Python 基礎

江清水 撰寫

東吳大學資訊管理系教授

(2015/11)

初階

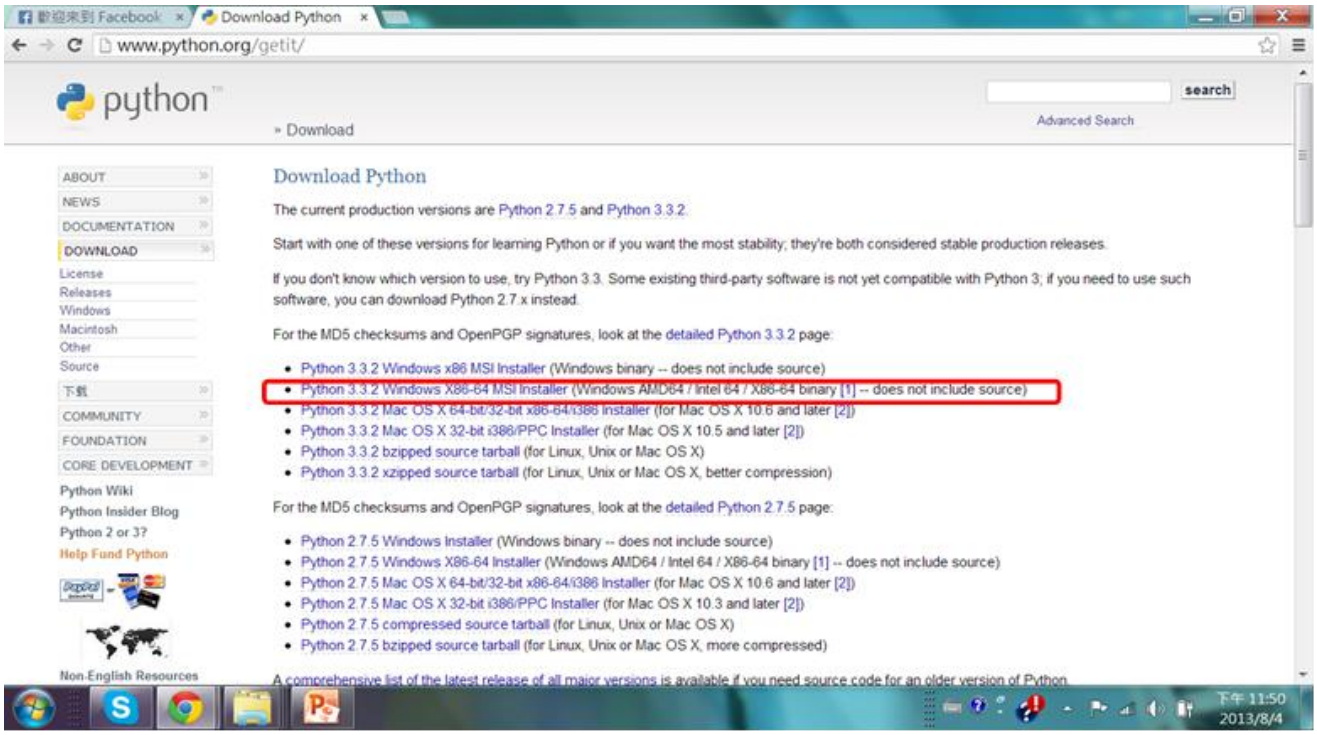
python 介紹

我們依照以下步驟安裝 python 3.3.2，其中 3.3.2 表示他的版本。步驟如下:

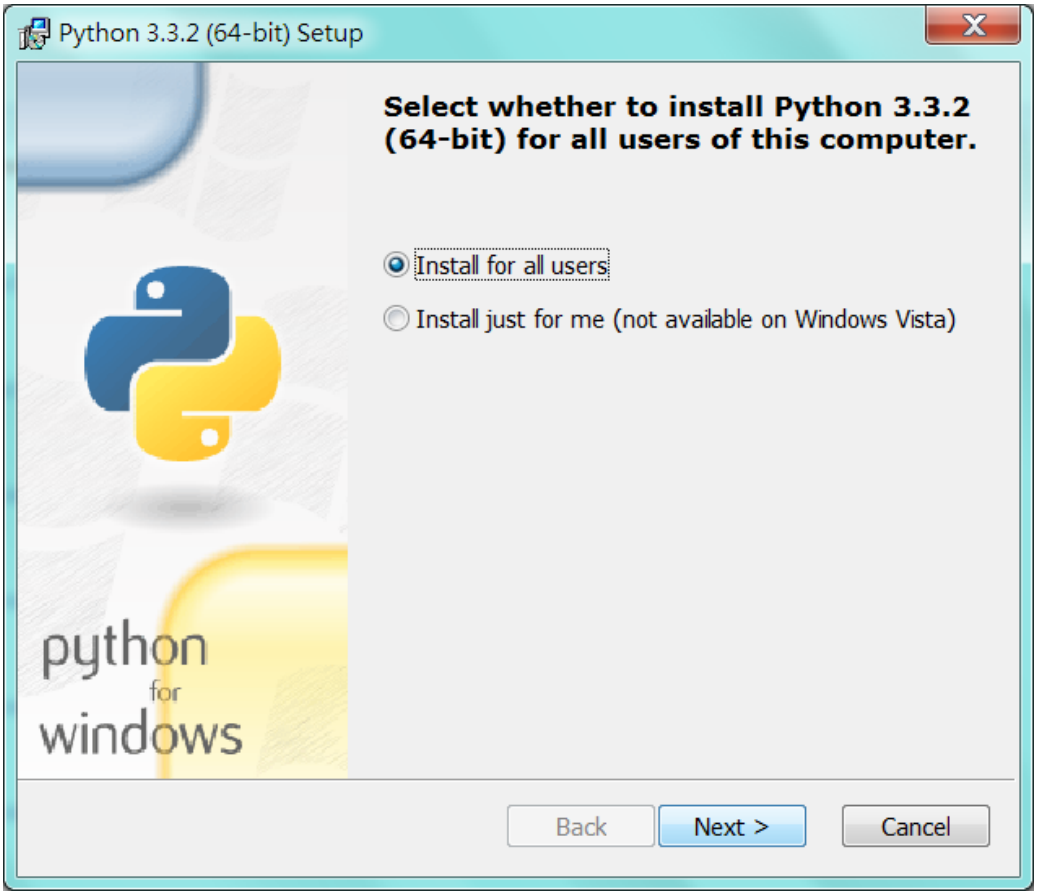
1. 進入 http://www.python.org/getit
2. 選擇體要的作業系統，如:
   Python3.3.2 Windows X86-64MSI Installer
   下載相關軟體
3. 執行 Python-3.3.2.amd64.msi
4. 進入 C:\python33\lib\idlelib 執行 idle(windows 批次檔案)

詳述如下:

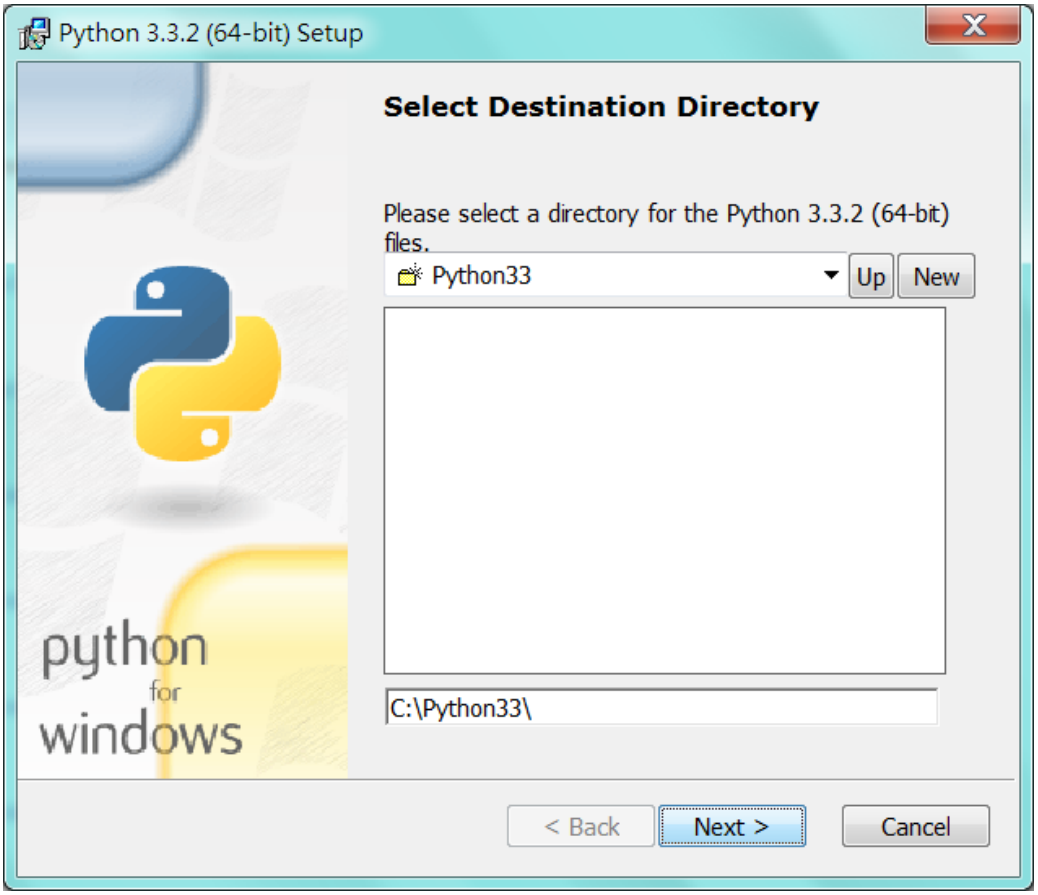
1. 進入 http://www.python.org/getit
2. 選擇體要的作業系統，如:
   Python3.3.2 Windows X86-64MSI Installer
   下載相關軟體



(3) 執行 Python-3.3.2.amd64.msi，畫面如下:



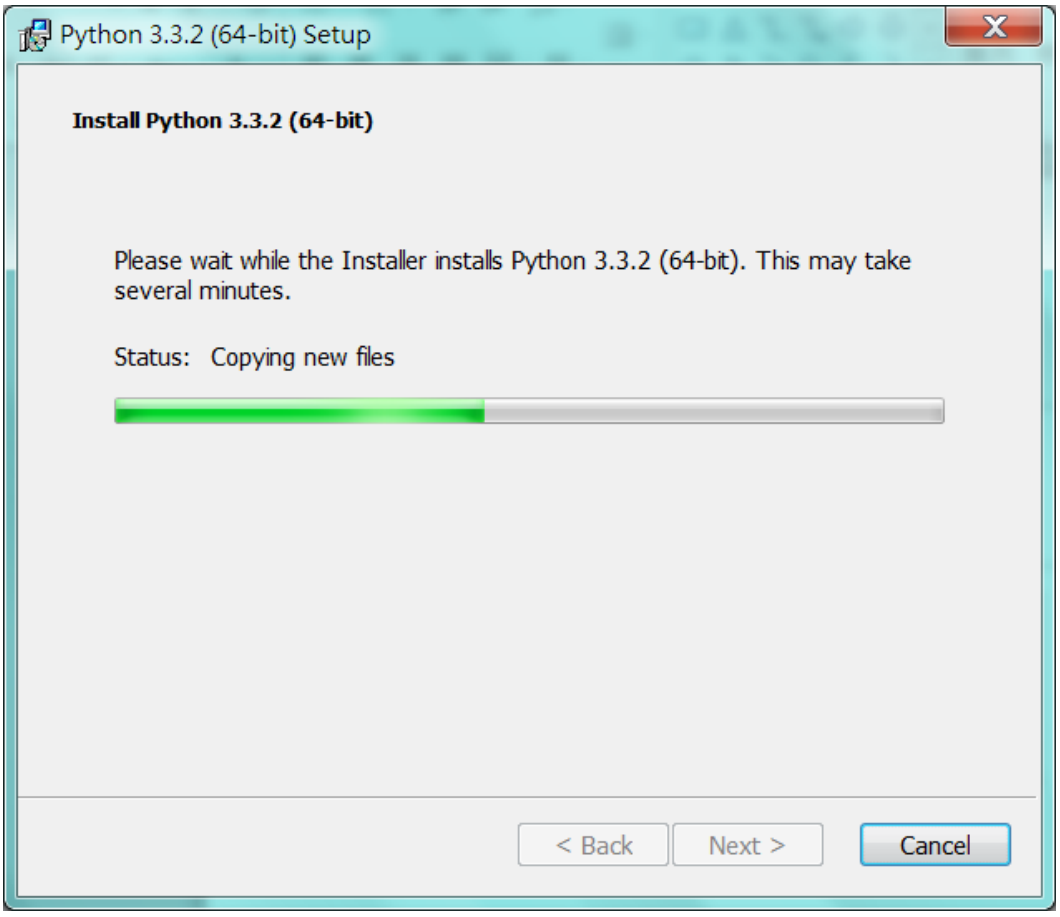
選擇 Install for all users，按 Next


Python 3.3.2 (64-bit) Setup
Select Destination Directory
Please select a directory for the Python 3.3.2 (64-bit) files.
Python33
Up
New
C:\Python33\
python for windows
< Back
Next >
Cancel

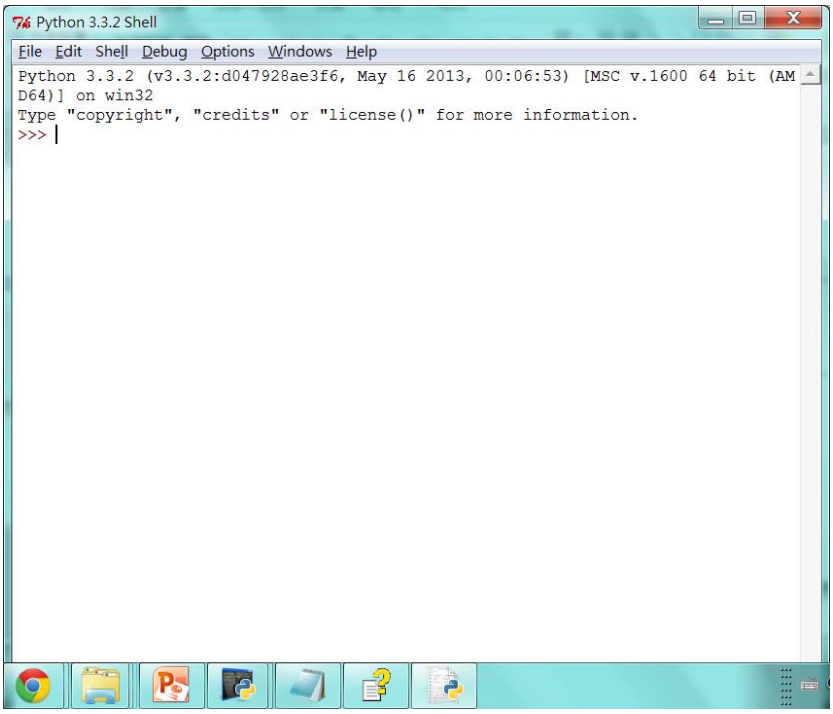
按 Next


Python 3.3.2 (64-bit) Setup
Customize Python 3.3.2 (64-bit)
Select the way you want features to be installed.
Click on the icons in the tree below to change the way features will be installed.
Python
Register Extensions
Tcl/Tk
Documentation
Utility Scripts
Test suite
Add python.exe to Path
Python Interpreter and Libraries
This feature requires 21MB on your hard drive. It has 5 of 6 subfeatures selected. The subfeatures require 38MB on your hard drive.
python for windows
Disk Usage
Advanced
< Back
Next >
Cancel

按 Next



執行中，請耐心等待



執行完成

我們安裝完畢後，就要測試如何執行。我們可以再>>>後方打入 3+5，如果有正確回答，就表示安裝大致完成。

資料型態:數及其運算的介紹

我們可以將 Python 程式當作一個計算機，在上面做數值上的計算。在學 python 初期，我們緊緊針對整數介紹，這些整數包括正整數、負整數和零。在這些整數上的計算，我們介紹+, -, *, /, //, %, **七種運算。如下:

+, -, *, /: 就是我們最常學的加、減、乘、除。

//: 整數除法，就是兩整數相除後取其商。所以 9//2=4

%: 除法餘數，就是兩整數相除後取餘數。所以 9%4=1

: 次方。23 就是二的三次方，也就是 8。

我們不打算在此教整數的運算，但是針對整數的運算，如何使用 python 求解，才是我們探討的問題。注意到上面的七個不同的運算，其中六個的計算結果都是整數，只有除法不一定會產生整數的結果，所以 python 會給一個有小數點的數當結果，因此 6/3=2.0。

練習:

使用 python 來計算以下的算式結果:

(1) 1+3

(2) 9-5

(3) 4*8-9

(4) 25/4

(5) 25//4

(6) 25%4

(7) 2**10

Assignment statement 介紹 by 江清水(2015/11/22)

assignment 的語法如下:

var = expression

以 a=3 為例，左邊有一個變數(variable)a, 中間有一個'='號,右邊有一個描述(expression)，這個可以是一個數 3，也可以是數的四則運算，如 3+5 等。我們記做 a=3 或 a=3+5。它的功能是將某一個描述 expression 運算後的值，存在叫做 a 的變數中。我們可以把變數 a 想成一個個抽屜的標籤，例如有一個叫做'a'的抽屜，打開抽屜裏面放著一個含有數值(計算後的 e)的卡片。

我們用幾個不同的程式(指令)來介紹他的動作。

1.

```
a=3
```

a 第一次出現，所以我們找一個空的抽屜，給它命名為 a，做一張寫有 3 的字卡放入抽屜 a 中。

2.

```
a=3*5+2
```

a 第一次出現，所以我們找一個空的抽屜，給它命名為 a，先計算 3*5+2=17，做一張寫 17 的字卡放入抽屜 a 中。

3.

```
a=3
a=5
```

這個題目開始有兩個 assignment statement。在程式中，後寫的程式都是比較晚執行，它的執行方式是由上往下，當上一個 statement(assignment 是 statement 的一種,以

後還會介紹 if statement 等)結束後，下一個 statement 就開始。在此例中，一開始 a=3, a 第一次出現，所以我們找一個空的抽屜，給它命名為 a，做一張寫有 3 的字卡放入抽屜 a 中。
接著 a=3，我們找到名子叫做 a 的抽屜，把裡面的字卡拿出來，不管它裡面存甚麼，都把它改成 5，然後放回抽屜。

4.

```
a = 2
b = 3*a + 5
```

一開始 a=3，找一個新抽屜，命名 a，把值 2 放入 a 抽屜。
第二個指令 b=3*a+5，注意到這裡的 a 不是找抽屜，而是找抽屜裏面字卡的值，目前是 3，然後計算 3*3+5=14,因為目前還沒有一個抽屜叫做 b,所以我們找一個空的抽屜，命名為 b，拿一張卡片寫上 14 放入抽屜 b。

5.

```
a = 3
a = a+1
```

一開始 a=3，找一個新抽屜，命名 a，把值 2 放入 a 抽屜。
接著，a=a+1，注意到現在有兩個 a，一個在等號的左邊，另一個在右邊。和以上的例子一樣，在等號左邊的我們找抽屜，右邊的找抽屜裏面的值。因此，這個指令計算右邊的 a+1=4，然後將值 4 放回抽屜 a 裡面。注意到前面的值 3 已經被值 4 蓋過去不見了。

6.

```
a = b
```

這個指令在找 b 的值時，因為找不到一個叫做 b 的抽屜(以前沒有建立過)，所以沒有辦法執行。

7.

```
total = 0
i = 5
total = total + i
```

這三個指令結束後，有兩個抽屜 i, total，裡面的值各別是 i=5, total = 5。

8.

```
total = 0
total = total + 1
total = total + 2
total = total + 3
total = total + 4
total = total + 5
```

這六個指令結束後,有一個抽屜,它的值是 total = 1+2+3+4+5 = 15。當然，我們可以用 total = total + 1 + 2 + 3 + 4 + 5 來取代最後的指令。

9.

```
i = 0
total = 0
i = i + 1
total = total + i
i = i + 1
total = total + i
i = i + 1
total = total + i
i = i + 1
total = total + i
i = i + 1
total = total + i
```

這些指令結束後，有兩個抽屜 i,和 total，裡面的值各別是 i=5(i = i + 1 執行了五次)，total = 15。

注意到我們重複在計算兩個指令，i = i + 1, total = total + i。如果我們能有其他的指令來執行這些重複計算的指令就太好了。未來我們講到 for 的時候會再介紹。

在每一個程式結束，我們可以加一個 print(a) 的指令將 a 的值印出來,或者用 print(i, total) 把 i 和 total 兩數的值印出來。

Python 有一個指令，就是左邊有一串變數(variables)，右邊有同數目的一串描述(expressions)，python 會將右邊的每一個描述先計算出來，先存成一個暫時的數列，然後在一個個放入左邊的變數中。

