

程式人



如何用十分鐘

快速瞭解一個程式語言

《以 JavaScript 和 C 語言為例》

陳鍾誠

2016 年 3 月 7 日

# 話說

- 從我大學到現在，學過不少程式語言。

# 大部分的程式語言

- 其實都長得差不多！

偶爾有幾個長得很不一樣的

- 其實都沒有多少人用
- 所以學不會也沒關係！

# 到底

- 哪些程式語言是長的一樣的？
- 哪些程式語言又長得不太一樣呢？

先讓我想想看

# 長得都差不多的程式語言有

- Pascal, C, C++, Java, VB, C#, Ruby, Python, JavaScript, Go, R, Foxpro 等等

# 長得很不一樣的程式語言

- 好像不算太多，我學過的就只有 Prolog 和 LISP 兩個，沒學過的聽說還有 ML, Haskell 等語言



# 那些長得差不多的

- 通常是《程序式的語言》，也就是有 if, while, function 的那些《結構化流程式語言》

# 而長得很不一樣的

- 通常屬於《宣告式語言》  
(declarative programming language )，這類語言通常沒有迴圈，得用遞迴替代迴圈。

# 那種沒有迴圈的語言

- 通常無法變成主流語言
- 我也通常不會去用，所以自然也學不好！

# 而那種有迴圈的語言

- 因為長得都很像
- 所以只要學會一個之後，要學另一個就很快！

# 有時花個十分鐘看語法

- 然後就可以開始寫了！

# 現在

- 我決定以過來人的經驗
- 分享一下自己學新語言時的做法！

# 當我想學一門新的程式語言時

- 我通常會先買一本那門程式語言的書
- 書還沒到之前，可以先上網看一下範例。

# 然後

- 我會看看這個語言的
  - 變數怎麼宣告
  - if 語法怎麼寫
  - 迴圈語法怎麼寫
  - 陣列（和字典）的存取如何做
  - 函數如何宣告和使用



# 大約花上十分鐘看完後

- 就可以開始動手寫程式了！

# 當然、動手之前

- 一定要先安裝《開發環境》

# 舉例而言

- 如果要學 javascript ，我會先安裝 node.js
- 如果要學 C ，我會先安裝 gcc  
( 或包含 gcc 的開發環境，像是 Dev C++ 等 )

# 接下來

- 就直接開始試著寫寫看！

到底應該寫些甚麼程式呢？

# 根據程式人不成文的慣例

# 第一個程式通常是寫 Hello World!

檔案: hello.js

JavaScript 版

```
console.log('hello 你好!');
```

執行結果

```
$ node hello.js  
hello 你好!!
```

檔案: hello.c

C 語言版

```
#include <stdio.h>  
  
int main() {  
    printf("hello 你好! \n");  
}
```

執行結果

```
D:\code>gcc hello.c -o hello  
D:\code>hello  
hello 你好!
```

# 請注意

- 在上述程式中，我輸出  
《Hello！ 你好！》  
順便測試一下中文有沒有問題！
- 有不少開發環境的中文會出現問題



# 所以

- 要小心一些事情

- 像是檔案資料夾名稱最好不要用中文。

- 路徑裡最好不要有空白

- (windows 中盡量不要將檔案存在桌面)

# 還有、對於新手而言

- 要學會使用命令列與 Shell
  - 像是路徑的切換
  - 列出有哪些檔案
  - 等等指令！

# 以下是命令列 Shell 的使用方式

DOS 指令	Shell 指令	說明	範例	範例解說
cd	cd	change directory	cd /coc/code/	切換到 /coc/code/ 資料夾
dir	ls	directory	dir	顯示目前資料夾中的檔案與子資料夾
d:	無磁碟機概念	切換磁碟機	d:	切換到 d 槽



windows

MAC/Linux

# 以下是我執行 hello.js 的案例

```
C:\Users\user>d:
```

```
D:\>dir
```

```
...
```

2015/10/11	上午 08:48	<DIR>	sport
2016/03/02	下午 08:32	<DIR>	temp
2015/12/25	下午 08:45	<DIR>	upload
2015/12/25	下午 09:06	<DIR>	照片

```
0 個檔案          0 位元組
```

```
16 個目錄 15,897,038,848 位元組可用
```

```
D:\>cd jscode
```

```
D:\jscode>dir
```

```
...
```

