自行車竊案分析

函式 (I)

天啊，怎麼又要學新東西？可以不要嗎？

- 如果你想要
  - 寫出簡潔易懂的程式碼
    - 隱藏實作細節，減輕閱讀負擔
  - 以有效率的方式撰寫與維護程式
    - 減少大量重複冗碼
  - 和伙伴共同開發程式
    - 有效分享彼此專長知識
  - 理解使用第三方套件的方式

我看，還是學一下吧！

為什麼要用函式?

- 寫出簡潔易懂的程式碼
  - 隱藏實作細節，減輕閱讀負擔

```
min_ind = max_ind = 0
data = [int(e) for e in input().split()]
for i in range(1, len(data)):
    if data[i]> data[max_ind]:
        max_ind = i
    elif data[i]< data[min_ind]:
        min_ind = i

data[max_ind], data[min_ind] = data[min_ind], data[max_ind]
```

如果有 max(),min(),和 index()，程式就可以變成 ……

```
data = [int(e) for e in input().split()]
a = data.index(max(data))
b = data.index(min(data))
data[a], data[b] = data[b], data[a]
```

為什麼要用函式?

- 以有效率的方式撰寫與維護程式
  - 減少大量重複冗碼
    - 重覆冗碼的壞處
      - 程式碼變長，難以閱讀。
      - 程式碼散落各地，難以修改維護。

```
...

PrintTable(data)

...

PrintTable(data)
```

為什麼要用函式?

- 和伙伴共同開發程式
  - 有效分享彼此專長知識

```
...

data = remove_outliers(data)
ranked = sorted(data)

...
```

  - 不需理解或陷入標準差、排序等細節，仍可以完成工作。

什麼是函式?

- 函式將一段程式碼包裝成一個單元，程式員呼叫函式以執行該段程式碼。
  - 該段程式碼會共同完成一個明白、特定的功能。
  - 函式的輸入/參數列代表執行該段程式碼需要的資料。
  - 函式的輸出/回傳值代表執行該段程式碼所得的結果。

```
def len(data):
    ans = 0
    for e in data:
        ans += 1
    return ans

d = [1, 2, 10, 4, 88]
print(len(d))
```

```
5
```

最後一行涉及兩個函式呼叫：第一次呼叫 len()，在傳入 d 並計算後，回傳 5；第二次呼叫 print()，將 5 傳入並列印在螢幕上。

如何撰寫函式

看例子學最快

基本型
(無傳入傳出)

```
def hello():
    print('Hello Python')

hello()
```

```
Hello Python
```

和 if、for 句型一樣，
函式裡的程式碼要內縮。

傳入資料

```
def echo(word):
    print('Hello ' + word)

echo('NTNU')
```

```
Hello NTNU
```

傳入傳出

```
def sum(a, b, c):
    return a+b+c

print(sum(10, 9, 8))
```

```
27
```

函式的進入與返回

- 今天之前我們寫的程式
  - 由上而下執行緊貼著最左邊對齊的程式碼。

```
print('第一行')
print('第二行')
if False:
    print('以 if 來看，這句一開始還是緊貼著最左邊的')
print('第三行')
```

```
第一行
第二行
第三行
```

- 今天之後我們寫的程式（有了函式）
  - 沒有被呼叫的函式就沒有作用。

```
print('第一行')
print('第二行')

def f():
    print('沒人用我，就不會印出這行')

print('第三行')
```

```
第一行
第二行
第三行
```

函式的進入與返回

- 當函式被呼叫，程式會進入到函式內部去執行其程式碼。

```
print('開始')

def echo(word):
    print('現在在 echo() 函式裡面')
    print('Hello ' + word)

echo('1')
print('------')
echo('2')
print('結束')
```

```
開始
現在在 echo() 函式裡面
Hello 1
------
現在在 echo() 函式裡面
Hello 2
結束
```

函式的進入與返回

return 會結束函式

```
def f():
    print('進來 f()')
    return
    print('這一行跑不到')

f()
```

```
進來 f()
```

```
def avg(data):
    #if data:     #這個寫法也是判斷 data 非空
    if len(data)!=0:
        return sum(data)/len(data)
    else:
        return 0

print(avg([]))
print(avg([1, 3, 4]))
```

```
0
2.6666666666666665
```

return 可以傳回單值、list 或多個值 (tuple)

```
def max_min(data):
    return max(data), min(data)

M, m = max_min([3, 4, 1, 2])
print(M, m)
```

```
4, 1
```

```
def remove_zeroes(data):
    return [e for e in data if e!=0]

data = remove_zeroes([3, 2, 0, 3, 9, 0])
print(data)
```

```
[3, 2, 3, 9]
```

如果函式 f() 沒有回傳值，a = f() 會令 a 的值為 None。
試試這行：print(print())。

函式的可用點

- 有些時候呼叫函式會發生「素未謀面」的錯誤。

```
f()

def f():
    print('f()')
```

程式在呼叫 `f()` 的時候，還沒看過 `f()` 函式的定義。它就會說：

```
NameError: name 'f' is not defined
```

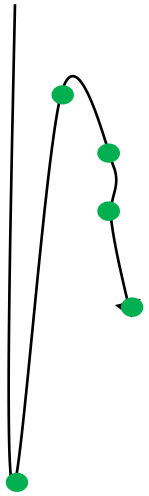
```
def f():
    print('f()')

f()
```

程式在呼叫 `f()` 的時候，已經看過 `f()` 函式的定義。

一切順利。☺

函式的可用點

- Python 的「見過」與否是以執行流程來定，不是單純以程式碼的位置來定。



```
def f():
    print('f()')
    g()

def g():
    print('g()')

f()
```

雖然以程式碼位置來看，g() 函式的定義在 g() 的呼叫點之後，但以程式流程來看，呼叫 g() 的時候，已經看過 g() 的定義了。

所以這個程式執行起來是沒問題的。

函式的可用點

- 常見的技法是定義一個主函式 main()，讓它作所有事，然後在程式檔的最下面呼叫它。

```
def main():
    f()
    print('.')
    g()

def f():
    print('f()')
    g()

def g():
    print('g()')

main()
```

```
f()
g()
.
g()
```