舉例來說:

```
a=3
b=2
print(a,b)
a,b = b, a
print(a,b)
```

print 是一個印出結果的程式，會將內部的所有數值印出，也就是 print(a)會印出 a 的值，print(a,b)則是將 a, b 的值印在同一列(print 的使用容後再詳述)。在此程式中，第一個 print 會印出 3, 2。在執行 a,b = b,a 時，python 會先將 b,a 計算出來，也就是 2, 3，然後再把 2,3 放到左邊的 a,b,也就是抽屜 a 放 2,抽屜 b 放 3。這個指令結束後，第二個 print 就會印出 2,3 了。

注意到 a, b=b,a 不能用 a=b, b=a 兩個指令來取代。考慮以下程式:

```
a=3
b=2
print(a,b)
a=b
b=a
print(a,b)
```

在這個指令中，第一個 print 印出 3,2。然後，執行 a=b 時，我們把 b 的值 2 放入抽屜 a 中，目前的狀況是抽屜 a 和抽屜 b 各別儲存數值 2。當我們再執行下一個指令 b=a,我們取出 a 的值 2，放入抽屜 b 中，並沒有改變 a,b 抽屜都放 2 的事實。因此,最後一個 print statement 會印出 2,2。與前一個例子印出 2,3 是不同的。

要避免使用 a,b = b,a 的指令，但要能讓 a,b 的值互換，我們需要另一個變數 c 來當中間的一個變數，也就是:

```
a=3
b=2
print(a,b)
c=a
a=b
b=c
print(a,b)
```

這時，第一個 print 會印出 3,2，第二個 print 就會印出 2,3 了。

作業

1. 寫一個程式印出 1+3+5，並將結果印出。

2. 給定 a, b,c 三個變數並給定三個值，寫出一個數將 a 的值給 b,b 的值給 c,c 的值給 a，但是不要使用 a,b,c=c,a,b 這個指令。以下面程式為例:

```
a=2
b=3
c=5
print(a,b,c) # 印出 a, b,c，也就是 2,3,5
#.........(你寫的程式)
print(a,b,c)# 將 a,b,c,現在應該是 5,2,3
```

後記: 在電腦的術語中，記憶體存放很多的數值，而記憶體有編號，這些編號稱為位址。位址在我們文中就是我們的抽屜名稱，把一個數值 5 放到位址(假設 8bits)01011101 不容易讓初學的懂，因此我們用抽屜來模擬資料儲存於記憶體的方法。

資料型態:字串的介紹

在 python 中，字串是使用""來包含，也就是說 Hello World 的字串是"Hello World".我們可以將一個字串給一個變數，如 s = "Hello World"。所以，"13"是一個長度為 2 的字串，而 13 是一個數。在字串中有許多的運算，包括
str(n)是將一個數字轉換成一個字串，例如
str(12) = "12"
int(str)則是將一個字串改為數字，例如
int("12") =12。
len(str)是回傳字串 str 的長度，例如
len("Hello")=5。

另外，字串也可以有家法，例如"Hello"+"World" = "HelloWorld"，"Hello "+"World"="Hello World"。注意到第二個例子多了一個空格，空格在字串中也要計算在內，因此，len("Hello ")=6。

我們如何取到一個字串中的一個字母?舉例來說，s="Hello"，我們給 s 一個從 0 開始計算的指標，就可以指定到特定的一個字母。也就是說 s[0]=H，s[1]=e, s[2]=s[3]=l, s[4]=o。另外，我們也可以由後往前數，如此數，s[-1]=o, s[-2]=s[-3]=l，s[-4]=e, s[-5]=H。

資料型態:Boolean 的介紹

把一個描述的結果能夠用 True 或 False 來回答，我們稱這一個描述為 boolean expression。舉例來說，今天有下雨這個描述，可以回答 True 或者是 False。當今天真的下雨，答案就是 True，否則就是 False。

兩數的大小比較有大於(>)，小於(<)，等於(==)，大於等於(>=)，小於等於(<=)和不等於(!=)。我們可以利用這些大小比較來產生 boolean expression。所以 boolean expression 可以是 5>3(True)或者是 5==3+1(False)。注意到比較"等於"是用兩個等號來表達，以免跟 assignment 搞混。

在 Boolean 的運算中有三個不同的運算元，叫做 not, and, or，介紹如下:

1. not C

not 將 True 改為 False，將 False 改為 True。也就是說當 C 是 True，not C 就變成 False。反之亦然。舉例來說，Boolean expression 是 not (今天下雨)，今天真下雨，(今天下雨)的值是 True，not(今天下雨)意思就是今天沒下雨，可是今天真下雨，因此就變成 False。

2. A and B

and 連接兩個運算子 A 和 B，當 A 和 B 的值都是 True 時，A and B 才會是 True。否則就是 False。
所以我們有:
True and True : True
True and False: False
False and True: False
False and False: False

我們以下面例子說明:
a = 8
(0<a) and (a<10)的結果會是 True
(a<10) and (a 是奇數)的結果則是 False
如果 Boolean expression 是:

(a 是奇數) and (a 是偶數),不論 a 的值是甚麼，只要 a 有定義(有一個叫做 a 的抽屜存在),(a 是奇數) and (a 是偶數)的值都是 False。

3. A or B
or 連接兩個運算子 A 和 B，當 A 和 B 的值都是 False 時,A and B 才會是 False。否則就是 True。
所以我們有:
True and True : True
True and False: True
False and True: True
False and False: False

我們以下面例子說明:
a = 8
(0<a) or (a<10)的結果會是 True
(a<10) or (a 是奇數)的結果則是 True
(a>10) or (a 是奇數)的結果則是 True
如果 Boolean expression 是:
(a 是奇數) or (a 是偶數)，只要 a 有定義，(a 是奇數) or (a 是偶數)的值都是 True。

練習:說明以下 Boolean Expression 的值
1. True and False
2. True or False
3. (True and False) or False
4. True or (False or (False or False))
5. False and (True and (True or (False and False)))
6. -1 < 1
7. 2015 - 1961 > 50
8. (9 < 8) and (9 < 10)
9. (9 < 8) or (9 < 10)
10. (10 是奇數) or (10 是偶數)

思考:
2 乘以自己十次(2x2 算兩次),和 10 乘自己兩次，哪一個比較大?可否將此問題寫成一個 Boolean Expression，經由 python 程式去驗算你的答案是對的?

答:
```
a = 2*2*2*2*2*2*2*2*2*2
b = 10 x 10
print(a>b)
```

在程式中，你先計算 a 是把 2 乘以自己十次，b 是把 10 乘以自己兩次,然後 Boolean Expression 兩次，然後假設 a>b，問 python 是否如此。當 python 回答 True 時表示你的假設是對的，也就是二的十次方大於十的二次方。若 python 回答 False,則表示十的二次方大於等於二的十次方。答案是哪一個，就進入 python 試試。

Print Statement

目前我們介紹了三種不同的資料，包括數、字串和 Boolean。如何將這些結果印出來呢?在 python 中採用 print statement 來列印結果。舉例來說，print(3)印出數字 3,print("3")印出字串"3"。print(True)印出 True。同樣的，我們也可以印出變數，我們用不同的範例來介紹所印出的值。

範例一:印出變數:

```
a=3
b="13"
print(a)
print(b)
```

此程式印出
3
13
注意到字串並不會將"的符號印出來。

範例二:印出描述(expression)的結果
```
print(5>3)
```
此程式印出
```
True
```

範例三:印出字串
```
print("Beauty is my name")
```
此程式印出
```
Beauty is my name
```

print 的參數可以不只一個，也就是我們可以同時列印一個以上的數在同一列。但是每一個 print 結束後都會跳行。我們多舉幾個範例。

範例四:字串和變數共用
```
a=3
b=2
print("a+b=", a+b)
```

此程式會印出:
```
a+b=5
```
在 print 中有兩個參數,第一個參數是字串"a+b"，第二個參數是一個描述，我們須先將他的值算出來,也就是 3+2=5,所以在列印時,會印出 a+b=5。

範例五:字串與變數共用，變數的值是字串。
```
name = Mary
print("My name is ", name, ".")
```

此程式會印出:
```
My name is Mary.
```

所以，print(a,b,c,d,....)會先判斷每一個的資料型態，從頭到尾一一列印。當它是字串時，將字串內的自印出，當它是變數時，將變數的值印出，儘可能全部列印在同一列上面。

練習:

1. 使用 Python，將以下資料列印出來:

(a) False
(b) "Hello World"

(c) "98765a321"
(d) 7+8

2. 請討論以下兩程式是否相同:
程式 A:
```
print("My name is ", "Mary")
```

程式 B:
```
print("My name is")
print("Mary")
```

if … then … else 介紹 (2015/11/22)

if 的語法如下:
```
if cond:
    statement
```
來表達，通常 cond(conidition)為一個 Boolean Expression，表某一種結果為 True(是、真、正確)或 False(否、偽、錯誤)的情況。
注意到 statement 在程式中並沒有和 if 對齊，而是向右空了一些空間。這空間不可缺少，容後再述。

當 cond = True 時，就會執行 statement 指令。否則,就會跳過 statement 指令,繼續往下執行。

舉例來說，我們有以下兩個例子:

```
cond = True
if cond:
    print(1)
print(2)
```

在此程式中,因為抽屜cond中存放的值是True，所以程式會先印出1,然後跳出if statement後，解下來印出2。若我們將一開始的cond值設定為False，或者直接寫 if False，如下:

```
if False:
    print(1)
print(2)
```

這個程式會因為 cond=False，所以不執行print(1)，整個程式就只有印出2。

我們簡單介紹一個程式:

程式A:給定兩個數,請由小到大將它印出,如下:

```
a=3
b=2
if a>b:
    print(b, a)
if a<=b:
    print(a, b)
```

我們可以改變a和b的值，然後再從新跑程式，由小印到大。

接下來我們要介紹if的延伸,也就是加入了else，它的語法是:

```
if cond:
    Statement1
else:
    Statement2
```

這指令是說當cond = True時,執行Statement1，否則執行Statement。現在，程式A可以修定為:

```
a=3
b=2
if a>b:
    print(b, a)
else:
    print(a, b)
```

練習: 給定三個數，將這三個述有小到大列印出來。

while 介紹 (2015/11/15)

在表達while之前，我們給一個趣味問題供大家思考。

給大家兩個思考題，這可以是一個程式設計很好的題目。
(1)一瓶可樂一元，兩個可樂空瓶可以換一瓶可樂。書蟲小美有十元，請問最多可以喝到幾瓶可樂?
(2)一瓶可樂一元，兩個可樂空瓶可以換一瓶可樂，四個空瓶也可以換一瓶可樂。書蟲小美有十元，請問最多可以喝到幾瓶可樂?

我們分三部分來討論這個問題。第一部份說明思考的過程,第二部分介紹一個程式的指令,叫做while，第三部分則是連接思想與程式,將整個結果擴充到可解隨意給一個數,請電腦算出可喝多少瓶可樂。

1. 思考過程

思考過程中，我們需要找到解題的 pattern。在這個題目中，pattern 應該是:

買可樂、喝光可樂、空瓶換可樂、喝光可樂、空瓶換可樂…….。但是我們必須保證這個過程會結束，通常我們結束在沒有辦法再換可樂為止，也就是空瓶數小於二。

也就是我們將 pattern 改成:

買可樂

當空瓶數可以換可樂時，空瓶換可樂並喝光。重複此步驟直到無法換可樂為止。

2. While 語法

在程式中，我們有一個指令叫做 while，它的語法如下:

```
While C:
    Do Something
```

這指令是說當 C 是 True(是、真、正確)的時候，就執行”Do Something”

為了要能回答總共喝多少瓶，我們要有一些 bookkeeping 的動作。加上這些動作，剛才的 pattern 可以寫成:

```
While 空瓶數大於等於 2:
    空瓶換可樂,，喝光可樂，收集空瓶，記錄一下已經喝多少了。
```

在 while 裡面，做四件事，這四件事做完了，就會檢查空瓶是否大於等於 2，如果答案是肯定的，就再做一次這四件事，如此一直重複。每一次稱為一個 iteration。

在這個例子中，我們的思考模式(加上 bookkeeping)如下:

一開始可買十瓶可樂，喝光可樂後收集到十個空瓶，記錄喝十瓶。

第一個 iteration: 可以換五瓶可樂，喝光可樂後收集到五個空瓶，記錄共喝十五瓶。

第二個 iteration: 可以換兩瓶可樂,剩下一個空瓶,喝光可樂後收集到三個空瓶,記錄共喝十七瓶。

第三個 iteration: 可以換一瓶可樂,剩下一個空瓶,喝光可樂後收集到兩個空瓶,記錄共喝十八瓶。

第四個 iteration: 可以換一瓶可樂，沒剩下空瓶，喝光可樂後剩下一個空瓶，記錄共喝十九瓶。

在第四個 iteration 後,空瓶數為一,使得”空瓶數大於等於二”這件事變成 False(否、偽、錯誤)，因此，整個 while 就結束了。這時我們知道去換了四次可樂，共喝了十九瓶，還剩下一個空瓶。

3. 程式

在 Python 程式中使用#來代表接下來的字到跳行為止是給人看得(不給人看給誰看?)，別忘了，程式部分是給電腦看得。我們使用#來加強程式的閱讀性。所以:

```
cola = 10        # 紀錄可樂數
```

是告訴電腦給一個變數叫做 cola，裡面存入數值 10。同時，提醒人 cola 紀錄的是可樂數。

我們在程式中需要一些變數來記錄數字，以此程式為例，我們用 cola, rem, empty, total 來記錄可換的可樂數，剩下空瓶數，收集到的空瓶數，已經喝掉的瓶數，程式就可以是:

```
cola = 10              #買可樂
total = cola               #記錄喝幾瓶
empty = cola           #喝光可樂，可樂數變空瓶數
while empty >= 2:          #當可樂瓶子數可以換可樂時
    cola = empty // 2    #換可樂, //是整數除法，也就是算到整數為止。如 7//3=2, 8//3=2.
    rem = empty%2    #剩下的空瓶數, a%b 是表示求 a 除以 b 的餘數。
    total = total + cola  #喝光可樂，記錄喝了幾瓶
    empty = cola           #喝光可樂後，可樂數變成空瓶數
    empty = empty + rem #收集空瓶，現在的空瓶數加上換可樂時的剩餘空瓶
    # 在這裡可以加一些指令，將你的過程印出來，例如:
    print(“換”, cola, “瓶可樂, 剩下”, rem, “個空瓶, 收集到”, empty, “個空瓶，總共喝了”,total”, 瓶)
```

這個程式的執行結果，如下:

```
>>>
換 5 瓶可樂, 剩下 0 個空瓶, 收集到 5 個空瓶, 總共喝了 15 瓶
換 2 瓶可樂, 剩下 1 個空瓶, 收集到 3 個空瓶, 總共喝了 17 瓶
換 1 瓶可樂, 剩下 1 個空瓶, 收集到 2 個空瓶, 總共喝了 18 瓶
換 1 瓶可樂, 剩下 0 個空瓶, 收集到 1 個空瓶, 總共喝了 19 瓶
>>>
```

此程式也可以用另一個不同的概念來解，程式如下:

```
def cola(n, rem):
    empty = n + rem
    if empty < 2:
        return n
    else:
        return n + cola(empty // 2, empty % 2)
```

這個程式使用到 recursive 的概念,也就是定義 cola 的同時,也會用"較簡單的版本"呼叫 cola 自己。在這種呼叫方式中，我們並沒有使用到 while。同時，此程式用函數定義(function definition)等

方法，還需要程式呼叫(function call)來使用它，以此題為例，print(cola(10,0))就是呼叫該程式計算並印出 10 元可以喝多少可樂的呼叫方式。這些方法與概念，容中級的 python 介紹再述。

現在我們將問題擴充為:
(2)一瓶可樂一元，兩個可樂空瓶可以換一瓶可樂，四個空瓶也可以換一瓶可樂。書蟲小美有十元，請問最多可以喝到幾瓶可樂?

有的人在知道第一題的答案，10 元可以喝多少瓶可樂，得到了 19 瓶，這 19 瓶用蓋子可以換 4 瓶剩三個空蓋，這新的四瓶又產生四個空蓋換一瓶，這一瓶的蓋子加上第一次換所剩下的三空蓋就再換一瓶，因此總共換了 4+1+1 瓶。所以用蓋子換了 6 瓶，加上用瓶子換得的 19 瓶，總共就有 25 瓶，請問此想法有何問題?

注意到這個解法沒有思考到用蓋子換得可樂，其空瓶也可以換可樂。所以，我們必須在蓋子換可樂和空瓶換可樂之間游離，兩者都要考慮到。但是，如果我們考慮修改程式，我們將所有當初考慮空瓶的因素和考慮空蓋的因素同時考量，也就是與有空瓶相關的變數，我們也增加蓋子相關的變數，程式就變成:

```
cola = 10        #買可樂
total = cola          #記錄喝幾瓶
cap = empty = cola  #喝光可樂，瓶蓋數=空瓶數= 剛換的可樂數
print("一開始可買", cola)
while empty >= 2 or cap>=4: #當可樂瓶子或瓶蓋可以換可樂時
    cola = empty//2 + cap//4     #換可樂, //是整數除法，也就是算到整數為止。如 7//3=2, 8//3=2.
    rem = empty%2    #剩下的空瓶數, a%b 是表示求 a 除以 b 的餘數。
    remc = cap%4          #剩下的空蓋數
    cap = empty = cola  #喝光可樂，可樂數變成空瓶數
    total = total + cola#記錄喝了幾瓶
    empty = empty + rem #收集空瓶，現在喝的可樂數(empty)加上換可樂時的剩餘空瓶
    cap = cap + remc     #收集空蓋
    # 在這裡可以加一些指令，將你的過程印出來，例如:
    print("換", cola, "瓶可樂, 剩下", rem, remc,"個空瓶和瓶蓋, 收集到", empty,cap, "個空瓶及瓶蓋，總共喝了",total," 瓶")
```

我們在程式中對空瓶的兩個變數 empty, rem 增加了對空蓋的兩個變數 cap, remc，並在相對應的地方增加了對蓋子的計算(以上畫底線部分)，就可以產生答案。注意到在程式中，每一個 iteration 都考慮空瓶與空蓋，有別於考慮完瓶子再考慮蓋子，然後再考慮瓶子、蓋子這種模式。以上的程式得到的結果如下:

```
一開始可買 10
換 7 瓶可樂, 剩下 0 2 個空瓶和瓶蓋, 收集到 7 9 個空瓶及瓶蓋, 總共喝了 17 瓶
換 5 瓶可樂, 剩下 1 1 個空瓶和瓶蓋, 收集到 6 6 個空瓶及瓶蓋, 總共喝了 22 瓶
換 4 瓶可樂, 剩下 0 2 個空瓶和瓶蓋, 收集到 4 6 個空瓶及瓶蓋, 總共喝了 26 瓶
換 3 瓶可樂, 剩下 0 2 個空瓶和瓶蓋, 收集到 3 5 個空瓶及瓶蓋, 總共喝了 29 瓶
換 2 瓶可樂, 剩下 1 1 個空瓶和瓶蓋, 收集到 3 3 個空瓶及瓶蓋, 總共喝了 31 瓶
換 1 瓶可樂, 剩下 1 3 個空瓶和瓶蓋, 收集到 2 4 個空瓶及瓶蓋, 總共喝了 32 瓶
換 2 瓶可樂, 剩下 0 0 個空瓶和瓶蓋, 收集到 2 2 個空瓶及瓶蓋, 總共喝了 34 瓶
換 1 瓶可樂, 剩下 0 2 個空瓶和瓶蓋, 收集到 1 3 個空瓶及瓶蓋, 總共喝了 35 瓶
>>>
```

因此，此題中 10 元可以喝到 35 瓶可樂。

再第一例(考慮空瓶，不考慮蓋子)中，當我們要考慮另一個數的時候，例如 20 元可以喝多少可樂，只要將第一列的 cola = 10 改成 cola = 25 就可以有答案了，如下:

```
>>>
換 12 瓶可樂, 剩下 1 個空瓶, 收集到 13 個空瓶, 總共喝了 37 瓶
換 6 瓶可樂, 剩下 1 個空瓶, 收集到 7 個空瓶, 總共喝了 43 瓶
換 3 瓶可樂, 剩下 1 個空瓶, 收集到 4 個空瓶, 總共喝了 46 瓶
換 2 瓶可樂, 剩下 0 個空瓶, 收集到 2 個空瓶, 總共喝了 48 瓶
換 1 瓶可樂, 剩下 0 個空瓶, 收集到 1 個空瓶, 總共喝了 49 瓶
>>>
```

注意到程式協助了快速計算的部分，只要數字一換(從 10 換成 25)，人類重算的速度就遠遠比不上程式了(只要你寫過一次)。因此，有人問，我都已經知道怎麼解了，為什麼還要學寫程式?就好像問:”我都知道怎麼跑很快了，為什麼還要學開車”一樣，肌肉無論如何發達，都無法對抗機械力量，腦力無論如何聰明，都無法趕上電腦的計算與記憶。無法使用新的技術解題，就像無法使用新設備(汽車、智慧手機、網路等)過生活，在各方面都會受到局限的。

