15 運算思維與程式設計

15-1. 用電腦解決問題

15-1-1 用電腦解決問題的四個步驟

- 階段一：Analysis - 運算思維。
- 階段二 : Design - 程式設計/設計程式。
- 階段三：Coding - 撰寫程式。
- 階段四：Testing - 測試與除錯。

The Process of Computational Problem Solving

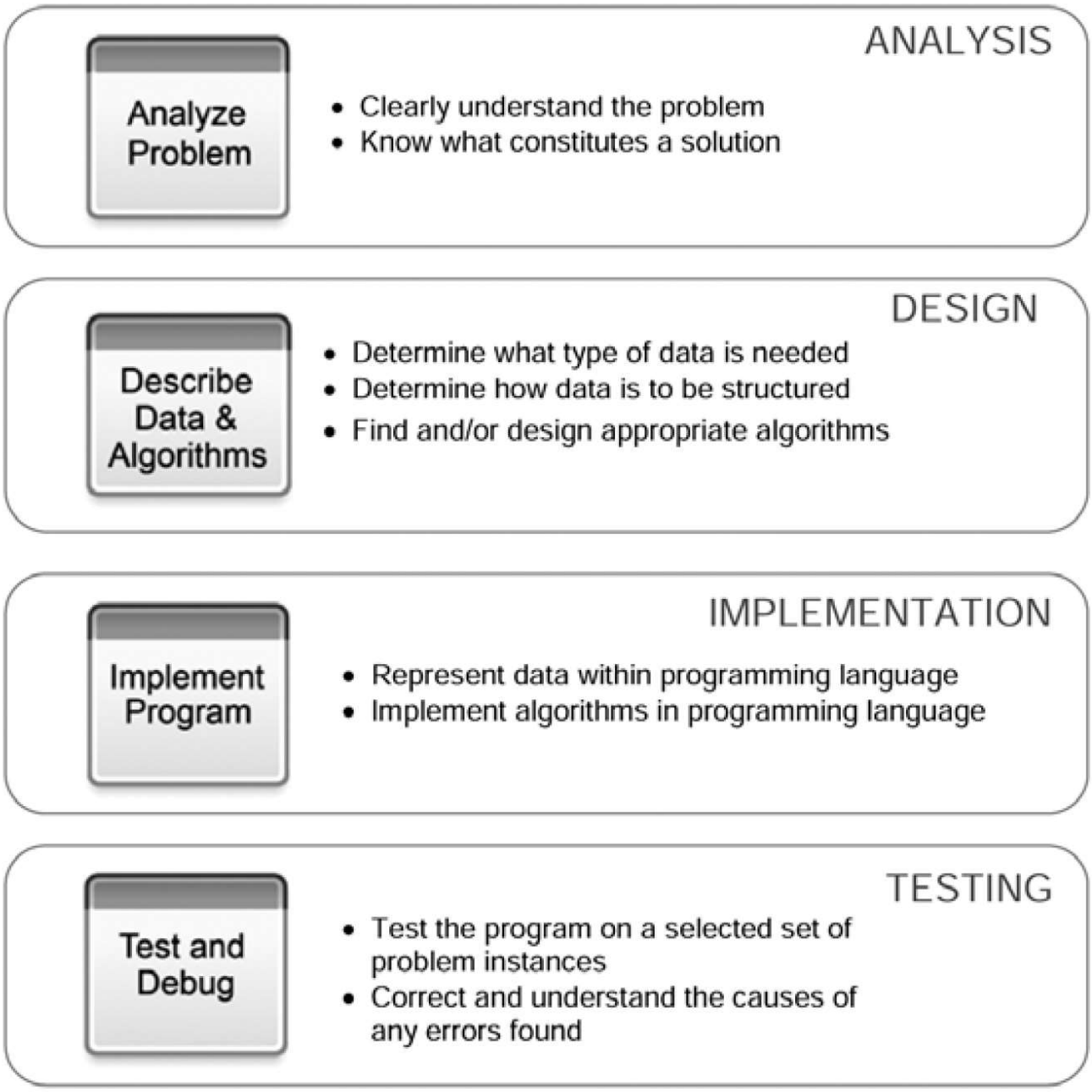

FIGURE 1-22 Process of Computational Problem Solving

15-1-2. 撰寫程式小技巧

Step1: 運算思維 - Problem Analysis 問題分析

問題分析分成瞭解問題（發現問題、定義問題、分析目的）與解決問題。

- **要解決問題，要先能發現問題**。
- **思考，瞭解問題 + 進行分析 + 找出解決問題的方法 => 請具體描述問題並說明解決問題的方法。**

寫程式小技巧1: 請你先想解題思路，即在『不用電腦』的情況下，腦中已經想清楚解題的每一個步驟。寫程式並不是從步驟三的指令開始，只要電腦不出現紅字就ok。

Step2: Program Design 程式設計/設計程式

在這個階段不用寫程式、寫任何指令，瞭解問題並知道解決問題的方法後，得要思考這個問題怎麼用電腦來處理，要怎樣落實用電腦解決問題的方法呢？也就是說，請做出導航的路線圖。就像你還沒開車上路，有了導航地圖，就知道要怎麼開了！

- **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。**
  - **資料抽象化：Describing the Data Needed**
  - **程序抽象化：Describing the Algorithms Needed**

寫程式小技巧2: 同學可以配合電腦IPO的架構，用虛擬碼（或流程圖）寫出你的解題思路。首先要寫的是Output，先寫出來經過程式執行後，確實要呈現的輸出樣式，而不只是答案而已。其次，再想一想Input，你要用到什麼資料、從哪裡獲得資料、怎麼給電腦資料。最後，請你規劃一下怎樣利用「序列、決策和重複結構」的組合去設計你的程式架構，會最清楚、最有效率！

Step3: Program Implementation - Coding 撰寫程式碼

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。

- 撰寫程式碼：有系統地依照「輸入-處理-輸出」的步驟寫出程式碼
