運算思維於程式設計

Computational Thinking

PROFESSOR CHIH-HUNG WU

吳智鴻教授

國立臺中教育大學 數位內容科技研究所

CHWU@MAIL.NTCU.EDU.TW

WEBSITE: CHWU.WEEBLY.COM

6 DEC, 2021

Computational thinking (CT) 運算思維的起源

2006年3月，美國卡內基・梅隆大學計算機科學系主任周以真（Jeannette M. Wing）教授因提出並倡導「計算思維」而享譽計算機科學界。

她在美國計算機權威期刊《Communications of the ACM》雜誌上給出並定義計算思維（Computational Thinking）。周教授認為：計算思維是運用計算機科學的基礎概念進行問題求解、系統設計、以及人類行為理解等涵蓋計算機科學之廣度的一系列思維活動。

在她看來，「計算思維是一種普適思維方法和基本技能，所有人都應該積極學習並使用，而非僅限於計算機科學家。」同時，周以真博士也是卡內基梅隆大學計算思維中心的創始人和負責人。

https://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97%E6%80%9D%E7%BB%B4

Why我們需要學運算思維

好處：

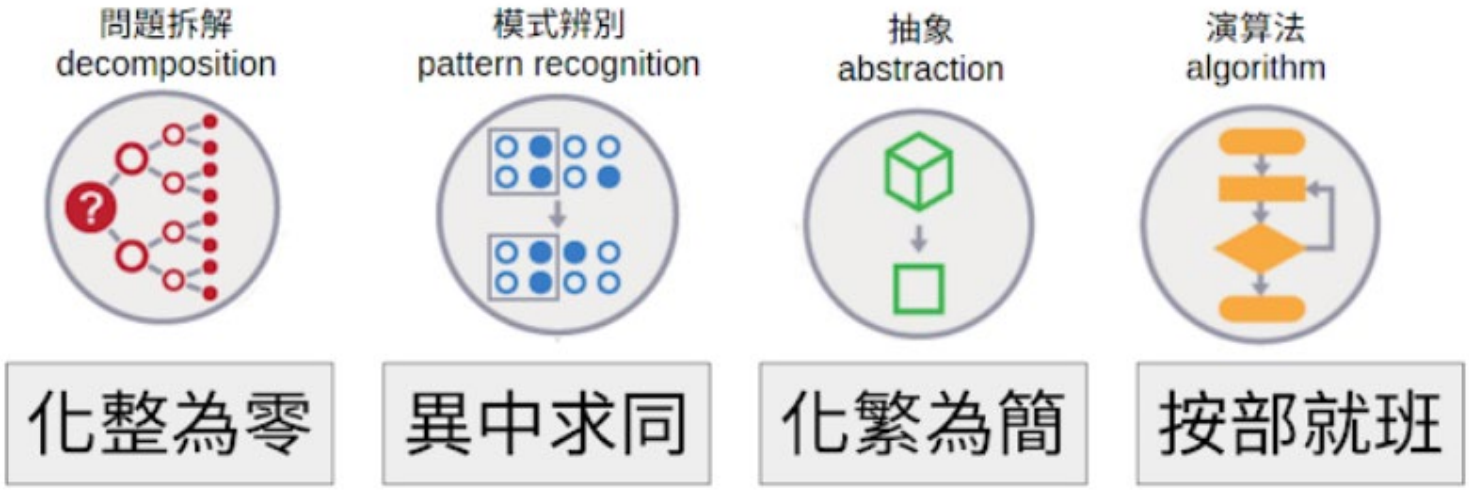
運算思維可以幫助建立邏輯觀念，幫助提昇解決問題能力。

這些能力都是有助於 程式設計的能力提昇。

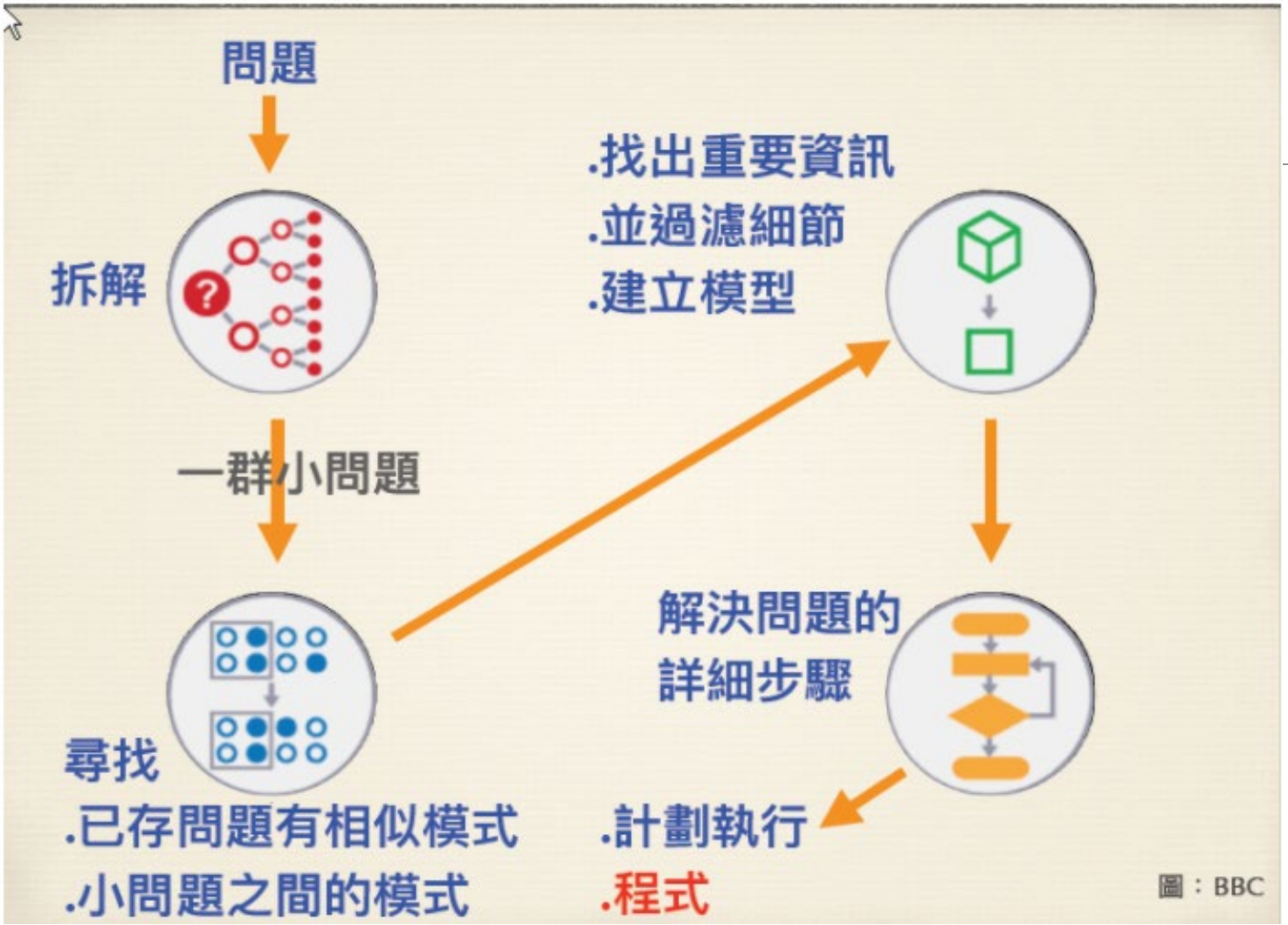
使用時機：

在開始寫程式之前，先透過運算思維的步驟，將規劃與設計整個程式的邏輯與流程。然後再開始寫程式。

運算思維四個步驟



圖源：http://www.bbc.co.uk/education/guides/zxxbgk7/revision

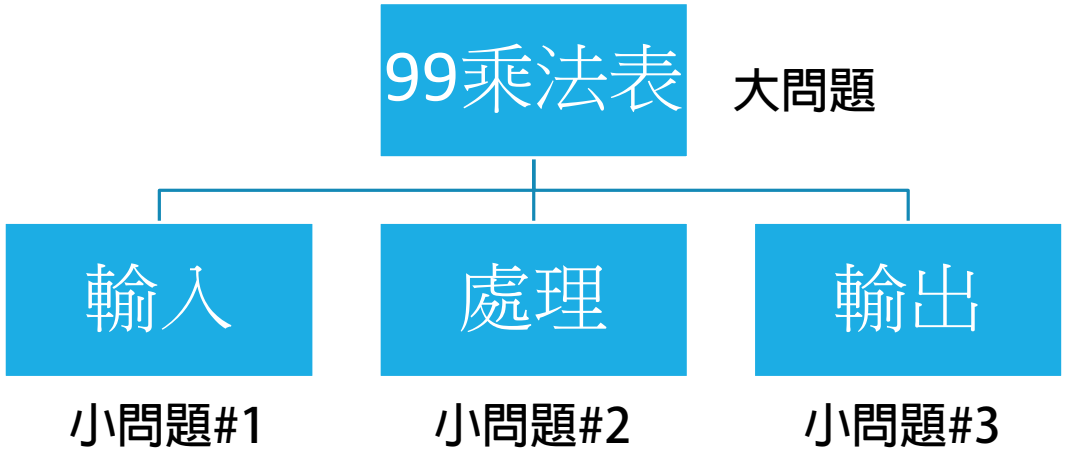

問題
拆解
一群小問題
尋找
.已存問題有相似模式
.小問題之間的模式
.找出重要資訊
.並過濾細節
.建立模型
解決問題的
詳細步驟
.計劃執行
.程式
圖：BBC

