python 程式設計

第 九 講

函式：function

函式

- 函式：將部份程式碼集結一起完成某功能
- 語法

```
def  fn( arg1 , arg2 , ... ) :
    body
```

- 範例：階乘函式

```
def  factorial( n ) :
    """計算 n 階乘"""
    p = 1
    for i in range(2,n+1) : p *= i
    return p
```

  - 函式說明：函式內以三個雙(單)引號夾住的字串(可跨行)，可當成函式的說明文字，以「函式名稱.__doc__」可得到此字串，即 **factorial.__doc__**
  - 回傳資料：使用 **return arg** 回傳 **arg** 值並離開函式，可有多個 **return**。若無 **return** 則函式回傳 **None**
  - 函式可重複定義，程式執行僅會使用最後更新的函式

函式參數設定 (一)：位置對應

- 參數位置對應：依參數次序一一對應

```
def  power( a = 10 , n = 1 ) :
    p = 1
    for i in range(n) : p *= a
    return p
```

可以下列方式使用：

函式參數設定	運算結果	說明
power(3,2)	9	a = 3 , n = 2
power(3)	3	a = 3 , n = 1(預設值)
power()	10	a = 10(預設值) , n = 1(預設值)

函式參數設定 (二)：位置對應

- 預設值的設定順序由末尾往前設定

```
# 錯誤：末尾的 n 沒有設定預設值
def  power( a = 10 , n ) :
     ....

# 正確：預設值由參數列末尾位置往前設定
def  power( a , n = 1 ) :
     ....
```

函式要在使用前定義 (一)

- 串列元素 n 次方：無回傳

```
def  powers( foo , n = 1 ) :          # 先定義函式
    for i in range(len(foo)) :
        foo[i] = foo[i]**n
    return

a = [2,3]                             # 後使用
powers(a,4)                           # a = [16,81]
```

- 串列元素 n 次方：有回傳

```
def  powers( foo , n = 1 ) :          # 先定義函式
    return [ c**n for c in foo ]

a = [2,3]
b = powers(a)                         # a = [2,3] b = [2,3]
c = powers(a,3)                       # a = [2,3] c = [8,27]
```

函式要在使用前定義 (二)

- 倒裝寫法：大型程式的開發方式

  將主要程式執行步驟寫於檔案前端的 **main** 函式內，其餘函式撰寫於後，最後在檔案末尾執行 **main** 函式。例如：

```
def main() :                                    # (1) 定義主函式 main()
    while True :                                # (2) 程式步驟寫在主函式內
        n = int( input("> ") )
        for i in range(1,n+1) :
            a = list(range(0,i))
            print( i , powers(a,2) )
def  powers( foo , n = 1 ) :                    # (3) 其餘函式定義於主函式之後
   return [ c**n for c in foo ]
main()                                          # (4) 最後執行主函式
```

輸出：

```
> 3
1 [0]
2 [0, 1]
3 [0, 1, 4]
```

```
> 4
1 [0]
2 [0, 1]
3 [0, 1, 4]
4 [0, 1, 4, 9]
```

函式參數設定：名稱對應

- 參數名稱對應：不管設定參數次序，直接以名稱傳遞數值

```
# 計算年齡
def  age( byear , cyear = 2017 ) :
     return  cyear - byear
```

函式參數設定	運算結果	說明
age(byear=2000)	17	byear = 2000, cyear = 2017(預設值)
age(cyear=2016,byear=2000)	16	byear = 2000, cyear = 2016
age(cyear=2016)	錯誤	byear 沒有設定

星號參數 (一)

- 星號參數：傳遞不等數量參數到函式成為串列或字典
- 單星號成常串列，雙星號成字典
- 星號式子須在參數末尾，不可同時有一個以上的單星號或雙星號式子
- 單星號範例

```
# args 是常串列(tuple)
def sum( n , *args ) :
    s = n
    for c in args : s += c
    return s
```

函式參數設定	運算結果	說明
sum()	錯誤	n 沒有指定
sum(2)	2	n=2 , args=()
sum(2,3)	5	n=2 , args=(3,)
sum(2,3,4)	9	n=2 , args=(3,4)

