單位常數/20.0)
98         我.速度(0)
99         我.提筆()
100        我.前往(x, y)
101        我.顏色(白)
102        我.顯龜()
103
104    def 座標們(我, 列, 行):
105        包裹, 餘數 = divmod(行, 5)
106        x = (3 + 11 * 包裹 + 2 * 餘數) * 水平單位常數
107        y = (2 + 3 * 列) * 垂直單位常數
108        return x - 幕寬常數 // 2 + 水平單位常數 // 2, 幕高常數 // 2 - y - 垂直單位常數 // 2
109
110    def 令移動(我, x, y):

```



```

111         if 我.遊戲.狀態 != 捻遊戲類.正在跑狀態:
112             return
113         我.遊戲.控制者.通知移動(我.列, 我.行)
114
115
116 class 捻遊戲視圖類(object):
117     def __init__(我, 遊戲):
118         我.遊戲 = 遊戲
119         我.幕 = 遊戲.幕
120         我.模型 = 遊戲.模型
121         我.幕.色模式(255)
122         我.幕.追蹤(假)
123         我.幕.背景色((240, 240, 255))
124         我.寫手 = turtle.龜類(visible=假)
125         我.寫手.提筆()
126         我.寫手.速度(0)
127         我.棒子們 = {}
128         for 列 in 範圍(3):
129             for 行 in 範圍(最大棒子數常數):
130                 我.棒子們[(列, 行)] = 棒子類(列, 行, 遊戲)
131         我.展現("... 請等一下 ...")
132         我.幕.追蹤(真)
133
134     def 展現(我, 訊息1, 訊息2=無):
135         我.幕.追蹤(假)

```

```

136         我.寫手.清除()
137     if 訊息 2 is not 無:
138         我.寫手.前往(0, - 幕高常數 // 2 + 48)
139         我.寫手.筆色(紅)
140         我.寫手.寫(訊息 2, align="center", font=("Courier",18,"bold"))
141     我.寫手.前往(0, - 幕高常數 // 2 + 20)
142     我.寫手.筆色(黑)
143     我.寫手.寫(訊息 1, align="center", font=("Courier",14,"bold"))
144     我.幕.追蹤(真)
145
146 def 設立(我):
147     我.幕.追蹤(假)
148     for 列 in 範圍(3):
149         for 行 in 範圍(我.模型.棒子們[列]):
150             我.棒子們[(列, 行)].顏色(S 顏色常數)
151     for 列 in 範圍(3):
152         for 行 in 範圍(我.模型.棒子們[列], 最大棒子數常數):
153             我.棒子們[(列, 行)].顏色(白)
154     我.展現("輪到你！點擊一根棒子來移走它及其右邊的棒子")
155     我.幕.追蹤(真)
156
157 def 通知移動(我, 列, 行, 最大分裂數, 玩家):
158     if 玩家 == 0:
159         色彩 = H 顏色常數
160         for s in 範圍(行, 最大分裂數):

```

```

161         我.棒子們[(列, s)].顏色(色彩)
162     else:
163         我.展現(" ... 思考中 ... ")
164         time.sleep(0.5)
165         我.展現(" ... 思考中 ... aaah ...")
166         色彩 = 顏色常數
167     for s in 範圍(最大分裂數-1, 行-1, -1):
168         time.sleep(0.2)
169         我.棒子們[(列, s)].顏色(色彩)
170         我.展現("輪到你！點擊一根棒子來移走它及其右邊的棒子")
171
172 def 通知結束(我):
173     if 我.遊戲.模型.贏家 == 0:
174         訊息 2 = "恭喜，你贏了！"
175     else:
176         訊息 2 = "歹勢，電腦贏了！"
177     我.展現("再玩一場請按空白鍵。要離開請按 ESC。", 訊息 2)
178
179 def 清除(我):
180     if 我.遊戲.狀態 == 撿遊戲類.遊戲結束狀態:
181         我.幕.清除()
182
183
184 class 撿遊戲控制者類(object):
185

```