有人說有些語言要求 define higher，而 Python 要求 *define earlier*。

函式的可用點

- 在 Python 中，沒實際發生就不算數。

```
def f():
    print('f()')
    g()

n = int(input())
if n==3:
    f()
print('一切順利')

def g():
    print('g()')
```

輸入 4，程式輸出「一切順利」。
（沒有呼叫 f()。）

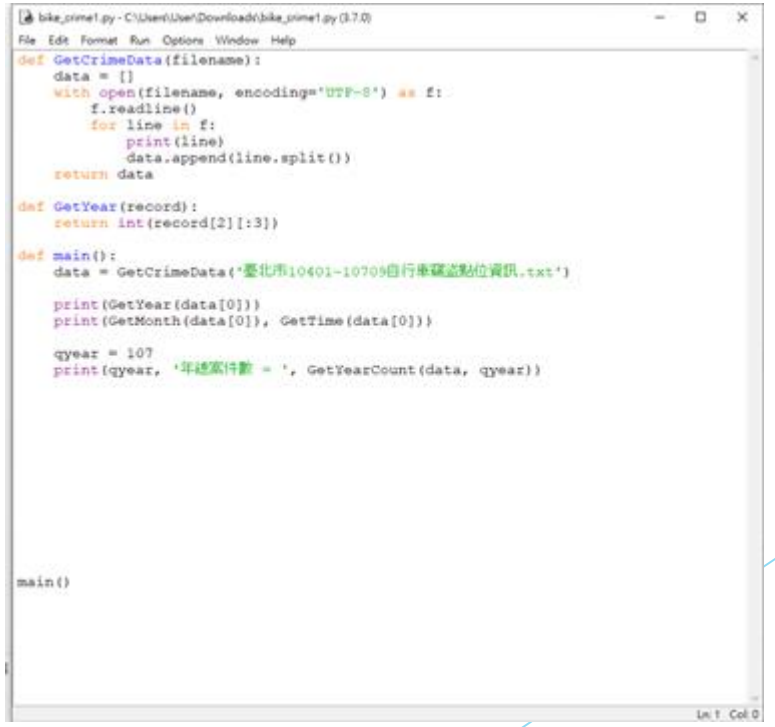
```
4
一切順利
```

輸入 3，程式發生錯誤。
（呼叫 f()，而程式執行至此還沒看過 g()。）

```
3
f()

NameError: name 'g' is not defined
```

任務：自行車竊案統計

- 自 moodle 平台取得兩個檔案 (右鍵另存新檔於桌面)
  - *臺北市 10401-10709 自行車竊盜點位資訊.txt*
    - 來源：https://data.taipei/dataset/detail/metadata?id=5c5e9e13-9803-47c0-bbd2-1a4b3c11c49b
  - *bike_crime1.py*



任務：自行車竊案統計

任務說明

- 以指定欄位的指定條件進行統計
  - 統計 105, 106, 107年的竊案總數。
  - 統計 2, 8, 12月的竊案總數。
- 撰寫與應用函式
  - 撰寫取出竊案發生月份的函式。
  - 撰寫查詢某年發生竊案總數的函式。
  - 撰寫查詢某月發生竊案總數的函式。

任務：自行車竊案統計

bike_crime1.py

```
def GetCrimeData(filename):
    data = []
    with open(filename, encoding='UTF-8') as f:
        f.readline()
        for line in f:
            print(line)  #加上井號註解就不會列印出所有內容
            data.append(line.split())
    return data

def GetYear(record):
    return int(record[2][:3])

def main():
    data = GetCrimeData('臺北市10401-10709自行車竊盜點位資訊.txt')

    print(GetYear(data[0]))
    print(GetMonth(data[0]), GetTime(data[0]))

    qyear = 107
    print(qyear, '年總案件數 = ', GetYearCount(data, qyear))

main()
```

GetCrimeData() 會開啟 filename 檔名之文字檔，將內容讀取成為一個（二維）列表並回傳。

GetYear() 會取出一筆竊案資料的發生年份。

直接執行範例程式，會列印出檔案內容。因為未定義 GetMonth()、GetTime() 和 GetYearCount()，程式執行到中間便停止。
你的任務就是在此處完成上述三個函式。

任務：自行車竊案統計

臺北市10401-10709自行車竊盜點位資訊 - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

```
編號 案類 發生(現)日期 發生時段 發生(現)地點
1 自行車竊盜 1020630 13~15 台北市松山區三民路151~180號
2 自行車竊盜 1040101 16~18 台北市大安區住安里四維路124巷1~30號
3 自行車竊盜 1040104 07~09 台北市大安區敦煌里仁愛路四段266巷1~30號
4 自行車竊盜 1040106 13~15 台北市萬華區國興路1~30號
5 自行車竊盜 1040108 07~09 台北市北投區清江里三合街二段481~510號
6 自行車竊盜 1040108 16~18 台北市松山區延壽街391~420號
7 自行車竊盜 1040109 04~06 台北市南港區三重里南港路一段南港展覽館站
8 自行車竊盜 1040109 13~15 台北市內湖區金湖里[illegible]路三段31~60號
```

bike_crime1.py

```
def GetCrimeData(filename):
    data = []
    with open(filename, encoding='UTF-8') as f:
        f.readline()  #略過第一行
        for line in f:
            data.append(line.split())
    return data
```