星號參數 (二)

- 雙星號範例

```
def  member( name , **rec ) :
    print( name , ":" )
    for  k , v in rec.items() :
        print( k , ":" , v , sep="" )
```

函式參數設定	輸出	說明
member("Amy")	Amy : 	name = "Amy" rec = {}
member("Amy","Tom")	錯誤	無法接收 "Tom"
member("John",age=20)	John : age:20	name = "John" rec = {'age':20}
member("Tom",age=24, weight=55)	Tom : age:24 weight:55	name = "Tom" rec = {'age':24, 'weight':55}

星號參數 (三)

- 混合單星號與雙星號

```
def  member( name , *friends , **rec ) :
    print( name , ":" , friends )
    for k , v in rec.items() :
       print( k , ":" , v , sep="" )
```

函式參數設定	輸出	說明
member(”Amy”)	Amy : ()	name = 'Amy' friends = () rec = {}
member("Amy","Lee","Tom")	Amy : ('Lee','Tom')	name = 'Amy' friends = ('Lee','Tom') rec = {}
member("Amy",age=24)	Amy : () age:24	name = 'Amy' friends = () rec = {'age':24}
member("Amy","Bob","Joe", age=24,weight=50)	Amy : ('Bob','Joe') age:24 weight:50	name = 'Amy' friends = ('Bob','Joe') rec = {'age':24, 'weight':50}
member("Amy",age=24,"Tom")	錯誤	參數順序錯誤

拆解串列或字典傳遞參數 (一)

- 單星號拆解串列傳入函式:

```
def sum(a,b) :
    s = 0
    for i in range(a,b+1) : s += i
    return s

# 計算 2+3+4
print( sum(2,4) )              # a = 2  b = 4  印出: 9
no = (2,4)
print( sum(no[0],no[1]) )      # 同上

print( sum(*no) )              # 同上, 單星號拆解串列, 依次對應到函式參數
print( sum(*(2,4)) )           # 同上

print( sum(no) )               # 錯誤, sum 需要兩個參數
```

拆解串列或字典傳遞參數 (二)

- 雙星號拆解字典傳入函式：字典的索引需與函式參數名稱一樣

```
def  avg( math , phy , eng=0 ) :
    if eng == 0 :
        return (3*math + 2*phy) // 5
    else :
        return (3*math + 2*phy + eng) // 6

Tom = { 'phy':60 , 'math':90 }
Amy = { 'phy':60 , 'math':90 , 'eng':30 }

# math , phy , eng = 90 , 60 , 0
print( avg(90,60) )                    # 78
print( avg(**Tom) )                    # 同上，雙星號拆解字典

# math , phy , eng = 90 , 60 , 30
print( avg(90,60,30) )                 # 70
print( avg(**Amy) )                    # 同上，雙星號拆解字典
```

拆解串列或字典傳遞參數 (三)

- 若忘記雙星號或者輸入的字典索引與函式參數名不同, 都會造成錯誤

```
Lee = { 'phy':60 , 'math':90 }
Joe = { 'phy':60 , 'math':90 , 'chm':30 }

print( avg(Lee) )              # 錯誤, 忘了雙星號
print( avg(**Joe) )            # 錯誤, avg 函式沒有 chm 參數名
```

函式內的變數 (一)

- 局部變數：定義於函式內的變數都在函式內部使用，不對外影響

```
def  add_n( num , n ) :
    num += n                    # num 與 n 都為局部變數，不對外影響
    return  num

num = 3                         # num = 3
print( add_n(num,5) )           # 印出 8 , num 仍不會改變
print( n )                      # 錯誤，n 尚還未定義
```

函式內的變數 (二)

- 全域變數：在函式內部的變數前若有 **global** ，代表此變數為整個程式碼共用

```
# 印出半徑由 1 到 9 的圓面積

def  main() :
    global pi                                  # pi 為全域變數
    pi = 3.14
    for r in range(1,10) :
        print(circle_area(r))

def  circle_area( rad ) :
    global pi                                  # pi 為全域變數
    return rad*rad*pi

main()
```

可變更與不可變更型別 (一)