學習程式的過程，學習思考方式遠遠比學會寫程式本身重要。應該說，程式本身可以驗證學習者的思維是否正確，而計算式的思考，可以協助學習者較容易與程式設計接軌。

在 while 的使用上，我們常常搭配一個變化來測試自然樹是否滿足某些條件。舉例來說，有名的

練習：一筐雞蛋，1 個 1 個拿正好拿完，2 個 2 個拿還剩 1 個，3 個 3 個拿還剩 1 個，4 個 4 個拿還剩 1 個，5 個 5 個拿還剩 1 個，6 個 6 個拿還剩 1 個，7 個 7 個拿正好拿完，問筐裡最少有多少雞蛋？

range 介紹

range 的參數可以有一個、兩個、或三個，也就是 range(a), range(a, b)和 range(a, b, c)。分述如下

1. range(10)是一個由 0 開始數到 9 的整數數列(排成一列的整數),注意到此數列不包括 10。我們寫作:range(10)=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)。所以 range(3)=(0, 1, 2), range(5)=(0, 1, 2, 3, 4)。
2. range(2, 6)是一個由 0 開始數到 9 的整數數

列（排成一列的整數）。也就是 range(2,6)=(2,3,4,5)。range(-5, 5)=(-5, -4, -3, -2, -1, 0, 1,2,3,4)

3. range(1, 10, 2)是一個由 1 開始，以 2 為公差，數到小於 10，但下一數大於等於十的整數數列（排成一列的整數），也就是 range(1,10,2)=(1,3,5,7,9)，注意到下一個數 11 已經大於 10。也注意到 range(1,10,2)=range(1, 11,2)。此數列可以往下數，也就是 range(10, 1, -2)=(10, 8, 6,4,2)。

整體而言，range(a, b, c)是指給訂上界 a 和下界 b，以及公差 c(c>0)，找一串的數列，此數列從下界 a 開始，找出他的等差級數 a, a+c, a+2c, ...a+nc，此等差級數的最後一數 a+nc 仍小於上界(a+nc<b), 但下一數大於等於上界(a+(n+1)c>b)為止。通常，下界的預設值為 0，公差為 1。當公差 c<0，我們數列的數越來越小，最後一數要仍大於 c，但下一數小於等於 c 為止。

作業: 寫出以下數列的範圍
1. range(5)
2. range(7)
3. range(2,7)
4. range(10, 20)
5. range(-3,3)
6. range(-5, 5, 2)
7. range(5, -5, -2)
8. range(10, 50, 5)

For 介紹

For 的語法如下:

```
for i in lista:
    statement
```

這個指令說，把所有在 lista 內的資料 a，一個一個先執行 i=a，然後執行 statement，直到所有在 lista 裡的資料都用完為止。

舉例來說:

```
#程式 A:
for i in [0,1,2]
    print(i)
```

這程式執行順序則是:

```
#程式 B:
i = 0          第一個資料
print(i)
i = 1
print(i)
i = 2
print(i)
```

這兩個程式印出的結果都是印出 0, 1, 2。

我們也可以利用 range 來產生(0,1,2)，也會產生相同的結果，如以下程式:

程式 C:

```
for i in range(0,3):
    print(i)
```

for 和 while 在程式中都叫 loop(迴圈)，也就是它內部的 statement 會執行多次。每一次叫做一個 iteration。for loop 和 while loop 最大的不同，在於 for loop 在進入之前，就知道有幾個 iteration，而 while loop 通常事先並不一定知道有幾個 iteration，因為它是由緊接著 while 的 cond(condition)之條件來決定是否要多一次的 iteration，當 cond=True 時是，當 cond 變為

False 時則否，因而結束了 while loop。

因為 for 有一點複雜，我們舉幾個例子來說明。和 while 一樣，我們從一個數學問題進入，逐漸介紹 For 的使用。

(1) 一個島上有兩顆基因特異的樹，每年都會各別生出個一棵無法生育的樹。十年以後，島上有幾棵樹?(等差級數)

(2) 一個島上有一棵樹，每一棵樹每年都會各別生出個另一棵樹。十年以後，島上有幾棵樹?(等比級數)

(3) 在一個島上有一棵樹種，每一個數種需要一年才能變為成樹，成樹需要一年才能生小樹種，請問十年後島上有幾棵樹(假設樹都不會死)(Fibonacci 數列)。

(4) 在一個島上有一棵樹種，每一個數種需要兩年才能變為成樹，成樹需要一年才能生小樹種，請問十年後島上有幾棵樹(假設樹都不會死)。

我們分四節來描述以上的四個問題。

(1) 一個島上有兩顆基因特異的樹，每年都會各別生出個一棵無法生育的樹。十年以後，島上有幾棵樹?(等差級數)

基本上這個問題是等差級數的問題。主要的是一開始有兩棵樹，公差(每年增加的樹木數)也是2，第一年生兩棵樹(共四棵)，我們從此(第一年尾)開始記錄，因此第一個數是 4，如此一直到第十年尾(第十個數)則是 22。這一題的程式如下:

程式 D: 等差級數的計算

```
total = 2          #到目前的樹木總數
for i in range(1, 11):    #每年生兩棵，到第十年止
    total = total + 2    # 每一次都加公差 2
print(total)
```

這個程式有三個 statement，第一個 statement 是 assignment，第二個是 for loop，第三個 print statement 則將結果印出來。注意到第三個 statement 和 for 對齊，因此它是 for statement 的下一個 statement。最後這個 print statement 印出 22。一開始有兩棵，第一年有四棵，第二年有六棵,....., 到第十年有 22 棵樹。這時，我們結束了 total 的計算，叫做 total 的抽屜存放有一張寫著 22 的紙卡。在我們結束了 for loop 後，就執行下一個 print statment，因而印出 22。

如果我們將第三個 print statement 不和 for 對齊，而和 total 對齊，會有甚麼結果，我們看以下的程式:

程式 E: 等差級數的計算

```
total = 2          #到目前的樹木總數
for i in range(1, 11):    #每年生兩棵，到第十年止
    total = total + 2    # 每一次都加公差 2
    print(total)         # 印出到目前的 total
```

python 把這個 print statement 當作是 for loop 內部的 statement。也就是 i=1 時，會執行 total = total + d，然後把 total 印出來，然後在執行 i=2，又執行 total = total + d，然後把 total 印出來，如此反覆在 i=3, i=4, ... i=10 總共十個 iteration，每一個 iteration 都執行一次 print。因此，這程式會印出(total 先加 2 在印出，所以從第二個數開始印)4,6,8,10,12,14,16,18,20,22(每個數字間會跳行)，各別在第一個到第十個 iteration 時印出。

(2) 一個島上有一棵樹，每一棵樹每年都會各別生出個另一棵樹。十年以後，島上有幾棵樹?(等比級數)

島上的在第一年尾會有一棵樹生一棵樹，共兩棵

(1x2)，在第二年尾會有兩棵樹生兩棵樹，因此有四棵(1x2x2)。在第三年尾有四棵樹生四棵樹，因此有八棵(1x2x2x2)。如此反覆，我們知道十年(尾)後會有 1x2x2x2x2x2x2x2x2x2x2 棵樹。每年數的成長以等比級數成長，公比為 2。

程式 F: 等比級數的計算

```
total = 1          #到目前的樹木總數
for i in range(1, 11):    #每年生兩棵，到第十年止
    total = total * 2    # 每一次都加公比 2
print(total)          # for loop 結束後印出 total
```

最後，這個程式印出 1024。

(3) 在一個島上有一棵樹種，每一個數種需要一年才能變為成樹，成樹需要一年才能生小樹種，請問十年後島上有幾棵樹(假設樹都不會死)(Fibonacci 數列)。

程式 G: Fibonacci 數列

```
baby = 1
adult = 0
for i in range(1, 11):
    baby, adult = adult, baby+adulty
print(baby,adult)
```

一開始我們給定樹種的數量使用變數 young，成樹則使用 adult。一開始我們有一個數種(baby = 1)而沒有成樹(adult = 0)。這時我們進入 10 個 iteration 的 for loop。每一個 iteration，去年的成數就是今年的樹種數，因為每一年的每一棵成樹都會產生一個數種，而今年的成樹就是去年的成數加上今年長大的樹(去年的樹種數)，我們使用 baby, adult = adult, baby+adult 來完成這個動作。在整個 for loop 結束後，我們發現print 列印出baby 和adult 的值各別為34,55，也就是 10 年後島上將有 89 棵樹。

(4) 在一個島上有一棵樹種，每一個數種需要兩年才能變為成樹，成樹需要一年才能生小樹種，請問十年後島上有幾棵樹(假設樹都不會死)。

```
baby = 1
teen = 0
adult = 0
for i in range(1, 11):
    baby,teen,adult=adult,          baby,
teen+adult
print(baby,teen,adult)
```

此程式最後印出 9,6,13。也就是十年後，島上有 9 個樹種，6 個一年大的樹苗，13 棵成樹，共 28 棵樹。

目前為止，我們的 for loop 都介紹我們在 loop 內部要執行幾次。在 loop 裡面，我們並沒有使用到變數 i。接下來我們來介紹 for loop 中的變數使用。同樣的，我們使用題目的計算來介紹變數在 loop 中的使用。

題目:從 1 加到 n 的程式如何撰寫。

我們回憶在講述 assignment 時的程式:

```
i = 0 = total = 0
i = i + 1
total = total + i
i = i + 1
total = total + i
i = i + 1
total = total + i
i = i + 1
total = total + i
i = i + 1
total = total + i
print(total)
```

在這個程式中，一開始 i=total = 0。我們將 i 加 1 使 i 變成 1，然後加到 total 裡面，此時 total 存了 0+1，接著我們再將 i+1，使 i 變成 2，然後

加到 total 裡面，使 i 變成 0+1+2，經過如此的反覆五次，我們得到 total=0+1+2+3+4+5。因此 total=15。

現在，我們學了 for，for 會每次將後面一個 list 的元素帶入 for 的變數中，然後一一執行 for loop 內部的 statement。因次，我們可以以上的程式改成:

```
total = 0
for i in (1,2,3,4,5)
    total = total + i
print(total)
```

這麼一改，程式變得更短。但是，如果我們要計算從 1 加到 1000，那要寫很長的一串由 1 到 1000 的數列，不過我們可以用 range 來代表這個數列，因此，程式可以改成:

```
total = 0
for i in range(1, 1001):
    total = total + i
print(total)
```

注意到，range(1,1001)是包括 1 但不包括 1001 之間的整數。我們將程式這麼一改，程式就有了擴充性，也就是我們只要改 range，就可以計算任何數之間的和。舉例來說，將 range 改為 range(101, 201)就可以計算從 100 加到 200 的和。如果要計算 1000 到 2000 之間的奇數之總和，我們就可以將 range 改為 range(1000, 2001, 2)。

for loop 也可以有兩層或兩層以上，以九九乘法表為例，兩層的 for 如下:

```
for i in range(1,10):
    for j in range(1,10):
        print(i, "x", j, "=", i*j)
```

但是，兩層的 for loop 必較不容易懂，我們將在未來介紹。

作業:

1. 如何修改程式 G，使每一年的樹木數，包括數種數和成樹數都能列印出來。
2. 如何計算 1 到 100 的和?
3. 如何計算 1 到 100 的奇數總和?偶數總和?
4. 可否在一個程式中計算以上兩題，並驗算基數總和加上偶數總和會等於所有數的總和。程式應該印出四個資料，第一個是印出 1 到 100 的總和，第二和三個是印出 1 到 100 的積數總和和偶數總和， 最後一個是印出結果是否相等(True 或 False)。

5.
如何撰寫程式計算 1x2x3x4x5?

6. 如何撰寫程式計算 1x2x3x4x.....x50

Function 介紹

函數(Function)的介紹分兩部分，第一部分是定義(function definition)，第二部分則是使用(function call)。

函數的定義分三部分，第一部分是名稱(name)，第二部分是參數(parameter)，第三部分是傳回的值(return value)，它的語法如下:

```
def name(para1, para2, ...., paran):
    function body
```

先看第一列

```
def name(para1, para2, ...., paran):
```

這一列包含了函數名稱(function name)和參數(parameter)，參數的數量可以是零個、一個、

兩個、甚至 n 個。至於第二列的 function body 則是在計算此 function 的結果，然後將結果用 return statement 傳回給呼叫此 function 的指令。

舉例來說，我們給定一個函數，它的功能是將兩個數的較大數找出，因此我們可以定義此函數 max 如下:

```
def max(a,b):
    if a>b:
        return a
    else:
        return b
```

def 表示我們接著在定義函數，此函數的名稱叫做 max，有兩個參數 a 和 b。在此函數的 function body 中只有一個 if statement。在這個 if statement 中，有一個 return statement，它會將 return 後的描述(expression)的值計算出來，然後當作成此 function 的回傳值，回傳給呼叫(call)它的程式碼。當在執行 function body 時，只要一碰到 return statement，這程式就結束執行，回到原來呼叫它的程式。

說完 function definition，就要提如何呼叫。其實，呼叫很簡單，只要名稱相同，參數數量相同，且所要求的參數型態也符合就可以了。舉例來說，以下程式在呼叫剛定義的 max 程式:

```
c = max(5,8)
print(c)
```

這個程式一開始就呼叫函數 max，這時，參數的對應先讓 a=5, b=8，然後去計算函數的 function body，也就是:

```
if a>b:
    return a
else:
    return b
```

經計算結果執行 return b，也就是將 8 傳回給呼叫它的 max(5,8)。此時，繼續執行 c = max(5,8)，也就是將 8 放到叫做 c 的抽屜(抽屜不存在則建立新抽屜)。最後一個 print 指令則將結果印出。

我們介紹五個不同的例子來說明 function 的應用:

(1) 給定兩個數 a, b，請算出它的平均數 (a+b)/2。

(2) 給定一個數，是回答此數是否是奇數(True or False)。

(3) 給定三個數，請將最大數找出來。

(4) 計算從 1 到 n 的總和

(5) 給定一個正整數 n，請寫一程式幫忙回答 n 是否是質值數(True of False)。

我們將此五個程式的定義與使用一一明述於後。

(1) 給定兩個數 a, b，請算出它的平均數 (a+b)/2。

```
def average(a,b):
    return (a+b)/2
```

這個函數很簡單，直接傳回(return)計算的平均質即可。要如何測試你的定義(function definition)對不對，我們可以寫一個主程式來測試，這個主程式給予 a,b 的值，然後將結果列印出來，如下:

```
a = 8
b = 10
print("The average of ", a, " and ", b, "is", average(a,b))
```

這個主程式將會印出:

```
The average of 8 and 10 is 9.0
```

(2) 給定一個數，是回答此數是否是奇數(True or False)。

奇數的判斷，我們可以使用此數除以二的餘數來判斷，當餘數是 1(n//2==1)則是奇數，否則就是偶數。因此，這函數的定義如下:

```
def odd(n):
    if n//2 == 1:
        return True
    else:
        return False
```

這程式也可以寫成:

```
def odd(n):
    if n//2 == 1:
        return True
    return False
```

可以寫成如此主要是因為當 n//2==1 時，兩個程式都會 return True，但當 n//2==0 時，地一個程式走 else:，所以 return False，第二個程式因為 if 已經結束了,所以執行 if 的下一個指令，也是 return False。

如果想測試 3 是否為奇數，在測試此程式的主程式如果寫:

```
odd(3)
```

python 會將所算的結果 True 傳回給 odd(3)，但螢幕上沒有任何的結果。但是我們如果寫:

```
print(odd(3))
```

python 在將結果 True 傳回給 odd(3)時，print 指令會將結果印出來，所以銀幕上才會出現 True。

(3) 給定三個數，請將最大數找出來。

```
def max3(a,b,c):
    maxnumber = a
    if b>maxnumber:
        max = b
    if c>maxnumber:
        max = c
    return maxnumber
```

此程式一開始將 a 的值存在一個叫做 maxnumber 的抽屜裡，然後將此抽屜內的值與 b 比較，若 b 更大，則將 b 的值存到 maxnumber 裏，也就是目前比到 b，最大的數是 maxnumber。重複此計算跟 c 比，將 maxnumber 和 c 的，所得的較大數放到 maxnumber 裡，並將所求得的值傳回。

我們曾經定義 max(a,b)的函數，其實也可以使用該函數，所以程式可以變成:

```
def max3(a,b,c):
    maxnumber = max(a,b)
    maxnumber = max(maxnumber, c)
    return maxnumber
```

注意到一個函數可以被另一個函數來使用。但要使用前，一定要先有定義。

相同的，要測試此函數是否正確，不要忘了用 print 將結果印出。所以:

```
print(max3(8,3,9))
```

將會印出 9。

```
print(max3(8,3,4))
```

則會印出 4。

(4) 計算從 1 到 n 的總和

```
def sum(n):
    total = 0
    for i in range(1, n+1):
        total = total + i
    return total
```

我們曾經說明，如果我們要計算從 1 加到 1000，我們可以用 range 來代表要倍加的這個數所產生的數列，因此，程式可以是:

```
total = 0
for i in range(1, 1001):
    total = total + i
```

```
print(total)
```

在此程式的定義(function definition)中，我們將 1000 當作參數 n，然後由 range(1,n+1)先產生一個由 1 到 n 的數列，最後經由 total = total + i 逐一地將數列中的數都加總到 total 裡。執行完後，所計算出的 total 值就是我們函數的回傳值。

至於呼叫此函數，我們可以使用:

```
m = 100
print("The sum from 1 to ", m, " = ", sum(m))
```

(5) 給定一個正整數 n，請寫一程式幫忙回答 n 是否是質數(True of False)。

質數的定義在於除了 1 和自己以外，沒有其他的因數，也就是比它小的正整數中，沒有一個數是它可以整除的。我們如何判斷"整除"呢?所謂整除，就是相除以後的餘數為零。因此 9%3==0 和 9%4==1 表示 9 可以整除三，但無法整除四。

所以，我們用以下的函數定義來檢查某一個數是否為質數。

```
def prime(n):
    for i in range(2, n):
        if n%i == 0:
            return False
    return True
```

在這個程式中，我們只要找到任何一個介於 2 到 n-1 間的數可以被 n 整除(n%i==0)，馬上就結束程式，回傳 False。當 for loop 整個結束，也就是找不到任何一個數字界於 2 到 n-1 可以被 n 整除，此時我們就回傳 True。

我們可以使用以下程式來測試此函數定義是否正確:

```
m = 13
if prime(m):
    print(m, "is a prime number")
else:
    print(m, "is not a prime number")
```

基本上，一個數無法真正測試出含數定義是否正確，我們需要多一點的資料來確認函數定義的正確性。我們使用以下程式來測試 2 到 100 間的所有質數:

```
for i in range(2, 100):
    if prime(i):
        print(i, "is a prime number")
```

此程式的執行結果如下:

```
2 is a prime number
3 is a prime number
5 is a prime number
7 is a prime number
11 is a prime number
13 is a prime number
17 is a prime number
19 is a prime number
23 is a prime number
29 is a prime number
31 is a prime number
37 is a prime number
41 is a prime number
43 is a prime number
47 is a prime number
53 is a prime number
59 is a prime number
61 is a prime number
67 is a prime number
71 is a prime number
73 is a prime number
79 is a prime number
83 is a prime number
89 is a prime number
97 is a prime number
>>>
```

通常，專業軟體的寫法有防呆的機制，所謂防呆，就是要防止不會使用的人來用軟體的時候發生狀況，我們軟體的撰寫需要告知"呆子"它的錯誤。以此題為例，目前為止我們都假設數字是整數，但在考慮 prime 這個函數時，我們要的是大於等於 2 的正整數。因此，當有人輸入 prime(-2)時，我們要告訴他說它的參數值是錯誤的。我們可以

將資料的測試放在主程式，因此多寫了一個 main(n)，如下:

```
def main(n):
    if type(n)!=int or n<2:
        print(n, type(n), "is a wrong parameter")
    else:
        if prime(n):
            print(n, "is a prime number")
        else:
            print(n, "is not a prime number")
```

注意到在此函數中我們使用到一個函數 type，它會告訴我們資料的型態，是整數(int)、字串(str)、資料串(list)、還是 boolean(bool)。

我們測試四個不同的資料在程式 main 中，如下:

```
main(3)
main(20)
main(-2)
main("3")
```

或者可以寫成:

```
for i in [3, 20, -2, "3"]:
    main(i)
```

得到以下的結果:

```
3 is a prime number
20 is not a prime number
-2 <class 'int'> is a wrong parameter
3 <class 'str'> is a wrong parameter
>>>
```

注意到列印出來的第一列和最後一列，第一列輸入的 3 是整數，最後一列則是長度為 1 的字串"3"。因此，輸出時特別說<class 'str'>，來表明此 3 不是數字，而是字串。

雖然防呆的機制是專業軟體的重要項目，在學程式的初期，了解語法、學會使用、邏輯訓練、懂得除錯等項目遠比防呆重要，因此，我們將專注在這些項目，暫時忽略防呆的機制，以免混淆學習的重點。

這一節的最後面，我們再介紹一個範例:
今年(2015)網路上有一個遊戲，如下:
先將你手機的最後一個號碼乘以 2，然後加 5，結果乘以 50，再加上 1765，最後減掉你出生的西元年，得到一個數，這數的最後兩位數是您的年紀，第一位數(百位數)則是你的手機末碼。若是此數為兩位數，表示手機的末碼為零。

注意到者裡面有兩個變數，一個是您手機的最後一碼，另一個是您的出生西元年。

假設你不知道如何以數學方式分析此題。試寫一個函數利用此計算順序來回傳計算值。同時，假定你是 1961 年出生的，寫一個主程式驗算當手機末碼為 0 到 9，各別產生的結果為何?觀察第一數是否與手機末碼相同?