2015/10/15	上午 08:30	683 gensentence.js
2013/03/14	上午 08:53	22 hello.js
2013/03/18	下午 01:53	240 HelloWeb.js

```
...
```

```
D:\jscode>node hello.js
```

```
Hello!
```

# 您可以看到

- 我先用 d: 切換磁碟機
- 然後再用 cd 切換資料夾  
到我的程式所在位置
- 最後才用 node hello.js  
執行我寫的程式

```
C:\Users\user>d:
```

```
D:\>dir
```

```
...
```

2015/10/11	上午 08:48	<DIR>	sport
2016/03/02	下午 08:32	<DIR>	temp
2015/12/25	下午 08:45	<DIR>	upload
2015/12/25	下午 09:06	<DIR>	照片

```
0 個檔案          0 位元組
```

```
16 個目錄 15,897,038,848 位元組可用
```

```
D:\>cd jscode
```

```
D:\jscode>dir
```

```
...
```

2015/10/15	上午 08:30	683 gensentence.js
2013/03/14	上午 08:53	22 hello.js
2013/03/18	下午 01:53	240 HelloWeb.js

```
...
```

```
D:\jscode>node hello.js
```

```
Hello!
```

# 雖然對於熟手而言

- 這非常的簡單，根本就是常識！

但是對於新手而言

# 最麻煩的

- 就是那些大家都不講的《常識》！



# 而且

- 每個作業系統和開發工具的常識都長得不太一樣！
  - Win Xp, Win7, Win8, Win10
  - MAC, Linux, ...
  - iOS, Android, ...
  - 甚至有些系統改版時還會大搬風一下！

# 像是上次我要寫 Android 版 PhoneGap 的 Hello World 時

- 怎麼試都沒辦法在自己的 Galaxy Tab4 手機上面執行！
- 結果發現這個，顯然 google 把程式人員當 007 在訓練

## 手機設定的問題

不管是 phonegap, cordova 或 react native 我都跑不起來，結果發現下列網頁：

- <http://www.technipages.com/galaxy-tab-4-enable-usb-debugging>

在《設定/一般/關於裝置》中找到《版本號碼》，連續點七次之後就會開啟《開發人員模式》..... XD 這到底是哪招阿！有任何一本《開發人員手冊》上會這樣寫的嗎？

然後還要把 USB 偵錯打勾！

- <http://www.samsung.com/us/support/answer/ANS00030524/237280/>

Google 官方的設定說明

- <http://developer.android.com/tools/device.html>

# 這時候

- 如果有一位熟手或老師在旁邊，想必會省去你很多摸索的時間！

好了，現在你應該可以試試

- 自己執行那個 hello 程式了！

# 像這樣

檔案：hello.js

## JavaScript 版

```
console.log('hello 你好!');
```

執行結果

```
$ node hello.js  
hello 你好!!
```

檔案：hello.c

## C 語言版

```
#include <stdio.h>  
  
int main() {  
    printf("hello 你好! \n");  
}
```

執行結果

```
D:\code>gcc hello.c -o hello  
D:\code>hello  
hello 你好!
```

# 如果您能夠正確執行 hello

- 那麼恭喜你！
- 踏出了成功的第一步！

# 剩下的

- 就是程式怎麼寫的問題了！

# 就像前面所說的

- 通常每個語言都會有這些語法
  - 變數怎麼宣告
  - if 語法怎麼寫
  - 迴圈語法怎麼寫
  - 陣列（和字典）的存取如何做
  - 函數如何宣告和使用



