

入門概論 - 運算思維與程式設計（二）

1 用電腦解決問題的步驟

一般人認為「學寫程式」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。

前兩個階段是運算思維，第三個階段是大家認為的程式設計，第四個階段是確認寫的程式正確無誤、並且能解決問題。

The Process of Computational Problem Solving

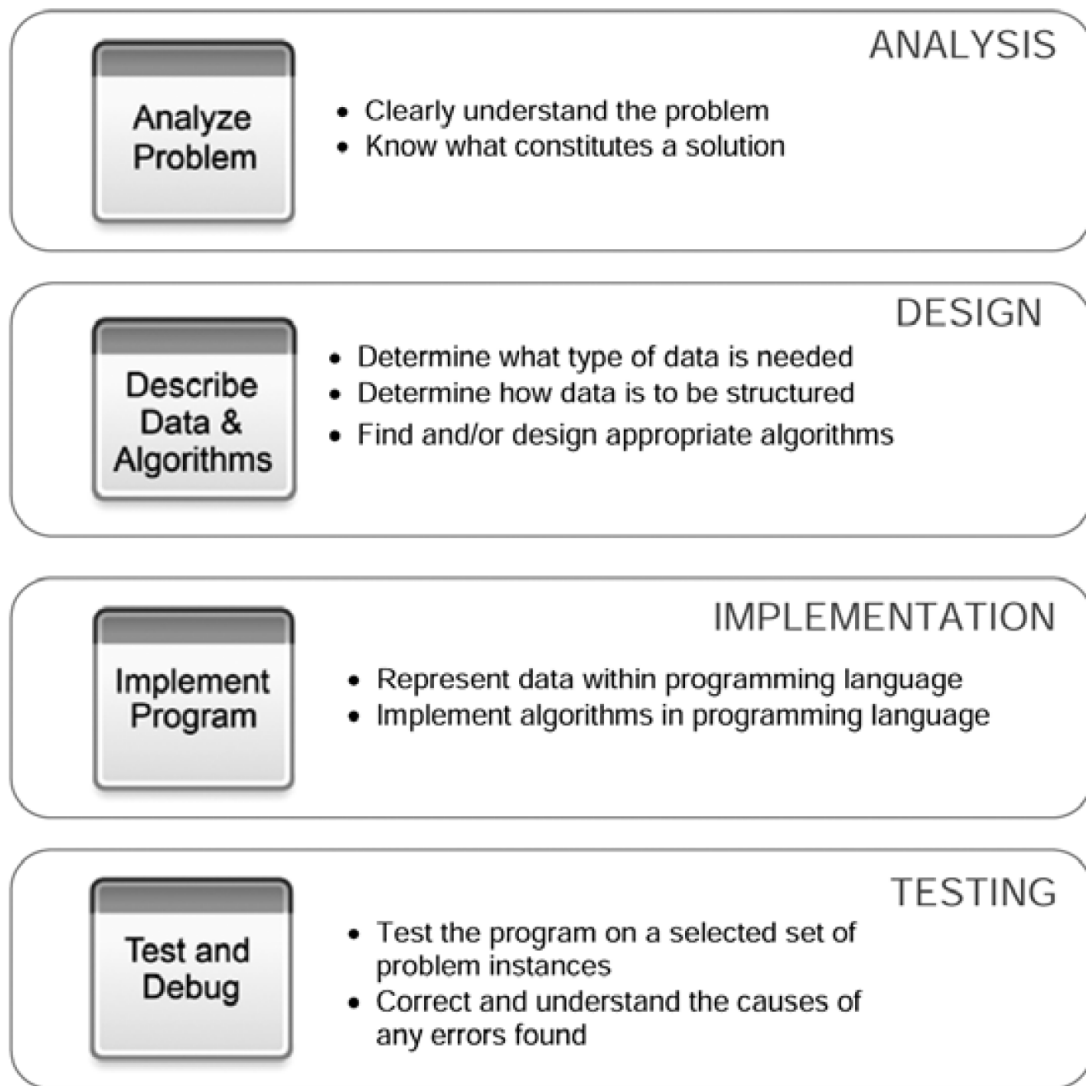


FIGURE 1-22 Process of Computational Problem Solving

(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.

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。

- 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、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。

～先瞭解問題是什麼、要如何解決，想清楚描述問題和解決問題的方法。然後判斷一下這個問題是否可以利用電腦來解決，以及要如何用電腦處理。

(2) Program Design 程式設計

描述資料與演算法。在這個階段不用寫程式、寫任何指令，但是要思考這個問題如果用電腦來處理，要怎樣落實解決問題的方法呢？也就是說，你還不用開車，但要做出個導航的路線圖一樣！

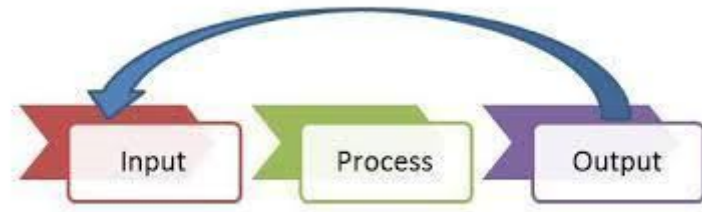
(1)輸入、處理與輸出(Input-Process-Output Model，IPO模式)

