

# 入門概論 - 運算思維與程式設計（一）

---

## 1 電腦科學

無論是日常生活還是工作，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！

### 1-1 什麼是電腦科學

**Computer science is fundamentally about computational problem solving.**

～用電腦解決問題

- 有問題要解決，需要瞭解問題，並把解決問題的方法找出來。
- 透過程式語言展現運算思維，將解決問題的演算法指令寫出來。
- 電腦依照指令工作，上述解決問題的程式設計可以讓電腦替我們工作。

～ 運算思維與程式設計是學習電腦科學的核心。

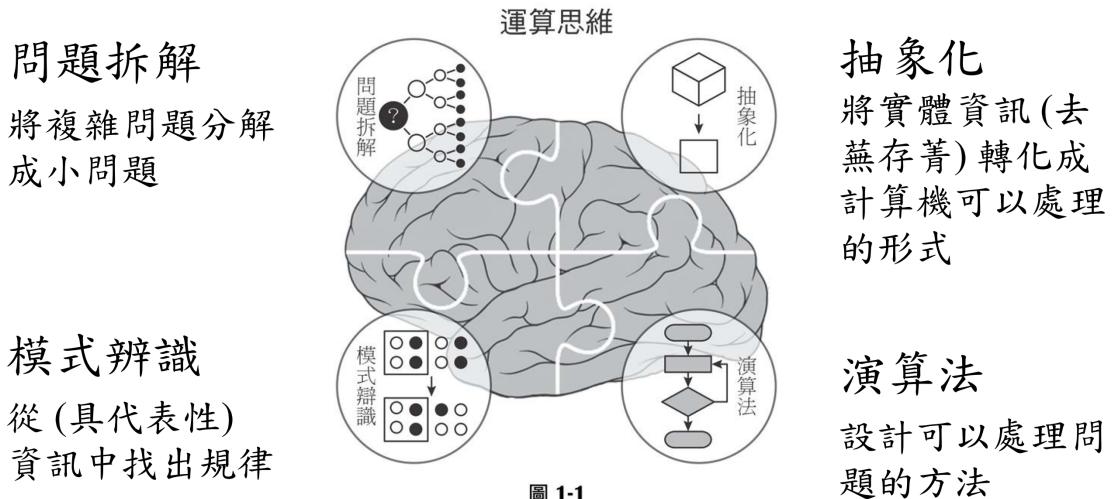
學習運算思維與程式設計，用電腦幫助我們解決問題是現代人必備的技能。

## 1-2 運算思維

[Youtube: 什麼是運算思維](#)

運算思維是一種解決問題的方法，包含四個基石：

### 運算思維：是一種解決問題的方法



(1)拆解問題(Decomposition)，化繁為簡，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

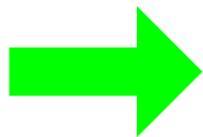
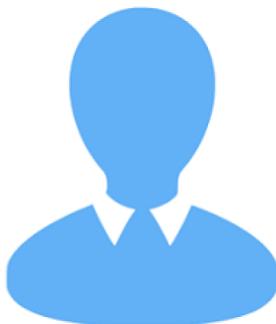
解之前簡單，可以更有效率的處理，也方便與他人分工合作。

(2)抽象化(Abstraction)，去蕪存菁，識別並擷取可表達主要概念的重要訊息。

將實體資訊轉換成計算機可以處理的形式，包括資料抽象化與程序抽象化。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)  
：

例：請問從世新大學管理學院如何到政治大學大門口？你可以化一張很簡單的地圖，從木柵路一直開開到沒有路時右轉，又沒有路時右轉，第一個紅路燈左轉，即可到達。在描述整個過程，我們畫的地圖只是示意圖，不是真實的路況，這就是「程序抽象化」。