- 注意規矩，即語法和風格。
  - 加註解 # ：多多使用虛擬碼，寫出你腦中的運算思維與程式設計邏輯，再對程式結構和語法做說明。
  - 運算子前後加空格 : x = 1 + 2 => x = 1 + 2
  - 逗號後面空一格：print(1, 2, 3, 4, 5) => print(1, 2, 3, 4, 5)

寫程式小技巧3: 依照程式指南寫作風格的建議方式寫程式，讓程式好看好懂好除錯。

=> 良好的程式設計要具備的條件

- 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。

Step 4. Test and Debug - Testing 測試與除錯

程式撰寫完成之後需要不斷測試並修改，以求程式在各種環境下都能順利執行。我們稱這種除錯的動作為除蟲。常常我們花在程式除錯的時間，比創造程式草稿還要長。

=> 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。

- 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。
- 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)

語法錯誤是我們在陳述或編寫時出了差錯。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**。
  - 比方不正確的內縮、遺漏了必要的冒號，或是左、右小括號沒有相互對應。
  - 中英文的切換，例如用的逗點要用英文字型、但因為習慣而誤用中文字型。

- 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

邏輯錯誤是指當指令執行時演算法沒有照計畫進行，出現程式執行結果與預期結果不同。

- 邏輯有問題：很難處理。人很難知道自己的邏輯有錯，甚置於能找出錯在哪裏。
- 解決方法：
  - 先用除錯器找出錯在哪一行。
  - 才會有機會抓出邏輯上的錯誤。

=> 第三類常見的錯誤：執行時期錯誤(Runtime Error)

執行時期才會發生的錯誤，導致程式不正常結束。

執行時期錯誤的處理和語法錯誤的處理是相似的。因為這些錯誤皆是在程式執行時，由直譯器加以偵測的。一般而言，語法錯誤是較容易發現與更正的，因為Python可以給予其錯誤的位置與引起錯誤的原因。相對而言，執行時期錯誤會比較不容易。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。

