



# 運算思維、程式設計與程式邏輯

- 運算思維的基礎 - 五大領域
- 程式設計是什麼
- 演算法
- 演算法思考
- 程式邏輯的基礎
- 流程圖
- fChart程式設計教學工具
- 結構化程式設計





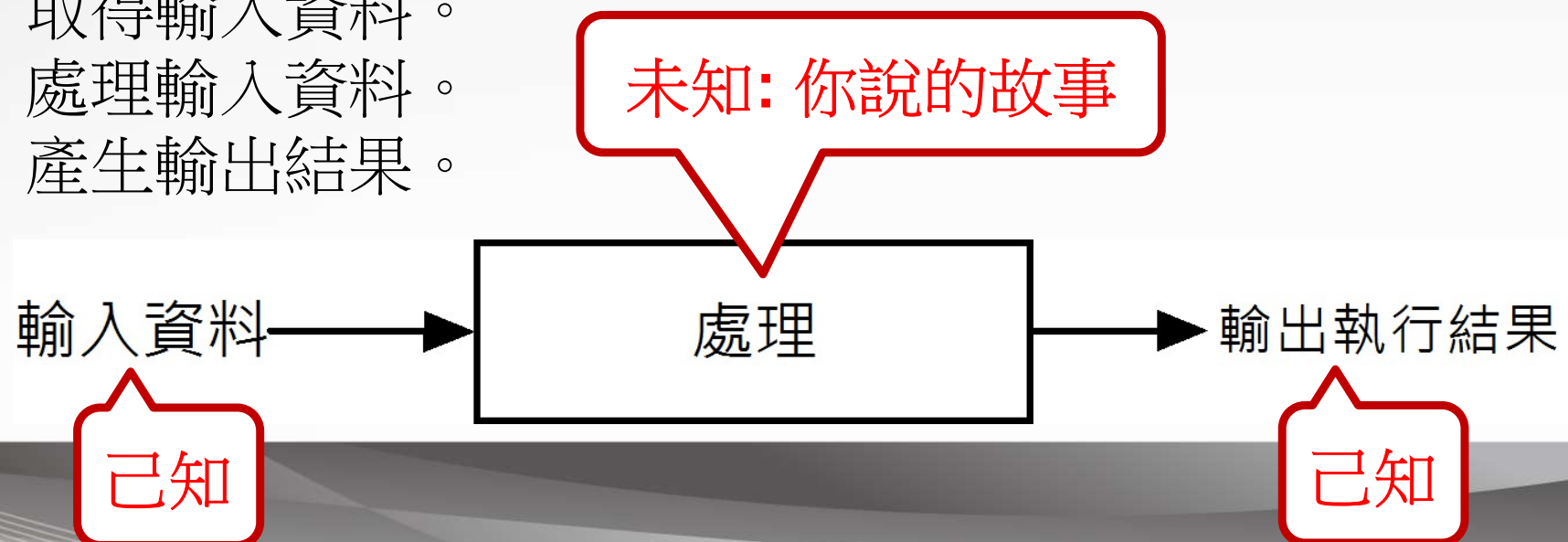
# 運算思維的基礎 - 五大領域

- **抽象化 (Abstraction)**：思考不同層次的問題解決(複雜問題簡化成核心資訊，將皮球抽象化成一個球)。
- **演算法 (Algorithms)**：將解決問題的工作思考成一序列可行且有限的步驟。
- **分割問題 (Decomposition)**：了解在處理大型問題時，我們需要將大型問題分割成小問題的集合，然後個個擊破來一一解決。
- **樣式識別 (Pattern Recognition)**：察覺新問題是否和之前已解決問題之間擁有關聯性，以便可以使用已知或現成的解決方法來解決問題。
- **歸納 (Generalization)**：了解解決的問題可能是用來解決更大範圍問題的關鍵之一。



# 程式設計是什麼 - 使用程式邏輯說故事

- 我們目前學的程式設計主要是指結構化程式設計，就是使用程式的邏輯說一個故事，可以從程式的輸入逐步依故事的流程轉換成最後程式的輸出結果。而說故事的流程就稱為演算法。
- 事實上，幾乎所有程式都可以簡化成三種基本元素，如下所示：
  - 取得輸入資料。
  - 處理輸入資料。
  - 產生輸出結果。





# 演算法 - 說故事的流程

- 如同建設公司興建大樓有建築師繪製的藍圖，廚師烹調有食譜，設計師進行服裝設計有設計圖，公司行號的標準作業程序SOP，程式設計也一樣有藍圖，哪就是演算法。
- **演算法（Algorithms）**簡單的說是一張食譜（recipe），提供一組一步接著一步（Step-by-step）的詳細過程，包含動作和順序，可以將食材烹調成美味的食物來完成工作，例如：製作蛋糕的食譜是一個演算法，如下圖所示：

