

Week 2 入門概論

Question 1: 什麼是運算思維？什麼是程式設計？二者有什麼關係？又，程式設計與運算思維重要嗎？對你重要嗎？

2-1 電腦科學

Motivation

無論是生活還是工作，人類越來越倚重電腦的使用，現代公民需要對電腦科學有更多的認識！

2-1-1 什麼是電腦科學

Computer science is fundamentally about computational problem solving.

- Computer science is about programming computers.
 - Programming languages and computers are tools.
 - Programming is a primary activity of computer science.
 - Computer science solves problems by the use of computation.

~ 運算思維與程式設計是學習電腦科學的核心。

電腦科學是以程式語言和電腦為工具，我們使用程式語言這個工具寫出指令，讓電腦這個工具依照指令工作。我們有解決問題的想法，也就是運算思維，才能透過程式設計用電腦幫助我們解決問題。也就是說，當一個問題有了解決的步驟與方法，透過程式語言來展現運算思維，用指令將解決問題的方法設計出來，讓電腦進行自動化運算解決我們的問題。

學習運算思維與程式設計，讓電腦來幫助我們解決問題，是現代人必備的重要技能。

~ 課程安排

這門課程將先介紹什麼是運算思維，並且認識電腦這個工具的軟硬體，然後以Python程式語言為工具，一步步地用Python學運算思維、讓電腦幫助我們解決問

題。最後，從資料科學的角度，看看大數據、人工智慧的發展與成就，並且實做聊天機器人。

2-1-2 運算思維

The Essence of Computational Problem Solving

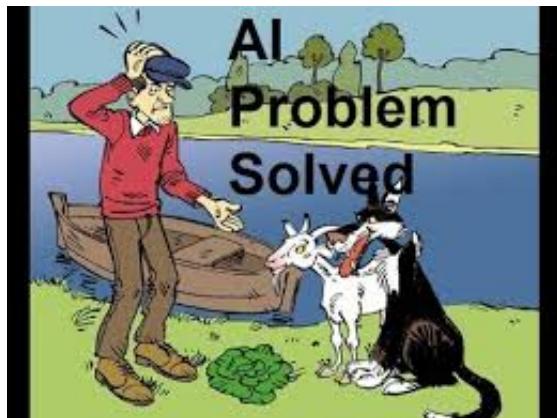
In order to solve a problem computationally, two things are needed:

- **a representation** that captures all the relevant aspects of the problem.(omits all the irrelevant details)
- **an algorithm** that solves the problem systematically by a series of steps.

要使用電腦來解決問題之前，需要：(1)去蕪存菁，把問題是什麼說清楚，(2)知道有系統解決問題的系列步驟方法。

實例：渡河之謎

Example: Man, Cabbage, Goat, Wolf Problem



- Description
 - You are on the east side of a river with a boat, a cabbage, a goat, and a wolf.
 - Your task is to get everything to the other side.
 - How do you solve the problem?
- Restrictions:
 - only you can handle the boat

- when you're in the boat, there is only space for one more item
- you can't leave the goat alone with the wolf because the wolf will eat the goat.
- you cannot leave the goat alone with the cabbage because the goat will eat the cabbage.

隨堂討論：請同學先把渡河問題說清楚，然後提出解決之道！！！

~運算思維是從運算角度解決問題的方法，包含下面四個基石：

運算思維：是一種解決問題的方法

問題拆解

將複雜問題分解成小問題

模式辨識

從(具代表性)資訊中找出規律

運算思維

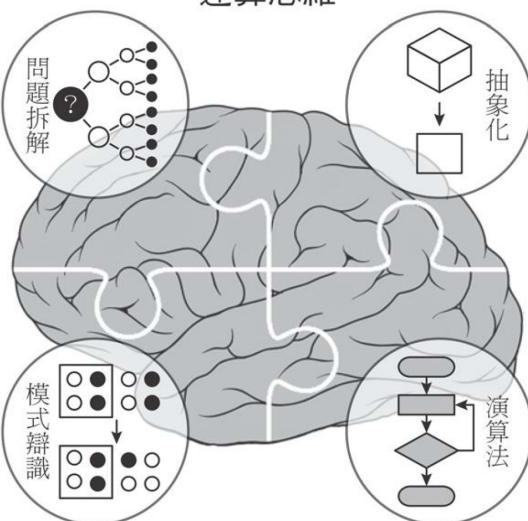


圖 1-1

抽象化

將實體資訊(去蕪存菁)轉化成計算機可以處理的形式

演算法

設計可以處理問題的方法

例：如何做一個Pizza、南瓜派。

(1) 問題拆解(Decomposition)：將複雜的問題拆解成較小的問題。

一個真實的問題常常相當複雜，也不容易表達清楚，更難直接處理。因此，拆解克服(Divide and Conquer)，將問題適當拆解常常是解決問題的第一步。拆解之後應該會讓整個工作更清楚，每部分的工作也明顯地比未拆解之前簡單，可以更有效率的處理，也方便與他人分工合作。

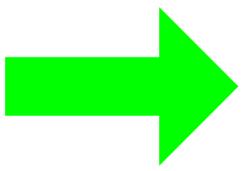
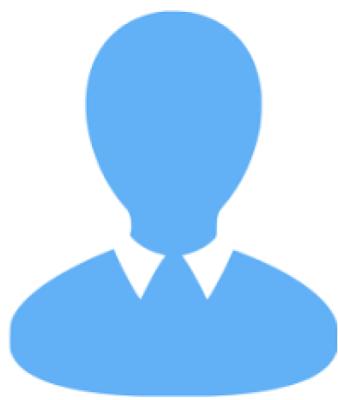
(2) 抽象化(Abstract)：去蕪存菁，決定關鍵步驟與資料，描述問題。

抽象化就是去蕪存菁，將實體資訊轉換成計算機可以處理的形式，包括程序抽象化與資料抽象化。

例如從地圖上，說明如何坐公車/走路從世新到政大。

例如用某些資料代表一個學生。大學資訊系統有學生的姓名、性別、學號、系級等資料，來代表某位學生，這就是對學生做「資料抽象化」的結果。

抽象化



客戶

姓名：John Smith

性別：M

婚姻狀況：M

子女個數：2

信用卡數目：4

年收入：120K (USD)

：

(3)模式辨識(Pattern recognition)：找出型態的一般性通則

找出最主要導致此模式的原則或因素。一般是從具有代表性資訊中找出規律，例如用敘述統計表示特性，南部民眾說台灣話的比例高於北部，台北人大多使用國語交談。擁有10張以上信用卡者，欠債不還款比例較高。

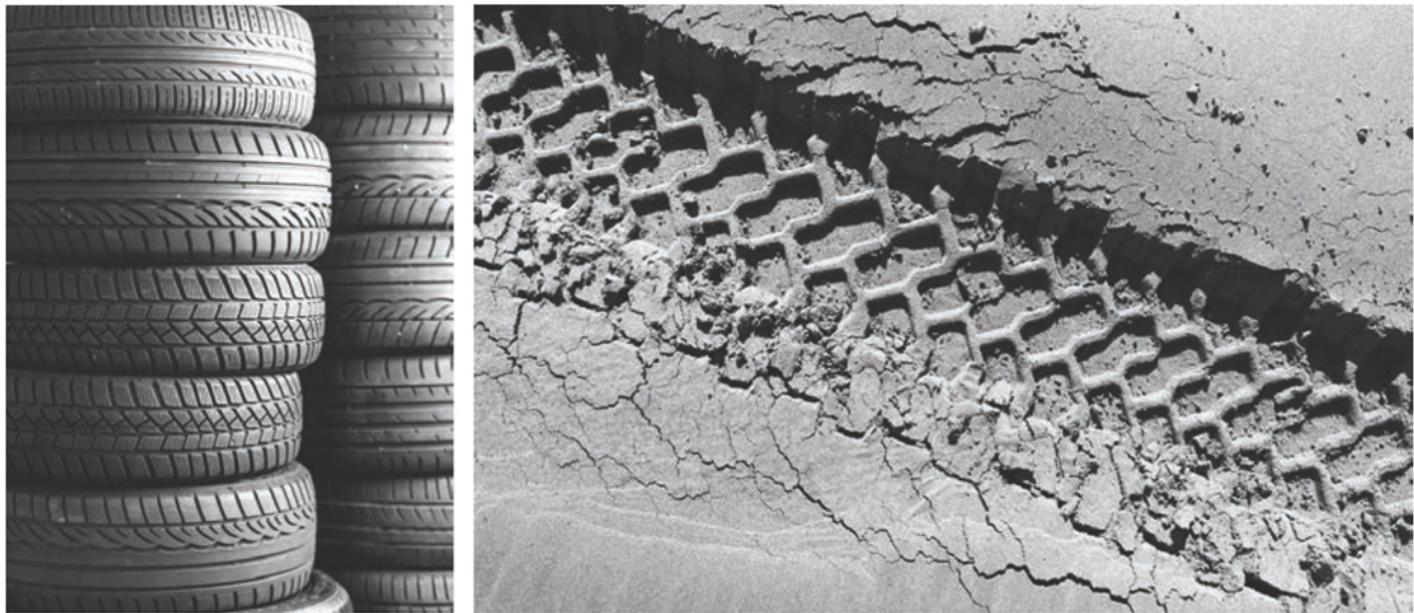


圖 1-20 (圖片來源：Pixabay)

(4)演算法(Algorithms)：設計出能夠解決此問題的流程步驟。

演算法是透過一連串的操作步驟（指令），可以解決某一個特定的問題方法。

～渡河之謎 Answer: Computational Thinking

Step 1: Divide and Conquer

Step 2: Abstraction

- Only the relevant aspects of the problem need to be represented
- All the irrelevant details can be omitted.
 - In this case, is the color of the boat relevant? The width of the river? The name of the man? No.
 - The only relevant information is where each item is at each step.

Step 3: Pattern recognition

What would be an appropriate representation for this problem?

- The state of the problem in this case is the collective location of each item.
 - the start state of the problem can be represented as follows.
 - the symbol E denotes that each corresponding object is on the east side of

the river.

man cabbage goat wolf
[E, E, E, E]

Step 4: Algorithms

- A solution to this problem is a sequence of steps that converts the initial state, [E, E, E, E] in which all objects are on the east side of the river, to the goal state , [W, W, W, W] in which all objects are on the west side of the river.
 - Each step corresponds to the man rowing a particular object across the river (or the man rowing alone).
 - The task is to develop or find an existing algorithm for computationally solving the problem using this representation.

~什麼是運算思維

<https://www.youtube.com/watch?v=ApvNdzyvGFA>

2-1-3 程式設計

Algorithms and computers are a perfect match!

You learn programming language so you can express your ideas to accomplish tasks with computers.

