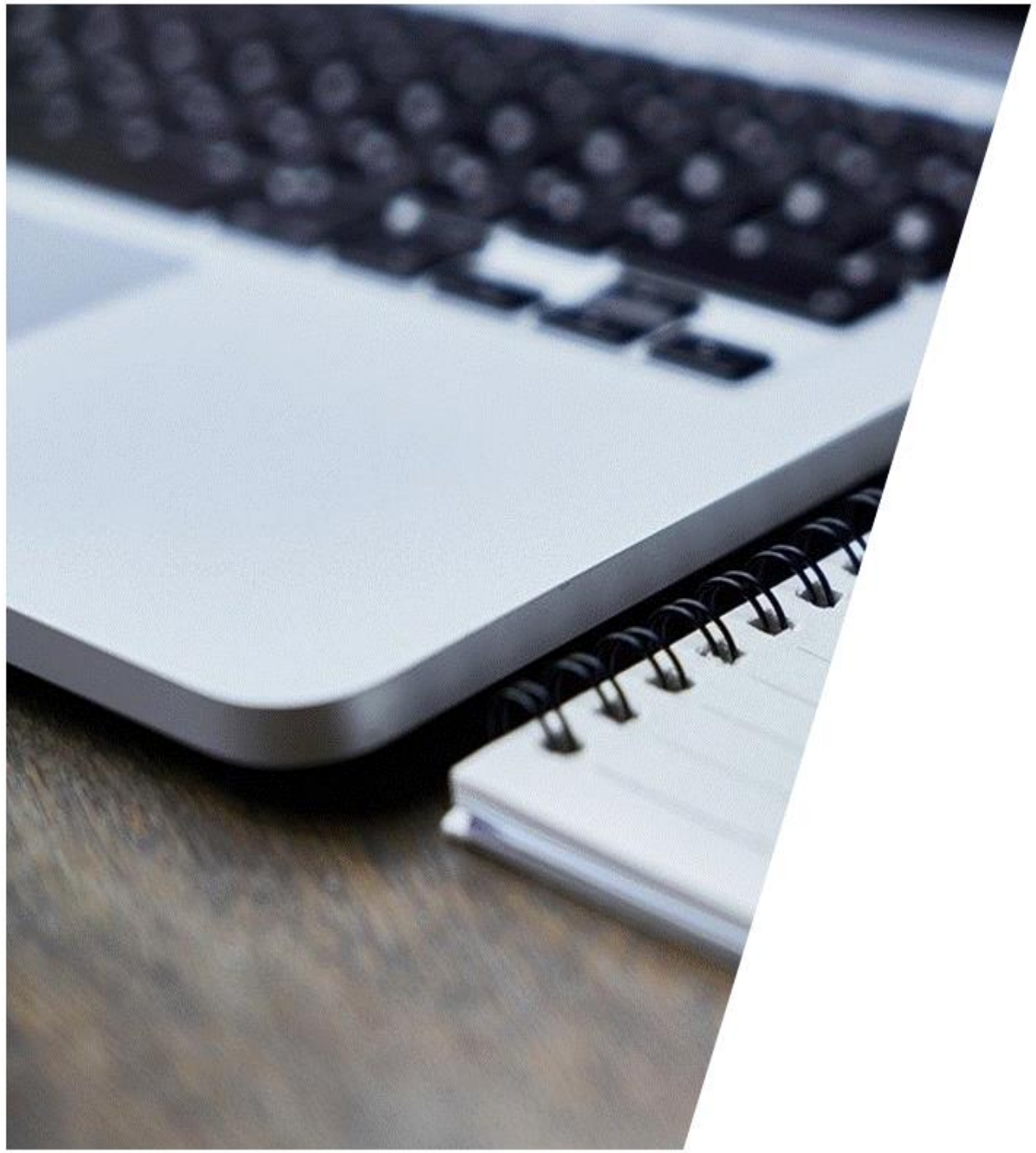


C++

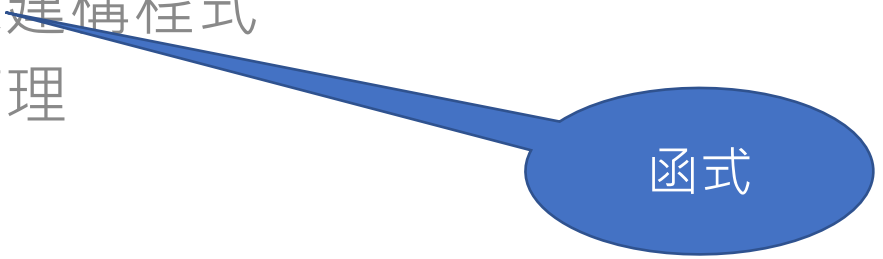
函式

各個擊破

是解決真實問題電腦程式的好方法

真實的電腦程式比我們的練習題大很多，維護大型程式的方法：
以較小的單元或模組來建構程式

小單元要比大程式好管理



C/C++語言=
C標準函式庫+程式設計師所寫的新函式

main( )是一個函式

```
#include <iostream>
using namespace std;

int main( ) {
    int x;
    cin>>x;
    cout<<x;
    return 0;
}
```

main( )是程式設計師自行發展的主程式，主程式是函式的一種類型，主程式配合標準函式庫stdio.h產生一個程式的新功能

#include <iostream>是什麼?

```
#include <iostream>
using namespace std;

int main() {
    int x;
    cin>>x;
    cout<<x;
    return 0;
}
```