演算法

=

一張食譜

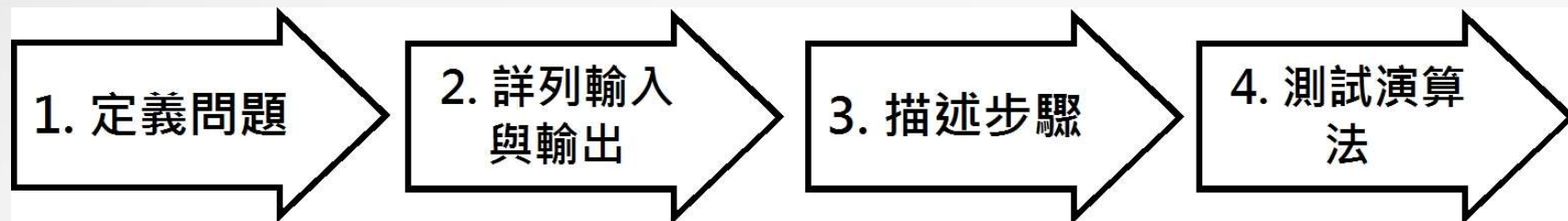
=

一組指令步驟



# 演算法 - 設計演算法的步驟(圖例)

- 因為演算法是描述解決問題的步驟，如同蓋房子的藍圖，在真正實際撰寫程式碼之前，我們需要先設計演算法，其基本步驟如下圖所示：





# 演算法 - 設計演算法的步驟(說明)

- **Step 1. 定義問題**：使用明確和簡潔的詞彙來描述欲解決的問題。
- **Step 2. 詳列輸入與輸出**：列出欲解決問題的資料（Input），和經過演算法運算後，需要產生的結果（Output）。
- **Step 3. 描述步驟**：描述從輸入資料轉換成輸出資訊的步驟。
- **Step 4. 測試演算法**：使用測試資料來驗證演算法是否正確。



# 演算法 - 表達方法

- 演算法的表達方法是在描述解決問題的步驟，所以沒有固定方法，常用表達方法，如下所示：
  - **文字描述**：直接使用一般語言的文字描述來說明執行步驟。
  - **虛擬碼（Pseudo Code）**：一種趨近程式語言的描述方法，並沒有固定語法，每一行約可轉換成一行程式碼。
  - **流程圖（Flow Chart）**：使用標準圖示符號來描述執行過程，以各種不同形狀的圖示表示不同的操作，箭頭線標示流程執行的方向。





學習程式設計就是希望你能  
成為一位演算法思考者，擁  
有解決問題的能力。

## 演算法思考

- **演算法思考(Algorithmic Thinking)**是一種能力來了解、執行、評估和設計演算法，**演算法思考者(Algorithmic Thinker)**是一位有能力執行演算法思考的人。因為演算法思考能力是可以轉移的，如果你有能力設計解決程式問題的演算法，你就一定有辦法分析和解決其他各種領域的問題。
- 請注意！演算法思考在設計演算法時需要先確認執行此演算法的**目標執行者(Target Executer)**是誰？你需要以目標執行者能夠了解和執行的步驟來設計與描述演算法步驟，不然，就算你設計好演算法步驟，目標執行者依然沒有能力依據步驟來完成工作。





# 演算法的目標執行者

- 目標執行者(Target Executer)決定設計演算法時每一步驟的指令描述，例如：飛機起飛前都有一個起飛前檢查清單(Preflight Checklist)，其目標執行者的機長，他看的懂，如果讓你去執行，你連指令都看不懂，更不說如何執行它。
- 所以，針對程式問題設計的演算法，其目標執行者是電腦，電腦很笨(只是一台運算超快的計算機)，你需要以電腦可以執行的步驟來設計演算法，這就是**程式邏輯(Program Logic)**。



# 程式邏輯的基礎-說明

- 我們使用程式語言的主要目的是撰寫程式碼建立應用程式，所以需要使用電腦的**程式邏輯**（**Program Logic**）來寫出程式碼，如此電腦才能執行程式碼解決我們的問題。
- 讀者可能會問撰寫程式碼執行**程式設計**（**Programming**）很困難嗎？事實上，如果你可以一步一步詳細列出活動流程、導引問路人到達目的地、走迷宮、從電話簿中找到電話號碼或從地圖上找出最短路徑，就表示你一定可以撰寫程式碼。



# 程式邏輯的基礎-問題

- **問題：**開車從高速公路北上到台北市大安森林公園，在Google地圖顯示圓山交流道至大安森林公園之間的地圖，如右圖所示：





# 認識程式邏輯-人類邏輯(目標執行者：人)

■ **人類邏輯**：我們只需檢視地圖，即可輕鬆寫下開車從高速公路北上到台北市大安森林公園的步驟(演算法)，如下所示：

- Step 1：中山高速公路向北開。
- Step 2：下圓山交流道（建國高架橋）。
- Step 3：下建國高架橋（仁愛路）。
- Step 4：直行建國南路，在紅綠燈右轉仁愛路。
- Step 5：左轉新生南路。



# 認識程式邏輯-電腦邏輯(目標執行者：電腦)

■ **電腦邏輯**：在人類邏輯的步驟描述需要提供更多資訊給電腦（請改用電腦邏輯來思考），才能讓電腦開車到達目的地，如下所示：

- 從哪裡開始開車（起點），中山高速公路需向北開幾公里到達圓山交流道。
- 如何分辨已經到了圓山交流道？如何從交流道下來？
- 在建國高架橋上開幾公里可以到達仁愛路出口，如何下去。
- 直行建國南路幾公里可以看到紅綠燈？左轉或右轉？
- 開多少公里可以看到新生南路，如何左轉？接著需要如何開？如何停車？