演算法是運算思維的具體展現。有了演算法表示已經針對問題提出解決之道，接下來是能用程式寫出相對應的指令，讓電腦來幫助我們處理問題了。

- 如果沒有問題意識，不知道要解決什麼問題，無須談運算思維與程式設計。**要有問題意識需要專業知識與經驗，以及好奇的心**。
- 如果**知道要問什麼問題**，但**不能以專業知識、普通常識與經驗提出解決之法**，也**無須談運算思維與程式設計**。不同的問題，特性不同，解決方法亦不同。
- 有了問題意識，知道解決問題的方法，剩下的就需要靠程式設計來讓電腦為我們

工作。

~ 程式設計是指設計與編寫電腦語言，以表達某問題之解決方案的藝術。

- 程式設計大部分的工作都花在尋找和改進此解決方案。
- 只有透過實際的電腦語言編寫才能完全掌握此能力。

~ 電腦科學領域中解決問題的步驟如下：

- 階段一：分析問題。
- 階段二：描述資料與演算法。
- 階段三：建置程式。
- 階段四：測試與除錯。

The Process of Computational Problem Solving

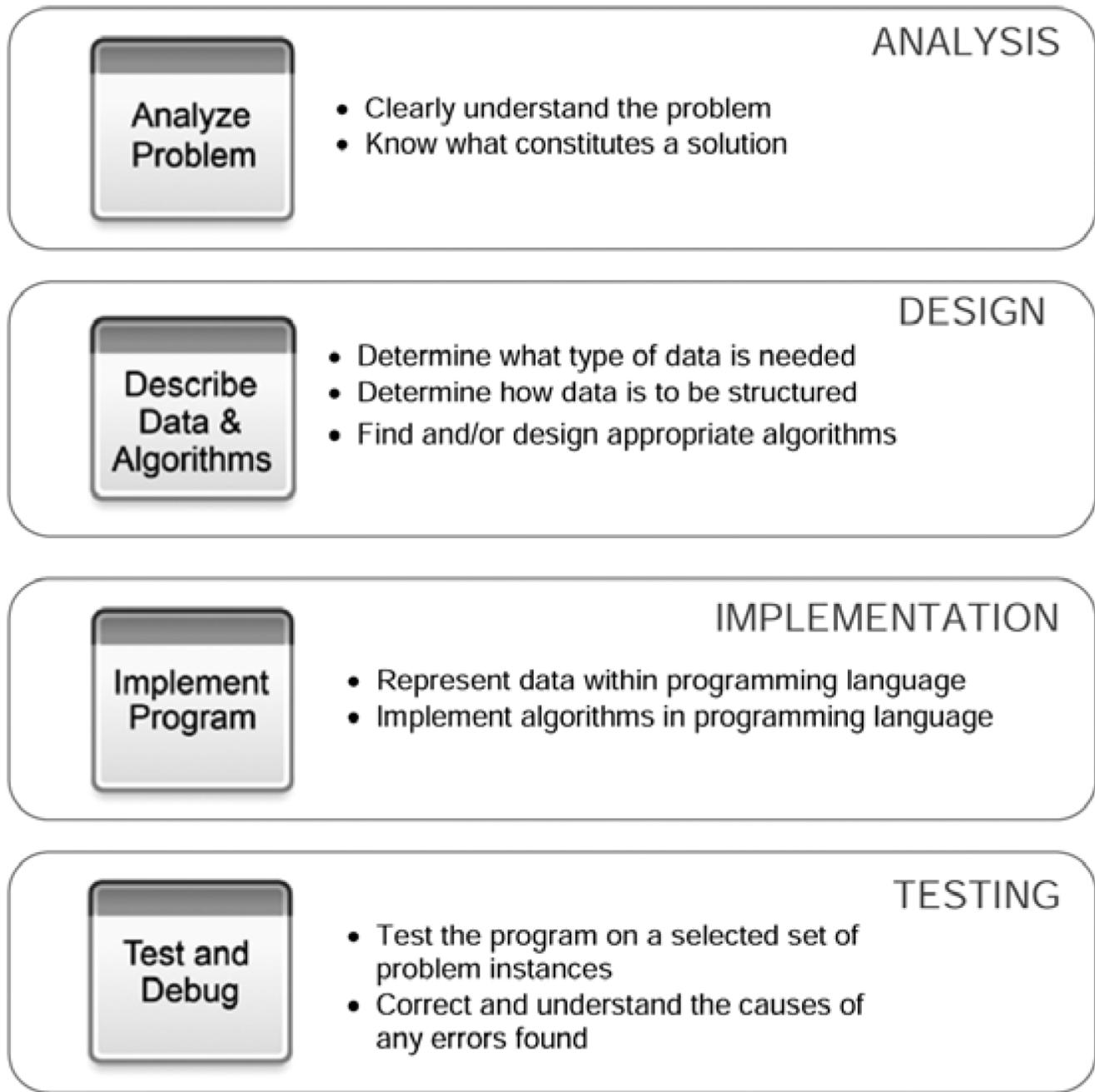


FIGURE 1-22 Process of Computational Problem Solving

(1) Problem Analysis

在寫程式前，應先有運算思維，也就是想清楚問題是什麼，以及如何解決問題，這就是問題分析。

～分析 (analysis)階段的工作重點如下：

定義問題 (本質、範圍與目標)、提出解決方案、評估可行性、蒐集與分析現有的軟

體、訂定軟體的需求規格。

Understanding the Problem

Once a problem is clearly understood, the fundamental computational issues for solving it can be determined. The representation will be straightforward.

Knowing What Constitutes a Solution

Besides clearly understanding a computational problem, one must know what constitutes a solution.

- For some problems, there is only one solution.
- For others, there may be a number (or infinite number) of solutions.

(2) Program Design

IPO模式：輸入、處理與輸出(Input-Process-Output Model)。

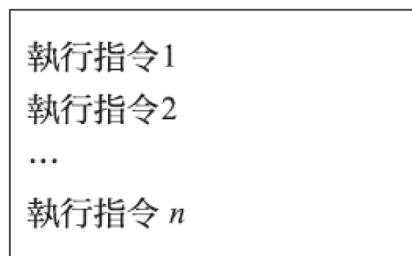
- 輸入 (input)
- 輸出 (output)
- 處理 (process)
 - 明確性 (definiteness)
 - 有效性 (effectiveness)
 - 有限性 (finiteness)

在寫程式前，並先列出所需要的輸入資訊以及可能的輸出資訊，並且使用有意義的變數名稱並善用註解，可以提高程式的可讀性。處理時，則有三種基本的程式設計結構：

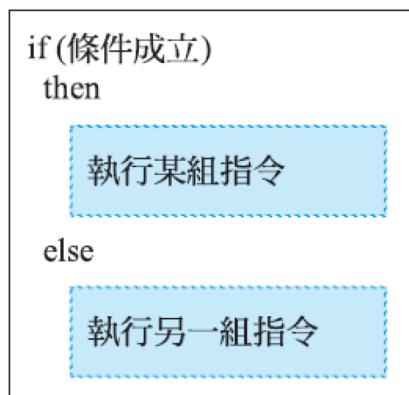
- 序列是指程式指令由上到下依序執行。
- 決策則是指依照條件情況執行特定分支指令。
- 重覆是以迴圈的方式反覆執行某組指令。

演算法通常是由下列三種結構所組成：

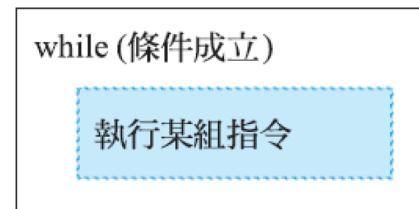
- 序列 (sequence)
 - 決策 (decision)
 - 重覆 (repetition)



(a) 序列結構



(b) 決策結構



(c) 重覆結構

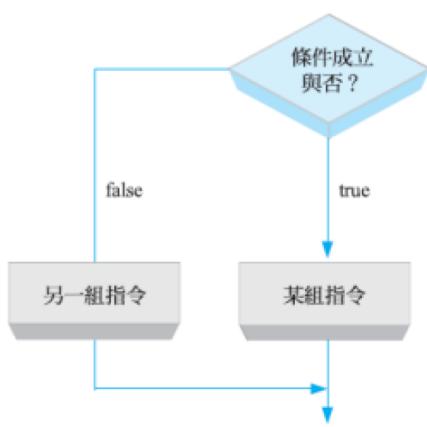


有經驗的程式設計師會繪製流程圖 (flow chart)、撰寫虛擬碼 (pseudocode) 與軟體文件 (document)，其中軟體文件又分為下列幾種：使用者文件 (user document)、系統文件 (system document)、技術文件 (technical document)。

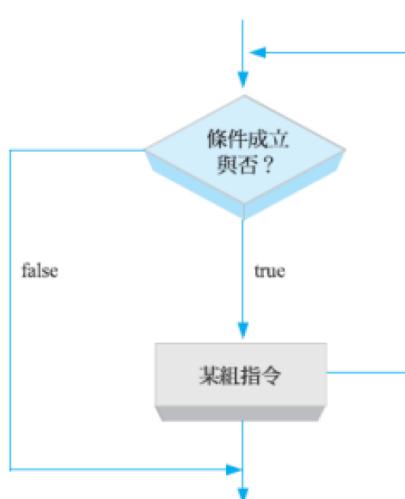
流程圖（flowchart）是以圖形符號表示演算法，下圖是使用流程圖表示序列、決策及重覆等三種結構。



(a) 序列結構



(b) 決策結構



(c) 重覆結構



除了用「視覺化的流程圖」來幫助思考解決問題的步驟之外，也可以使用「文字式的

虛擬碼」來呈現程式設計的邏輯與步驟。

虛擬碼就是用口語文字來撰寫程式。

虛擬碼（pseudocode）是以近似於英文但語法更為精確的方式來描述問題的解法，下圖是使用虛擬碼表示序列、決策及重覆等三種結構。

instruction 1
instruction 2
...
instruction n

if (condition)
then
 one set of instructions
else
 another set of instructions

while (condition)
 one set of instructions
end while

(a) 序列結構

(b) 決策結構

(c) 重覆結構

擬好演算法、寫好虛擬碼之後，就可以開始寫程式了。

Summary

設計 (design) 階段的目的是針對軟體的需求規格發展一套詳細的計畫，該計畫將會在下面的建置階段轉化為程式，包括資訊系統的輸入輸出如檔案與資料庫需求，以及演算法能明確有效地處理問題等需求。

Describing the Data Needed

For the Man, Cabbage, Goat, Wolf problem, a list can be used to represent the correct location (east and west) of the man, cabbage, goat, and wolf as discussed earlier, reproduced below, [W, E, W, E].

The appropriate representation of data is a fundamental aspect of computer science.

Describing the Needed Algorithms

When solving a computational problem, either suitable existing algorithms may be

found or new algorithms must be developed.

For the MCGW problem, there are standard search algorithms that can be used.

Algorithms that work well in general but are not guaranteed to give the correct result for each specific problem are called heuristic algorithms.

(3) Program Implementation

在寫程式(code)時，動手把上述的步驟（演算法）一一寫成程式敘述，並檢查程式語法是否正確，這個動作叫做演算法實作(Algorithmic implementation)。

Summary

建置 (implementation) 階段的目的是將軟體的建置計畫轉化為程式，其工作重點包括專案排程、撰寫程式、建立檔案與資料庫，常見的專案排程工具有甘特圖 (Gantt chart)。

Design decisions provide general details of the data representation and the algorithmic approaches for solving a problem. The details, however, do not specify which programming language to use, or how to implement the program. That is a decision for the implementation phase.

Since we are programming in Python, the implementation needs to be expressed in a syntactically correct and appropriate way, using the instructions and features available in Python.

範例：



「規劃」該怎麼做？



- 先畫「螢幕輸出」



將「螢幕輸出」轉為「程式碼」



Hello! → "Hello!" → print("Hello!")
你好嗎？ → "你好嗎？" → print("你好嗎？")

print("Hello!")
print("你好嗎？")



開始撰寫程式碼...



- 規劃 → 程式碼

```
print( "Hello!" )  
print( "你好嗎？" )
```



註解



- # → Python 的註解符號

- # 單行註解

- """ 多行註解
這是第二行 """



```
# 印出打招呼訊息  
print( "Hello!" )  
print( "你好嗎？" )
```

4. Program Testing

在測試(test)時，你必須驗證你的程式，經由輸入不同的資料，逐一確認結果是否符合預期。

在除錯(debug)時，依據錯誤的情況找出有邏輯錯誤的程式碼，更正處理的步驟（演

算法) 並修改程式碼。

Summary

測試階段的目的是追求軟體的品質保證，其工作重點是針對軟體進行除錯與驗證。

Writing computer programs is difficult and challenging.

- Programming errors are pervasive, persistent and inevitable .
- Software testing is a crucial part of software development.
- Given the inevitability of programming errors, it is important to test a piece of software in a thorough and systematic manner.
- Any changes made in correcting a programming error should be fully understood as to why the changes correct the detected error.