- python 提供的基本型別，有些可在原位址變更內容，有些則需另找空間儲存新資料，前者稱為 mutable 型別，後者稱為 immutable 型別，例如：以下的 a 為整數，當 a 更改數值時，a 的位址也會隨之變更，不會保留在原位址。

```
>>> a = -10
>>> id(a)                    # 原 a 位址：139977189736752
>>> a = 1000
>>> id(a)                    # 新 a 位址：139977327589520
```

但若是串列型別，變更資料並不會影響儲存位址，串列為 mutable 型別。

```
>>> b = [500,1000]
>>> id(b)                    # 輸出地址：139977189760136
>>> b.append(2000)
>>> id(b)                    # 地址不變，同上
```

可變更與不可變更型別 (二)

若更改 **b[0]** 元素值，因為 **b[0]** 為整數型別，更改後 **b[0]** 會移到新空間儲存資料。但原有串列為 **immutable** ，更改元素並不會影響串列位址。

```
>>> id(b[0])                    # b[0] 原位址：139977327589520
>>> b[0] = 600
>>> id(b[0])                    # b[0] 新位址：139977189736784
>>> id(b)                       # b   位址不變：139977189760136
```

可變更與不可變更型別 (三)

- **python** 設定的可變更 **(M)** /不可變更 **(I)** 型別

type	型別	M/I	樣式
int	整數	I	23, -761
float	浮點數	I	1.2, -234e-2
str	字串	I	"hello", "123"
boolean	布林數	I	True, False
complex	複數	I	1-1j, -2.7+3j
list	串列	M	[3], [7,"abc",3+2j]
tuple	常串列	I	(4,), ("ab",2)
set	集合	M	{8,2}, set()
frozenset	凍集合	I	frozenset([8,2])
dict	字典	M	{"貓":'cat', "狗":'dog'}
bytearray	位元組序列	M	bytearray(b'cat')
bytes	常位元組序列	I	b'cat', bytes([99,97,116])

函式參數值的更改 (一)

- **immutable** 型別參數在函式內任何變更數值，將會更換儲存空間，不會影響原傳入物件值。
- **immutable** 型別參數傳入函式內部時，參數資料並沒有複製，僅是多了一個名稱可取用此參數
- **mutable** 型別參數在函式內外都佔用同樣空間，在函式內的更動會影響原傳入物件

函式參數值的更改 (二)

```
def  change_vals( a , b ) :
    print(a)                          # 列印傳進來的 a 參數值
    a = 10                            # a 另找空間儲存 10
    b.append(5)                       # 串列增加一個元素

# foo 為整數(I), bar 為串列(M)
foo , bar = 3 , [4,9]                 # foo = 3 , bar = [4,9]

# 在函式內變更兩參數資料
change_vals(foo,bar)

# foo 整數不變, bar 串列會變動
print( foo , bar )                    # foo = 3 , bar = [4,9,5]
```

❖ python 無法透過函式參數傳遞來修改 immutable 參數

函式參數值的更改 (三)

- 在函式中，串列若指定到新串列，並不會影響原來傳入串列：

```
def  change_lists( a , b ) :
   a = [3]                          # a 改指定到新串列
   b[0] = [5]                       # 修改 b[0] 元素

foo , bar = [9] , [2,7]
change_lists( foo , bar )

print( foo , bar )                  # foo = [9] , bar = [5,7]
```

函式參數值的更改 (四)

- 大樂透程式

  以下集合物件 **fset** 儲存不重複的樂透號碼，**fset** 物件透過大樂透號碼函式產生六個號碼存入集合物件，程式利用集合物件可在函式中被修改的特性。

```
from random import *

def  lottery( n , bset ) :
    while True :
        bset.add( randint(1,49) )
        if len(bset) == n : return
fset = set()
lottery( 6 , fset )

print( " ".join( map( str , sorted(fset) ) ) )
```