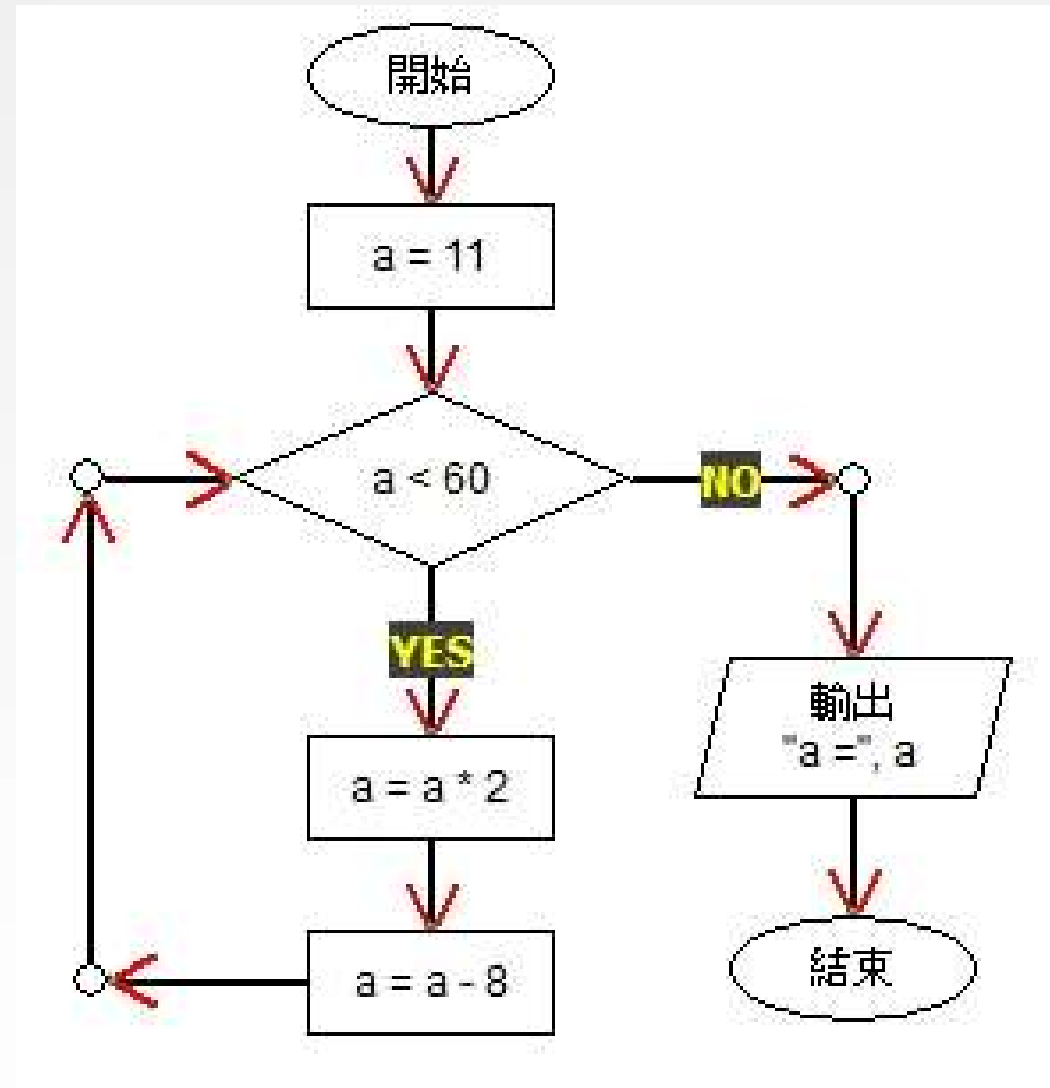
撰寫程式碼時需要告訴電腦非常詳細的動作和步驟順序，如同教導小孩作一件從來沒有作過的事，例如：綁鞋帶、去超商買東西或使用販賣機。





# 流程圖-說明

- 不同於文字描述或虛擬碼是使用文字內容來表達演算法，流程圖是使用簡單的圖示符號來描述解決問題的步驟。







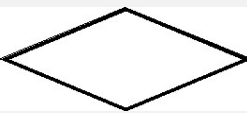
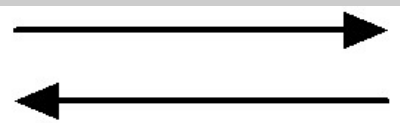




# 流程圖-認識流程圖

- 流程圖是使用簡單的圖示符號表示程式邏輯步驟的執行過程，提供程式設計者一種跨程式語言的共通語言，作為與客戶溝通工具和專案文件，如果我們可以畫出流程圖的執行過程，就一定可以將它轉換成指定程式語言。
- 就算是一位沒有寫過程式碼的初學者，也一樣可以使用流程圖來描述執行過程，以不同形狀的圖示符號表示操作，在之間使用箭頭線標示流程的執行方向，筆者稱它為圖形版程式（對比程式語言的文字版程式）。
- 目前演算法使用的流程圖是由**Herman Goldstine**和**John von Neumann**開發與製定。



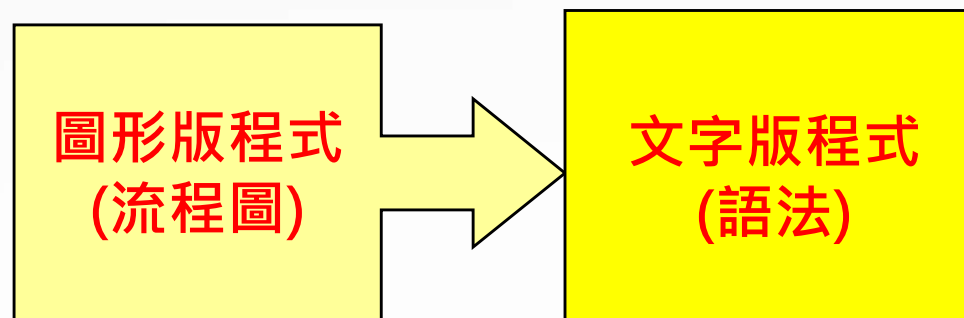
# 流程圖-流程圖的符號圖示

流程圖的符號圖示	說明
	長方形的【動作符號】（或稱為處理符號）表示處理過程的動作或執行的操作
	橢圓形的【起止符號】代表程式的開始與終止
	菱形的【決策符號】建立條件判斷
	箭頭連接線的【流程符號】是連接圖示的執行順序
	圓形的【連接符號】可以連接多個來源的箭頭線
	【輸入/輸出符號】（或稱為資料符號）表示程式的輸入與輸出