# 只要學會了這些語法

- 還有其用途
- 那麼就可以開始寫程式了！

# 學會寫 Hello 之後

- 讓我們來看看，到底程式是甚麼？

# 所謂的程式

- 是一連串要求電腦做事的指令動作。
- 這比較像算術，但不像數學。

# 舉例而言

- 如果你看到

$x=3;$

$y=5;$

$x=y;$

這樣幾個指令時。

# 這代表著

`x=3;` // 執行完這行後 `x` 會是 3

`y=5;` // 執行完這行後 `y` 會是 5

`x=y;` // 執行完這行後 `x` 會是 5

因為 `y` 把 5 這個值丟給 `x`

其中的 `=` 是指定的意思，

而非數學上的比較或相等。

# 所以、當你看到

`x=5;`

這樣的程式碼時，千萬不要以為他寫錯了，然後說：

`x=x+1;`

`x = x+1`

根本就是不可能的。

因為、這裡的 `=` 是指定，而不是比較或數學的等號。

所以執行完後 `x` 會變成 6。

# 每年、我教大一程式設計的時候

- 一開始都有學生誤會，然後百思不解的問我。
- 其實這些學生沒什麼錯，只是它們用數學的等號想法來理解程式裡面的  $=$  符號，所以才會無法理解。

# 事實上、在很多程式語言中

- 都會用一個等號 `=` 代表《指定》，因為這個運算太常用了。
- 然後用兩個等號 `==` 代表《判斷》，這個符號才比較像是數學上的那個相等。



所以、您應該要能判斷  
下列程式會印出甚麼？

```
a = 3;  
b = 5;                                JavaScript 版  
  
if (a==b)  
    console.log("a 等於 b");  
else  
    console.log("a 不等於 b");
```

答案是印出 《a 不等於 b》

# 有了這樣的基本概念之後

- 我們就可以來看看基本的程式該怎麼寫了！

# 我很喜歡用 $1+..+10$

## 當範例說明程式的用途

檔案：wsum.js

### JavaScript 版

```
sum=0;
i=1;
while (i<=10) {
    sum = sum + i;
    console.log("i=", i, " sum=", sum);
    i = i + 1;
}
```

檔案：wsum.c

### C 語言版

```
#include <stdio.h>

int main() {
    int i=1, sum=0;
    while (i<=10) {
        sum = sum + i;
        printf("i=%d sum=%d\n", i, sum);
        i = i + 1;
    }
}
```

# 您可以看到

JavaScript(js) 和 C 兩者的版本，其實結構差不多！

但是 C 語言必須要宣告型態（int 是整數）與主程式 main，比較冗長

檔案：wsum.js

## JavaScript 版

```
sum=0;
i=1;
while (i<=10) {
    sum = sum + i;
    console.log("i=", i, " sum=", sum);
    i = i + 1;
}
```

檔案：wsum.c

## C 語言版

```
#include <stdio.h>

int main() {
    int i=1, sum=0;
    while (i<=10) {
        sum = sum + i;
        printf("i=%d sum=%d\n", i, sum);
        i = i + 1;
    }
}
```

# 這是因為

- JavaScript 是弱型態語言，不需要宣告變數型態。
- 但 C 語言是強型態語言，需要宣告變數型態。

# 另外、兩者在執行過程上也有差別

- JavaScript 是直譯式語言，不需要編譯就能執行
- C 語言是編譯式語言，需要編譯才能執行。

執行結果：

JavaScript 版

```
D:\jsbook>node wsum.js
```

```
i= 1  sum= 1  
i= 2  sum= 3  
i= 3  sum= 6  
i= 4  sum= 10  
i= 5  sum= 15  
i= 6  sum= 21  
i= 7  sum= 28  
i= 8  sum= 36  
i= 9  sum= 45  
i= 10 sum= 55
```

```
D:\Dropbox\cccw\db\c\code>gcc wsum.c -o wsum
```

```
D:\Dropbox\cccw\db\c\code>wsum
```

```
i=1 sum=1  
i=2 sum=3  
i=3 sum=6  
i=4 sum=10  
i=5 sum=15  
i=6 sum=21  
i=7 sum=28  
i=8 sum=36  
i=9 sum=45  
i=10 sum=55
```

C 語言版

不過、JavaScript 和 C 兩者所使用的

- if, for, while 語法都非常類似
- 指定 = 與比較 ==, !=, >=, <=, >, <  
還有 +, -, \*, /, %, &, |, !, &&, ||  
等運算，還有 +=, -=, \*=, /= 這些運算  
都是一樣的。