進階排序 (一)：設計函式比大小

- **foo.sort(key=fn)**：**foo** 串列依 **fn** 函式設定排列，沒有回傳
- **sorted(foo,key=fn)**：回傳 **foo** 串列依 **fn** 函式設定排列，但 **foo** 保持不變
- 使用到一般函式設計複雜的排序問題

```
# 比較日期：先年後月，由小到大
def  by_small_date(x) :
    s = x.split('/')
    return ( int(s[1]) , int(s[0]) )

# 比較日期：先年後月，由大到小
def  by_big_date(x) :
    s = x.split('/')
    return ( -int(s[1]) , -int(s[0]) )

dates = [ '2/2010' , '9/2010' , '7/2009' , '7/2008' ]
```

進階排序 (二)：設計函式比大小

```
# 印出日期由小到大的排序結果
print( sorted(dates,key=by_small_date) )

# 印出日期由大到小的排序結果
print( sorted(dates,key=by_big_date) )
```

分別輸出：

```
['7/2008', '7/2009', '2/2010', '9/2010']
['9/2010', '2/2010', '7/2009', '7/2008']
```

eval 求值函式 (一)

- **eval(foo): 對 foo 字串式子作運算**

```
>>> eval( "4 + 2 * ( 6 / 3 )" )
8.0

>>> a = eval( "[ 2*x for x in range(3)]" )
>>> a
[0, 2, 4]
```

- **eval 與 input 結合可成一個計算器**

```
while True :
    print( eval( input("> ") ) )
```

執行後得

```
 >>> 3+4
 7
 >>> 2*(4-1)*8
 48
```

eval 求值函式 (二)

- 使用 **eval** 讀取多筆資料
  - 資料以逗點分離

```
>>> a , b , c = eval( input("> ") )
> 3 , "cat" , 2.8                    # 字串要有單(雙)引號

>>> a , b , c
(3, 'cat', 2.8)
```

  - 資料以空格分離

```
>>> x , y , z = map( eval , input("> ").split() )
> 5 "dog" 2.8

>>> x , y , z
(5, 'dog', 2.8)
```

❖ 以上的輸入資料通常不會是同一種型別

產生器函式 (一)：generator function

- 產生器函式：函式使用來 **yield** 送出運算過程資料，資料送出後函式仍繼續執行

```
# 計算前 n 項的等比數列，a 為初值，r 等比數
def  gparray( a , r , n ) :
    for i in range(n) :
        yield a
        a *= r

for x in gparray(3,2,5) : print(x,end=" ")

print()
print( "->".join( [ str(x) for x in gparray(2,3,6) ] ) )
```

輸出：

```
3 6 12 24 48
2->6->18->54->162->486
```

❖ 產生器函式使用 yield 將資料送出，函式繼續執行。但一般函式在執行 return 後，函式隨即中止。

產生器函式 (二)

- 產生器函式內 **yield** 送出的資料可使用 **list** 轉成串列

```
>>> foo = list( gparray(2,3,5) )
>>> foo
[2, 6, 18, 54, 162]
```

❖ 若 yield 送出的資料個數為無限個，則不能使用此種方式

產生器函式 (三)

- 產生器函式可回傳產生器物件，使用 **next** 取得 **yield** 所送出的資料

```
# 無窮迴圈用以產生初值為 a, 等比為 r 的等比數列
def  gparray2( a , r ) :
    while True :
        yield a
        a *= r

# itr 為產生器物件
itr = gparray2(2,3)

# 產生前三項
print( "->".join( [ str(next(itr)) for i in range(3) ] ) )

# 接續產生四項
print( "->".join( [ str(next(itr)) for i in range(4) ] ) )

# 計算等比數列的前十項和:
itr2 = gparray2(3,2)
print( sum( [ next(itr2) for i in range(10) ] ) )
```

輸出:

```
2->6->18
54->162->486->1458
3069
```

微分方程式數值求解 (一)