例：計算直角三角形面積的演算法？

- 步驟一輸入底的值
- 步驟二輸入高的值
- 步驟三面積等於底乘以高除以二
- 第四印出面積的值。

=> Type1 : 如果將步驟三不小心寫成 面積=底x高÷2多寫了一個除號，這就會發生語法錯誤。 => Type 2: 如果將步驟三不小心寫成面積=底x高x 2將除號寫成乘號，這就是所謂的邏輯錯誤。 => Type 3: 如果你要計算很多三角形的面積，你把資料檔放在D槽，但你的程式寫成C槽，那程式沒有語法錯誤、也沒有邏輯錯誤，只有等到程式實際執行時，它會告訴你找不到資料檔案的錯誤訊息。

大多數的程式語言整合開發環境會在撰寫程式時，也就是還沒執行犯下語法錯誤前，就在編輯器中提醒你、幫你找出問題點。然而邏輯錯誤，因為語法完全正確，如果不小心寫錯，就需要自己一步一步的找出問題。執行時出錯，電腦會給你錯誤訊息，不用太擔心！有問題時，記得多請教Google大神就對了。

寫程式小技巧4: 每寫一兩行或兩三行或幾行指令，就執行一次，確認程式沒有問題，不要等到程式全部寫完再執行才知道有沒有錯誤。

15-2 運算思維四基石

15-2-1 什麼是運算思維與四基石

1. **運算思維是一個思考的程序。**

運算思維的目的是闡明問題，並呈現其解決方案，讓電腦（與人）能夠有效率地執行。

用電腦解決問題。人要能請電腦幫我們做事，得知道如何讓電腦去執行任務。除了熟悉指令之外，更需要的是瞭解撰寫程式背後的邏輯。**學習程式設計最重要的是學邏輯、學會思考的方式。**

2. **運算思維四基石**

問題拆解(Decomposition) = >模式辨(Pattern Recognition)識 = > 抽象(Abstraction)化 = > 演算法(Algorithm)

15-2-2 抽象化

抽象化是去蕪存菁，識別並擷取可表達主要概念的重要訊息。

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

處理真實世界的資料時，得忽略主題當中與當前問題或目標無關的特徵，將注意力集中到重要特徵上，這個過程就是「抽象化」。由於處理的目標是資料，這種抽象化的過程也被稱為「資料抽象化」。

例1: 用某些資料代表一個學生的特徵。

圖 3-1

「抽象化」的結果依不同的問題而有不同。例如丁可樂這個人在就讀學校的眼中，關心的資料主要是：姓名、學號、科系等；在開戶銀行方面，關心的資料可能是：姓名、信用卡額度、年收入等。換句話說，我們在不同的問題上，用不同的抽象化資料來代表丁可樂這個人。這些抽象化資料的欄位值，各自有各自的資料

型態，可能是數字、文字，或者是布林等，這就是程式語言中的資料類型(data type)。（不同程式語言使用的資料類型可能會不同。）

例2：地圖：從A到B如何轉乘? 捷運的簡化抽象圖。

現在要處理的問題是【轉乘】，請先確認捷運站的點（相關的資料），再列出路線（解決問題的方法）。此時，我們透過捷運圖形就可以知道答案。

將注意力集中到跟當前問題或目標有關的運算上，將問題具體化。

從A到B = > 在什麼地方要坐什麼線、在哪裡轉乘、轉乘什麼線、在哪裡下車。=>這就是具體細節的演算法)

15-2-3 演算法

演算法是運算思維的具體展現，也就是解決問題的具體步驟與流程。

在設計程式時，必須先將問題分解成許多小步驟，然後再依照一定的次序逐步執行，我們將這個描述問題解決程序的方法稱做演算法 。有了演算法表示已經對問題有了解決之道，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。

例 ：讓機器人來幫我們計算直角三角形的面積。

- 第一步輸入底的值
- 第二步輸入高的值
- 第三步面積等於底乘以高除以二
- 第四步印出面積的值。

=> 演算法就像汽車導航一樣

- 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

15-2-4 模式辨識

模式辨識，找出資料中可見的樣式、規律、趨勢。

每個人看問題的方式不同，解題思路也不同。

透過下面的題目，你會再次發現原來我們的邏輯沒有想像中好。自己可以憑直覺就給出分組的「正確」答案，但不知道當中真正的邏輯是什麼，也就是你還無法一步一步的說明你的解題思路與步驟。記得：除非你能先說清楚你的直覺與邏輯是什麼，不然是無法轉成程式碼的。