- 輸出/入函式來自於iostream函式庫。
- include是將iostream函式庫引入程式，因為cin與cout來自於iostream函式庫。

標準函式庫

- 包含
  常用的數學運算
  字串處理
  字元處理
  輸入輸出
- 目的
  提供程式設計師所需要的大部分功能，減輕程式設計師的負擔

C++

的自訂函式定義

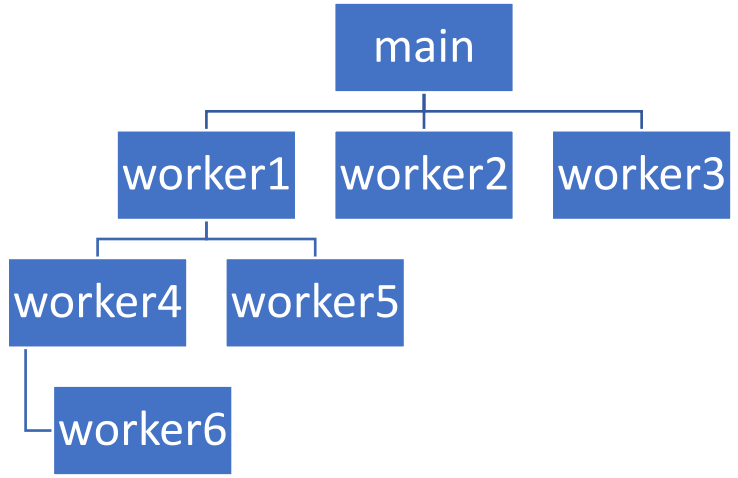
自訂函式-又稱爲副程式(子程式)結構

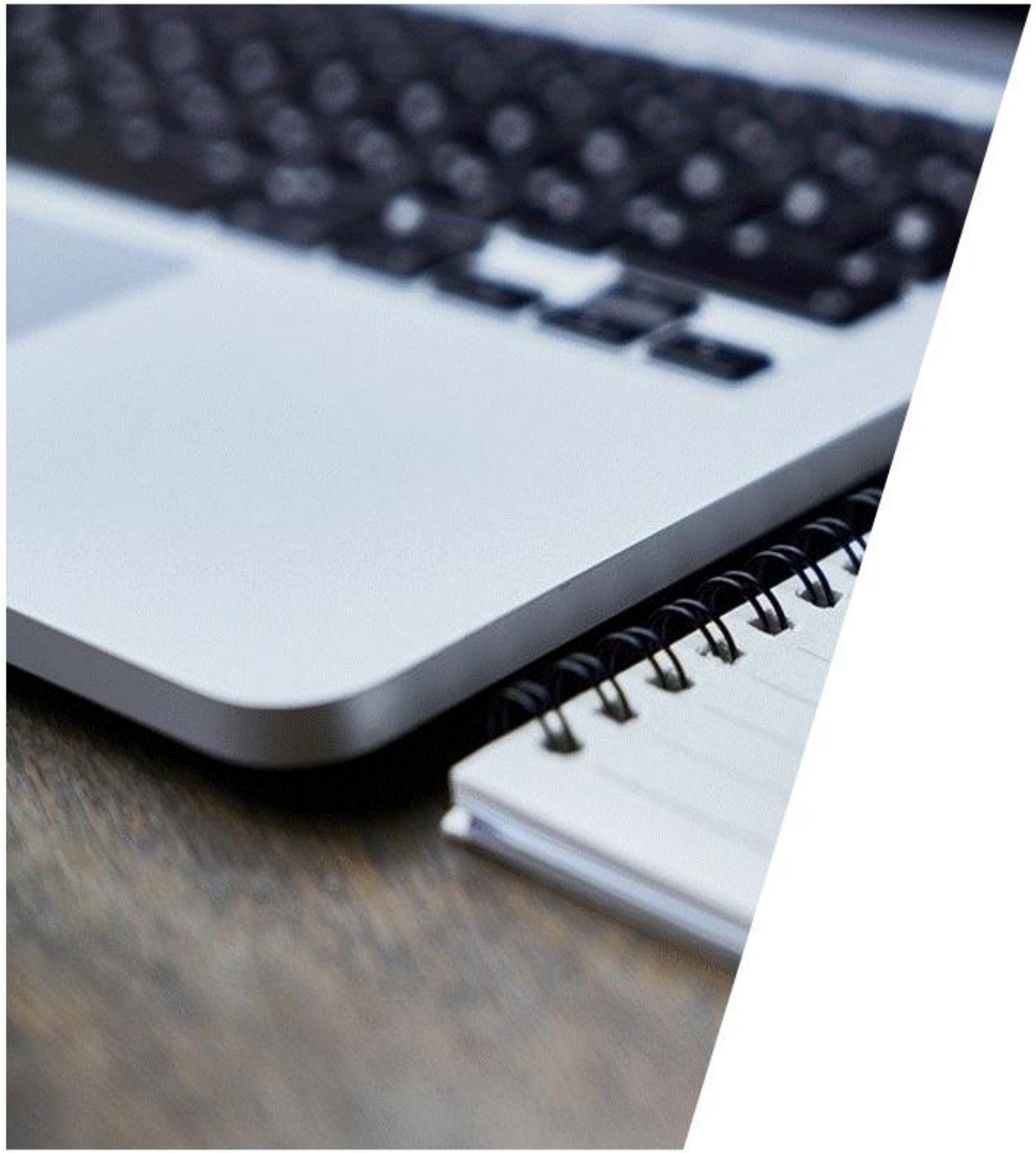
- 副程式(子程式)結構
  - 如果有一些程式段在不同程式的地方反覆出現，可以將這些程式段做爲相對獨立的整體

- 好處
  - 可以縮短程式
  - 節省內存空間
  - 減少程式編譯時間

函式的使用概念

- 函式經由呼叫(function call)的方式引用(invoked)，函式的呼叫指明了要引用的函式名稱，並提供資訊（當作引數）給受呼叫函式，以執行受呼叫函式的工作
- 道理十分類似人事管理，老闆（呼叫函式或呼叫者）要求某位員工（受呼叫的函式）去執行某項工作，並在工作後完成回報。
- 例如老闆函式呼叫員工函式printf去執行這項工作，然後printf將資訊顯示出來，並且在顯示之後回報、返回或呼叫函式，但老闆函式並不知道員工函式如何執行，老闆函式也可以再呼叫其他函式