```

186 def __init__(我, 遊戲):
187     我.遊戲 = 遊戲
188     我.棒子們 = 遊戲.視圖.棒子們
189     我.繁忙狀態 = 假
190     for 棒子 in 我.棒子們.values():
191         棒子.在點擊時(棒子.令移動)
192     我.遊戲.幕.在按鍵時(我.遊戲.模型.設立, 空白鍵)
193     我.遊戲.幕.在按鍵時(我.遊戲.視圖.清除, 脫離鍵)
194     我.遊戲.視圖.展現("請按下空白鍵開始遊戲")
195     我.遊戲.幕.聽()
196
197 def 通知移動(我, 列, 行):
198     if 我.繁忙狀態:
199         return
200     我.繁忙狀態 = 真
201     我.遊戲.模型.通知移動(列, 行)
202     我.繁忙狀態 = 假
203
204
205 class 撿遊戲類(object):
206     被創造狀態 = 0
207     正在跑狀態 = 1
208     遊戲結束狀態 = 2
209     def __init__(我, 幕):
210         我.狀態 = 撿遊戲類.被創造狀態

```

```

211         我.幕 = 幕
212         我.模型 = 撿遊戲模型類(我)
213         我.視圖 = 撿遊戲視圖類(我)
214         我.控制者 = 撿遊戲控制者類(我)
215
216
217     def 主函數():
218         主螢幕 = turtle.幕類()
219         主螢幕.模式(角度從東開始逆時針)
220         主螢幕.設立(幕寬常數, 幕高常數)
221         撿 = 撿遊戲類(主螢幕)
222         return "事件迴圈"
223
224 if __name__ == "__main__":
225     主函數()
226     turtle.主迴圈()

```

16-3 程式碼執行解說

```
from turtle_tc import *
```

引入非官方的 turtle_tc 模組，可利用中文名稱呼叫 turtle 模組中的函數及變數，但呼叫後也可使用原英文名稱呼叫。

```

if __name__ == "__main__":
    主函數()
    turtle.主迴圈()

```

`__name__` 變數通常用來偵測本程式是否作為 獨立的主程式 被作業系統呼叫來執行。如其程式是直接執行而非呼叫執行，則 `__name__` 變數會等於 `'__main__'`，程序便會執行其下列動作。

主函數 `()` 被呼叫執行，其為本程式內部函數。

主迴圈 `()` 被呼叫執行，其為 `turtle_tc` 的函數，呼叫後進入使用者圖形介面(GUI)的主迴圈 `mainloop()`，等候使用者進一步使用滑鼠鍵盤來控制。

```
幕寬常數 = 640
幕高常數 = 480

最小棒子數常數 = 7
最大棒子數常數 = 31

垂直單位常數 = 幕高常數 // 12
水平單位常數 = 幕寬常數 // ((最大棒子數常數 // 5) * 11 + (最大棒子數常數 % 5) * 2)

S 顏色常數 = (63, 63, 31)
H 顏色常數 = (255, 204, 204)
顏色常數 = (204, 204, 255)
```

宣告在最外部，所以程式皆可以存取。

```
def 主函數():
```

主函數 `()`，其括號為空白，代表無任何傳入變數。主函數通常為程式的主體架構，由此建構出程式骨架，再由此向外呼叫附加函數。

```
def 主函數():
    主螢幕 = turtle.幕類()
    主螢幕.模式(角度從東開始逆時針)
    主螢幕.設立(幕寬常數, 幕高常數)
    捻 = 捻遊戲類(主螢幕)
    return "事件迴圈"
```

由幕類建造出一個物件名為主螢幕。

設置主螢幕的模式角度從東開始逆時針。

設定螢幕寬度及高度。

建立 撿遊戲類 之物件，在此就直接命名為 撿，將主螢幕傳入。準備開始建造遊戲。

最後回傳 "事件迴圈" 字串。

```
class 撿遊戲類(object):
```

```
    被創造狀態 = 0
```

```
    正在跑狀態 = 1
```

```
    遊戲結束狀態 = 2
```

建造 撿遊戲類(object) 類別，建立遊戲系統。

宣告參數用來區別系統狀態。

```
class 撿遊戲類(object):
```

```
    被創造狀態 = 0
```

```
    正在跑狀態 = 1
```

```
    遊戲結束狀態 = 2
```

```
    def __init__(我, 幕):
```

```
        我.狀態 = 撿遊戲類.被創造狀態
```

```
        我.幕 = 幕
```

```
        我.模型 = 撿遊戲模型類(我)
```

```
        我.視圖 = 撿遊戲視圖類(我)
```

```
        我.控制者 = 撿遊戲控制者類(我)
```

建立 撿遊戲類 類別的初始設定。

設定目前狀態為 被創造狀態。

並設定傳入的 幕類，為幕。

將整個系統分為三個架構：模型、視圖、控制者。

創造 撿遊戲模型類(我) 設定為模型。

創造 撿遊戲視圖類(我) 設定為視圖。

創造 撿遊戲控制者類(我) 設定為控制者。