GetCrimeData() 會開啟 filename 檔名之文字檔，將內容讀取成為一個（二維）列表並回傳。

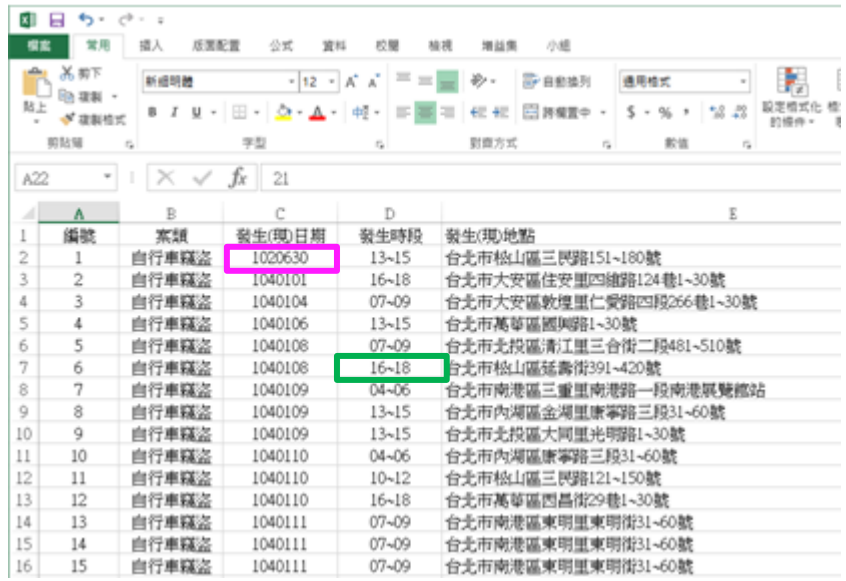
(上星期學的沒有白費 XD)

```
[
['1', '自行車竊盜', '1020630', '13~15', '台北市松山區三民路151~180號'],
['2', '自行車竊盜', '1040101', '16~18', '台北市大安區住安里四維路124巷1~30號'],
['3', '自行車竊盜', '1040104', '07~09', '台北市大安區敦煌里仁愛路四段266巷1~30號']
...
]
```

任務：自行車竊案統計

常見的文字資料檔格式

大家熟悉的試算表格式




```
[
['1', '自行車竊盜', '1020630', '13~15', '台北市松山區三民路151~180號'],
['2', '自行車竊盜', '1040101', '16~18', '台北市大安區住安里四維路124巷1~30號'],
['3', '自行車竊盜', '1040104', '07~09', '台北市大安區敦煌里仁愛路四段266巷1~30號']
...
]
```

接下來在程式中我們以 data[0][2] 取用第一筆竊案資料的第三個欄位 (發生日期)，
以 data[5][3] 取用第六筆竊案資料的第四個欄位 (發生時段)，應該還算能接受且符合直覺。

任務：自行車竊案統計

bike_crime1.py

```
def main():
    data = GetCrimeData('臺北市10401-10709自行車竊盜點位資訊.txt')

    print(type(data))
    print(data[:3])

    ... [略]
```

如果不了解 GetCrimeData() 傳出的結果是什麼，我們可以在這裡自行加幾行程式碼。

例如先加 print(type(data)) 得知 data 是個 list。再加 print(data[:3]) 印出前三筆。

```
<class 'list'>
[['1', '自行車竊盜', '1020630', '13~15', '台北市松山區三民路151', '~', '180號'], ['2', '自行車竊盜', '1040101', '16~18', '台北市大安區住安里四維路124巷1~30號'], ['3', '自行車竊盜', '1040104', '07~09', '台北市大安區敦煌里仁愛路四段266巷1~30號']]
102
```

從 print() 結果我們能精確理解 data 的內容與格式，以便撰寫程式處理。

臺北市10401-10709自行車竊盜點位資訊 - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

```
編號,案類 發生(現)日期 發生時段 發生(現)地點
1 自行車竊盜 1020630 13~15 台北市松山區三民路151 ~ 180號
2 自行車竊盜 1040101 16~18 台北市大安區住安里四維路124巷1~30號
3 自行車竊盜 1040104 07~09 台北市大安區敦煌里仁愛路四段266巷1~30號
4 自行車竊盜 1040106 13~15 台北市萬華區國興路1~30號
5 自行車竊盜 1040108 07~09 台北市北投區清江里三合街二段481~510號
```

任務：自行車竊案統計

bike_crime1.py

```
def GetYear(record):
    return int(record[2][:3])

def main():
    data = GetCrimeData('臺北市10401-10709自行車竊盜點位資訊.txt')

    print(GetYear(data[0]))

    ...[略]
```

data

```
[
['1', '自行車竊盜', '1020630', '13~15', '台北市松山區三民路151~180號'],
['2', '自行車竊盜', '1040101', '16~18', '台北市大安區住安里四維路124巷1~30號'],
['3', '自行車竊盜', '1040104', '07~09', '台北市大安區敦煌里仁愛路四段266巷1~30號']

...
]
```

data 是二維列表，包含所有竊案資料。data[0] 是一維列表，包含一筆竊案資料。

呼叫 GetYear(data[0]) 時，會令 record 指稱 data[0]。
因此，在 GetYear() 內寫 record[2] 就彷彿寫 data[0][2] 一樣的意思。

任務：自行車竊案統計

字串小補充

- 把字串想成字元的列表。

```
s = ''          #空字串
s = '1'         #單一字元
s = 'abc'       #多個字元

print(s[0])     #印出 a
print(s[1:])    #印出 bc
print(s[::-1])  #印出 cba

t = s[1:] + 'def'
print(t)        #印出 bcdef

#s[0] = '4'     #TypeError: 'str' object does not support item assignment
```

```
a
bc
cba
bcdef
```

任務：自行車竊案統計

請完成

- GetMonth()
  - 回傳一筆竊案的發生月份。
  - m = GetMonth(data[0])，m 等於整數 6。
- GetTime()
  - 回傳一筆竊案的開始時間與結束時間。
  - s, e = GetTime(data[0])，s 和 e 分別為整數 13 和 15。