C++

函式的定義

函式定義的語法形式

函示標頭

傳回值的資料類型 函式名稱(參數列表){

函式主體
(由執行的語句所構成)

}

函式定義的語法形式

傳回值的資料類型 函式名稱(參數列表){

函式主體
(由執行的語句所構成)

}

- **傳回值的資料類型 ：**
  函式傳回主程式值的類型，若是void則表示無傳回值，未指定的話，編譯器會視為整數
- **函式名稱**
  主程式的名字一定是main ，其餘函式的名字可以是任何合法的是別字
- **參數列表**
  - 可以是空的，不管有沒有參數，( )都不能去掉
  - 也可以多個，參數之間以逗號隔開
  - 參數必須有類型說明，可以是變數、陣列或指標
- 函式最外層是一對{}

C++

函式的宣告和呼叫

函式的宣告

寫法1：

原型，int slave();

然後將子程式寫在主程式後

```
#include <iostream>
using namespace std;

int slave();

int main() {
    cout<<slave()<<enld;
    return 0;
}

int slave(){
    return (1+10)*10/2;
}
```

寫法2：

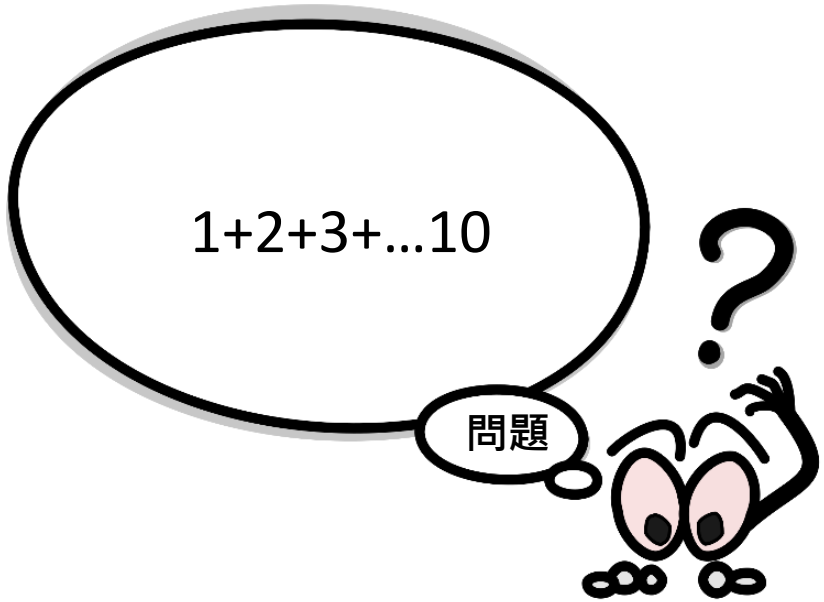
將子程式提到主程式之前

```
#include <iostream>
using namespace std;

int slave(){
    return (1+10)*10/2;
}

int main() {
    cout<<slave()<<enld;
    return 0;
}
```

函式的形式(類型1)-無參數函數與傳值回主程式

```
#include <iostream>
using namespace std;

int slave( );

int main( ) {
    cout<<slave( )<<enld;
    return 0;
}

int slave( ){
    return (1+10)*10/2;
}
```