```
class 撿遊戲模型類(object):
```

創建 撿遊戲模型類 類別，其設定遊戲的開始狀態（棒子的數量）、結束狀態及移動棒子。

```
class 撿遊戲模型類(object):  
    def __init__(我, 遊戲):  
        我.遊戲 = 遊戲
```

初始設定，設定遊戲為傳入的 撿遊戲類。

```
class 撿遊戲模型類(object):  
    def 設立(我):  
        if 我.遊戲.狀態 not in [撿遊戲類.被創造狀態, 撿遊戲類.遊戲結束狀態]:  
            return  
        我.棒子們 = [隨機列(), 隨機列(), 隨機列()]  
        我.玩家 = 0  
        我.贏家 = 無  
        我.遊戲.視圖.設立()  
        我.遊戲.狀態 = 撿遊戲類.正在跑狀態
```

方法 設立()，當遊戲開始或結束要開啟下一局時，設定棒子、玩家及視圖為初始狀態，並將遊戲狀態改為 正在跑狀態(=1)。

當遊戲的狀態並不是 被創造狀態(=0) 及 遊戲結束狀態(=2)，等於當遊戲狀態為 正在跑狀態(=1)，則 return 亦即跳出這方法。

如無跳出，則將三排的棒字類隨機決定數量。

並宣告屬性玩家變數為 0(代表玩家為使用者)，宣告贏家變數為無。

呼叫視圖的方法設立，重新設定視圖。

並將狀態設為 正在跑狀態(=1)，開始玩遊戲。

```
class 撿遊戲模型類(object):  
    def 移動(我, 列, 行):  
        最大分裂數 = 我.棒子們[列]  
        我.棒子們[列] = 行  
        我.遊戲.視圖.通知移動(列, 行, 最大分裂數, 我.玩家)  
        if 我.遊戲結束():  
            我.遊戲.狀態 = 撿遊戲類.遊戲結束狀態
```



```

        我.贏家 = 我.玩家
        我.遊戲.視圖.通知結束()
    elif 我.玩家 == 0:
        我.玩家 = 1
        列, 行 = 電腦揀棒子(我.棒子們)
        我.移動(列, 行)
        我.玩家 = 0

```

方法 移動，傳入參數列、行，用來移動棒子。

將要移動列可移動的棒子數傳入最大分裂數。

並將剩下可移動的數字存回 我.棒子們[列]。

呼叫 遊戲.視圖.通知移動(列, 行, 最大分裂數, 我.玩家)，改變顏色。

如果呼叫 遊戲結束() 回傳為真，則代表無可動棒子，遊戲結束。

設 遊戲.狀態 為 遊戲結束狀態 (=2)。

並設定贏家為 玩家，因其動了最後的棒子。

最後呼叫 遊戲.視圖.通知結束()，讓視圖顯示遊戲結束。

如果呼叫 遊戲結束() 回傳為假且 玩家 為 0(代表玩家為使用者)。

則此時設為 1(代表玩家為電腦)。

且呼叫 電腦揀棒子，傳入 棒子們，接受回傳的列、行。

再將玩家設為 0。

```

class 撿遊戲模型類(object):
    def 遊戲結束(我):
        return 我.棒子們 == [0, 0, 0]

```

方法 遊戲結束，無傳入參數，查看遊戲是否結束。

如果三列都無可移動的棒子，則回傳真，代表遊戲結束。

```

class 撿遊戲模型類(object):
    def 通知移動(我, 列, 行):
        if 我.棒子們[列] <= 行:
            return
        我.移動(列, 行)

```

方法 通知移動，傳入參數列、行，用來查看棒子是否可以移動。

如果傳入的 行 參數小於 我.棒子們[列]，亦即代表其並不是可移動的棒子，則 `return` 亦即跳出這方法。

否則呼叫 移動(列, 行)，移動棒子。