# fChart流程圖直譯工具 - 程式邏輯訓練

- fChart是一套流程圖直譯工具，不只可以繪製演算法的流程圖，它更是一個**可執行**的流程圖，能夠訓練學生邏輯思考和追蹤程式執行的能力，在完成程式邏輯訓練後，才開始學習程式語法。
- 因為所有程式語言的程式碼都是源於流程圖，如果能夠繪出解決問題的流程圖，就等同寫出圖形版程式，我們可以輕鬆將流程圖轉換成各種語言的程式碼。





# fChart程式設計教學工具

- <https://fchart.github.io>
- <http://fchart.is-best.net>

fChart程式設計教學工具

關於 流程圖直譯器 程式碼編輯器 Arduino版 Python版 Node版 ArduBlockly Micro:bit 文件與下載

## 台灣的下一步，就從學 **Coding** 開始

「輕鬆使用 **fChart** 流程圖和 **Blockly** 積木學習基礎程式設計」

fChart 程式設計教學工具 - 學 Coding 的好幫手

fChart 分類150個流程圖專案 - 程式邏輯訓練的好幫手

下載 fChart6 標準版   fChart6使用手冊   分類150個流程圖專案



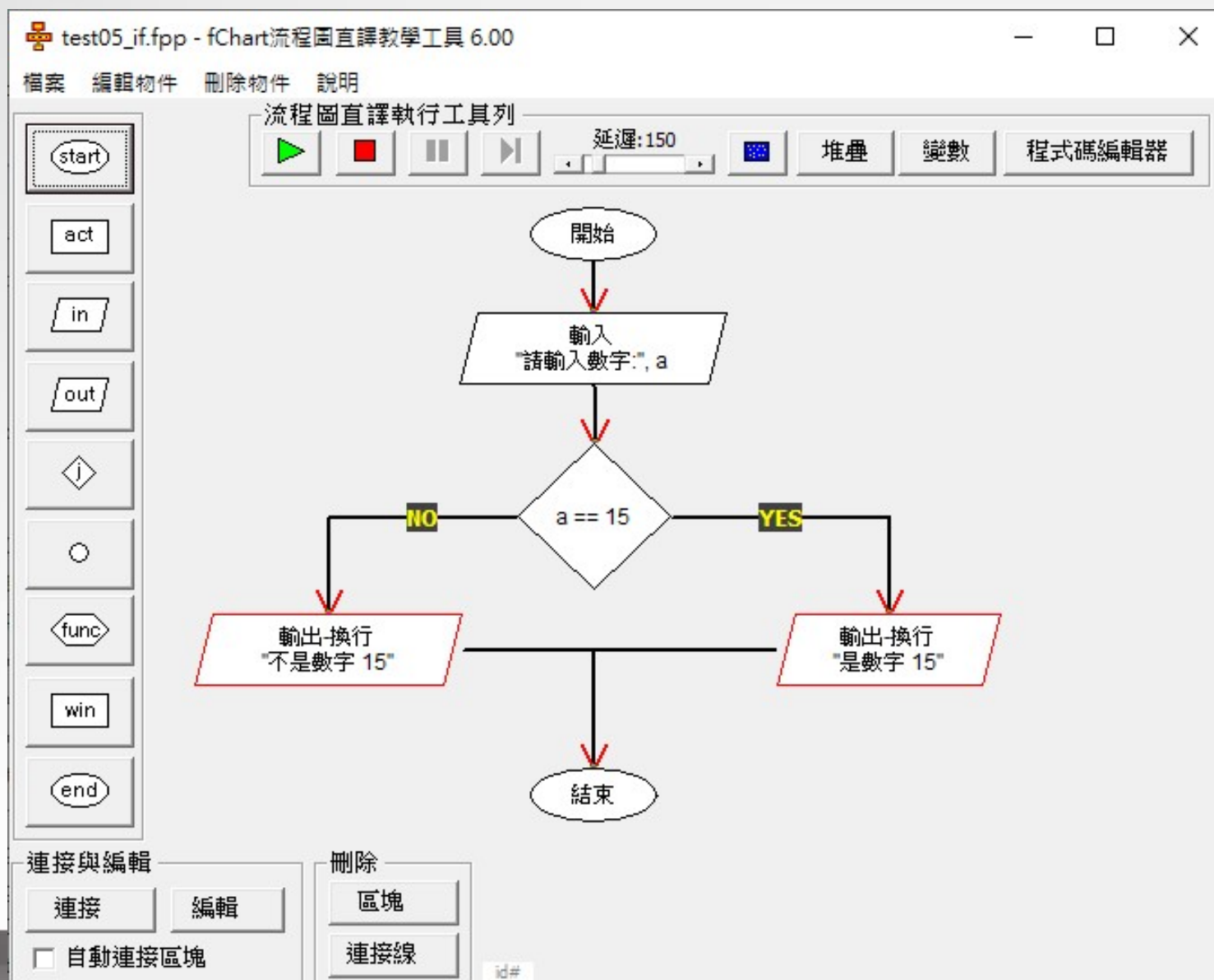
# fChart程式設計教學工具 – 分類範例

■ <https://github.com/fchart/fChartExamples2>

fchart Update README.md		Latest commit 23b90ef 11 days ago
01.變數與輸出輸入	Add files via upload	27 days ago
02.運算子與運算式	Add files via upload	27 days ago
03.條件判斷	Add files via upload	27 days ago
04.迴圈結構	Add files via upload	27 days ago
