PythAPCS123

Python 程式設計從 APCS 實作 1 級到 3 級

課程內容：

這個課程是為了沒有程式基礎的人學習基礎的 Python 程式設計而設計的，學習者只需要具有簡單的英文與數學基礎(大約國中程度)就可以。課程內容與學習的短期目標設定為可以在 APCS 實作題考試中達到三級分的程度，所以課程的名稱中 123 是 1 級到(to)3 級之意，APCS 一級分是完全不會寫程式的程度。

在此具體目標下，課程中不會對 Python 的指令與函數做鉅細靡遺的說明，而是挑選重要與必須的部分作學習指引與練習。同時，針對考試的需求，課程中使用與考場相同的程式撰寫環境 Python IDLE 3，以及以 ZeroJudge 為主要練習的環境，以便學習者熟悉 Auto Judge 的解題環境，並且可以在課後做隨時的練習。

這份講義是上課的摘要，並非完整的教科書，很多說明是在教學影片中。

教材訊息發布於臉書社團「APCS 實作題檢測」。

作者：吳邦一，2021 年 12 月

目錄

0. 基本知識

什麼是程式設計

電腦的基本組成：輸入、輸出、運算與資料儲存

程式是控制電腦執行工作的腳本，程式由指令組成，一個指令相當一句話，一支程式則是一篇文章。

一道一道的命令不外乎：對某資料做某運算，或者這道指令做完後做哪一道指令(控制流程)。

程式的功能各異，但每個程式中的指令不外乎是做運算，或是控制流程。

如何設計程式？

透過某一種程式語言。電腦硬體能夠解讀的程式是機器碼，人不容易直接寫，因此發展出程式語言，程式設計師以程式語言撰寫程式再轉換成機器碼，這個翻譯者稱為編譯器(compiler)或直譯器(Interpreter)，Python 是直譯式語言，其他編譯的語言比較多。

程式語法：

程式語言的語法是很嚴謹的，不像中英文，程式必須完全依照語法來撰寫，否則無法翻譯，稱為語法錯誤(CE, compilation error)。每個程式語言有自己的語法甚至格式的要求，差一個字、差一個標點符號、甚至差一個空白都可能是錯誤，所以學習程式設計第一要務是嚴格遵守紀律，不可以自己發明語法。

程式是要完成某個特定的工作，正確語法不一定做出正確的事情，應該把兩數相加而做成相減，語法雖然正確，但語意錯誤，在解題時稱為 WA(wrong Answer)。

一個可以跑出正確答案的程式，但別人的程式跑一秒而這支程式要跑一萬年才能算出答案，這樣的程式效率太差，解題程式都有執行時間的限制，通常在數秒，超過執行時限稱為 time limit exceed(TLE)錯誤。

如何驗證程式的正確：

程式的測試是一個非常複雜的事情，在解題程式(競技程式)的世界裡，每一個題目都有設計好的測試資料，選手的程式繳交後，將測試資料餵給繳交的程式，將輸出結果與設定好的正確輸出做比對，如果通過就算正確，稱為 AC (accept)。通常為了測試程式

是否正確，每個題目都會有多筆測資，競賽時通常要通過全部的測資才能得到分數，APCS 則是根據通過的測資筆數給分。

Auto Judge：一個比賽與考試的系統，選手在自己的電腦上撰寫程式，繳交後傳送到裁判的電腦，進行編譯、測試與比對答案，根據結果通知選手該程式是 CE, WA, TLE 或者 AC。

- 目前大部分的比賽採用即時的裁判系統(online judge, OJ, 戲稱 orange juice)，選手繳交後回立刻回覆結果，如果有錯誤，選手可以在更正後繼續繳交。
- APCS 屬於賽後裁判(後測)，考試時只會幫你測試範例但不做完整的測試評分，考試結束後才會使用 auto judge 做評分。

Python IDLE 介紹

IDE (整合發展環境)是一個軟體，方便設計者撰寫與執行程式，並有一些工具可以方便進行測試與除錯。有非常多的 IDE，有些具有複雜的功能，初學者的程式通常很短，一開始不一定要先學 IDE 的複雜功能，只要可以編輯與執行就可以用了。

Python 的 IDLE 是一個極輕量的 IDE，也是 APCS 考試環境提供的 Python IDE。我們一開始只需要編輯存檔與執行等功能就好了。

(再來看一下實際的操作，打開 Python IDLE 試用一下)python 是直譯式語言，所以它有一個交談介面(shell)。Python 對於初學者來說是個友善的程式語言，其中一個原因是他是直譯式，可以透過交談介面來嘗試指令的作用。C 與 Jave 這一類編譯的語言，必須寫下一個完整的程式才能執行，這對初學者來說是比較困難的。在本課程中，我們一開始只在 shell 上面操作個別指令，然後才會引導寫完整的程式。

(介紹在 shell 簡單的操作)

```
print('Hello')
a = 5
a
a+3
b
```

(安裝是很簡單的事情，找到 python 官網下載 IDLE 會同時安裝 python，注意要安裝 python3 而非 python2)

ZeroJudge 介紹

ZJ 是一個提供高中生練習的解題網站，上面有很多題目。在簡單的註冊之後，我們可以自由地針對某個題目在自己的電腦上撰寫程式後繳交，它有自動裁判會我們判定答案，有了解題程式，可以一天 24 小時全年無休的練習程式。

國內外還有很多解題網站，很多高中的社團也建置了解題網站，如建中、台中一中、台中女中以及南一中，ZJ 適合初學者與高中生。

特別留意基礎題庫與 APCS 題庫。

(瀏覽一下 ZJ，示範一下繳交程式。)

1. 輸出與資料存取

輸出函數 print()

寫程式不是所有的事情都由程式設計者完成，就像蓋房子需要工具，Python 提供了很多工具來幫助程式設計者完成想要的功能，這些工具就是 Python 內建的函數。每個函數有一個名字，後面會跟著一對小括號，例如這裡要介紹的 print()。程式中透過呼叫函數來執行特定的動作，所謂呼叫，就是直接寫上函數的名字。我們慢慢會接觸很多 python 的內建函數。

print()函數顧名思義就是將一些東西印出到輸出裝置(通常是螢幕)，呼叫 print()就會將()內的東西印到螢幕上。

(在 shell 練習 print()指令)(#後面是說明，不需要輸入)(python 中#後面的是註解，程式會忽略#到行末這些東西)

```
print("Hello") # 印出一段文字
print(5, 4, 6.12, 8) # 印出多個數字，項目之間預設以一個空白間隔
print('Hello', "Python", 12345) # 文字可以單引號或雙引號
print('Hello', "Python", 12345, sep=',') # 輸出時項目之間的分隔符號可以改變
```

(自己練習一下簡單的輸出，並熟悉 python 的 shell)

變數與資料存取

變數是什麼：程式要做運算，運算的對象是資料，所以資料存取是程式中的重要課題。資料在電腦中儲存在記憶體，記憶體的存取是根據他們的地址，在高階語言(如 python)不直接使用地址，而是透過名字，也就是變數。變數是個名字，對應到記憶體的某個位置，這個對應我們不用管。

變數命名：每個程式語言都有變數命名的規則，簡單來說，以下列方式不會錯：名字只使用英文字母數字與底線組成，非數字開頭，不要太長、不能撞關鍵字(所以最好不要取完整的英文字)，例如 test1, arr2, foo, bar。不要用中文不要用空白。

等號的意義是賦值(assignment)，把等號右邊的值給予等號左邊的變數，運作的順序是先計算等號右邊的值然後將值存入等號左邊的變數中。

```
y = 3
x = 5 + 3*6
```

指令的中間與後方可以加空白，但指令的前方不可以任意加空白，縮排有特別的意義。

將等號右邊的值算出來之後放入等號左邊的記憶體位置，左邊一定是個變數(位置)

```
y+3 = 6 #(這是錯的)
```

在 shell 中隨時可以打變數的名字，會告訴你目前的值

```
a=6
b=5
a [enter]
a, b [enter]
```

等號左右可以有逗號分開的若干個，但個數必須一樣

```
a, b = b, a
```

會將兩個數交換，為何？記得等號的運作順序：先計算右邊的值再放入左邊的記憶體。

```
a,b,c = a+b, b-a, c+6 按下輸入會如何？
```

(自己試試看使用變數、等號(賦值))

(練習題)

```
a = 5; b = 3 # 多行指令可以分號隔開放在同一行
a = b; b = a # 這會將 a 與 b 的內容交換嗎？為何？
a = 5; b = 3
t = a; a = b; b = t # 這三個指令的效果是什麼
c = 7
```

如果要將 a,b,c 三個變數循環交換(a 變成 b, b 變成 c, c 變成 a)，應該如何下指令

(結合 print 來使用)print 中可以印出變數的值

```
a,b,c = 5, 3, 6
print(a,b,c)
a = 8
print('a=', a)
print(a+3, b-2, a+b, sep=',')
```

2. 整數的算術運算

Python 的資料有分成整數、浮點數以及文字(字元字串)，數字(整數與浮點數)可以做算術運算，文字不行，所以程式中必須區分資料型態。我們上一章與這一章看到的都是整數。

整數的算術運算有六種： a+b, a - b, a*b, a//b, a%b, a**b

注意整數除法是兩個//，除法的除數不可為 0，a**b 是 a 的 b 次方。(根號可以用 a**0.5 但運算結果不是整數而是浮點數)

先乘除後加減，可以用括號，不確定時最好用括號，多重括號時都是小括號，其他的括號都有不同意義。

(自己練習一下寫運算式)，運算式中可以有整數與變數

```
x = 5+6*7
y = 7*x - 6*3 # x 會改變嗎? y 會改變嗎?
y = (7+x)*2 - 3
x = x+1 # 可以嗎？想想 = 的意義
x + y =5*6 # 可以嗎？想想 = 的意義
x = r*7 - 3 # 會如何
```

平方怎麼打呢?

```
r =4
area = r*r
```

次方也可以用**

```
w = r**2
x = r**3
r = 4
y = r**0.5 # 敲 y 看看他的值，注意 y 不再是整數
```

運算後賦值

類似 x = x+10 這樣的指令，可以把它解釋成，將 10 加進去 x 中，因為常會用到這樣的指令，python 中給予簡潔的寫法。別忘了等號的運算順序，先算出等號右邊的，再加(或減乘除..)入等號左邊的變數中。

```
x=4; y=5
x += 10
x -= 3 + y
x *= 2*(3+10)
x //= 4
x %= 3
```

```
x **= y+1
```

使用內建函數 max()與 min()

取出若干個數字的最大值/最小值。函數中可以有函數，也就是合成函數，呼叫時候從最內層算出來才算外層的函數。

```
x = max(1, 5, 2, -14, 2)
y = min( 7, -3, 2, 0, 1, 7)
x = max(7, 3*5, 5+x*2, y*2)
print( max(1,3, max(4*y-20, x-3), min(x-8, y-7)) )
```

建議算式不要寫得太複雜，可以拆開寫。

(練習變數與算術運算與 max, min, print)

l,w,h 為長方體的長寬高，輸出長方體表面積、體積、以及最大(最小)一面的面積。

3. 浮點數與字元字串

浮點數

Python 的數字資料分成整數與浮點數，有小數點的數字稱為浮點數(floating point)。加減乘都與整數相同，但除法是

```
x = 6.2 / 7.0
```

注意，在電腦中浮點數無法精確表達，試試看輸入

```
print( 0.1 + 0.2 )
```

因為 0.3 在二進位中是無限循環小數，就像在十進制下 1/7 = 0.142857142857…是無窮小數，0.3 在二進制是無窮小數。

浮點數與整數運算後會變成浮點數，整數做/運算後也會變成浮點數。

```
3.2 + 4
4*1.0
x = 4
x**0.5
x/1
```

在自動裁判的考試與比賽中，答案是 4.0 與 4 會是不一樣的，如果答案的格式要求整數，那麼輸出 4.0 很可能就是錯的，所以必須分清楚整數或是浮點數。

(浮點數的練習)

將 r 設成某數，用 print()輸出半徑為 r 的圓面積，球體積。

若 a,b,c 是一元二次方程式的三個係數且有兩實根，先大後小輸出兩實根。

承上題，若 a,b,c 均為整數，已知方程式為有兩相異整數根，請以整數形式先大後小輸出兩根。

本金 m，利率 r，期數 n，輸出以單利與複利計算的本利和。

字元字串

Pyhton 中的文字以字串來處理，單一個字母就稱為字元，多個字元就是字串。這裡的字元字串包含 ASCII 內的字元，大小寫英文字母'A'~'Z', 'a'~'z', '0'~'9'以

及一些符號。字元與字串以單撇或雙撇來表示，但左右要一致。字串不能做算術運算，相加+表示字串的連接

```
a = 'x'  # 字串也可以放進變數
b = "hello"
c = a+b  # 是什麼？
d = b+a # a+b differs from b+a
```

試試（跳脫字元）

```
a = 'test \n a new line?'
```

要打出\ 如何打

```
print('abc\\ndef')
```

試試如何印出

```
It's a book
His name is "John"
```

轉換函數

轉換函數 int(), str(), float(), ord(), chr()

- int(): 字串或浮點數轉整數
- str(): 數字轉字串
- float: 字串轉浮點數
- ord(): 字元的 ASCII code
- chr(): ASCII 轉字元

```
a = '123'
b = a+2
```

字串不能加整數

```
b = int(a) + 2
a, b [enter]
c = str(b+12345678)
d = int('3.1415')
pi = float('3.1415926')
```

字元與 ASCII 轉換

```
w1 = 'R' ; w2 = 'r'; print(w1, ord(w1), w2, ord(w2))
```

多行指令可以放在同一行，用分號隔開，執行順序是由左到右一一執行。但不要把指令寫得太長，萬一有錯搞不清楚錯在哪裡，而且可讀性差，通常我們只把"一組"指令放在同一行

ASCII 數字轉字元

```
print(chr(65),chr(66),chr(67),chr(65+32),chr(66+32))
```

16 進位的表示法 0x41

```
x = 0x41
x [enter]
```

ASCII 大寫 A~Z 是二進位 0x41~ 小寫與大寫差 0x20

```
print(chr(0x41), chr(0x42), chr(0x41+0x20), chr(0x42+0x20))
```

(練習)

(練習轉換函數以及字串的相加)

(故意打錯函數名稱看看會發生何事)

(對不能轉換的進行轉換函數會發生何事，例如 int('123.4'), int('123s'))

4. 常用的輸入方式

input 函數與輸入整數

執行 input()後，會等待使用者在鍵盤上輸入，直到按下[enter]後，系統將一行以字串形式輸入。注意字串是不能做算術運算的。

```
s = input() # 會輸入一行，讀進來 a 是一個字串。
a =int(s) #s 轉換為整數
a = int(input()) # 將輸入的字串轉換為整數，整數才能加減乘除
n = int(input('please input n:'))  # 可以輸出一段訊息再讀入
```

寫軟體的時候常用，但解題程式用不到而且千萬不能用(OJ 不可輸出任何規定以外的東西)。

(練習題) 連續輸入兩行，每行一個整數，將兩數相加後輸出結果。

如何在同一行輸入兩個整數(空白間格)：

```
a,b = map(int, input().split())
```

input().split() 的作用：將 input()字串以空白間格切斷成多個字串。試試

```
a,b = "foo bar".split()
```

map() 的作用：map(某函數名, 一些東西) 把這些東西一一丟進該函數運作。

在一行輸入三個整數的方法：

```
a,b,c = map(int, input().split())
```

在一行輸入不定個數的整數：

```
a = [int(x) for x in input().split() ]
```

這是 list comprehension 的方式，先把它的用法記一下，在後面單元會再說明，讀進來的東西都轉整數放在一個 list 中

如果是"輸入一個 n，以下有 n 行，每行一個整數"，怎麼讀呢？

```
a=[int(input()) for i in range(3)]
```

這也是 list comprehension 的方式，先把它的用法記一下，在後面單元會再說明，讀進來的東西都轉整數放在一個 list 中

簡單介紹 list

前面所提到的變數，每個變數代表一個記憶體位置，如果需要很多變數怎麼辦呢？

在 python 中有可以一次定義一大群變數的方法，就是用一個名字代表一群變數，搭配編號來區分每一個變數，這就是 list。list 是一個列表(串列)，很多元素串在一起，可以想成一群變數排成一列，有一個共同名稱，依照排列的順序給予編號 0, 1, 2,…。

好比說在一個班級裡有很多學生，但我們可以(班級+座號)的方式來稱呼每一位學生，例如一年三班 14 號、或二年甲班 22 號。也可以想成是整條馬路上很多房間排成一列，每個元素是個房間。中山北路是一個 list，很多房間是 0, 1, 2, … 號。(從 0 開始)