```
class 撿遊戲視圖類(object):
```

創建 撿遊戲視圖類 類別，負責處理遊戲系統的繪圖顯示。

```
class 撿遊戲視圖類(object):
    def __init__(我, 遊戲):
        我.遊戲 = 遊戲
        我.幕 = 遊戲.幕
        我.模型 = 遊戲.模型
        我.幕.色模式(255)
        我.幕.追蹤(假)
        我.幕.背景色((240, 240, 255))
        我.寫手 = turtle.龜類(visible=假)
        我.寫手.提筆()
        我.寫手.速度(0)
        我.棒子們 = {}
        for 列 in 範圍(3):
            for 行 in 範圍(最大棒子數常數):
                我.棒子們[(列, 行)] = 棒子類(列, 行, 遊戲)
        我.展現("... 請等一下 ...")
        我.幕.追蹤(真)
```

建立 撿遊戲視圖類 類別的初始設定。

設定遊戲為傳入的 撿遊戲類。

並設定其屬性幕為遊戲的 幕類。

設定其模型為遊戲的 撿遊戲模型類。

設定其幕的色模式為 255(RPG 的值範圍區間為 0..255)。

將幕的追蹤設置為假，畫面停止更新。

設定幕的背景色為 (240, 240, 255) 。

建立龜類為寫手，`visible=假`，龜指標不可見。

寫手提筆，移動時不畫線。速度設為 0 最快速，移動時指標直接移到結果位置。

建立屬性棒子們為內建字典型態(dict)，裡面為空。

建立 for 迴圈，重複 3 次(0~2)，因總共有 3 列棒子。

在裡面再建立一個 for 迴圈，重複 最大棒子數常數 次(31 次=0~30)，因每列的棒子最多有 31 根。

將列行參數及遊戲傳入 棒子類，建立棒子。並將列行當作 key 值，棒子類當作 value 值存入棒子們。

兩個 for 迴圈執行完，總共會創造 3*31 根棒子。

呼叫 展現 方法，在畫面顯示字串"... 請等一下 ..."。

並將幕的追蹤設置為真，畫面顯示更新。

```
class 撿遊戲視圖類(object):
    def 展現(我, 訊息1, 訊息2=無):
        我.幕.追蹤(假)
        我.寫手.清除()
        if 訊息2 is not 無:
            我.寫手.前往(0, - 幕高常數 // 2 + 48)
            我.寫手.筆色(紅)
            我.寫手.寫(訊息2, align="center", font=("Courier",18,"bold"))
        我.寫手.前往(0, - 幕高常數 // 2 + 20)
        我.寫手.筆色(黑)
        我.寫手.寫(訊息1, align="center", font=("Courier",14,"bold"))
        我.幕.追蹤(真)
```

方法 展現，傳入參數 訊息1, 訊息2=無，如無傳入 訊息2 則預設為無。

將幕的追蹤設置為假，畫面停止更新。

將寫手畫的圖全部清除，但位置不移動。

如果不是無，則前往到座標 $(0, - \text{幕高常數} // 2 + 48) = (0, -240 + 48)$ ($//$ 與/一樣但為整數)。設定筆色為紅，且寫下 訊息 2。

之後前往到座標 $(0, - \text{幕高常數} // 2 + 20) = (0, -240 + 20)$ 。

設定筆色為黑，寫下 訊息 1。

將幕的追蹤設置為真，畫面開始更新，將字顯現在螢幕上。

```
class 撿遊戲視圖類(object):  
    def 設立(我):  
        我.幕.追蹤(假)  
        for 列 in 範圍(3):  
            for 行 in 範圍(我.模型.棒子們[列]):  
                我.棒子們[(列, 行)].顏色(s 顏色常數)  
        for 列 in 範圍(3):  
            for 行 in 範圍(我.模型.棒子們[列], 最大棒子數常數):  
                我.棒子們[(列, 行)].顏色(白)  
        我.展現("輪到你！點擊一根棒子來移走它及其右邊的棒子")  
        我.幕.追蹤(真)
```

方法 設立，無傳入參數，設定遊戲視圖。

將幕的追蹤設置為假，畫面停止更新。

建立 for 迴圈，重複 3 次(0~2)，。

在迴圈裡再建立 for 迴圈，根據 模型.棒子們 的數字決定次數。

將棒子顏色改變，等於將亂數決定出每排有幾根棒子變色。

將棒子都變色為結束 for 迴圈。

建立 for 迴圈，重複 3 次(0~2)，。

在迴圈裡再建立 for 迴圈，重複 最大棒子數常數 的數字扣掉 模型.棒子們 數字。

將剩下沒被變色的棒子設為白色。

將棒子都變色為結束 for 迴圈。

呼叫方法 展現，在畫面顯示"輪到你！點擊一根棒子來移走它及其右邊的棒子"。

將幕的追蹤設置為真，畫面開始更新，將棒子顏色跟字顯現在螢幕上。