問題拆解（Decomposition)

問題拆解是化整為零，將一個大問題拆解成數個小問題。

以程式設計為例，通常至少可以區分成以下小問題。

例如：99乘法表



模式識別（Pattern Recognition)

模式識別就是異中求同。

在龐雜的訊息中，找到相似之處，那個相似之處就是模式(pattern)。

以程式設計而言，就是找出重複被執行的地方（指令）

例如99乘法表


```
#  Example3  99乘法表
for  i  in  range(2,10):
        for  j  in  range(2,10):
                print  ('%d  *  %d  =  %2d  '  %(i,  j  ,  i*j),end="")
        print()
```

```
2 * 2 =  4 2 * 3 =  6 2 * 4 =  8 2 * 5 = 10 2 * 6 = 12 2 * 7 = 14 2 * 8 = 16 2 * 9 = 18
3 * 2 =  6 3 * 3 =  9 3 * 4 = 12 3 * 5 = 15 3 * 6 = 18 3 * 7 = 21 3 * 8 = 24 3 * 9 = 27
4 * 2 =  8 4 * 3 = 12 4 * 4 = 16 4 * 5 = 20 4 * 6 = 24 4 * 7 = 28 4 * 8 = 32 4 * 9 = 36
5 * 2 = 10 5 * 3 = 15 5 * 4 = 20 5 * 5 = 25 5 * 6 = 30 5 * 7 = 35 5 * 8 = 40 5 * 9 = 45
6 * 2 = 12 6 * 3 = 18 6 * 4 = 24 6 * 5 = 30 6 * 6 = 36 6 * 7 = 42 6 * 8 = 48 6 * 9 = 54
7 * 2 = 14 7 * 3 = 21 7 * 4 = 28 7 * 5 = 35 7 * 6 = 42 7 * 7 = 49 7 * 8 = 56 7 * 9 = 63
8 * 2 = 16 8 * 3 = 24 8 * 4 = 32 8 * 5 = 40 8 * 6 = 48 8 * 7 = 56 8 * 8 = 64 8 * 9 = 72
9 * 2 = 18 9 * 3 = 27 9 * 4 = 36 9 * 5 = 45 9 * 6 = 54 9 * 7 = 63 9 * 8 = 72 9 * 9 = 81
```

參考來源：http://blog.ncut.edu.tw/userfile/2992/第7章.pdf

抽象化（Abstraction)

抽象化就是化繁為簡。把複雜概念聚焦在重要地方。

以程式設計而言，就是把複雜計算寫成副程式、或者找出可用函數

例如99乘法表，

可以把計算i*j 寫成一個副程式

輸出的函數print

原始程式

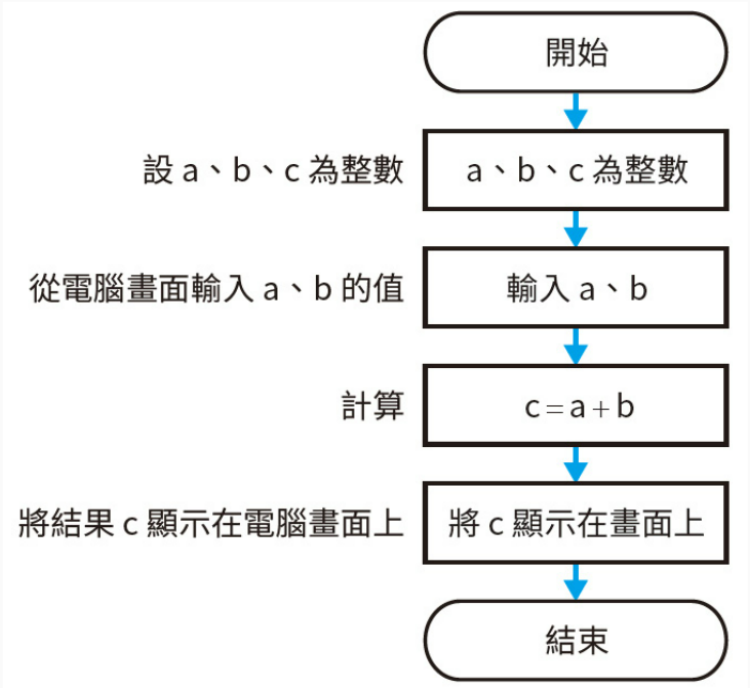
```
#  Example3  99乘法表
for  i  in  range(2,10):
        for  j  in  range(2,10):
                print  ('%d  *  %d  =  %2d  '  %(i,  j  ,  i*j),end="")
        print()
```