要稱呼一個房間必須寫出路名與號碼：中山北路 1 號。

List 用一對方括號括起來，中間以逗號分格

```
a = [1, 5, 4, 3, 7, 2, 4, 1]
```

印出 list 內容：

```
print(a)  # python 井字號之後該行是註解，註解是給人看的，編譯器不看
print(*a)  # 這個先記一下，相當是 C 語言的指標
print(*a, sep=',') # 分隔符號可以更改
```

List 元素存取用法是 list_name[index]，如 a[0]，a 是路名(中山北路)，0 是元素編號，a[0]就是中山北路 0 號。個別成員可以單獨被改變，也可以單獨被取用，index 也可以是變數，那時存取的元素就看當時 index 變數的值。

```
a = [0,1,2,3,4]
print(a[0], a[3], a[1]) #0 3 1
a[5] = 100
a[4] = a[2]+a[1]*4
a   按[enter]
i = 2
a[i] = -2
a[i] += 1
a[i] += a[i-1]
```

下面的指令會發生何事？

```
a = [ 1,4,0,2]
b = a
b[2] = 3
print(a,b)
```

注意：a 是單純資料(如整數浮點數字串)時， b = a 會將 a 的內容拷貝到 b，兩者是各有各的記憶體位置。但若 a 是 list 的時候， b = a 並非將 a 複製一份到 b，兩者會是相同的記憶體位置，也就是同一個 list 有兩個名字。上例中，當 b[2]被改的時候，印出來的 a 與 b 是一樣的。要將 a 複製一份，可以用：

```
c = a.copy()
```

這才是將 a 複製一份給 c，試試看改變 a 與 c 的內容，看兩者已經脫離關係了，後面也會教其他的方式。

關於句點：python 中若 a 是物件，a.xxx()是指對於 a 這個物件執行 xxx 這個函數(方法)，此處 a.copy()是將 a 這個 list 複製一份，後面會再看到類似的東西。

List 常用的計算函數

前面有介紹過的 max()與 min()都可以作用在 list 上，不管 list 有多長，都可以算出整個 list 的最大值與最小值。此外，sum()這個函數也可以作用在 list 上，功能是取得總和。

```
a = [5, 3, 4]
large = max(a)
small = min(a)
middle = sum(a) - large - small
```

list 還有一個非常重要而常用的函數就是排序，排序的意思是將 list 的元素依照某種順序重新安排位置，元素不會增加或減少，只是會重新安排位置。我們先看一下最常見的排序，數字由小排到大。

假設 a 是一個 list 且元素都是數字，a.sort()就會將 a 進行由小到大的排序

```
a = [3, 1, 5, 7, 3, 4, 2, 5]
a.sort()
a # [1,2,3,3,4,5,5,7]
```

如果要由大而小排序呢？可以這樣下指令

```
a.sort(reverse=True)
```

sort 的用處很大，還有一些其他的用法，我們在後面的章節會再來說明。

5. 將指令組合成程式 --- 循序式指令結構

在本機寫程式

一行一行的指令放在一個檔案就組成一支程式，指令的流程預設是由上到下，一行一行執行，前一行做完才會做下一行，可看成分解動作。我們現在學了輸入輸出與基本的計算，已經可以開始做簡單的題目了，讓我們來試試。

(練習題 5.1)輸入有兩行，每行一個整數，輸出的第一行是兩數的和，第二行是兩數的差。

(示範)

(練習題 5.2)輸入有兩行，每行三個整數(空白間隔)，找出每行數字的最大值 x1 與 x2，輸出一行包含三個數字，依序是 x1, x2, x1+x2，兩數間以一個空白間隔。

(示範)

在 ZJ 練習程式

讓我們完成第一支 AC 程式：

ZJ a001

- 上網看題目並登入
- 在 IDLE 開一個檔案

```
who = input()  # 輸入一個名字 Pika
print('hello, ', who)  # 輸出'hello, Pika'  注意逗號與空格
```

- 存檔與執行
- 測試
- 上傳

再練習一題

a002 簡易加法

(ZJ 基礎題庫有很多可以練習的題目)

6. 碰到錯誤怎麼辦 debug

Python 程式執行時，如果碰到語法錯誤或執行的錯誤，程式會停止執行，並且輸出一段錯誤訊息，這個訊息會指出錯誤所在的位置以及原因，我們可以根據錯誤訊息來找問題並修正，然後再執行看看，一個程式可能有多個錯誤，所以除錯的過程可能會持續多次。下面做個簡單的示範

```
y = 3
v1 = true
x = inpuut()
a[1] = x
print(a)
```

在 OJ 與 APCS 考試的時候，你看不到執行的結果，所以要在本機自己做一些簡單的測試(例如題目中的範例)。

語法錯誤好抓，語意錯誤比較難找。

除錯不變心法：錯誤重現，故障隔離。

不是每次都會發生的錯誤比較難找，運用相同輸入，可使錯誤重現。

故障隔離：不是所有的錯誤都會指出錯誤發生的指令位置，而且發生錯誤的地方可能是因為先在別處發生了錯誤所導致的。

還有一種常碰到的情形，程式跑下去後，沒有反應。這種情形有可能是因為程式需要執行很久，或者程式因為錯誤進入了無窮迴圈。(迴圈後面才會教)

另外一種錯誤是：程式可以順利執行但計算結果與預期不同。

除錯有一些工具，將來程式寫得比較大時可以再學學。除錯最簡單最基本的方法就是利用 print 印出一些訊息來找到問題發生位置

a. 確定輸入讀取正確
b. 印出重要參數

展示簡單的例子

```
x = input()
y = x+5
a = int(input())
s = (x+y)//(a-5)
print(s)
```

7. 邏輯運算

前面都是一行一行循序式的指令，我們準備介紹控制流程的指令，要先講邏輯運算。一個邏輯表示式通常在比較兩個變數或常數的值，包含相等、不相等、大於、大於等於、小於，以及小於等於，例如在 shell 輸入以下的比較式：

```
5 > 3
x = 7 # 別忘了一個等號是賦值
x <=7
7.1 > x
x == 0 #注意判斷等於必須兩個==
y = 3
(x+y)/2 >= (x*y)**0.5
1 < x < 9
```

邏輯表示式的結果一定是真或假： True, False。

python 可以寫 1<x<9 但其他語言不行，所以不太建議這樣寫。字串也可以比較

```
s1 = 'Python'; s2 = 'python'; s3 = 'algorithm'
s1 == s2
s1 <= s2
s1 <= s3 <= s2
```

字串的等於與不等於很好理解，字串的大小是甚麼意思呢？是字典順序，這個我們到後面會再介紹。

比較的結果可以存在變數，稱為布林變數，變數內的值只會是 True 或 False （注意大寫）

試試看

```
print(1<4)
x = (3<7); print(x)
y = False
```

數字轉換成布林值時：0 是假，非 0 是真。

字串轉換成布林：非空字串是真，空字串是假。

邏輯表示式的組合運算： and or not

邏輯表示式可以做組合運算，可能的運算有 and, or, not：

x<3 and y>0, 且, 兩者同時為真的時候才為真，否則為假

x<3 or y>0, 或，兩者有一個為真的時候為真，只有兩者皆假的時候為假

not x<3, 否, 真變假，假變真

(練習：直接在 shell 敲下邏輯表示式以及設定到布林變數，印出布林變數)

組合可以多個，注意運算順序，不清楚就加括號，一樣只用小括號，可以多重。

```
x = (1<4); y=(1>4)
x and 3<7 or y
not x or 3<7 and y
(x or ((not x) and (y or 2<4))) or (not 1<3)
(x and not y) or (not x and y)  # exclusive or 兩者恰為一真一假
```

笛摩根定律

不偷不搶的否定是什麼？又偷又搶？

x = (not a) and (not b)

not x 等價於 a or b

偷或搶，並不是又偷又搶。

又高又帥的否定並不是不高又不帥，應該是

不高或不帥

not(a and b) 等價於 (not a) or (not b)

and or 的否定稱為笛摩根定律。

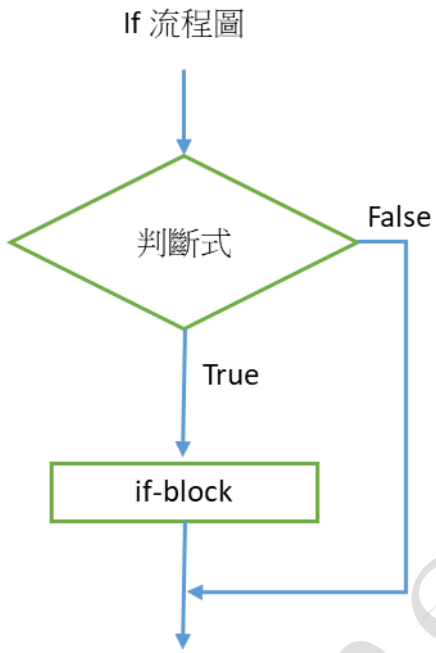
數理邏輯與一般口語上的意義有差異。

程式中不適合寫太複雜的邏輯表示式，可以拆開來寫。

8. 控制程式流程—分支選擇指令

分支：根據不同狀況執行不同的指令，就像是道路的分岔，程式的流程會根據當時狀況選擇其中一條路往下走。

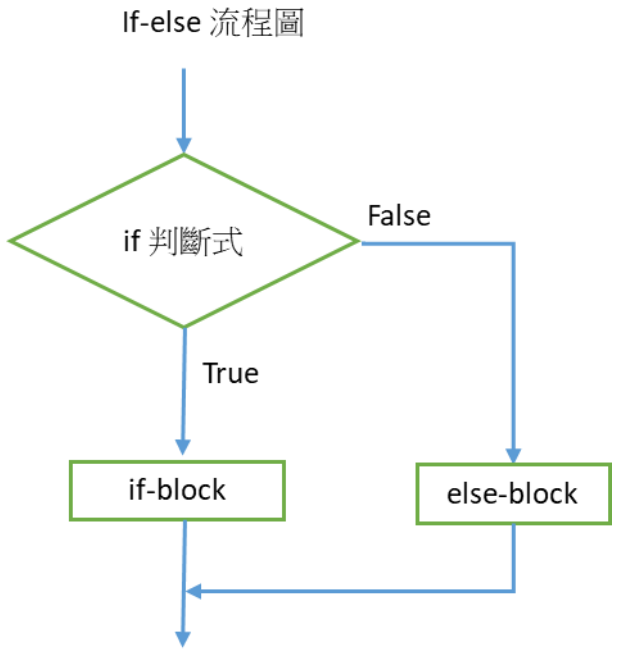
簡單型 if 的流程圖




```
if 2<3: print(5)
```

(shell 中敲下 if 指令後要多敲一行才會顯示)

if-else 流程圖




```
if a<b: c=a
else: c=b
```

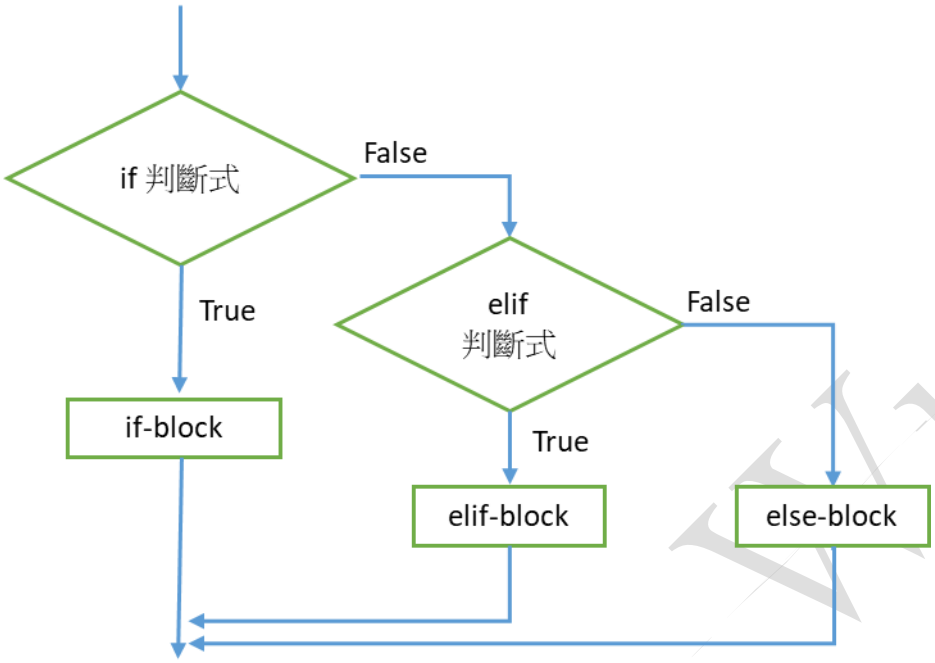
注意冒號，這個寫法不適合 if 中有多個指令的時候，用換行後縮排的方式，在 if 中的多個指令都要縮排

```
if a<b:
    c=b
    d=b-a
else:
    c=a
    d=a-b
```

縮排可用[tab]鍵，也可以用若干空白(習慣上用 4 個，有人喜歡 2 個)，但同一個區塊要一致，不要混用。

if 的完整流程

If 完整流程圖，可以有多個 elif




```
if score>90:
    grade='A'
elif score>80:
    grade = 'B'
elif score>70:
    grade = 'C'
elif score>60:
    grade = 'D'
else: grade = 'F'
```

巢狀 if，if 內有 if

if 區塊內(縮排的部分)可以是任一段程式，當然也可以有另外一個 if

```
if x > 5:
    print(x)
    if (y>3): print(y)
    else: print(y+8)
else:
    print(x+3)
```

練習：(示範)

輸入一個整數，輸出他的絕對值(用 if 判斷>0)

輸入一元二次方程式的三個係數 a,b,c，輸出'different roots', 'same root', 'no real root'

輸入三個整數 a,b,c 利用 if 將它們由小到大輸出於一行

=>窮舉判斷也可以用多層 if 判斷

ZJ 練習題：a006 一元二次方程式(示範)

輸入三個整數要先讀進來

```
a,b,c = map(int, input().split())
```

計算判別式

```
d = b**2 -4*a*c
```

根據判別式分成三種狀況

```
if d>=0:
elif d==0:
else:
```

第一種狀況：

```
x1 = (-b + d**0.5)/(2*a)
x2 = (-b - d**0.5)/(2*a)
```

但這樣寫有點問題，題目雖有保證結果是整數，但程式出來會是浮點數，要寫成

```
x1 = (-b + int(d**0.5))//(2*a)
x2 = (-b - int(d**0.5))//(2*a)
```

第二三種情形比較簡單。

9. 控制程式流程—for 迴圈

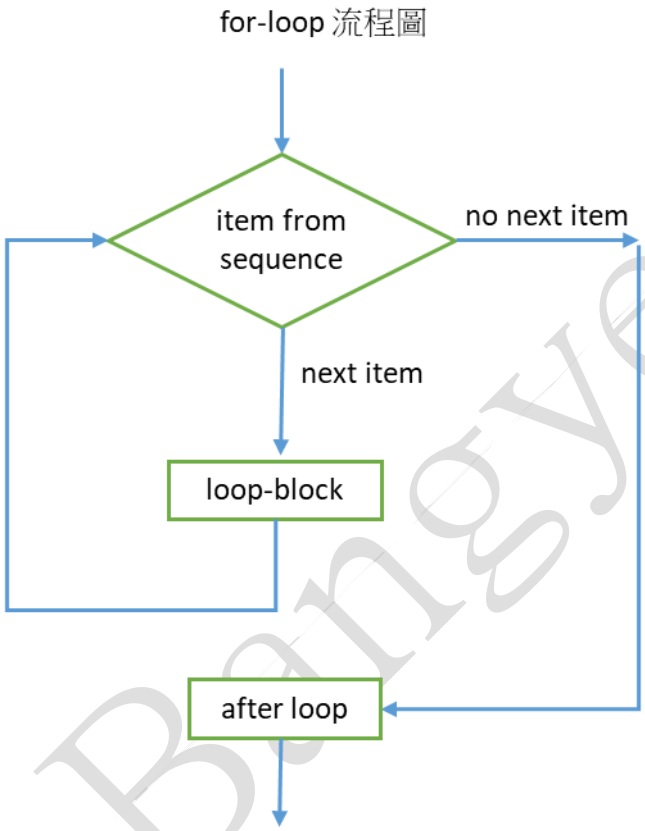
迴圈是重複執行一段程式直到某個條件為止：

跑操場直到太陽下山、吃到撐不下、睡到自然醒。

迴圈是一種分支指令的變型，因為常用且重要，單獨列出一種程式控制結構。

相同的指令要重複執行多次，不管資料對象是否相同，就是使用迴圈的時機。

for-loop：對一群資料的每一個，要做相同(類似)的事，就可以運用 for-loop。



for-loop ：對於在 A 中的每一個成員 x，一一執行以下指令。所謂 A 中的每一個是有固定順序的，也就是 A 中的順序。也就是，對一群資料都要執行某段程式，一個一個來、依照順序來、每個都要來。而所謂的一群資料常見的有：某個範圍的整數、在 List 中的每個成員、在字串中的每個字元。