(3)模式辨識(Pattern recognition)，找出資料中可見的樣式、規律、趨勢。

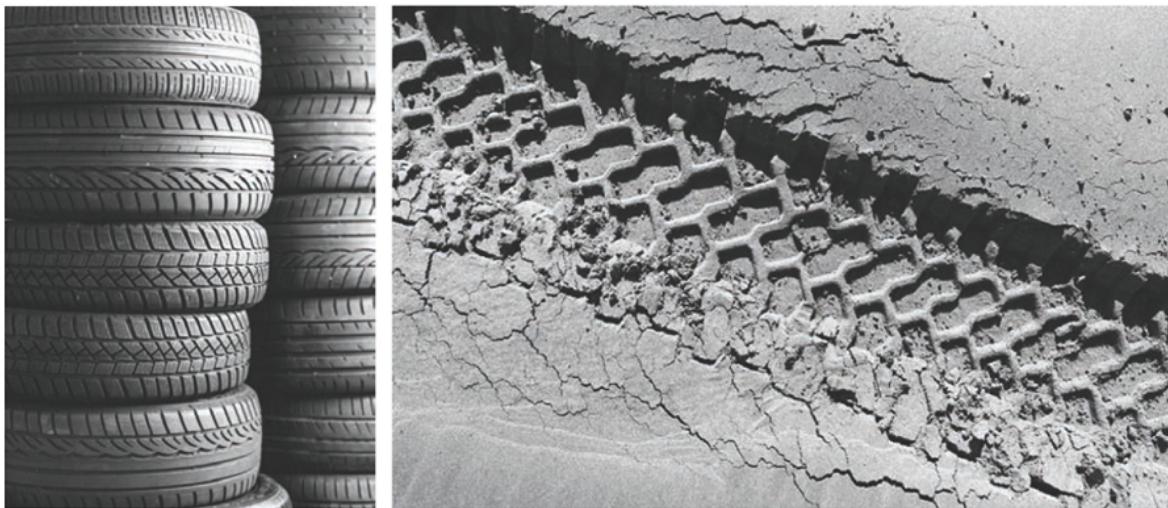


圖 1-20 ( 圖片來源：Pixabay )

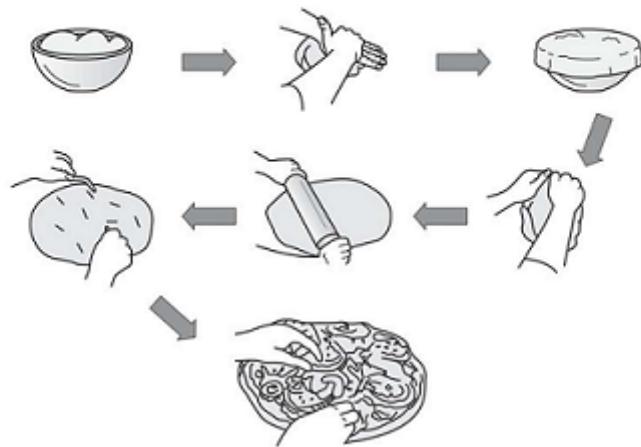
從具有代表性資訊中找出規律，例如用敘述統計表示特性，南部民眾說台灣話的比例高於北部，台北人大多使用國語交談。擁有10張以上信用卡者，欠債不還款比例較高。

例：從世新到政大的路線，用手直接畫出來，也可以看到路線的型態。

(4)演算法(Algorithms)，解決問題的具體步驟與流程。

演算法是運算思維的具體展現。在設計程式時必須先將問題分解成許多小步驟，然後再依一定的次序逐步執行，我們將這個描述問題之解決程序的方法稱做演算法。有了演算法表示已經針對問題提出解決之道，接下來就是程式設計，用程式寫出相對應的指令，讓電腦來幫助我們處理問題。