抽象化副程式

```
#  Example3  99乘法表
def  product(i,j):
    return  i*j

for  i  in  range(2,10):
        for  j  in  range(2,10):
                print  ('%d  *  %d  =  %2d  '  %(i,  j  ,  product(i,j)),end="")
        print()
```

演算法(Algorithm) ->流程圖
a+b=c

考量這些條件，表示計算a + b = c程式的流程圖如下：



左側是用文句來說明，右側為流程圖。檢視流程圖中的每一個框，比較容易設想若要變成程式語言該寫些什麼。

從細節來看，流程圖無可避免地比用程式語言所寫的程式粗略，但運用流程圖做為寫程式「之前一步」，對替代程式設計來說非有用。

拆解問題

Input

Process

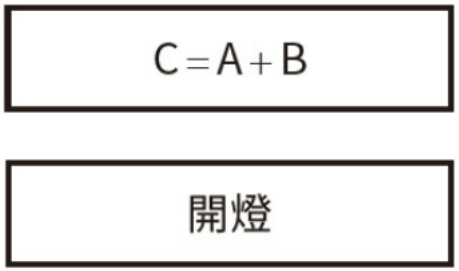
Output

流程圖

► 處理

長方形表示「處理」（process）。

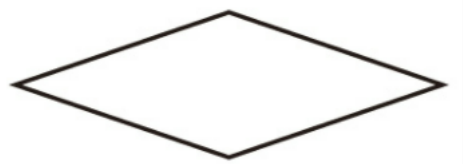
處理係指一般的動作或計算，其內容寫在長方形裡。框內的書寫內容沒有特別的限制。因此，雖然一個框裡也可以寫多項處理，但考量避免處理順序出錯及之後容易修正，一個框盡可能只寫一項處理，這樣比較適當。



【範例】

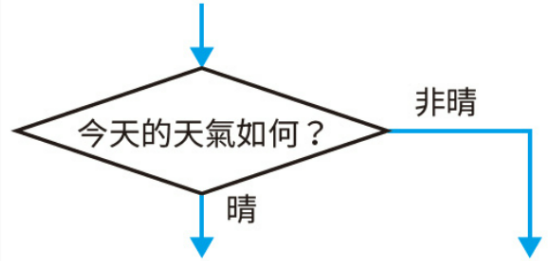
► 判斷

菱形表示因判斷而出現的分歧，有單一入口、多個出口。



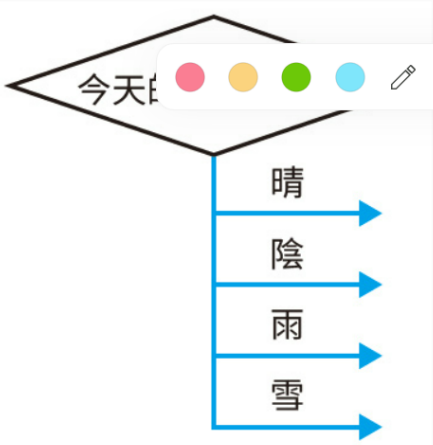
菱形裡書寫分歧條件（分歧判定）。通常最上面的角為入口，其他三個角做為出口。出口的地方記載著分歧條件的判斷結果。有時一個出口也會彙集分出多個分歧，但判斷結果寫在旁邊，看圖就知道在哪裡出現分歧。