05.迴圈和條件	Add files via upload	27 days ago
06.巢狀迴圈	Add files via upload	27 days ago
07.函數	Add files via upload	27 days ago
08.陣列	Add files via upload	27 days ago
09.演算法	Add files via upload	20 days ago
10.遞迴函數	Add files via upload	27 days ago
img	Add files via upload	14 days ago
README.md	Update README.md	11 days ago



# fChart流程圖直譯工具 - 程式邏輯訓練







# 演算法 vs 流程圖

- 流程圖（Flow Chart）是演算法的一種常用表達方法，其他表達方法有：文字描述和虛擬碼（Pseudo Code，一種趨近程式語言的描述方法）

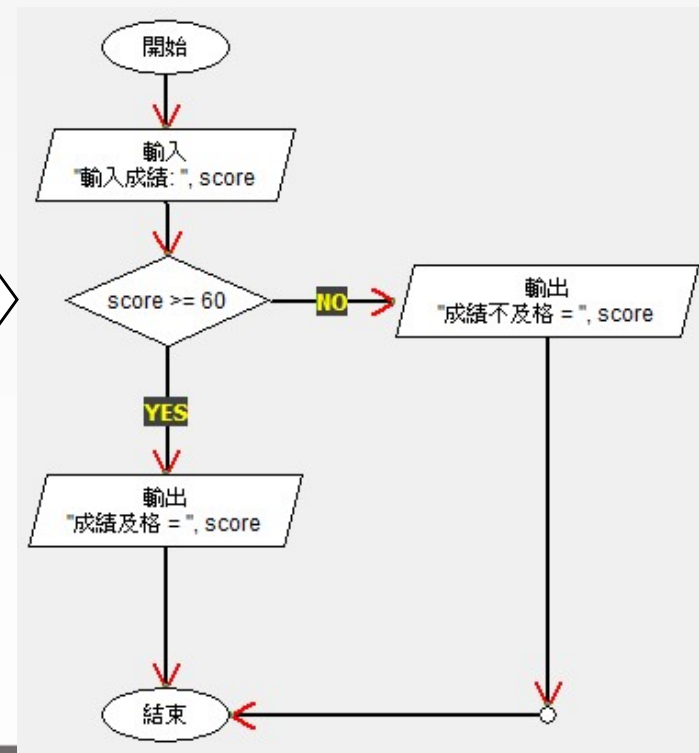
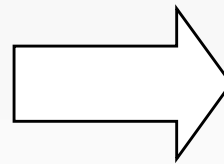
演算法步驟，如下所示：

**Step 1：輸入成績score。**

**Step 2：判斷是否超過60分：**

**(a). 超過，顯示及格。**

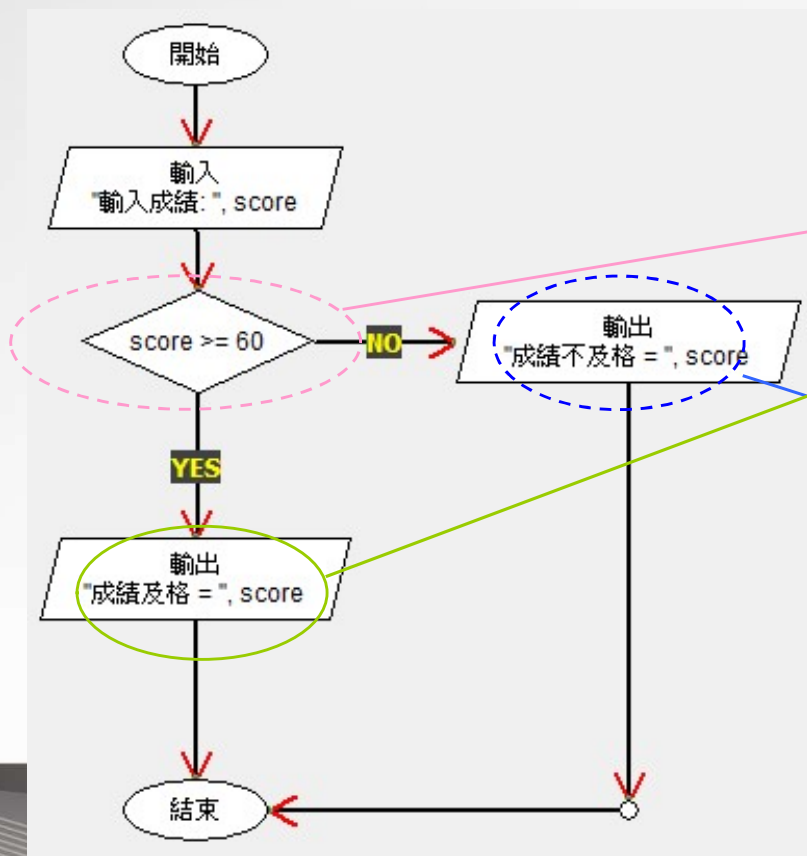
**(b). 沒有超過，顯示不及格。**





# 流程圖 vs 程式碼

- 對於程式語的程式碼來說，如果我們可以畫出流程圖的執行過程，就一定可以轉換撰寫成指定程式語言的程式碼，如下圖所示：



```
int score = 85;  
if (score >= 60) {  
    System.out.println(  
        "成績及格 =" + score);  
}  
else {  
    System.out.println(  
        "成績不及格 =" + score);  
}
```