此函數可以定義為:

```
def compute(num, year):
    return (num*2+5)*50+1765-year
```

數學分析:事實上，(num*2+5)*50+1765-year 可以簡化為 100*num+(2015-year)，100*num 使得各位數 num 出現在百位數，而 2015-year 則表達你的二位數的年紀。注意到出題者假設了來答題的都不會超過 100 歲。

假設我們不知道以上的數學分析，但想了解是否手機的末碼在十個可能性(0 到 9)中，所求的百位數都與手機末碼相同，我們的主程式可以試(假設我是 1961 年出生的):

```
year = 1961
for i in range(10):
    print("末碼 = ", i, "計算後 = ", compute(i, year)
```

程式的執行結果如下:

```
末碼 =  0 計算後 =  54
末碼 =  1 計算後 =  154
末碼 =  2 計算後 =  254
末碼 =  3 計算後 =  354
末碼 =  4 計算後 =  454
末碼 =  5 計算後 =  554
末碼 =  6 計算後 =  654
末碼 =  7 計算後 =  754
末碼 =  8 計算後 =  854
末碼 =  9 計算後 =  954
>>>
```

注意到末碼和計算後數字的百位數相同，而計算後數字的最後兩位數是我的年紀。

練習:

1. 寫一個函數 absnumber 來計算一整數的絕對值。舉例說明，|3|=|-3|=3。事實上，有一個函數 abs 在執行此結果，我們期望初學者自己設計此程式。

2. 請寫一函數計算 n!=1x2x3x.....xn，並寫一主程式來測試 2!到 10!的答案是對的。

3. 還記得小時候有這麼一個遊戲:
請你想一個數
乘以 2
加 30
除以 2
減掉你剛才想的數
最後答案會等於 15

好神啊!可否用數學分析和程式解碼兩種方法來說明為什麼神。數學分析可以分析到所有的數，程式解碼呢?假使你想的數是 i，i 從 1 是到 100，看是否所有的結果都是 15?

List 介紹

目前我們介紹了三種不同的資料型態，就是整數、boolean和字串。現在再介紹一個新的資料型態，叫做 List。我們可以稱它做資料列。資料列裡面儲存著一連串的資料，python 使用中括號來代表資料列。因此，[1,2,3,4,5]代表一個資料列存五個資料，各別是 1,2,3,4,5，而["John", "Mary", "Alice"]則代表一個資料列，儲存了"John", "Mary"和"Alice"三個字串。

是時尚資料列不一定指儲存相同資料形式的資料。舉例來說，資料列也可以代表三個不同的資料，例如第一個資料代表名字，第二個代表年紀，第三個則是結婚狀況，因此["John", 29, True]可以帶表 John 這個人，29 歲，已婚，而["Mary", 22, False]則可代表 Mary 這個人，22 歲，未婚。

在我們前面所教的 function 中，它的參數值和 return 值通常是一個數，現在我們可以使用一個數列來傳不同數量的參數，也可以 return 一個以上的數，只要把它放在一個 List 中就可以了。舉例來說，假設 L=[5,3,2,8,7]，我們可以寫一個排序的函數 sort，將這個資料列 L 排序，回傳值為[2,3,5,7,8]。

接下來，我們介紹兩個不同的函數定義，第一種是回傳值為資料列，第二種則是參數為資料列。參數與回傳都是資料列的函數，如以上所說的 sort 函數，將會在進階的教學中在提及。

我們先舉兩個簡單的例子。如下:
(1) 寫一個 sort2(a,b)的函數，將傳入的兩個數 a, b 由小排到大放在一個資料列 L 中，然後回傳 L。
(2) 寫一個 sort3(a,b,c)的函數，將傳入的三個

數 a,b,c 從小排到大放在一個資料列 L 中，然後回傳 L。

(1) 寫一個 sort2(a,b)的函數，將傳入的兩個數 a, b 由小排到大放在一個資料列 L 中，然後回傳 L。

函數如下:

函數 A: 將兩數排序

```
def sort2(a,b):
    if a<b:
        return [a,b]
    else:
        return [b,a]
```

(2) 寫一個 sort3(a,b,c)的函數，將傳入的三個數 a,b,c 從小排到大放在一個資料列 L 中，然後回傳 L。

函數 B: 將三個數排序

```
def sort3(a,b,c):
    if b>a:
        a,b = b,a    # 目前保證 a<=b
    if c<a:
        return [c,a,b]
    if c>b:
        return [a,b,c]
    return [a,c,b]
```

此函數在第一個 if statement 後保證 a<=b，之後就測試 c 在哪一個範圍，是比 a 小(第二個 if statement)，還是比 b 大(第三個 if statement)，或者是介於 a,b 之間(都不再第二個和第三個 if 所測的範圍內)。

我們要如何取得資料列內的資料?python 將資料列的資料從 0 開始，逐次往後編號，我們使用以下格式來取得編號 i 的一個資料，它的格式是: L[i]

舉例來說，L=[5,3,2,8,7]，則 L[0]就是 5，L[4]就是 7。

另外，我們想要知道資料列的長度，可以使用一個內建的函數叫做 len，也就是當 L=[5,3,2,8,7]時，len(L) = 5。

有了取資料的方式以及資料列長度的計算，我們要撰寫兩個函數定義:

(1) 給定一個都是整數的資料列，定義一個函數，回傳此資料列的資料是否遞增(True or False)。遞增的定義在於後面的數值大於等於前面的數值。寫測試程式，將 L1=[2,3,6,6,8]和 L2=[2,3,5,8,7]兩個數列測試，其結果應該各別是 True(L1)和 False(L2)。

(2) 給定一個都是整數的資料列，定義一個函數將所有的資料加總傳回。

(1) 給定一個都是整數的資料列，定義一個函數，回傳此資料列的資料是否遞增(True or False)。遞增的定義在於後面的數值大於等於前面的數值。寫測試程式，將 L1=[2,3,6,6,8]和 L2=[2,3,5,8,7]兩個數列測試，其結果應該各別是 True(L1)和 False(L2)。

```
def inorder(L):
    for i in range(len(L)-1):
        if L[i]>L[i+1]:
            return False
    return True
```

此函數的 for loop 中將現在指到的數與下一個數比較，當現在比下一個大，表示此資料烈沒有遞增，就直接回傳 False。如果全部都比較完畢，沒有發現有一前一個比後一個大的情況，表示這個資料列的數字是遞增，我們就回傳 True。

注意到 for loop 的範圍是從 0 到 len(L)-1，因為我們要從 0 指到倒數第二個數，如此倒數第二個數才能跟最後一個數比較。

主程式的測試可以是:

```
L1=[2,3,6,6,8]
L2=[2,3,5,8,7]

for L in [L1, L2]:
    if inorder(L1):
        print(L, "is in increasing order")
    else:
        print(L, "is not in increasing
order")
```

主程式中使用 L in [L1, L2]:來將下面的 if statement 執行兩次,第一次用 L1,第二次用 L2。

(2) 給定一個都是整數的資料列,定義一個函數將所有的資料加總傳回。

```
def sum_list(L):
    total = 0
    for i in range(len(L)):
        total = total + L[i]
    return total
```

測試程式可以是:

```
L = [5,3,2,7,8]
print("The sum for the elements in", L, "is",
sum_list(L))
```

注意到參數列的長度為 5。在 sum_list 的 function body 中,它的 for loop 採用的 range(len(L)),也就是由 0 到 4,因此將 0+L[0]+L[1]+L[2]+L[3]+L[4]全部加總到變數 total,最後將 total 的值傳回。

介紹了 function 和 List,我們接著介紹如何使用一層的 for loop 和 function call(另一個 for 在 function 裡面) 印出九九乘法表,。

我們分兩部分完成此事:

(1). 針對一個 1 到 9 的數 n,寫一個函數 onerow 將它從 nx1, nx2, nx3, ...nx9 的結果以字串方式排成一列。

(2). 寫一個 for loop,對 1 到 9 的每一個數,去呼叫函數 onerow,如此完成九九乘法表的程式。

(1). 針對一個 1 到 9 的數 n,寫一個函數 onerow 將它從 nx1, nx2, nx3, ...nx9 的結果以字串方式排成一列。

此函數定義如下:

```
def onerow (n):
    L = ""
    for i in range(2, 10):

L=L+str(n)+"x"+str(i)+"="+str(n*i)+"    "
    return L
```

針對一個數 n,一開始給一個空白的字串,然後使用一個 for loop 將所有的 nxi 的表達方式用一個字串連接在原字串後,也就是用

```
L=L+str(n)+"x"+str(i)+"="+str(n*i)+"    "
```

注意到最後空的空格是希望下一個 nxi 不要靠得太近。計算結束後我們將一列的字串 L 傳回。

(2). 寫一個 for loop,對 1 到 9 的每一個數,去呼叫函數 onerow,如此完成九九乘法表的程式。

```
for i in range(2,10):
    print(onerow(i))
```

此時,我們使用一個 for loop 將所有 1 到 9 的數都呼叫一遍 onerow。

此函數的執行結果,如下:

```
2x2=4   2x3=6   2x4=8   2x5=10   2x6=12   2x7=14   2x8=16   2x9=18
3x2=6   3x3=9   3x4=12   3x5=15   3x6=18   3x7=21   3x8=24   3x9=27
4x2=8   4x3=12   4x4=16   4x5=20   4x6=24   4x7=28   4x8=32   4x9=36
5x2=10   5x3=15   5x4=20   5x5=25   5x6=30   5x7=35   5x8=40   5x9=45
6x2=12   6x3=18   6x4=24   6x5=30   6x6=36   6x7=42   6x8=48   6x9=54
7x2=14   7x3=21   7x4=28   7x5=35   7x6=42   7x7=49   7x8=56   7x9=63
8x2=16   8x3=24   8x4=32   8x5=40   8x6=48   8x7=56   8x8=64   8x9=72
9x2=18   9x3=27   9x4=36   9x5=45   9x6=54   9x7=63   9x8=72   9x9=81
>>>
```

因為程式中的空格都是空三格,但有些兩數相程是一位數,有些是兩位數,所以沒有對齊。我們可以修該程式,將結果都預留兩個位數,舉例來說,2x3=6,6 是一位數,我們就將該字串 str(n*i)

改成" "+str(n*i)，最後就會印出 2x3= 6(6 前多一個空白)。這修改只要在 for 前加一個 if statement 來測量 n*i 是一位數還是兩位數即可(len(str(n*i)==1?)。如此修改，其結果如下:

```
2x2= 4   2x3= 6   2x4= 8   2x5=10   2x6=12   2x7=14   2x8=16   2x9=18
3x2= 6   3x3= 9   3x4=12   3x5=15   3x6=18   3x7=21   3x8=24   3x9=27
4x2= 8   4x3=12   4x4=16   4x5=20   4x6=24   4x7=28   4x8=32   4x9=36
5x2=10   5x3=15   5x4=20   5x5=25   5x6=30   5x7=35   5x8=40   5x9=45
6x2=12   6x3=18   6x4=24   6x5=30   6x6=36   6x7=42   6x8=48   6x9=54
7x2=14   7x3=21   7x4=28   7x5=35   7x6=42   7x7=49   7x8=56   7x9=63
8x2=16   8x3=24   8x4=32   8x5=40   8x6=48   8x7=56   8x8=64   8x9=72
9x2=18   9x3=27   9x4=36   9x5=45   9x6=54   9x7=63   9x8=72   9x9=81
>>>
```

注意到我們雖然在此程式和 onerow 函數都是使用 i 當變數，但此兩個 i 沒有相互影響。在此程式中，當呼叫 onerow 時，會將這裡的 i 值先取出，然後傳給 onerow(n)的參數 n，因此，這裡的 i 並沒有將"抽屜"傳過去，只有把值傳過去。因此，兩邊的變數 i 沒有相互的關係。在此涉及到局部變數(local variable)和全域變數(global variable)，等中階課程在介紹。

練習

趣味程式

我們已經學了一些資料的型態，包括:
Boolean
整數(integer)
字串(string)
資料列(list)

也學了一些程式的指令，包括:
assignment
print
if
while
for

同時，我們也學了如何定義函數以及呼叫函數。有了這些基礎，我們可以玩很多的思考謎題(puzzle)與遊戲(game)。這一節就開始找一些趣味的謎題與遊戲來練習程式。

趣味程式一: 有一個深井高 7 公尺，有一隻蝸牛在井底，白天往上爬 2 公尺，晚上因為睡覺，會下滑 1 公尺，要在第幾天才能爬出井?
數學分析:
寫作程式前，我們都要幾個簡單的例子來尋求答案(除非答案很簡單)，如此才能知道所做的程式是否正確。因此，數學的分析有它的必要性。

以此題為例，我們用三個數 i, day, night 來代表第 i 天蝸牛的行程，第一天，蝸牛白天往上走 2，晚上下滑 1，因此是(1, 2, 1)，第二天則是(2, 1+2, 1+2-1)=(2, 3, 2)，如此反覆，紀錄如下:
(1,2,1)
(2,3,2)

(3,4,3)
(4,5,4)
(5,6,5)
(6,7,
此時我們已經到了井口，所以答案應該是 6 天。

程式分析:

```
def solve():
    now = 0
    day = 0
    while now<7:
        day = day + 1
        now = now + 2
        if now >= 7:
            return day
        now = now - 1

print("井深", well, "公尺，蝸牛白天往上爬", dayup, "公尺，晚上下滑", night, "公尺，需要", solve(), "天才能爬出井")
```

我們定義一個沒有參數的函數 solve。在這個函數中，變數 day 紀錄已經過幾天了，而變數 now 紀錄從井底爬了幾公尺，也就是從井底開始往上算距離，現在蝸牛的位置。在 while loop 中，我們先檢查目前是否還要往上爬(now<7)，若沒有，新的一天開始(day = day + 1)，白天多爬了 2 公尺(now = now + 2)，這時候檢查這兩公尺是否已經足夠爬出井(now >=7)，如果可以，回傳天數，如果不行，睡覺(所以下滑一公尺;now = now - 1)。然後再重複 while loop。

這個程式可以解出此題，但是程式本身卻缺乏擴充性。舉例來說，如果今天將題目稍作修改，將井改成 8 公尺，蝸牛白天上升 3 公尺，晚上下滑 1 公尺，程式就要在不同的地方改不同的數字。因此，我們應該將這題目的三個主要的數字變成變數(或稱全域變數，也就是到處都可以使用到它，包括函數定義內)，以後類似的題目，我們只要改變數的值即可。因此，數學分析和程式分析可以是:

我們可以將上一個程式修改成:

```
up = 2
down = 1
well = 7

def solve():
    now = 0
    day = 0
    while now<well:
```

```
        day = day + 1
        now = now + up
        if now >= well:
            return day
        now = now - down

print("井深", well, "公尺，蝸牛白天往上爬", dayup, "公尺，晚上下滑", night, "公尺，需要", solve(), "天才能爬出井")
```

這個程式執行的結果是:

```
井深 7 公尺，蝸牛白天往上爬 2 公尺，晚上下滑 1 公尺，需要 6 天才能爬出井
>>>
```

我們將上個程式的前三個 assignment 改成

```
up = 3
down = 1
well = 8
```

執行結果則變成:

```
井深 8 公尺，蝸牛白天往上爬 3 公尺，晚上下滑 1 公尺，需要 4 天才能爬出井
>>>
```

經過簡單的分析:

(1,3,2)

(2,5,4)

(3,6,5)

(4,8

所以答案 4 天是正確的。

多做幾個簡單的驗算，我們要擴充到大的數字，要經過人們冗長的計算，但電腦卻一下子就有答案，答案的正確信就更容易被相信了。

基本上使用變數的情況，我們已經將這一類的問題一網打盡。但是，我們能否將整個問題寫成一個函數供他人使用呢?也就是有人給定參數 well, up, down，就可以列印出天數。當然可以，程式如下:

```
def solve(well, up, down):
    now = 0
    day = 0
    while now<well:
        day = day + 1
        now = now + up
        if now >= well:
```

```
            return day
        now = now - down

print("井深 7 公尺，蝸牛白天往上爬 2 公尺，晚上下滑 1 公尺，需要", solve(7, 2, 1), "天才能爬出井")
```

趣味程式二:雞兔問題

二元一次聯立方程式

雞兔同籠的數學運題，出於中國的古代數學「孫子算經」中，這本書書寫於南北朝，約在西元 420-589 年。原問如下:

問：今有雉、兔同籠，上有三十五頭，下有九十四足。問：雉、兔各幾何？

雞兔同籠，從上面看有 35 個頭，只看下面卻有 94 隻腳，請問雞兔各有幾隻?

我們採用三個方法來對此問題求解，第一個方法叫做暴力法(brute force)，也就是用程式嘗試所有的可能性，從全兔，1 雞 34 兔，2 雞 33 兔……一直到 34 雞 1 兔和全雞，然後計算腳數是否符合題目的 94 隻腳。第二種和第三種方法則是先分析題目來簡化程式的寫作。第二種採用孫子算經的方法，第三種則與第二種類似，只是求解的想法不同而已。

在說明孫子算經的解法前，我們先思考第一種方法使用電腦程式來解題。使用電腦程式解題，我們可以嘗試所有可能性的組合，就是將所有可能的雞兔數都先列出，然後計算每一總可能的腳數，當腳數符合就得到解答。也就是說，我們使用 for loop 來計算雞數 i 和兔數 n-i，得到腳數 2i+4(n-i)，這裡的 n 就是頭的數(35)，然後看此數是否等於 94。我們的程式可以是:(程式中用 headno 代表頭數 35，legno 代表腳數 94):

```
headno = 35
legno = 94
track = True

def main():
    for i in range(headno+1):
        if 2*i+4*(headno-i) == legno:
            chicken = i
        if track:
            print(i,"隻雞和", headno-i,"隻兔，共有", leg(i), "隻腳")
    print("雞兔同籠，有",headno,"個頭, 和",legno,"隻腳，則我們算出有",chicken, "隻雞和", headno-chicken, "隻兔")

main()
```

在這個程式中，我們加入一個寫程式的技巧來幫助除錯，所謂除錯，就是利用印出中間值來了解我們的思路與電腦的執行是否一致。在此程式中，我們加入了一個變數叫 track，先設定他是 True。當 track == True 時，我們會將中間的值，也就是在 for loop 中的所有過程都列印出來。但是當 track == False 時，我們指列印出最後的結果。注意到我們在執行 if track == True:時是在 for loop 裡面，而在列印結果是 for loop 結束以後。在 for loop 裡當我們找到了答案，就記錄下來，以便 for loop 結束時使用。

當 track == True，此程式印出:

```
0 隻雞和 35 隻兔，共有 140 隻腳
1 隻雞和 34 隻兔，共有 138 隻腳
2 隻雞和 33 隻兔，共有 136 隻腳
3 隻雞和 32 隻兔，共有 134 隻腳
4 隻雞和 31 隻兔，共有 132 隻腳
5 隻雞和 30 隻兔，共有 130 隻腳
6 隻雞和 29 隻兔，共有 128 隻腳
7 隻雞和 28 隻兔，共有 126 隻腳
8 隻雞和 27 隻兔，共有 124 隻腳
9 隻雞和 26 隻兔，共有 122 隻腳
10 隻雞和 25 隻兔，共有 120 隻腳
11 隻雞和 24 隻兔，共有 118 隻腳
12 隻雞和 23 隻兔，共有 116 隻腳
13 隻雞和 22 隻兔，共有 114 隻腳
14 隻雞和 21 隻兔，共有 112 隻腳
15 隻雞和 20 隻兔，共有 110 隻腳
16 隻雞和 19 隻兔，共有 108 隻腳
17 隻雞和 18 隻兔，共有 106 隻腳
18 隻雞和 17 隻兔，共有 104 隻腳
19 隻雞和 16 隻兔，共有 102 隻腳
20 隻雞和 15 隻兔，共有 100 隻腳
21 隻雞和 14 隻兔，共有 98 隻腳
22 隻雞和 13 隻兔，共有 96 隻腳
23 隻雞和 12 隻兔，共有 94 隻腳
24 隻雞和 11 隻兔，共有 92 隻腳
25 隻雞和 10 隻兔，共有 90 隻腳
26 隻雞和 9 隻兔，共有 88 隻腳
27 隻雞和 8 隻兔，共有 86 隻腳
28 隻雞和 7 隻兔，共有 84 隻腳
29 隻雞和 6 隻兔，共有 82 隻腳
30 隻雞和 5 隻兔，共有 80 隻腳
31 隻雞和 4 隻兔，共有 78 隻腳
32 隻雞和 3 隻兔，共有 76 隻腳
33 隻雞和 2 隻兔，共有 74 隻腳
34 隻雞和 1 隻兔，共有 72 隻腳
35 隻雞和 0 隻兔，共有 70 隻腳
雞兔同籠，有 35 個頭，和 94 隻腳，則我們算出有 23 隻雞和 12 隻兔
>>>
```