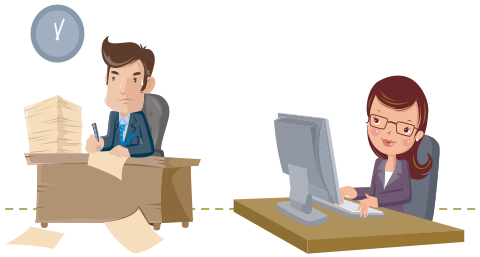
檔案中已完成 GetYear()，請參考之。

臺北市10401-10709自行車竊盜點位資訊 - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

編號,案類 發生(現)日期 發生時段 發生(現)地點
1 自行車竊盜 1020630 13~15 台北市松山區三民路151 ~ 180號
2 自行車竊盜 1040101 16~18 台北市大安區住安里四維路124巷1~30號
3 自行車竊盜 1040104 07~09 台北市大安區敦煌里仁愛路四段266巷1~30號
4 自行車竊盜 1040106 13~15 台北市萬華區國興路1~30號
5 自行車竊盜 1040108 07~09 台北市北投區清江里三合街二段481~510號

任務：自行車竊案統計



bike_crime1.py

```
def GetYear(record):
    return int(record[2][:3])
```

好啦！換你試試看！寫出 GetMonth() 與 GetTime()。

GetMonth() 回傳一筆竊案的發生月份。
m = GetMonth(data[0])，m 等於 6。
GetTime() 回傳一筆竊案的開始時間與結束時間。
s, e = GetTime(data[0])，s 和 e 分別為 13 和 15。

```
def main():
    data = GetCrimeData('臺北市10401-10709自行車竊盜點位資訊.txt')

    print(GetYear(data[0]))
    print(GetMonth(data[0]), GetTime(data[0]))

    ...[略]
```

data

```
[
['1', '自行車竊盜', '1020630', '13~15', '台北市松山區三民路151', '~', '180號'],
['2', '自行車竊盜', '1040101', '16~18', '台北市大安區住安里四維路124巷1~30號'],
['3', '自行車竊盜', '1040104', '07~09', '台北市大安區敦煌里仁愛路四段266巷1~30號']

...
]
```

任務：自行車竊案統計

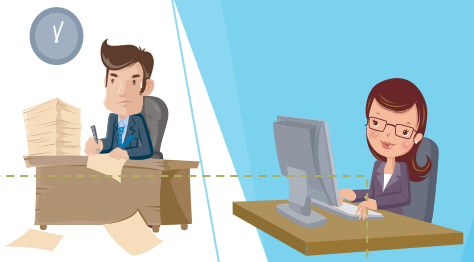
請完成

- GetYearCount()
  - 傳入所有竊案資料，回傳某年度的竊案數目。
  - c = GetYearCount(data, 107)，c 等於 183。

- 寫完的人可以寫其它的統計函式，例如
  - GetMonthCount()
  - GetTimeCount()
  - GetPlaceCount()

  - 等等。

任務：自行車竊案統計



```
... [略]

def GetYearCount(data, year):
    count = 0
    for record in data:
        if GetYear(record) == year:
            count += 1
    return count

def main():
    data = GetCrimeData('臺北市10401-10709自行車竊盜點位資訊.txt')

    print(GetYear(data[0]))
    print(GetMonth(data[0]), GetTime(data[0]))

    qyear = 107   #未來可以改成由使用者決定查詢年份
    print(qyear, '年總案件數 = ', GetYearCount(data, qyear))
```

對 data 中的每一筆竊案資料，運用剛剛寫好的 GetYear() 得到發生年份，如果等於查詢年份，案件數加 1。

data

```
[
['1', '自行車竊盜', '1020630', '13~15', '台北市松山區三民路151', '~', '180號'],
['2', '自行車竊盜', '1040101', '16~18', '台北市大安區住安里四維路124巷1~30號'],
['3', '自行車竊盜', '1040104', '07~09', '台北市大安區敦煌里仁愛路四段266巷1~30號']
...
]
```

任務：自行車竊案統計

```
def GetYearCount(data, year):
    count = 0
    for record in data:
        if GetYear(record) == year:
            count += 1
    return count

def GetMonthCount(data, month):
    count = 0
    for record in data:
        if GetMonth(record) == month:
            count += 1
    return count

def GetTimeCount(data, hour):
    count = 0
    for record in data:
        start, end = GetTime(record)
        if start<=hour<end:
            count += 1
    return count
```

```
def GetYear(record):
    return int(record[2][:3])

def GetMonth(record):
    return int(record[2][3:5])

def GetTime(record):
    return int(record[3][:2]), \
           int(record[3][-2:])
```

熟悉 list comprehension 的程式員可以寫成：

```
def GetYearCount(data, year):
    return len([e for e in data if GetYear(e) == year])
```

或者

```
def GetYearCount(data, year):
    return sum([1 for e in data if GetYear(e) == year])
```

撰寫函式查詢任一年份、月份和小時發生的總竊案數。

任務：自行車竊案統計

- 在這個任務裡，我們學會

「

對多欄位資料集以指定欄位的指定條件進行統計。

」

- 例如：

「

對台北市自行車竊案資料集計算 10 月份的竊案總數。

」

自行車竊案分析

函式 (II)

變數的可用點

基本型：看前後位置

```
b = 3
a = b
```

程式正常執行。後面看得到前面。

```
a = b
b = 3
```

錯誤。前面看不到後面。

NameError: name 'b' is not defined.

```
a = 3

if True:
    print(a)
```

```
3
```

程式正常執行。句子不分裡外。

```
if True:
    a = 3

print(a)
```

```
3
```

程式正常執行。句子不分裡外。

（這和 C/C++ 不同。）

變數的可用點

變化型：看執行流程

```
if True:
    a = 3
else:
    b = 4

print(a)
print(b)
```

錯誤。
NameError: name 'b' is not defined.

else: 那一段沒有執行到，b 未定義。

```
for i in range(2):
    if i==1:
        print(i, j)
    j = 9

print(i, j)
```

```
1 9
1 9
```

程式正常執行。

第一次進入迴圈時，i 為 0，此時不執行 print(i, j)。緊接著，j 被定義。

第二次進入迴圈時，i 為 1，此時執行 print(i, j)，j 已被定義，可正常列印。

變數的可用點

- 函式摻一咖：函式裡面可以使用外面的全域變數。

```
i = 9
def f():
    print(i)

f()
```

```
9
```

程式正常執行。列印出 9。

函式有分裡外，裡面看得到外面。

```
def f():
    print(i)

i = 9
f()
```

```
9
```

程式正常執行。列印出 9。

函式有分裡外，裡面看得到外面。
（呼叫 f() 前已經看過 i 了。）

```
i = 9
def f():
    print(i)

i = 8
f()
```

```
8
```

程式正常執行。列印出 8。

函式有分裡外，裡面看得到外面。
（呼叫 f() 前 i 被改成 8 了。）

變數的可用點

- 函式摻一咖：函式外面無法使用函式裡面的變數

```
def f():
    fi = 9

print(fi)
```

錯誤。
NameError: name 'fi' is not defined.