- 「除錯」 = 找出錯誤所在之處

- 兩種「錯誤」

- 語法錯誤 (Syntax Error)
 - 指令不合文法。
 - 可 Google 正確文法並修正之。
- 語意錯誤 (Semantic Error) :
 - 指令合文法，但整體邏輯有誤。
 - 以「除錯器」找出邏輯錯誤所在行數。

- 解決下列困境

- 錯誤訊息看不懂：Google 「錯誤訊息 “意思” 」
- 指令用法不懂：Google 「“ Python” 指令本身 “指令” “用法” 」
- 不知道下一步怎麼做：原始碼 + 你的目的 + 目前困境 + 環境 + 試過方法

第一類常見的錯誤：語法錯誤(Syntax Error)

- 打錯字、不合乎文法
- 解決方法：
 - 電腦會很仔細的告訴你錯在哪裡，看懂錯誤訊息即可修正
 - 不然直接把整個錯誤丟到Google查詢也可以

第二類常見的錯誤：語意錯誤(Semantic Error)

- 邏輯有問題
- 解決方法：
 - 用除錯器找出錯在哪一行
 - 初學者的問題：錯誤訊息看不懂、指令用法不瞭解、不知道下一步做什麼
 - 錯誤訊息：{錯誤訊息} + "意思" -> 請多用Google
 - 指令用法："Python" + {指令名稱} + "指令" + "用法" -> 請多用Google
 - 不知道下一步做什麼：只能問有經驗的人（問老師、上網、上論壇）



- `print("Hello")`
→ SyntaxError: EOL while scanning string literal

{錯誤訊息} + “意思” → Google





「指令用法不瞭解」怎麼辦？

- “python” + {指令名稱} + “指令” + “用法”



「不知道下一步做什麼」怎麼辦？

- 只能問老師，或上論壇問別人了



- Python 版本
- 你的作業系統版本
- 你的目標
- 你目前做到哪一步
- 你嘗試過哪些方法
- 附上原始碼

Summary: 程式設計過程

開發程式前應該多多思考，先瞭解問題找出解決方法，接著寫程式，最後再測試程式是正常運作，若有問題則找出異常摒除錯，重複執行這個流程直到程式通過所有測試為止。讓程式正確執行是第一步，其次才是寫出結構化(structure)、可讀高

(readability)且維護容易的程式。

2-2 電腦系統

電腦系統的組成

一個完整的電腦系統包含硬體與軟體兩部分，前者指的是組成電腦的電子電路及各項設備，而後者指的是告訴電腦做什麼的指令程式。

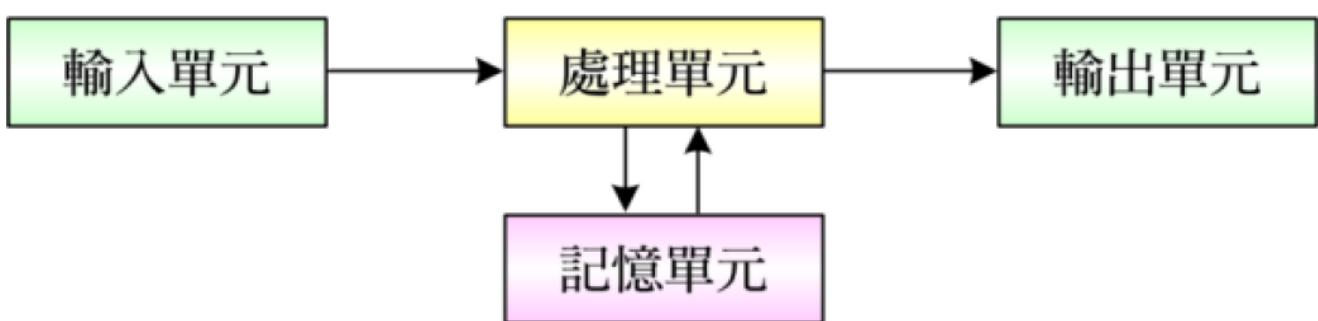
- 電腦硬體
 - 看得到、摸得到的設備
- 電腦軟體
 - 作業系統層:大公司設計負責控制電腦
 - 應用軟體層:為特殊目的而撰寫的程式

2-2-1 電腦硬體

一、四個基本單位

電腦硬體的基本組成包括下列四個單元：

- 輸入單元 (input unit)
- 處理單元 (processing unit)
- 記憶單元 (memory unit)
- 輸出單元 (output unit)



1. 輸入單元(input unit)

用來接收外面的資料，包括文字、圖形、聲音與視訊，然後將這些資料轉換成電腦看得懂的格式，傳送給處理單元做運算。

例如鍵盤、滑鼠、觸控版、數位相機、數位攝影機、掃描器、搖桿。

- 鍵盤：是電腦主要的輸入設備，可以用來接收使用者的指令和外界的資料等，以做為電腦運作的依據。
- 滑鼠：滑鼠也是電腦主要的輸入設備，使用者只要移動滑鼠，螢幕上的指標就會移動到目標的文字或圖示，再透過滑鼠的按鍵，即可將訊息傳遞給電腦。



2. 處理單元 (processing unit)

「中央處理器」(CPU)，負責執行電腦的算術運算與邏輯運算。



3. 記憶單元(memory unit)

用來儲存CPU運算時所需要的資料或程式，以及儲存CPU的運算結果。

記憶單元又分為記憶體和儲存裝置兩種類型。

- 記憶體用來暫時儲存資料，例如暫存器、快取記憶體、主記憶體等。
- 儲存裝置用來長時間儲存資料，例如硬碟、光碟、隨身碟、記憶卡與固定硬碟。



4. 輸出單元 (output unit)

將CPU運算完畢的資料轉換成使用者能夠理解的文字、圖形、聲音與視訊，然後顯示出來。

例如螢幕、印表機、喇叭、投影機等。

- 顯示器：又稱為「螢幕」，是電腦最主要的輸出設備，可以將電腦處理後的結果如文字或影像顯示出來。一般來說，螢幕都有畫面調整扭，可以讓使用者用來調整螢幕的亮度、大小、垂直及水平的中心點等。
- 鍵盤：是電腦主要的輸入設備，可以用來接收使用者的指令和外界的資料等，以做為電腦運作的依據。

- 滑鼠：滑鼠也是電腦主要的輸入設備，使用者只要移動滑鼠，螢幕上的指標就會移動到目標的文字或圖示，再透過滑鼠的按鍵，即可將訊息傳遞給電腦。
- 硬碟機：硬式磁碟機是電腦的工作磁碟，可以用來開機啟動作業系統，及存取主要的應用程式與使用者的資料。硬式磁碟機通常被電腦的主機外殼所蓋住，要拆開來才看的見。硬式磁碟機採用密閉的裝置設計，所以不容易受到污染。
- 光碟機：一般的光碟機，僅能讀取光碟片上的資料，而無法將資料再寫入，也有可讀寫光碟機常見的有MO和CD燒錄器。
- 印表機：常見的印表機種類有點矩陣式、噴墨式和雷射印表機等。印表機是將電腦上的文字和圖形，輸出列印到紙上的一項重要設備。
- 喇叭：喇叭是多媒體電腦中，不可或缺的一項重要設備，它可以將電腦的聲音或音樂效果輸出，讓使用者有身歷其境的感覺效果。



上述的電腦硬體又被稱為「主機」與「週邊設備」。

- 電腦的主機就像人類的大腦一樣，是電腦中最重要的部分。
 - 電腦的「主機」包含中央處理器、記憶體、及介面卡等，放在主機板上，而且外面以外殼保護著。
- 相對於電腦主機以外的硬體，都稱為電腦的「週邊設備」。

- 常見的週邊設備有顯示器、鍵盤、滑鼠、磁碟機、光碟機、印表機、數據機等。

二、計算、儲存與連接三種功用

計算用、儲存用與連接用三種電腦硬體

1. 計算用硬體：CPU、FPU、GPU



- 定義：負責計算的硬體元件



中央處理器 (CPU)
Central Processing Unit



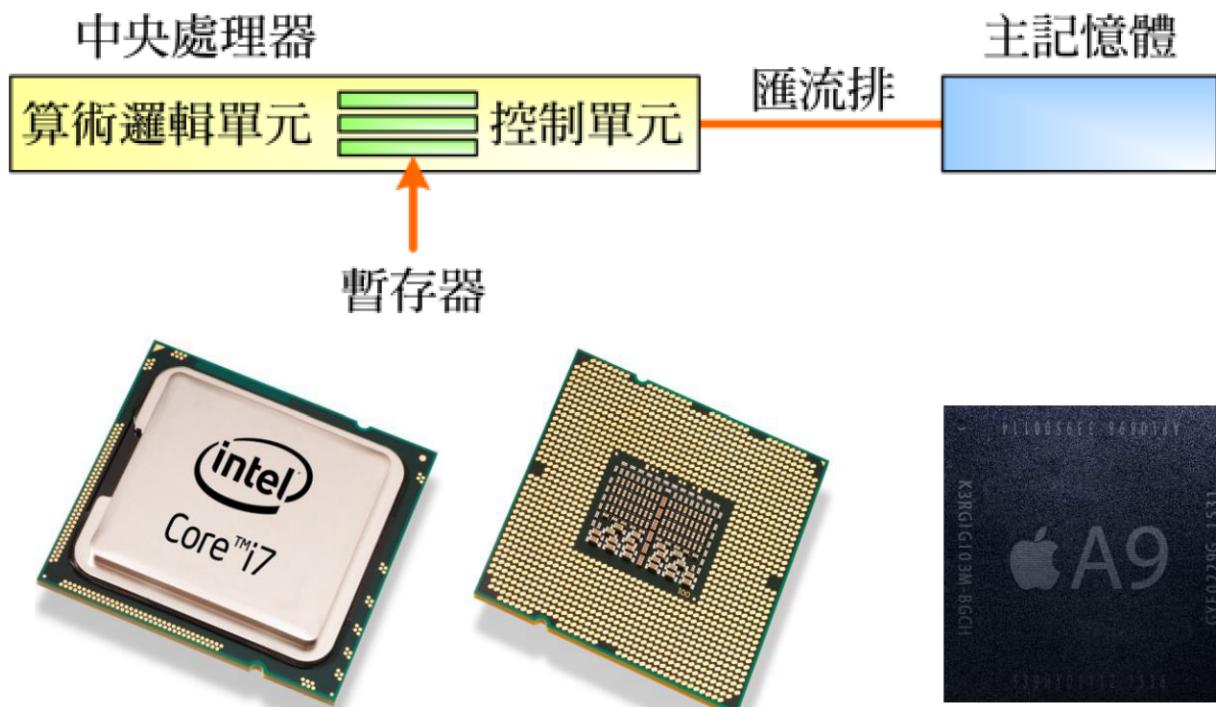
浮點運算器 (FPU)
Floating Point Unit



圖形處理器 (GPU)
Graphic Processing Unit

(1) 中央處理器(Central Processing Unit，CPU)

中央處理器是電腦的主要裝置之一，主要功能是進行算術運算與邏輯運算，由控制單元 (control unit) 、算術邏輯單元 (arithmetic logic unit) 及部分記憶單元的暫存器 (register) 所組成。