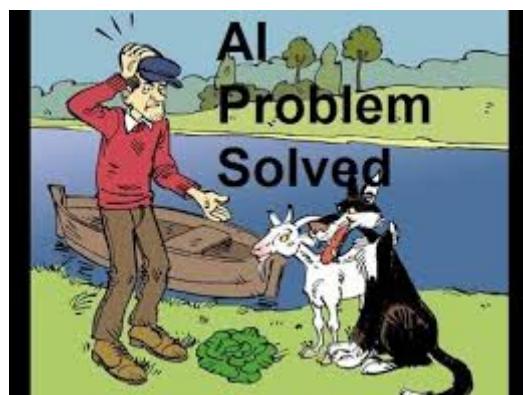
# 演算法



**Algorithms and computers are a perfect match!**

實例：渡河之謎（一）

～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！！！

渡河之謎的答案，就是解決問題的方法，也就是運算思維！

## **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.

## 隨堂練習：渡河之謎（二）

假設一個大人與兩個小孩要渡河，三位都會划船。由於船很小，一次最多只能渡一個大人，或者是兩位小孩，大人與小孩不能一起渡河。請問要如何讓三位渡河到對岸呢？

### ～運算思維，演算法

- Step 1: 兩個小孩一起渡河
- Step 2: 其中一個小孩划船返回
- Step 3: 大人渡河
- Step 4: 另一位小孩划船返回
- Step 5 : 兩個小孩再次一起渡河

## 隨堂練習：渡河之謎（三）

河岸有三對夫婦，他們都要過河，可是只有一條能乘兩個人的小船，而且，這三個男人都很保守，他們不希望自己的妻子在他本人不在的情況下和別的男人在一起。請想想，用什麼辦法把他們都渡過去。當然，船要他

(她) 們自己會划。每次渡河都要有人划回原處，直至全渡過去為止。

## 1-3 程式設計

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

**Question:** 為什麼要寫程式?

「解決問題」是「寫程式」最原始的動機與最基本的要 求。大部分的程式都是為了解決某一特定問題而產生。

- 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。電腦依照這些指令的步驟逐一執行計算或資料處理的動作。
- 程式語言 (program language) 是用來撰寫程式的語言。
- 程式設計(program design)泛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。

以汽車駕駛導航為例

- 導航（行車路線） = 演算法（程式的內容，讓電腦運作的處理程序）
- 駕駛人 = 程式設計師
- 駕駛 = 程式設計（使用特定程式語言來寫程式）
- 方向盤、油門 = 程式語言（用來寫程式的語言）
- 汽車 = 電腦