```
class 撿遊戲視圖類(object):
    def 通知移動(我, 列, 行, 最大分裂數, 玩家):
        if 玩家 == 0:
            色彩 = H 顏色常數
            for s in 範圍(行, 最大分裂數):
                我.棒子們[(列, s)].顏色(色彩)
        else:
            我.展現(" ... 思考中 ... ")
            time.sleep(0.5)
            我.展現(" ... 思考中 ... aaah ...")
            色彩 = 顏色常數
            for s in 範圍(最大分裂數-1, 行-1, -1):
                time.sleep(0.2)
                我.棒子們[(列, s)].顏色(色彩)
            我.展現("輪到你！點擊一根棒子來移走它及其右邊的棒子")
```

方法 撿遊戲視圖類，傳入參數列、行、最大分裂數、玩家，將移動後的棒子換色。

如果 玩家 為 0(代表玩家為使用者)，使用者在移動，將 色彩設為 H 顏色常數。

藉用 for 迴圈將傳入的參數 行 到 最大分裂數，將棒子從左到右改變顏色。

(行 為玩家挑選變色的棒子行數，最大分裂數 為最多可變色的位置。)

否則代表如今為電腦移動，則在畫面顯示" ... 思考中 ... "。

停止 0.5 秒，在畫面顯示" ... 思考中 ... aaah ... "。

將 色彩設為 顏色常數。

藉用 for 迴圈將傳入的參數 最大分裂數-1 到 行-1，將棒子從右到左改變顏色。

之後在畫面顯示"輪到你！點擊一根棒子來移走它及其右邊的棒子"。

```
class 撿遊戲視圖類(object):  
    def 通知結束(我):  
        if 我.遊戲.模型.贏家 == 0:  
            訊息 2 = "恭喜，你贏了！"  
        else:  
            訊息 2 = "歹勢，電腦贏了！"  
        我.展現("再玩一場請按空白鍵。要離開請按 ESC。", 訊息 2)
```

方法 通知結束，無傳入參數，顯示贏家及遊戲結束。

如果此時 遊戲.模型.贏家 為 0，代表贏家為使用者，則將"恭喜，你贏了！"

傳入 訊息 2。

否則代表贏家為電腦，則將"歹勢，電腦贏了！"傳入 訊息 2。

之後將"再玩一場請按空白鍵。要離開請按 ESC。"及 訊息 2 顯示在畫面上。

```
class 撿遊戲視圖類(object):  
    def 清除(我):  
        if 我.遊戲.狀態 == 撿遊戲類.遊戲結束狀態:  
            我.幕.清除()
```

方法 清除，無傳入參數，將畫面清除。

如果 遊戲.狀態 為 遊戲結束狀態(=2)。則將螢幕畫面清除。

```
class 撿遊戲控制者類(object):
```

創建 撿遊戲視圖類 類別，控管點擊按鍵事件。

```
class 撿遊戲控制者類(object):  
    def __init__(我, 遊戲):  
        我.遊戲 = 遊戲  
        我.棒子們 = 遊戲.視圖.棒子們  
        我.繁忙狀態 = 假  
        for 棒子 in 我.棒子們.values():  
            棒子.在點擊時(棒子.令移動)  
        我.遊戲.幕.在按鍵時(我.遊戲.模型.設立, 空白鍵)  
        我.遊戲.幕.在按鍵時(我.遊戲.視圖.清除, 脫離鍵)  
        我.遊戲.視圖.展現("請按下空白鍵開始遊戲")
```