函式外面無法使用函式裡面的區域變數。
（這是一種保護措施，縮小變數的可用範圍，也縮小除錯時需檢查的程式碼範圍。）

```
def f():
    fi = 9

def g():
    print(fi)

g()
```

錯誤。
NameError: name 'fi' is not defined.

函式外面無法使用函式裡面的區域變數。

變數的可用點

函式摻一咖：函式裡的同名變數遮蔽現象

```
i = 9
def f():
    i = 10
    print(i)
    i = 11

f()
print(i)
```

```
10
9
```

程式正常執行。

函式裡面定義自己的區域變數 i，
和外面的全域變數 i 是不同的。

```
for i in range(2):
    print('a')

def f():
    for i in range(3):
        print('b')

f()
print(i)
```

```
a
a
b
b
b
1
```

程式正常執行。

函式裡面定義自己的區域變數 i，
和外面的全域變數 i 是不同的。

變數的可用點

函式摻一咖：費解的 UnboundLocalError

```
i = 9
def f():
    print(i)
    i = 10
    print(i)

f()
print(i)
```

錯誤。

```
UnboundLocalError: local variable
'i' referenced before assignment.
```

蛤？不是說函式裡面可以使用外面的全域變數嗎？
怎麼會有錯誤？

只要在函式中出現指派賦值（包含 a=, a+=, for a in …）的敘述，a 便被認定為一個區域變數。在函式中所有 a 都視為該區域變數 a。

本例中，因為 print(i) 出現在 i = 10 之前，所以視為未定義。

變數的可用點

函式摻一咖：費解的 UnboundLocalError

```
i = 9
def f():
    global i
    print(i)
    i = 10
    print(i)

f()
print(i)
```

```
9
10
10
```

在函式中使用 global 來定義變數 i，明確指出我們真的要使用全域變數 i。

程式碼中的 global 算是一個警示，提醒程式員這裡可能會修改到全域變數。

```
i = 9
def f(i):
    print(i)
    i = 10
    print(i)
    return i

i = f(i)
print(i)
```

```
9
10
10
```

較好的作法是透過傳遞引數和回傳值。

這讓程式員不必鑽進函式內部去除錯。

變數的可用點

函式摻一咖：費解的 UnboundLocalError

```
i = 9
def f():
    if False:
        i = 3

    if i==3:
        print()

f()
```

錯誤。

```
UnboundLocalError: local variable
'i' referenced before assignment.
```

不管該程式碼是否被執行，只要函式內出現賦值指派，該變數就被認為是區域變數。

引數與參數列

- 全域變數很好用，但也很危險。
  - 到處都可以用，但到處都可能改變它的內容。
- 區域變數很安全。
  - 只有該函式的程式片段可以使用/修改。
  - 要除錯時，只需專注在一小段程式碼。
- 別人看不到的時候怎麼辦？

  ⇒ 傳遞引數或回傳。

引數與參數列

傳遞引數給其它函式

```
def f():
    score = int(input('請輸入分數>'))
    check()

def check():
    if score>=60:
        print('及格')
    else:
        print('不及格')
f()
```

錯誤。
NameError: name 'score' is not defined.

```
def f():
    score = int(input('請輸入分數>'))
    check(score)

def check(sc):
    if sc>=60:
        print('及格')
    else:
        print('不及格')
f()
```

請輸入分數>99
及格

請輸入分數>59
不及格

引數與參數列

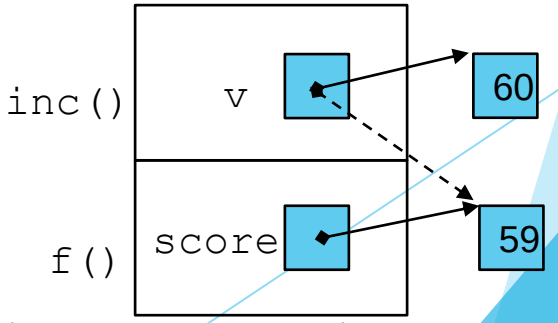
傳遞引數給其它函式：「指派」的意義

```
def f():
    score = int(input('請輸入分數>'))
    inc(score)
    print('In f():', score)

def inc(v):
    print('In inc():', v)
    v += 1                    # v = v+1，令 v 指稱另一個值
    print('In inc():', v)

f()
```

```
請輸入分數>59
In inc(): 59
In inc(): 60
In f(): 59
```

簡單的解決方案：利用回傳值（return v）。

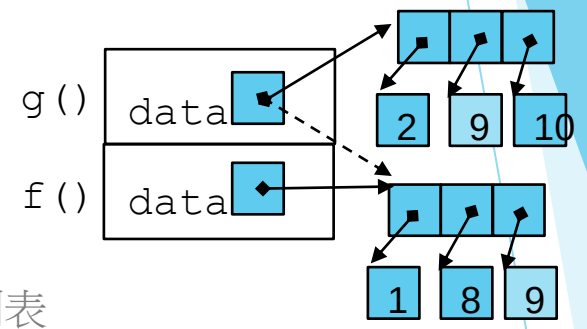
```
def inc(v):
    v += 1
    return v
```