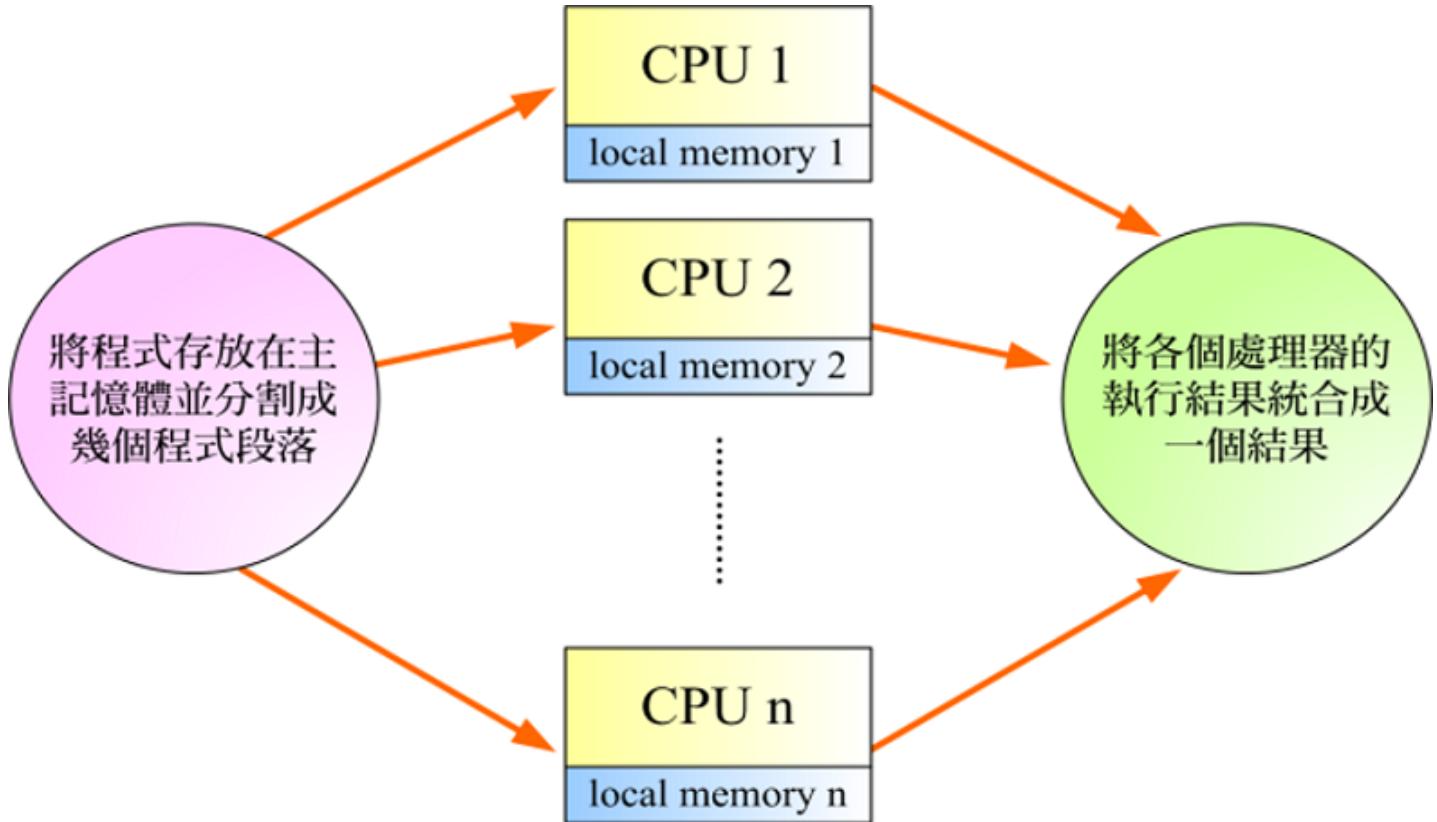


圖片來源：Intel、Apple

幾乎所有的CPU都使用二進位系統來表示數位。

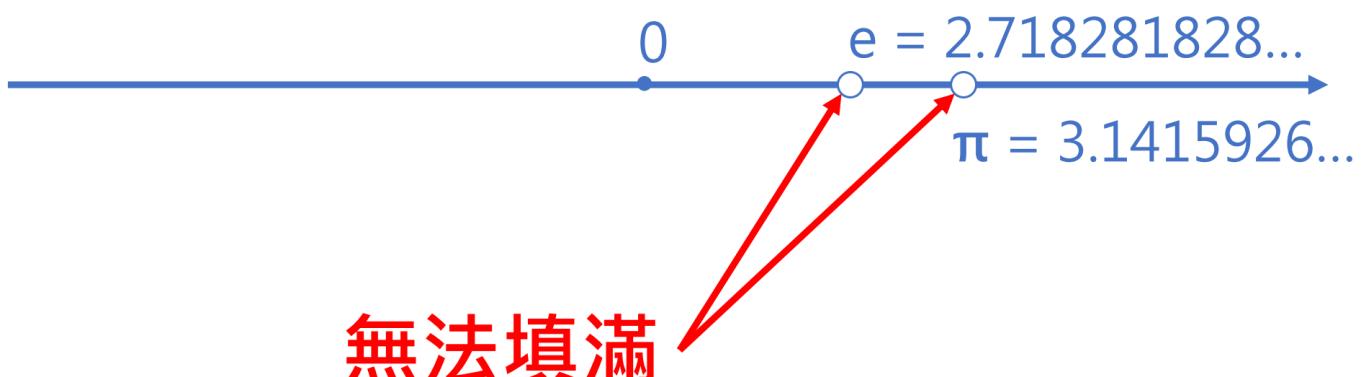
- 位元 (Bit)，亦稱二進制位，指二進位中的一位，以0或1來表示，是資訊的最小單位。
- 將八個位元合成一個單元來表示單一數字或字母，稱為位元組 (Byte) 。
 - $1\text{ Byte} = 8\text{ Bits}$ ，一個位元組可用來表示256個符號， $2^8 = 256$ 種組合。
 - 英文字以一個位元組來表示
 - 中文字則以兩個位元組來表示。
 - 一個8位元的CPU表示一次可以處理八個二進位的資料。

平行處理：一部電腦裡面有多個處理器，每個處理器都像一個CPU，可以獨立執行工作，至於主記憶體及I/O 則是共用。常見的多核心中央處理器有四核心、八核心等。

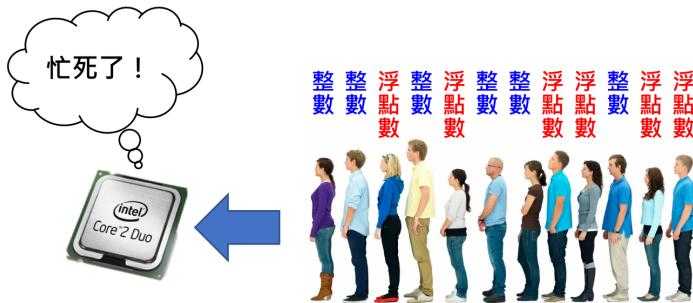


(2) 浮點運算器 (Floating Point Unit, FPU)

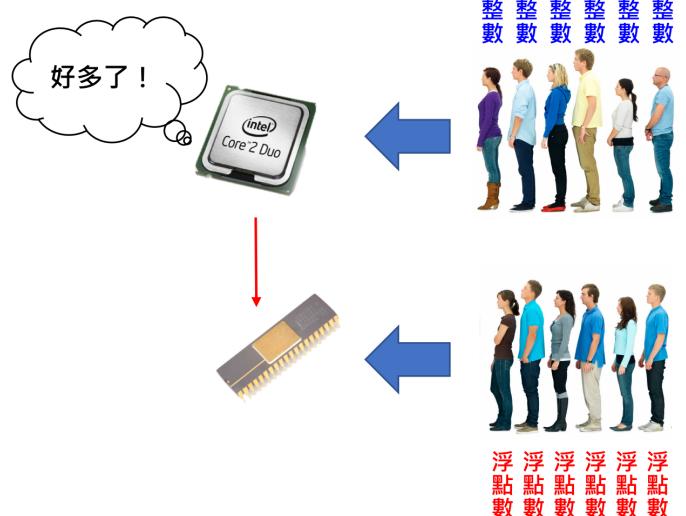
浮點運算器是來執行浮點運算。在現在的電腦架構中，浮點數運算跟整數運算是分開地，而浮點運算器大多被集成在CPU上，比較老的電腦硬體並不支援浮點數的運算。



• 沒有 FPU 幫忙



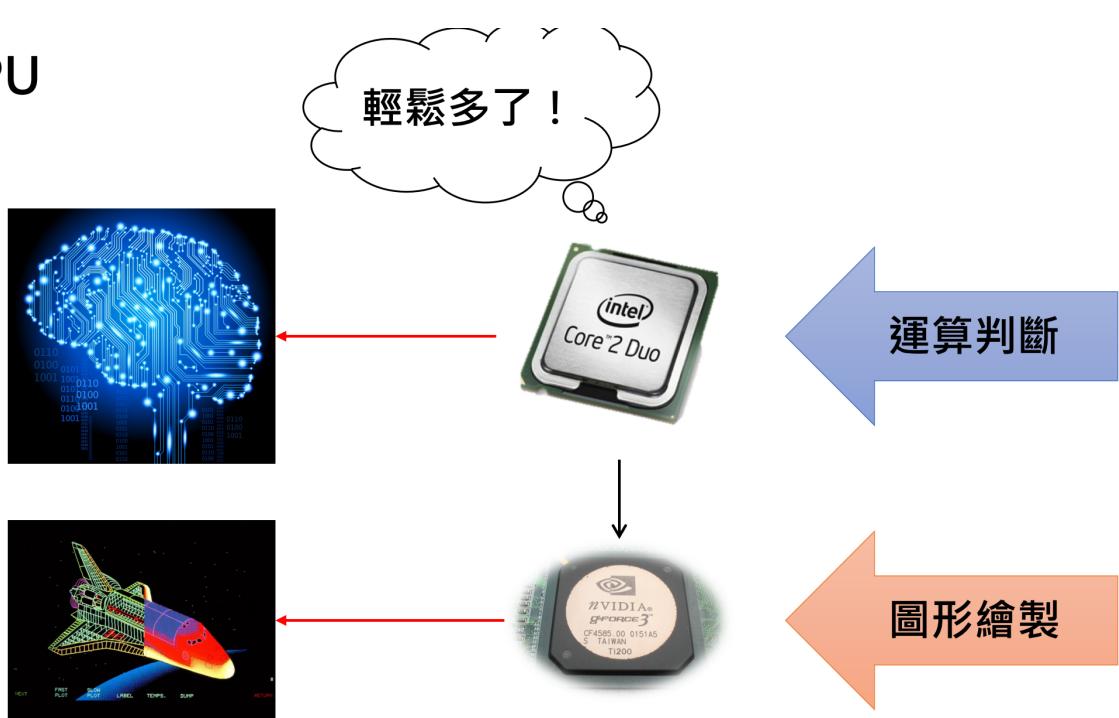
• 有 FPU 幫忙



(3) 圖形處理器 (Graphic Processing Unit, GPU)

圖形處理器是一種專門處理圖像運算工作的微處理器，能執行複雜的數學和幾何計算，擁有2D或3D圖形的加速功能，在浮點運算、平行計算上速度優越。有了GPU，CPU就從圖形處理的任務中解放出來，可以執行其他更多的系統任務，大大地提高電腦的整體效能。

• 現在：GPU



影片：GPU v.s.CPU的速度比較

<https://www.youtube.com/watch?v=1CkmDQdNZxY>

2. 儲存用硬體

儲存體 = 儲存程式碼 (指令 + 資料)

(1) 儲存體單位

- Bit : 位元，0或1
- Byte : 位元組，8 Bits
- KB (Kilo Byte) : 千位元組，常用於表示檔案大小， $1KB=1024Bytes = 10^3 Bytes$ 。
- MB (Mega Byte) : 百萬位元組，用於表示電腦主記憶容量， $1MB=1024KB = 10^6 Bytes$ 。
- GB (Giga Byte) : 十億位元組，用於表示硬碟容量， $1GB=1024MB = 10^9 Bytes$ 。
- TB (Tera Byte) : 兆位元組， $1TB=1024GB = 10^{12} Bytes$ 。
- PB (Peta Byte) : 千兆位元組， $1PB=1024TB = 10^{15} Bytes$ 。

(2) 儲存體材料



A. 一級儲存體：速度快、關機資料會消失

- 靜態隨機存取記憶體 (Static Random Access Memory, SRAM)
- 動態隨機存取記憶體 (Dynamic Random Access Memory, DRAM)

B. 二級儲存體：速度慢、關機資料不消失

- 快閃記憶體 (Flash)
 - USB
 - 安全數位卡 (Secure Digital Memory Card , SD卡)
 - 固態硬碟 (Solid State Disk)
- 其他
 - 機械硬碟 (Hard Disk)
 - 光碟 (Optical Disk)
 - 磁帶 (Tape)