# 您可以看到下列用 for 迴圈寫的加總版本，JS 和 C 也很像

這是因為 JavaScript (JS) 學 Java 的語法，Java 學 C 的語法，大家學來學去，最後都採用了 C 語言所提出來的那些語法。

檔案：sum.js

## JavaScript 版

```
sum=0;
for (i=1;i<=10;i++) {
    sum = sum + i;
    console.log("i=", i, " sum=", sum);
}
```

檔案：sum.c

## C 語言版

```
#include <stdio.h>

int main() {
    int i, sum=0;
    for (i=1;i<=10;i++) {
        sum = sum + i;
        printf("i=%d sum=%d\n", i, sum);
    }
}
```



# 學會使用迴圈之後

接下來可以學學函數的寫法

檔案: fsum.js

## JavaScript 版

```
function sum(n) {  
  s=0;  
  for (i=1; i<=n; i++) {  
    s = s+i;  
  }  
  return s;  
}  
  
sum10 = sum(10);  
console.log("1+...+10="+sum10);
```

執行結果:

```
nqr-192-168-61-142:code mac020$ node fsum.js  
1+...+10=55
```

檔案: fsum.c

## C 語言版

```
#include <stdio.h>  
  
int sum(n) {  
  int i, s=0;  
  for (i=1; i<=n; i++) {  
    s = s+i;  
  }  
  return s;  
}  
  
int main() {  
  int sum10 = sum(10);  
  printf("1+...+10=%d\n", sum10);  
}
```

```
D:\Dropbox\cccw\db\c\code>gcc fsum.c -o fsum  
D:\Dropbox\cccw\db\c\code>fsum  
1+...+10=55
```

# 必須注意的是

- 函數的參數是根據位置傳入的，而不是根據名字。

```
function max(a, b) {  
    if (a > b)  
        return a;  
    else  
        return b;  
}  
  
m = max(9, 5);  
console.log("max(9, 5)="+m);
```

所以在左邊的程式中，9 會傳給 a, 5 會傳給 b。

就算你故意用以下相反的命名方式，也不會影響《根據位置傳入的事實》。

```
b=9; a=5;  
max(b,a)
```

結果仍然是

(b 裡面的) 9 會傳給 a,  
(a 裡面的) 5 會傳給 b.

並不會因此而顛倒過來！

# 學會函數之後

- 你寫程式才能夠重複利用
- 不用相同的程式寫很多次
- 這種方法稱為模組化！

# 剛開始寫程式的人

- 常常沒學會利用函數
- 就開始用剪貼的方式
- 不斷複製程式！

# 這樣做雖然也能達到目的

- 但是程式會《愈來愈長、愈來愈亂、愈來愈複雜》
- 結果就是，當問題稍微更複雜一點，寫的人就陷入《一直改一直改一直改》的困境。

# 為甚麼會一直改呢？

- 因為只要有任何一個地方要改，就要改  $n$  次。（因為你已經複製了  $n$  次）
- 例如要算下列結果，結果用了三個迴圈，長得都很像
  - $s_{10}=1+\cdots+10$
  - $s_{100}=1+\cdots+100$
  - $s_{1000}=1+\cdots+1000$

# 這樣的話

- 絕對會陷入軟體工程的災難！

# 所以

- 函數很重要！
- 模組化很重要！
- 這就是軟體工程的第一步！



# 會了迴圈與函數之後

- 還有一個很簡單的東西，我們忘了講
- 那就是 `if` 條件判斷

# 舉例而言、以下程式 `max(a, b)` 可以取兩者的大值

檔案：max.js

## JavaScript 版

```
function max(a, b) {  
  if (a > b)  
    return a;  
  else  
    return b;  
}  
  
m = max(9, 5);  
console.log("max(9, 5)=" + m);
```

執行結果：

```
$ node max.js  
max(9, 5)=9
```

檔案：max.c

## C 語言版

```
#include <stdio.h>  
  
int max(int a, int b) {  
  if (a > b)  
    return a;  
  else  
    return b;  
}  
  
int main() {  
  int m = max(9, 5);  
  printf("max(9, 5)=%d\n", m);  
}  
  
D:\Dropbox\cccw\db\c\code>gcc max.c -o max
```

```
D:\Dropbox\cccw\db\c\code>max  
max(9, 5)=9
```