引數與參數列

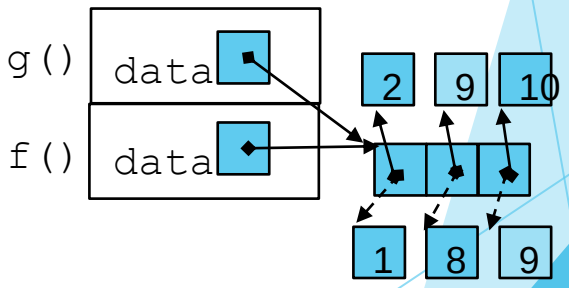
傳遞引數給其它函式：「指派」的意義

```
def f():
    data = [1, 8, 9]
    g(data)
    print('In f():', data)

def g(data):
    print('In g():', data)
    data = [e+1 for e in data]  # 令 data 指稱另一個列表
    print('In g():', data)

f()
```




```
In g(): [1, 8, 9]
In g(): [2, 9, 10]
In f(): [1, 8, 9]
```

```
def f():
    data = [1, 8, 9]
    g(data)
    print('In f():', data)

def g(data):
    print('In g():', data)
    for i in range(len(data)):
        data[i] += 1            # 令 data[i] 指稱另一個值
    print('In g():', data)

f()
```




```
In g(): [1, 8, 9]
In g(): [2, 9, 10]
In f(): [2, 9, 10]
```

任務：自行車竊案統計 (II)

- 自教學平台取得檔案
  - 臺北市 10401-10709 自行車竊盜點位資訊.txt
    - 來源：https://data.taipei/dataset/detail/metadata?id=5c5e9e13-9803-47c0-bbd2-1a4b3c11c49b
- 在前面的任務中，我們學會撰寫程式查詢特定年份、月份、時刻的竊案總數。
- 我們可以如法刨製查詢特定區域（如文山區）的竊案總數，但如果我想知道所有區域的竊案數呢？
  - 如果使用查詢特定區域的作法，我需要事先知道所有區的名稱，並且需要多次查詢才能得到所有區域的資料。
  - 該怎麼作才能看過資料一次就統計出所有數據呢？

任務：自行車竊案統計 (II)

- 萬法不離其宗 - 我們還是一筆一筆看過資料。

data

```
[
['1', '自行車竊盜', '1020630', '13~15', '台北市松山區三民路151', '~', '180號'],
['2', '自行車竊盜', '1040101', '16~18', '台北市大安區住安里四維路124巷1~30號'],
['3', '自行車竊盜', '1040104', '07~09', '台北市大安區敦煌里仁愛路四段266巷1~30號'],
['4', '自行車竊盜', '1040106', '13~15', '台北市萬華區國興路1~30號'],

...
]
```

data[0][4][3:6]

區域	竊案數
松山區	1

看過 data[0]

區域	竊案數
松山區	1
大安區	1

看過 data[0] data[1]

區域	竊案數
松山區	1
大安區	2

看過 data[0] data[1] data[2]

區域	竊案數
松山區	1
大安區	2
萬華區	1

任務：自行車竊案統計 (II)

思路

def main():

```
results = [] #存放統計結果的列表
for record in data:
    place = GetPlace(record)

    #檢查 results 裡面有沒有出現過 place
    #如果有，將竊案數加 1
    #如果沒有，新增 place，竊案數設為 1
```

```
def GetPlace(record):
    return record[4][3:6]
```

區域	竊案數

區域	竊案數
松山區	1

區域	竊案數
松山區	1
大安區	1

區域	竊案數
松山區	1
大安區	2

```
results  []
```

```
[
['松山區', 1]
]
```

```
[
['松山區', 1],
['大安區', 1]
]
```

```
[
['松山區', 1],
['大安區', 2]
]
```

任務：自行車竊案統計 (II)

思路

def main():

```
results = [] #存放統計結果的列表
for record in data:
    place = GetPlace(record)

    #檢查 results 裡面有沒有出現過 place
    #如果有，將竊案數加 1
    for e in results:
        if e[0]==place:
            e[1] += 1
            break
    #如果沒有，新增 place，竊案數設為 1
```

results

```
[]
```

→

```
[
['松山區', 1]
]
```

→

```
[
['松山區', 1],
['大安區', 1]
]
```

→

```
[
['松山區', 1],
['大安區', 2]
]
```

任務：自行車竊案統計 (II)

def main():

思路

```
results = [] #存放統計結果的列表
for record in data:
    place = GetPlace(record)

    #檢查 results 裡面有沒有出現過 place
    #如果有，將竊案數加 1
    newfound = True
    for e in results:
        if e[0]==place:
            e[1] += 1
            newfound = False
            break
    #如果沒有，新增 place，竊案數設為 1
    if newfound:
        results.append([place, 1])

print(results)
```

現在你知道近三年哪一區的竊案總數最多了。
這並不表示該區最容易被偷車喔！（各區的自行車數可能不同）

任務：自行車竊案統計 (II)

- 如果想知道每個地區每一時刻的竊案總數，該怎麼修改程式呢？

def main():

```
results = [] #存放統計結果的列表
for record in data:
    place = GetPlace(record)
    s, e = GetTime(record)

    #檢查 results 裡面有沒有出現過 place
    #如果有，將「時刻 s 到時刻 e」的竊案數各加 1
    newfound = True
    for e in results:
        if e[0]==place:
            將「時刻 s 到時刻 e」的竊案數各加 1
            newfound = False
            break
    #如果沒有，新增 place，將「時刻 s 到時刻 e」的竊案數設為 1
    if newfound:
        新增 place，將各時刻（24個小時）的竊案數設為 0
        將「時刻 s 到時刻 e」的竊案數各加 1

print(results)
```

任務：自行車竊案統計 (II)