```
我.遊戲.幕.聽()
```

建立 撿遊戲控制者類 類別的初始設定。傳入參數為遊戲。

設定遊戲為傳入的 撿遊戲類。

並設定其屬性棒子們為遊戲屬性視圖的 棒子類。

設定其屬性 繁忙狀態 為假。

藉由 for 迴圈抓取棒子們的 value 值，亦即 棒子類。

並設定棒子如果被點擊時則呼叫棒子類的方法 令移動。

之後如果使用者按下空白鍵則呼叫 遊戲.模型.設立 方法。

之後如果使用者按下脫離鍵(ESC 鍵)則呼叫 遊戲.視圖.清除 方法。

並呼叫 遊戲.視圖.展現，在畫面顯示"請按下空白鍵開始遊戲"。

最後呼叫 遊戲.幕.聽()，監聽鍵盤事件。

```
class 撿遊戲控制者類(object):
```

```
    def 通知移動(我, 列, 行):
```

```
        if 我.繁忙狀態:
```

```
            return
```

```
        我.繁忙狀態 = 真
```

```
        我.遊戲.模型.通知移動(列, 行)
```

```
        我.繁忙狀態 = 假
```

方法 通知移動，傳入參數列、行，通知棒子開始移動並移動中時設為繁忙狀態。

如果 我.繁忙狀態 等於真，則 return 亦即跳出這方法。

否則 我.繁忙狀態 設定為真。

並呼叫 遊戲.模型.通知移動(列, 行)，開始移動。

移動完，我.繁忙狀態 設定為假。

```
class 棒子類(turtle.龜類):
```

建立棒子類，繼承龜類。產生棒子物件，設定其屬性及方法。

```
class 棒子類(turtle.龜類):
```

```

def __init__(我, 列, 行, 遊戲):
    turtle.龜類.__init__(我, visible=假)
    我.列 = 列
    我.行 = 行
    我.遊戲 = 遊戲
    x, y = 我.座標們(列, 行)
    我.形狀(方形)
    我.形狀大小(垂直單位常數/10.0, 水平單位常數/20.0)
    我.速度(0)
    我.提筆()
    我.前往(x, y)
    我.顏色(白)
    我.顯龜()

```

棒子類初始設定，設定棒子屬性、位置、形狀大小。

`turtle.龜類.__init__(我, visible=假)`，設定指標不可見。

設定屬性列、行。並設定其遊戲是屬於哪個 撿遊戲類。

並且呼叫方法 座標們，將列行傳入，得到座標回傳存入 `x, y`。

並設定其龜指標形狀為方形和形狀大小。

設置速度為 0 即最快速，移動時指標直接移到結果位置。

提筆()，移動時指標不畫線。

並前往到指定座標 `(x, y)`。設定其顏色為白色。

顯龜()，並將龜指標顯示。

```

class 棒子類(turtle.龜類):
    def 座標們(我, 列, 行):
        包裹, 餘數 = divmod(行, 5)
        x = (3 + 11 * 包裹 + 2 * 餘數) * 水平單位常數
        y = (2 + 3 * 列) * 垂直單位常數
        return x - 幕寬常數 // 2 + 水平單位常數 // 2, 幕高常數 // 2 - y - 垂直單位常數 // 2

```

方法，傳入參數列、行，根據棒子的列行算出所在座標。

`包裹, 餘數 = divmod(行, 5)`，回傳行除以 5 的商數與餘數。

根據 包裹, 餘數 算出 x 座標, 將 包裹 乘上 11(原本應該是 10 但為了讓它每五個空一個, 所以再加上 1)剩下的 餘數 乘上 2, 兩個總和再加上 3(為了空下左邊), 最後全部乘上 水平單位常數。

將列乘上 3 加上 2, 最後乘上 垂直單位常數。

回傳計算出來的座標(因座標(0.0)在中心)。

```
class 棒子類(turtle.龜類):  
    def 令移動(我, x, y):  
        if 我.遊戲.狀態 != 撿遊戲類.正在跑狀態:  
            return  
        我.遊戲.控制者.通知移動(我.列, 我.行)
```

方法 令移動, 傳入參數 x,y, 亦即點擊座標。

如果遊戲的狀態不是正在跑狀態, 則跳出這方法(藉由 *return* 跳出)。

之後呼叫 遊戲.控制者.通知移動, 傳入列、行(此棒子的列行)。

