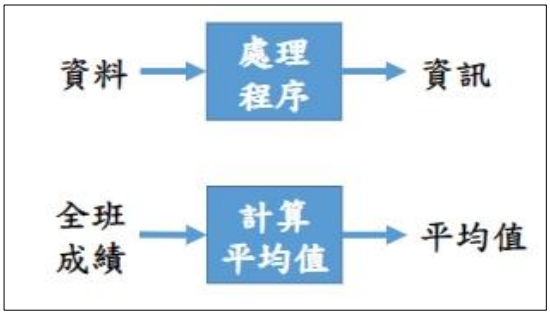
# 壹、資料結構簡介

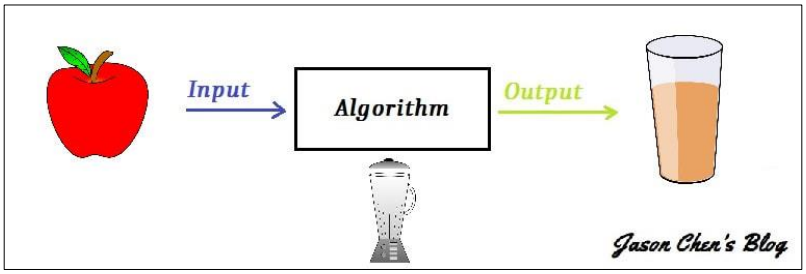
## 一、資料與資訊的關係

1. 資料(Data)：是客觀存在的、具體的、事實的
2. 資訊(Information)：資料處理過後的結果




## 二、何謂資料結構

1. 探討如何將資料更有組織地存放到電腦記憶體中，以提升程式的執行效率
2. 有組織地存放資料牽涉到「存放與取用的方法」&「如何分布在電腦記憶體內」
3. 演算法（Algorithm）五大原則：
   ①輸入資料（Input）：由演算法的外面給予資料(可有可無)
   ②輸出資料（Output）：產生結果
   ③有限性（Finiteness）：在有限的步驟內結束演算法
   ④有效性（Effectiveness）：正確且有效率產生結果
   ⑤明確性（Definiteness）：明確的執行動作或步驟




## 三、演算法（Algorithm）的效率評估

1. 演算法的簡單定義式：輸入 + 演算法 = 輸出
2. 用來計算某些演算法所撰寫的程式，在經過編譯之後實際執行的時間&使用及占用的記憶體空間
   ①時間複雜度（Time complexity）：從該演算法執行開始到結束所花的時間總長
   (1)執行時間 = 執行次數 x 每次執行所需時間
   (2)每次執行時間牽扯到 CPU 的執行速度

```
for( i=1 ; i<=100 ; i++)  ← 執行101次
{
        printf("%d",i);  ← 執行100次
}
此演算法總共執行次數：201 次
```

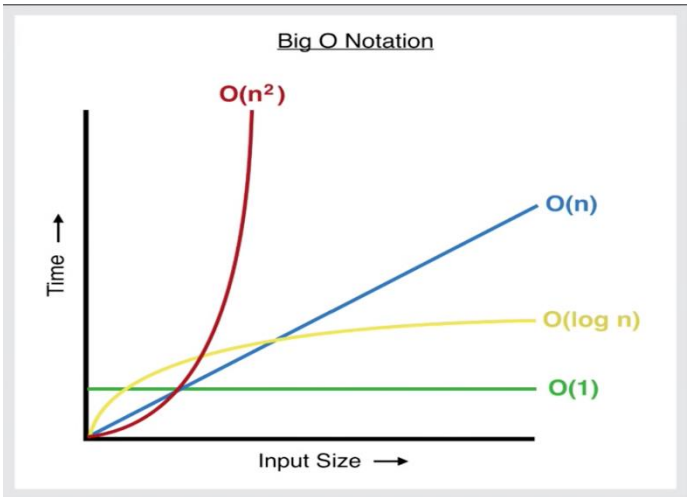
```
for( i=1 ; i<=n ; i++)  ← 執行n+1次
{
        printf("%d",i);  ← 執行n次
}
此演算法總共執行次數：2n+1 次
```

   ②空間複雜度（Space complexity）：從該演算法載入記憶體開始占用與使用的記憶體空間

## 四、Big-O of n

1. Big O 符號是用來描述一個演算法在輸入 n 個東西時，總執行時間與 n 的關係。
2. O(n)
   ①用來表示理論的「上限」
   ②若演算法的執行次數為 f(n)，若說其具有 Big-O of g(n)，就表示存在某個常數 c，可以使得當 n 大於某個正整數時，存在 f(n)小於或等於 c*g(n)
3. 時間複雜度的等級

$$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)<O(2^n)<O(n!)$$




**※請問以下執行次數：**

(1)假設兩矩陣 a,b 皆為 n*n

```
void add(int a[ ][ ], int b[ ][ ], int c[ ][ ], int n)
{
int i, j;

for(i=0; i<n; i++)
for (j=0; j<n; j++)
c[i][j] =a[i][j]+b[i][j]; }
```

(2)

```
int sum(int arr[ ], int n)
{
int i, total=0;

for (i=0; i < n; i++)
total += arr[i];
return total; }
```