菱形表示法範例



這張圖是表示詢問「今天的天氣如何？」時，答案為「晴」就往下方箭頭前進，「非晴（＝陰、雨、雪等，晴以外的所有狀況）」則沿著右邊的箭頭前進。

下圖是從一個出口分出多個分歧的書寫範例。

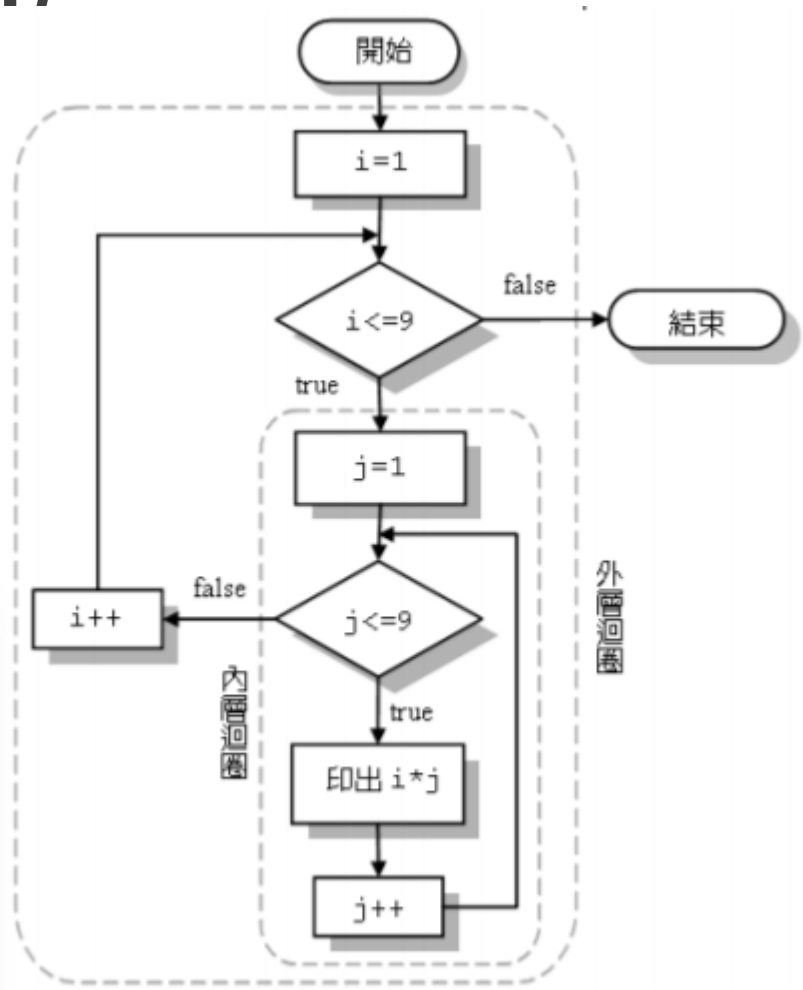


演算法（Algorithm）

演算法，就是按部就班，是規劃解決問題的詳細步驟。

可以用文字描述，也可以用流程圖去描述。

運用前面階段之產出，運用所有可用的副程式或函數，規劃出程式執行的步驟與流程。



- 迴圈裡又有另一層迴圈，稱爲巢狀迴圈
- 九九乘法表的列印：

```
/* prog7_9 OUTPUT------
1*1= 1  1*2= 2  ...  1*9= 9
2*1= 2  2*2= 4  ...  2*9=18
  ...
9*1= 9  9*2=18  ...  9*9=81
------------------------*/
```

```
01  /* prog7_9, 巢狀 for 迴圈印出九九乘法表 */
02  #include <stdio.h>
03  #include <stdlib.h>
04  int main(void)
05  {
06     int i,j;
07     for (i=1;i<=9;i++)      /* 外層迴圈 */
08     {
09        for (j=1;j<=9;j++)   /* 內層迴圈 */
10           printf("%d*%d=%2d  ",i,j,i*j);
11        printf("\n");
12     }
13     system("pause");
14     return 0;
15  }
```