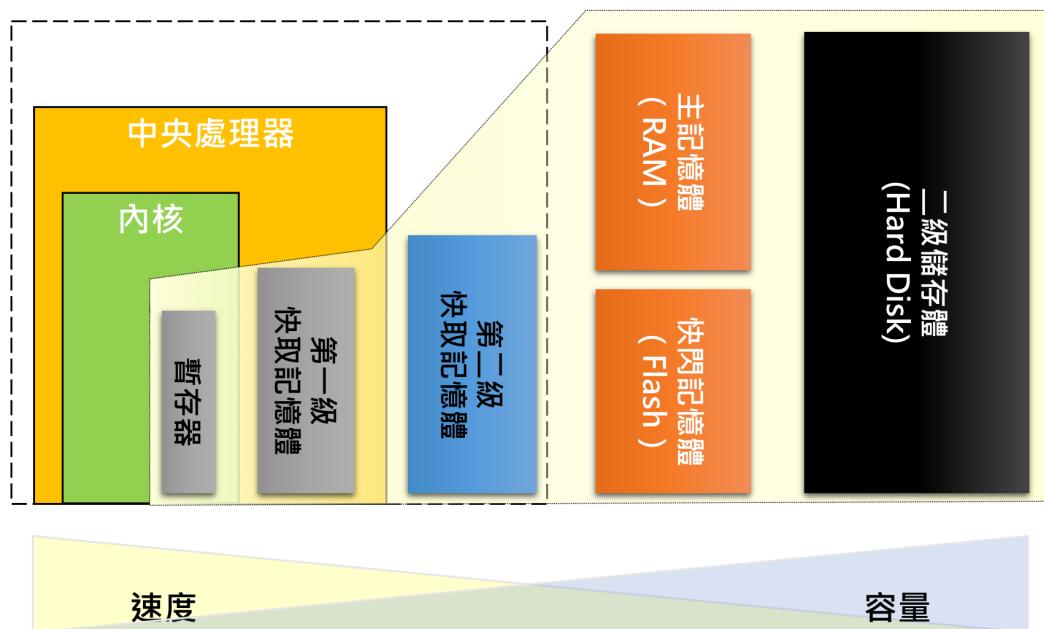
隨機存取記憶體 (Random Access Memory , RAM) ，也叫主記憶體，是電腦內部最主要的記憶體，用來載入各式各樣的程式與資料，以供CPU直接執行與運用。它可以隨時讀寫，而且速度很快，通常作為作業系統或其他正在執行中的程式的臨時資料儲存媒介。

當使用硬體電源關閉後，儲存於記憶體中的資料會消失，因此，又稱為動態隨機存取記憶體。如果保持通電，記憶體儲存的資料可以恆常保持，就是靜態隨機存取記憶體。然而，當電力供應停止時，SRAM儲存的數據還是會消失，這與在斷電後還能儲存資料的ROM或快閃記憶體是不同的。在電源關閉後，原本寫入的資料仍可保存於快閃記憶體中，可以非常快速的存取資料。再加上小體積、大容量的特性，快閃記憶體廣泛應用於許多可攜式的3C產品。



記憶體系統分布狀況

資料存取途徑



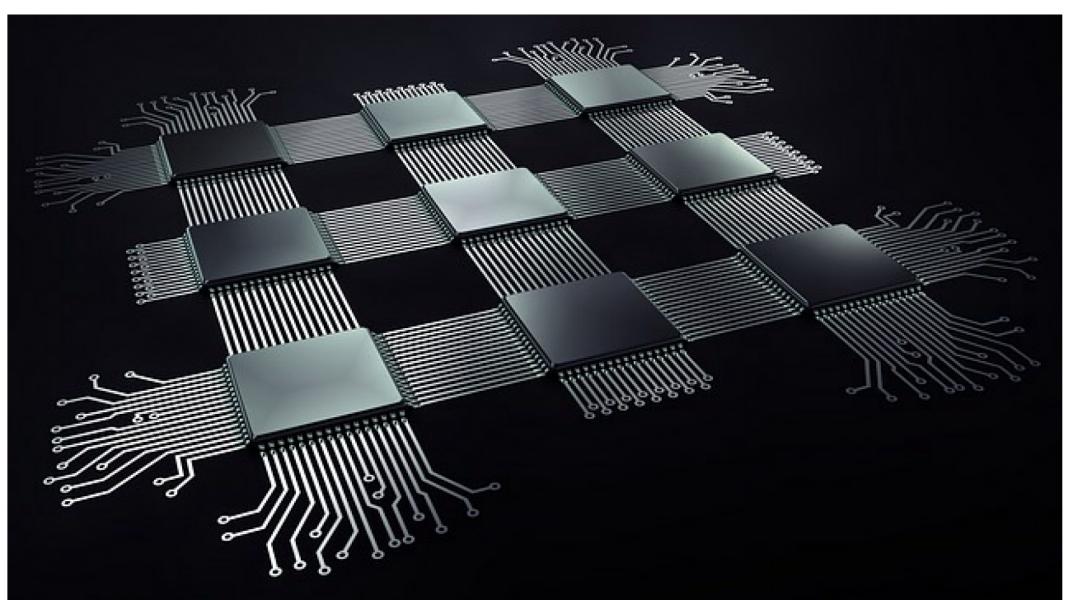
3. 連接用硬體

- 匯流排 (Bus)



何謂「匯流排」(Bus)

- 把兩個硬體連結在一起的一排電線



匯流排是主機板上面的鍍銅電路，由三組不同的電路所組成，電腦內部的電子訊號是由匯流排進行傳送。如果說主機板（Mother Board）是一座城市，那麼匯流排就像是城市裡的公共汽車（bus），能按照固定行車路線，傳輸來回不停運作的位元。這些線路在同一時間內都僅能負責傳輸一個位元，因此同時採用較多條的線路，就可以傳送較多的資料。

- 電腦的週邊設備

2-2-2 電腦軟體



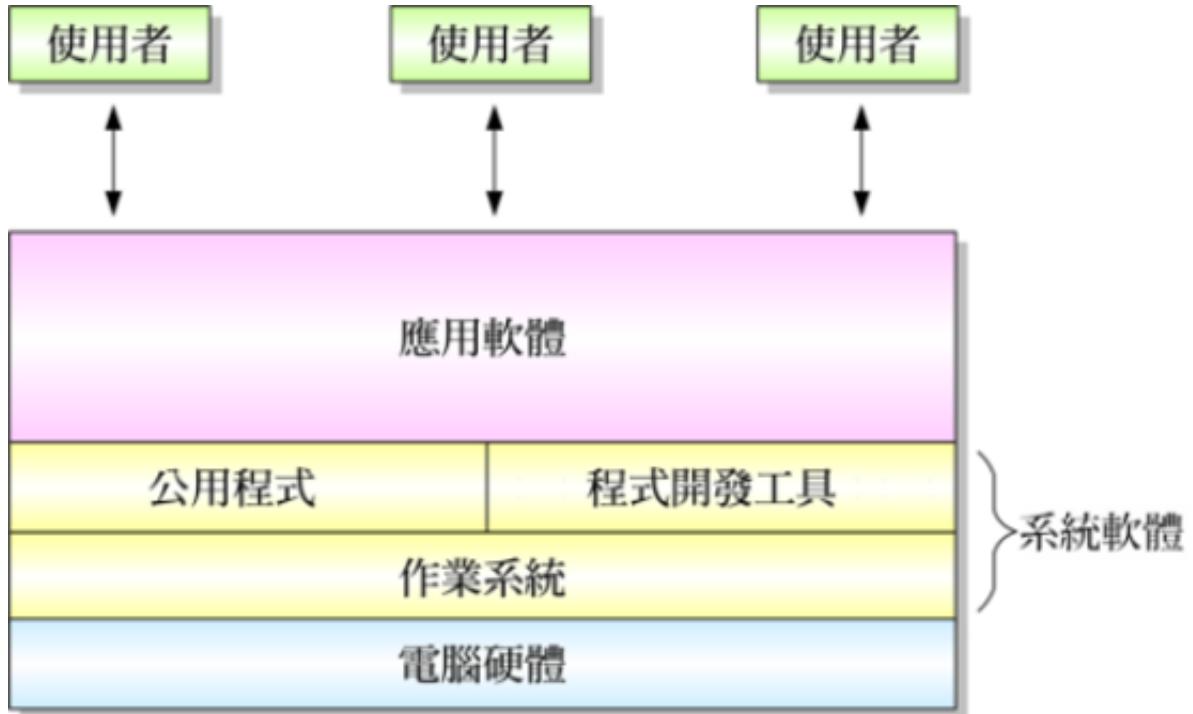
Windows 作業系統屬於系統軟體



Word 文書處理軟體屬於應用軟體

電腦軟體可以分為下列兩種類型：

- 1. 系統軟體 (system software)：負責支援電腦運作的程式，包括：**作業系統** (operating system)、**公用程式** (utility)、**程式開發工具** (program development tool)。
 - 作業系統是介於電腦硬體與應用軟體之間的程式，除了提供執行應用軟體的環境，還負責分配系統資源。
 - 公用程式是用來管理電腦資源的程式。
 - 程式開發工具是協助程式設計人員開發應用軟體的工具，包括文字編輯器等。
- 1. 應用軟體 (application software):針對特定事務或工作所撰寫的程式，目的是協助使用解決問題。



電腦軟體 - 作業系統

作業系統是管理電腦硬體與軟體資源的系統軟體，同時也是電腦系統的核心與基石。

作業系統層分為三層



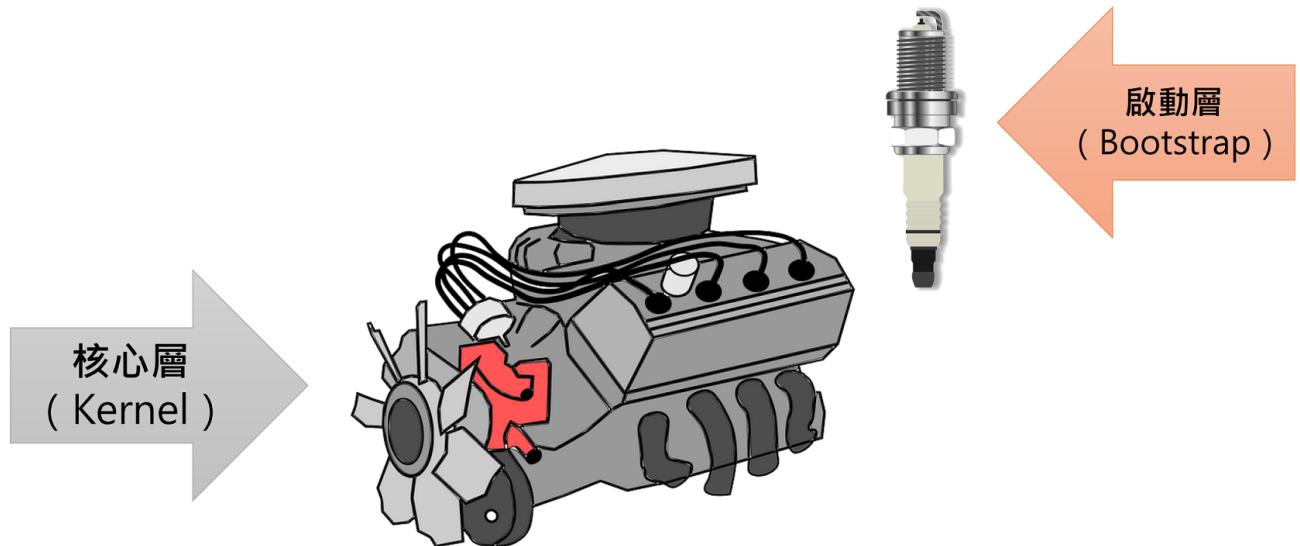
介面層 (Shell)

核心層 (Kernel)

啟動層 (Bootstrap)

(1) 啟動層 Bootstrap

開機程式是系統啟動後的第一個執行的程式，可將硬體（CPU與記憶體）初始化，讀進作業系統，直到可以迎接核心層的程度。



(2) 核心層 Kernel

核心是一個電腦程式，進行的是應用軟體和電腦硬體的互動工作。核心層下管硬體層，任何程式要用硬體都得跟核心層申請。核心層上管程式層，負責監督、協調與控制作業系統中的處理程式，讓所有執行中的程式和平相處正常運作。