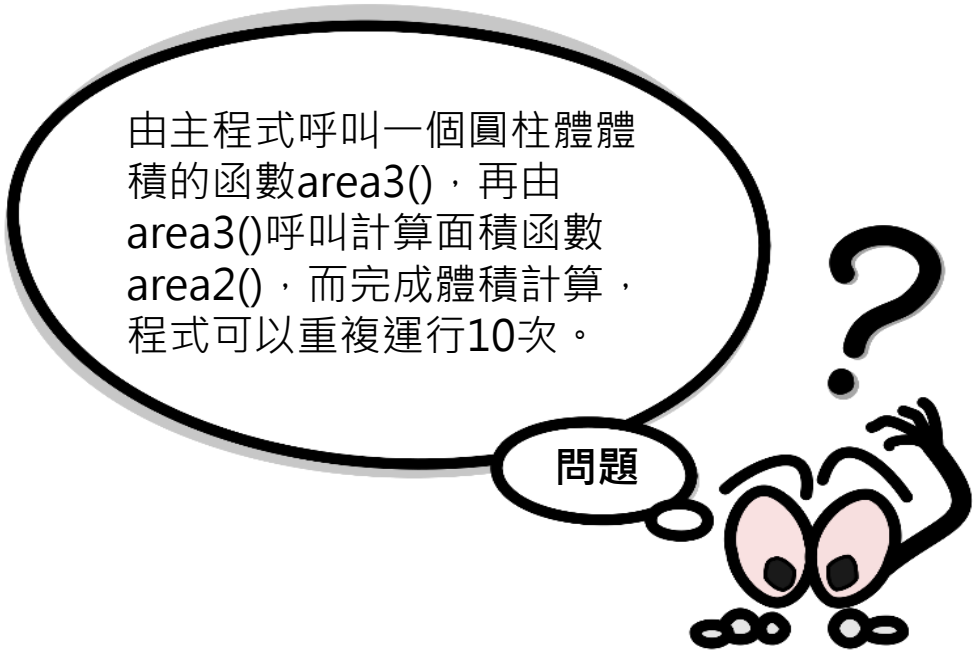
序號	說明
1	呼叫副程式slave
2	傳回slave副程式執行結果
3	傳回值是一個整數類型



函式的形式(類型2)-有參數函式與傳值回主程式

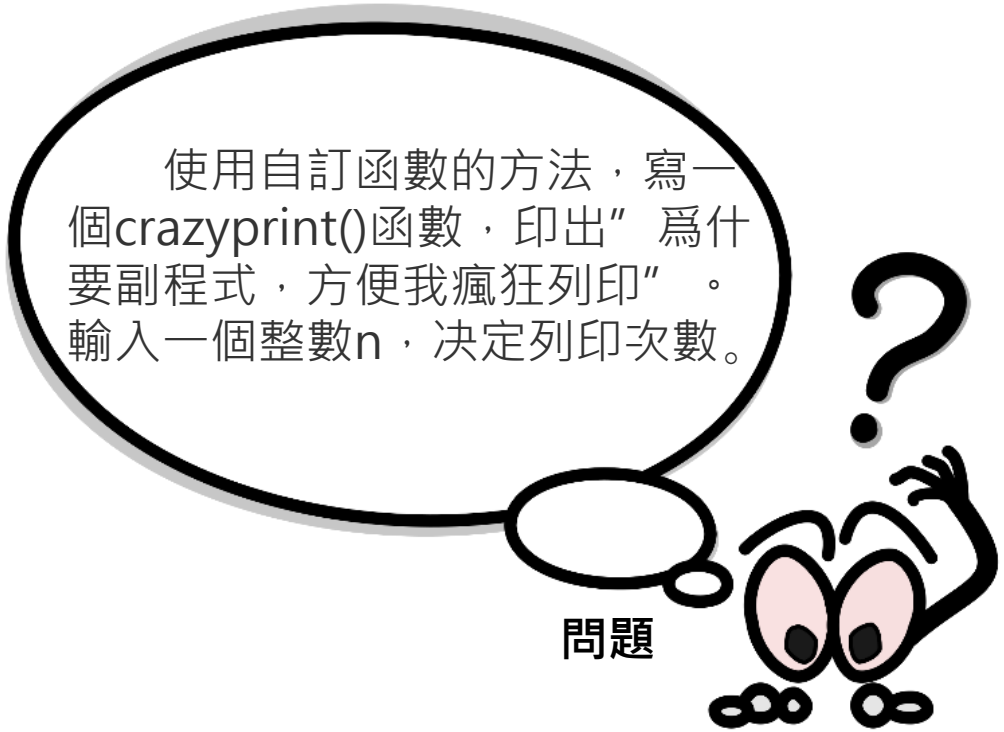
```
#include <iostream>
using namespace std;

float area3(float r,float h);
float area2(float r);

int main(){
    float p=3.14,r,h;
    int i;
    for(i=1;i<=10;i++){
        cin>>r>>h;
        cout<<"圓柱體體積"<<area3(r,h))<<endl; // 1
    }
    return 0;
}

// 2
float area3(float r,float h){
    return area2(r)*h;
}

// 3
float area2(float r){
    p=3.14;
    return 3.14*r*r;
}
```