程式設計可以分成三大部分思考：

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

電腦運作的方式：程式經由輸入資料，加以處理產生結果後，再行輸出。在這些步驟中可能產生錯誤，此時必須加以除錯，再重新執行，直到產生正確結果為止。

- 輸入的資料要正確，才能避免垃圾進垃圾出的錯誤結果。
- 輸出結果首先要求正確，其次則是美觀。
- 處理過程則是程式的核心。



撰寫解決問題的程式時，需要輸入資料，然後處理資料，最後輸出結果。

(2) 在電腦處理的過程，程式設計包括資料描述與演算法兩大部份。

列所有可用資源，將實體資訊轉換化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。

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.

資料抽象化：Describing the Data Needed

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

- Determine what type of data is needed.
- Determine how data is to be structures.

當確認問題可以用電腦解決之後，我們需要有代表問題中項目的變數名稱，才好指派資料給變數。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.

- Find and/or design appropriate algorithms.

三種程式結構

有了資料之後，就需要利用演算法來解決問題，因而形成不同的流程控制。

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是好的程式結構。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。通常，我們希望能「結構化程式」。也就是從上而下將複雜的程式分解成一個個小程序（即模組），每一個小程序負責執行一件獨立的工作。所以，這些小程序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。因為每個小程序的功能是獨立的，不但有助於偵錯也更容易閱讀及修改。

每一個小程序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。

(1)序列結構Sequence：是指程式指令由上到下循序執行的結構。

程式由上而下，依序一行一行逐行執行。第一行執行完畢後執行第二行，第二行執行完畢後執行第三行，直到程式執行結束。

(2)重覆結構Repetition：是以迴圈的方式反覆執行某組指令。

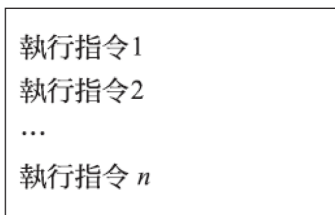
又稱迴圈（**Loop**），是指部分程式片段可重複執行多次，直到某測試的條件發生為止。例如：求 $1+2+3+\dots+1000$ ，使用重複結構可在很短時間內重複執行相加的程式，直到求出加總的結果。

(3)決策結構Decision：是指依照條件情況選擇執行特定分支指令。

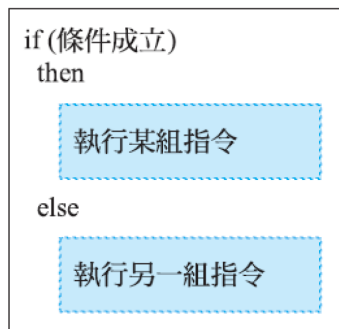
或稱決策（**Decision**），如果條件成立就執行某段程式碼，不成立則執行另一段程式碼。例如：若成績大於等於60分，則輸出及格，否則輸出不及格。

演算法通常是由下列三種結構所組成：

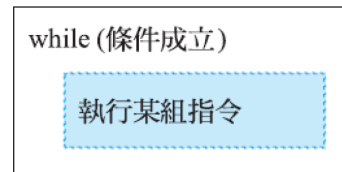
- 序列 (sequence)
- 決策 (decision)
- 重覆 (repetition)



(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。

～運算思維與程式設計的核心為演算法

- 瞭解並分析程式需求：輸入和輸出的訊息是什麼（切分成小任務＋資料抽象化）
- 處理程式結構：序列、決策、重覆（切分成小任＋程序抽象化）
- 設計測試案例：預想測試資料及相應結果（用例子確認處理邏輯）
- 規劃解決步驟：需對輸入的資訊做什麼操作或運算（模式辨識＋演算法）

(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：有系統地依照輸入-處理-輸出的步驟撰寫程式碼

- 確認輸出：「螢幕輸出格式」是什麼。
- 決定輸入：依照輸出的需求規劃出需要輸入的資料。
- 撰寫程式：根據規劃的演算法，選擇適當的程式語言，依照流程和語法撰寫程式碼。（記得加註：可先用虛擬碼寫運算思維，再對程式結構和語法做說明。）



