

函數撰寫

什麼時候該寫函數

2015 資訊之芽/語法班, Gin

函數使用時機

- 什麼時候該寫函數?
- 跟迴圈有點類似:
- 1.需要重複做一件事的時候
- 2.模組化(modular)你的程式，讓它變好看，容易除錯(debug)

使用時機 - 1.重複做一件事

- 你的程式經常需要算兩點之間的距離，
- 天啊，每次都要寫一堆乘號 根號 括號？
- 想像如果有一個函式 `distance()`，數字丟進去就幫你算好！

```
int a = distance(x1,y1,x2,y2)
```

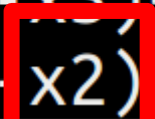
搞不好在 `cmath` 裡面有 !?

- 別傻了孩子，自己寫吧

Bad : 不寫函數

非常混亂，難以閱讀，容易有bug

```
int a = sqrt( (x1-x2)*(x1-x2)+(y1-y2)*(y1-y2) );  
int b = sqrt( (x2-x3)*(x2-x3)+(y2-y3)*(y2-y3) );  
int c = sqrt( (x3-x1)*(x3-x2)+(y3-y1)*(y3-y1) );
```



```
int a = sqrt( (x1-x2)*(x1-x2)+(y1-y2)*(y1-y2) );  
int b = sqrt( (x2-x3)*(x2-x3)+(y2-y3)*(y2-y3) );  
int c = sqrt( (x3-x1)*(x3-x1)+(y3-y1)*(y3-y1) );
```

Good : 使用函數

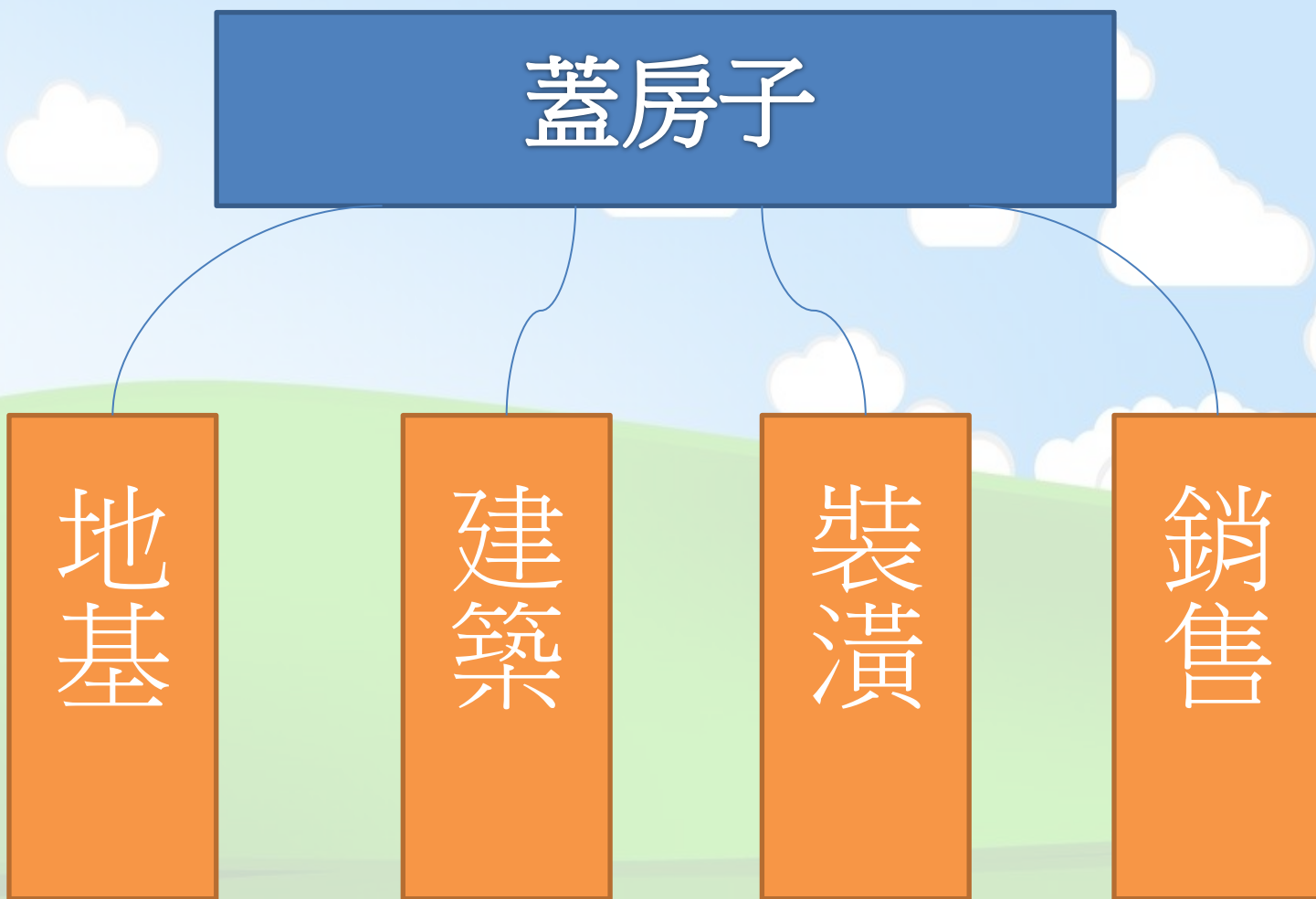
程式簡潔，易於閱讀，沒有bug....

```
int a = distance(x1, y1, x2, y2);  
int b = distance(x2, y2, x3, y3);  
int c = distance(x3, y3, x1, y1);
```

使用時機 – 2.模組化

- 模組化是什麼呢？
- 就是把大問題切成小問題，一步一步處理的方法！
- 一個組裝的概念，把東西切成很小塊之後拼起來！
- 一個難寫的程式，把問題分成很多函數實作之後，合體！

使用時機 – 2. 模組化



使用時機 – 2.模組化

- 設想一個問題：
輸入兩個陣列，請分別對這兩個陣列的最大值取最大公因數之後輸出。
- 步驟：
 - 先找出第一個陣列的最大值
 - 再找出第二個陣列的最大值
 - 兩個數字取最大公因數
 - 最大公因數：從1跑到比較小的那個值

Bad : 不寫函數

難懂 複雜，看不出每一塊的作用，難debug

```
int aMax = a[0];
for(int i = 0; i < aLen; i++)
    if(a[i] > aMax)
        aMax = a[i];

int bMax = b[0];
for(int i = 0; i < bLen; i++)
    if(b[i] > bMax)
        bMax = b[i];

int gcd;
for(int i = 1; i <= min(aMax, bMax); i++)
    if(bMax % i == 0 && aMax % i == 0)
        gcd = i;
cout << gcd << endl;
```

Good : 使用函數

三行收工，程式簡潔，極易閱讀，bug好處理

```
int aMax = findMax(a);  
int bMax = findMax(b);  
cout << gcd(aMax, bMax);
```

使用函數的好處

- 包裝(黑盒子)：即使裡面做的事很複雜，看上去只是一行就解決了
`sqrt()`
- 把問題變簡單：有效的將大問題切成許多小問題(函數)
- 易於維護/debug：需要更改時只要改變函數內部的程式碼就好，不用在茫茫程式碼中尋找.....

注意事項

- 函數也要命名，請取好名字
isPrime(), findMax(), lcm()
不要 f(), func(), compute()
- 寫大一點、較難的程式時請把函數記在腦海，不要懶得寫函數，不然之後debug的痛苦你就知道了