# 接著就可以學習陣列的用法

檔案：array.js

## JavaScript 版

```
var a=[1,6,2,5,3,6,1];

for (i=0;i<a.length;i++) {
    console.log("a[%d]=%d", i, a[i]);
}
```

執行結果

```
D:\jsbook>node array.js
a[0]=1
a[1]=6
a[2]=2
a[3]=5
a[4]=3
a[5]=6
a[6]=1
```

檔案：array.c

## C 語言版

```
#include <stdio.h>

int main() {
    int i;
    int a[]={1,6,2,5,3,6,1};
    for (i=0;i<7;i++) {
        printf("a[%d]=%d\n", i, a[i]);
    }
}
```

執行結果

```
D:\Dropbox\cccw\db\c\code>gcc array.c -o array

D:\Dropbox\cccw\db\c\code>array
a[0]=1
a[1]=6
```

# 然後學學字串的用法

× 陳鍾誠 / 教科書 / JavaScript 語言

## JavaScript 操作 字串運算

```
> x = "hello"  
'hello'  
> x.length  
5  
> x[3]  
'l'  
> x[2]  
'l'  
> x[1]  
'e'  
> x[0]  
'h'  
> x[5]  
undefined  
> x[4]  
'o'
```

檔案：string.c

## C 語言版

```
#include <stdio.h>  
#include <string.h>  
  
int main() {  
    int i;  
    char s[]="hello!";  
    for (i=0;i<strlen(s)+1;i++) {  
        printf("s[%d]=%c ascii=%d\n", i, s[i], s[i]);  
    }  
}
```

D:\Dropbox\cccw\db\c\code>gcc string.c -o string

D:\Dropbox\cccw\db\c\code>string

s[0]=h ascii=104  
s[1]=e ascii=101  
s[2]=l ascii=108  
s[3]=l ascii=108  
s[4]=o ascii=111  
s[5]=! ascii=33  
s[6]=

# 學會了上述這些語法之後

- 其實您就可以寫出很多程式了，像是
  - 計算  $f(a, n) = a^n$
  - 把一個陣列或字串反轉
  - 印出九九乘法表
  - ...

# 還有、很多老師喜歡讓學生

- 練習印三角形，像是

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

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

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

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

- 甚至是列印菱形，這都是為了練習迴圈與判斷的概念而已。

# 你可能會覺得很無聊

- 但這些練習其實就像小學要你一直算《加減乘除》是一樣的道理。
- 如果你真的確定自己會了，那跳過也是可以的！

# 不過、要小心一件事

- 很多人其實只是《覺得自己會了》，但是真的要寫卻寫不出來。
- 這種情況還是多練幾題會好一點。



# 舉例而言

- 像是《判斷質數、向量與矩陣的加減乘除、複數加減乘除、字串反轉、九九乘法表》之類的，都是不錯的練習題。

當您把這些基本功練得差不多的時候

- 就可以學一些進階的語法和技巧

# 像是

- 回呼 callback
- 遞迴 recursive
- 物件 object

以下讓我們

- 很快地看一下這些技巧

# 所謂的回呼 callback

- 就是將函數當參數傳入後，在適當時機被呼叫的那種情況。
- 舉例而言，如果有一個微分函數  $df(f, x)$ ，其中的  $f$  是待微分函數，那麼當我們呼叫  $df(\sin, \pi/4)$  的時候，就可以利用傳進去的  $\sin$  函數，來計算該函數在  $\pi/4$  的導數。

# 以下就是用 callback 作微分的範例

```
function df(f, x) {  
    var dx = 0.001;  
    var dy = f(x+dx) - f(x);  
    return dy/dx;  
}
```

```
function square(x) {  
    return x*x;  
}
```

```
console.log('df(x^2,2)=' + df(square, 2));  
console.log('df(x^2,2)=' + df(function(x) { return x*x; }, 2));  
console.log('df(sin(x/4),pi/4)=' + df(Math.sin, 3.14159/4));
```

## JavaScript 版

```
D:\Dropbox\cccweb\db\js\code>node df.js
```

```
df(x^2,2)=4.0009999999999699
```

```
df(x^2,2)=4.0009999999999699
```

```
df(sin(x/4),pi/4)=0.7067535793015001
```

```
#include <stdio.h>  
#include <math.h>
```