```
def 隨機列():  
    return random.randint(最小棒子數常數, 最大棒子數常數)
```

函數 隨機列(), 無傳入參數, 回傳隨機亂數。

回傳隨機亂數, 值範圍在 7 到 31。

```
def 電腦揀棒子(狀態):  
    互斥或 = 狀態[0] ^ 狀態[1] ^ 狀態[2]  
    if 互斥或 == 0:  
        return 隨機移動(狀態)  
    for z in 範圍(3):  
        s = 狀態[z] ^ 互斥或  
        if s <= 狀態[z]:  
            移動 = (z, s)  
    return 移動
```

函數 電腦揀棒子, 傳入參數 狀態(棒子可移動數字), 讓電腦移動棒子。

將串列的三個數字用位元互斥(^)運算。

如果其等於 0，則為安全局面，此時不管怎麼取，都會在不安全的局面，因此可隨便取得數字，呼叫函數 隨機移動，傳入 狀態，之後將結果回傳。

利用 for 迴圈執行下列動作，重複 3 次(0~2)，因為棒子共有三列。

狀態[z] ^ 互斥或 得到的數字，如果小於等於 狀態[z] 數字。

則將 z 與 s 回傳。(選擇第 z 列的第 s 行棒子，可達到安全局面)

(可參考下方舉例)

舉例：

假設目前棒子狀態為 4 根 3 根 2 根	
狀態=[4,3,2]	狀態[0] = 100 狀態[1] = 011 狀態[2] = 010
將三者狀態互斥可得知是否為安全的局面，如結果為 0 則為安全的局面，在此為 5 所以是不安全的局面，因此需設法改為安全的局面	
互斥或 = 狀態[0] ^ 狀態[1] ^ 狀態[2] 互斥或 => 5	100 => 4 011 => 3 ^) 010 => 2 <hr/> 101 => 5
由此可知須更改第一跟第三位的數字，才能將結果變為 0。因此我們將 5 與三個狀態互斥，得到各排改變的數字，如果數字比狀態的數字小，則代表拿取此狀態的棒子可改變。	
狀態[z] ^ 互斥或 = s = 1 狀態[z] ^ 互斥或 = s = 6 狀態[z] ^ 互斥或 = s = 7	001 => 可拿取 110 => 數字大於本身可拿取數量 111 => 數字大於本身可拿取數量

將拿取棒子數量與互斥或的關係可參考下方公式推導	
狀態[0] ^ 狀態[1] ^ 狀態[2] = 互斥或 = 互斥或 ^ s ^ 狀態[1] ^ 狀態[2] => 互斥或 ^ 0 = 互斥或 => s ^ 狀態[1] ^ 狀態[2] = 0	$100 \wedge 011 \wedge 010 = 101$ $= (101 \wedge 001) \wedge 011 \wedge 010$ $101 \wedge 000 = 101$ $\Rightarrow 001 \wedge 011 \wedge 010 = 000$
因此我們拿取的 0 排的棒子，拿取到剩 1 個，此時狀態為 1 根 3 根 2 根，將狀態互斥可得到 0，所以可得知此時為安全的局面	
狀態=[1,3,2] 互斥或 = 狀態[0] ^ 狀態[1] ^ 狀態[2] 互斥或 => 0	$001 \Rightarrow 1$ $011 \Rightarrow 3$ $\wedge) 010 \Rightarrow 2$ <hr/> $000 \Rightarrow 0$

def 隨機移動(狀態):

m = max(狀態)

while 真:

z = random.randint(0,2)

if 狀態[z] > (m > 1):

break

亂數 = random.randint(m > 1, 狀態[z]-1)

return z, 亂數

函數 隨機移動，傳入參數 狀態(棒子可移動數字)，。

m = max(狀態)，找出串列最大的數字。

while 真，當為真則重複執行下列動作(因此此處會一直執行直到跳出)。

z = random.randint(0,2)，從 0~2 隨機選出一個數字。

如果其最大數字大於 1 且 狀態[z] 也大於 1 或 0(真為 1 假為 0)，則跳出 *while* 迴圈。

亂數 = random.randint(m > 1, 狀態[z]-1)，從 1 或 0(真為 1 假為 0)到指定的最大可移動數減 1，隨機選一個數字。

回傳 z, 亂數。