「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？" )
```



開始撰寫程式碼...



- 規劃 → 程式碼

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

• # → 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，以確保程式都能正確無誤地執行結果。

(2) 在除錯(debug)時，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。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。

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。

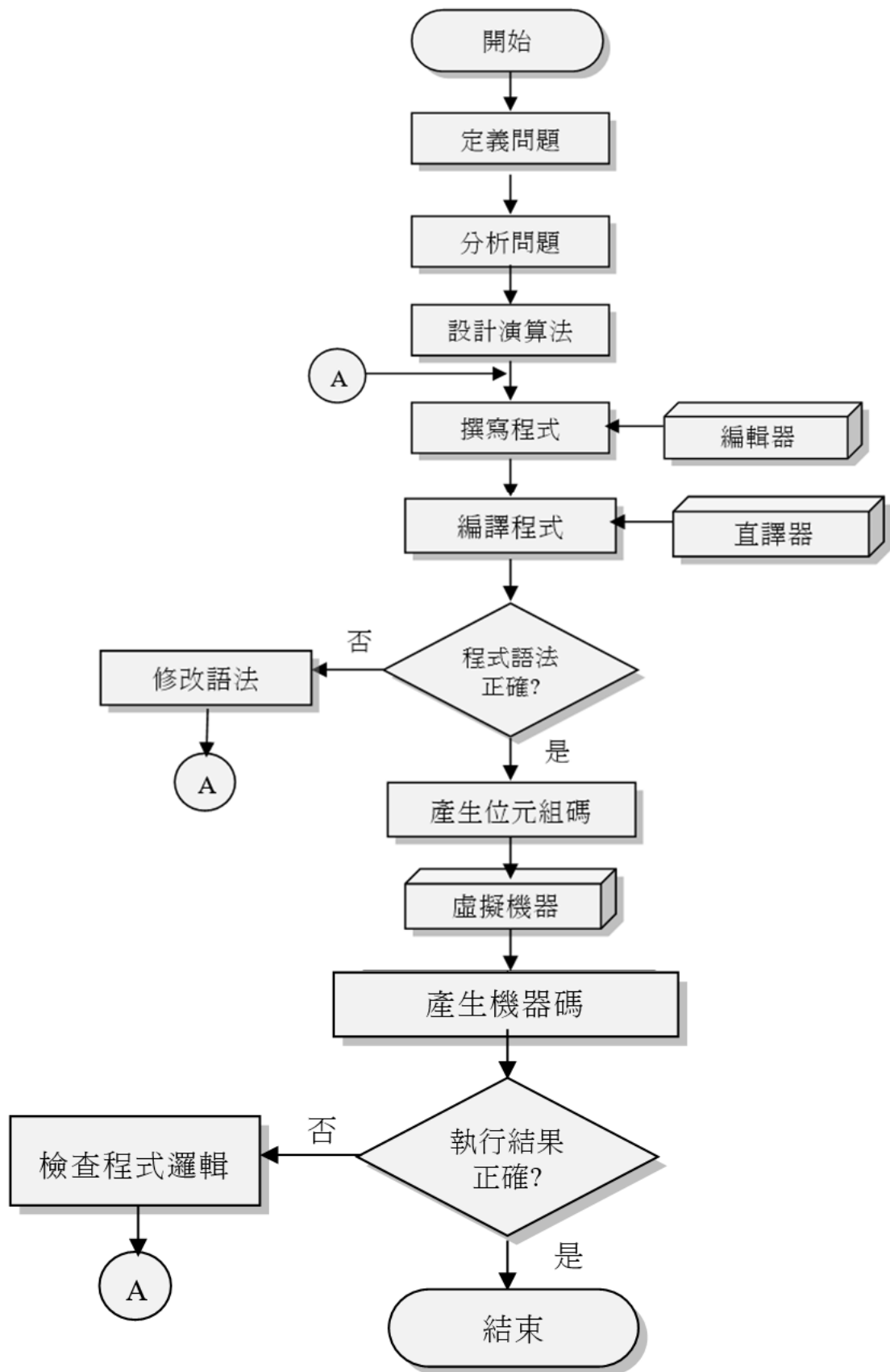
～若錯誤訊息看不懂、指令用法不瞭解、不知道下一步做什麼

- 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

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。



- 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。
- 學習程式語言不能只是盲目學語法，有效拆解問題才是程式設計的基石。