# 程式語言到底有幾種？

- 國內目前常用程式語言主要有：Python、C、C++、Java、VB、C#、JavaScript和PHP語言等。
- 事實上，如此多種程式語言只有二類：
  - 結構化程式設計的程序式語言
  - 物件導向程式語言
- 學習程式設計的重點是學會結構化程式設計和物件導向程式設計的觀念。



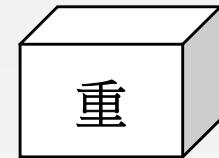
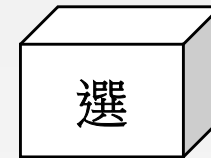
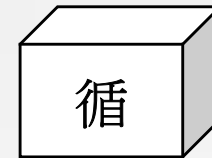
# 結構化程式設計

- **結構化程式設計（Structured Programming）** 是使用由上而下設計方法（Top-down Design）找出解決問題的方法，在進行程式設計時，首先將程式分解成數個主功能，然後一一從各主功能出發，找出下一層的子功能，每一個子功能是由**1**至多個控制結構組成的程式碼，這些控制結構只有單一進入點和離開點。
- 我們可以使用三種流程控制結構：**循序結構（Sequential）**、**選擇結構（Selection）**和**重複結構（Iteration）**來組合建立出程式碼。



# 結構化程式設計的程式-積木的組合

■ 程式是由積木所組成。



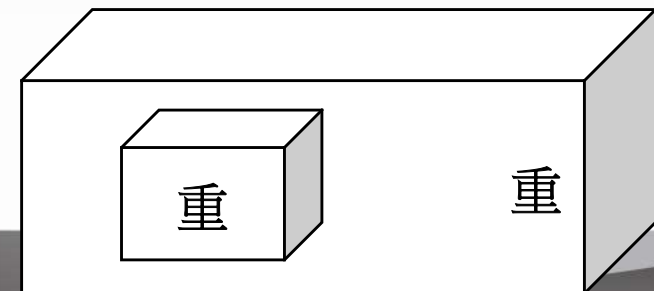
■ 積木分成三種：循序結構（**Sequential**）、選擇結構（**Selection**）和重複結構（**Iteration**）。