某個範圍的整數

```
for x in A:
```

最常見的 A 是 0,1,2,3,4…直到正整數，記住這個用法

敲下 range(5)看看。range(n)就是 0,1,2,3,…, n-1。

for i in range(5): print(i) #對於 0,1,2,3,4 一一印出

跟 if 一樣，迴圈內可以是一行指令，也可以是一段(非常大段)的指令。

python 用縮排來表示哪一個區段是迴圈內的指令，哪個地方開始離開了迴圈。

```
n=10
for i in range(n):
    x = i*2+3
    print(x)
print('after loop')
```

range 的一般型 range(a, b, c) 從 a 到 <b 每次增加 c 的等差數列，
a, a+c, a+2c, …。注意最後是 < b，也就是 b 是不在內的，一個[a,b)的左閉右開區間。

```
for a in range(3, 20, 4): print(a)
```

範例題：

輸入正整數 n，將<n 的正整數全部加起來 (練習不用公式)

如果產生小於 n 的所有正整數? range(1,n)

如果把一些東西加起來？用一個變數紀錄總和(總和變數)，將這群東西一一加進去

```
total = 0 # 做人勿忘初衷，變數勿忘初值
for i in range(1,n): # 一個一個來、依照順序來、每個都要來
    total += i  # 加進 total
```

(練習)輸入正整數 n，將<n 的所有正奇數字全部加起來 (練習不用公式)

輸入正整數 n，判斷 n 是否為質數。

n 是質數 ⇔ 沒有因數 ⇔ 2,3,…, n-1 都無法整除 n

⇔ n%i !=0 for i in range(2,n)

要怎樣判斷這些東西都不能整除呢？

```
no_factor = True
for i in range(2,n):
    if n%i == 0:
        no_factor = False # 只要有一次就永遠都是假了
```

```
if no_factor:print(n, 'is a prime')
else: print(n, 'is not a prime')
```

因數的範圍可以只檢測到 n/2 甚至根號 n，如何修改?

range(2, int(n**0.5 + 0.5)) 避免開根號造成捨位誤差

List 的每個成員

迴圈搭配 if 是二級分題目很常使用到的運用，再搭配 list 就是三級分最重要的技巧，不管是不是 APCS，程式大部分的運用就是 loop+if+list(array)

回顧一下 list

```
a = [1, 6, 3, 5, 4, 3, 1]  # a[0]=1, a[1]=6, a[2]=3
```

for 搭配 list 有兩種用法：需要 index 與不需要 index

```
isum=0
n = len(a)  #length of a
for i in range(n):
    isum += i*a[i]
```

index 不需要的時候可以這樣用

```
isum = 0
for x in a:
    isum += x*x
```

前面教過 max(a) 會找出 list a 中的最大元素，這是利用內建函數，我們也可以用基本指令自己寫，這也是 loop+if 很基本的練習。在一群東西找出最大者，我們可以想像成一群人打擂台，擂台上有一個衛冕者寶座，一開始隨便一個先坐上衛冕者寶座，然後其他人，一個一個來，全部都要來跟衛冕者挑戰，如果輸了就離開，如果挑戰者贏了就由挑戰者當衛冕者。當全部的人都挑戰過了，留在衛冕者寶座上的就是冠軍。要實現這個想法，我們設一個變數 winer

```
a = [1, 6, 3, 5, 4, 3, 1]
winner = a[0]
for x in a:
    if x > winner:
        winner = x
```

是不是很簡單。有時我們也需要知道冠軍在哪個位置，這時可以使用 index 的 for 迴圈

```
a = [1, 6, 3, 5, 4, 3, 1]
winner = 0
for i in range(len(a)):
    if a[i] > a[winner]:
        winner = i
```

```
print('winner is a[',winner,'] =', a[winner])
```

(練習題)

計算費氏數列的前 10 項 f(1)=f(2)=1, f(n)=f(n-1)+f(n-2) for n>2

前綴和：輸入一個數列 a，計算前綴和 b，對每一個 i，b[i] = a[0]+a[1]+…+a[i]。

輸入第一行是 n，第二行是含 n 個整數的數列，輸出數列中所有奇數的和。

輸入第一行是 n 與 x,y，第二行是含 n 個整數的數列，計算數列中在[x,y]區間的平方和。

字串中的每個字元

一個字串是由字元組成，字串與 list 其實是很像的。例如要計算一個字串中有多少個'x'，我們可以將字串中的每個字元來拿檢查一下。

```
num =0 # 紀錄'x'的個數
s = 'thexFFxg X ggXxb'
for c in s: # 對於 s 中的每個字元 c 做以下事情
    if c == 'x': #如果 c=='x' 則 num 加 1
        num += 1
print(num)
# s.count('x')
```

(練習題)輸入一個字串，計算其中大小寫的字母個數。

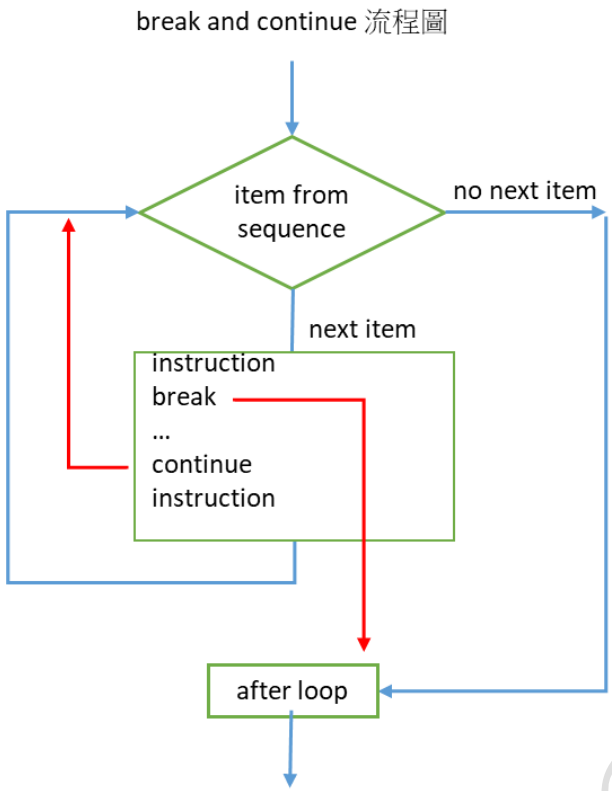
(練習題)輸入一個字串，計算每個字母(不分大小寫)的出現次數，輸出出現次數最多的字母以及其出現的次數。

提示：需要 26 個記數器？很多個變數時可否一次"啟用"(宣告)很多個變數？List 就是一次很多個，利用 ASCII 的特性與 list，可以使用一個 List 當 26 個記數器'A'紀錄在 L[0], 'B'紀錄在 L[1],…

提早結束迴圈 break 與 continue

break：跳離迴圈，到迴圈結束的下一個指令。

continue：略過本回合迴圈剩下的部分，跳到迴圈的起點，是否進入迴圈的下一回合要看是否到達迴圈結束條件。




```
for i in range(10):
    if i%5 == 4: break  # 除以 5 餘 4 的話跳離迴圈
    print(i)
print('after loop')
```

輸出？

```
for i in range(10):
    if i%5 == 4: continue
    print(i)
print('after loop')
```

(練習題)

先輸入一個 list，再輸入 3 個數，找到這 3 個數在 list 中第一次出現的位置，如果找不到就輸出 no

sample input

```
10
5 7 -3 1 7 4 5 1 -3 2
-3
6
4
```

```
n = int(input())
a = [int(x) for x in input().split()]
k = int(input())
for i in range(n):  # range(len(a))
    if a[i] == k: break
# 迴圈有兩個可能出口，離開迴圈後必須檢查是哪一種情形
if a[i] == k: print(i)
else: print('no')
# 再重複查詢的程式兩次
```

如果要查 100 個 k 呢？重複執行某些程式碼就想到用迴圈，我們其實可以把查詢的程式碼包在一個迴圈內做 k 次就好了。

雙重迴圈，巢狀迴圈

迴圈內可以是任何一段程式碼，所以當然可以含有另一個迴圈。

```
for i in range(5):
    for j in range(3):
        print(i,j)
```

執行結果是什麼？ i 會從 0 ~ 4，對於每一個 i，j 都會從 0 ~ 2，因此會印出

```
0 0
0 1
0 2
1 0
1 1
1 2
…
4 0
4 1
4 2
```

(練習題) 輸入 n 印出類似下面的三角型，最下面一排是 n 個*

```
*
**
***
```

我們要印出 n 排，每排的星星個數從 1~n，要印 n 排就跑個迴圈做 n 次

```
for i in range(n):
    print(?)
```

i=0 時要印 1 個*，i=1 時要印 2 個*，…，也就是要印 i+1 個*。要印 i+1 個*可以想成 print('*')做 i+1 次，那麼跑一個迴圈來做，但注意印星星的過程不能換行，整排印完才換行

```
for i in range(n):
```

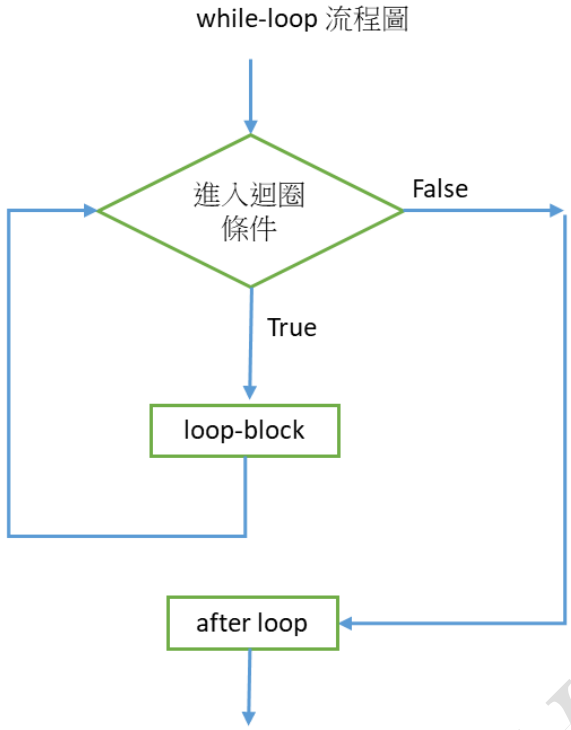
```
    for j in range(i+1):
        print('*', end='')
    print()  # 換行
```

運用 Python 的字串處理，也可以寫成這樣

```
for i in range(n):
    print('*' * (i+1)) # 字串 * 3 就是重複 3 次的意思
```

ZJ 簡單迴圈練習題:a005, a010(while), a024(while), a038(while), a059

10. 控制程式流程—while 迴圈



(ZJ a024)輾轉相除法計算最大公因數 gcd(x,y)。人怎麼算？若 y=mx+r，y 除 x 得商 m 餘數 r，則

gcd(y,x) = gcd(x,r)，且當 r=0 時，gcd 就是 x。根據這個定理就有輾轉相除法。

```
x = 48; y=60
while x>0: # 一直算到 x=0 為止
   r = y%x # 餘數
   y,x = x,r  # gcd(y,x) = gcd(x,r)
print(y)
```

也可以簡化寫成

```
x = 48; y=60
while x>0:
   y,x = x, y%x
print(y)
```

在迴圈內，我們永遠都是要求 gcd(y,x)，但 y 與 x 的值會在每一回合改變。程式的運作方式通常都是這樣，指令不動，資料變動，原因是執行相同的指令可以使用迴圈。就好像拿刀要切東西，刀裝在機器上不動，要切的東西一一運送過來，以物就刀而非以刀就物。

(練習題)計算 n 的因數總和

```
i = 1; ans=0; n=24
while i*i < n:
    if n%i == 0:
        ans += i + (n//i)
    i += 1
if i*i == n: ans += i  # 留意，根號 n 要單獨處理
print(ans)
```

(練習題 ZJ a010)算出 n 的質因數分解，例如 24 = 2^3*3, 60 = 2^2*3*5

如果對於每個小於等於 n 的數一一加以檢測是否為質數是否為因數，那麼會耗費太多時間。想像 n 由他的質因數組成，每當發現一個質因數時，就將該質因數從 n 中"抽離"，也就是把 n 除以這個質因數。那麼當我們計算到 i 時，n 中已經沒有小於 i 的因數，這時 i 如果能整除 n，則 i 必為 n 的質因數。

```
n = 45276
i = 2
first = True # 是否為第一個輸出的因數
while n>1:
    if n%i != 0:
        i += 1; continue
    times = 0 # 可整除，計算有幾個因子
    while n%i == 0:
        times += 1
        n //= i
    if not first: print(' * ',end='') # 如果不是第一個，要印*
    first = False
    print(i,end='') # 不要換行
    if times>1: print('^',times,sep='',end='') # times>1 才印指數
    i += 1
print()
```

這題有可以加快速度的地方，while 只需要檢測到 i*i <= n，否則剩下的 n 必然後是質數，結束 while 輸出的地方要做一些修改，留給各位自行研究。

(練習題)輸入正整數 n，將 n 的每一位數字全部加起來，例如 n=123 =>1+2+3

(練習題)輸入 n 與一連串 n 個非負整數，算出哪個數的每位數字總和最大

例如

```
4
10020 949 1381 99
```

=>輸出

```
949
22
```

(練習題 ZJ a038)輸入正整數 n，將 n 的每一位數字反轉順序變成另外一個數 m，例如

n=123 => m=321

n=120 => m=21

(練習題 ZJ a149)輸入正整數 n，將 n 的每一位數字乘起來。(小心輸入 n=0 的情形)

(練習題)reverse and add：輸入正整數 n，m 將 n 反轉得到的數，將 n+m 存回 n，重複以上步驟直到 n==m (這題在學字串反轉後會有比較簡單的做法)

11. APCS 實作二級分題目範例

APCS 實作題考試

考試時間兩個半小時，一共四題，每題 100 分，共 400 分。官網雖未公佈，但以往都一直是難度由簡到難。第一題就是二級分的題目(輸入輸出運算 if 與迴圈)，第二題是三級分的題目(陣列 List 的模擬運用)，第三第四題則是四五級分的題目(資料結構與演算法)。每一題總共 20 筆測資，每通過一筆測資得 5 分，每題可能分 2~3 個子題，子題會對輸入資料做一些特別的限制，也就是說，如果該題不會做 100 分的程式，可以嘗試只通過某子題的解法(看了題目後會更清楚)。

總分 50(含)以上是二級分，所以只要答對第一題的前半題就可以得到二級分，答對第一題與第二題的前半題則可以達到三級分(150 分)。此外，第三第四題雖是難題，但可能有簡單的子題可以拿到 20~40 分，雖然不能得到四級分，但可以做為三級分的保險分數。

要參加 APCS 考試還是應該先到官網看考試的相關資訊，另外可以加入一些臉書社團得到一些分享的訊息(臉書 APCS 實作題檢測社團)。以下兩點是考試時要特別注意的事情：

- 繳交後系統會幫你測範例，但不代表你的得分。
- 可以修改多次繳交但只有最後一次才會被計分

考試看到題目後，先思考要如何解題，想好程式的流程。第一第二題都是簡單的模擬操作，基本上根據題目的要求做就行了，初學者需要想的就是：

1. 輸入如何讀進來；
2. 要用什麼變數記錄什麼東西；
3. 輸出的要求是什麼。

考古題解題示範

以下看解題範例說明，這些都是 ZJ 上有的題目，大家可以很方便的自己做練習，建議不要每題都看解答說明，看了幾題當學習之後，剩下的題目安排時間自己練習想看看，當作模擬測驗，做完了再看範例題解的說明。否則一直看解答，自己的思考能力不會進步。以下是題目列表。

- c294. APCS-2016-1029-1 三角形辨別
- c290. APCS 2017-0304-1 秘密差

- c461. apcs 邏輯運算子 (201710)
- e286. 籃球比賽(201906)
- f579. 1． 購物車 -- 2020 年 7 月 APCS
- f312. 1． 人力分配 -- 2020 年 10 月 APCS
- f605. 1． 購買力 -- 2021 年 1 月 APCS
- g275. 1． 七言對聯 -- 2021 年 9 月 APCS
- g595. 1． 修補圍籬 -- 2021 年 11 月 APCS

c294.三角形辨別(APCS201610_1)

(ZJ 看題目)

本題要判斷輸入的三個正整數是否可以構成三角形，如果可以可以構成三角形，則進一步判斷此三角形是直角、銳角或鈍角三角形。題目中已經給了判斷三角形的方法，所以只要依照題目給的條件來做就行了，要注意的是，輸出時必須先將輸入的三個數字從小到大輸出，再輸出判斷結果。

(輸入怎麼讀、要做什麼事、要輸出什麼)