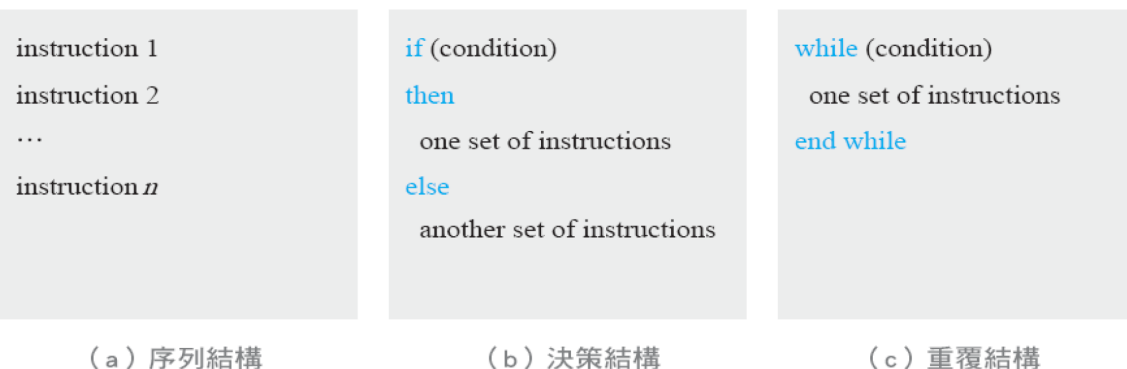
- 學習程式設計不該只是盲目死記難懂的指令及符號，而是培養邏輯、勇敢嘗試、並實現創意的過程。

2. 流程圖與虛擬碼

程式是書寫要讓電腦執行的處理，也就是寫出程式設計師腦中思考的東西。有經驗的程式設計師會撰寫「文字式的虛擬碼」或繪製「視覺化的流程圖」(flow chart)，幫助我們思考如何解決問題，並呈現程式設計的邏輯與步驟。

2-1 虛擬碼

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。



虛擬碼是一種類似自然語言的程序語言，是一種介於一般語言與程式語言之間的語言碼。虛擬碼以文字方式來描述電腦處理問題的步驟，所發展出一套簡化且具結構化的描述語言，可以很方便地轉換成各種程式語言。有些程式設計師不喜歡繪製流程圖，而用很像程式碼的虛擬碼來表示程式的邏輯架構與執行程序。如圖所示，這些虛擬碼多半使用英文來表示，不過也會有人使用中文來表示。使用虛擬碼的最大好處是，它很像在寫程式，這對於整天都在寫程式的程式設計師來說是最直接的一種思考方式。

實例：為四人餐桌百放碗盤的虛擬程式碼。

以下重複四次

- 1 前往餐桌下一個空位
- 2 放一個碗在此處
- 3 在碗左側放一張餐巾
- 4 在碗右側放一把湯匙

讓思維條理化

人的思維可能比想像中更沒有邏輯，製作程式時必須以邏輯化的方式思考。這時要釐清腦中模糊的想法，先從讓腦中的思緒調理化開始。