～運算思維，是指為了實現某個目的，以邏輯化的思考方式導出給電腦的正確指令組合。

～即使不會開車，看地圖也能夠指引行車路線一樣。就算不懂程式語言，仍然可能思考用電腦來執行的內容，並以邏輯化的方式書寫、時限內容的處理方法。

～如果提供錯誤的行車路線，電腦會毫不猶豫地依照指令執行錯誤的處理。由於人類無法避免與電腦共處，為了思考與電腦共存的方法，瞭解程式究竟是什麼樣的東西越來越重要。

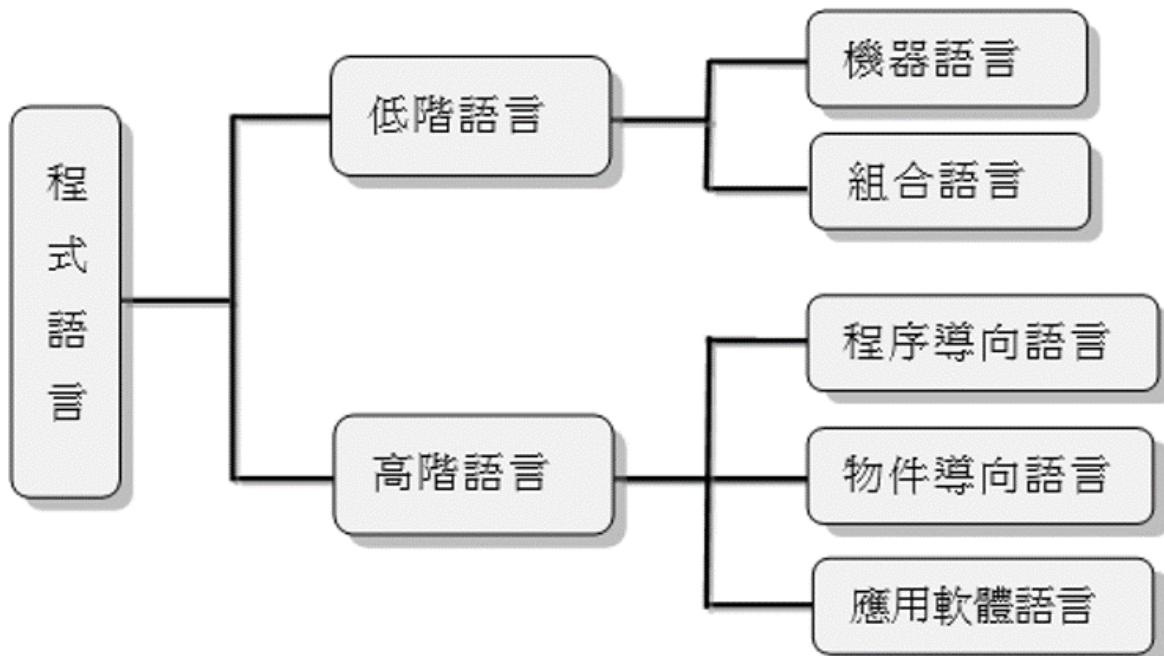
～培養使用程式語言正常動作的程式製作能力，以及不依賴程式語言而是直接思考適用於程式處理的演算法能力。

## 程式語言

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。

## 紀老師的解釋-程式語言

程式語言是指使用者用來與電腦溝通之文字記號所形成的指令集合，也就是電腦能夠接受的語言。就好像人類的語言如中文、英文、法文或西班牙文等，程式語言的種類繁多，而且各有其特定的語法結構與不同的適用範圍。



程式語言依發展的過程可大致分為四類，其中「機器語言」與「組合語言」屬於低階語言，而「程序式語言」與「資料庫查詢語言」屬於高階語言。

(1) 低階語言，早期發展的程式語言是接近機器能辨識的符號，稱為低階語言，包括機器語言與組合語言。低階語言有程式執行速度快、占用記憶體少以及能發揮硬體效能等的優點。不過，低階語言學習門檻高，編寫程式要花費大量的時間，而且程式碼可讀性差不易維護。低階語言是屬於機器導向語言，和機器相依性高。

機器語言 是用0、1二進位的機器指令，來編寫電腦能直接識別和執行的機器碼。由於不需經過翻譯就能直接執行，因此程式執行速度最快。機器語言有可以直接執行、占用記憶體少以及執行速度快的優點。但是使用機器語言編寫程式要花費大量的時間，而且程式碼可讀性差且難維護，所以除了電腦硬體的專業人員外已經少人學習。