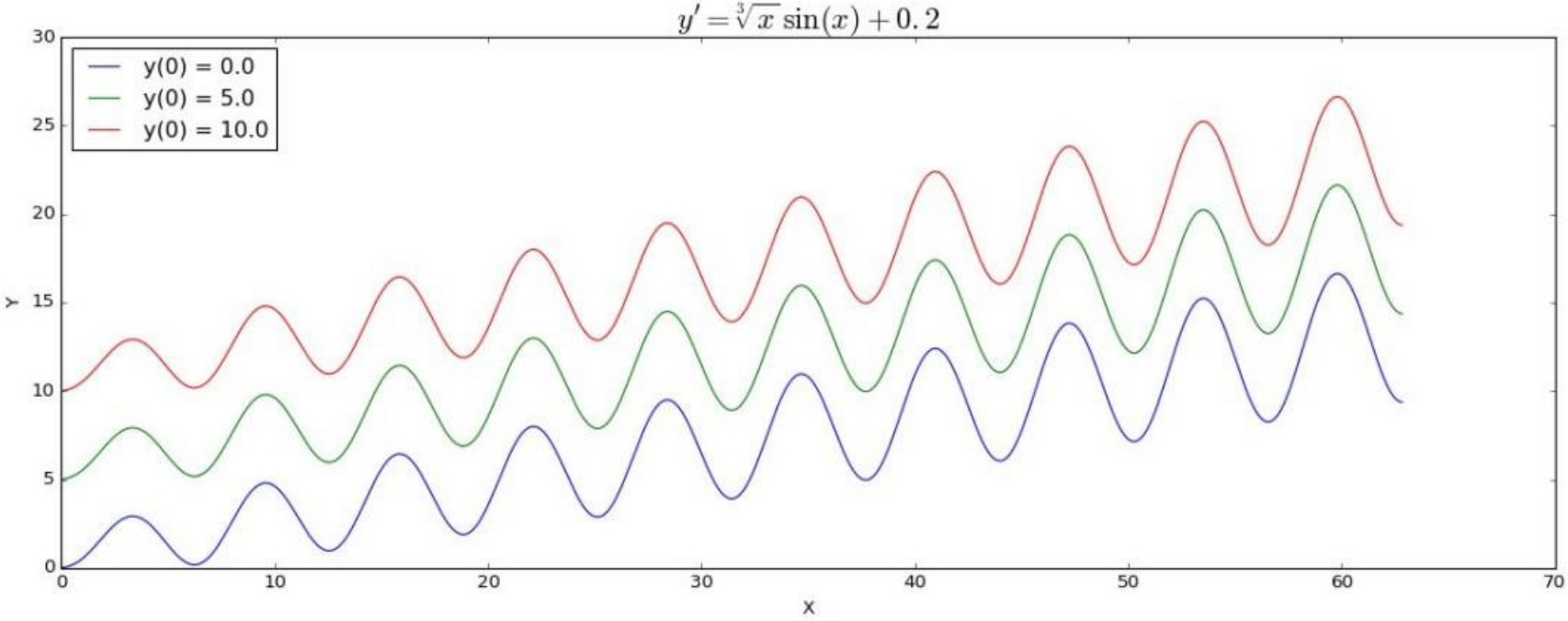
- 一次微分方程式: $y' = \sqrt[3]{x}\sin x + 0.2$ 求解

  這裡使用 Euler method 來求解微分方程式，微分方程式在 $x = x_i$ 為 $y_i' = f(x_i, y_i)$，y_i 代表 $y(x_i)$，微分使用差分公式 $y_i' = \frac{y_{i+1} - y_i}{\triangle x}$ 代入微分方程式後得:

  $$y_{i+1} = y_i + \Delta x\ f(x_i, y_i) \qquad i = 0, 1, 2, \ldots$$

  以上公式須設定起始條件，這裡讓 $y_0 = c$ 。在計算上，Δx 經常設定為固定值，也就是 $\{x_i\}$ 呈現等距分佈。一般來說，使用 Euler method 來計算微分方程式的數值解是很簡單，但缺點為精度偏低。

微分方程式數值求解 (二)


$y' = \sqrt[3]{x}\sin(x) + 0.2$
y(0) = 0.0
y(0) = 5.0
y(0) = 10.0
Y
X
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70