人們透過過去的經驗，「理所當然」地處理各式各樣的情況，而這些未被明確定義的表現形態的往往含有「模糊曖昧」之處。電腦無法理解這些「模糊曖昧」，所以要先去除電腦所無法處理的「模糊曖昧」，讓一無所知的電腦能夠處理的「理所當然」。

例：煮飯，用家裡的電子鍋煮飯的演算法。

～ 請挑出下面的「模糊曖昧」。

- Step1: 開始
- Step2: 將生米放入內鍋
- Step3: 洗米
- Step4: 加水
- Step5: 把內鍋放進電子鍋
- Step6: 設定煮飯模式
- Step7: 按下「炊飯」鍵
- Step8: 結束

例：早上出門時，天空看起來怪怪的，要不要帶傘？

～電腦要能替代人做判斷，就得把「看起來怪怪」的模糊判斷決策模式，改成有明確定義規則的方式。

問題1: 如果要依循規則來決定帶傘或不帶傘，應該如何思考比較好？依據降雨率，決定要不要帶傘出門。

解：單純地思考，降雨率超過「某數值」就帶傘，不到「某數值」不帶傘，比如降雨率為50%。

- Step1: 決定以降雨率50%為帶不帶傘的準則。
- Step2: 查看網路或手機確認降雨機率。
- Step3: 降雨機率超過50%就帶傘。

像這樣以基準值判定的決策模式，是電腦可以處理的判定方式。

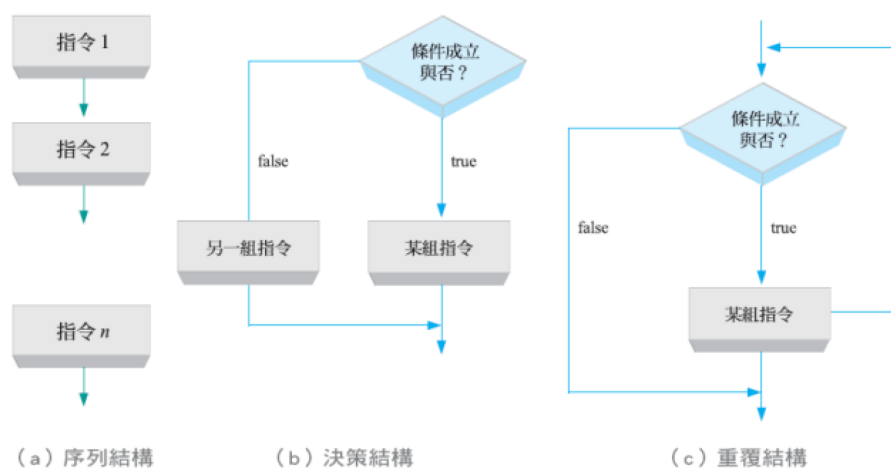
問題2：現在有長傘和折疊傘。想盡量減少隨身物品，如果感覺不大會下雨，就想帶折傘。如果要依據規則來判定，那應該用什麼基準做決策？

解：若降雨機率超過30%，不超過60%，帶折傘。若超過60%，帶長傘。

- Step1: 決定降雨率超過30%、不到60%，就帶折傘。