內層迴圈　外層迴圈

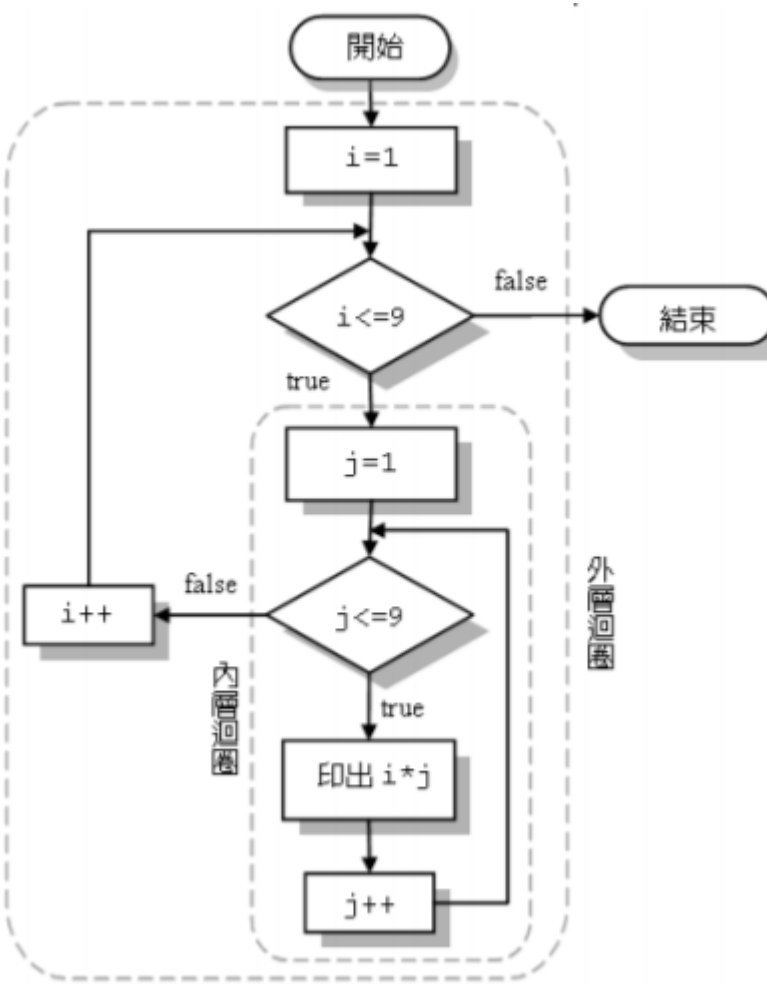
演算法

- 迴圈裡又有另一層迴圈，稱爲巢狀迴圈
- 九九乘法表的列印：

```
/* prog7_9 OUTPUT------
1*1= 1  1*2= 2  ...  1*9= 9
2*1= 2  2*2= 4  ...  2*9=18
  ...
9*1= 9  9*2=18  ...  9*9=81
------------------------*/
```

```
01  /* prog7_9, 巢狀 for 迴圈印出九九乘法表 */
02  #include <stdio.h>
03  #include <stdlib.h>
04  int main(void)
05  {
06    int i,j;
07    for (i=1;i<=9;i++)      /* 外層迴圈 */
08    {
09      for (j=1;j<=9;j++)    /* 內層迴圈 */
10        printf("%d*%d=%2d  ",i,j,i*j);
11      printf("\n");
12    }
13    system("pause");
14    return 0;
15  }
```

內層迴圈　外層迴圈



〔資料〕流程圖符號

本書的流程圖基本上只用了三種符號。然而，大家看到坊間的流程圖時，若心生「這是什麼？」的疑問會有點難堪，所以本篇介紹JIS X 0121−1986（資訊處理用流程圖、程式網狀圖、系統資源圖符號）的規格中比較常見的符號。

下表摘錄自JIS X 0121−1986並經適當修改。項目欄圓括弧內的數字是根據JIS規格所載，跳過的數字是一般少用的符號，省略不記。

分類	項目	符號	說明
基本資料符號	(1)資料 data		表示非特定媒介的資料
個別資料符號	(2)儲存資料 stored data		表示處理為適當形式儲存的資料。非特定媒介
	(1)內部儲存 internal storage		表示內部儲存媒介的資料
	(2)循序存取儲存 sequential access storage		表示只能循序存取的資料。磁帶、匣式磁帶和卡式磁帶等
	(3)直接存取儲存 direct access storage		表示能直接存取的資料。磁碟、磁鼓（mag− netic drum，依靠磁介質的鼓形體資料儲存裝置）和軟性磁碟等
	(4)文件 document		表示人可讀取的媒介上的資料。文字輸出、微縮膠片、計算紀錄、傳票等
	(5)人工輸入 manual input		表示手工操作輸入資訊到各種媒介上的資料
	(8) 顯示 display		表示將人使用的資訊顯示到各種媒介上的資料。顯示裝置的畫面等

基本處理 符號	(1)處理 process		表示任何類型的處理功能
個別處理 符號	(1)已定義處理 predefined process		表示次常式（subroutine）或模組等在其他地方已定義好的一個以上操作或指令群所組成的處理
	(2) 人工操作 manual operation		表示人工進行的任何處理
	(3) 準備作業 preparation		表示會影響其後動作的指令或指令群的修改。多重判斷（switch）的設定、常式（routine）的初始設定等
	(4) 決策判斷 decision		表示具有一個入口和數個擇一的出口，根據符號中定義好的條件評估，選擇唯一出口的決策判斷功能或多重判斷形式的功能
	(5)平行處理 parallel processing		表示讓兩個以上平行的處理同步化
	(6)迴圈極限 loop limit		由兩個部分組成，表示迴圈的開始和結束。有起點和終點（垂直反向）
基本線符號	(1)流線 line		表示資料或控制的流向。必須明確表示流向時，加上箭頭
個別線符號	(2)通訊 communication		表示透過通訊線來傳遞資料
	(3)虛線 dashed line		表示兩個以上的符號之間為擇一的關係
特殊符號	(1)連接點 connector		表示往同一流程圖中其他部分的出口或從其他部分進來的入口。對應的連接點含有同一個特定的名稱

	(3) 起止符號 termination		表示往外部環境的出口或外部環境進來的入口。程式的開始、結束等
非標準符號	跨頁連接點 off-page connector		表示往同一流程圖中他頁的出口或從他頁進來的入口。用以與同頁所用的連接點做區別

運算思維練習(示範)