■ 積木的組合方式有兩種：

- 一個接著一個。
- 在大盒子中有小盒子(巢狀結構)。



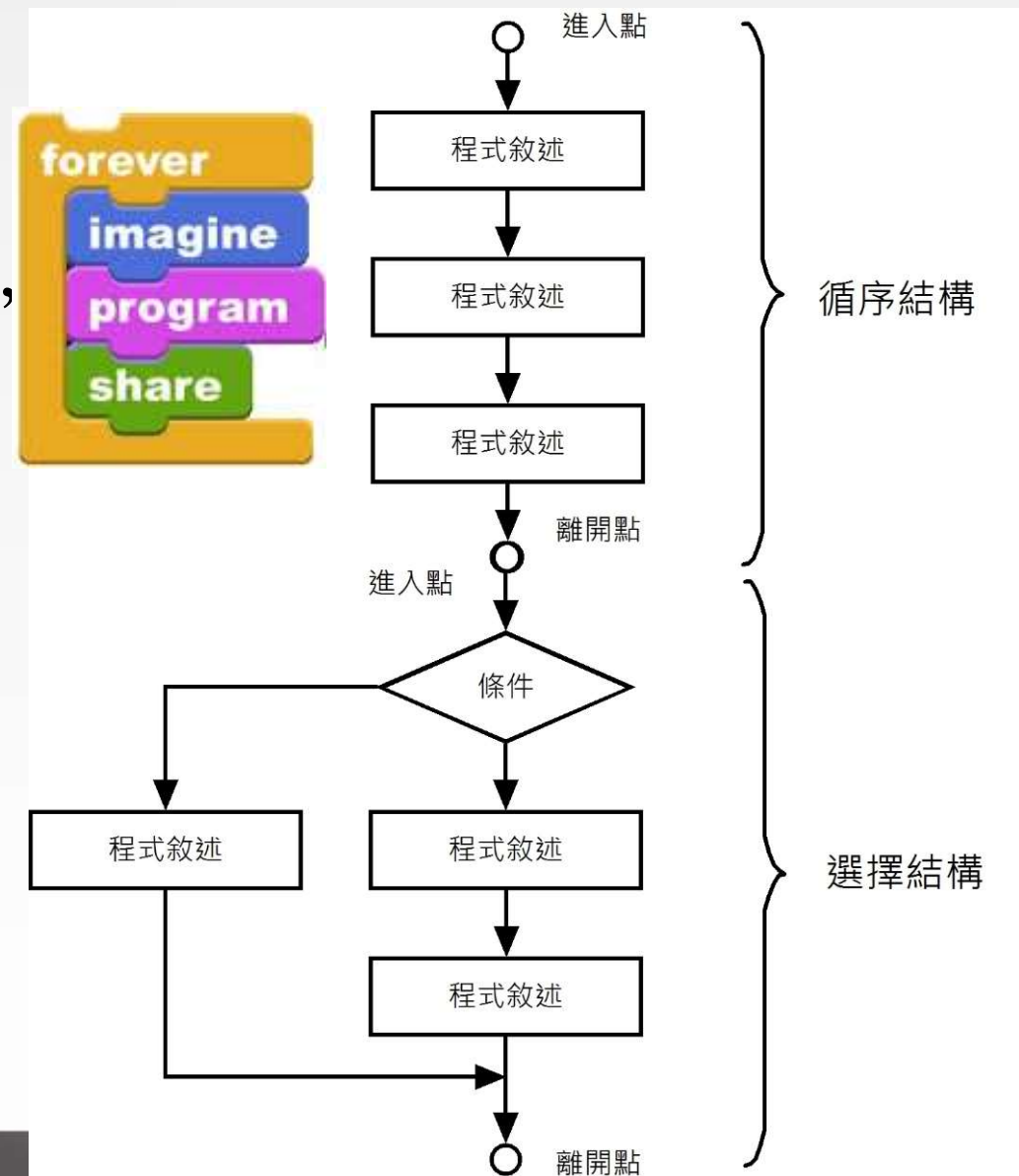
這也是為什麼Scratch可以  
使用積木來寫程式





# 結構化程式設計-程式範例

- 程式碼是從一個控制結構的離開點，連接至另一個控制結構的進入點，結合多個不同流程控制結構來撰寫程式碼。
- 如同小朋友玩堆積木遊戲，三種控制結構是積木方塊，進入點和離開點是積木連接點，透過連接點組合出成品，如右圖所示：

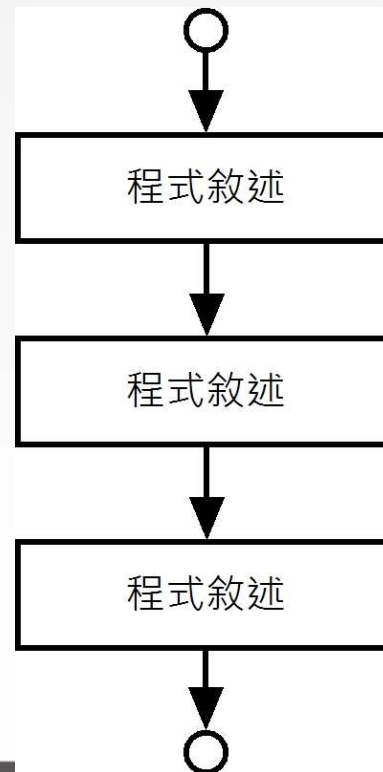






## 流程控制結構-循序結構(一條路徑)

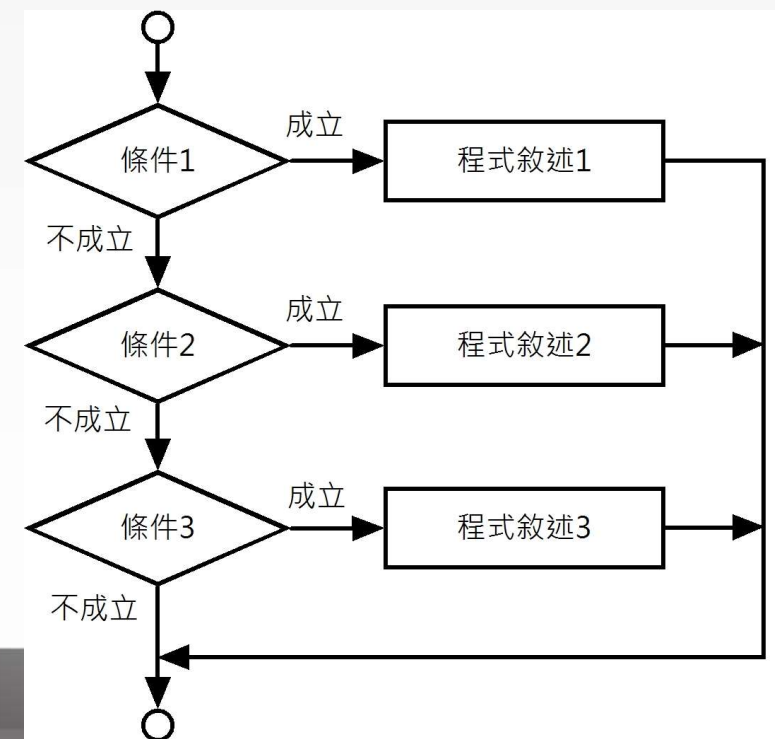
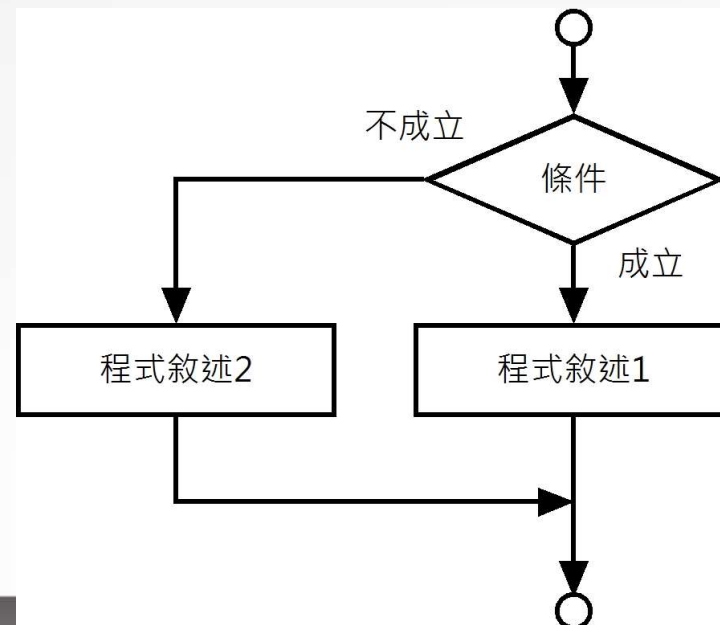
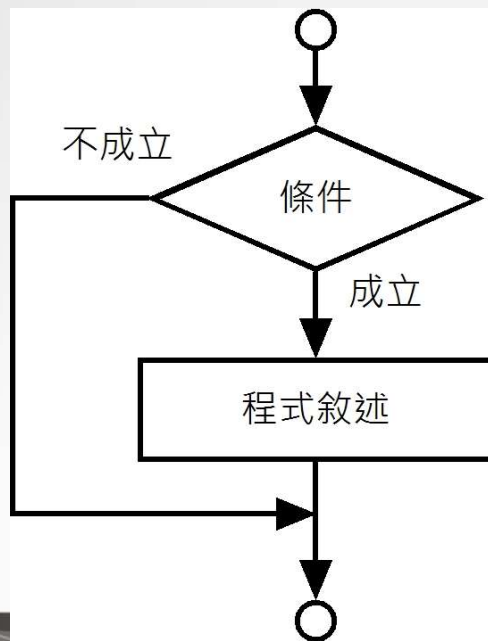
- 循序結構是程式預設的執行方式，也就是一個敘述接著一個敘述依序的執行(即輸入, 輸出, 變數和運算式)，如下所示：