但若 track == False，程式就會只印出最後一列。

```
>>>
雞兔同籠，有 35 個頭，和 94 隻腳，則我們算出有 23 隻雞和 12 隻兔
>>>
```

基本上我們稱以上的方法為暴力法(brute force)，也就是一一嘗試的方法，在電腦快速運算的優勢下，有時暴力法可以讓數學分析能力較弱的人也有求解的程式能力。

接著我們說明非暴力的方法，先經由數學分析再來寫程式。先說孫子算數的解法(第二種)，再說(第三種)另一個類似的解法。

孫子算經的解法是將腳的數字先除以二，得 47，再將此數減掉頭的數字，所得的就是兔子的數目。

然後我們用頭的數減掉兔子的數，就會得到雞的數目。

孫子算數解法，我們可以用以下方式來思考:

1. 叫所有的動物把腳舉起來，共有 94 隻腳。
2. 每個動物把他一半的腳放下，也就是雞放下一隻腳，兔子放下兩隻腳，還高舉的腳數剩下愛 94/2=47 隻腳。
3. 每隻動物都再放下一隻腳，還剩下高舉的腳數是 47-35=12 隻腳。注意到目前雞的兩腳都放下了，而每隻兔子都剩下一隻腳。
4. 結論:每隻兔子都舉一隻腳，雞的腳都放下了，所以現在高舉的腳數 12 就等於兔子的腳數。雞的腳數就用雞兔總數減去兔的數，也就是 23。所以，此題的答案是雞有 23 隻，兔有 12 隻。我們驗算 23x2+12x4=94。

在數學上,此範圍是解二元一次聯立方程式。假設雞的數是 x,兔的數是 y,得到兩個方程式是 x+y=35, 2x+4y=94。我們可以先將第二個式子除以 2,改成 x+2y=47,然後減掉第一個式子,得到 y=47-35=12,因此得到答案。

程式分析:

因為在數學分析後我們得到一個公式:

兔數 =（腳數/2） - 頭數

雞數 = 頭數 - 兔數

所以，我們很快地就可以撰寫程式為:

```
def main():
    rabbit = int(legno/2) - headno
    chicken = headno - rabbit
    print("雞兔同籠，有",headno,"個頭, 和",legno,"隻腳，則我們算出有",chicken, "隻雞和", headno-chicken, "隻兔")

main()
```

注意到這個程式中，我們用到 ing(legno/2)。若是我們使用 legno/2，算出來會是有小數點的數，因為這個小數點的數，我們會算出 rabbit = 12.0，chicken = 23.0。如果我們加入 int()，他會把他的參數改為整數(integer)，因而讓最後印出的與上一題相同。

是否有其他的思考方式，通常答案都是有的。一個數學的解題通常有很多不同的方式來解，我們再多介紹一種想法(第三種)，如下:

1. 叫所有的動物把腳舉起來，所以目前有 94 隻腳舉起。
2. 叫所有的動物放下兩隻腳，所以剩下 94-35x2 =24 隻腳。
3. 目前的狀況是雞的兩隻腳都放下了，而每隻兔子都舉兩隻腳，我們把剩下腳數 24 除以 2，所得就是兔子的數字 12。

4. 將總頭數 35 減去兔子數，即得雞數 23。

同樣的，在聯立方程式中，x+y=35, 2x+4y=94，我們先將第一式乘以 2，得 2x+2y=70，然後用第二式減去乘以 2 後的第一式，得 2y=24，最後將此式子除以二，得 y=12 即為兔數。

此題的程式設計方式與上一題相同，我們把它當作練習作業。

趣味程式三: 曉華將平常使用的一元、五元和十元的銅板都存在一個撲滿裡，有一天，他將撲滿裡的錢都拿出來，發現有銅板 200 個，價值 800 元，您知道分別有幾個一元、五元和十元的銅板?我們可以使用暴力法求解,但在暴力法中,我們能不能將所有的可能的組合都列出來?(我們所有的組合都試,當然可以列出所有的組合)。

暴力法的程式我們已經介紹，基本上是利用 for loop 來測試每一種可能性。此題中，我們有三種不同的銅板，需要兩個變數，一個紀錄十元的數量，另一個紀錄五元的數量，然後用兩百減去兩種銅板的數量，就得到一元的數量。我們嘗試各種不同的十元數量和五元數量的組合(一元數量自動計算出來)，因此需要兩層的 for loop(nested loop)來執行，程式如下:

```
coin = 200       # 銅板數
value = 800      # 銅板價值

def ijvalue(i,j): #十元和五元各有 i,j 個，則全部的銅板價值
    return i*10+j*5+(coin-i-j)

def main():
    k = 0        # 計算已經發現多少解，一開始都沒發現，所以等於 0
    for i in range(value//10+1):          # 十元最多只有 value//10=80 個
        for j in range(value//5+1):       # 五元最多只有 value//5=160 個
            if ijvalue(i,j)==value:       # 計算出來的值正好是 800
                print("十元",i,"個，五元",j,"個，一元",coin-i-j,"個，共",ijvalue(i,j),"
元")
                k = k + 1  # 累計多發現一個組合
    print("總共有", k, "種不同的組合")

main()
```

以上程式執行的結果是:

```
>>>
十元 0 個, 五元 150 個, 一元 50 個, 共 800 元
十元 4 個, 五元 141 個, 一元 55 個, 共 800 元
十元 8 個, 五元 132 個, 一元 60 個, 共 800 元
十元 12 個, 五元 123 個, 一元 65 個, 共 800 元
十元 16 個, 五元 114 個, 一元 70 個, 共 800 元
十元 20 個, 五元 105 個, 一元 75 個, 共 800 元
十元 24 個, 五元 96 個, 一元 80 個, 共 800 元
十元 28 個, 五元 87 個, 一元 85 個, 共 800 元
十元 32 個, 五元 78 個, 一元 90 個, 共 800 元
十元 36 個, 五元 69 個, 一元 95 個, 共 800 元
十元 40 個, 五元 60 個, 一元 100 個, 共 800 元
十元 44 個, 五元 51 個, 一元 105 個, 共 800 元
十元 48 個, 五元 42 個, 一元 110 個, 共 800 元
十元 52 個, 五元 33 個, 一元 115 個, 共 800 元
十元 56 個, 五元 24 個, 一元 120 個, 共 800 元
十元 60 個, 五元 15 個, 一元 125 個, 共 800 元
十元 64 個, 五元 6 個, 一元 130 個, 共 800 元
總共有 17 種不同的組合
>>>
```

趣味程式四:

阿基米德與國王下棋，國王輸了，國王問阿基米德要什麼獎賞？阿基米德對國王說：我只要在棋盤上第一格放一粒米，第二格放 2 粒，第三格放 4 粒（2*2=4），第四格放 8 粒（4*2=8），第五格放 16 粒（8*2=16）…，每一格是前一格的倍數，按這個比例放滿整個棋盤六十四格就行。

國王心裡想：這個死讀書的阿基米德，真不貪心，我還怕他要的是真珠寶石，那知道他居然要的是最不值錢的米，國王認為一牛車的米就可把六十四格填滿，這真要不了多少米，就一口答應了阿基米德的要求。

國王傳令開始依阿基米德的要求擺米，一格一格擺米時，國王的臉色也從輕蔑、輕鬆、觀望、疑惑、到睜大眼張大嘴…，直呼『這怎麼可能呢？』。

一個糧倉的米居然還擺不完一半的棋格子，最後的計算結果：即使將全世界的米擺上都不夠！全部擺滿是一個驚人的天文數字！

(以上故事摘錄自網路)

能否寫一個程式來計算全部六十四格的米總共有多少粒?

早期沒有計算機或電腦的輔助，大數量的計算十分複雜，現在有了電腦程式的協助，很簡單的一個 for loop 就可以將起數算出來，如下:

```
now = 1
total = 1
for i in range(1,65):
    print("第", i, "格,", now, "粒米，共拿了", total, "粒米")
    now = 2 * now
    total = total + now
```

但是，這程式的輸出有 64 列，如果我們不想看那麼多列，我們可以每四列印出一列，只要將 print 的指令改成:

```
    if i%4==0:
        print("第", i, "格,", now, "粒米，共拿了", total, "粒米")
```

我們得到的輸出列印是:

```
>>>
第 4 格, 8 粒米, 共拿了 15 粒米
第 8 格, 128 粒米, 共拿了 255 粒米
第 12 格, 2048 粒米, 共拿了 4095 粒米
第 16 格, 32768 粒米, 共拿了 65535 粒米
第 20 格, 524288 粒米, 共拿了 1048575 粒米
第 24 格, 8388608 粒米, 共拿了 16777215 粒米
第 28 格, 134217728 粒米, 共拿了 268435455 粒米
第 32 格, 2147483648 粒米, 共拿了 4294967295 粒米
第 36 格, 34359738368 粒米, 共拿了 68719476735 粒米
第 40 格, 549755813888 粒米, 共拿了 1099511627775 粒米
第 44 格, 8796093022208 粒米, 共拿了 17592186044415 粒米
第 48 格, 140737488355328 粒米, 共拿了 281474976710655 粒米
第 52 格, 2251799813685248 粒米, 共拿了 4503599627370495 粒米
第 56 格, 36028797018963968 粒米, 共拿了 72057594037927935 粒米
第 60 格, 576460752303423488 粒米, 共拿了 1152921504606846975 粒米
第 64 格, 9223372036854775808 粒米, 共拿了 18446744073709551615 粒米
>>>
```

我們看到了一堆數字，知道它很大，卻不知道是多少袋的米。我們假設第七天的 64 粒米是 1 公克的米(連同前六天就有約兩公克的米)，第八天就變 2 公克，我們將她轉換成公斤(0.002 公斤)，重新計算公斤數，所得結果是:

```
>>>
第 4 格, 0.000125 公斤, 共拿了 0.000234375 公斤
第 8 格, 0.002 公斤, 共拿了 0.003984375 公斤
第 12 格, 0.032 公斤, 共拿了 0.063984375 公斤
第 16 格, 0.512 公斤, 共拿了 1.023984375 公斤
第 20 格, 8.192 公斤, 共拿了 16.383984375 公斤
第 24 格, 131.072 公斤, 共拿了 262.143984375 公斤
第 28 格, 2097.152 公斤, 共拿了 4194.303984375 公斤
第 32 格, 33554.432 公斤, 共拿了 67108.863984375 公斤
第 36 格, 536870.912 公斤, 共拿了 1073741.823984375 公斤
第 40 格, 8589934.592 公斤, 共拿了 17179869.183984376 公斤
第 44 格, 137438953.472 公斤, 共拿了 274877906.9439844 公斤
第 48 格, 2199023255.552 公斤, 共拿了 4398046511.103985 公斤
第 52 格, 35184372088.832 公斤, 共拿了 70368744177.66399 公斤
第 56 格, 562949953421.312 公斤, 共拿了 1125899906842.624 公斤
第 60 格, 9007199254740.992 公斤, 共拿了 18014398509481.984 公斤
第 64 格, 144115188075855.88 公斤, 共拿了 288230376151711.75 公斤
>>>
```

我們在第六十四格的時候，總共拿了 288230376151 公噸左有的米。這些米到底有多少?我們用一些數字來表達。

農委會在 2014/9/30 下午發佈糧食供需年報，稻米食用量持續下降，去年每人僅吃 44.96 公斤，創 10 年新低。假設台灣的人口以 2300 萬人計算，每人每年吃 50 公斤的米，一年台灣的總需求量則是 1150000 公噸。這 64 格的米，夠台灣的老百姓以同等的速度吃上二十五萬年。

趣味數學五:

孫子算經.卷下第十七問給我們描述的就是著名的「盪杯問題」的程式。題曰:「今有婦人河上盪杯。津吏問曰:『杯何以多？』婦人曰:『有客。』津吏曰:『客幾何？』婦人曰:『二人共飯，三人共羹，四人共肉，凡用杯六十五。不知客幾何？」

說明:客人中，兩個人共用一個碗吃飯，三個人共用一個碗喝湯，四個人共用一個碗吃肉，只知道用了 65 個碗，請問客人有幾個人?

我們先考慮這個程式是否可以用 for loop。當我們使用暴力法時，我們最好能知道答案的範圍。舉例來說，我們假設知道客人數不會超過 100 人，那我們就可以從 0 人測試到 100 人，用 for i in range(101): 來測試。此題目我們可以使用 while 來解題，如下:

```
guest = 2
while not (guest%4 == 0 and guest % 3 == 0 and guest//4 + guest//3 + guest//2 == 65):
    guest = guest + 1
print(guest)
```

程式先假設客人有 2 人，對每一個數目的客人，我們測試數目是否可以整除 4(同時就整除 2)，也整除 3。同時，用肉的碗數(guest//4)加上用湯的碗數(guest//3)加上用飯的碗數三數相加要等於 65。如果沒有滿足這些條件(while not .....)，我們試下一個數字(guest = guest + 1)。因此，當我們結束 while loop 的那一個 guest 數，就滿足 guest%4 == 0 and guest % 3 == 0 and guest//4 + guest//3 + guest//2 == 65。這個程式在 print(guest)指令中印出 60。

在數學分析裡，題目就很簡單。我們在找一個數 N，使得 N/2+N/3+N/4=65，將式子成以 12，得到 N=(65x12)/13=60。用現在人的智慧解古代人的問題，很多東西都很簡單，但使用新的技術(程式語言)，可賦予這些問題新的生命，充滿了樂趣。

趣味程式六: 我的舅舅名叫黃炳煌，每次跟人打賭，就說:【我輸了名字就倒著念】，可是他並不敢說: 【我輸了名字就倒著寫】。在文字中，倒著寫的句子可以稱為對稱句，例如[上海自來水來自海上]、[西湖垂柳絲柳垂湖西]等。數字上也有數字是左右對稱的，例如 12321。

(1)可否定義一個函數，判斷一個字串是否是對稱?(輸入字串，回傳 True 或 False)
(2)小華是在上一個對稱年(以 2016 開始計算)出生的，曉華的哥哥是在上上個對稱年出生的，請問小華和他的哥哥差幾歲?
(3)可否寫一程式，將西元年的所有數，0, 1, 2…..到 2015 年，計算出(並列印出)所有的對稱數有幾個?

(1) 字串判斷是否是對稱，我們可以比較第一個和最後一個是否相同，第二個和倒數第二個是否相同…等等，只要有一個不同，救回傳 False，一直比到最中間為止。若結束必叫後仍相同，救回傳 True。程式可以如下:

```
def symm(s):                    #s is a string
    le = len(s)
    for i in range(le//2):
        if s[i] != s[-1-i]:
            return False
```

```
    return True
```

注意到我們比較到字串 s 的指標是從 0 到 le//2，le 是字串的長度。當字串的長度是偶數，例如 6，我們比較前三個字母，從 s[0]和 s[-1]=s[5]比，s[1]和[-2]=s[4]比，s[2]和 s[-3]=s[3]比。也就是 s[i]和 s[-1-i]比, i= 0,1,2。當字串是奇數，例如 7，我們一樣比較到 le//2=3，是 s[0]和 s[-1]=s[6]比，s[1]和 s[-2]=s[5]比，s[2]和 s[-3]=s[4]比，此時，s[3]是最中間數，沒有跟任何人比。

(2) 基本上我們設定曉華和他哥哥的年紀變數為 huang 和 elder，剛開始設定為 0。以現在是 2016 年往前推，找到第一個對稱數(使用 symm 函數)給曉華，第二個則給他哥哥。當找到他哥哥的出生年，這個 while loop 就結束，我們就將結果印出，程式如下:

```
elder = 0
huang = 0
now = 2016
while elder == 0:
  if symm(str(now)):
     if huang == 0:
         huang = now
     else:
         elder = now
  now = now - 1
print("曉華是", huang, "年出生的他的哥哥是",elder,"年出生的，差",elder - huang,"歲")
```

```
>>>
曉華是 2002 年出生的他的哥哥是 1991 年出生的, 差 11 歲
>>>
```

(3)使用 symm 函數，程式可以很簡單，一個 for loop 就解決了，如下:

```
L=[]
for i in range(2017):
    if symm(str(i)):
        L.append(i)
print("對稱年包括以下年", L)
print("共",len(L), "個數")
```

程式印出如下:

```
>>>
對稱年包括以下年 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99, 101, 111, 121, 131
, 141, 151, 161, 171, 181, 191, 202, 212, 222, 232, 242, 252, 262, 272, 282, 292, 303, 313, 323, 333
, 343, 353, 363, 373, 383, 393, 404, 414, 424, 434, 444, 454, 464, 474, 484, 494, 505, 515, 525, 535
, 545, 555, 565, 575, 585, 595, 606, 616, 626, 636, 646, 656, 666, 676, 686, 696, 707, 717, 727, 737
, 747, 757, 767, 777, 787, 797, 808, 818, 828, 838, 848, 858, 868, 878, 888, 898, 909, 919, 929, 939
, 949, 959, 969, 979, 989, 999, 1001, 1111, 1221, 1331, 1441, 1551, 1661, 1771, 1881, 1991, 2002]
共 120 個數
>>>
```

趣味程式七: Chinese Remainder Theorem

孫子算經第二十六題:
有物不知其數，三三數之剩二，五五數之剩三，七七數之剩二。問物幾何？

說明:有一個數不知道是多少，三個一數剩下二(除以三餘二)，五個一數剩下三，七個一數剩下二，請問這個數是多少?

事實上這個數有無窮多個，我們把題目改成找出最小的一個正整數，滿足除以三餘 3，除以五餘三，除以七於二。

程式分析: 這程式很簡單，一個 while loop 就可以解決，如下:

```
i = 0
while not (i%3==2 and i%5 == 3 and i % 7 == 2):
    i = i + 1
print(i)
```

此程式會印出 23。

注意到以上的程式採用到一個樣式(pattern)來測試所有的自然數，直到找到一個數滿足某一個測試 P 為止。這個樣式是：

```
i=0
while not P:
    i = i+1
# 找到數 i 滿足 P
```

數學分析:在 Chinese Remainder Theorem 中，限定三個除數(上例的 3, 5, 7)要是質數，所求解的方法對小學生有些複雜，我們留待中級的 Python 再做說明。不過，我們上面的暴力法求解，是可以不限定除數是否為質數。

我們現在考慮另一個題目，如下：
一筐雞蛋，1 個 1 個拿正好拿完，2 個 2 個拿還剩 1 個，3 個 3 個拿還剩 1 個，4 個 4 個拿還剩 1 個，5 個 5 個拿還剩 1 個，6 個 6 個拿還剩 1 個，7 個 7 個拿正好拿完，問筐裡最少有多少雞蛋？

有以上的說明，我們只要改變測試 P 為(i%2==1 and i%3==1 and i%4==1 and i%5==1 and i%6==1 and i%7==0)，這個程式就可以很簡單的寫出來了：

```
i = 0
while not (i%2==1 and i%3==1 and i%4==1 and i%5==1 and i%6==1 and i%7==0):
    i = i+1
print(i)
```

以上程式將會印出 301。

事實上，我們可已經由人腦將所測試 P 簡化。我們知道當 6 個 6 個拿剩 1 個，這種情況保證 2 個 2 個拿還剩 1 個和 3 個 3 個拿還剩 1 個，所以測試 P 可以改成(i%4==1 and i%5==1 and i%6==1 and i%7==0)。

假使，我們希望找的不是一個最小的數，而是要滿足測試的五個數，我們要如何來處理？基本上我們可以使用兩層的 while，第一層測試是否已經有五個答案（存答案的 list 長度等於 5 就表示答案都找到了)，第二層測試下一個答案。用此觀念，程式變成：

```
i = 0
L=[]
while len(L)<5:
    while not (i%4==1 and i%5==1 and i%6==1 and i%7==0):
        i = i + 1
    L.append(i)
    i = i + 1
print(L)
```

此程式會印出

```
[301, 721, 1141, 1561, 1981]
```

以上我們都是使用程式來解一個獨立的問題。我們是否可以利用程式來寫一個遊戲?答案當然是肯定的。接下來我們撰寫三個不同的程式，有兩個猜數字遊戲和一個井字遊戲。

趣味程式八:過大過小猜數字

(1) 電腦想一個數字由您猜，電腦會提示過大過小，直到您猜對為止。

(2) 同上，但限定您只能猜 6 次。

趣味程式九:?A?B

(3)電腦想一個四位數字由您猜，電腦會提示多少個數字與位置皆正確(多少 A)，和多少個數字正確但位置錯誤(多少 B)。您經由這些線索繼續猜題，直到您猜對為止。

(4) 同上，但限定您只能猜 8 次。

趣味程式十:井字遊戲

您先在井字遊戲中加入 o，電腦會自動選一個位置加入 x，誰先排成一列、一行、或斜角邊同一個符號者獲勝。

練習:

1. 試寫一程式採用第三種雞兔同籠問題的解法來求解。

2. 有個動物園的管理員決定計算動物園裡的獅子和鴕鳥的數量,他算出了總共有35個頭和78條腿,你知道分別有多少隻獅子和多少隻鴕鳥?