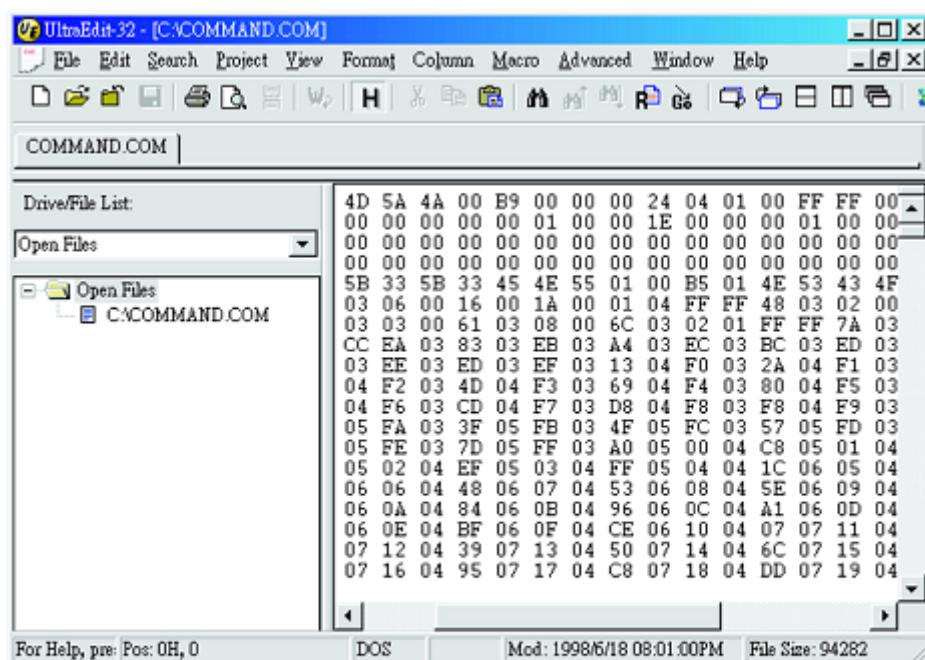


圖6-1.5 藉由文字編輯器Ultra Editor的幫助，將程式檔案command.com以機器語言的模樣顯示。

為了增進與電腦之間的溝通速度，程式設計師們設計出了一種使用英文簡寫來代表各種基本運算的語言，是由字母和數字組成的助憶碼，取代0、1的機器語言符號。例如以 dee 代表「加」，以 fgh 代表「減」，以 ijk 代表「移動」。這些以英文簡寫所構成的語言叫做組合語言。組合語言必須先用組譯器翻譯成機器語言後才能被電腦接受。



↑ 圖6-1.6 組譯器運作示意圖。

(2) 由於低階語言不易學習，所以發展接近人類日常語言的程式語言，稱為高階語言，其語法簡單易懂。執行一行高階語言程式，就等於執行一長串的機器語言程式碼。高階語言具有容易學習、編寫程式較為快速，程式碼具有可讀性高、容易維護和可攜性高等的優點。但是，有程式執行速度稍慢、占用記憶體較多的缺點。

● 機器語言：

```
10001011 01000110 11111100  
10001011 01010110 11111110  
10001001 01000110 00001000  
10001001 01010110 00010000
```

● 組合語言：

```
mov ax,WORD PTR [bp-4];final  
mov dx,WORD PTR [bp-2]  
mov WORD PTR [bp+8],ax;new  
mov WORD PTR [bp+10],dx
```

● 高階語言：

```
new = final;
```

↑ 圖6-1.7 機器語言、組合語言與高階語言。

表6-1.2 低階語言和高階語言的比較

比較項目	低階語言	高階語言
執行速度	快	慢
程式編寫難易	難	易
維護難易度	難	易
跨平台性	低	高
可攜性	低	高

- 程序導向語言是按照程式敘述的先後順序、指令邏輯和流程執行的程式語言。常用的程序導向語言有**BASIC**、**C...**等。
- 物件導向程式設計，是一種以物件為視角，利用多個物件來組成程式。常用的物件導向語言有**Python**、**C++**、**Java**、**C#...**等。
- 應用軟體語言是應用軟體專屬的程式語言，用來擴展該應用軟體的功能，常用的應用軟體語言有**VBA**、**JavaScript**。