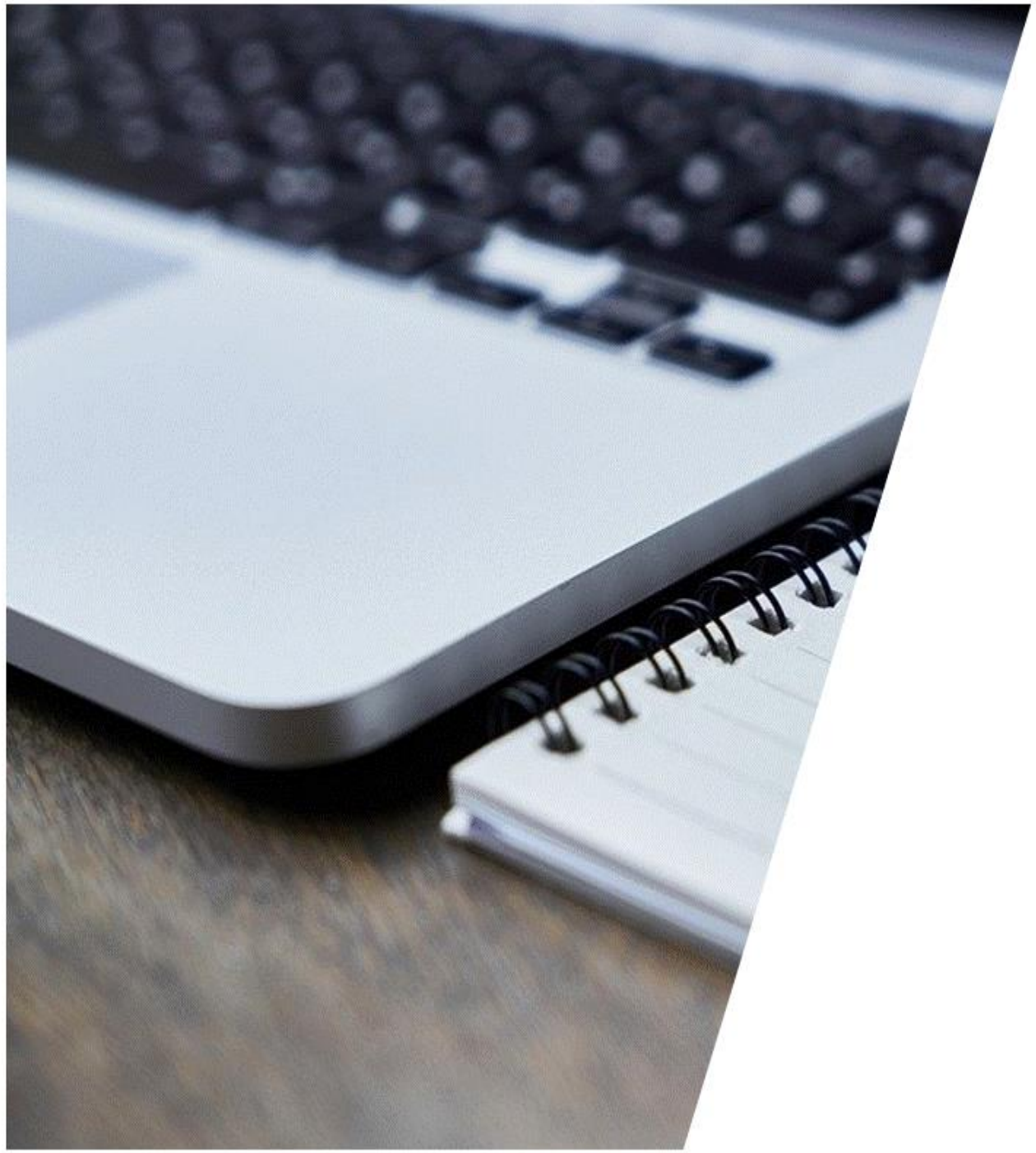
序號	說明
1	呼叫子程式area3(r,h) area3含有2個參數，分別代表半徑與高
2	參數函數中必須定義半徑與高的資料型態
3	子程式可以再細分工作呼叫另一個子程式