- Step2: 決定降雨率超過60%，就帶長傘。
- Step3: 查看網路或手機確認降雨機率。
- Step4: 降雨機率超過60%就帶長傘。
- Step5: 降雨機率超過30%就帶折傘。

2-2 流程圖

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。













「流程圖」是使用各種不同的圖形、線條及箭頭來描述問題的解決步驟以及進行的順序。也就是指以各種特定的幾何圖形符號來表示演算法，藉以說明處理方法與步驟的一種特定圖表，它擁有以下優點：

- 易於把握重點，減少錯誤
- 有助於偵錯
- 有助於日後程式之修改
- 有助於程式的順利完成

即使不懂程式語言，只用簡單的符號，也能近似實際程式的結構來寫程式處理。流程圖無可避免地比用程式語言粗略，但運用流程圖作為寫程式「之前一步」，對替代程式設計來說非常有用。

流程圖目前採用美國國家標準學會(ANSI)於1970年公佈的流程圖符號。常用的有四種：橢圓=開始或結束、長方形=程序處理、平行四邊形=資料輸入輸出、菱型=選擇決策。

符號	功能	符號	功能
	開始 / 結束		判斷比較
	程序處理		工作流向符號
	輸入或輸出		預設處理作業
	連結符號		文件
	儲存資料		磁碟

知道能夠以流程圖這樣的形式寫出適用於程式的處理，就可以用這種方式來讓腦中的思維和解決方案條理化。

～一個繪製良好的流程圖，必須符合下面原則：

- 流程圖必須使用標準符號，以方便共同閱讀和研討分析。
- 繪製流程圖的方向應由上而下，自左到右。