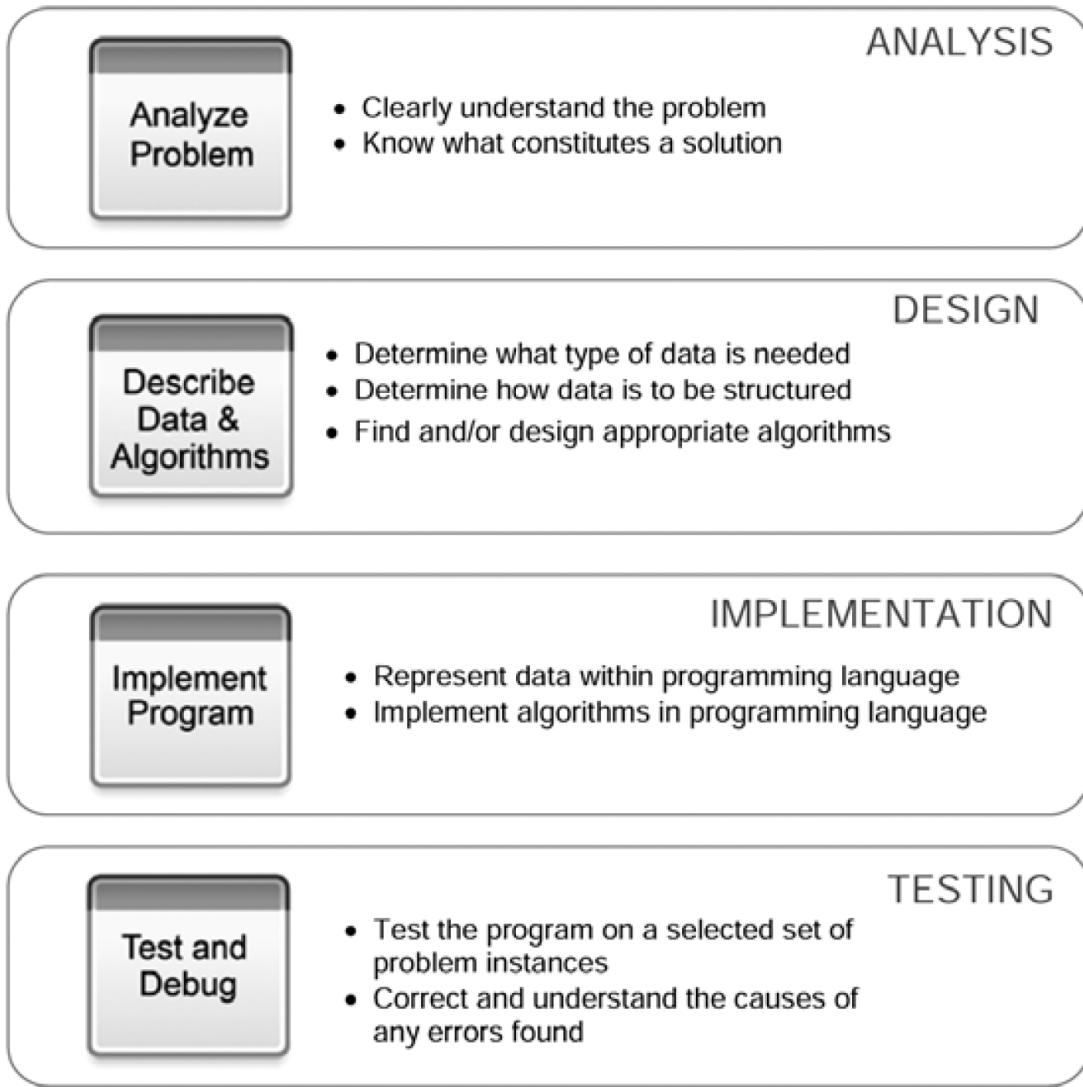
- 查詢語言是以問題為導向，只描述問題不用撰寫解決問題的步驟，可以提高程式設計的產量。例如電子試算表-Excel、資料庫管理查詢系統-Oracle...等。

## 2 用電腦解決問題的步驟

一般人認為「寫程式」就是學會某一種程式語言的指令語法，接著鍵入各式指令到電腦中使電腦工作便算完成程式發展的步驟。事實上，「撰寫程式碼」的工作只是程式發展流程的一部分而已，用電腦解決問題的完整流程如下：

- 階段一：分析問題。
- 階段二：設計解決方案。
- 階段三：撰寫程式。
- 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 問題分析

- 瞭解問題：**Clearly Understand the Problem**

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

- 如何解決問題：**Know what Constitutes a Solution**

Besides clearly understanding a computational problem, one must know what constitute a solution. For some problems, there is only one solution. For others, there may be a number(or infinite) of solutions.