- 統計每個地區每一時刻的竊案總數

```
def main():
    data = GetCrimeData('臺北市10401-10709自行車竊盜點位資訊.txt')

    counts = []
    for record in data:
        place = GetPlace(record)
        start, end = GetTime(record)

        newfound = True
        for e in counts:
            if e[0] == place:
                for t in range(start, end):
                    e[1][t] += 1
                newfound = False
                break
        if newfound:
            counts.append([place, [0]*24])
            for t in range(start, end):
                    counts[-1][1][t] += 1

    for e in counts:
        print(e)
```

counts 列表長這樣：

```
[
['松山區', [0, 5, 6, ..., 0, 6, 6]],
['大安區', [4, 26, 25, ..., 1, 21, 21]],
['萬華區', [2, 10, 8, ..., 1, 8, 8]],
['北投區', [0, 6, 6, ..., 1, 7, 7]],
...
]
```

任務：自行車竊案統計 (II)

- 統計每個地區每一時刻的竊案總數
  - 以函式包裝起來

```
def AddCount(slots, record):
    start, end = GetTime(record)
    for t in range(start, end):
        slots[t] += 1

def GetPlaceTimeCounts(data):
    counts = []
    for record in data:
        place = GetPlace(record)

        newfound = True
        for e in counts:
            if e[0] == place:
                AddCount(e[1], record)
                newfound = False
                break
        if newfound:
            counts.append([place, [0]*24])
            AddCount(counts[-1][1], record)

    return counts
```

任務：自行車竊案統計 (II)

- 在這個任務裡，我們學會

「

對多欄位資料集以指定欄位分群進行統計。

」

- 例如：

「

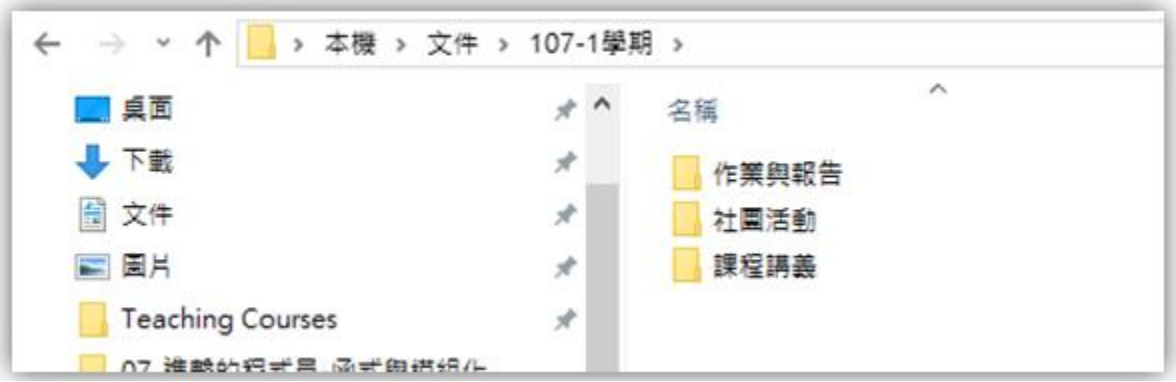
對台北市自行車竊案資料集計算各地區的竊案總數。

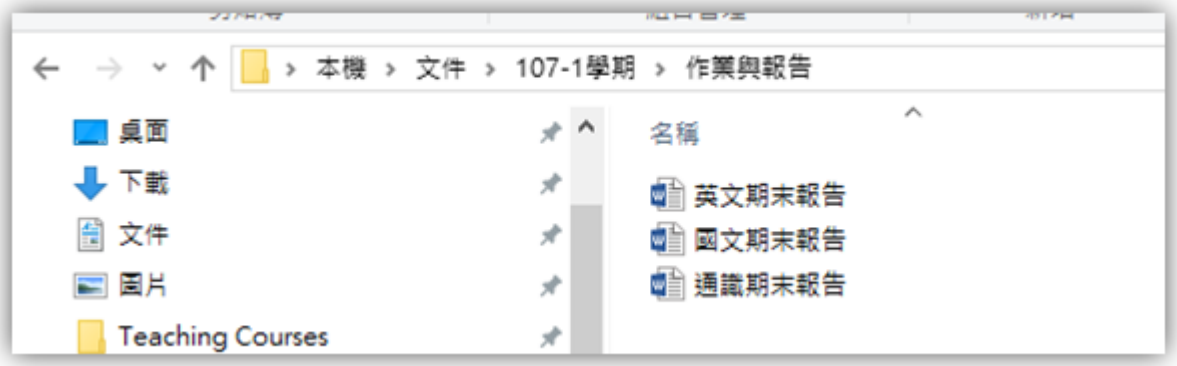
」

程式模組

模組

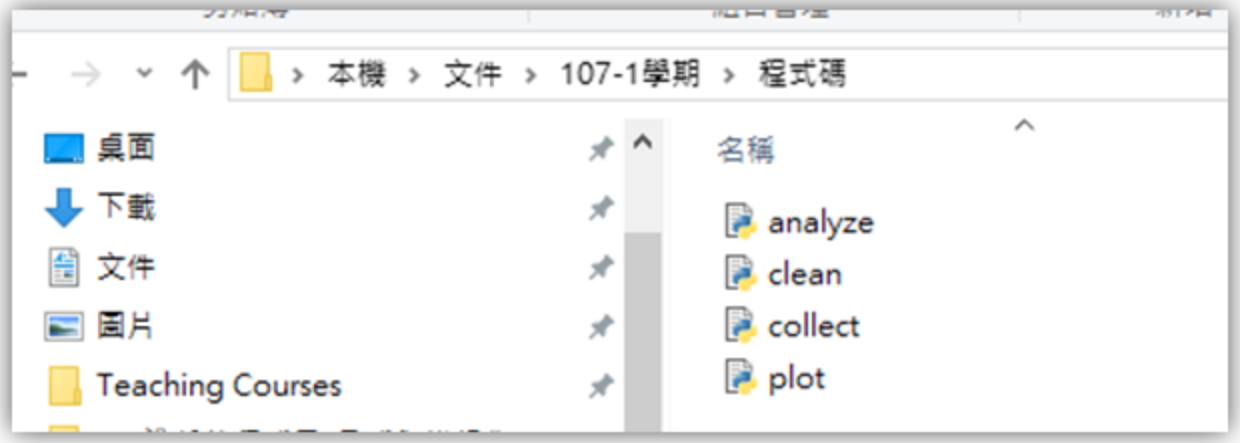
- 你都怎麼組織管理你的資料呢？
  - 大家都很熟悉將資料分門別類放在不同檔案甚或資料夾（目錄）中吧!





