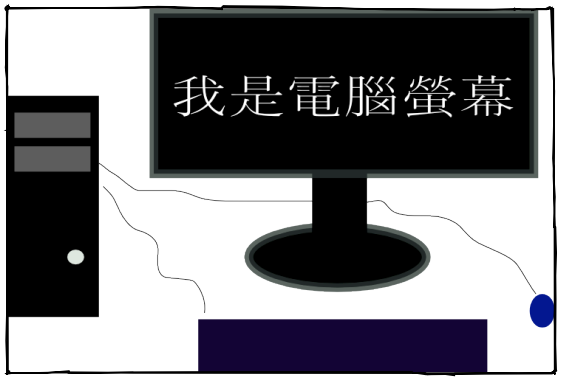
第零篇 踏進電腦這扇門

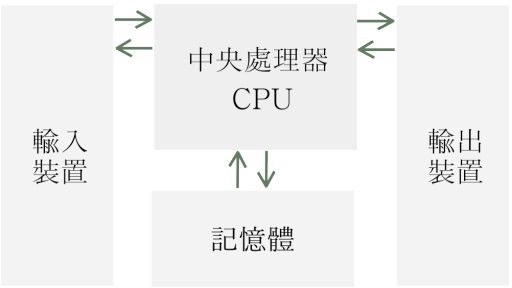
序章 關於電腦

電腦是什麼呢？好像這是一個很簡單的問題，當有人問了這個問題，我們常常把食指指向放在桌上，有螢幕、主機以及滑鼠、鍵盤的個人電腦。對，那是個人電腦，提供我們上網、聊天、打報告的個人電腦。



電腦的英語原文computer，他的意思原本指的是「從事計算的人」，compute的中文意思是「計算」，字尾加er，原本動作的意思轉變成做動作的人，由此，「計算」就成了「計算者」，也可以說是「從事計算的人」。那便是computer的本意，計算的動作從心裡默數，隨著計算量的增加，慢慢變成由輔助工具幫助記憶。然而時代演進，計算的輔助工具可能從遠古時期的排石頭、結繩子，到幾百年前的算盤、齒輪鐘，二十世紀以後，電子技術的突飛猛進，形成今日便利我們生活的電腦。

對，就是電腦，你瞭解什麼是電腦了嗎？簡單來講，他就是一種計算的輔助工具，只不過，今日的電腦化身為各種樣貌，他可能藏在房子裡，也可能躲在人行道上，還可能，也是我們最當然以為放在我們電腦桌上的---------個人電腦。



我們重新來談談今天「電腦」這個詞所代表的涵義。

如圖，這是個相當典型的模式，先以個人電腦為例，輸入裝置有滑鼠、鍵盤，輸出裝置如螢幕，中央處理器及記憶體則安裝在的主機板上，主機板則用一個金屬外殼，也是我們稱之為主機的箱子包裝起來。

其中有一個很重要的東西，就是記憶體，他暫時性的載入儲存所有指揮電腦運作的命令。有如我們運動會的行程表一般，早上八點開幕，來賓致詞，九點賽跑、十點拔河、十一點趣味競賽……等等。記憶體中載入一連串要求電腦做的事，連結網址、播放音樂、方塊遊戲……等等。

由記憶體儲存我們所想要指揮電腦的命令，這提供給我們一個方便，我們可以先寫好這些命令，然後把這些命令輸入電腦，接著電腦就會去做這些事情。事實上，在我們今天所用的作業系統，已經把很多工作做好了，如今天最多人使用的MS-Windows系統，要播放音樂，點擊開始選單，然後點擊Windows Media Player的圖示，就開啟了Windows Media Player這個播放音樂的軟體，當然，就可以播放音樂啦！

作者：張凱慶，Email：kaichingc@gmail.com

我們打開電腦電源，啟動作業系統之後，作業系統已經做了很多事情，如維持開機狀態、圖形介面等等，因此電腦使用者往往只需要開啟軟體，接著按照軟體的操作方式去做他想做的事。所以學電腦有點像是學習作業系統的使用，然後知道做什麼樣的事情就用什麼樣的軟體，更進一步的就是學會使用這些軟體。

但是電腦不是「計算的輔助工具」嗎？怎麼個人電腦做的事情，好像跟計算沒多大的關係呢！

計算的輔助工具

如果我們深入探討個人電腦，將中央處理器，也就是CPU一點一點的放大，我們會發現CPU由非常多微小的電子線路所組成，整體來說這些線路組合成許多不同的部份，如程式計數器、指令暫存器、控制單元、算術邏輯單元等等。仔細一看，這些的確都是用來執行計算的工作。