# 流程控制結構-選擇結構

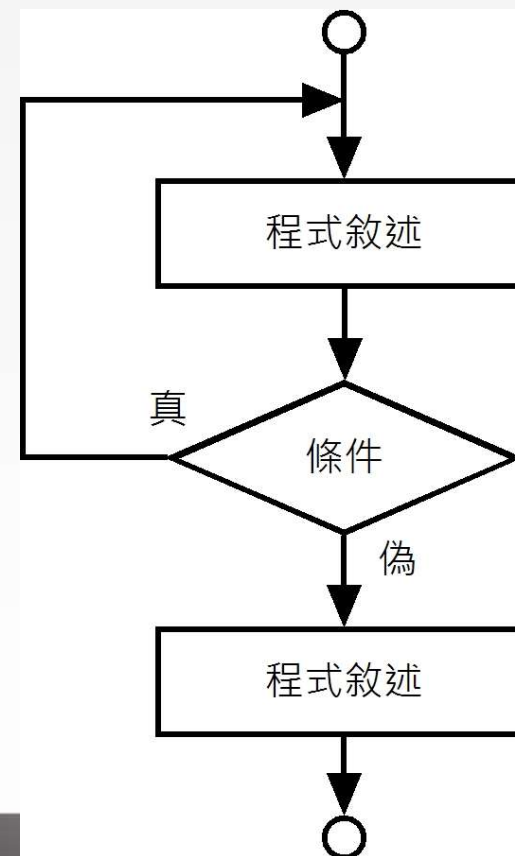
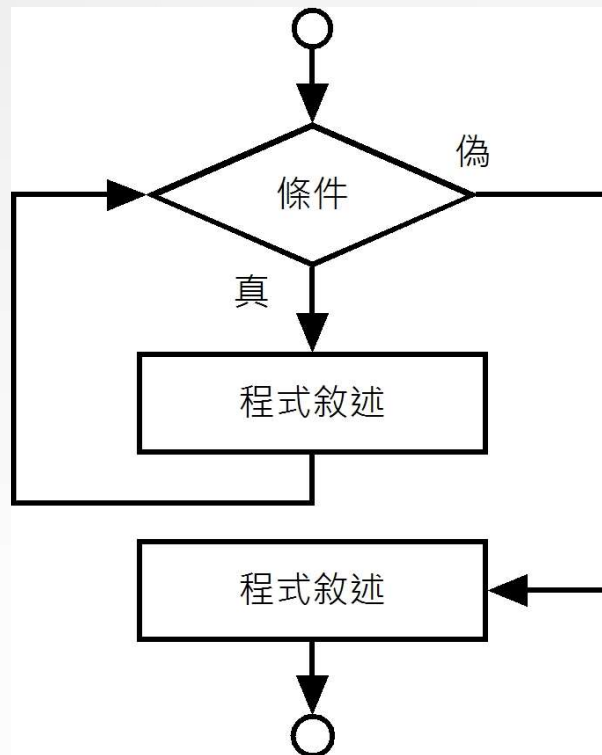
- 選擇結構是條件判斷的選擇題，分為是否選(替代路徑)、二選一(二條路徑)或多選一(多條路徑)三種。程式執行順序是依照關係或比較運算式的條件，決定執行哪一個區塊的程式碼，如下所示：





# 流程控制結構-重複結構(跳圈圈)

- 重複結構是迴圈控制，可以重複執行一個程式區塊的程式碼，提供結束條件結束迴圈的執行，如下所示：





# 學習基礎程式設計就是在堆積木

- **循序結構 ( Sequential )** : 一種積木(即輸入, 輸出, 變數和運算式)。
- **選擇結構 ( Selection )** : 是否選(替代路徑)、二選一(二條路徑)和多選一(多條路徑)三種積木。
- **重複結構 ( Iteration )** : 前測式迴圈和後測式迴圈二種積木。

**程式就是這六種積木的排列組合**