完成以下程式。

使用者輸入n，顯示1~n的星星

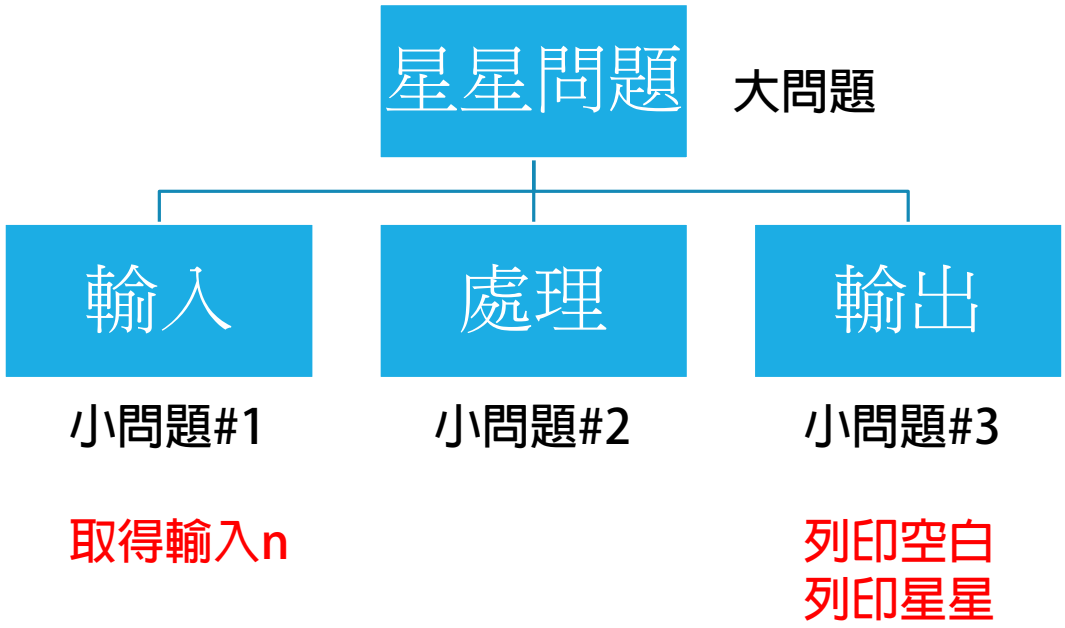
```
Prg #D
    *
   **
  ***
 ****
*****
```

問題拆解（Decomposition)

問題拆解是化整為零，將一個大問題拆解成數個小問題。

以程式設計為例，通常至少可以區分成以下小問題。

例如：星星



Prg #D

*

**

***

****

*****

模式識別（Pattern Recognition)

模式識別就是異中求同。

在龐雜的訊息中，找到相似之處，那個相似之處就是模式(pattern)。

以程式設計而言，就是找出重複被執行的地方（指令）

星星

Prg #D

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

抽象化（Abstraction)

抽象化就是化繁為簡。把複雜概念聚焦在重要地方。

以程式設計而言，就是把複雜計算寫成副程式、或者找出可用函數

主題：星星

產出: 抽象化副程式:

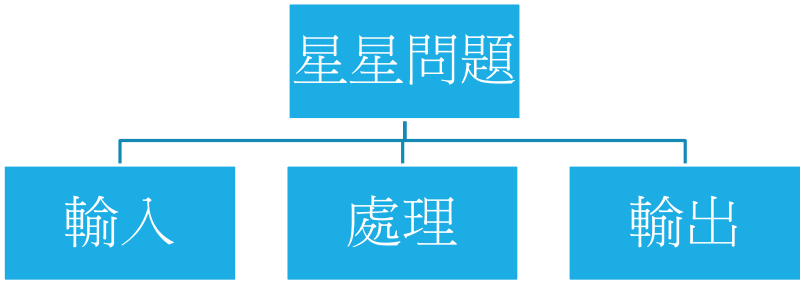
可以把列印星星寫成一個副程式

列印空白副程式 print_space

列印星星副程式 print_star

可用函數

1. input
2. print



小問題#1

取得輸入n
Input()