～思考，瞭解問題、定義問題、進行分析。

釐清問題的關鍵與特性，清楚地瞭解問題到底是什麼；找出解決問題的方案，並評估方案的可行性。

- 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，能定義問題，還知道解決問題的方法，皆下來就該思考能不能用電腦處理，以及如何用電腦、讓電腦為我們工作了。

## (2) Program Design 程式設計

描述資料與演算法。

輸入、處理與輸出(**Input-Process-Output Model**，**IPO模式**)

- 輸入 (input)
- 輸出 (output)
- 處理 (process)

列所有可用資源，將實體資訊轉換成計算機可以處理的形式，並簡單描述將可用資源轉換為預期結果的過程，包括資料抽象化與程序抽象化。

## 資料抽象化：**Describing the Data Needed**

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

資料有數字、文字與布林值三種，將這些純量資料彙整，可變成儲存資料的容器。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，以及善用註解、多做說明。

## 程序抽象化：**Describing the Needed Algorithms**

When solving a computational problem, either suitable existing algorithms may be found or new algorithms must be developed.

在設計程式時，除了講究如何得到正確的結果之外，尚須注意所撰寫的程式是否 好的程式結構。一個程式結構的好壞對於整個程式的開發及維護會有很大的影響。

「結構化程式」是從上而下將複雜的程式分解成一個個小程式（即模組），每一個小程式負責執行一件獨立的工作。這些小程式通常只有一個入口及一個出口，使整個程式的結構更加明朗化。因為每個小程式的功能是獨立的，不但有助於偵錯也更容易閱讀及修改。

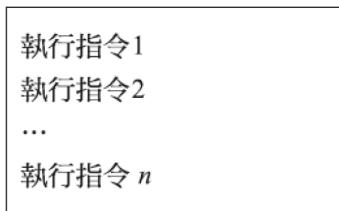
好的演算法應具備五大要件：(1)有限性：必須能在有限的步驟內解決問題。(2)明確性：每一個步驟都必須很明確地表達出來。(3)有效性：必須能在有限的時間內完成。(4)輸入資料：包含零個或一個以上的輸入資料。(5)輸出資料：至少產生一個輸出結果。

每一個小程式或模組通常用下列三種結構來組成，這三種結構稱為程式的基本結構，茲介紹如下：

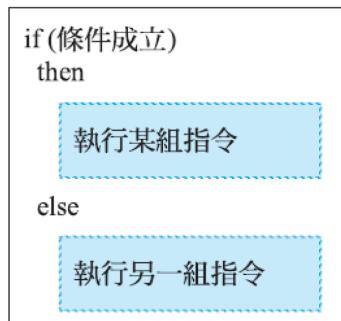
- **序列結構** 是指程式指令由上到下依序執行的結構。
- **決策結構** 則是指依照條件情況執行特定分支指令。
- **重覆結構** 是以迴圈的方式反覆執行某組指令。

演算法通常是由下列三種結構所組成：

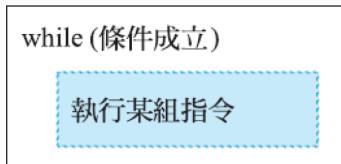
- 序列 (sequence)
- 決策 (decision)
- 重覆 (repetition)



(a) 序列結構



(b) 決策結構



(c) 重覆結構



～本課程以三種資料類型與三種程式結構為骨幹教授運算思維與程式設計。

- 數字資料與序列結構：對話機器人專題
- 文字資料與重覆結構：海龜繪圖專題
- 布林資料與決策結構：遊戲設計專題

#### 舉出三種基本結構的例子

**Q** 想一想，日常生活中有哪些事情符合以上介紹的三種結構？和同學討論一下，並分別舉出一個例子。

**A** 參考答案：

1. 循序結構：到醫院看病時要依序「掛號」→「候診」→「醫生診療」→「開處方」→「批價」→「領藥」。
2. 選擇結構：速食店的員工詢問顧客要點幾號套餐，並依顧客所點之餐號準備相關餐點。
3. 重複結構：電玩遊戲終結時會詢問是否再玩一次，若選擇「是」即可繼續遊戲關卡。