(3) 介面層 Shell

一個接受並處理使用者的指令，讓使用者與系統互動的操作介面。

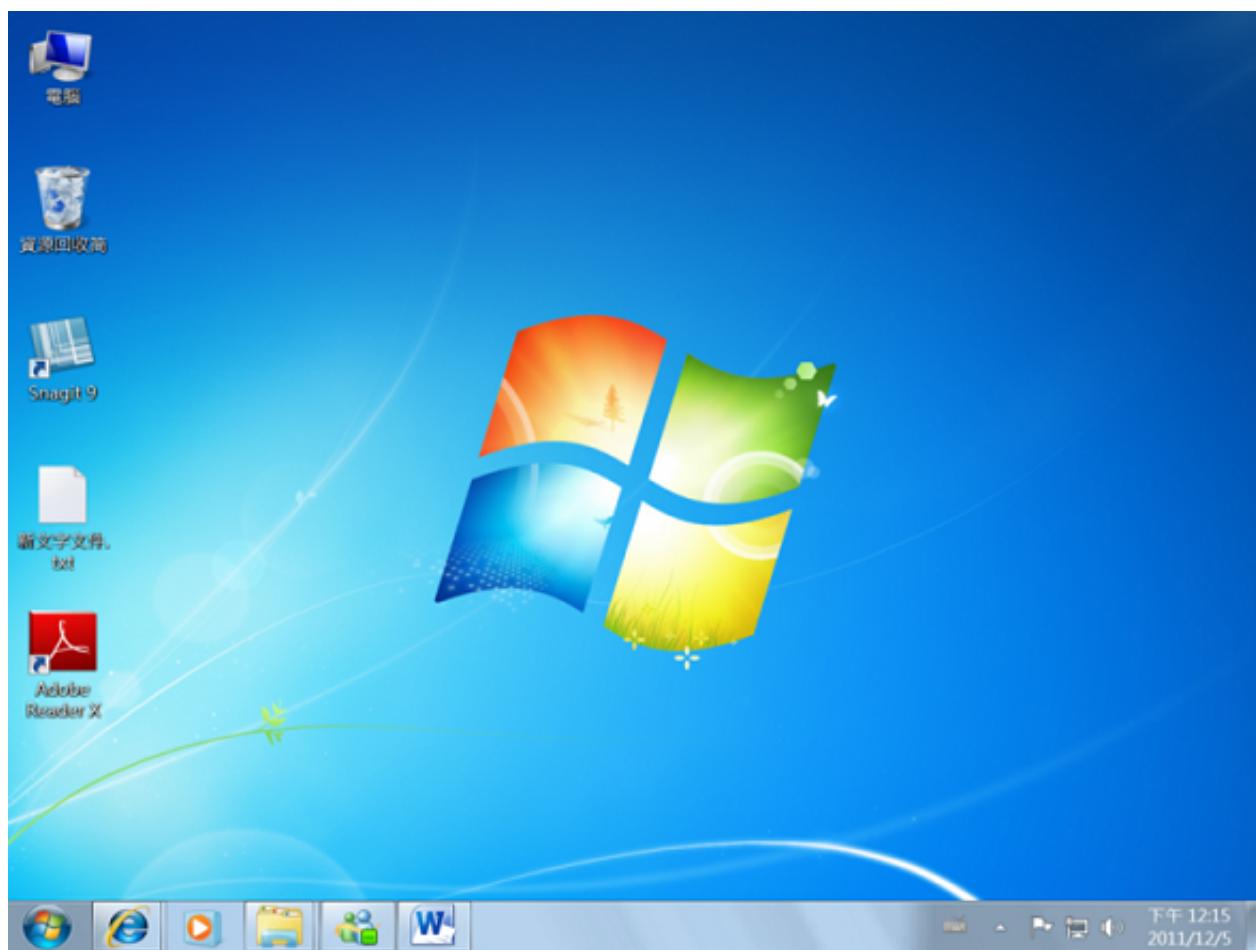
- 命令列使用者介面(CLI, command-line interface)：用鍵盤在命令提示列上鍵入文字指令。
- 圖形化使用者介面 (GUI, graphical user interface)：用滑鼠按一下按圖形鈕。

```
C:\>dir
磁碟區 C 中的磁碟是 SYSTEM
磁碟區序號: 58F0-4105

目錄: C:\

2004/07/26 11:09a    <DIR>      WINNT
2004/07/26 11:11a    <DIR>      Documents and Settings
2004/07/26 11:12a    <DIR>      Program Files
2004/07/30  01:25p    <DIR>      TMP
2004/09/04  02:59p    <DIR>      WUTemp
2004/09/13  02:43p    <DIR>      php
2005/01/14  01:24a    <DIR>      downloads
                           0 個檔案          0 位元組
                           7 個目錄        48,616,669,184 位元組可用

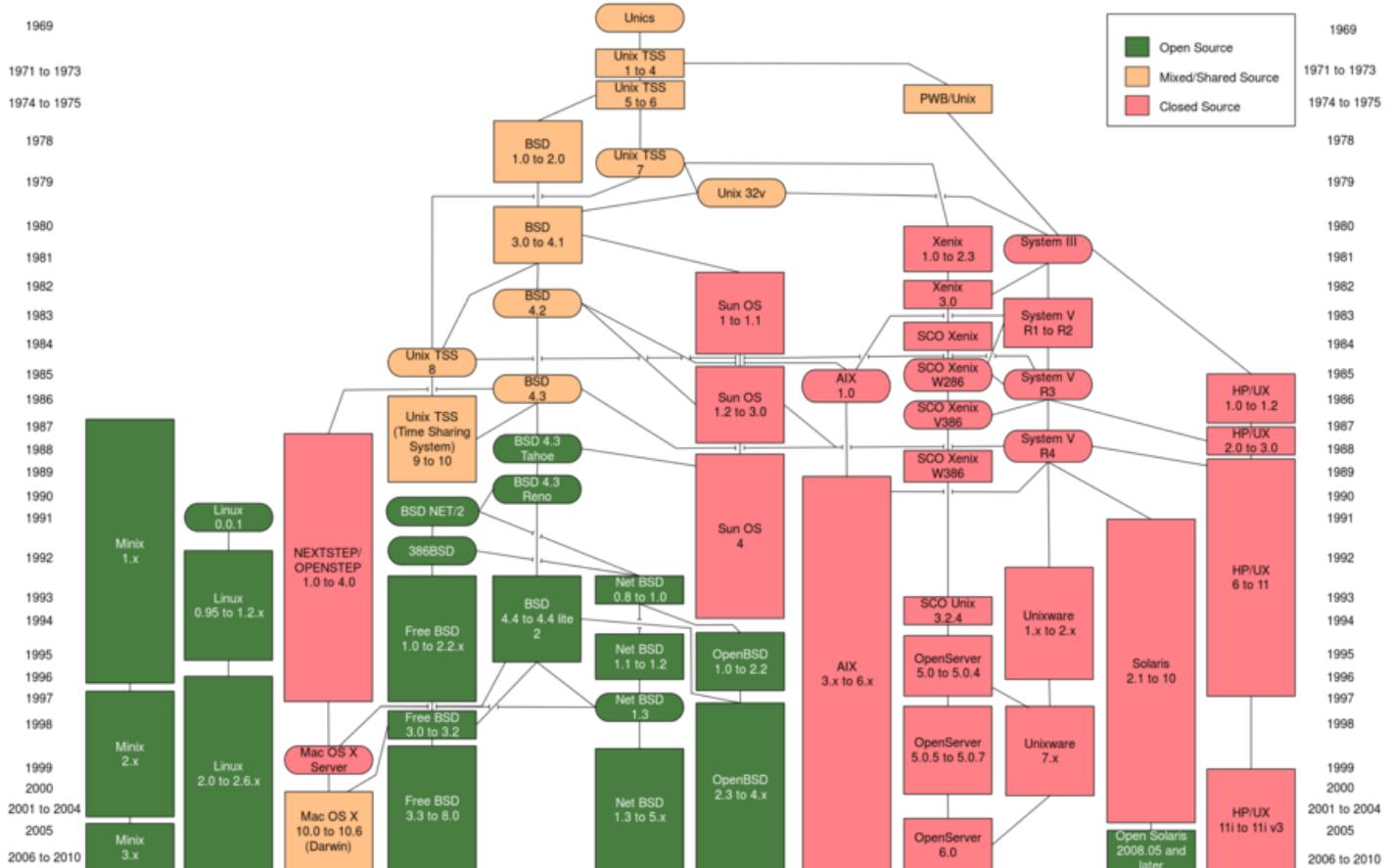
C:\>
```



常見的作業系統

UNIX HISTORY

1969 – 2010



UNIX電腦作業系統是由美國AT&T公司的貝爾實驗室，在1969年開發成功的，具有多工、多用戶的特徵。所有作業系統都或多或少衍伸自Unix系統。最受歡迎的Unix Shell是Bash (the Bourne Again Shell)，可在Linux系統上使用；Apple電腦的OSX系統為BSD(Berkely Software Distribution)，是目前商業化最成功的Unix作業系統；Window電腦則是採Unix-like的PowerShell。

(1) Linux系統

Linux是由芬蘭大學學生林納斯·托華斯 (Linus Torvalds)，於1991年以UNIX為基礎，開發出來安裝在個人電腦上的作業系統。近年來開放原始碼系的Linux越來越受歡迎，是市佔率最高的統，例如Ubuntu

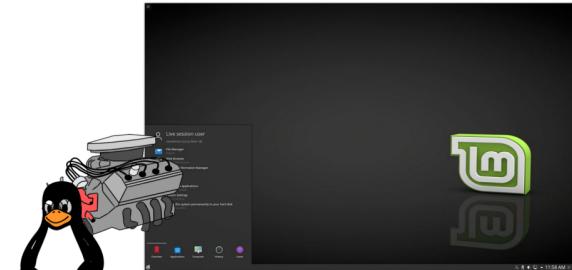


- 發行版 = Distribution
 - Linux 內核 + 任選一介面層 & 搭配軟體 = 發行版本

Ubuntu (w/Linux + Gnorm)



Linux Mint (w/Linux + KDE)



(2) Windows 系統

MS-DOS (Microsoft disk operating system)採用命令列使用者介面，是Microsoft公司於1975年針對IBM PC所推出的作業系統，使用者必須透過鍵盤輸入指定的指令集，才能指揮電腦完成工作，為Unix的仿作。

```
root@nas01:~# find . -name '*.pl'  
./project/man2html-page.pl  
./project/man2txt-page.pl  
./project/www-bot.pl  
./project/man2pdf-file.pl  
./project/test.pl  
.fetch.bots.pl  
root@nas01:~# find . -type f -name '*.pl'  
./project/man2html-page.pl  
./project/man2txt-page.pl  
./project/www-bot.pl  
./project/man2pdf-file.pl  
.fetch.bots.pl  
./project/test.pl  
root@nas01:~# find /etc/ -iname 'resolv.conf'  
/etc/resolv.conf  
root@nas01:~#
```



```
C:\Temp> dir
Volume in drive C is C
Volume Serial Number is 74F5-B93C

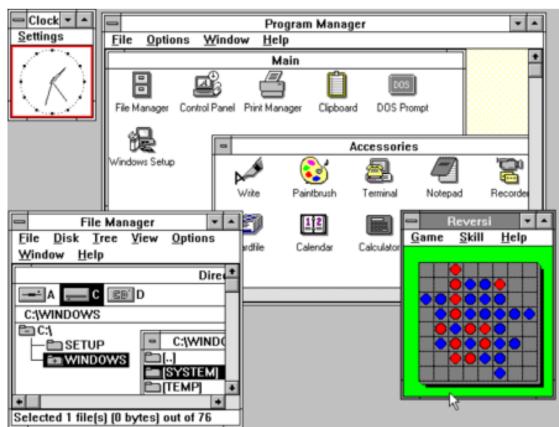
Directory of C:\Temp

2009-08-25  11:59    <DIR>          .
2009-08-25  11:59    <DIR>          .
2007-03-01  11:37      2,321,600 AdobeUpdater12345.exe
2009-04-03  10:01      27,988 dd_depcheckdotnetfx30.txt
2009-04-03  10:01      764 dd_dotnetfx3error.txt
2009-04-03  10:01      32,572 dd_dotnetfx3install.txt
2009-06-09  13:46      35,145 GenProfile.log
2009-08-05  12:11      155 KB969856.log
2009-04-20  08:37      402 MSI29e0B.LOG
2009-04-09  16:34      38,895 offcn1n1.log
2009-04-03  16:02    <DIR>          OfficePatches
2009-07-14  14:30    <DIR>          OHotfix
2009-08-25  10:52      16,384 Perflib_Perfda
2009-04-03  10:01      1,744 uxeventlog.txt
2009-08-25  11:42      50,245,632 WFVF2.F TMP
2009-04-20  10:07      1,397 {AC76BA86-7AD7-4
2009-04-20  10:13      617 {AC76BA86-7AD7-4

13 File(s)      52,723,295 bytes
4 Dir(s)   83,570,208,768 bytes free
```