- 流程圖中的文字說明要力求簡潔，而且要明確具體可行。
- 流程圖中線條應該避免太長或交叉，此時可以使用連接符號。

範例：計算十個整數的平均數

撰寫「計算十個整數之平均值」的演算法

Q 首先，想一想「計算十個整數之平均值」應該如何做？

A 求一列整數的平均值是以該列整數的和除以該列整數之個數，配合一般程式語言語法，一個演算法如下列步驟：

第一步：程式開始

第二步：讀入一個整數

第三步：將這個整數的值加入總和變數

第四步：判斷是否已讀入10個整數了？

若已經讀入10個整數則跳到第五步

否則跳到第二步（再讀入一個整數）

第五步：計算平均值（將總和變數除以10）

第六步：印出平均值

第七步：程式結束

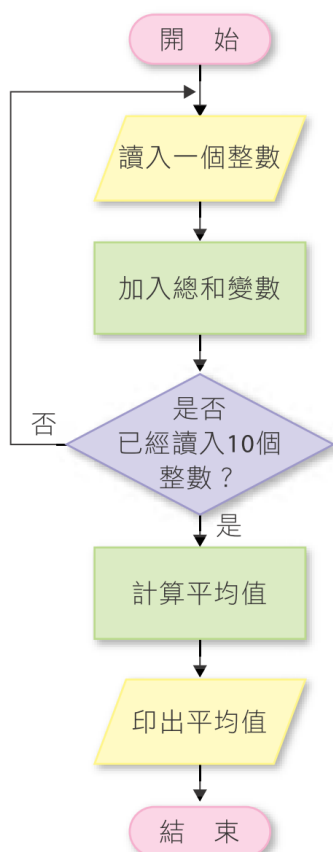
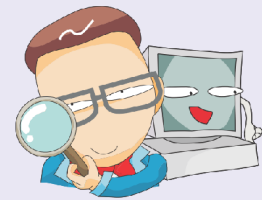


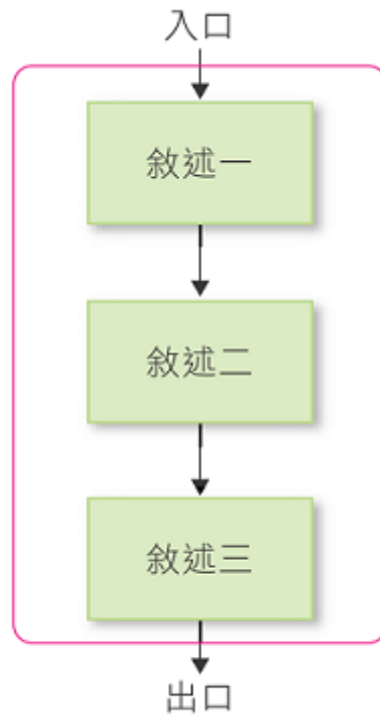
圖6-1.2 「計算10個整數之平均值」的流程圖。

- 1： 程式開始
- 2： 讀入一個整數
- 3： 加入總和變數
- 4： 若已經讀入10個整數
- 5： 計算平均值
- 6： 印出平均值
- 7： 否則
- 8： 回到第2步（讀入整數）
- 9： 結束判斷
- 10： 程式結束

圖6-1.3 以虛擬碼表示「計算10個整數之平均值」。

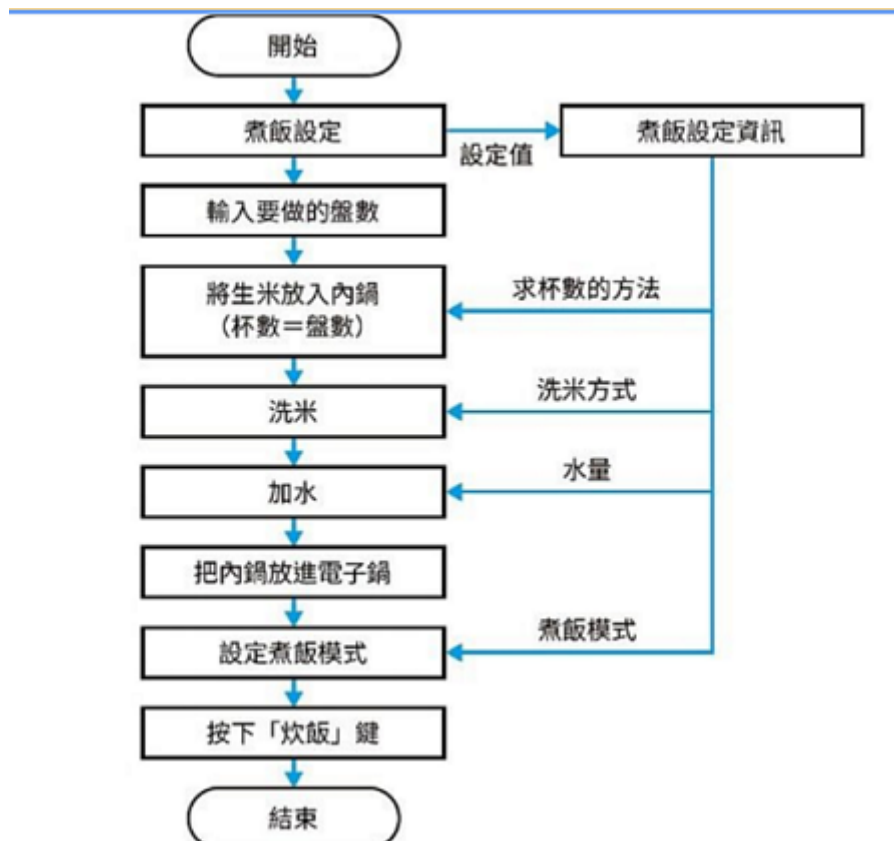
2-3 三種程式結構（後面章節會教）

～循序處理：由上往下



↑ 圖6-1.9 循序結構。