3. 曉華將平常使用的一元和五元的銅板都存在一個撲滿裡,有一天,他將撲滿裡的錢都拿出來,發現有銅板200個,價值800元,年知道分別有幾個一元和五元的銅板?

4. 孫子算數內有一個「和尚分菓」的問題:

問:百和尚,分饃百,大和尚每人著三,小和尚三人著一,問大小和尚各若干人?

說明:有一百個和尚分一百枚菓。大和尚一個人吃三枚,小和尚三個人吃一個枚,請問有幾個大和尚,幾個小和尚?

(a)請用暴力法寫一程式求解。
(b)請先用數學分析然後求解。
提示:大和尚一人吃三枚,小和尚三人吃一枚,所以這四個和尚共吃四枚。

中階: Python + Visual

顏色介紹:
電腦處理顏色,通常使用三個數字,就是RGBA。RGB代表三種顏色,就是紅色(Red)、綠色(Green)、和藍色(Blue)。Python要求這三個數界於0到1之間。假設我們要求RGB三個數只能是0或1,則有八種顏色,如下:

(0,0,0): 黑色(Black)　(1,1,1): 白色(White)
(1,0,0): 紅色(Red)　(0,1,0): 綠色(Green)　(0,0,1): 藍色(Blue)
(1,1,0): 黃色(Yellow)　(1,0,1): 錠紫色(Magenta)　(0,1,1): 青色(Cyan)

我們使用五個程式來介紹顏色的使用,個別介紹:
1. 如何畫出一個紅色的圓。
2. 如何畫出八個同心圓,由大到小個別包含了上面所介紹的八個顏色。
3. 如何畫出灰色的圓。

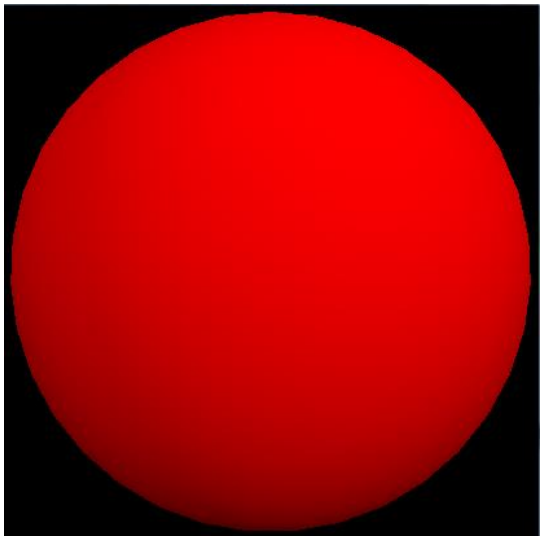
1. 如何畫出一個紅色的圓。
程式如下:

```
from visual import *                    #1
```

```
sphere(pos=(0,0), radius=1, color=(1,0,0))       #2
```

指令介紹:
#1: 匯入 Vpython 的模組，使得該模組內的庫存函數(別人已經寫好的函數)可以供您使用
#2: 劃出一個圓，它的圓心在座標(0,0)(pos=(0,0))，半徑為 1(radius=1)，用紅色畫出(color=(1,0,0))。

此程式的執行結果是:



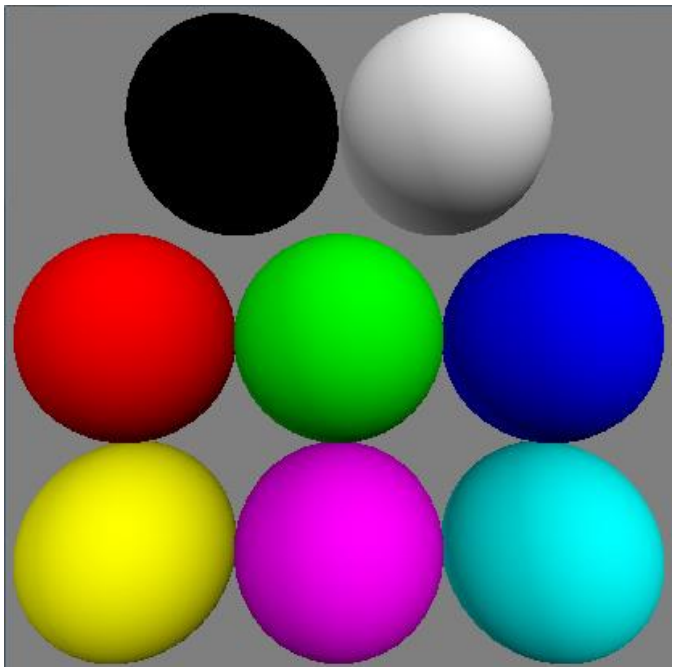
當程式執行時，有一些預設(default)的值會被使用。舉例來說，此程式自動的使用了黑色當背景，黑色就是當你沒有設定背景顏色時，他就預設背景顏色為黑色。
注意到 sphere 這個單字是三度空間的球體，我們設定他的圓心為(0,0)，他的第三個元素的預設值是 0，也就是此球體的圓心是在(0,0,0)，但是因為 python 預設我們是由 Z 軸往 XY 平面看，因此球體變成了圓型。

2. 如何畫出八個圓，包含了上面所介紹的八個顏色。
程式如下:

```
from visual import *

scene.background=(0.5,0.5,0.5)                    #1
sphere(pos=(-1,2), radius=1, color=(0,0,0))
sphere(pos=(1,2), radius=1, color=(1,1,1))
sphere(pos=(-2,0), radius=1, color=(1,0,0))
sphere(pos=(0,0), radius=1, color=(0,1,0))
sphere(pos=(2,0), radius=1, color=(0,0,1))
sphere(pos=(-2,-2), radius=1, color=(1,1,0))
sphere(pos=(0,-2), radius=1, color=(1,0,1))
sphere(pos=(2,-2), radius=1, color=(0,1,1))
```

指令介紹:
#1: 將背景的顏色設定為(0.5,0.5,0.5)，基本上是介於白色(1,1,1)和黑色(0,0,0)之間，所以是黑色。

這程式執行的結果是:



注意到右上方的白球內有一些顏色是灰色的，基本上是因為 python 在處理圖形時有預設燈光，因此所有的球都會有陰影，而白色看得最明顯。

3. 如何畫出灰色的圓。
當 r=g=b=0 時我們會畫出黑色，r=g=b=1 時則是白色。當 r=g=b 且他的值介於 0 到 1 之間(但不等於 0 或 1 時)，我們會得到灰色。以下程式我們設定一個數 N，然後將 0 到 1 分成 N 個不同的值，從最黑的(0,0,0)一值畫到最白的(1,1,1)，程式如下:

```
from visual import *

N=8
scene.background=(0,0.8,0.8)
for i in range(-N+1,N,2):                          #1
    a = (i+N-1)/(2*N-2)                            #2
    sphere(pos=(i,0), radius=1, color=(a,a,a))     #3
```

指令說明:
#1 使用 for loop，設定一個 i 值，希望經由此 i 值求得球體的座標以及灰階的顏色
#2 設定一個灰階的顏色，當 i 是第一個元素和最後一個元素時，計算出的值個別是 0 和 1。中間的值則會等分 0 到 1。

#3 劃出圓。注意到所畫出圓的座標在(i,0)，也就是 X 軸上，顏色則是 r=g=b=a(#2 所計算出的值)。

此程式所計算出的結果為:

if then elif
偷竊者(869)
873
整數分析
99 乘法(for + function)
885

中階
(863)
871
啊 879
897
899
nested for
模劍初階系列
磨劍中階系列
List[i,j]

Recursive 介紹

n 階層的定義在於從 1 乘到 n(記做 n!)，也就是:
n! = 1x2x3x..... xn
所以，1!=1, 2!=2, 3!=6, 4!=24, 5!=120, ...。用此定義，我們可以寫一個函數計算階層，也就是輸入參數為 n，回覆值為 n!。此函數的程式如下:

```
def factorial(n):
    total = 1
    for i in range(1, n):
        total = total * i
    return total
```

我們也可以使用遞迴(recursive)的方式來定義 n!，定義方式如下:

$$n! = \begin{cases} 1 & \text{if } n = 1 \\ n \times (n-1)! & \text{if } n > 1 \end{cases}$$

通常，遞迴的概念是使用自己來定義自己，說得明白一點，是使用簡單得自己來定義複雜的自己，但同時要定義一個最簡單的自己。以階層為例，我們使用(n-1)!來定義 n!，但也要定義一個最簡的 1!=1。以 4!為例，4!=4x3!=4x(3x2!)=4x(3x(2x1!))=4x(3x(2x1))=4x(3x2)=4x6=24。

對階層的遞迴定義，我們也可以寫一個函數，如下:

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```

Fibb 故事介紹

$f_0 = 0,\ f_1 = 1,\ f_n = f_{n-1} + f_{n-2}$

```
def fibb(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fibb(n-1) + fibb(n-2)
```

回顧我們在講述 while 時，有一個題目，如下:

一瓶可樂一元，兩個可樂空瓶可以換一瓶可樂。書蟲小美有十元，請問最多可以喝到幾瓶可樂?

我們使用 while 來解此題目。但是，我們也可以使用遞迴的概念來解此問題，如下:

```
def cola(n, rem):
    empty = n + rem
    if empty < 2:
        return n
    else:
        return n + cola(empty // 2, empty % 2)
```

在上面程式中，n 代表目前的可樂數，rem 代表目前的空瓶數。因此，一開始的十元，我們可以買 10 瓶可樂(n=10)，還沒買時有 0 個空瓶(rem=0)，所以呼叫程式 cola(10,0)就會計算出 10 元可以喝到幾瓶可樂。程式一開始就計算所有的空瓶(empty)等於目前可樂數(n)加上以前留下來的空瓶數(rem)，就是 empty = n + rem，之後，我們判斷是否可以換可樂，當空瓶數小於 2 時，就是我們碰到最簡單版本的自己，我們回傳可喝的可樂數 n，當空瓶數大於 2 時，除了我們可以喝的數 n 以外，我們可以將空瓶所換得可樂數 empty//2，和換可樂以後剩下的空瓶數 empty%2 當作參數，再去呼叫自己(比較簡單版本的自己)，也就是 n+ cola(empty // 2, empty % 2)。

練習:

使用遞迴的方式來定義從 1 加到 n 的合，也就是計算$\sum_{i=0}^{n} i = 1 + 2 + 3 + \cdots + n$。同時，寫一個函數程式來計算 1 加到 n 的合。

precedence, real number,

高階： Python + Visual; Python +

Curve

Berstein Polynomal

描述於此:

Bezier Curve

Surface

Animation

Normal + tangent + bi

如何繪製球體

如何繪製對稱一軸的曲面(Surface of Revolution)

我們考慮在 XY 平面上繪製一個曲線，然後繞著 Y 軸旋轉一圈，這種方法所產生的曲面，對稱 Y 軸。我們考慮在 xy 平面上的曲線，因此，曲線的參數式是:

C(t) = (X(t), Y(t), 0), 0≤t≤1

在此參數數對 Y 軸旋轉後，所得的曲面參數式則是:

S(s,t) = (X(t)cos(s), Y(t), X(t)sin(s)), 0≤s≤2π

我們可以將以上的公式寫成矩陣式，就是

$$S(s,t) = [X(t),Y(t),0]*\begin{bmatrix}\cos(s) & 1 & \sin(s)\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}$$

在程式的撰寫部分，我們可以先考慮一個小線段的 surface of revolution，然後再考慮一整個曲線所旋轉出的曲面。這個程式需要一些其他人已經撰寫的庫存函數，叫做 Vpython(V 代表 Visual)，因此，第一個指令就是將 visual 輸入，也就是 from visual import *。我們分兩部分來介紹此程式，包括:

(1) 一個線段所產生的曲面。

(2) 一連串的點所產生的曲面。

(3) 一個曲線所產生的曲面。

(4) 如何繪製球體

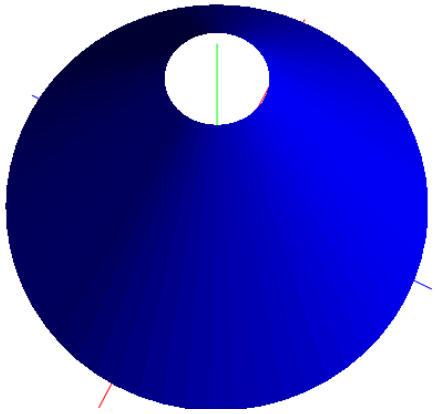
(1) 一個線段所產生的曲面。

```
from visual import *
scene.background = (1,1,1)                  # 將背景設為白色
n = 8                                       #給定視窗的寬度與高度
curve(pos = ((-n,0,0),(n,0,0)), color = color.red)          # 用 r,g,b 三色畫出三軸
curve(pos = ((0,-n,0),(0,n,0)), color = color.green)
curve(pos = ((0,0,-n),(0,0,n)), color = color.blue)

f = faces(pos = [])                         # 設定所要繪製的 faces，一開始是空的
step = 0.1                                  # 設定旋轉時角度的粗細
def rotate_Y(p, theta):                     # 將 p 點對 Y 軸逆時鐘旋轉 theta 角
    return [p[0]*cos(theta), p[1], p[0]*sin(theta)]

def line_revolution(p0, p1):                # 將 p0, p1 所連成的線段對 Y 軸旋轉一圈
    op0 = p0
    op1 = p1
    for theta in arange(0, 2*pi+step, step):
        q0 = rotate_Y(op0, step)            # ur;
        q1 = rotate_Y(op1, step)
        #print(p0,p1,q0,q1)
        for p in [op0,q0,q1,q1,op1,op0]:
            f.append(pos=p)                 # 產生的 faces 放到 f 裡面
        op0 = q0
        op1 = q1
    f.make_normals()                        # 幫所有的 faces 找到法向量
    f.color = color.blue                    # 設定顏色為藍色
#    f.make_twosided()                      # 是否要繪製曲面的兩面?
```

我們的主程式設定則是:

```
line_revolution([1,7],[7,1])
```

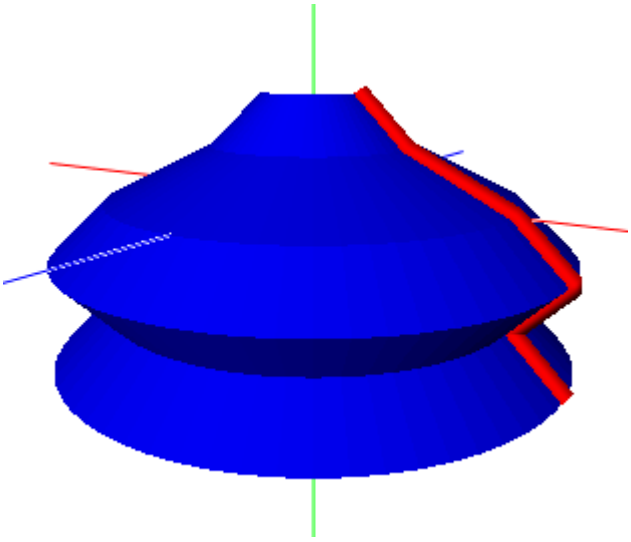
所產生的圖形如下圖:



(2) 一連串的點所產生的曲面。

現在考慮將一個 List 裡面所有的點都做 surface of revolution，我們修改主程式，變成:

```
L=[[1,2],[2,1],[4,0],[5,-1],[4,-2],[5,-3]]          # 定義在 XY 平面上的點
curve(pos = L, color = color.red, radius=0.2)       # 將此曲線用紅色粗線畫出來
for i in range(len(L)-1):
#           rate(10)                                # 是否控制繪圖的速度
    line_revolution(L[i], L[i+1])                   # 劃出 list 中的兩點所製作之曲面
```



(3) 一個曲線所產生的曲面。

因為此程式所產生的例子較大，我們先將程式中的 n 改成 18

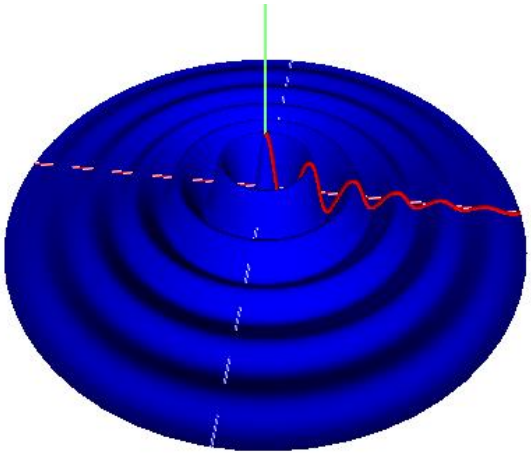
```
n = 18
```

然後主程式改成:

```
def main():
    L = []
    for xx in arange(0, n+step, step):          # 產生函數上的點，放到 L
        yy = 5*cos(2*xx)*exp(-0.2*xx)
```

```
            L.append([xx,yy])
        curve(pos = L, color = color.red, radius=0.2)  # 用紅色劃出曲線
        for i in range(len(L)-1):                      # 用藍色劃出曲面
#               rate(10)
            line_revolution(L[i], L[i+1])
main()          # 呼叫 main()
```

此程式畫出了曲面如下:



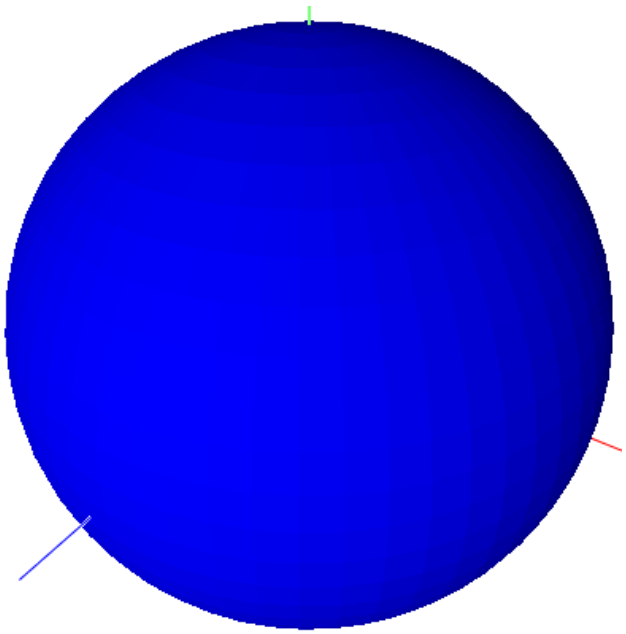
事實上，在 Vpython 裡面有函數可以幫忙做 surface of revolution，叫做 extrude。

add something here

(4) 如何繪製球體

事實上，球體也是 surface of revolution 的一種，我們只要在 XY 平面上繪製出一個半圓即可。我們在此程式中將半圓型的紅色曲線不繪製。因此，我們可以將主程式改為(設定此時 n=1):

```
def main():
    L = []
    for theta in arange(0, pi+step, step):      # 產生半圓形的點，放到 L
        L.append([sin(theta),cos(theta)])
    for i in range(len(L)-1):                   # 用藍色劃出曲面
        line_revolution(L[i], L[i+1])
main()          # 呼叫 main()
```



為何北極圈會有 24 小時的白天?可否模擬?

```
from visual import *
import math
import numpy as np
from numpy.linalg import inv
# import random
angle = 23.5*pi/180
scene.background = color.gray(0.2)
#1 draw 3 axis.
angleaxis = 23.5*pi/180
xaxis = arrow(pos=(0,0,0), axis=(3,0,0), shaftwidth=0.01, color=color.red)
yaxis = arrow(pos=(0,0,0), axis=(0,3,0), shaftwidth=0.01, color=color.green)
zyaxis = arrow(pos=(0,0,0), axis=(0,0,3), shaftwidth=0.01, color=color.blue)
eaxis = arrow(pos = (0,0,0), axis=(2*sin(angleaxis), 2*cos(angleaxis), 0),shaftwidth=0.01, color = color.yellow)

earth = sphere(pos = (0,0,0), radius = 1.0, material = materials.earth)
s = sin(angle)
c = cos(angle)
e1 = [s, c, 0]
e2 = [-c, s, 0]
e3 = [0,0,1]
```

```
#e1 = [a,b,c]=[0,-1,0]
#e2 = [d,e,f]=[0,0,1]
#e3 = [g,h,i]= [-1,0,0]
M = np.array([e1, e2,e3])
#print(M)
#w = [0,0,0]
e = [0,s,0]
minv = inv(M)
p = [c*s,c*c,0]

def minuslist(L1, L2):
    return [L1[i]-L2[i] for i in range(len(L1))]

#print(minv)
# w, p, M represent in world coordinates
# e is in eye coordinats
def w2e(w, p, M):
    return dot([w[i]-p[i] for i in range(len(w))], inv(M))

def e2w(e, p, M):
    M1 = dot(e, M)
    return [M1[i]+p[i] for i in range(len(M1))]

#print(w2e(w, p, M))
print(e2w(e, p, M))

step = 0.01
ang=0
rL = []
while 1:
        rate(20)
#         moon.pos = (5*cos(ang), 0, 3*sin(ang))
        earth.rotate(angle = step, axis = (2*sin(angleaxis), 2*cos(angleaxis), 0), origin = (0,0,0))
        ang = ang + step
        rL.append(e2w([0, s*cos(ang), s*sin(ang)], p, M))
        points(pos = rL, color=color.red, retain = 2*pi/step)
        if ang > 2*pi: ang = 0
```