根據題意，要先將把三個邊長由小到大輸出，因此要找出由小到大的三邊長度，因為只有三個整數，我們可以挑出最小值與最大值，那麼中間值怎麼找呢？可以用總和減去最小與最大值。三角形的判斷就根據流程規劃中的方式來做。注意 Python 的 else if 是用 elif。

```
a,b,c = map(int, input().split())
e1 = min(a,b,c)
e3 = max(a,b,c)
e2 = a+b+c-e1-e3
print(e1,e2,e3)
if e1+e2 <= e3:
    print('No')
elif e1*e1+e2*e2 == e3*e3:
    print('Right')
elif e1*e1+e2*e2 < e3*e3:
    print('Obtuse')
else:
    print('Acute')
```

我們也可以用交換的方式來將三個數字排序。Python 交換兩個變數的方式很簡單

```
a, b = b, a
```

就可以將 a 與 b 的內容交換，原因呢？因為 = 是賦值(assign)，分解動作是將等號右邊的值取出來放入等號左邊的變數中。

會做交換了，那要如何排序三個數字呢？我們先把最大值 swap 到 c，然後再把 a 與 b 的位置擺對。看下面的範例程式，我們用兩個 if：如果 a>c 就交換，如果 b>c 就交換。這時 c 就一定換成最大值了。接下來檢查看看 ab 的順序是否正確，若不正確就交換。這樣 abc 的順序就排好了。

```
a,b,c = map(int, input().split())
#swap max to c
if a>c:
    a,c = c,a
if b>c:
    b,c = c,b
# swap min to a
if a>b:
    a,b = b,a
print(a,b,c)
if a+b <= c:
    print('No')
elif a*a+b*b == c*c:
    print('Right')
elif a*a+b*b < c*c:
    print('Obtuse')
else:
    print('Acute')
```

如果會使用 list 與 sort，這一題的寫法就非常簡單了。第 1 行在抓取輸入時直接轉換成數字的 list，在第 2 行將它排序，然後在第 3 行。

```
e = [int(x) for x in input().split()]
e.sort()
#print(e[0], e[1], e[2])
print(*e)
if e[0] + e[1] <= e[2] :
    print('No')
elif e[0]*e[0] + e[1]*e[1] == e[2]*e[2] :
    print('Right')
elif e[0]*e[0] + e[1]*e[1] < e[2]*e[2]:
    print('Obtuse')
else:
    print('Acute')
```

初學者可能還不會 list 與 sort，可以先用土法煉鋼來做，但稍微熟練之後建議要學習 list 與 sort，因為到三級分以上還是必須要使用的。

c290.秘密差(APCS201703_1)

輸入一個正整數，將它的每個位數拆解出來，奇數位數與偶數位數分別相加後，再計算兩者差值的絕對值就可以了。如果只想到使用數字的方式來處理，就必須將一個整數的每一位數找出來，這個流程是很多練習題的基本運算，它需要的是兩個知識：

(1). 一個整數除以10的餘數就是它的個位數字；

(2). 一個整數除以10的商就是於它右移一位的結果。

重複利用這兩個特性可以將一個整數的各個位數逐一求出。例如對於第 1 子題，因為輸入固定是 4 位整數，我們可以利用數學的思考把 4 個位數求出：

```
# 求出一個四位數 x 的各個位數
x = int(input())
d0 = x % 10;
d1 = (x // 10) % 10;
d2 = (x // 100) % 10;
d3 = (x // 1000) % 10;
diff = d0 + d2 - d1 - d3;
if diff<0: diff = -diff
print(diff)
```

上述方法的壞處是只能處理固定四位數(或者四位數以下)的數字，很多剛開始學程式的初學者會這樣寫，是因為缺少「迴圈」的觀念。第 2 子題是九位以內的整數，如果仿照這個方法寫，程式碼會變得很長，而且如果是 100 位或甚至 1000 位怎麼辦呢？

我們要引入迴圈的思維，迴圈是重複執行一段程式碼，也就是說，做很多次一樣的事情。在這個題目裡，我們可以每次都求除以 10 的餘數來取出個位數，「除 10 取餘數」這件事不改變，我們去改變資料，每一次將數字右移一位讓下一位數字變成個位數，這樣就可以重複運用「除 10 取餘數」這個運算來逐一取出各個位數。以下是運用迴圈的流程，因為我們需要知道是奇數或偶數位數，所以用一個迴圈變數 i 會較方便。因為 Python 是可以處理大數運算(與 C 不同)，所以第二與第三子題都可以通過。

```
x = int(input())
odd = 0 # 分別存偶數與奇數位數的和
even = 0
i = 0 # 第 i 位
while x!=0:
    d = x%10 # 每次都取出個位數
    if i%2==1: # 奇數
        odd += d
    else: # 偶數
        even += d
    x = x//10  # 右移一位
    i += 1

diff = odd - even
if diff<0: diff=-diff
```

```
print(diff)
```

其實把輸入的數字看成輸入一個字串，字串原本就是字元的陣列，所以只要把每一個數字字元轉換成整數後，利用迴圈將陣列的奇數與偶數分別相加求差值就可以了，這個流程更為簡單，這裡我們運用了絕對值函數 abs()。

```
line = [int(x) for x in input()]
diff = 0
for i in range(len(line)):
    if i%2: diff += line[i]
    else: diff -= line[i]
print(abs(diff))
```

c461.邏輯運算子(APCS201710_1)

本題中給了三種邏輯二元運算的真值表，輸入 a, b, r，問你「若 a X b = r，則 X 可能是三種運算中的哪幾種。」題目中所列的三種運算是真實的 AND、OR 與 XOR，前兩者多數人都熟悉，XOR 有些人可能不熟悉。事實上，這個題目只要依照要求來做驗算與輸出，即使題目定義任意不存在的運算也都是可以做的。

程式的流程非常簡單，先輸入 a, b, r，再依照題目給的運算順序逐一判斷是否滿足運算的要求，如果都不滿足，要輸出 IMPOSSIBLE，為了要知道是否是 IMPOSSIBLE，我們介紹以下常用的方法：

如果想要知道在某一段運算之中是否曾經發生過某事件，可以在進入這一段之前先設立一個旗標變數為 False，在運算過程中只要該事件發生就將旗標設為 True。

以下是範例程式，請注意互斥或 exclusive or 的寫法。

```
a,b,c = map(int, input().split())
flag = False
if a!=0: la = True
else: la = False
if b!=0: lb = True
else: lb = False
if c!=0: lc = True
else: lc = False
if (la and lb) == lc :
    print('AND')
    flag=True
if (la or lb) == lc :
    print('OR')
```

```
    flag=True
if ((la and (not lb)) or ((not la) and lb)) == lc :
    print('XOR')
    flag=True
if not flag: print('IMPOSSIBLE')
```

a 是否不為 0 這件事也可以直接用下面的寫法。

```
a,b,c = map(int, input().split())
flag = False
la = (a!=0)
lb = (b!=0)
lc = (c!=0)
if (la and lb) == lc :
    print('AND')
    flag=True
if (la or lb) == lc :
    print('OR')
    flag=True
if ((la and (not lb)) or ((not la) and lb)) == lc :
    print('XOR')
    flag=True
if not flag: print('IMPOSSIBLE')
```

e286.籃球比賽(APCS201906_1)

本題第一個步驟是要計算出兩隊得分，所以我們設計一個變數儲存主隊的得分，一個變數儲存客隊的得分。因為有兩場比賽，所以這個步驟要執行兩次。此外，最後還要判斷出主隊勝了幾場，題目保證每場比賽一定有勝負，只要計算主隊勝了幾場，就可以判斷主隊兩場比賽的結果是勝是負或是平手。為了判斷兩場比賽的結果，除了兩隊總得分之外，還需要一個變數計算主隊的勝場數。

```
01 win = 0
02 a = [int(x) for x in input().split()]
03 host = sum(a)
04 a = [int(x) for x in input().split()]
05 guest = sum(a)
06 if host > guest :
07     win += 1
08 print(str(host)+':'+str(guest)) # print(host, guest, sep=':')
09 # second game
10 a = [int(x) for x in input().split()]
11 host = sum(a)
12 a = [int(x) for x in input().split()]
```

```
guest = sum(a)
if host > guest :
    win += 1
print(str(host)+':'+str(guest))
if win == 2: print('Win')
elif win == 0: print('Lose')
else: print('Tie')
```

下面的範例程式使用迴圈架構並合併輸入與加總的指令，這樣可以寫得更簡短一些。

```
win = 0
for i in range(2):
    host = sum([int(x) for x in input().split()])
    guest = sum([int(x) for x in input().split()])
    if host > guest: win += 1
    print(str(host)+':'+str(guest))
    # print(host, guest, sep=':')
if win == 2: print('Win')
elif win == 0: print('Lose')
else: print('Tie')
```

f579.購物車(APCS202007_1)

對於每一筆清單，要檢查是否 a 與 b 兩種商品皆有購買，最後輸出在多少筆清單中兩種商品皆被購買。商品 a 與 b 是先指定好的，所以對於每一筆清單，我們只要關注 a 與 b 即可，其他商品都可忽略。商品可能拿進來又移出去，那麼我們可以將拿進來次數減去移出次數來計算淨購買量就可以了。

```
a, b = map(int, input().split())
t = int(input())
ans=0
for i in range(t):
    l = [int(x) for x in input().split()]
    na=0; nb=0
    for x in l:
        if x==a:
            na += 1
        elif x==-a:
            na -= 1
        elif x==b:
            nb += 1
        elif x==-b:
            nb -= 1
    if na>0 and nb>0:
        ans += 1
```

```
17 print(ans)
```

利用 count()函數來計算元素在 list 中出現次數，可以簡化程式碼。

```
a, b = map(int, input().split())
t = int(input())
ans = 0
for i in range(t):
    l = [int(x) for x in input().split()]
    na = l.count(a) - l.count(-a)
    nb = l.count(b) - l.count(-b)
    if na>0 and nb>0:
        ans+=1
print(ans)
```

f312.人力分配(APCS202010_1)

定義 y1 與 y2 分別是 x1 與 x2 的二次函數。對於給定人數 n，要將 n 分成 n=x1+x2 兩個整數，使得 y1+y2 要越大越好。題目的說明與舉例中，看得出來是要檢查所有(x1,x2)的可能性，並在其中挑選最大值，所以只要依照題意以迴圈來枚舉嘗試所有可能性就可以了。

這一題的主要架構是枚舉每一種將整數 n 分成兩個整數的可能，對於每一個分割，計算出獲利，在這些獲利中取出最大值。將一個整數 n 分成 x1 與 x2 兩個整數且 x1 + x2 = n，我們可以枚舉所有的 x1，然後 x2 = n - x1。

```
a = [int(x) for x in input().split()]
b = [int(x) for x in input().split()]
n = int(input())
best = -100000000
for x1 in range(n+1):
    y1 = a[0]*x1*x1 + a[1]*x1 + a[2]
    x2 = n-x1
    y2 = b[0]*x2*x2 + b[1]*x2 + b[2]
    best = max(best, y1+y2)
# end for
print(best)
```

f605.購買力(APCS202101_1)

有 n 個物品，每個物品有三個價格，先檢查這三個價格的最大價差是否達到輸入中給定的門檻值 d，如果有達到，則計算三個價格的平均值。最後要統計出有幾個物品的價差達到門檻值以及他們的均值總和。

這一題的 n 個物品之間是獨立沒有關係的，我們啟用一個迴圈，每次處理一個物品，我們需要一個變數來記錄買了多少物品以及一個變數來記錄總價。對於一個物品，我們只要會計算三個整數中最大與最小數的差值，然後判斷是否達輸入門檻，如果達到，就計算三個數字的平均。

```
n,d = map(int, input().split())
num = 0
total = 0
for i in range(n):
    p = [int(x) for  x in input().split()]
    if max(p)-min(p) >= d:
        num += 1
        total += sum(p)//3
    # end if
print(num, total)
```

g275.七言對聯(APCS202109_1)

一首七字對聯兩句，每句七個字。輸入這 14 個字的平仄，0 代表平聲而 1 是仄聲，目標是判斷題目中定義的三個規則有無違反。對於每個規則，可以根據它的定義來直接做 0 與 1 的比對來判定，所以只要適當的安排使用 if 來判斷就可以了。

完全解有 n 首對聯，每一首對聯是各自判斷，所以外層可以用一個迴圈執行 n 次，迴圈的每一次判斷一首對聯。對於一首對聯，要判斷違反了 ABC 中的哪幾條規則，有違反的規則要依照順序輸出規則編號，我們先注意這裡不能用 else if，因為不管 A 有無違反，規則 BC 都是需要檢查的。我們碰到的第一個問題是題目要求所有規則都沒有違反時要輸出 None，那麼上述的流程中何時要輸出 None 呢？類似的情況常常遇到，常用的解決方法是一開始豎立一個旗號，也就是設定一個變數 no_error = 1，表示無違反。在檢查每一條規則時，如果違反就將旗號設為 no_error = 0。這樣，在最後只要檢查旗號就知道是否都沒有違反。注意第幾個字的編號由 0 開始，所以與題目的編號差 1。

```
n = int(input())
for i in range(n):
    p = [int(x) for x in input().split()]
    q = [int(x) for x in input().split()]
    no_error = True
    if p[1]==p[3] or p[1]!=p[5] or q[1]==q[3] or q[1]!=q[5]:
        print('A',end='')
        no_error = False
    if p[6] != 1 or q[6] != 0:
        print('B',end='')
        no_error = False
    if p[1]==q[1] or p[3]==q[3] or p[5]==q[5]:
        print('C',end='')
        no_error = False
    if no_error:
        print('None')
    else: print()
```

這一題的輸入可以不一定需要轉換成整數，這裡我們直接用字串的方式來寫。下面的範例中的第 2~3 行，我們用 list comprehension 的方式讀入一行，並將他以空格分隔的方式切成若干字串放在 List 中。此外為了可閱讀性，我們在 List 前方放入一個[0]，這樣編號位置就可以跟題目一樣從 1 開始。

我們的輸出是設在一個字串 ans 中，在規則判定時，如果要輸出某一條規則編號時，就將它附加入 ans 的尾端，例如 ans += 'A'就會將 A 放入目前 ans 的字尾，在規則判斷完畢後，我們用 len(ans)就可以知道 ans 是否為空的，如果是空的就要輸出'None'。

```
n = int(input())
for i in range(n):
    p = [0]+[x for x in input().split()]
    q = [0]+[x for x in input().split()]
    ans = ''
    if p[2]==p[4] or p[2]!=p[6] or q[2]==q[4] or q[2]!=q[6]:
        ans += 'A'
    if p[7]!='1' or q[7]!='0':
        ans += 'B'
    if p[2]==q[2] or p[4]==q[4] or p[6]==q[6]:
        ans += 'C'
    if len(ans)==0:
        ans='None'
    print(ans)
```

g595.修補圍籬(APCS202111_1)

輸入一個數列，0 表示需要修補的位置，修補的高度是左右比較小的數字，因為題目保證不會有連續的 0，所以只要對於每一個 0，把他左右較小的數字取出，然後對所有取出的數字加總就可以了。要注意的是邊界，左右邊界如果是 0 的話，要特別處理，因為左邊界只有右邊鄰居而右邊界只有左邊鄰居。

既然要加總，我們就需要設一個變數來存加總的和，這一題的完全解有多個數字要判斷，我們打算用迴圈來處理，但左右邊界不同，所以最簡單的方法是把左右邊界單獨處理，其他的位置就用迴圈一起處理。

```
n = int(input())
h = [int(x) for x in input().split()]
total = 0
if h[0]==0: total += h[1]
if h[n-1]==0: total += h[n-2]
for i in range(1,n-1):
    if h[i]==0: total += min(h[i-1],h[i+1])
print(total)
```

利用前後補上無限大的數字(這題超過 100 就可以)，我們可以將頭尾一併納入迴圈處理。甚至可以把迴圈寫入 sum 函數中，但並不鼓勵初學者這樣寫。我們不必追求太精簡的程式碼，而是要找自己思路可以理解的寫法。

```
n = int(input())
h = [1000]+[int(x) for x in input().split()]+[1000]
total = sum(min(h[i-1],h[i+1]) for i in range(1,n+1) if h[i]==0)
print(total)
```

12. 再探 List(串列)與字串操作

list 是個物件，有內建函數可以對這個物件進行操作用法是

```
obj.func()
```

obj 是物件變數的名字，func 是它內建函數的名字，底下看例子就很容易記得。

List 的串接、重複、尾端新增與刪除元素。

a 是一個 list, list 的相加是串接，list 的 append 是在其後增加一個元素，pop(i)是刪除第 i 個元素, -1 是指最後一個。執行尾端的增加與刪除是很快速的，但如果 list 很長的時候，刪除串列中間的元素可能會花費比較多時間，尤其重複執行多次。

```
a = [1,3,4]+[4,6,7,8] # [1,3,4,4,6,7,8]
b = [0,2]*3 #[0,2,0,2,0,2]
b *= 2 # [0,2,0,2,0,2,0,2,0,2,0,2]
a.append(9)
a.pop(-1)
```

list slicing 是指將 list 切出其中連續的一段，留意幾件事：

- 編號由 0 開始。
- 區間定義都是左閉右開(不含右端)。
- 負的編號是倒數，最後一個是-1。
- 第一個編號不寫代表開頭，第二個編號不寫表示一直到結尾都包含。

```
b = a[1:3]
c = a[-3:-1]
d = a[ :-2]
e = a[ -4: ]
```