MS-Windows (Microsoft Windows)則是微軟在MS-DOS基礎上設計的圖形作業系統。1985年、1987年所推出的Windows 1.0和Windows 2.0順利轉型成為IBM相容PC的標準圖形使用者介面系統，1990年推出的Windows 3.0獲得了空前迴響。



1990 MS-Windows 3.0



1995 MS-Windows 95

1995年推出的Windows 95不再包含MS-DOS，Windows從殼層程式轉變為真正的作業系統。之後Microsoft公司不斷推出新版的Windows作業系統，包括Windows Me、Windows XP、Windows Vista 和Windows 7/8/10。現在的Windows系統皆是建立於Windows NT核心，可以在32位元和64位元的Intel和AMD的處理器上運行。

最新版 MS-Windows 10



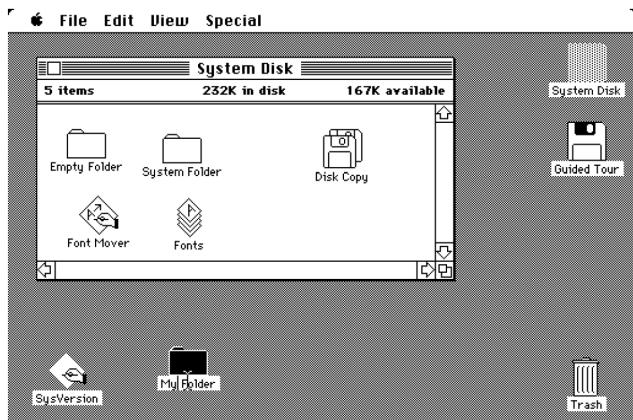
2015 MS-Windows 10

(3) MAC系統

macOS，稱「Mac OS X」或「OS X」，是一套執行於蘋果Macintosh系列電腦上的

作業系統。Apple公司的創始人Steve Jobs意識到圖形使用者介面的重要性及未來前景，於1983年、1984年推出採用圖形化使用者介面的個人電腦Macintosh (麥金塔)。Mac OS指的就是安裝於Apple Macintosh電腦的作業系統，以卡內基美隆大學開發的Mach做為核心，是第一個商用化成功的圖形化使用者介面系統，最終版本為1999年推出的Mac OS 9。之後Apple公司改以BSD UNIX為基礎推出OS X，從OS X 10.8開始在名字中去掉Mac，僅保留OSX和版本號。2016年蘋果公司將OS X更名為macOS，現行的最新的系統版本是macOS Mojave。

1984 ~ 2001 : Classic Mac OS



Mac OS 1.0 (1984)



Mac OS 9.2 (2001)

2001 ~ 現今 : Mac OS X



版本	代號
10.0	Cheetah
10.1	Puma
10.2	Jaguar
10.3	Panther
10.4	Tiger
10.5	Leopard
10.6	Snow Leopard
10.7	Lion
10.8	Mountain Lion
10.9	Mavericks
10.10	Yosemite
10.11	El Capitan
10.12	Sierra
10.13	High Sierra
10.14	Mojave

應用軟體 - 程式語言

～什麼是程式語言

簡單的說，程式只是一堆指令的集合體，而這些指令是用來告訴電腦應該要作那些事情。 程式語言是用來向電腦發出指令，讓程式設計師能夠準確地定義電腦所需要使用的資料，並精確地定義在不同情況下所應當採取的行動。

- 指令 (instruction) 是指揮電腦完成一項基本任務的命令。
- 程式 (program) 是一組有順序的指令集合。
- 程式語言 (program language) 是用來撰寫程式的語言。

為什麼要學程式語言？

- 不能直接用中文或英文嗎？
- 自然語言，一種意思有多種說法
- 程式語言，限制多種說法、所以可順利和電腦溝通。
- 該學什麼程式語言？
 - 依照需要和興趣
 - 資料科學：R 和 Python (Julia)

紀老師 <https://www.youtube.com/watch?v=01HzKVnXnPU&t=212s>

https://www.youtube.com/watch?v=Lnw_NTGk5zc&t=557s

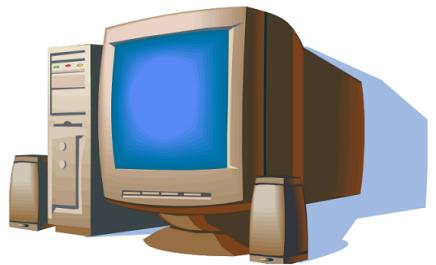
不能用「中文」或「英文」嗎？



你好？



0110101101010
10111



一種意思，多種說法



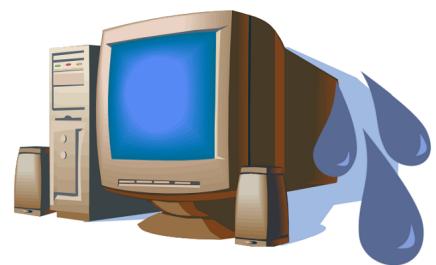
要不要去吃飯？



你現在會餓嗎？



一起去用餐吧！

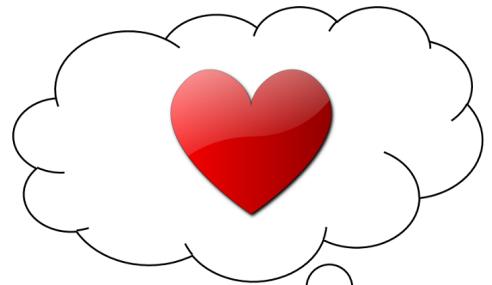




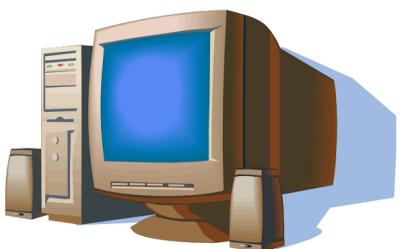
蹭飯語言



f (你, 我, 吃飯)



f (你, 我, 吃飯)



f (你, 我, 吃飯)

程式層

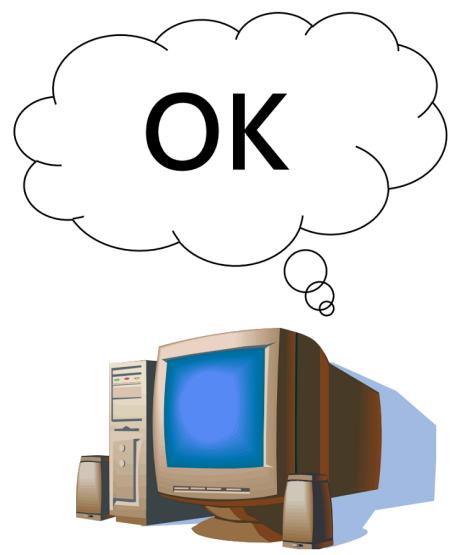
程式層分為三部分：原始碼、執行檔與開發工具。

(1) 如何寫出原始碼

- 原始碼 (Source code)，是指一系列人類可讀的電腦語言指令。



原始碼 = 溝通事項清單



- 為了編譯出電腦程式，要用好的文字編輯器紀錄下來



好用的「純」文字編輯器



Sublime Text



- 原始碼格式通常是文字檔案，純文字檔 + 特殊的副檔名



原始碼 = 純文字檔 + 特殊副檔名

```
43 <body> <php begin></body>
44 <div id="fb-root"></div>
45 <script>(function(d, s, id) {
46   var js, fjs = d.getElementsByTagName('script')[0];
47   if (!d.getElementById(id)) {
48     js = d.createElement('script');
49     js.id = id;
50     js.src = "https://connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.6&appId=1493423244264549";
51   }
52   if (!js) {
53     d.parentNode.insertBefore(js, d);
54   }
55   fjs.parentNode.insertBefore(js, fjs);
56 })(document, 'script');
```



python™

=



+ .py



Java

=



+ .java



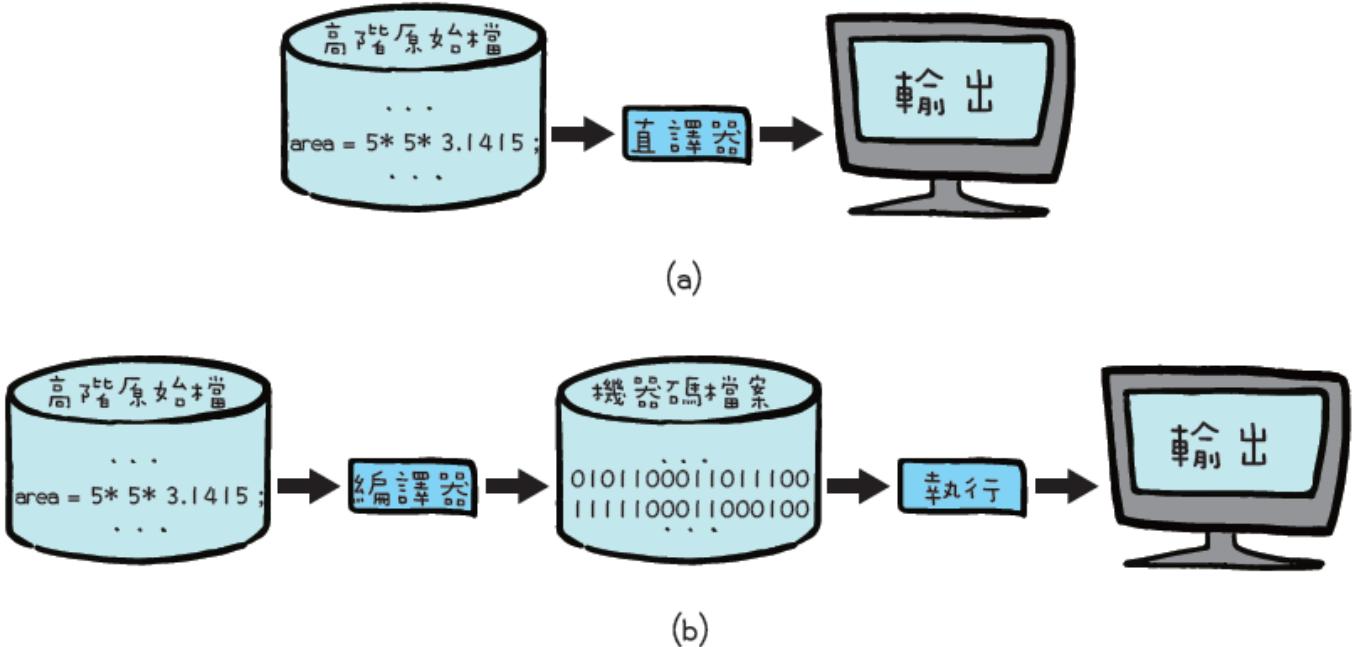
=



+ .cs

(2) 如何執行編譯

將人類可讀的文字翻譯成執行檔，一種內容可被電腦解釋為程式的電腦檔案。通常執行檔內，指令與資料都用0與1二進制編碼，也因此執行檔有時稱為二進制檔。這個過程叫做編譯，無論是直譯器還是編譯器都會根據文法對原始程式進行語法分析及語意剖析。不同的是直譯器不會產生目的碼，而是每翻譯一行敘述，就立刻執行該敘述；編譯器則是將以程式轉換成目的碼，裡面包含機器指令、資料值及這些項目的位址，從頭到尾一次執行。