例：煮飯，用家裡的電子鍋煮飯。請畫流程圖。



～選擇結構-條件決策

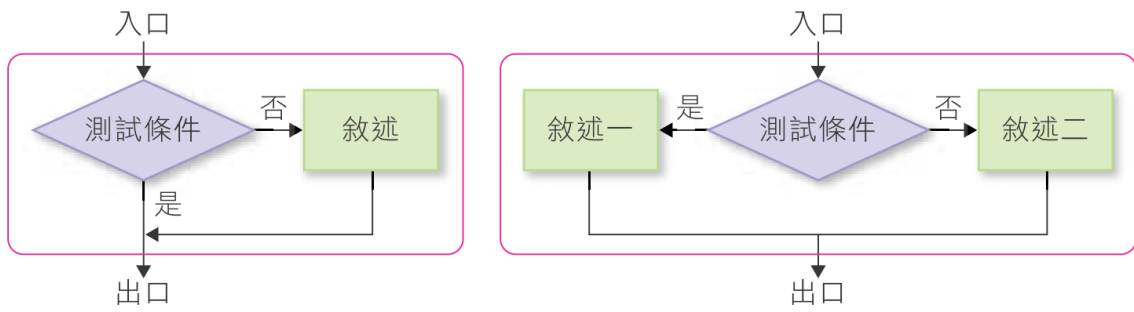


圖6-1.10 選擇結構。

～重複結構-反覆執行

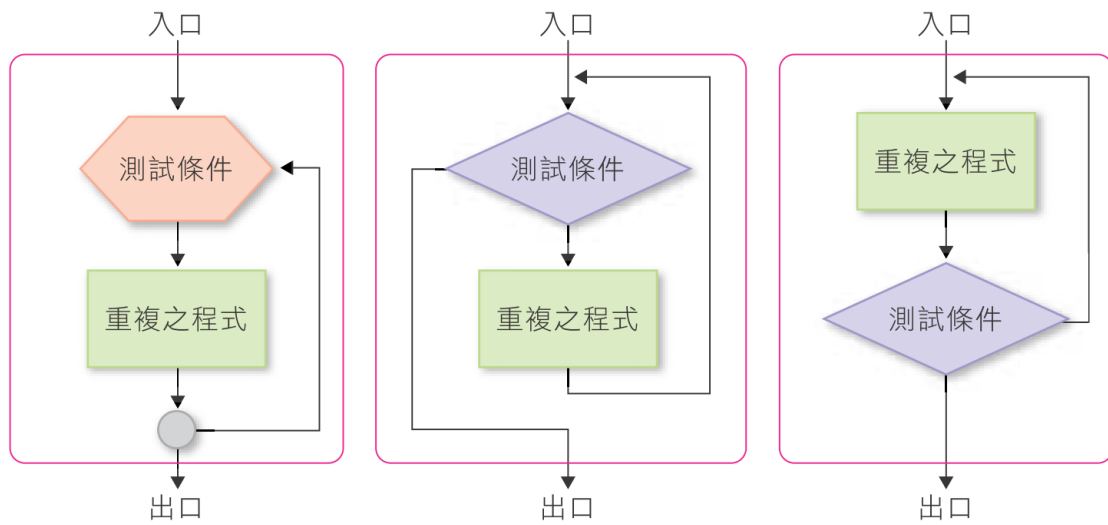
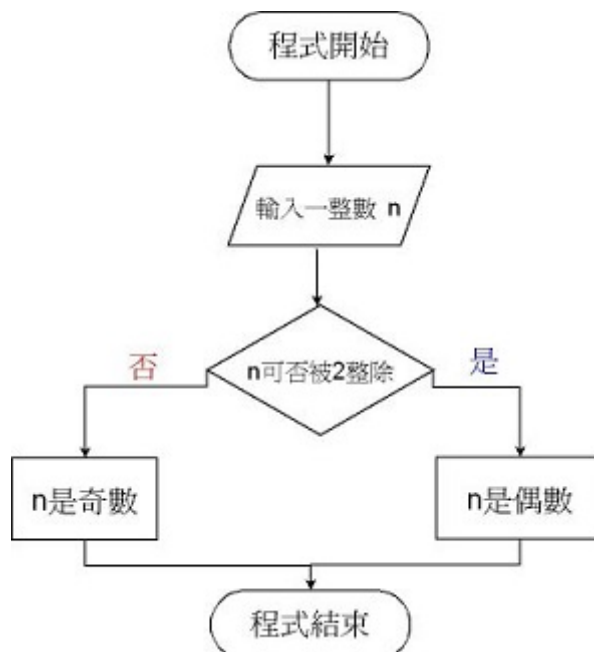
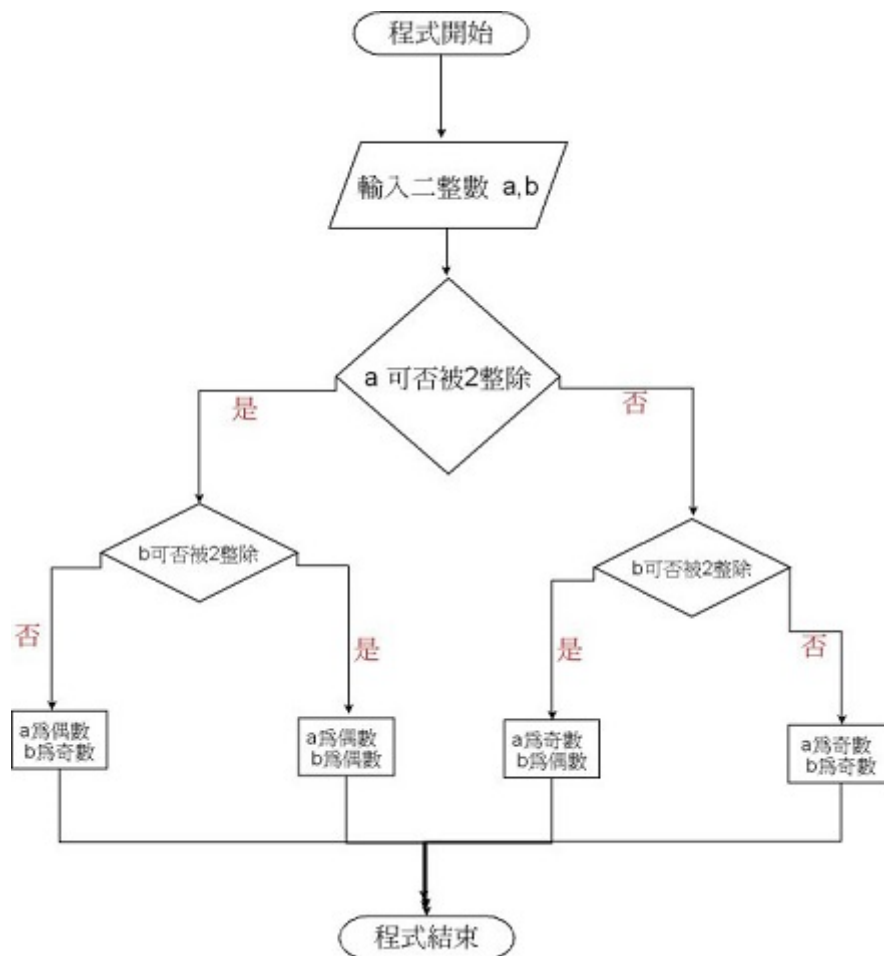


圖6-1.11 重複結構。

【隨堂練習1】輸入一正整數，判斷為奇數或偶數。用流程圖方式表示。

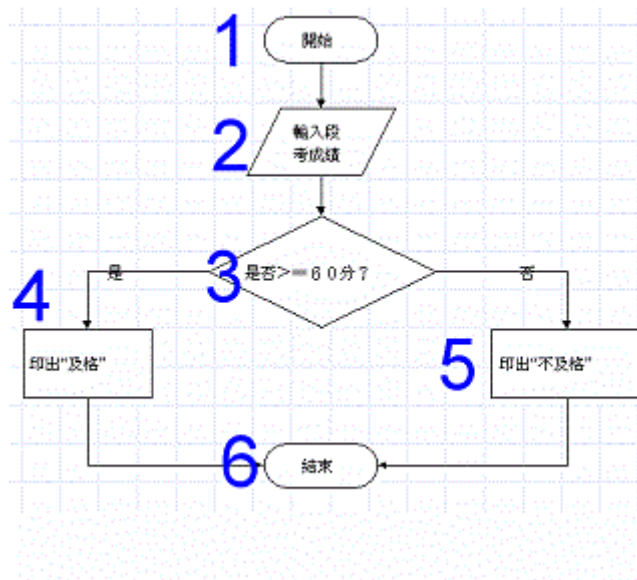


【隨堂練習2】輸入兩正整數，判斷此二數是否為一個奇數與一個偶數，用流程圖表示。

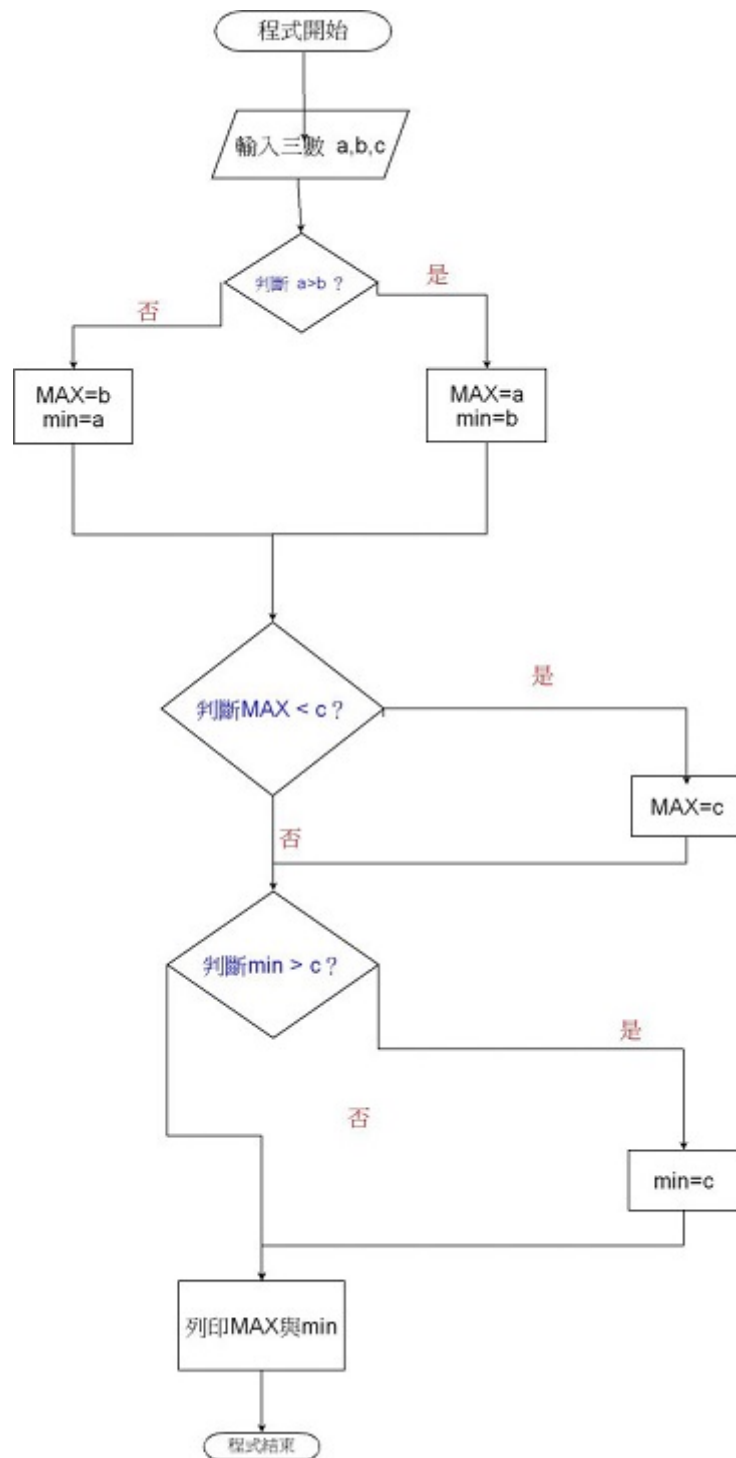


【隨堂練習3】輸入一學生成績，由程式判斷若成績在60分以上，請輸出“及格”，反之則為“不及格”。用流程圖方式表示。

- Step 1、開始
- Step 2、取得某科的段考成績
- Step 3、判斷是否 ≥ 60
- Step 4、若是，則印出“及格”
- Step 5、若否，則印出“不及格”
- Step 6、結束



【加分題】 輸入三數，找出最大值與最小值。



～ 要解決一個問題可能有很多種演算法可以處理，可以從中選擇較佳的演算法，以提高程式的執行效率。要判斷演算法的好壞，原則上是以操作步驟越少越好，以及使用的記憶體空間越少越好。