模組

- 當你撰寫的程式碼越來越多，把它們分門別類放在不同檔案中，通常也有助於管理。
  - 將性質相近的函式放在同個檔案，方便找尋和分享。
  - 將程式碼分散在多個檔案，方便修改、測試與抽換。
  - 將程式碼分散在多個檔案，方便團隊工作。



程式碼的進化

程式碼

當你發現有些程式碼經常被使用或是想要隱藏實作細節，便可將完成一特定功能的程式碼片段包裝成一個函式。

函式

該函式的參數代表使用者能控制此函式的彈性與權力。

模組

當你發現多個函式具有相似性、相關性或相互輔助性時，便可將這些函式歸納在一個模組（.py 檔案）。

應用程式

將你手邊有許多好用的模組，你就可以輕鬆地取用其中需要的函式，組成出你想要的應用程式。

多個模組還可以組成套件 (package)，這個就留待有緣者繼續探索吧！

模組

簡單範例

- 手邊有兩個程式碼檔 *main.py* 和 *lib.py*。
- 在 *main.py* 呼叫 *lib.py* 的函式 `mean()`。

main.py - I

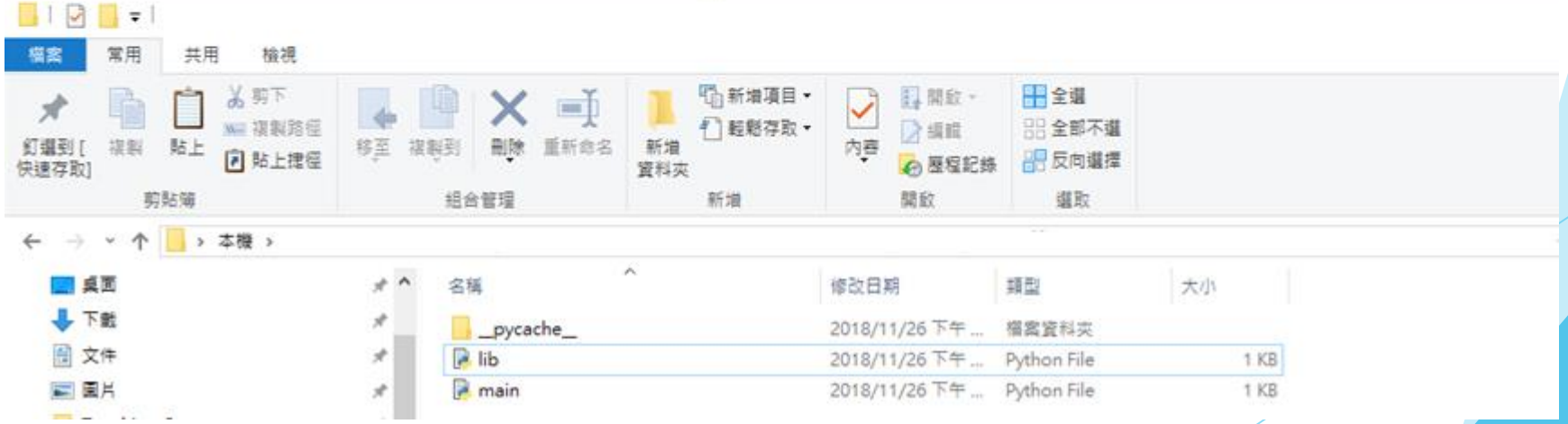
File Edit Format Run Options Window Help

```python
import lib

print(lib.mean([3, 5, 8]))
```

lib.py -

File Edit Format Run Options Window Help

```python
def mean(data, prec=2):
    if data:
        return round(sum(data)/len(data), prec)
    else:
        return 0
```



模組

import 模組

- 帶入另一個程式碼檔案（模組）
  - 在前頁的檔案狀態下，*main.py* 中 import lib 可使得 *lib.py* 中的函式變得可用。

main.py

```
import lib
print(lib.mean([3, 5, 8]))
```

  - 以下 *main.py* 都無法成功呼叫 *lib* 中的函式。

```
print(lib.mean([3, 5, 8]))
```

```
NameError: name 'lib' is not defined.
```

```
print(mean([3, 5, 8]))
```

```
NameError: name 'mean' is not defined.
```

```
import libb
print(mean([3, 5, 8]))
```

```
ModuleNotFoundError: No module named 'libb'
```

模組

from 模組 import 函式

main1.py

```
import lib
from lib import f, g

f()
g()
lib.h()
#h() #error
```

```
f()
g()
h()
```

main2.py

```
from lib import *

f()
g()
```

```
f()
g()
```

lib.py

```
def f():
    print('f()')
def g():
    print('g()')
def h():
    print('h()')
```

注意：若有同名函式，會依程式流程先後順序，將該函式定義為最後一個版本。

模組

- [from …] import … as
  - 名稱很長的時候，可以利用 as 縮為簡寫。
  - 為了避免名稱衝突時，可以利用 as 改名稱。

```
import this_is_a_lib_with_very_long_name as lib

lib.f()
lib.g()
```

```
from lib import this_is_a_function_with_a_very_long_name as f

f()
```

```
from lib import sort as mysort, filter as myfilter

mysort()
```

模組

- 檔案放在資料夾中
  - 假設 *lib.py* 放在 *folder* 資料夾中。

```
import folder.lib
from folder.lib import g
import folder.plot as plt

library.lib.f()
g()
plt.show() #假設 plot.py 裡面有函式 show()
```