A. 直譯器 (interpreter)

- 使用直譯器時，程式語言變成了一個會和你互動的環境。
 - 在命令提示列上直接打指令時，直譯器會立刻執行該命令。
 - 把所有指令統統輸入到某個檔案裡面，然後呼叫直譯器讀取該檔案，並執行檔案中的指令亦然。
- 如果所下的指令有錯誤產生，直譯器會進入偵錯模式 (debugger)，並且顯示相關錯誤訊息，以便對程式除錯。
 - 這種方式可以立刻看到指令的執行結果，並迅速修正錯誤。
 - 直譯器會使用到很多的記憶體，執行效率通常比編譯器差。

B. 編譯器 (compiler)

- 使用編譯器時，必須先把程式碼統統寫到檔案裡面，然後執行編譯器來編譯程式。
 - 在提示命令列可以執行你編譯好的程式，看看它是否可以運作。
 - 如果編譯器不接受所寫的程式，就必須一直修改程式，直到編譯器接受且把你的程式編譯成執行檔為止。

- 編譯器與直譯器的差別

- 很明顯的，使用編譯器並不像直譯器般可以馬上得到結果。
- 編譯器執行效率高，並在編譯時順便最佳化你的程式，但是直譯器卻不行。
- 當你想把你寫好的程式拿到另外一台機器上跑時，你只要將編譯器編譯出來的可執行檔，拿到新機器上便可以執行，而直譯器則必須在新機器上重新執行一次。

(3) 開發工具

- 程式設計需要很多的軟體工具協助

- 文字編輯器：撰寫原始碼
- 編輯器 / 直譯器：將原始碼翻譯成0與1
- 除錯器：找出錯誤在哪一列





每個工具的用途



純文字編輯器 (Text Editor)

- 儲存時，除了文字本身，不存任何格式設定的軟體
- 如：NotePad (Word 就不是純文字編輯器)



直譯器 / 編譯器 (Interpreter / Compiler)

- 直譯器：一次翻譯一列成為 0/1 的軟體
- 編譯器：一次翻譯一個檔案成為 0/1 的軟體



除錯器 (Debugger)

- 一次執行一列，讓你慢慢找到底程式寫錯在哪一列的軟體
- 如果程式一次就寫對，那麼就不需要執行除錯這個步驟



有沒有把所有工具整合起來的軟體？



▶ 有！叫做「整合式開發環境」

IDE
Integrated Development Environment



3 個願望，一次滿足

- 整合式開發環境

- Integrated Development Environment, IDE
- 整合程式設計需要的工具

這一堆「器」都是為了開發程式而存在的工具性軟體，每個都是各自獨立散落的。因為都會用到，把需要的程式開發工具統整合（Integrated），在同一個程式操作畫面，彷彿置身於一個開發「環境」（environment）內，所以稱為整合開發環境。今日多數的程式開發工具都是IDE型態，一旦啟動開發用，程式畫面中就包含各種功能，既可以編輯程式碼文字，也可以編譯、連結、除錯等。

Summary:

～程式語言的演化歷史大概分成三個階段，由機器語言->組合語言->高階語言。

- 機器語言是最初的的程式語言，就直接使用0和1來撰寫程式。
- 大家發現根本不大可能直接用0和1來寫程式，所以CPU廠商制定了一套指令集，由CPU負責『翻譯』成機器語言。
- 組合語言還是太基本，不適合人類直接撰寫，所以我們開發出了高階語言，你今天所有聽到的C,Python, JAVA, Swift程式語言都屬於高階語言的範疇，所以在執行程式的時候要有個翻譯器把『高階語言』翻譯成『組合語言』。

～翻譯器依照本身的設計，有編譯器與直譯器兩種；翻譯器依照本身的版本也也所不同，如Python 2與 Python 3；也會依照使用者電腦作業系統的差異，有32bit與64bit的不同。

～當你要寫程式讓電腦執行時，你需要翻譯器，你也需要文字編輯器撰寫程式。寫完的程式先讓電腦跑跑看，有沒有錯誤，沒有錯誤才算寫好程式。不幸的是我們常常寫錯，所以還需要除錯器來幫忙修正。幸運的是，我們有完整的輔助工具，也就是一個包含翻譯器、文字編輯器，以及除錯器的整合開發環境，讓程式設計這件事變得輕鬆些。整合開發環境通常跟用一種高階語言有關，例如在Rstudio整合開發環境寫R程式語言（用Rstudio學R），在Jupyter Notebook整合開發環境寫Python程式語言（用Jupyter Notebook學Python、用Pycharm學Python）等等。

補充資料 - 學習建議：用Python學運算思維

學習Python不能只是盲目學語法，跟隨整合專題的引導，有效拆解問題才是程式設計的基石。

學習程式設計，不該只是盲目死記難懂的指令及符號，而是培養邏輯、勇敢嘗試、並

實現創意的過程。

理解並釐清問題是程式開發的基石。

寫程式不是被動的學習，要自己動手才能真的學到。建議理解範例程式的細節後，自行將程式碼Key-In一遍，不要Copy-Paste，強化學習效果。然後驗證觀念，確定自己懂了多少，並做些練習多思考多理解。

Philosophical Note on Learning _____ (I)

- You learn _____ to become a better programmer and **thinker**.
- You learn _____ not just for learning a new syntax; you want to learn new and better ways of **solving problem** and building software.
- **Is it hard?**

It depends, but considering how long it takes to learn a new language, such as **English!**

(Good news) Becoming a good programmer is easier and faster.

(Bad news) But **not** as easier and faster as most people would like it to be.

Philosophical Note on Learning ____ (II)

- You need to understand the principles (瞭解原理).
- You need to have your hands dirty (親自動手) to fully appreciate it.
- Skills come with practice!

學習方法：笨辦法更簡單 The Hard Way Is Easier

這本小書的目的是讓你起步程式設計。雖然書名說是“笨辦法”，但其實並非如此。所謂的“笨辦法”是指本書教授的方式。在這本書的幫助下，你將通過非常簡單的練習學會一門程式設計語言。**做練習是每個程式師的必經之路：**

- | | |
|---|-----------------|
| 1 | 1. 做每一道習題 |
| 2 | 2. 一字不差地寫出每一個程式 |
| 3 | 3. 讓程式運行起來 |

就是這樣了。剛開始這對你來說會非常難，但你需要堅持下去。如果你通讀了這本書，**每晚花個一兩小時做做習題**，你可以為自己讀下一本程式設計書籍打下良好的基礎。通過這本書你學到的可能不是真正的程式設計，但你會學到最基本的學習方法。

這本書的目的是教會你程式設計新手所需的三種最重要的技能：讀和寫、注重細節、發現不同。

I. 讀和寫

很顯然，如果你連打字都成問題的話，那你學習程式設計也會成問題。尤其如果你連程式原始程式碼中的那些奇怪字元都打不出來的話，就根本別提程式設計了。沒有這

樣基本技能的話，你將連最基本的軟體工作原理都難以學會。

為了讓你記住各種符號的名字並對它們熟悉起來，你需要將代碼寫下來並且運行起來。這個過程也會讓你對程式設計語言更加熟悉。

II. 注重細節

區分好程式師和差程式師的最重要的一個技能就是對於細節的注重程度。事實上這是任何行業區分好壞的標準。如果缺乏對於工作的每一個微小細節的注意，你的工作成果將缺乏重要的元素。以程式設計來講，這樣你得到的結果只能是毛病多多難以使用的軟體。

通過將本書裡的每一個例子一字不差地打出來，你將通過實踐訓練自己，讓自己集中精力到你作品的細節上面。

III. 發現不同

程式師長年累月的工作會培養出一個重要技能，那就是對於不同點的區分能力。有經驗的程式師拿著兩份僅有細微不同的程式，可以立即指出裡邊的不同點來。程式師甚至造出工具來讓這件事更加容易，不過我們不會用到這些工具。你要先用笨辦法訓練自己，等你具備一些相關能力的時候才可以使用這些工具。

在你做這些練習並且打字進去的時候，你一定會寫錯東西。這是不可避免的，即使有經驗的程式師也會偶爾寫錯。你的任務是把自己寫的東西和要求的正確答案對比，把所有的不同點都修正過來。這樣的過程可以讓你對於程式裡的錯誤和bug更加敏感。

A. 不要複製粘貼

你必須手動將每個練習打出來。複製粘貼會讓這些練習變得毫無意義。這些習題的目的是訓練你的雙手和大腦思維，讓你有能力讀代碼、寫代碼、觀察代碼。如果你複製粘貼的話，那你就是在欺騙自己，而且這些練習的效果也將大打折扣。

B. 對於堅持練習的一點提示

在你通過這本書學習程式設計時，我正在學習彈吉他。我每天至少訓練 2 小時，至少花一個小時練習音階、和聲、和琶音，剩下的時間用來學習音樂理論和歌曲演奏以

及訓練聽力等。有時我一天會花 8 個小時來練習，因為我覺得這是一件有趣的事情。對我來說，要學好一樣東西，每天的練習是必不可少的。就算這天個人狀態很差，或者說學習的課題實在太難，你也不必介意，只要堅持嘗試，總有一天困難會變得容易，枯燥也會變得有趣了。

在你通過這本書學習程式設計的過程中要記住一點，就是所謂的“萬事開頭難”，對於有價值的事情尤其如此。也許你是一個害怕失敗的人，一碰到困難就想放棄。也許你是一個缺乏自律的人，一碰到“無聊”的事情就不想上手。也許因為有人誇你“有天分”而讓你自視甚高，不願意做這些看上去很笨拙的事情，怕有負你“神童”的稱號。也許你太過激進，把自己跟有 20 多年經驗的程式設計老手相比，讓自己失去了信心。

不管是什麼原因，你一定要堅持下去。如果你碰到做不出來的加分習題，或者碰到一節看不懂的習題，你可以暫時跳過去，過一陣子回來再看。只要堅持下去，你總會弄懂的。

一開始你可能什麼都看不懂。這會讓你感覺很不舒服，就像學習人類的自然語言一樣。你會發現很難記住一些單詞和特殊符號的用法，而且會經常感到很迷茫，直到有一天，忽然一下子你會覺得豁然開朗，以前不明白的東西忽然就明白了。如果你堅持練習下去，堅持去上下求索，你最終會學會這些東西的。也許你不會成為一個程式設計大師，但你至少會明白程式是怎麼工作的。

如果你放棄的話，你會失去達到這個程度的機會。你會在第一次碰到不明白的東西時(幾乎是所有的東西)放棄。**如果你堅持嘗試，堅持寫習題，堅持嘗試弄懂習題的話，你最終一定會明白裡邊的內容的。**

如果你通讀了這本書，卻還是不知道程式設計是怎麼回事。那也沒關係，至少你嘗試過了。你可以說你已經盡過力但成效不佳，但至少你嘗試過了。這也是一件值得你驕傲的事情。

C. 紿“小聰明”們的警告

有的學過程式設計的人讀到這本書，可能會有一種被侮辱的感覺。其實本書中沒有任何要居高臨下地貶低任何人的意思。只不過是我比我面向的讀者群知道的更多而已。如果你覺得自己比我聰明，然後覺得我在居高臨下，那我也沒辦法，因為你根本就不屬於我的目的讀者群。

如果你覺得這本書裡到處都在侮辱你的智商，那我對你有三個建議：

1. 別讀這本書了。我不是寫給你的，我是寫給需要學習的人的。
2. 放下架子好好學。如果你認為你什麼都知道，那你就很難從比你強的人身上學到什麼了。
3. 學 Lisp 去。我聽說什麼都知道的人可喜愛 Lisp 了。

對於其他在這裡學習的人，你們讀的時候就想著我在微笑就可以了，雖然我的眼睛裡還帶著惡作劇的閃光。