仔細想想，有些事情是不是必須由數種結構組合才能完成？的確如此，事實上寫程式也是類似的狀況。

～ 運算思維是指問題分析和程式設計這兩部分，主要包含定義問題、分析問題與設計演算法。

1. 定義問題：描述問題時要使用明確的語句，清楚地定義出要解決的問題，來方便程式設計者對問題有正確和深入的了解。
2. 分析問題：根據輸出格式的需求規劃出需要輸入的資料，並明確訂定出各種輸出入的限制。
3. 設計演算法：程式設計者根據問題的輸出入需求，規畫解決問題的步驟，每個步驟都必須明確描述。這些解決問題的步驟，就稱為演算法。

～可繪製流程圖或使用虛擬碼呈現運算思維。

### (3) Program Implementation - Code

**Represent data within programming language**

**Implement algorithms in programming language**

Specify which programming language to use, or how to implement the program.

在確定解決方案之後，選擇合適的程式語言並依據解決方案的步驟，逐一將其撰寫成電腦可執行的程式碼。不同的程式語言有不同的語法規則與適用的環境，必須根據需求做出最合適的判斷。

～撰寫程式：依照規劃的演算法，選擇適當的程式語言後，再根據演算法步驟撰寫程式碼。

### 範例：IPO

確認輸出 -> 決定輸入 -> 如何處理，對應程式碼並加註說明



## 「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？" )
```



## 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？" )
```

## 設計良好的程式具備的條件

- 程式的執行結果必須符合預期，而且要正確無誤。
- 程式碼應該具可讀性，而且程式中重要部分需要有詳細的註解說明，以方便日後維護程式。
- 程式應具模組化和結構化，以方便程式的修改、擴充和更新。
- 程式的架構有完整的說明，以及相關的參考技術與使用手冊。
- 程式的執行效率和相容性要高，不會因更換設備而造成錯誤或速度變慢。

## (4) Program Testing - Test and Debug

### **Test the Program on a Selected Set of Problem Instances**

### **Correct and Understand the Causes of Any Errors Found**

針對軟體進行除錯與驗證，確保軟體品質。程式撰寫完成之後仍須不斷測試並修改，以求程式在各種環境下都能順利執行。測試時除了找出程式的語法錯誤之外，也必須找出邏輯與控制上的錯誤。一般我們稱這種除錯的動作為除蟲，因為這些潛藏在程式中的錯誤就像蟲子一樣令人討厭。當程式經過測試及修改後，程式設計師還必須編寫程式相關文件，包括說明操作方式的「使用手

冊」，以及詳記程式邏輯架構與測試結果的「系統文件」。

在程式測試與維護步驟中，包含程式驗證、測試、除錯與維護四大部份。

(1) 在測試(test)時，經由輸入不同的資料，逐一確認結果是否符合預期，以確保程式執行結果都能正確無誤。

(2) 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。程式可能發生的錯誤有三類：語法錯誤、語意錯誤和執行時期錯誤。

## 第一類常見的錯誤：語法錯誤(Syntax Error)

程式有不符合程式語言的文法規則，編譯器會自動檢查出語法錯誤，這是最容易發現的錯誤。

- 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) = 邏輯錯誤

語意錯誤發生時程式仍能執行，只是執行結果不正確；語意錯誤則必須要仔細測試才能發現，因為是邏輯問題。

- 邏輯有問題
- 解決方法：
  - 用除錯器找出錯在哪一行

### 第三類常見的錯誤：執行時期錯誤(**Runtime Error**)

執行時期才發生的錯誤，也是很難發現的錯誤。為避免此類錯誤發生，程式碼必須考慮更周延，並善用例外處理。

～若錯誤訊息看不懂、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）

## Summary - 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.

要用電腦解決問題之前應該多多思考，先瞭解問題，把問題的關鍵表示出來，然後找出解決的方法並進行規劃。之後才是寫撰寫程式實作、測試程式是正常運作。若有問題則找出異常摒除錯，直到程式通過所有測試為止。讓程式正確執行是最重要的第一步，其次才是寫出結構化、執行有效率的程式。