怎麼說呢？細節有點像是程式計數器記錄程式執行到哪個記憶體位置，然後經由控制單元把記憶體位置的指令載入指令暫存器，依指令解碼後的需求，如需計算則將資料送進算術邏輯單元，或是儲存到另一個記憶體位置，或是……

好麻煩喔！不是嗎？可是因為電子移動的速度相當快，於是電腦在連眨眼都不用的瞬間就完成一個動作，隨著計算量加大，才有可能多花點時間。然而電子線路不是都小到肉眼看不見嗎？我們學電腦需要徹底了解這些繁瑣又無趣的線路嗎？

現在大可不必！不過早期電腦與人之間的溝通確實直接由線路著手，藉由線路的通電與否，1表示通電，0表示不通電，從而衍生出**機器語言**，利用輸入如0001010001101110的指令，控制電路執行工作。

不過，對大多數人來講，0、1的數字顯然不是一種容易習慣的表示方法，所以有人很快的發展出**組合語言**，如下所示。

```
MOV AH, 01
INT 21H
……
```

組合語言相對機器語言比較容易理解，因為組合語言用將0、1的排列用文字來代替。不過由於組合語言僅僅是直接把機器語言翻譯過來，因而利用組合語言跟電腦溝通實際上與機器語言相當類似，對於電腦的許多細節都要有所了解，所以機器語言和組合語言又合稱**低階語言**。

程式語言的觀念

低階語言是程式語言的一種，也是電腦發展初期最早出現的人機溝通方式。所謂的程式類似食譜，或是摺紙、做模型的教學圖示，數學上稱為**演算法**。程式要求電腦為我們去做些事情，這些事情被詳細的一步一步用某種格式儲存在電腦硬體之中。程式也被稱為**軟體**，不過通常我們所說的軟體是指已經包裝好有所特定用途，如繪圖軟體、文書處理軟體等。

程式語言則是用來規範程式的撰寫方法，這樣的語言被稱為**形式語言**，同樣的例子廣泛應用在各種領域，有些是人們習慣後可以直接理解的，如路標、地圖等，另有一些應用如化學式、方程式等。

其實電腦只能執行機器語言，或稱為**機器碼**。組合語言要透過組譯的方式，將組合語言翻譯成機器碼，電腦才能執行。畢竟低階語言不是那麼的親切！很多人仍是希望用近於口語的方式來跟電腦溝通，於是電腦的歷史過程又很快的有人發展出各式各樣的程式語言，這些新的程式語言被稱之為**高階語言**。

高階語言主要分為**編譯**與**直譯**兩種。編譯式的語言有Fortran、COBOL、Pascal、C、C++、Java等，而直譯式的語言有BASIC、Smalltalk、Perl、Python、Ruby等，兩者最大的差別是編

譯式的語言在程式執行前須將**原始碼**完全翻譯成機器碼，直譯式的語言則是透過直譯器，一次翻譯原始碼的一行成機器碼，然後執行。

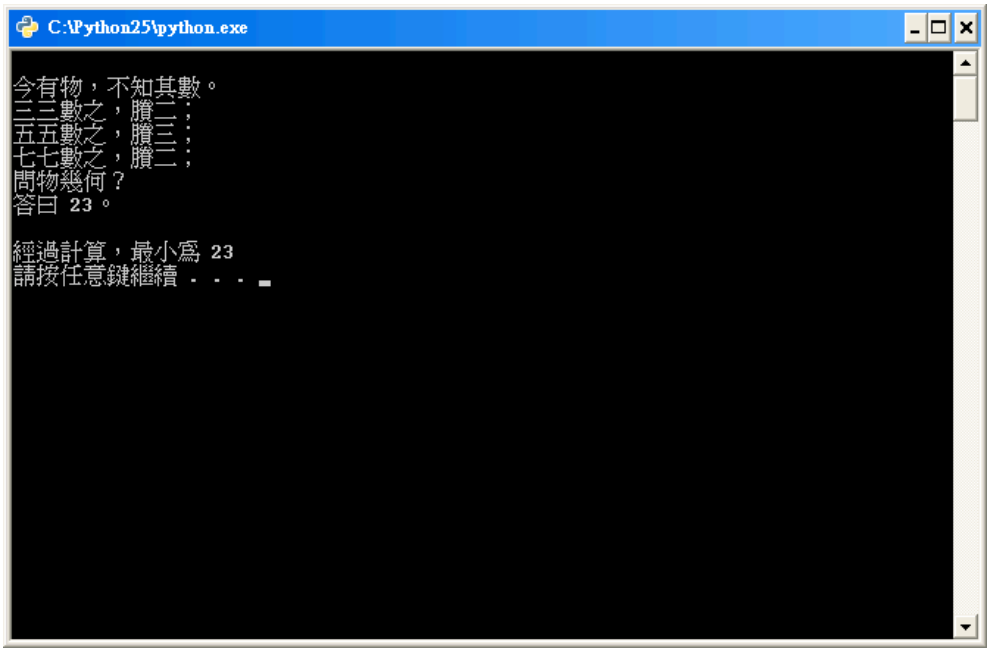
孰優孰劣？各有各個優點及缺點，直譯式語言以往最令人詬病的就是執行速度，正因為依賴直譯器一次一行的翻譯執行，第一行原始碼透過直譯器翻譯成機器碼，再透過直譯器執行，接著第二行……，然後第三行……。比起完全編譯後的執行程式，直接一行接一行的執行機器碼，速度上確實慢了許多。