範例: 分組報告

因為教學上的需求，老師要將全班40位同學分組做報告。老師規定依座號順序，每五位同學一組。也就是，1號到5號一組，6號到10號一組，以此類推。請寫一個程式允許使用者輸入座號，輸出分組的組別。

~ 預覽結果 輸入座號，例如「19」，計算結果顯示在螢幕如下。 請輸入座號？19 組別為 4

＝>運算思維：模式辨識

1 ，1/5 .....= 0.2 -> 商數0餘數1 -> 0,0

2，2/5 .....= 0.4 -> 商數0餘數2 -> 0,1

3，3/5 .....= 0.6 -> 商數0餘數3 -> 0,2

4， 4/5 .....= 0.8 -> 商數0餘數4 -> 0,3

5，5/5 ..... = 1 -> 商數1餘數0 -> 0,4

```python
## 運算思維 - 模式辨識
## 方法（一）：餘數位置從0開始 + 商數為組數
seats = int(input('請輸入座號？')) #文字轉數字
groups = (seats - 1)//5 + 1
print('組別為', groups)

## 方法（二）：函式（上限為組別）
import math
seats = int(input("請輸入座號？"))
groups = math.ceil(seats/5)
print("組別為",groups)

## 方法（三）：用餘數為條件分組
seats = int(input("請輸入座號？ "))
if seats%5 == 0:
    print("組別為", seats//5)
else:
    print("組別為", seats//5 + 1)
```

「模式辨識」是從具有代表性的資訊中找出規律，這是最困難的部分。

【加分題】分糖果遊戲

在糖果屋中，有6位小朋友在玩分糖果遊戲。這6位小朋友的編號為1, 2, 3, 4, 5, 6，並按照自己的編號坐在一張圓桌旁。它們身上都有不同的糖果，從1號小朋友開始，將自己的糖果均分成3份，如果有多的就立刻先吃掉，然後自己留1份，其餘2份給身邊的小朋友。接著2號、3號、4號、5號、6號同學這樣做。問一輪後，每位小朋友手上有多少糖果。


```
## Input Data
a = int(input("a = "))
b = int(input("b = "))
c = int(input("c = "))
d = int(input("d = "))
e = int(input("e = "))
f = int(input("f = "))
## Process Data
a = a//3; b = b + a; f = f + a
print("a = ", a, "b = ", b,  "c = ", c, "d = ", d, "e= ", e, "f= ", f)
b = b//3; a += b; c += b
print("a = ", a, "b = ", b,  "c = ", c, "d = ", d, "e= ", e, "f= ", f)
c = c//3; b += c; d += c
print("a = ", a, "b = ", b,  "c = ", c, "d = ", d, "e= ", e, "f= ", f)
d = d//3; c += d; e += d
print("a = ", a, "b = ", b,  "c = ", c, "d = ", d, "e= ", e, "f= ", f)
e = e//3; d += e; f += e
print("a = ", a, "b = ", b,  "c = ", c, "d = ", d, "e= ", e, "f= ", f)
f = f//3; e += f; a += f
print("a = ", a, "b = ", b,  "c = ", c, "d = ", d, "e= ", e, "f= ", f)
```

15-2-5 問題拆解

把一個複雜的大問題，先分割成許多小問題，再把這些小問題個個擊破；小問題全部解決後，原本的大問題也就解決了。

有些問題乍看之下，感覺相當複雜，這個時候我們可以把大問題拆解成幾個小問題，讓每個小問題比較容易解決。當每個小問題都被驗證，可以成功地解決後，把它們的結果串接起來，大問題也就迎刃而解了。

- **拆解克服（Divide and Conquer)，會讓整個工作流程更清楚，是解決問題的第一步。**

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，拆解可以讓整個工作更清楚，每部分的工作比未拆解前簡單，可以更有效率的處理，也方便與他人分工合作。

- **找到好的切入點，就可以成功地把大問題拆解成幾個小問題。**

每個部分的工作比未拆解之前簡單，就可以更有效率的處理，也方便與他人分工合作。

- **正確拆解問題，需要專業知識、普通常識與生活經驗。**

切生日蛋糕，請問垂直切還是水平切？