## C 語言版

```
double dx = 0.001;
```

```
double df(double (*f)(double), double x) {  
    double dy = f(x+dx) - f(x);  
    return dy/dx;  
}
```

```
double square(double x) {  
    return x*x;  
}
```

```
int main() {  
    printf("df(x^2,2)=%f\n", df(square, 2.0));  
    printf("df(sin(x),pi/4)=%f\n", df(sin, 3.14159/4));  
}
```

```
D:\Dropbox\cccweb\db\c\code>gcc df.c -o df
```

```
D:\Dropbox\cccweb\db\c\code>df
```

```
df(x^2,2)=4.001000
```

```
df(sin(x/4),pi/4)=0.706754
```

# 當然也可以用 callback 做積分

## JavaScript 版

```
function integral(f, a, b) {  
  var dx = 0.001;  
  var area = 0.0;  
  for (var x=a; x<b; x=x+dx) {  
    area = area + f(x)*dx;  
  }  
  return area;  
}  
  
function square(x) {  
  return x*x;  
}  
  
console.log('integral(x^2, 0, 1)=' + integral(square, 0, 1));  
console.log('integral(sin(x), 0, pi)=' + integral(Math.sin, 0, 3.14159));
```

```
D:\Dropbox\cccweb\db\js\code>node integral.js  
integral(x^2, 0, 1)=0.33283350000000095  
integral(sin(x), 0, pi)=1.9999999540411524
```

## C 語言版

```
#include <math.h>  
#define dx 0.001  
  
double integral(double (*f)(double), double a, double b) {  
  double x, area = 0.0;  
  for (x=a; x<b; x=x+dx) {  
    area = area + f(x)*dx;  
  }  
  return area;  
}  
  
double square(double x) {  
  return x*x;  
}  
  
int main() {  
  printf("integral(x^2, 0, 1)=%f\n", integral(square, 0, 1));  
  printf("integral(sin(x), 0, pi)=%f\n", integral(sin, 0, 3.14159));  
}
```

```
D:\Dropbox\cccweb\db\c\code>gcc integral.c -o integral
```

```
D:\Dropbox\cccweb\db\c\code>integral  
integral(x^2, 0, 1)=0.332834  
integral(sin(x), 0, pi)=2.000000
```

# 這種 callback 技巧

- 在 JavaScript 和 C 當中都很常用
- JavaScript 常用在  
非阻斷式 (non-blocking) 呼叫上
- C 則在驅動程式的註冊與呼叫時常被使用。



# 接著讓我們看看

- 遞迴 recursive 這個技術

# 所謂的遞迴

- 就是一個函數《自己呼叫自己》

```
function f(...) {  
  if (...) return;  
  f(...);  
}
```

← 函數 f 的裡面還呼叫了 f()

# 舉例而言

- 假如我們要計算  $s(n)=1+\dots+n$  這個函數
- 那麼我們可以將  $s(n)$  改寫成遞迴式如下：

$$s(n) = 1+\dots+(n-1)+n = s(n-1)+n$$

$$s(1) = 1$$

於是我們可以根據  
遞迴式寫出下列程式

$$s(n) = s(n-1) + n$$
$$s(1) = 1$$

```
function s(n) {  
    if (n==1) return 1;  
    var sn = s(n-1)+n;  
    return sn;  
}
```

JavaScript 版

```
console.log("s(10)=%d", s(10));
```

```
nqu-192-168-61-142:code mac020$ node s10  
s(10)=55
```

```
#include <stdio.h>
```

```
int s(int n) {  
    if (n==1) return 1;  
    int sn = s(n-1)+n;  
    return sn;  
}
```

C 語言版

```
int main() {  
    printf("s(10)=%d", s(10));  
}
```

```
D:\Dropbox\cccw\db\c\code>gcc s10.c -o s10
```

```
D:\Dropbox\cccw\db\c\code>s10
```

```
s(10)=55
```

# 如果你在中間加上印出的動作 就可以觀察到執行過程

```
function s(n) {  
    if (n==1) return 1;  
    var sn = s(n-1)+n;  
    console.log("s(%d)=%d", n, sn);  
    return sn;  
}
```