也可以對 list slice 賦值，這就改變了 list 的內容，甚至改變 list 的大小

```
a[1:3] = [7, 8, 9]
a
a[1:4] = [0,0]
a
a[-3: ] = a[1:5]
a
a[ :0] = [9,9,9]
```

```
a
a[:] = []
a
```

善用 slice 可以做到很多事情，完整的 slice 包含 list[start,stop,step]，其中 step 表示每次增加幾個位置，step=-1 會反轉 list

```
a = [0,1,2,3,4,5]
a[ :4:2] # [0,2]
a[ : :3] # [0,3]
a[::-1] # [5,4,3,2,1,0] 這個很好用
a[ 2: :-1] # [2,1,0] 從 2 開始反向走到頭
```

list comprehension，以說明 List 內容的方式建構一個 list。相當於把 for 迴圈內含在 list 建構中，可以雙迴圈，甚至可以加 if 條件式來過濾成員。

```
a = [ 2*i+1 for i in range(6)]
a = []
for i in range(6): a.append(2*i+1)

b = [ x for x in a[2:5]]; # b = a[2:5]

c = [ x**3 for x in a]; d = [ x//3 for x in a if x%5 !=0]
d = [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
```

字串：與 list 操作類似，相加是串接，乘整數是重複。

```
a = 'This is a Book.'
b = a + ' ' + 'I love Python.'
s = 'abc' * 3 # abcabcabc
s *= 2
for c in b[3: 8]: print(c)
d = ''  # empty string
for x in b:
    if x>='A' and x<='Z': d += x # b 中大寫字母擷取出來的字串
e = [ x for x in b if x>='a' and x<='z'] # list consisting char
```

字串與 list 一樣可以做 slice，但字串內容不可更改。

```
a = 'This is a Book.'
a[1] # h
a[1:6] # 'his i'
a[1] = 'x' # error
b = a[:4]+'PY'+a[6:]
c = a[::-1] # 反轉字串，記一下這用法
```

字串與 list 都可以用 in 檢查是否為元素的判斷式，但如果字串與串列很長時，可能花費很多時間。注意字串可以檢查字元也可以檢查子字串。

```
a = [5, 4, 1, 7, 8]
if 3 in a: print('3 is in a')
else: print('3 is not in a')
4 in a # True
```

```
[5,4] in a # False
s = 'This is a book'
'T' in s # True
'b' in s # True
'S' in s # False
'is' in s # True
'The' in s # False
```

(練習題)輸入一個字串，計算有幾個大寫字母

(練習題)計算有幾個 word(被一個或多個空白間格) --- 自己算或是運用內建函數 split()

```
s = input()
nword = 0
state = 0 # 0=space, 1=not_space
for c in s:
    if c == ' ' and state == 1:
        state = 0
    if c != ' ' and state == 0:
        state = 1 # a new word start
        nword += 1
print(nword)
print(len(s.split()))
```

(練習題 ZJ d673)No problem，簡單陣列模擬題

(練習題 ZJ c015)用字串做 Reverse and add:輸入一個正整數，將它反轉後與原數字相加，重複這步驟直到該數字變成一個迴文。(注意本題輸入的是迴文數字的話答案不是 0 次，也就是至少做一次後才開始判斷)

```
ncase = int(input())
for it in range(ncase):
    n = int(input())
    n += int(str(n)[::-1])
    times = 1
    while True:
        s = str(n)[::-1] # reverse string of n
        m = int(s)
        if n == m: break
        n += m
        times += 1
    print(times,n)
```

**List 與字串有些函數可用，但初學或許不容易記太多函數，其實只要有基本的知識，有些函數功能自己做也不會耗時太久，ASCII 不需要記，可以用 ord()轉換得來，字串處理記得 ASCII code 的以下性質即可： 0~9 是連續的, A~Z 是連續的, a~z 是連續的。

13. list of list 實作二維陣列

二維矩陣

List 中可以擺任何型態的資料

```
a = [1, 3.1416, 'Python', 'x']
```

list 中可以放 list。

```
b = [1, [2,3,4,5], [6,7]]
b
b[0]
b[1]
b[1][2]  # b[1]是一個 list，b[1][2]則是 b[1]的第 2 個
```

所謂二維陣列，也可以說是二維矩陣，可以看成一張表格，m*n 的矩陣有 m 個橫列(row)與 n 個直行。台灣 row 稱為列，column 稱為行，但中國大陸剛好與我們相反。以下是個 3*4 的陣列 a。

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Python 用 list of list 來實作二維陣列，也就是 a 是一個有 m 個元素的 List，而 a 的每個元素都是一個有 n 個元素的 list。如果執行下面的程式

```
a = [[0]*4 for i in range(3)] # 初始化一個 3*4 矩陣，初值均為 0
b = [[0]*4]*3 # 這樣寫是錯的 b[1] = b[0]; b[2] = b[0]，但 b[0]是 List
c = [[1,2,3]]*2 # repeat 相當於 c[0]=[1,2,3]; c[1]=c[0]
x = 0
for i in range(3): # for each row
    for j in range(4): # for each column
        a[i][j] = x # 存取元素的方式
        x += 1
# end
```

陣列內容為

0	1	2	3
4	5	6	7
8	9	10	11

也就是 a = [[0,1,2,3],[4,5,6,7],[8,9,10,11]]。這例子是簡單讓大家了解二維陣列與雙迴圈搭配與執行順序。二維陣列常用二層迴圈，以下是另外兩個簡單的例子。

```
a = [0]*5
b = [[0]*5 for i in range(3)]
for i in range(3):
    for j in range(5):
        b[i][j] = i*100+j
```

建一個九九乘法表

```
t = [[0]*10 for i in range(10)]
for i in range(1,10):
    for j in range(1,10):
        t[i][j] = i*j
```

常見的二維陣列的輸入格式是由上而下由左而右，以上面的例子來說，若輸入資料為

```
3 4
0 1 2 3
4 5 6 7
8 9 10 11
```

第一行出現的兩個數字是 m 與 n，我們可以用以下方式讀入：

```
m,n =map(int, input().split())
a = []
for i in range(m):
    a.append( [int(x) for x in input().split()])
```

也可以先將 a 初始化為 m 個空 list

```
m,n =map(int, input().split())
a = [[] for i in range(m)]
for i in range(m):
    a[i] = [int(x) for x in input().split()]
```

比較推薦在使用前先初始化變數與 list，除非事前無法得知元素個數。

轉置矩陣與矩陣乘法

這一節我們透過說明一些線性代數的運算，讓大家一方面了解矩陣的基本運算，一方面了解 Python 的二維陣列操作。

向量內積：一個向量就是一個數字的 List，例如 a = [1,2,4,0]，兩個向量的內積就是將對應項次相乘後再將所有的乘積相加，所以兩個向量的內積是一個數字。例如 b = [3,0,1,1]，則 a 與 b 的內積 = 1*3 + 2*0 + 4*1 + 0*1 = 7。程式如下：

```
inner_p = 0
for i in range(n): # n =len(a)
    inner_p += a[i]*b[i]
```

如果運用 sum 函數也可以寫做：

```
inner_p = sum(a[i]*b[i] for i in range(n))
```

向量與矩陣相乘：一個 m*n 的矩陣 A 與一個 n 個元素的向量 x 相乘，結果是一個 m 個元素的向量 b，b 的第 i 個元素是 b[i]是 A 的第 i 列與 x 的內積。程式碼：

```
b = [0]*m
for i in range(m): # 算b[i]
    for j in range(n):
        b[i] += A[i][j]*x[j]
```

運用 sum()函數與列表生成式也可以簡潔的寫成：

```
b = [sum(A[i][j]*x[j] for j in range(n)) for i in range(m)]
```

矩陣轉置：一個 m*n 的矩陣 A，它的轉置矩陣是一個 n*m 的矩陣 B，A 的第 i 列就是 B 的第 i 行。也就是說 B[i][j] = A[j][i]，對所有 i,j。要求 A 的轉置，可以直接根據定義來寫：

```
B = [[0]*m for i in range(n)] # initial a n*m matrix
for i in range(n):
    for j in range(m):
        B[i][j] = A[j][i]
```

假設 A 是方陣(m=n)，有時我們需要將轉置後的矩陣放回到 A，可以很簡單的再轉置到 B 後複製回到 A

```
for i in range(n):
    for j in range(m):
        A[i][j] = B[i][j]
```

注意，單純的寫 A = B[:]是不行(很有危險)的，這樣做 A 與 B 是同一群 list，而非拷貝後獨立的兩個矩陣。B 的每一個元素都是一個 list，所以以下的寫法是可以的：

```
B= [ x[:] for x in A]
```

我們也可以直接在 A 的內部以元素交換的方式完成轉置，這樣可以不需要用額外的空間：

```
for i in range(n):#m=n
    for j in range(i+1,n):
        A[i][j], A[j][i] = A[j][i], A[i][j] #swap
```

我們對所有 i<j，交換 A[i][j]與 A[j][i]。注意程式中的雙迴圈 j 從 i+1 開始，如果對所有 i,j 皆做交換，則換過去又會換回來，整個陣列最後沒改變。

矩陣相乘：若 A 是 m*n 的矩陣，B 是 n*p 的矩陣，則兩個矩陣相乘 A×B 是一個 m*p 的矩陣 C，C[i][j]是 A 的第 i 列與 B 的第 j 行的內積。請注意，第一個矩陣的行數必須等於第二個矩陣的列數，否則無法相乘，因為兩個向量必須有相同的分量數才能做內積。矩陣相乘的程式可以根據上述定義來寫即可

```
c = [[0]*p for i in range(m)]
for i in range(m):
    for j in range(p):
        for k in range(n):
            c[i][j] += A[i][k]*B[k][j]
```

二維陣列走訪

很多題目都是模擬在二維陣列上，例如在一個方格棋盤上行走移動，或是在二維平面的整數點座標的處理。這一節我們提出一些常見在二維陣列上的模擬技巧。

位置與其鄰居：二維陣列的模擬題經常需要紀錄位置，有時候題目矩陣的位置(row, column)來說明，有時用的是平面座標(x,y)來說明。通常只要在題目中保持一致就可以，使用題目中給的方式即可，但要注意的是方向與 index 是不一樣的。若(r,c)表示在第 r 列第 c 行，則往上下左右分別是(r-1,c)、(r+1,c)、(r,c-1)與(r,c+1)；而座標的形式則是(x,y+1)，(x,y-1)，(x-1,y)與(x+1,y)。

陣列模擬題大部分都有邊界，要經常注意出界的問題。假設陣列名稱為 a 目前的位置在(r,c)，要檢查四個鄰居的最小值，我們可能用以下的寫法。

```
minr = r-1; minc = c # 紀錄最小鄰居所在的位置，初設在上方
if a[r+1][c] < a[minr][minc]:
    minr = r+1; minc = c
if a[r][c-1] < a[minr][minc]:
    minr = r; minc = c-1
if a[r][c+1] < a[minr][minc]:
    minr = r; minc = c+1
```

這個想法是正確的，但是沒有注意到出界的問題，因為(r,c)如果是在(上下左右)邊界上，以上的程式執行就會出現 index out of range 的錯誤。如果在每個要使用到陣列的地方都檢查是否超過邊界，我們會改成這樣：

```
minr = r-1; minc = c # 紀錄最小鄰居所在的位置，初設在上方
if r<m-1 and a[r+1][c] < a[minr][minc]:
    minr = r+1; minc = c
if c>0 and a[r][c-1] < a[minr][minc]:
    minr = r; minc = c-1
if c<n-1 and a[r][c+1] < a[minr][minc]:
    minr = r; minc = c+1
```

三個 if 改好了，但初始上方鄰居為最小的地方要怎麼改呢？如果要考慮各種情形會非常的麻煩，所以在找最小值(或最大)時有個常用的技巧：初設為一個大到不可能的值，在從第一個開始檢查。假設陣列內的數字是 10000 以下的整數，我們可以這樣寫：

```
imin = 10001; minr = -1; minc = -1 # 紀錄最小鄰居所在的位置，初設在不可能
```

```
if r>0 and a[r-1][c] < imin:
    imin = a[r-1][c]; minr = r-1; minc = c
if r<m-1 and a[r+1][c] < imin:
    imin = a[r+1][c]; minr = r+1; minc = c
if c>0 and a[r][c-1] < imin:
    imin = a[r][c-1]; minr = r; minc = c-1
if c<n-1 and a[r][c+1] < imin:
    imin = a[r][c+1]; minr = r; minc = c+1
```

注意到上面出界檢查的 if 中，都是先檢查 index 再檢查 a[][]的值，例如

```
if r>0 and a[r-1][c] < imin:
```

這裡要提醒 if 中 and 的順序是不可以顛倒寫成

```
if a[r-1][c]< imin and r>0:
```

Pythony 在運算 and 的邏輯式時，會先計算第一個邏輯式，如果第一個為真才計算第二個，否則略過，因為第一個為 False 的話結果必定為 False，這樣就可以避免在算第二個時會出界，這個稱為 short-circuit evaluation。如果顛倒順序的話，算第一個就發生出界的錯誤。

檢查邊界在陣列問題中經常出現寫起來又囉哩囉嗦，有沒有可以省事的方法呢？有兩個方式，一個是把出界檢查寫成一個自訂函數(下一章的主題)，另一個方式是自己在陣列的四周圍一圈不可能的值，以上面的例子來說，陣列中最大不超過 10000，而且每次都要找最小，我們可以把四周圍上一圈 oo=10001。例如在以下格式輸入時，我們加上外圍。

```
m,n =map(int, input().split())
oo = 10001; m += 2; n += 2 # 多兩列兩行
a = [[] for i in range(m)]
a[0] = [oo for j in range(n)] # 第 0 列 n 個 oo
for i in range(1,m-1):
    a[i] = [oo] + [int(x) for x in input().split()] + [oo] #左右加 oo
a[m-1] = [oo for j in range(n)] # 最後一列 n 個 oo
```

在檢查鄰居時就不必檢查出界了

```
imin = 10001; minr = -1; minc = -1 # 紀錄最小鄰居所在的位置，初設在不可能
if a[r-1][c] < imin:
    imin = a[r-1][c]; minr = r-1; minc = c
if a[r+1][c] < imin:
    imin = a[r+1][c]; minr = r+1; minc = c
if a[r][c-1] < imin:
    imin = a[r][c-1]; minr = r; minc = c-1
if a[r][c+1] < imin:
    imin = a[r][c+1]; minr = r; minc = c+1
print(minr, minc, imin)
```

但要注意兩件事：第一，原來的 r 與 c 編號在這個陣列中各增加了 1；第二，在某些狀況下須要檢查找出來的鄰居，如果它的值是外圍值，就代表要找的鄰居不存在。

另外一個常用技巧是針對常常在很多地方都要寫四個方向，所以類似的程式碼要寫四次，既然是重複的，可以有迴圈式的寫法嗎？

是可以的，我們可以使用向量的觀念，先定義在每一個方向的 r 差值與 c 差值
dr = [-1,0,1,0]; dc =[0,1,0,-1] # 上右下左
我們把四個方向訂一個順序"上右下左"，用"0,1,2,3"分別代表上右下左。也就是說(r,c)的上方是(r+dr[0],c+dc[0]) = (r-1,c)，右方是(r+dr[1],c+dc[1]) = (r,c+1) 。前面的程式碼可以改寫成

```
dr = [-1,0,1,0]; dc =[0,1,0,-1] # 上右下左
imin = 10001; minr = -1; minc = -1 # 紀錄最小鄰居所在的位置，初設在不可能
for d in range(4):
    nr = r+dr[d]; nc = c+dc[d];
    if a[nr][nc] < imin:
        imin = a[nr][nc]; minr = nr; minc = nc
```

其中 dr 與 dc 的設定只需要一次。而程式中使用到的地方可能有多處，因此這樣寫可以讓程式碼簡潔一些，但如果出現次數不多或者考試時沒空多想，多寫幾行也未嘗不可。此外這個方法還有其他好處，例如在某些題目中方向是有順序的，我們可以依照給定的順序改變定義；另外變化向量的概念也可以用來處理不同定義下的"鄰居"，例如象棋中的馬與西洋棋中的騎士。