微分方程式數值求解 (三)

```
import pylab

#------------------------------ -------
# y' = x**(1/3) sin(x) + 0.2
#
# i.c.  y(0) = val     val = range(0,11,5)
#---------------------------------------
def fn(x) :
    return  (x)**(1/3) * pylab.sin(x)
             + 0.2

# 設定周邊空白為白色
pylab.figure(facecolor='white')

a , b , n = 0 , 20*pylab.pi , 501
dx = (b-a)/(n-1)

# 設定 xs , ys
xs = [ a + i*dx for i in range(n) ]
ys = [None] * n
```

```
for c in range(0,11,5) :

    ys[0] = c

    for i in range(1,n) :
        ys[i] = ys[i-1] + dx * fn(xs[i-1])

    sym = 'y(0) = ' + str(ys[0])

    pylab.plot(xs,ys,label=sym)

# 設定圖形標頭文字
pylab.title(r"$y' = \sqrt[3]{x}\,
            \sin(x) + 0.2$",fontsize=20)

# 設定 x 軸與 y 軸文字
pylab.xlabel('X')
pylab.ylabel('Y')

# 設定各線條圖例位置
pylab.legend(loc='upper left')

pylab.show()
```

大樂透對獎 (一)

- 用程式產生大樂透中獎號碼與十組彩券號碼，印出中獎的號碼個數與號碼，輸出如下：

```
 4 15 16 25 27 37
24 33 39 42 45 48 : 0 -->
 1 12 15 19 24 42 : 1 --> 15
 9 14 25 38 41 46 : 1 --> 25
 2  9 10 33 38 45 : 0 -->
 5 18 23 30 42 47 : 0 -->
 5 25 34 41 44 48 : 1 --> 25
 6 11 13 14 30 41 : 0 -->
 1  3 11 28 38 44 : 0 -->
 5 16 24 31 33 46 : 1 --> 16
 4 15 28 31 37 40 : 3 --> 4 15 37
```

大樂透對獎 (二)

```
from random import *

def main() :

    fset , cset = set() , set()
    lottery( 6 , fset )

    # 樂透中獎號碼
    wset = frozenset(fset)

    print( " ".join(map(lambda x : "{:>2}"
          .format(str(x)),sorted(wset))) )

    # 彩券號碼
    for i in range(10) :

        fset = set()
        lottery( 6 , fset )

        cset = check_num(wset,fset)

        print( " ".join(map(lambda x : "{:>2}"
              .format(str(x)),sorted(fset))) ,
              ':', len(cset) , '-->' , " "
              .join(map(str,sorted(cset))) )
```

```
# 產生樂透號碼
def  lottery( n , bset ) :
   while True :
       bset.add( randint(1,49) )
       if len(bset) == n : return

# 中獎號碼
def  check_num( aset , bset ) :
    return aset.intersection(bset)

# 執行主函式
main()
```

印製年曆 (一)

- 印製年曆需使用求星期幾的公式，但使用公式前需先修改年月數字，此公式以每年的三月為當年開始月份，四月為二月，次年的一、二月為當年十一、十二月，例如：西元 2000 年 1 月 1 日的 Y = 1999, M = 11, D = 1, 西元 1999 年 12 月 31 日的 Y = 1999, M = 10, D = 31，推算公式如下：

$$(Y+[Y/4]-[Y/100]+[Y/400]+[2.6\times M-0.2]+D) \bmod 7$$

這裡的 [x] 為 x 的整數部份，mod 為餘數運算子，公式回傳值介於 0 到 6 之間，分別代表星期日到星期六。

印製年曆 (二)

```
def main() :
    wstrs = ( "Sun" , "Mon" , "Tue" , "Wed" , "Thu" , "Fri" , "Sat" )
    w = 4
    fmt = "{:>" + str(w) + "}"
    while True :
        y = int( input("輸入西元年> ") )
        for m in range(1,13) :
            print( " "*12 , m , "月" )
            for s in wstrs : print( fmt.format(s) , end="" )
            print()
            wday , mdays = weekday(y,m,1) , mondays(y,m)
            print( " "*int(w*wday) , end="" )
            for d in range(mdays) :
                print( fmt.format(d+1) , end= ("" if wday<6 else "\n" ) )
                wday = ( wday + 1 if wday < 6 else 0 )
            print("\n")
```