然而，這種差異在CPU時脈還不快的年代的確很明顯，不過，當個人電腦的CPU的時脈越來越快的時候，CPU的迅速即時多少彌補了直譯式語言的缺點，因而Python、Ruby等搖身一變成為新一代熱門的程式語言。

數學問題

聽說寫程式會用到很多數學，電腦科學又被稱為演算法的科學，進入**程式設計**的這扇門，好像到處都是數學，真的嗎？

「假作真時真亦假，無為有處有還無。」好像說的人多了，真的會是怎麼樣？很多時候焦點都被模糊。如果煩惱的是數學解題技巧或是繁複的計算工作，不用擔心，計算交給電腦去做，解題方面，也許程式的領域可以給我們不同的思考方向。



這是《孫子算經》中的一個問題，也是我們學過整數、因數、倍數、除法後可能練習過的習題，也許我們會想著列出如下的數列。

```
2, 5, 8, 11, 14, 17, 20, 23, 26, 29……
3, 8, 13, 18, 23, 28, 33, 38, 43……
2, 9, 16, 23, 30, 37, 46, 53……
```

從上面所列出的數字，我們可以找出符合題意最小的正整數為23。當然，《孫子算經》有提供解法：「凡三三數之剩一，則置七十；五五數之剩一，則置二十一；七七數之剩一，則置十五；一百六以上，以一百五減之即得。」於是

```
70×2+21×3+15×2＝233 (>106)
233-105＝128 (>106)
128-105＝23
```

當符合的數字越變越大的時候，紙筆計算就顯得很麻煩。程式怎麼算出來的呢？這是個很有趣的問題，我們接下去逐步了解怎麼寫出這個程式後，也許會感到意外，原來把加減乘除交給運算快速的電腦時，數學彷彿簡單了許多！

寫程式會盡是像這樣處理數學問題嗎？不完全是這樣的。程式設計是一種找出問題解決方法的途徑，程式本身的目的就在於解決問題，其中有很大的組成比例，因為需要量化的關係，所以讓人感覺起來寫程式就像在處理數學問題。

我們必須體認問題的本質，從而提出解決方法，然後嘗試去解決，若是結果不能解決或是不能完全解決，再依據情況適當的調整方法。其實，這跟我們所學過的科學方法：觀察→提出假說→實驗，實驗結果如果證實假說則可以進一步的將其擬為學說，基本上這些道理及精神都是相通的。

Python，讓寫程式更簡單

不論編譯或直譯的程式語言都有千萬種，我們為什麼選擇Python呢？以下摘錄Tim Peters的The Zen of Python：

> 美麗優於醜陋，明講好過暗諭。
> 簡潔者為上，複雜者次之，繁澀者為下。
> 平舖善於層疊，勻散勝過稠密；以致輕鬆易讀。
> 特例難免但不可打破原則，務求純淨卻不可不切實際。
> 斷勿使錯誤靜靜流逝，除非有意如此。
> 在模擬兩可之間，拒絕猜測的誘惑。
> 總會有一種明確的寫法，最好也只有一種，
> 但或須細想方可得。
> 凡事雖應三思後行，但坐而言不如起而行。
> 難以解釋的實作方式，必定是壞方法。
> 容易解釋的實作方式，可能是好主意。
> 命名空間讚，吾人多實用。

Note

Zen在佛教中是禪的意思，這篇原文在PEP 20，翻譯取自PyTUG的PotWiki。

「明講」也就是清楚，「平舖」也可以用乾淨來講，因而清楚、簡潔、乾淨這三者為Python語法中最為突出的特色，不但對於程式的撰寫及維護容易，同時形成易學、易讀，這可是相當適合初學者的呢！

Python的發明者Guido van Rossum曾提出一個口號：“There is only one way to do it.”，中文意思是：「做一件事情只有一個方法。」固然我們知道一個問題的解決方法往往不只一個，各種不同的方法可能會帶來不同的影響與成效，然而當我們把目光焦點集中在程式語法的表達上時，或許，當語法描述一種方法只有一種寫法時，某種程度上來講，我們更能洞察問題的本質。

Python，讓寫程式更簡單！