Morphing

我們分三部分來說明:

(1) 對一個點做 morphing，但保留(顯示)中間的過程。

(2) 對一個點做 morphing，但不保留中間的過程。

(3)對一條線做 morphing，但保留中間的過程。

(4) 對一條線做 morphing，但不保留中間的過程。

(5) 對一個曲線做 morphing，但保留中間的過程。

(6) 對一個曲線做 morphing，但不保留中間的過程。

HW:

1. 對一個點做 morphing，但保留(顯示)中間的過程。

```
from visual import *

xaxis = arrow(pos=(0,0), axis=(1,0), shaftwidth=0.01, color=color.red)
p0 = [0,0]
p1 = [1,0]
n = 10
def twop (p0, op, p1):        # op=' + ', return p0+p1, op=' - ', return p0-p1
    if op == '-':
        return [p0[0]-p1[0], p0[1]-p1[1]]
    else:
        return [p0[0]+p1[0], p0[1]+p1[1]]
def morphing(p0, p1):
    v1 = twop(p1, '-', p0)
    step = [(1/n)*v1[0], (1/n)*v1[1]]
    p = p0
    for i in arange(0, n):
        rate(10)
        p = twop(p, '+', step)
        points(pos = p, radius = 0.5)
 morphing(p0, p1)
```

2. 對一個點做 morphing，但不保留中間的過程。

```
from visual import *

xaxis = arrow(pos=(0,0), axis=(1,0), shaftwidth=0.01, color=color.red)

p0 = [0,0]
```

```
p1 = [1,0]
n = 10

ball = sphere(pos=p0, radius = 0.05)
def twop (p0, op, p1):       # Same as in 1

def morphing(p0, p1):
    v1 = twop(p1, '-', p0)
    step = [(1/n)*v1[0], (1/n)*v1[1]]
    p = p0
    for i in arange(0, n, 1):
        rate(10)
        p = twop(p, '+', step)
        ball.pos = p
morphing(p0, p1)
```

3. 對一條線做 morphing，但保留中間的過程。

```
from visual import *

xaxis = arrow(pos=(0,0), axis=(1,0), shaftwidth=0.01, color=color.red)
yaxis = arrow(pos=(0,0), axis=(0,1), shaftwidth=0.01, color=color.green)

p0 = [0,0]
p1 = [1,0]
q0 = [0,1]
q1 = [0,0]
n = 100

def twop (p0, op, p1):    # same as in 1
def midpoint(p0, p1, t):
    return [p0[0]*(1-t) + p1[0]*t, p0[1]*(1-t) + p1[1]*t]
def morphing(p0, p1, q0, q1):
    for i in arange(n+1):
        rate(10)
        t = i/n
        pp = midpoint(p0, p1, t)
        qq = midpoint(q0, q1, t)
        points(pos = [pp, qq])
        ball2.pos = midpoint(q0, q1, t)
        curve(pos = [pp, qq])
```

```
morphing(p0, p1, q0, q1)
```

4. 對一條線做 morphing，但不保留中間的過程。

```
from visual import *
xaxis = arrow(pos=(0,0), axis=(1,0), shaftwidth=0.01, color=color.red)
yaxis = arrow(pos=(0,0), axis=(0,1), shaftwidth=0.01, color=color.green)

p0 = [0,0]
p1 = [1,0]
q0 = [0,1]
q1 = [0,0]
n = 10
ball = sphere(pos=p0, radius = 0.05)
ball2 = sphere(pos=q0, radius = 0.05)
cc = curve(pos = [p0, q1])

def twop (p0, op, p1):      # same as in 1
def midpoint(p0, p1, t):    # Same as in 3
def morphing(p0, p1, q0, q1):
    for i in arange(n+1):
        rate(10)
        t = i/n
        ball.pos = midpoint(p0, p1, t)
        ball2.pos = midpoint(q0, q1, t)
        curve(pos = [ball.pos, ball2.pos])

morphing(p0, p1, q0, q1)
```

5.

我們曾經寫過如何經由四個控制點產生 Bezier Curve，現在我們要將一個由 p0, p1, p2, p3 四個控制點所定義的 Cubic Bezier Curve 變形為由 q0, q1, q2, q3 四個控制點所定義的 Cubic Bezier Curve，因此，我們必須寫一個 morphing 程式，裡面有八個參數，如下:

```
def bezier(p0, p1, p2, p3):
def morphing(p0, p1, p2, p3, q0, q1, q2, q3)
```

Data Visualization： Python + matplotlib

2D Visualization

程式範例一: 函數繪圖
給定一個二度空間的函數 f(x)=sin(x)，如何將圖形匯出。

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,2*np.pi)       # 1
y = np.sin(x)                    # 2
plt.plot(x,y, color=(1,0,0))     # 3
plt.show()                       # 4
```

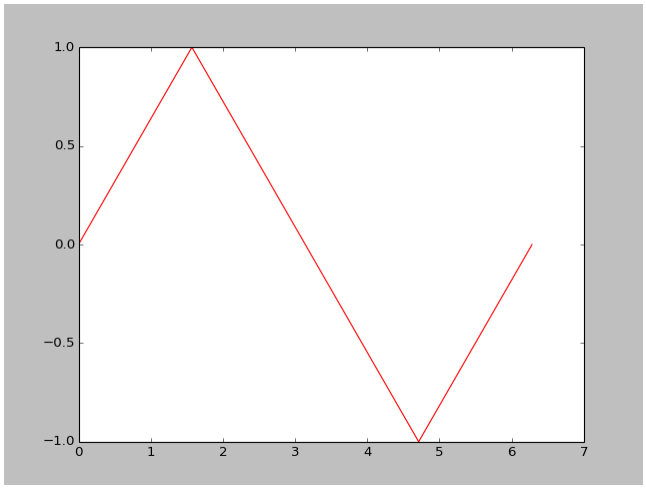
指令說明:
#1: x=np.linspace(0, 2*np.pi)
np.linspace 後有兩個參數，0 和 2π。另有一個重要的第三個參數，預設值為 50。基本上此指令的回傳值是一個數列，此數列將 0 到 2π之間大致等分為 49 份，也就是數列長度是 50。第一個數是 0，最後一個數是 2π。
#2: y=np.sin(x)
由於 x 是一個數列，np.sin(x)將數列中的 50 個數個別求取 sin(x)，然後產生一個新的數列 y 來代表所求得的 50 個 sin(x)值。相對位置產生相對地值，也就是說，在 x 中的第八個數，他的 sin 值為在 y 中的第八個數。
#3: plt.plot(x,y, color=(1,0,0))
將所求得的 x 和 y 對應的 50 個點用紅色描繪(plot)出來。
#4: plt.show()
依照前面的設定展現出圖形。

注意到 x 的長度與 y 的長度相同。當 x 的長度越長，將 sin(x)的定義域切的越細，所畫出來的圖形就越精準。我們從很簡單的例子開始，假使我們將程式做一個小改變，將 0 和 2π之間切成四等分(x 的長度為 5)，同時將 x 和 y 都印出，程式如下:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0,2*np.pi, 5)        # 紅色字為新增部分
y = np.sin(x)
print("x=", x)
print("y=", y)
plt.plot(x,y, color=(1,0,0))
```

```
plt.show()
```

此程式的匯出如下。



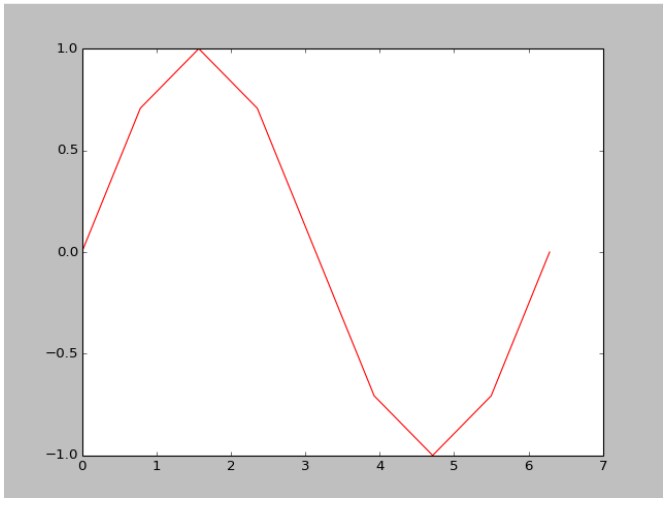
圖一: 用五個點(四等份)趨近 sin(x)

同時會印出:

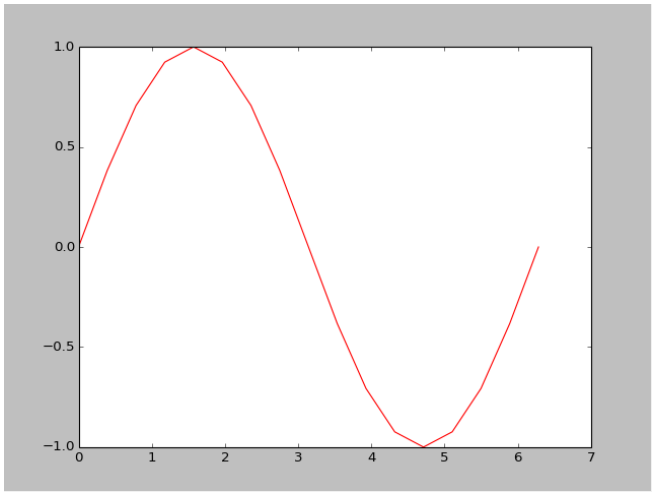
```
x= [ 0.          1.57079633  3.14159265  4.71238898  6.28318531]
y= [  0.00000000e+00   1.00000000e+00   1.22464680e-16  -1.00000000e+00
  -2.44929360e-16]
>>>
```

注意到 x 的五個值大致等於 0, π/2, π, 3π/2, 2π，而相對應的 y 值個別約為 0, 1, 0, -1, 0。

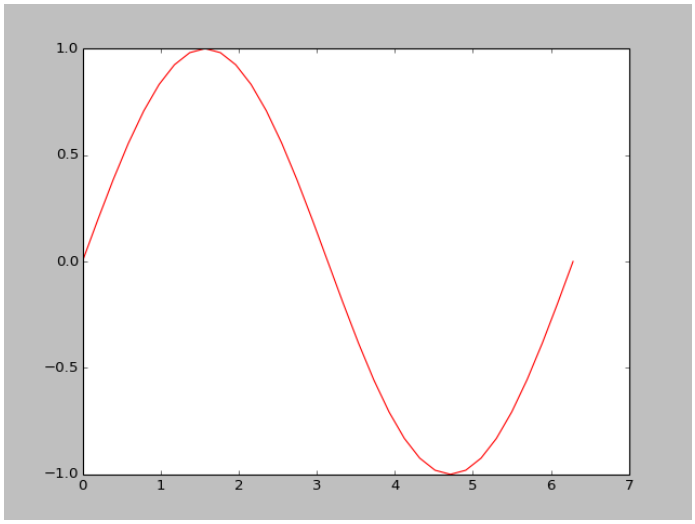
我們將 sin(x)切成四等分，八等分，十六等分，三十二等分，圖形如下:



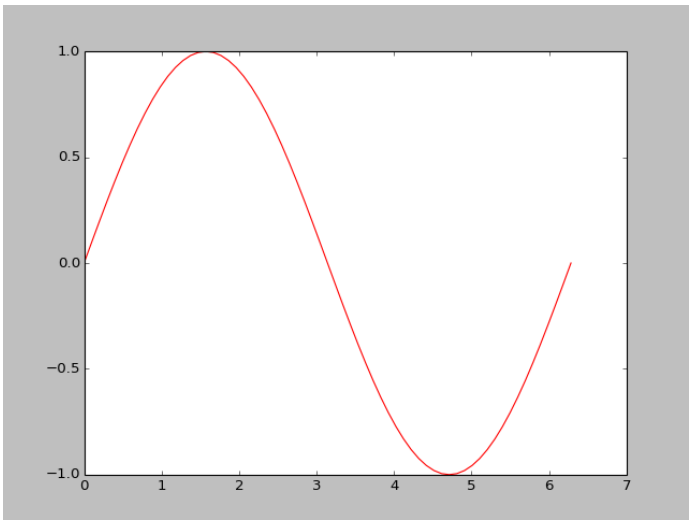
(a) 八等分



(b) 十六等分



(c) 三十二等分



(d) 六十四等分

圖二:以不同的點數等分 0 到 2π來趨近 sin(x)

雖然等分數越大，所求得的越精細，但是螢幕的解析度有限，到達一定的精準以後，再增加等分數就不再會影響顯示的結果，反而增加了計算的時間。以上圖為例，預設值為 50 應該是不錯的選擇。

程式範例二: Bar Chart

東吳大學某教授教授通識課程"Python 程式設計"，經統計，該校六學院來修該課的人數，理學院 102 人，文學院 15 人，外語學院 20 人，法學院 19 人，商學院 60 人，巨資學院 77 人，共 293 人。可否用 bar chart 將這個分布繪出。

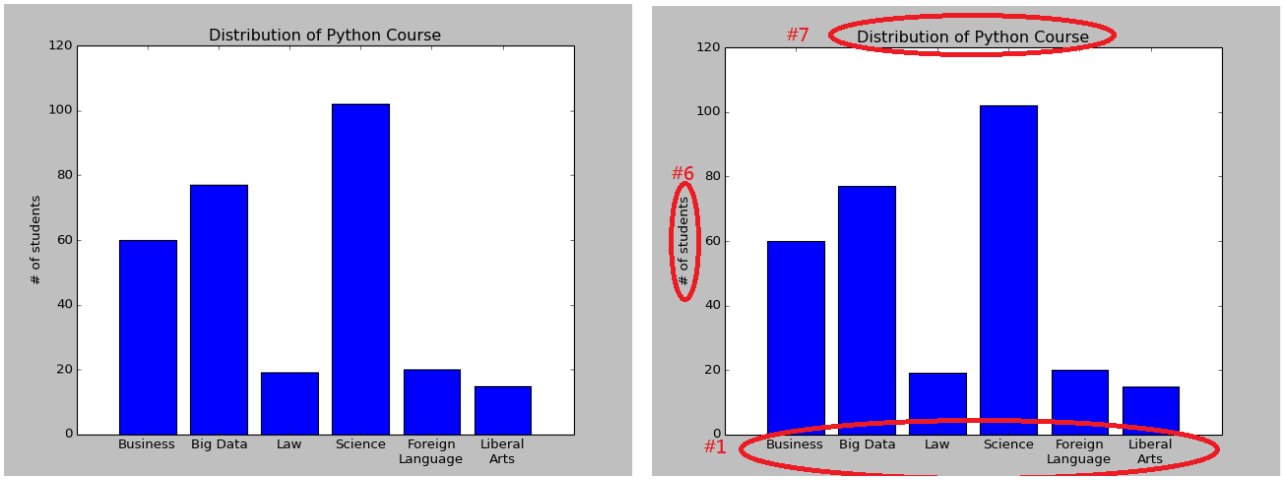
程式如下:

```
import numpy as np
import matplotlib.pyplot as plt

objects=('Business','Big Data','Law','Science','Foreign\nLanguage','Liberal\nArts') #1
y_pos = np.arange(len(objects))                                    # 2
performance = [60,77,19,102,20,15]                                 # 3

plt.bar(y_pos, performance, align = 'center')                      # 4
plt.xticks(y_pos, objects)                                         # 5
plt.ylabel('# of students')                                        # 6
plt.title('Distribution of Python Course')                         # 7

plt.show()
```

指令說明:

#1: 設定六個學院的英文名稱(使用中文輸入在此會產生問題)。在文字中有\n 的符號代表跳行。如 Foreign\nLanguage 代表印出 Foreign 後跳到到下一行在印出 Language。

#2: 產生 y_pos 為一個 0 到 5(共六個數)的數列。
#3: 將 performance 數列儲存各相對學院的修課學生數。
#4: 所繪製的 bar chart 中，第一個參數 y_pos 表示在 x 軸的位置有六等分(y_pos 為 0 到 5 的數列)，第二個參數 performance 表達每一個相對位置 bar chart 的高度，第三個參數 align='center' 表示 bar chart 對其中間，所以兩邊所留的空白相等。
#5: plt.xticks 的第一個參數在 X 軸上設定六個等分的空間，在他的下方將第二參數 object 的六個學院名稱依序排出。
#6: 將 plt.ylabel 的參數'# of students'寫在 Y 軸旁邊。
#7: 給定表格的名稱。
#1, 6, 7 可參考圖三(b)。



(a) 結果　　(b) 指令說明

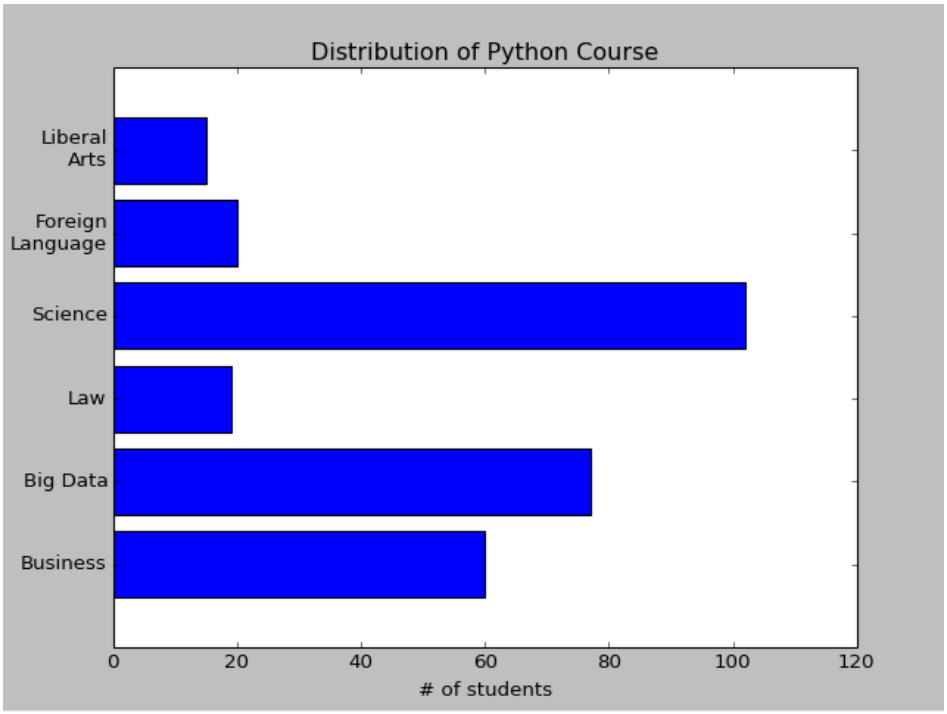
圖三 程式範例二

程式範例三 Bar Chart - Horizontal
同程式範例二，我們想將所有的"bar"由左到右延伸，也就是將圖三中的 X 軸和 Y 軸的資料對換，程式如下:

```
import numpy as np
import matplotlib.pyplot as plt

objects = ('Business', 'Big Data', 'Law', 'Science', 'Foreign\nLanguage', 'Liberal\nArts')
y_pos = np.arange(len(objects))
performance = [60,77,19,102,20,15]

plt.barh(y_pos, performance, align = 'center')
plt.yticks(y_pos, objects)
plt.xlabel('# of students')
plt.title('Distribution of Python Course')
```

```
plt.show()
```

程式執行結果如下圖:



指令說明:

我們在此程式中，在 plt.bar, plt.xticks, plt.ylabel 三個指令只改了三個英文字母(以紅色的字母顯現)，改為 plt.barh, plt.yticks 和 plt.xlabel，這指令的改變將由下往上直立式的 bar chart 改成左往右的橫躺式 bar chart。

程式範例四 Pie Chart

同程式範例二，我們想將所有的資料使用 pie chart 來呈現修 python 課程學生人數在各學院所佔的百分比，程式如下:

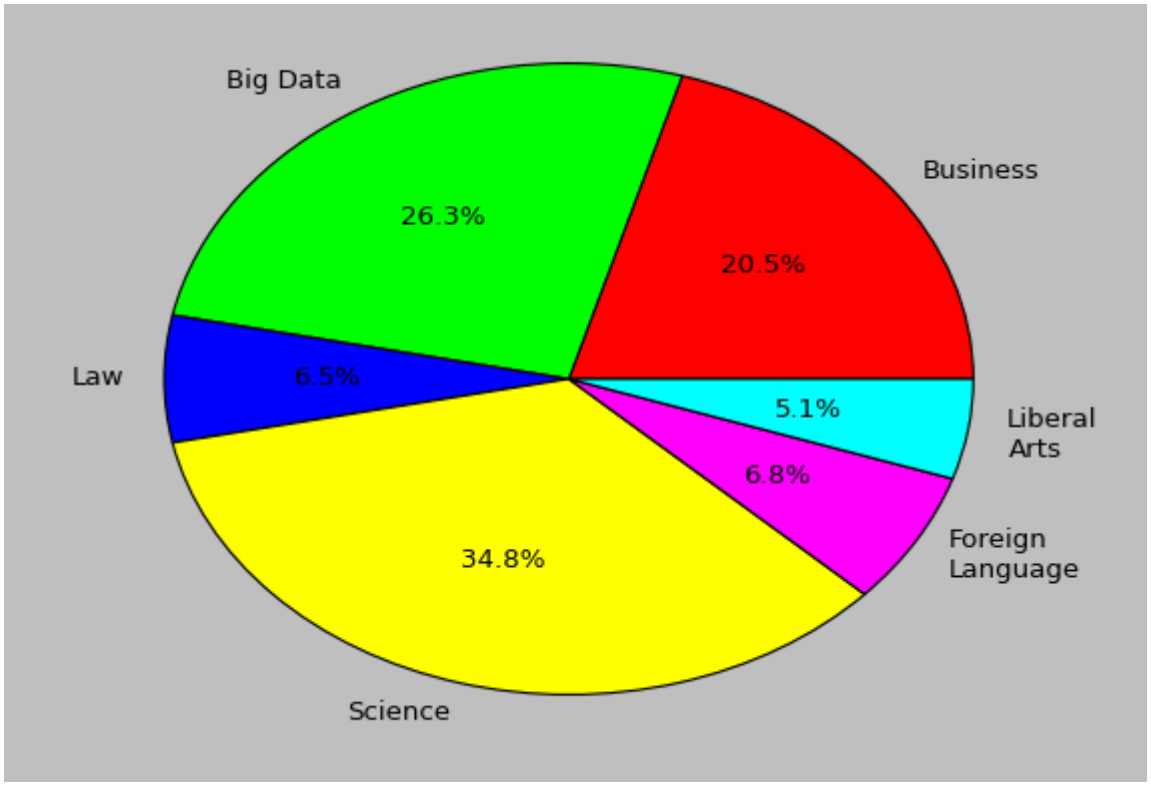
```
import matplotlib.pyplot as plt

objects = ('Business', 'Big Data', 'Law', 'Science', 'Foreign\nLanguage', 'Liberal\nArts')
performance = [60,77,19,102,20,15]
colors = [(1,0,0), (0,1,0), (0,0,1), (1,1,0),(1,0,1),(0,1,1)]              #1
plt.pie(performance, labels=objects, colors=colors, autopct='%.1f%%')   #2
plt.show()
```

指令說明:

#1 設定紅色(對應到 Business)、綠色、藍色、黃色、紅紫色和青色六色來對應六個學院

#2 使用學生的修課人數(performance)，學院名稱(labels=objects)，#1 所設定的顏色將 pie chart 畫出來。在 pie chart 中的數字百分比，用 autopct='%.1f%%' 的格式列印出來。

此程式的執行結果，如下:



增加 autopct 的說明

注意到我們設定學院的順序，或者是顏色的順序，是由 X 軸開始，逆時鐘旋轉。也就是一開始的紅色是從 X 軸向右的向量(表示 0^0)開始，綠色是下一個，接著是藍色如此逆時鐘旋轉。注意到此 pie chart 看起來不像圓形，而是像橢圓形。我們在給予此 pie chart 多一些指令來產生一些變化(以下程式紅色部分)，如下:

```
import matplotlib.pyplot as plt

objects = ('Business', 'Big Data', 'Law', 'Science', 'Foreign\nLanguage', 'Liberal\nArts')
performance = [60,77,19,102,20,15]
colors = [(1,0,0), (0,1,0), (0,0,1), (1,1,0),(1,0,1),(0,1,1)]
explode = (0, 0, 0, 0.1, 0, 0)                                    #1
plt.pie(performance, explode=explode, labels=objects, colors=colors, autopct='%.1f%%',
shadow=True, startangle=90)                                       #2
# Set aspect ratio to be equal so that pie is drawn as a circle.
plt.axis('equal')                                                 #3
plt.show()
```

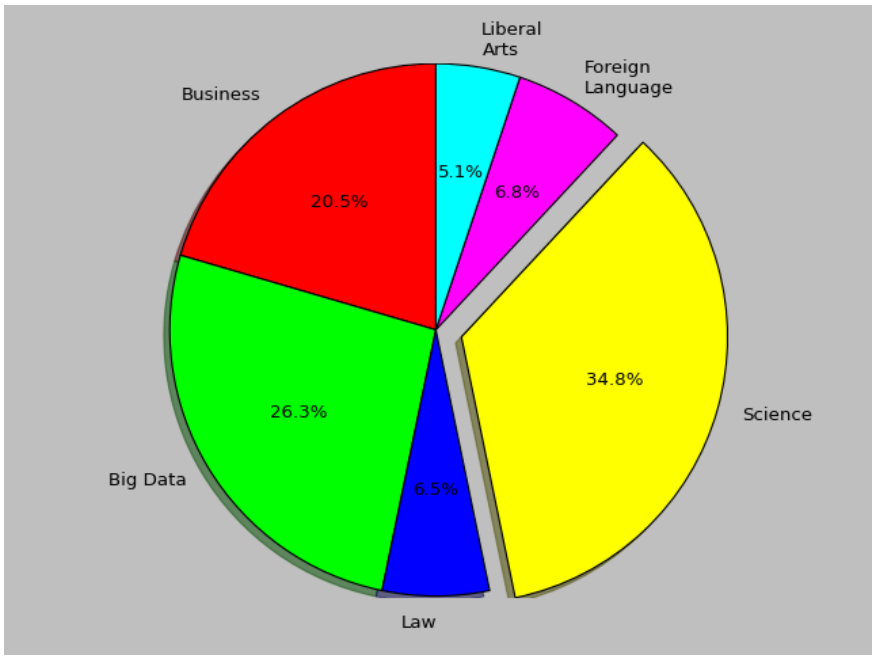
指令說明:

#1 指定最多人修課的學院(第四個學院) 凸出(設定第四個值為 0.1)pie chart 一點。

#2 第四個學院凸出(explode=explode),pie chart 設陰影(shadow=True),一開始從 90^0 的地方開始。

#3 將 X 軸和 Y 軸的單位長設定長度相同。

此程式執行結果如下:



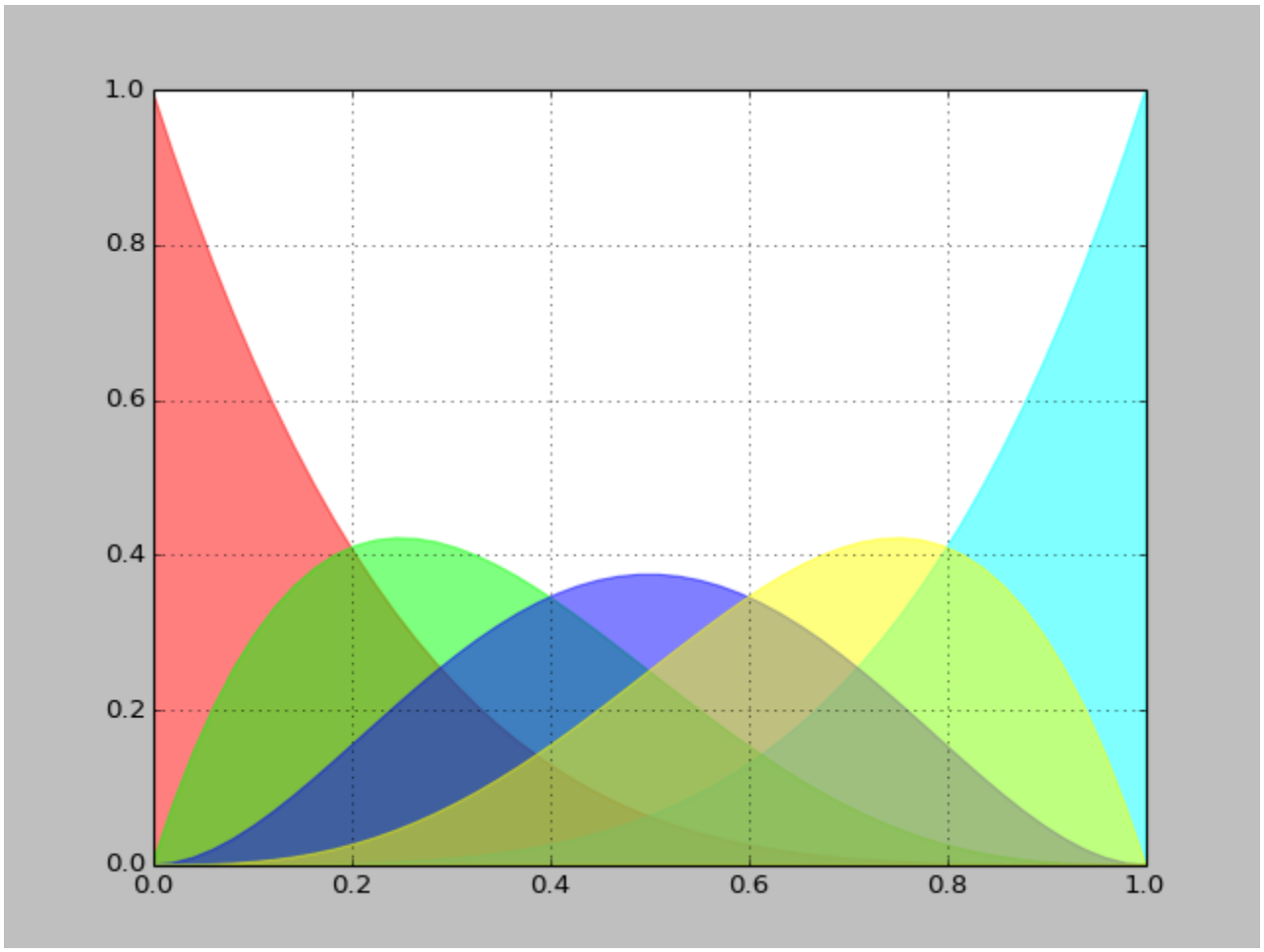
與前一途相比，注意到紅色的位置不大相同，黃色(第四個學院)的扇形凸出，且圖形有陰影。同時，橢圓形也改變成了圓形。

fill

範例: Bernstein Polynomial

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,1)
y0 = (1-x)**4
y1 = 4*(1-x)**3*x
y2 = 6*(1-x)**2*x**2
y3 = 4*(1-x)*x**3
y4 = x**4

plt.fill_between(x,0,y0, color=(1,0,0), alpha=0.5)
plt.fill(x,y1,color=(0,1,0), alpha = 0.5)
plt.fill(x,y2,color=(0,0,1), alpha = 0.5)
plt.fill(x,y3,color=(1,1,0), alpha = 0.5)
plt.fill_between(x,0,y4,color=(0,1,1), alpha = 0.5)
```

```
plt.grid(True)
plt.show()
```



3D Visualization

數的運算

Pi 的計算

我們介紹三種不同的方法來計算 Pi

（1） 祖沖之

（2） Leibniz

write something here

```
def pi(n):
    total = 0
    for i in range(1, 2*n, 2):
```

```
        if i%4 ==1:
            total = total + 1/i
        else:
            total = total - 1/i
    return 4*total

for i in [100, 1000, 10000, 100000, 1000000]:
    print("Compute pi to", i,"-th term using Leibniz formula is equal to ", pi(i))
```

```
Compute pi to 100 -th term using Leibniz formula is equal to  3.1315929035585537
Compute pi to 1000 -th term using Leibniz formula is equal to  3.140592653839794
Compute pi to 10000 -th term using Leibniz formula is equal to  3.1414926535900345
Compute pi to 100000 -th term using Leibniz formula is equal to  3.1415826535897198
Compute pi to 1000000 -th term using Leibniz formula is equal to  3.1415916535897743
>>>
```

(3) Monti Caro

Monte Carlo Methods 使用重複的隨機樣本來模擬複雜的數學或物理系統。給定一發生狀況的事件機率 p，程式來模擬所發生的狀況。我們使用隨機數來模擬發生的狀況，當事件所發生的次數除以隨機發生的狀況數就會逐漸趨近於機率 p。我們如何使用 Monte Carlo Method 來趨近π?

考慮一個半徑為 1 的圓(r=1)，假設其圓心在原點。同時，我們想像在他的外面包了一個正方形，如下圖。我們知道圓的面積為π($\pi r^2=\pi$ if r=1)，我們隨機取在方形內部的點，此點在圓內的機率 p 為:

$$p = \frac{\text{圓面積}}{\text{方形面積}} = \frac{\text{在圓內的點數}}{\text{在方形內的點數}}$$

因此，我們可以使用隨機點(x,y)，其中$-1 \leqq x \leqq 1$，$-1 \leqq y \leqq 1$，來測試一點是否在圓裡面($x^2+y^2 \leqq 1$)。我們想使用 100, 1000, 10000, 100000, 1000000 隨機點來趨近π，程式如下:

```
import random
def MonteCarlo(n):
    inp = 0
    for i in range(n):
        x=random.uniform(-1,1)
        y=random.uniform(-1,1)
#   the same as: if math.sqrt(x*x+y*y)<=1:
        if x*x+y*y<=1:
            inp = inp + 1
    return 4*inp/n
```

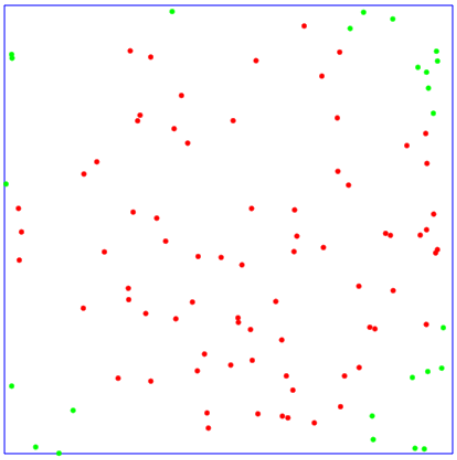
```
for i in [100, 1000, 10000, 100000, 1000000]:
    print("Pi simulates by Monte Carlo method ", i, "points is equal to ", MonteCarlo(i))
```

程式執行的結果如下:

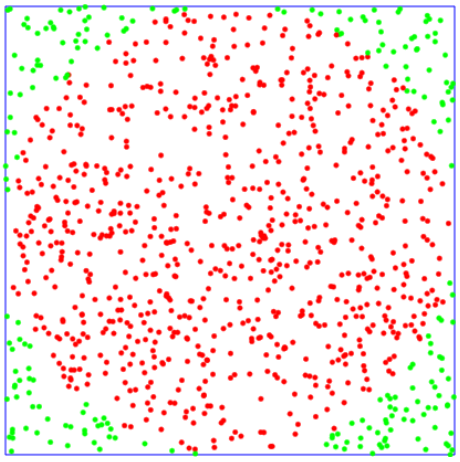
```
Pi simulates by Monte Carlo method  100 points is equal to  3.24
Pi simulates by Monte Carlo method  1000 points is equal to  3.2
Pi simulates by Monte Carlo method  10000 points is equal to  3.1516
Pi simulates by Monte Carlo method  100000 points is equal to  3.14156
Pi simulates by Monte Carlo method  1000000 points is equal to  3.140784
>>>
```

注意到當隨機所取得的點數越多，所得的值越接近真實的值。

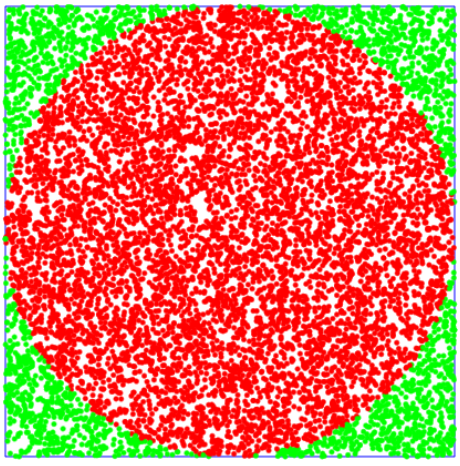
我們可以從圖形來看，以下個別是將 100, 1000, 10000, 100000 個隨機點顯示在螢幕上的情況，其中，旁邊的藍框範圍是-1≦x≦1，-1≦y≦1，紅色的點表示在圓內，藍色的則是在圓外。



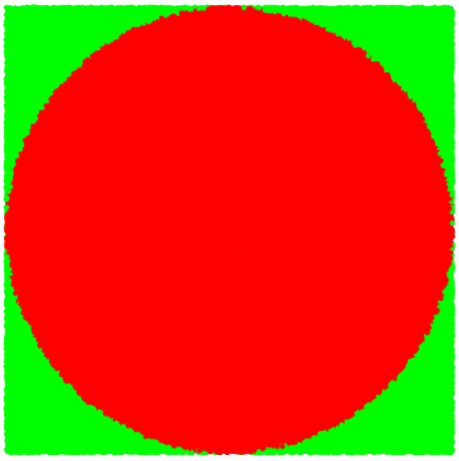
（a）100 個隨機點



（b）1000 個隨機點



（c）10000 個隨機點



（d）100000 個隨機點

初中　gcd

```
def gcd(a,b):
```

```
    if b==0:
        return a
    else:
        return gcd(b, a%b)
```

高中 chinese remainder theorem

Map

魔劍高階系列

理工科

vol 1: 資料與指令初階

vol 2: 趣味程式初階

vol 3: 整數的運算、實數的運算、虛數的運算(Class)

vol4: 幾何程式使用、幾何程式設計

vol5: 科學計算設計

vol6: 科學計算使用

vol7: 數學與幾何設計

vol8: 數學與幾何使用

vol9: 代數的設計與使用

文法商:

文字的處理(Dictionary)

Data Visualization

週	校本(理): Program your idea	城區(商、巨資): Program your data
1-3	Python Basic (含 installation)	Python Basic (含 installation)
4	Visualization (Geometry)	Data Structure 1: List, Dictionary
5	Curves	Data Structure 2: Tree
6	Recursive	Data Structure 3: Class
7	Matrix Computation(linalg)	Computational Thinking and Algorithm
8	Surface Display	Open Data
9	Animation(Visual)	Open Data
10	Morphing(Visual)	Data Display: matplotlib(2D)
11	Simulating(Visual)	Data Display: matplotlib(2D)
12	L-system(繪圖與植物)	Data Display: Seaborn(3D)
13	地球科學 project	Data display: Seaborn(3D)
14	Physical Project	Data Display: may
15	Chemistry Project(Perfect Gas)	Data Display: may
16	DNA project	Project: Data Access + Analysis + Display
其他	離散數學(C and P)	

Form: https://tw.answers.yahoo.com/question/index?qid=20110125000015KK01933 (2015/12/11)
劉徽《九章算術》是中國古代流傳下來的最早也是最重要的數學著作，幾乎集中了當時的全部數學知識。劉徽全面論述了《九章算術》所載的方法和公式，指出並糾正了其中的錯誤，在數學方法和數學理論上做出傑出貢獻，成為中國古代數學理論的奠基者。他的主要貢獻有：創立割圓術、運用樸素的極限思想證明圓面積公式及計算圓周率（π），得到π=157/50、3927/1250 兩個近似值；發展了天文觀測中的重差術，提出重表法、連索法、累距法三種基本方法；重視邏輯推理，同時又注意幾何直觀的作用，採取「析理以辭，解體用圖」的注釋方法；發展了「率」的理論、齊同原理和出入相補原理；提出劉徽原理，並用極限思想對之作了證明；在求體積問題上指出《九章算術》中球體積公式的錯誤，設計了「牟合方蓋」（直徑相同的兩個正交圓柱的公共部分），為日後祖（日恒） 原理的建立指明了方向。劉徽還創造了解線性方程組的互乘相消法；在中國第一次提出不定方程問題；建立了等差級數前 n 項和的公式；改進了許多問題的解法。經過劉徽注釋的《九章算術》影響深遠，支配中國數學的發展 1000 多年，成為東方數學的代表作。

在幾何方面，提出了"割圓術"，即將圓周用內接或外切正多邊形窮竭的一種求圓面積和圓周長的方法．他利用割圓術科學地求出了圓周率π=3.14 的結果． 他用割圓術，從直徑為 2 尺的圓內接正六邊形開始割圓，依次得正 12 邊形、正 24 邊形……，割得越細，正多邊形面積和圓面積之差越小，用他的原話說是“割之彌細，所失彌少，割之又割，以至於不可割，則與圓周合體而無所失矣。” 他計算了 3072 邊形面積並驗證了這個值．劉徽提出的計算圓周率的科學方法，奠定了此後千餘年中國圓周率計算在世界上的領先地位． 劉

儲列
數列
文字列
字串列