印製年曆 (三)

```
# 某年是否為閏年
def isleap( y ) :
    return True if y%400 == 0 or ( y%100 and y%4 == 0 ) else False

# 某年某月的日數
def mondays( y , m ) :
    days = [ 31 , 28 , 31 , 30 , 31 , 30 ,
             31 , 31 , 30 , 31 , 30 , 31 ]
    if m == 2 :
        return 29 if isleap(y) else 28
    else :
        return days[m-1]

# 計算某年月日星期幾
def weekday( y , m , d ) :
    ( y , m ) = ( y-1 , m+10 ) if m < 3 else ( y , m - 2 )
    return ( y + y//4 - y//100 + y//400 + int(2.6*m-0.2) + d )%7

# 執行主函式
main()
```

印製年曆 (四)

輸出：

```
> 2000
                                1 月
Sun Mon Tue Wed Thu Fri Sat
                                  1
  2   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  29
 30  31

                                2 月
Sun Mon Tue Wed Thu Fri Sat
                1   2   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  29

                                3 月
Sun Mon Tue Wed Thu Fri Sat
                1   2   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  29  30  31

                                4 月
Sun Mon Tue Wed Thu Fri Sat
                                  1
  2   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  29
 30

                                5 月
Sun Mon Tue Wed Thu Fri Sat
      1   2   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  29  30  31

                                6 月
Sun Mon Tue Wed Thu Fri Sat
                    1   2   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  29  30

                                7 月
Sun Mon Tue Wed Thu Fri Sat
                                  1
  2   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  29
 30  31

                                8 月
Sun Mon Tue Wed Thu Fri Sat
          1   2   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  29  30  31

                                9 月
Sun Mon Tue Wed Thu Fri Sat
                    1   2
  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  29  30

                               10 月
Sun Mon Tue Wed Thu Fri Sat
  1   2   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
 29  30  31

                               11 月
Sun Mon Tue Wed Thu Fri Sat
              1   2   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  29  30

                               12 月
Sun Mon Tue Wed Thu Fri Sat
                    1   2
  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  29  30
 31
```

中文成語筆畫排序 (一)

- 一些中文書末尾的索引通常依照各字的筆畫數由筆畫少排到筆畫多，如果第一個字筆畫一樣，則依次比之後的字。本題讀入筆畫檔與成語檔，依筆畫數將成語由筆畫少排到筆畫多，輸出時連同成語各字的筆畫數一起顯示以資比對，有關筆畫檔格式詳見第七章筆畫範例。

原始成語檔為每列一個成語：

灌夫罵座	郢匠揮斤	千慮一得	愚公移山	一箭雙雕	三人成虎
焚膏繼晷	舟中敵國	及瓜而代	才高八斗	一錢不值	三令五申
無所適從	門可羅雀	口若懸河	指鹿為馬	一飯千金	上行下效
燃膏繼晷	閉月羞花	口蜜腹劍	暴虎馮河	一語成讖	上下其手
班門弄斧	防微杜漸	同歸殊塗	木人石心	一髮千鈞	不值一錢
瓜田李下	馬革裹屍	國色天香	李下瓜田	一鳴驚人	九牛一毛
瓜李之嫌	負荊請罪	塞翁失馬	杜漸防微	一鼓作氣	亡羊補牢
瓜代有期	入木三分	大義滅親	殊途同歸	一敗塗地	人面桃花
畫餅充飢	出人頭地	天香國色	毛皮之附	一曝十寒	七上八下
病入膏肓	刻舟求劍	天涯海角	毛遂自薦	一毛不拔	八面玲瓏
盲人瞎馬	功虧一簣	季布一諾	江郎才盡	三生有幸	
繼晷焚膏	勢如破竹	守株待兔	沉魚落雁	三顧茅廬	
運斤成風	千鈞一髮	弄斧班門	一竅不通	三折其肱	

中文成語筆畫排序 (二)

要輸出的成語排列型式：

1 畫：
一毛不拔 1-4-4-8
一敗塗地 1-11-13-6
一飯千金 1-12-3-8
一鼓作氣 1-13-7-10
一語成讖 1-14-6-24
一鳴驚人 1-14-22-2
一髮千鈞 1-15-3-12
一箭雙雕 1-15-18-16
一錢不值 1-16-4-10
一竅不通 1-18-4-10
一曝十寒 1-19-2-12