式的形式(類型3) -沒有傳值回主程式

```
#include <iostream>
using namespace std;

void crazyprint();  void代表沒有傳回值

int main(){
    int i,n;
    cin>>n;
    for(i=1;i<=n;i++){
        crazyprint();
    }
    return 0;
}

void crazyprint(){
    cout<<"為什麼要副程式，方便我瘋狂列印。"<<endl;
}
```





C++

延伸閱讀

常見的錯誤

- 如果函式原型指定的傳回值不是int, 則在函式中忽略定義，將會倒指語法錯誤
- 望了從一個必須傳回值的函式傳回值，可能會產生不可預期的錯誤
- 從一個回傳行別宣告void的函式裡傳回數值，將導致語法錯誤
- 相同型別的宣告，不能簡略為(float x, y)，否則y會被視為預設的
- 參數列的右方有分號，將造成語法錯誤
- 在函式內將此函式的某個參數定義為區域變數，將造成語法錯誤
- 對於傳給某函式的引數，以及該函式定義中相對應的參數，盡量不要使用相同的名稱，避免混淆

```c
#include <stdio.h>

float area3(float r,float h);
float area2(float r);

int main(){
    float p=3.14,r,h;
    int i;
    for(i=1;i<=10;i++){
        scanf("%f %f",&r,&h);
        printf("圓柱體體積%.2f\n" ,area3(r,h)); // 1
    }
    return 0;
}

// 2
float area3(float r,float h){
    return area2(r)*h;
}

// 3
float area2(float r){
    p=3.14;
    return 3.14*r*r;
}
```

全域變數

- **定義：**
  在函數外部，沒有被小括號括起來的變數

- **作用域：**
  從變數的定義開始到文件結束

- 程式中的任何函數都可以使用

```
#include <stdio.h>

int x,y;   全域變數

int gcd(int x,int y){ 局域變數
    int r = x % y;
    while(r!=0){
        x = y;
        y = r;
        r = x % y;
    }
    return y;
}

int lcm(){
    return x * y / gcd(x,y);
}

int main() {
    cin>>x>>y;
    cout<<lcm()<<endl;
}
```

全域變數的說明

- 在一個函數內部，既可以使用函數定義的區域變數，也可以使用在此函數前定義的全域變數
- 使用時機
  使得函數間多一種傳遞資訊的方式，如果一個程式多個函數都要對一個變數進行處理
- 若沒有設定賦值，其預設值是0
- 執行過程中一直占用記憶體
- 副作用
  - 會增加除錯困難
  - 降低程式的通用性

區域變數

- 作用域是在定義該變數的函式內部。函式執行完畢，區域變數的空間就被釋放，值無法被保留到下次使用。
- 在不同的函式中變數名可以相同，記憶體中佔據不同的記憶體單元，互不干擾。
- 區域變數與全域變數是可以重名的，在相同作用域內，區域變數有效時，對全域變數無效。
- 副程式中定義的變數的存在時間和作用會被限制在該區塊中。
- 主程式main( )中的定義是一種區域變數。
- 若沒有給賦值，區域變數值是隨機的，區域變數受堆疊空間大小限制，大陣列要注意。

全域變數 / 區域變數

- **相同：**
  - 需要遵守變數的命名規則

- **相異：**
  - 變數範圍(scope)不同：scope 指的是變數可被使用或呼叫的地方，全域變數是在整份程式內都可以使用，區域變數只能在被定義的函示中使用。