JavaScript 版

```
console.log("s(10)=%d", s(10));
```

```
$ node sum_recursive  
s(2)=3  
s(3)=6  
s(4)=10  
s(5)=15  
s(6)=21  
s(7)=28  
s(8)=36  
s(9)=45  
s(10)=55  
s(10)=55
```

輸出結果

# 如果你看懂上述的遞迴寫法

- 那麼應該也可以寫出計算下列遞迴式的程式

$$f(n)=f(n-1)+f(n-2)$$

$$f(1)=1$$

$$f(0)=0$$

費氏數列的前幾項為 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, .....

您可以試著寫寫看！

最後、我們來看物件的概念



# 所謂的物件

- 基本上是把《函數》和《資料》封裝起來，這個封裝結構就稱為物件。

# 如果一個物件只有資料，沒有函數

- 那我們可以稱之為《結構》。

```
var dict={  
  name:"john",  
  age:30  
}; JavaScript 版
```

```
typedef struct {  
    char *name;  
    int age;  
} People;    C 語言版  
  
People peoples[] = {  
    { .name="john", .age=20},  
    { .name="mary", .age=30},  
    { .name="george", .age=40}  
};
```

# 但是如果結構中加上了函數 就會形成物件

```
var circle = {  
  r:3,  
  area:function() {  
    return 3.14*this.r*this.r;  
  }  
}  
  
console.log("circle.r=%d", circle.r);  
  
console.log("circle.area()=%d", circle.area());
```

JavaScript 版

```
NQU-192-168-60-101:ccc csienqu$ node circle.js  
circle.r=3  
circle.area()=28.259999999999998
```

C 語言並不支援物件導向語法，但可以實作物件導向概念，不過會有點複雜，在此我們就不舉例了。

# 在瞭解了這些基本語法之後

- 讓我們用一個範例作為總結
- 那就是一個《逐字翻譯系統》
- 其使用方法如下

```
$ node mt a dog chase a cat  
[ '一隻', '狗', '追', '一隻', '貓' ]
```

JavaScript 版執行結果

```
$ gcc mt.c -std=c99 -o mt
```

```
$ ./mt a dog chase a cat  
一隻 狗 追 一隻 貓
```

C 語言版執行結果

# 以下是《逐字翻譯系統》的程式碼

```
var e2c = { dog:"狗", cat:"貓", a: "一隻",  
chase:"追", eat:"吃" };
```

JavaScript 版

```
function mt(e) {  
  var c = [];  
  for (i in e) {  
    var eword = e[i];  
    var cword = e2c[eword];  
    c.push(cword);  
  }  
  return c;  
}
```

```
var c = mt(process.argv.slice(2));  
console.log(c);
```

```
#include <stdio.h>  
#include <string.h>
```

C 語言版

```
char *e[] = {"dog", "cat", "a", "chase", "eat", NULL};  
char *c[] = {"狗", "貓", "一隻", "追", "吃", NULL};
```

```
void mt(char *words[], int len) {  
  for (int i=0; i<len; i++) {  
    int ei = find(e, words[i]);  
    if (ei < 0)  
      printf(" _ ");  
    else  
      printf(" %s ", c[ei]);  
  }  
}
```

```
int main(int argc, char *argv[]) {  
  mt(&argv[1], argc-1);  
}
```

由於空間不足  
find() 函數  
在此未列出

# 如果、看完這次的 《十分鐘系列》之後

- 您能夠用 JavaScript 或 C 語言，寫出上面的那些範例。
- 而且是不看這個文件的情況下，親自動手寫出來的。
- 那我想您就已經學會那個語言的基礎了。

# 學程式有一件很重要的事情

- 就是一定要自己動手去寫
- 能寫得出來，才算是學會了

# 你可以參考語言的語法

- 或者去查查函式庫
- 但是不應該直接找到那個程式抄下來或複製



# 能夠自己寫出來

- 這樣才算是真正在學程式
- 也才能真正學會程式設計

# 希望

- 這次的十分鐘系列

對您會有所幫助

我們下回見囉！

Bye Bye !