2 畫：
七上八下 2-3-2-3
九牛一毛 2-4-1-4
入木三分 2-4-3-4
八面玲瓏 2-9-9-20
人面桃花 2-9-10-7

3 畫：
三人成虎 3-2-6-8
上下其手 3-3-8-4
三令五申 3-5-4-5
及瓜而代 3-5-6-5
三生有幸 3-5-6-8
上行下效 3-6-3-10
亡羊補牢 3-6-12-7
三折其肱 3-7-8-8
口若懸河 3-8-20-8
才高八斗 3-10-2-4
千鈞一髮 3-12-1-15
大義滅親 3-13-13-16
口蜜腹劍 3-14-13-15
千慮一得 3-15-1-11
三顧茅廬 3-21-8-19

4 畫：
木人石心 4-2-5-4
毛皮之附 4-5-3-7
天香國色 4-9-11-6
不值一錢 4-10-1-16
天涯海角 4-11-10-7
毛遂自薦 4-12-6-16

5 畫：
出人頭地 5-2-16-6
瓜代有期 5-5-6-12
瓜田李下 5-5-7-3
瓜李之嫌 5-7-3-13
功虧一簣 5-17-1-18

6 畫：
舟中敵國 6-4-15-11
江郎才盡 6-8-3-14
守株待兔 6-10-9-8
防微杜漸 6-13-7-14
同歸殊塗 6-18-10-13
...

中文成語筆畫排序 (三)

```
def main() :
    global sdict
    # 讀入筆畫檔
    sdict = {}
    with open( "strokes.dat" ) as infile :
        for line in infile :
            ucode , strokes = line.split()
            ch = chr(int(ucode[2:],16))
            sdict[ch] = int(strokes)
    # 讀入成語檔
    idioms = []
    with open("idioms.dat") as infile :
        for line in infile : idioms +=
                             [ line.strip() ]
    # 排序
    idioms.sort( key=by_strokes )
    s1 = 0
    for ws in idioms :
        s2 = sdict[ws[0]]
        if s1 != s2 :
            if s1 : print()
            print( s2 , "畫:" )
        print( ws , "-".join( map
        ( lambda c : str(sdict[c]) ,
          ws ) ) )
        s1 = s2

# 排序標準
def by_strokes( idiom ) :
    global sdict
    return [ sdict[c] for c in idiom ]

# 執行主函式
main()
```

修課時間排序 (一)

讀入一個修課時間檔案，依課程每周第一次上課時間排序，將結果印出來。以下左側為修課時間，每列包含課程名稱與授課時間，右側則為排序後的結果。例如：化學在星期一的第三節是一周中最早上課時間，其次是經濟學在星期二的第五節，其它依此類推。

微積分	四:78	五:56		化學	三:8	一:34
物理	三:12	二:7	---->	經濟	二:56	
化學	三:8	一:34		物理	三:12	二:7
經濟	二:56		---->	國文	三:56	
英文	五:34			體育	四:34	
體育	四:34		---->	微積分	四:78	五:56
國文	三:56			英文	五:34	

修課時間排序 (二)

```
def main() :
    global c2n
    cnum = '一二三四五'
    c2n = dict( [ ( b , a ) for a , b in
                enumerate(cnum) ] )
    with open("schedule.dat") as infile :
        schedules = infile.readlines()
    #依據課程在一周內最早上課時間排序
    schedules.sort( key=by_weekly_earlier_time )
    for s in schedules : print( s.rstrip() )

# 設定排序標準
def  by_weekly_earlier_time( schedule ) :
   global c2n
   course , *csect = schedule.split()
   all = []
   for p in csect :
        # 拆解上課時間
        a , b = p.split(':')
        w = c2n[a]

        # 將此門課所有上課時間以
        # 整數表示
        for c in b :
            s = int(c)
            all.append(w*10+s)

    # 回傳該門課在一周最早上課時間
    # 所代表的整數
    return sorted(all)[0]

# 執行主函式
main()
```