14. 控制程式流程—自訂函數

程式的流程一共有四種：循序式(一行一行往下做)，條件分支(if)、迴圈(for, while)，以及函數呼叫。自訂函數雖然不是 APCS 三級分必須學的，但是還是希望這個課程中對重要的程式觀念有完整的了解，所以這一章我們簡略的介紹自訂函數的語法與運用方式。自訂函數的以指令 def 開頭，有一個函數名字，接著小括號中若干個參數，然後該行以冒號結尾。接下來是函數的定義，通常應該是在接下來的若干行，都要用縮排來表示是函數內部。特別簡短時也可以跟迴圈或 if 一樣，直接寫在冒號後面。

```
def sos(x,y): #定義一個函數，記得關鍵字 def, 冒號, 縮排
    t = x**2+y**2
    if t > 20:
        return x+y
    else :
        return t+x+y

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

括號內的是傳入的參數，函數可以有傳入值，例如 sos(x,y)中的 x 與 y，也可以沒有，例如 hello()。執行時呼叫函數就如呼叫內建函數是一樣的，例如學過的 max()與 input()。

```
hello()
if 3<5: hello()
a = [ 1, 3, 5, 4,2]
b = [ sos(x, 2) for x in a ]
```

對於簡短的程式(如 APCS)，自訂函數不一定是必須的，也可以寫成全部的程式都在一個主程式中，但函數主要目的有幾個：可以使得程式碼清晰易讀、避免重複的程式碼、使用遞迴。

函數內可以是任何一段程式，可以有回傳值，也可以沒有回傳值。呼叫時流程跳到函數開始處執行，執行到 return 指令或函數結束處則流程返回到呼叫處的下一個指令。函數內容可以有 if、有迴圈、有其他的內建或自訂函數，都可以。試著執行以下程式兩次，輸入給 0 與非 0

```
def fib(n):
    a, b =0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()



x = int(input())
```

```
if x==0:
    fib(10) # 呼叫
    print(' x is 0') # 回到此處
else:
    fib(20)
print(' after if')
```

再舉一例，在前一章要判斷二維陣列是否出界，我們可以寫成一個函數。

```
def out(n,m,r,c):
    if r<0 or r>m-1:
        return True
    if c<0 or c>n-1:
        return True
    return False
```

函數內可以有多個 return 指令，每次執行時只要碰到 return，程式流程就會回到呼叫處，函數內剩下的指令也就都不會執行了，在上述的例子中，如果兩個 if 中的任何一個成立的話，就會執行到 return True 的指令，如果兩個都不成立，就會在最後一行 return False。

全域變數(global)

試試以下程式

```
def f(x):
    y = x - 3
    if y<0: return 2*x+1
    return x-1

y = -1
print(f(3),y)
```

函數中改變了 y 的值，但回到主程式後，y 的改變並未帶回主程式，好像到學校上課，回家後卻變成什麼都沒發生，豈不太糟？

區域變數：變數的有效範圍只在副程式中，副程式中的 y 與主程式的 y 是同名同姓，但並非同一人(變數是儲存位置，他們的儲存位置不同)

要如何用到主程式中的變數呢？我們可以透過全域變數，如果副程式中想用到的主程式中的變數，可以用 global 的尚將它宣告成全域變數。

```
def f(x):
    global y # y is global variable
    y = x - 3
    if y<0: return 2*x+1
    return x-1

y=-1
```

```
print(f(3),y)
```

此外，如果副程式中並未定義 y(沒給 y 的值)就拿來使用，這時候就會到主程式中找，好比在某個學校內找不到吳邦一，就擴大範圍到校外找。

不要使用含混不明的用法，那是給自己找麻煩。使用到全域變數就清楚宣告。全域變數在大型軟體中是不好的方式需要必盡量避免，但是在考試與比賽這種解題程式中，是一個很方便的方式，但不要混淆否則就是給自己找麻煩。

遞迴

函數(副程式)有個特殊的用法，就是遞迴，某個函數自己(直接或間接)呼叫自己稱為遞迴呼叫，數學上的遞迴是指一個函數用自己定義自己，兩者其實是同一回事。遞迴不會無限遞迴，所以遞迴一定會有終止條件。看以下有名的例子：

f(0) = f(1)= 1; f(n) = f(n-1)+f(n-2) for n>1.

寫成程式：

```
def f(n):
    if n<2: return 1
    return f(n-1)+f(n-2)
#main program
print(f(10))
```

這個程式可以跑，但當 n 太大(例如 40)時，會跑很久。遞迴是很重要的程式結構，也是很多重要演算法的基礎，有很多可以進一步學習的課題，但這些超過目前講義的講授範圍。這裡只做個基本介紹，在 APCS 四五級或以上的程度時，再來加以探討。

15. APCS 三級分範例題

Zerojudge 上 APCS 曾經出過的第二題考古題有以下題目。

- b266. 第 2 題 矩陣轉換
- c291. APCS 2017-0304-2 小群體
- c295. APCS-2016-1029-2 最大和
- c462. apcs 交錯字串 (Alternating Strings)
- e287. 機器人的路徑 -- APCS
- f313. 2. 人口遷移-- 2020 年 10 月 APCS
- f580. 2. 骰子 -- 2020 年 7 月 APCS
- f606. 2. 流量 -- 2021 年 1 月 APCS
- g276. 2. 魔王迷宮 -- 2021 年 9 月 APCS
- g596. 2. 動線安排 -- 2021 年 11 月 APCS

以下我們逐題來做示範解題。

b266.矩陣轉換(APCS201603_2)

本題是有關矩陣(二維陣列)的操作。題目定義了兩種操作：翻轉與旋轉，其中翻轉是上下的鏡射，而旋轉是順時針旋轉。題目的要求是給了一連串 M 個操作以及操作後的矩陣，要計算出原來的矩陣為何。這裡要小心別看錯，是給了結果求原來的矩陣，而不是給了初始的矩陣求操作後的結果。

這一題是個操作題，給了一連串的操作與操作後的矩陣，我們可以逆向的逐步去還原出原來的矩陣。對於翻轉，它的逆向操作也是相同的翻轉；對於順時針旋轉，它的逆向操作就是逆時針旋轉。對於列數與行數，翻轉不會改變列數與行數，而旋轉後的列數與行數會交換。

首先，我們要將輸入讀進來，因為旋轉會改變行數與列數，所以初始化的時候我們讓矩陣的行列數夠大，這題不會超過 10，所以我們用一個 10*10 的矩陣來做。我們先把輸入以及後面的流程寫好。

```
r, c, m = map(int, input().split())
mat = [ [0]*10 for x in range(12)]
for i in range(r):
    t = [int(x) for x in input().split()]
    for j in range(c):
        mat[i][j] = t[j]
op = [int(x) for x in input().split()]
# 對於 op 中的每個運算由後往前復原矩陣，並更改 r,c
```

```
#輸出 r,c 與矩陣

```

在細節方面，我們要設計出矩陣的翻轉與逆時針旋轉。先說明矩陣的上下翻轉。

矩陣的上下翻轉相當於將第一列與最後一列對調、第二列與倒數第二列對調，…。因為每一列都是 mat 的一個元素，我們可以藉由交換(swap)來做。

```
        for j in range(r//2):
            mat[j], mat[r-1-j] = mat[r-1-j], mat[j]
```

接下來說明逆時針旋轉的方法。基本上我們先將旋轉後的結果放在一個暫存矩陣中，然後在抄錄回原矩陣。位置的對應方式可以這樣思考，旋轉後的每一個元素如果是 tem[j][k]，那麼它原來是在倒數第 j 個 column(也就是第 c-1-j 的 column)，而原來第 k 列的會跑到第 k 行，也就是

tem[j][k] = mat[k][c-1-j]

得到這個轉換式，程式就很容易寫了。

```
        for j in range(c):
            for k in range(r):
                tem[j][k] = mat[k][c-1-j]
        r, c = c, r
        for j in range(r):
            for k in range(c):
                mat[j][k] = tem[j][k]
```

好了，我們要把細節都放進原來的程式，還有一件事，我們要如何以相反的順序來取得 op[]中的元素(運算)呢？有兩個方式可做，一個是用

for i in range(m-1,-1,-1): #op[i]是要找的運算

或者將 op 倒過來，也就是 op[::-1]。以下是完整的範例程式，這裡採用的是 reverse 的方法。

```
r, c, m = map(int, input().split())
mat = [ [0]*10 for x in range(12)] # 注意要夠旋轉後的大小
tem = [ [0]*10 for x in range(12)] # 暫存用
for i in range(r):
    t = [int(x) for x in input().split()]
    for j in range(c):
        mat[i][j] = t[j]
op = [int(x) for x in input().split()] # 操作
for opi in op[::-1]: # reverse order
    if opi == 1: #上下翻轉
        for j in range(r//2): # swap rows
            mat[j], mat[r-1-j] = mat[r-1-j], mat[j]
```

```
    else: # 逆時針旋轉
        for j in range(c):
            for k in range(r):
                tem[j][k] = mat[k][c-1-j]
        r, c = c, r  # 記得交換
        for j in range(r): # 抄回來
            for k in range(c):
                mat[j][k] = tem[j][k]
    # end if (op)
# end i-for
print(r,c)
for i in range(r):
    print(*mat[i][:c]) # 小心只能輸出 [ 0 : c ]，矩陣的範圍
```

(請注意本題在 ZJ 有兩題，另外一題是多測資的版本)

c291.小群體(APCS201703_2)

這個題目的敘述看起來不是太容易了解，關鍵點在題目所給的提示：「如果你從任何一人 x 開始，追蹤他的好友，好友的好友，….，這樣一直下去，一定會形成一個圈回到 x，這就是一個小群體。如果我們追蹤的過程中把追蹤過的加以標記，很容易知道哪些人已經追蹤過，因此，當一個小群體找到之後，我們再從任何一個還未追蹤過的開始繼續找下一個小群體，直到所有的人都追蹤完畢。」所以本題其實是希望你模擬提示所給的方法與流程。

首先我們看如何根據提示所說的來追蹤標記一個小群體。宣告一個陣列 visit[N]來標記一個人是否被拜訪過：visit[i]=False 表示 i 尚未被拜訪；而 visit[i]=True 表示 i 已經被拜訪過。初始的時候將所有都標記成未拜訪狀態。我們可以使用一個 while 迴圈來從 i 開始走訪並標記一連串的好友連結的好友，while 迴圈的終止條件設成走到開始的 i 為止，也可以設成走到一個曾經被拜訪的為止，在這一題來說是相同的，因為題目說不會有兩個人的好友是同一個人。以下是拜訪的流程。

```
# 假設 friend[i]是 i 的好友
# 從 i 開始拜訪並標記一連串的好友連結，直到遇到 x 為止
    visit[i] = True
    p = friend[i]
    while p != i :
        visit[p] = True
        p = friend[p]
    # end while
```

為了要找出小群體的個數，每次找到一個未拜訪的人就是一個新的小群體，以上述方式走訪標記後要繼續找新的小群體，所以我們可以將流程設計如下：我們由編號 0 開始，一一往下找，碰到未拜訪過的就沿著好友連結找出拜訪一個小群體，然後在從下一個編號繼續，這樣找過的人都不會重複拜訪。

```
n = int(input())
friend = [int(x) for x in input().split()]
visit = [False] * n
n_group = 0
for i in range(n):
    if visit[i] : continue
    n_group += 1
    visit[i] = True
    p = friend[i]
    while p != i :
        visit[p] = True
        p = friend[p]
    # end of while
# end of for
print(n_group)
```

c295.最大和(APCS201610_2)

這個題目分成兩個部份，第一個部分要在 N 群的每一群挑選一個數字，使得總和 S 最大，很明顯地，我們應該在每一群中挑選最大值。第二個部分要判斷每一群挑出的數字是否能整除那個總和 S。所以，整個題目是測驗：迴圈的運用、挑選最大值以及整除判斷的技巧。

根據題目的意思，每一群挑出最大值之後，其他的數字就沒有用了，所以，我們在輸入每一群的數字時，計算該群的最大值，將它儲存起來，為了記錄這些最大值，我們可以開一個 list(ar)，將這些最大值逐步加入 ar 中。當所有群的最大值都記錄後，我們可以計算這些最大值的總和 s，同時我們再進行第二個部分。在第二個部分，題目要求輸出可以整除 s 的那些最大值，也就是在 ar 中的成員那些可以整除 s，此外，最後要判斷是否一個都沒有整除。我們將可以將滿足條件的建構成一個 List(out)，根據 out 的成員個數就可以決定，輸出 out 的內容或者輸出-1。

```
n,m = map(int, input().split())
ar = [0 for i in range(n)]
for i in range(n):
    mylist = [int(x) for x in input().split()]
    ar[i] = max(mylist)
s = sum(ar)
print(s)
out = [x for x in ar if s%x == 0] # ar 中可以整除的依序放入
```

```
if len(out) == 0: # out 是空的
    print(-1)
else:
    print(*out)
#endif
```

c462.交錯字串(APCS201710_2)

輸入正整數 k 以及一個字串，要找出最長的連續一段是由 k 個小寫字母組成的片段與 k 個大寫字母組成的片段交錯組成，例如， k = 2，ABccERfw 就是一個 k-交錯字串。題目中給了兩種解法的提示，一種是從前往後掃描，一面掃描同時一面紀錄答案；另外一種是先把字串的連續大寫與連續小寫片段長度找出來，例如字串是 ABfADFccTRggg，先找出片段長度依序是(2, 1, 3, 2, 2, 3)，然後再找出最長有 4 個長度為 2 的連續片段，因為(3, 2, 2, 3)前與後長度 3 的片段可以只取其中兩個。

首先，字元在程式中是 ASCII 編碼形式存在的，判斷一個字元是大寫字母或寫小字母可以根據對 ASCII 編碼的基本知識也可以使用字元函數。在 ASCII 編碼中，大寫的英文字母從 A 到 Z 是連續的，小寫字母也是從 a 到 z 連續的，有這樣的知識就可以用「c>='a' and c<= 'z'」當作 c 是小寫字母的判斷，相同的也可以用「c>='A' and c<='Z'」來判定 c 是大寫字母。本題中只有英文字母，因為大寫的編碼比小寫的小，所以也可以用「c 是否小於 a 字元」來分別大寫與小寫。

以下先來看先說明三段式的解法。

- 先造一個 list，檢查每一個字母，如果是大寫就給 0，小寫則給 1。例如字串是 ABfADFccTRggg，先造出[0,0,1,0,0,0,1,1,0,0,1,1,1]
- 算出連續大寫與連續小寫片段的長度。seg 存的是目前每一段的長度，my_length 放著目前片段的長度，我們從 uorl[1]開始逐個檢查，如果與前一個一樣，表示在同一片段中，my_length 加一，否則是另一個片段的開始，我們將前一個片段長度放入 seg 中。留意在迴圈結尾必須將最後一個長度存入。上例中可建出[2, 1, 3, 2, 2, 3]
- 最後一部我們在 seg[]中要找連續的 k，從前往後掃描，le 找到第一個 k，然後 ri 從 le+1 開始滑到不是 k 為止，[le, ri-1]就是都是 k。要注意前後如果>k 則要列入計算。

```
# three pass; find the length of each segment first
k = int(input())
a = input()
uorl = [] # 01 list
for c in a:
    if c>='A' and c<='Z': uorl.append(0)
    elif c>='a' and c<='z': uorl.append(1)
# input string contains
seg = [] # length of each segment
my_length = 1
for i in range(1,len(uorl)):
    if uorl[i]==uorl[i-1]:
        my_length += 1
    else:
        seg += [my_length]
        my_length = 1
seg += [my_length]
#print(seg); print(uorl)
longest = 0
m = len(seg)
# find longest k
le = 0
while le<m:
    while le<m and seg[le]<k: #find first >=k
        le += 1
    if le>=m: break
    ri = le+1
    while ri<m and seg[ri]==k: # consecutive k
        ri += 1
    t = (ri-le)*k
    if ri<m and seg[ri]>k:
        t += k
    if t>longest: longest = t
    le = ri
#end while
print(longest)
```

最後說明只掃描一遍的一段式解法，以下是範例程式，這裡我們定義了 isupper() 的副程式。這個方法對初學者比較困難，重點是我們掃描字串的過程中要記錄某些狀態：目前是大寫或小寫 uorl、目前片段的長度 current、目前所在位置交錯字串的總長度 seg、以及目前為止以發現的最長交錯字串長度 ans。

```
01 def isupper(c):
02     if c>='A' and c<='Z':
03         return 1
04     return 0
05 # start main
06 k = int(input())
07 a = input()
08 uorl = -1 # 1 for upper, 0 for lower
09 current = 0
```