小問題#2

迴圈控制i

小問題#3

列印空白
列印星星
換行

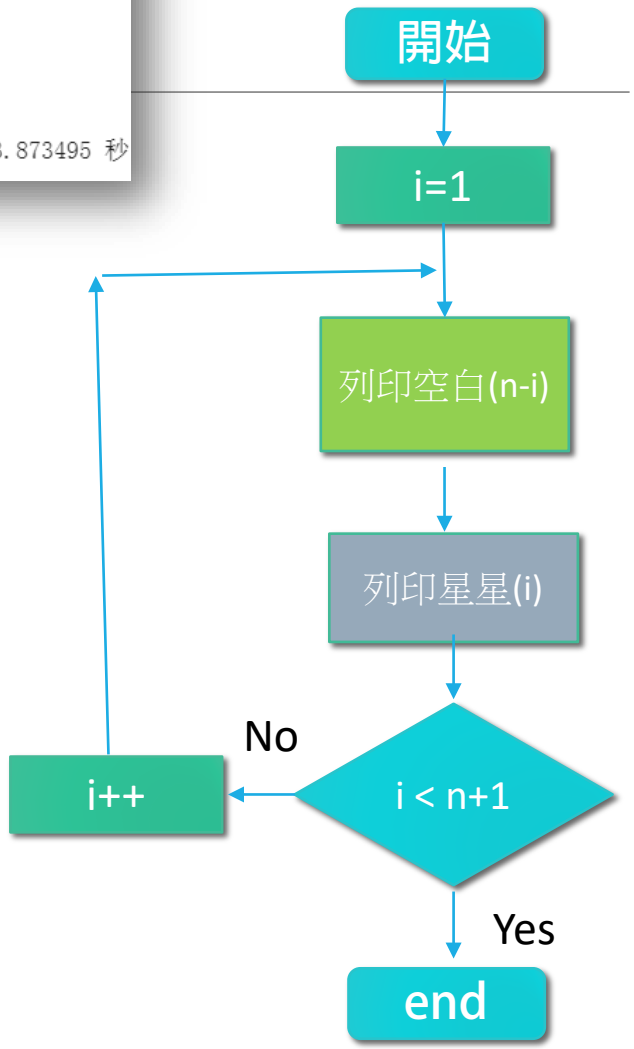
Print_space(?)
Print_star(?)
Print('\n')

演算法 （Algorithm)

流程圖

```
#  prg4  startA_sub
import  time
start  =  time.time()

n  =  int(input('請輸入數字  ?'))

#副程式
def  print_space(n):
    for  i  in  range(0,  n):
        print('  ',end="")
    return

def  print_star(n):
    for  i  in  range(0,  n):
        print('*',end="")
    return

#主程式
for  i  in  range(1,n+1):
    print_space(n-i)
    print_star(i)
    print('\n')

end  =  time.time()
print("花費時間  %f  秒"  %  (end-start))
```

```
請輸入數字 ?3
  *
 **
***
花費時間 3.873495 秒
```



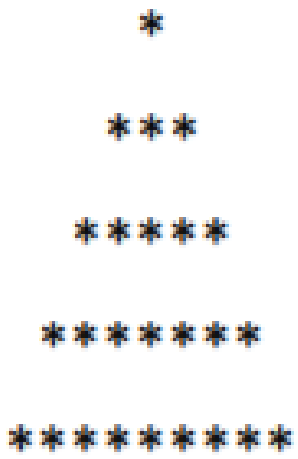
運算思維實作

以運算思維概念，分析以下題目後並完成題目。

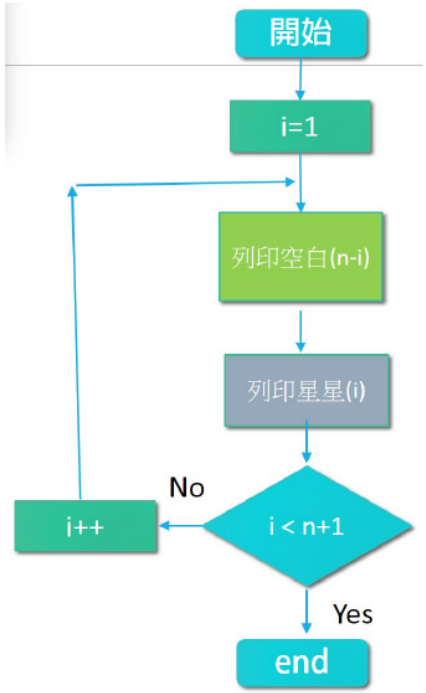
題目

題目：輸入n，印出對應的星星

Prg #E



完成以下的內容　示意

運算思維	回答
問題拆解 化整為零，將一個大問題拆解成數個小問題	輸入：取得輸入 n 處理： 輸出：列印空白、列印星星
模式識別 找出重複被執行的地方（指令）	重複執行之處： 空白列印、 單行星星列印
抽象化 把複雜計算寫成副程式、找出可用函數	可用函數：print、input 副程式：列印空白 print_space、列印單行星星 print_star
演算法 運用所有可用的副程式或函數，規劃出程式執行的步驟與流程。	流程圖 

完成以下的內容

程式碼 (示意)

```
#  prg4  startA_sub
import  time
start  =  time.time()

n  =  int(input('請輸入數字  ?'))

#副程式
def  print_space(n):
    for  i  in  range(0,  n):
        print('  ',end="")
    return

def  print_star(n):
    for  i  in  range(0,  n):
        print('*',end="")
    return

#主程式
for  i  in  range(1,n+1):
    print_space(n-i)
    print_star(i)
    print('\n')

end  =  time.time()
print("花費時間  %f  秒"  %  (end-start))
```

執行結果(示意)

```
請輸入數字 ?3
  *

 **

***

花費時間 3.873495 秒
```