```
10 seg = 0
11 ans = 0
12 for c in a:
13     state = isupper(c)
14     if state == uorl: # upper or lower
15         current += 1 # length of current segment
16     else: # change status
17         uorl = state
18         if current < k:
19             seg = 0
20         current = 1
21     if current == k: #find a segment
22         seg += k
23     if current > k: # too long, prefix of a new one
24         seg = k
25         current = k
26     ans = max(ans, seg)
27 print(ans)
```

e287.機器人的路徑(APCS201906_2)

這是一個模擬操作的題目，輸入一個方格棋盤，每一個格子有一個不同的數字。本題的第一件事情是要找到起點，起點是數字最小的格子。從起點開始，我們要模擬機器人的行走，每次會挑選上下左右四個格子中數字最小的格子，然後移動到那個格子，移動時有個限制：走過的格子不可以走，也不能走到範圍外。本題就是模擬這個動作直到不能移動為止。

這一題可以分成兩個部分：(1)找出一個二維陣列中的最小值，以該位置為起點；(2)模擬機器人的行走，對於某一個位置，找到上下左右四個鄰居中沒走過的最小值，而且要能辨識是否已經無路可走。

第一個部分很簡單，處理二維陣列(矩陣)，我們可以搭配一個雙迴圈，一方面讀取矩陣內容，同時檢查最小值。為了要模擬機器人的行走位置，我們以一對變數(si,sj)代表機器人所在位置在第 si 列 sj 行，這裡我們採取矩陣行列的定位方式，使用平面座標的定位方式也可以。

在第二個部分，我們每次要檢查上下左右四個鄰居，並且判斷是否出界以及是否曾經走過。判斷出界只要比較位置就可以，要如何判斷是否走過呢？常用的方法有兩種，一種是另外宣告一個矩陣 visit，將走過的位置設定成 True。另外一種常用的方法是將走過的位置改成一個不可能的值，在本題中，因為我們每次都要找最小值，所以可以把走過的位置改成比最大可能值還大的數字。以下的範例程式中採用第二種(走過的位置設成無窮大)。

```
def next(i,j):
    global mat, m, n
    mm = mat[i][j]
    ni=i; nj=j
    if i>0 and mat[i-1][j]<mm:
        ni=i-1; nj=j; mm=mat[i-1][j]
    if i<m-1 and mat[i+1][j]<mm:
        ni=i+1; nj=j; mm=mat[i+1][j]
    if j>0 and mat[i][j-1]<mm:
        ni=i; nj=j-1; mm=mat[i][j-1]
    if j<n-1 and mat[i][j+1]<mm:
        ni=i; nj=j+1; mm=mat[i][j+1]
    return ni,nj

m, n = map(int, input().split())
oo = 1000001
mat = [[] for i in range(m)]
for i in range(m):
    mat[i] = [int(x) for x in input().split()]
mm = oo
for i in range(m):
    for j in range(n):
        if mat[i][j] < mm:
            si=i; sj=j; mm=mat[i][j]
total = 0
while True:
    total += mat[si][sj]
    mat[si][sj] = oo
    si,sj = next(si,sj)
    if mat[si][sj] == oo:
        break
print(total)
```

在前面的講義中我們講過兩種可簡化二維陣列檢查的方法：一是外圍圍一圈不可能的值來簡化檢查出界；二是用四個方向的向量。以下是這樣寫法的程式。

```
m, n = map(int, input().split())
m += 2; n += 2
oo = 1000001
dr = [-1,0,1,0]; dc = [0,1,0,-1] # 上右下左
mat = [[] for i in range(m)]
mat[0] = [oo for j in range(n)]
mat[m-1] = [oo for j in range(n)]
for i in range(1,m-1):
    mat[i] = [oo]+[int(x) for x in input().split()]+[oo]
# find minimum
mm = oo
for i in range(m):
    for j in range(n):
        if mat[i][j] < mm:
            si=i; sj=j; mm=mat[i][j]
total = 0
```

```
while True:
    total += mat[si][sj]
    mat[si][sj] = oo # update to visited state
    mm = oo; ni = si; nj = sj
    for d in range(4):
        i = si+dr[d]; j = sj+dc[d]
        if mat[i][j] < mm:
            mm = mat[i][j]; ni = i; nj = j
    si,sj = ni,nj
    if mat[si][sj] == oo: # no way to go
        break
print(total)
```

f313.人口遷移(APCS202010_2)

在一個矩陣中，有些格子是城市，有些不是。每個城市有若干人口，每一天，人口的 1/k 會遷移到每一個鄰接的城市。本題要模擬每天人口的變動狀況。題目的重點是要能夠決定每一個城市有多少鄰接的城市，要注意非城市以及邊界的情形，然後只要依照題目的要求計算就可以了，並不需要特殊的方法。

這一題的主要問題在於如何記錄前一天與後一天的狀態，以及要能決定鄰接的位置(上下左右四方位)是否是城市。對於每一個城市，要扣除移出人口數，再加上移入人口數。要注意我們必須每次從今天的人口計算明天的人口，所以算出來的資料必須先放置在另外一個矩陣中，以免計算過程中資料混用而導致錯誤。每一天的模擬完畢後，再把資料放回原來的矩陣儲存位置，這用 list 交換的方式做要比拷貝的方式更方便有效率。

```
r,c,k,m = map(int, input().split())
p = [[] for i in range(r)]
for i in range(r):
    p[i] = [int(x) for x in input().split()]
temp = [[0]*c for i in range(r)]  # working space for next day
for day in range(m): # 從 p 算到 temp
    for i in range(r):
        for j in range(c):
            temp[i][j] = p[i][j]
            if temp[i][j]<0: continue # non-city
            q = p[i][j]//k # move out
            #check 4 neighbors
            if i>0 and p[i-1][j]>=0:
                temp[i][j] += p[i-1][j]//k - q
            if i<r-1 and p[i+1][j]>=0:
                temp[i][j] += p[i+1][j]//k - q
            if j>0 and p[i][j-1]>=0:
                temp[i][j] += p[i][j-1]//k - q
            if j<c-1 and p[i][j+1]>=0:
```

```
                temp[i][j] += p[i][j+1]//k - q
    temp, p = p, temp # swap, copy temp back to p
# end for
#output min and max
print(min(p[i][j] for i in range(r)
          for j in range(c) if p[i][j]>=0))
print(max(p[i][j] for i in range(r)
          for j in range(c)))
```

使用外圍圍一圈的方法以及方向向量的技巧，也可以寫成這樣：

```
r,c,k,m = map(int, input().split())
r += 2; c += 2
p = [[] for i in range(r)]
p[0] = [-1 for j in range(c)]
p[r-1] = [-1 for j in range(c)]
for i in range(1,r-1):
    p[i] = [-1]+[int(x) for x in input().split()]+[-1]
temp = [[-1]*c for i in range(r)]  # working space for next day
dr = [-1,0,1,0]; dc = [0,1,0,-1] # 上右下左
for day in range(m):
    for i in range(r):
        for j in range(c):
            temp[i][j] = p[i][j]
            if temp[i][j]<0: continue # non-city
            q = p[i][j]//k # move out
            #check 4 neighbors
            for d in range(4):
                ni = i+dr[d]; nj = j+dc[d]
                if p[ni][nj]>=0:
                    temp[i][j] += p[ni][nj]//k - q
    temp, p = p, temp # swap, copy temp back to p
# end for
#output min and max
print(min(p[i][j] for i in range(r)
          for j in range(c) if p[i][j]>=0))
print(max(p[i][j] for i in range(r)
          for j in range(c)))
```


f580.骰子(APCS202007_2)

題目中定義了骰子的兩種滾動方式，對若干個骰子，給予某些滾動的操作以及交換的操作，要問最後骰子的點數。這是操作模擬的題目，依照要求模擬骰子的狀態就可以了。這一題的主要問題在於如何記錄骰子的狀態，一個骰子的狀態事實上可以由兩個面(如上面與前面)所決定，也就是有 24 種狀態，但是千萬不要想去用窮舉 24 種狀態的方式

來做，那程式可就非常的冗長了。其實我們根本不需要知道骰子有多少種狀態，我們就老老實實的以一個陣列紀錄上下左右前後六個面的值就可以了。

所以這一題可以用二維陣列來做，每一個元素是一個 List 紀錄一顆骰子的六個面。在滾動時，會改變骰子的其中四個面，我們就一個一個去設定就可以，因為變動的四個面其實是一個循環，與交換其實是類似的(交換是兩個的循環)。

向右旋轉：左變上、下變左、右變下、上變右，也就是
`top,left,bottom,right = left,bottom,right,top`

向前旋轉：前變上、下變前、後變下、上變後，也就是
`top,front,bottom,back = front,bottom,back,top`

寫的時候把上下左右前後以 0~5 定義好就可以了，另外因為題目中定義的骰子編號從 1 開始，為了與題目一致(避免寫錯)，我們在前面加個空 list。

```
n, m = map(int, input().split())
d = [[]]
for i in range(n):
    d.append([1, 2, 6, 5, 4, 3])  #top, right, bot, left, front, back
for i in range(m):
    a, b = map(int, input().split())
    if b == -2:
        d[a][0],d[a][3],d[a][2],d[a][1] = d[a][3],d[a][2],d[a][1],d[a][0]
    elif b == -1:
        d[a][0],d[a][5],d[a][2],d[a][4] = d[a][5],d[a][2],d[a][4],d[a][0]
    else:
        d[a], d[b] = d[b], d[a]
    #end if
#end loop
ans = [d[i][0] for i in range(1,n+1)]
print(*ans)
```

f606.流量(APCS202101_2)

有 n 個伺服器與 m 個城市，每個伺服器到每個城市有若干資料流量。所謂一個方案是指定每一個伺服器放置的城市，對於每一個方案，要計算出城市到城市的資料流量，再根據題目的定義計算費用。這一題只要依照題目的要求計算就可以了，並不需要特殊的方法，要注意的是必須先算出城市到城市的總流量，因為費用的計算是根據總流量。

這一題只要是計算各方案的費用，挑選最少費用只是在各方案中選取最小值。計算每一個方案的費用可以分成兩大步驟，第一步是要先算出城市到城市的流量；第二步則根據定義與流量計算費用。

要先計算城市到城市的流量，我們使用一個二維陣列 qq[][]，以 qq[i][j]來紀錄城市 i 到城市 j 的流量。

```
n,m,k = map(int, input().split())
q = [[int(x) for x in input().split()] for i in range(n)]
mcost = 100000000 # 設個不可能的大
for case in range(k):
    c = [int(x) for x in input().split()] # city of server
    # quantity from city i to city j
    qq = [[0 for i in range(m)] for j in range(m)]
    for i in range(n): # for each server
        for j in range(m): # server i to city j
            qq[c[i]][j] += q[i][j]
    #end for i,j
    cost = 0
    for i in range(m):
        for j in range(m):
            if i==j:
                cost += qq[i][j]
            elif qq[i][j]<=1000:
                cost += qq[i][j]*3
            else: cost += 1000 + qq[i][j]*2
    mcost = min(mcost,cost)
# end for case
print(mcost)
```

g276.魔王迷宮(APCS202109_2)

這是一個模擬的題目，我們要寫一個程式來模擬題目所定義的動作以及計算它的結果。動作是一回合一回合進行的，所以我們要考慮的是如何記錄狀態，本題的狀態包括：那些位置有炸彈，每個魔王的所在位置。因為同一個格子的炸彈引爆時是同時引爆，所以只需要知道有無炸彈即可而不需紀錄該格子的炸彈數量。紀錄狀態有多種紀錄的方式，不同的紀錄方式會影響到程式的寫法，最好的記錄方式是方便我們執行要做的動作。

根據題目中的說明，模擬動作的結束條件是魔王全部消失時才結束，因為場地範圍有限，很顯然一個魔王會在若干回合後就消失(被炸死或移出範圍外)，程式的主要流程可以直接根據題目的說明設計如下：

```
輸入每一個魔王初始位置與移動向量;
while 還有魔王存在 do (每次一回合)
```

檢查那些格子的炸彈會引爆，並記錄哪些魔王會炸死;

每一個存在的魔王在所在位置放炸彈;

每一個存在的魔王移動到下一個位置並檢查是否出界;

endwhile

計算哪些格子有炸彈並輸出格子數;

接下來我們思考資料的儲存方式。對於魔王來說，我們要記錄魔王的位置與移動向量，此外我們還必須知道魔王是否還存活。對於炸彈來說，我們只需要知道每一個格子有無炸彈，而同一個格子裡是一枚或多枚炸彈並無差異，所有的炸彈也都是一樣的性質，不像魔王有不同的移動向量。因此，我們只要紀錄記錄每一個格子是否有炸彈即可。資料結構(儲存方式)可以規劃如下：

存活的魔王放在一個list中，每個魔王的資料包括：

魔王的目前位置：紀錄列與行。

魔王的移動向量：紀錄列與行移動量。

炸彈的資料：

陣列 bomb[i][j]紀錄(i,j)位置是否有炸彈。

除此之外，在每一回合計算過程中必須要注意，碰到炸彈引爆的情況時，需要處理完所有在一同一格的魔王，如果只炸死一個魔王就清除炸彈，則可能導致錯誤的答案。在以下的範例程式中，每一回合分成三段處理：

- 對每個魔王檢查所處位置是否有炸彈，如果有就將炸彈設定為引爆狀態(-1)，否則將魔王存入 temp。處理完畢後交換 temp 與 monster，這樣炸死的魔王就移除了。
- 檢查哪些炸彈已經爆炸(-1)，將其清除。
- 每個存活魔王將所在位置設置炸彈，並移動到下一個位置，如果沒出界才放入 temp。在檢查完畢之後交換 temp 與 monster，這樣以出界的就移除了。

```
# monster in list
m,n,k = map(int,input().split())
bomb = [[0]*n for i in range(m)]
monster = [] # (r,c,dr,dc)
for i in range(k):
    monster.append([int(x) for x in input().split()])
# input complete
while len(monster)>0:
    # check explosion, alive monster save in temp
    temp = []
    for p in monster:
        if bomb[p[0]][p[1]]:
            bomb[p[0]][p[1]] = -1 # explode
        else:
            temp.append(p)
    #end for p
    monster, temp = temp, monster
    # clear exploded bomb
    for i in range(m):
        for j in range(n):
            if bomb[i][j] == -1:
                bomb[i][j] = 0
    # move monster
    temp=[] # keep inside monster
    for p in monster:
        bomb[p[0]][p[1]] = 1
        p[0] += p[2]
        p[1] += p[3]
        if 0<= p[0] <m and 0<= p[1]<n:
            temp.append(p)
    # end for
    monster, temp = temp, monster
# end while
ans = sum(bomb[i][j] for i in range(m) for j in range(n))
print(ans)
```

這一題可以用 set 來寫會簡單一些。Python 可以用 set 來處理集合，查詢與集合差集都很方便也有效率，在以下程式中，我們把有炸彈的位置丟進一個集合 bomb，對每個魔王，檢查所在位置是否在 bomb 集合中，若是，則將炸彈放入引爆的集合，若否，則將魔王放入下一回合的清單中(存活)。第 17 行是做兩集合的差集，將引爆的炸彈移除。其他部份與前述程式類似。

```
00 # using set() for bomb. list() for devil
01 m,n,k = map(int,input().split())
02 devil = []
03 for i in range(k):
04     devil.append([int(x) for x in input().split()])
05 # [r, c, s, t], (r,c): position, (s,t):moving vector
06 bomb = set() # position of bomb
07 while len(devil)>0:
```

```
    # check each devil
    explode =set() # explore bomb
    newdevil = []
    for dev in devil:
        if tuple([dev[0],dev[1]]) in bomb:
            explode.add(tuple([dev[0],dev[1]]))
        else:
            newdevil.append(dev)
    # clear the exploding bomb
    bomb = bomb - explode
    devil = []
    # each devil move to next position
    for dev in newdevil:
        bomb.add(tuple([dev[0],dev[1]]))
        dev[0] += dev[2]
        dev[1] += dev[3]
        if 0<=dev[0]<m and 0<=dev[1]<n:
            devil.append(dev)
#end while
print(len(bomb))
```

g596.動線安排(APCS202111_2)

這是一個模擬的題目，我們要寫一個程式來模擬題目所定義的動作以及計算它的結果。題目的場景是個陣列，動作則包含加入柱子以及移除柱子，動作是一個一個進行的，所以我們要考慮的是如何記錄狀態。本題的狀態包括：每一個格子是柱子或是線，而線有水平與垂直的線，也可能一個格子同時有垂直與水平線通過。

根據題目中的說明，模擬的動作有加入與移除柱子兩種，但每次加入或移除都必須往上下左右四個方向做檢查，所以程式碼有點冗長繁瑣，但並不特別困難。我們先來看第一子題只有一維陣列時的做法，首先要考慮的是如何記錄狀態，因為格子內有柱子、線與空白 3 種可能，我們分別以 0 代表空格、1 代表線、以及 2 代表柱子。

將加入與移除的函數分別考量來設計如下：

```
#subtask 1, 1d-array
# 1 =line, 2= pillar 0=space
def rem(c): # remove
    global n
    # horizontal
    a[c] = 0
    i = c-1
    while i>=0 and a[i]==1: # clear while a[i]=1 (line)
        a[i] = 0
        i -= 1
    i = c+1 # clear right side
```

```
    while i<n and a[i]==1:
        a[i] = 0
        i += 1

def add(c):
    global n
    a[c] = 2 # 柱子
    c2 = c-1 # 往左找柱子
    while c2>=0 and a[c2]!=2: c2 -= 1
    if c2 >= 0: # 找到柱子
        for i in range(c-1,c2,-1): a[i] = 1 # 拉線
    c2 = c+1 # 往右
    while c2<n and a[c2]!=2: c2 += 1
    if c2<n:
        for i in range(c+1,c2,1): a[i] = 1

# start amain
total = 0 # 每回合非空格數量
imax = 0 # 紀錄最大值
m,n,h = map(int, input().split())
a = [0]*n
for it in range(h):
    r,c,indel = map(int, input().split())
    if indel==0:
        add(c)
    else:
        rem(c)
    # count total
    total = n - a.count(0) # 扣除空格
    if total > imax: imax=total
# end it
print(imax)
print(total)
```

接著考慮 100 分的完全解，也就是在二維陣列上做。其實觀念是一樣的，只是每次要檢查四個方向，此外，格子內多了幾種狀態，除了空白與柱子之外，有可能是水平線、垂直線、或水平垂直皆有。程式碼因此變得有點長。

```
# not using bit operation
# 1 =horizontal, 2= vertical, 1+2=3=cross, 0=space 4=pillar
def rem(r, c):
    global m,n # m row, n column
    # horizontal
    a[r][c] = 0
    i = c-1
    while i>=0 and (a[r][i]== 1 or a[r][i]==3): # hor or cross
        a[r][i] -= 1
        i -= 1
    i = c+1
```

```
    while i<n and (a[r][i]== 1 or a[r][i]==3):
        a[r][i] -= 1
        i += 1
    # delete vertical
    i = r-1
    while i>=0 and (a[i][c]== 2 or a[i][c]==3): # vert or cross
        a[i][c] -= 2
        i -= 1
    i = r+1
    while i<m and (a[i][c]== 2 or a[i][c]==3):
        a[i][c] -= 2
        i += 1

def add(r, c):
    global m,n
    #vertical
    if a[r][c]!=2 and a[r][c]!=3:
      r2 = r-1
      while r2>=0 and a[r2][c]!=4: r2 -= 1
      if r2 >= 0: # pillar found
        for i in range(r-1,r2,-1):
            a[i][c] += 2 # add vertical
      r2 = r+1
      while r2<m and a[r2][c]!=4: r2+=1
      if r2<m:
        for i in range(r+1,r2,1):
            a[i][c] += 2
    # insert horizontal
    if a[r][c]!=1 and a[r][c]!=3:
      c2 = c-1
      while c2>=0 and a[r][c2]!=4: c2-=1
      if c2>=0:
        for i in range(c-1,c2,-1):
            a[r][i] += 1 # add horizontal
      c2 = c+1
      while c2<n and a[r][c2]!=4: c2+=1
      if c2<n:
        for i in range(c+1,c2,1):
            a[r][i] += 1
    a[r][c] = 4

# start main
imax = 0
m,n,h = map(int, input().split())
a = [[0]*n for j in range(m)]
for it in range(h):
    r,c,indel =  map(int, input().split())
    if indel == 0: add(r,c)
    else: rem(r,c)
    # count total
    total = 0
    for i in range(m):
        total += n - a[i].count(0)
    if total > imax: imax=total
# end it
```

```
print(imax)
print(total)
```

16. 排序與搜尋

排序與搜尋是 APCS 四五級分程度的重要課題，但這裡我們說明一下 python 簡單的排序與搜尋方法，在某些二三級分的題目中，使用排序與搜尋可以有更簡潔的做法。這一部分已經屬於演算法的層次，對剛學程式的人有點難。

排序

排序是將一群資料，這群資料必須是可以比較大小的，通常它們是同一資料型態，例如說都是整數或都是字串。一群整數的排序是最簡單的

```
a = [5, 1, 3, 6, 3, 4, 8, 2, 1]
b = sorted(a)
a.sort()
```

請注意兩者用法的差異，執行 sorted(a)會將 a 複製出來後排序，a 的內容並沒有改變，如果我們不要破壞原資料的時候用 b=sorted(a)。a.sort()是對 a 這個東西套用 sort()，所以 a 的內容會被改變成排序後的結果。

排序預設是由小排到大，要由大排到小怎麼辦？

```
a.sort(reverse=True)
```

字串也可以排序

```
sa = ['algorithm', 'Data structure', 'alignment', 'algo', 'b2', 'b13', 'ba']
sb = sorted(sa)
```

字串排序的方式稱為字典順序，字元的順序是 ASCII code 的順序，數字<大寫<小寫。

所謂字典順序：從前往後比，前面小就小了，前面平手才比後面。

注意上例中：b13 < b2 < ba

a 是個 list of list，a 中元素每個都是 2(或 3 或更多)欄位的，例如

```
a = [[2,4],[1,3],[2,6], [2,1]]
```

預設排序是字典順序，執行 a.sort()之後，a 的內容是
[[1,3], [2,1], [2,4], [2,6]]

要改變比較的方式，用 lambda 函數，寫法如下例

```
a.sort(key=lambda x: x[1])
```

上述指令的意思是排序用的 key 值是對於每一個元素 x，取出 x[1]當作比較對象。當然可以用各種欄位運算的結果來排序，例如：

```
a.sort(key=lambda x: x[1]+x[0])
```

也可以由大排到小

```
a.sort(key=lambda x: x[1]+x[0], reverse=True)
```

線性搜尋

在沒有順序的資料中搜尋只能一個一個慢慢找

```
a = [5, 1, 3, 4, 1, 8, 2, 3, 3, 7]
x = 6
for i in range(len(a)):
    if a[i] == x: break
print('end with i =',i)
if a[i] == x: print(x, 'found at', i)
else: print(x,'not found')
```

如果只要知道在或不在，可以用 in 來檢查。

```
if x in a: print('yes')
else: print('no')
```

如果確定 x 在 a 中，可以用內建函數 index 來找位置。

```
i =  a.index(3) #可以找到第一個 3 的位置
i = a.index(3,i+1) # i+1 開始找
```

list.index(x)在 x 不存在時會引發錯誤，所以使用上要小心。python 可以用錯誤處理的方式做，但初學者不太容易使用，可以用 in 搭配 index 來做。

```
a = [5, 1, 3, 4, 1, 8, 2, 3, 3, 7]
x = 9
if x in a:
    print('yes, found at',a.index(x))
else: print('no')
```

二分搜尋

排序好的 list 可以做二分搜尋，速度非常快，在 100 萬個資料時只需要 20 次的比較就可以找出來。原理跟猜數字終極密碼的遊戲一樣，每次將搜群區間剖半並捨棄其中一半，這樣只需要 log(n)次比較就可以把區間範圍減少到 1。二分搜可以自己寫。

```
def bsearch(p, x): # find x in p
    le = 1; ri = len(p)-1 # range = [le, ri]
    while le <= ri:
        mid = (le+ri)//2 # middle
        if p[mid]==x: return mid
        elif p[mid]<x: le = mid+1 # [mid+1, ri]
```

```
        else: ri = mid-1 # [le,mid-1]
    # end while
    return -1  # not found

a = [5, 1, 3, 4, 1, 8, 2, 3, 3, 7]
a.sort()
print(a)
x = 9
i = bsearch(a,x)
if i<0 :
    print(x,'not found')
else:
    print(x,'found at',i)

x=3
i = bsearch(a,x)
if i<0 :
    print(x,'not found')
else:
    print(x,'found at',i)
```

二分搜也有內建函數可以呼叫，但必須先 import bisect，內建二分搜的速度要比自己寫快得多。

```
import bisect
a = [1,4,5,6,6,6,7,9,10]
i = bisect.bisect_left(a,6)
print(i)
i = bisect.bisect_left(a,7)
print(i)
```

17. 附錄

17.1. 一些講義中沒講的東西

這份講義以 APCS 三級為短期目標來撰寫，所以有些指令與函數沒有納入，另外有需要提醒的事情，以下做簡略說明。

for 迴圈的注意事項

我們知道 for 迴圈的使用方法是 for x in A: 對於 A 中的每一個 x，每個都要來、一個一個來、依照順序來。如果執行下面的程式會如何呢？

```
for i in range(5):
    print(i)
    i += 3
    print(i)
```

猜對了嗎? 如果你在迴圈內改變 i 的值，並不能控制迴圈執行的次數，事實上每回合進入迴圈時，i 都依照原訂的[0,1,2,3,4]被重新賦值。以下的程式呢？

```
a = [0,1,2,9,4]
for i in range(5):
    if a[i]<5: a.pop(i)
```

從表面上看，這程式對 a 中的每個元素檢查，如果<5 就將它刪除。但它是錯的。因為當執行 a.pop(i)時改變了 a 的內容，所以會造成 runtime error (index out of range)。即使加上 index 檢查也不是正確的結果。

```
a = [0,1,0,1,0]
for i in range(5):
    if i<len(a) and a[i] == 0: a.pop(i)
```

原因是當刪除發生時，後面的元素 index 改變了。下面的迴圈將導致無窮迴圈：

```
a = [1,2,3]
for x in a:
    print(x)
    a.append(x)
```

這裡就是告訴大家注意兩件事

- for i in range(n)的迴圈內改變 i 的值是不能控制迴圈的執行。
- 不要在 for x in A 的迴圈中改變 A，那是件危險的事情。

EOF 結尾的測資

所謂 EOF 是 End Of File 的意思，是檔案結尾的一個符號，但不同平台用的不一樣，windows 是 control-z，而 Linux 下是 control-d。在 ZJ 與網路上都有不少題目測資是 EOF 結尾的格式，例如題目這樣說：

輸入有多行，每行兩個數字，輸出每行兩數的和(直到 EOF)。

不管有沒有說直到 EOF，都意思是直到輸入檔案結束為止。每個程式語言處理 EOF 的方法各異，APCS 從來沒有考過這樣的測資，因為跟她檢測的理念不符吧(考一個很特定的用法，不會就認定不會寫程式，這大概也不適合)。

那 Python 到底怎麼處理 EOF 呢？執行下面的程式：

```
while True:
    try:
        a,b = map(int,input().split())
    #except: break
    except EOFError: break
    print(a+b)
```

執行程式後你可不知道怎麼讓程式正常結束，試著輸入幾行測資後按下 control-d (或 control-z)，程式就會正常結束了。「try: … except:…」指令的意思就是「嘗試做…，如果碰到錯誤就…」。這裡因為只會有 EOF 的錯誤，所以寫不寫 EOFError 都可以。

位元運算

所謂位元運算就是把一個整數以二進位的方式看待，然後對位元(bit)做運算，位元運算在某些場合有她的方便性，但並非不可取代。Python 的有以下位元運算：

```
a = 0b10101; b =0b01001 # 二進位 a=21, b=9
a & b  # 每一個對應的位元做 and 運算，結果是 1
a | b  # 每一個對應的位元做 or 運算，結果是 29 = 0b11101
a ^ b  # 每一個對應的位元做 exclusive or 運算，結果是 28 = 0b11100
a << 3 # 二進制左移 3 位，相當於乘以 8
a >> 2 # 二進制右移 2 位，相當於除以 4
```

在某些題目使用位元運算可以加速程式的執行，例如以(x&1)取代(x%2==1)來判斷奇數；以及(x<<=1)取代(x*=2)，因為位元運算要比乘除法快很多。詳細的運用方式這裡就不多說。

集合(set)

Python 的 set 是個資料結構，類似的還有 dictionry。這裡的 set 與中學數學學到的集合是一樣的，她可以 0 個或多個元素，但不可重複，沒有順序。

```
s0 =set() #空集合，不可用 s0={}
s1 = {3 ,1, 4, 8}
s2 = {x+1 for x in s1 if x<5}  #{2,4,5}
a = [1,3,1,4,2,3,3]; s3=set(a) # {1,2,3,4} 把 list 中的元素放入 s3 集合
s1 - s2 # 差集
s1 & s2 #交集
s1 | s2 # 聯集
s1 ^ s2 # 對稱差 (s1 | s2) - (s1 & s2)
3 in s1 # 元素檢查是否在
s1.add(9) # 加入一個元素，已存在則無變化
s1.remove(9) # 移除一個元素，不存在會錯誤
```

set 的運算是很快速的，同樣是 if x in S，如果 S 是 set，即使 S 很大，這個判斷需要的時間很短，但如果 S 是 list，這個判斷會花比較多的執行時間。

17.2. The Python Tutorial 官網教學文件

Python 官網的初學者教學是個很不錯的教學文件：

https://docs.python.org/3/tutorial/index.html

中文版： https://docs.python.org/zh-tw/3/tutorial/index.html

17.3. [官網教學手冊]list 函數

`list.`**append**(*x*)

Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

`list.`**extend**(*iterable*)

Extend the list by appending all the items from the iterable. Equivalent to `a[len(a):] = iterable`.

`list.`**insert**(*i*, *x*)

Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

`list.`**remove**(*x*)

Remove the first item from the list whose value is equal to *x*. It raises a `ValueError` if there is no such item.

`list.`**pop**([*i*])

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

`list.`**`clear`**`()`

Remove all items from the list. Equivalent to `del a[:]`.

`list.`**`index`**`(`*x*`[,` *start*`[,` *end*`]])`

Return zero-based index in the list of the first item whose value is equal to *x*. Raises a `ValueError` if there is no such item. The optional arguments *start* and *end* are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.

`list.`**`count`**`(`*x*`)`

Return the number of times *x* appears in the list.

`list.`**`sort`**`(*, key=None, reverse=False)`

Sort the items of the list in place (the arguments can be used for sort customization, see `sorted()` for their explanation).

`list.`**`reverse`**`()`

Reverse the elements of the list in place.

`list.`**`copy`**`()`

Return a shallow copy of the list. Equivalent to `a[:]`.

An example that uses most of the list methods:

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits.count('apple')
2
>>> fruits.count('tangerine')
0
>>> fruits.index('banana')
3
>>> fruits.index('banana', 4)  # Find next banana starting a position 4
6
>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']
>>> fruits.append('grape')
>>> fruits
```

```
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']
>>> fruits.pop()
'pear'
```

17.4. 常用指令摘要

- 輸出
  - print(x); print(x,y) 空白間格
  - a=[2,3,4]; print(a); print(*a)
  - print(*a, sep=',')
- 輸入：
  - s = input() 是輸入一行字串
  - a = int(input()) 將輸入的字串轉換為整數，整數才能加減乘除
  - 輸入兩個整數(空白間格) a,b = map(int, input().split())
    - input().split() 的作用
    - map() 的作用
  - 輸入三個整數 a,b,c = map(int, input().split())
  - 一行不定個數的整數 a = [int(x) for x in input().split()]
    - 用 list，一個序列 L1 = [1, 42, 3, 6]; L2 = ['a', 4, 3.14, "Dijkstra"]
    - list 的元素存取 L1[0], L2[3]
- 基本運算
  - a = 5+3-4*6 + b//7; c = d%7; e=1/2 (浮點數)
- 分支指令 if
  - ```
    if a<b: c=a
    else: c=b
    if a<b:
        c=b
        d=b-a
    else:
        c=a
        d=a-b
    ```
    ```

- if score>90:
  grade='A'
  elif score>80:
  grade = 'B'
  else: grade = 'F'

- 迴圈 for, while
  - for i in range(5):
    print(i)
    n = 20
    for i in range(2, n, 3):
    print(i)
  - for x in a: # for each element x in list a
  - for x in s: # for each char c in string s
  - for + if
    total = 0
    for i in range(20):
    if i%3 == 0: total += i*i
  - i = 1; ans=0; n=24
    while i*i < n:
    if n%i == 0:
    ans += i + (n//i)
    i += 1
    if i*i == n: ans += i
    print(ans)
  - x = 48; y=60
    while x>0:
    t = y%x
    y = x; x = t
    print(y)
  - break # 中斷迴圈，跳到迴圈之後
  - continue # 中斷本回合，跳到下次迴圈入口
- list
  - a = [0,1,2,3,4,5]
  - c = a.copy() 或 c = a[:]
  - c.append(7) 或 c += [7]
  - x = c.pop() # default is last element
  - x = c.pop(3)

- a = a[:3] + a[2:4] + a[-3:-1] + a[-2:]
- s[: :-1] # reverse a list or a string
- b = [x for x in a if x%2==1]
- d = [0]*10 # initial array
- e = [[0]*n for i in range(m)] # initial m*n array
- f = [[] for i in range(m)] # input 2d array

```
for i in range(m):
    f[i] = [int(x) for x in input().split()]
```

- 幾個簡單常用函數
  - imax = max(a,b,c); imin = min(a,b,c)
  - qq=[4,1,3,5]; imax=max(qq); imin=min(qq); isum=sum(qq)
  - w = sorted(qq) #w 是 qq 排序後的結果，qq 沒改變
  - qq.sort() # sort qq, qq is changed
  - qq.sort(reverse=True)
  - pp = [[x,10-x] for x in range(7)]
  - pp.sort() # lexicographic order
  - pp.sort(key=lambda x: x[1]) # using x[1] as key