運算思維
COMPUTATIONAL THINKING (CT)

呂聰賢整理

什麼是運算思維

- Computational Thinking (CT)是一種解決問題的過程，包括了一些特性和性格的。CT是計算機應用的發展至關重要，但它也可以被用來支持在所有學科，包括人文科學，數學和科學問題的解決。誰跨課程學習的學生CT可以開始看到學科之間的關係，以及生活之間的內部和課堂外。

- 計算機可以用來幫助我們解決問題。然而解決問題之前，應該要理解問題本身和它可行的解決方式。

資工 Computer science

- 怎麼表示它？
- 怎麼最有效的儲存它？
- 怎麼處理它？

資工是研究電腦計算及其應用的一門學科

運算思維是一門包含了撰寫程式時所運用的技巧以及思考方式的藝術

Computational Thinking 應用

計算思維概念	學科領域的應用
打破一個問題分成部分或步驟	文學：一首詩的分析分解成米，韻達，意象，結構，語調，用詞和意義的分析。
識別和發現模式或趨勢	經濟學：查找上升，該國的經濟下降週期模式。
制定指導解決問題或步驟的任務	烹飪藝術：寫食譜供他人使用。
概括的模式和趨勢進入規則，原則，或見解	化學：確定化學結合和相互作用的規則。 數學：找出規則為保第二次多項式

發現所有的技能都是CT技能或概念

CT技能被應用在文學，經濟學，烹飪藝術，和音樂使用

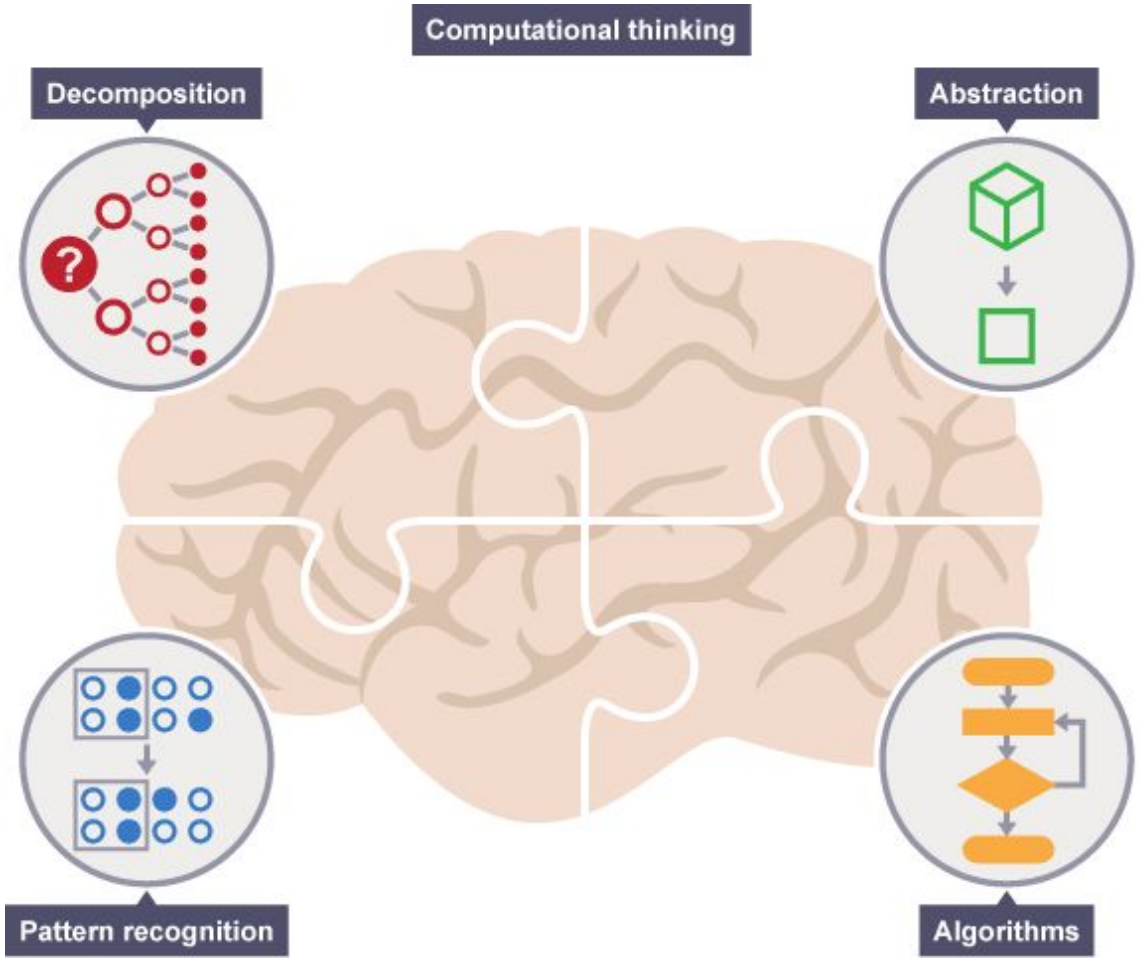
computational thinking & computer science

計算思維概念	計算機科學中的應用
打破一個問題分成部分或步驟	打破一個計算問題圖分為4個部分，每一個由不同的計算機處理器完成。
識別和發現模式或趨勢	可視化數據進行比較芯片材料及電腦速度注意到一個趨勢
制定指導解決問題或步驟的任務	編寫計算機程序對數據進行排序
概括的模式和趨勢進入規則，原則，或見解	實現複雜的數據結構需要比複雜的程序更少的代碼

什麼是運算思維？

- 運算思維不是寫程式，而是告訴電腦該如何做。
- 如果你今天和朋友約在一個從沒有去過的地方，你可能出門前，會先規劃路線，你會想那些路線可以到，以及那一條路最好。例如最短、最快、或有經過喜歡的商店，接下來就會按著計劃的一步一步進行。
- 以上計劃的過程就是運算思維，按照指示的方式執行就是使用程式。
- 能夠把複雜的問題轉換成我們容易理解的問題是非常有用的，事實上你可能已具備這項技能。

Computational Thinking



Finally, these simple steps or rules are used to **program** a computer to help solve the complex problem in the best way.

運算思維的四個步驟

- Decomposition(問題**分解**)
  - The breaking down of a system into smaller parts that are easier to understand, program and maintain.
  - 將一個複雜的問題分解成很多的小問題，進而能夠更容易的了解，處理跟維護
- Pattern Recognition(**模式識別**)
  - looking for similarities among and within problems
  - 尋找問題中的相似之處
- Abstraction(抽象-重點摘要)
  - focusing on the important information only, ignoring irrelevant detail
  - 只專注於重要的信息，忽視無關緊要的細節
- Algorithm Design(演**算法設計**)
  - developing a step-by-step solution to the problem, or the rules to follow to solve the problem
    開發解決這個問題的步驟、規則

For example-和朋友出遊

- 假如今天你要決定和你的朋友們做些什麼,如果你們喜歡的東西都不一樣,你需要決定:
- 你們可以做什麼
- 你們想去哪裡
- 誰想要做什麼
- 你過去成功的經驗
- 你們有多少資金，以及每個方案的花費
- 天候狀況的應變措施
- 你有多少時間

For example-和朋友出遊

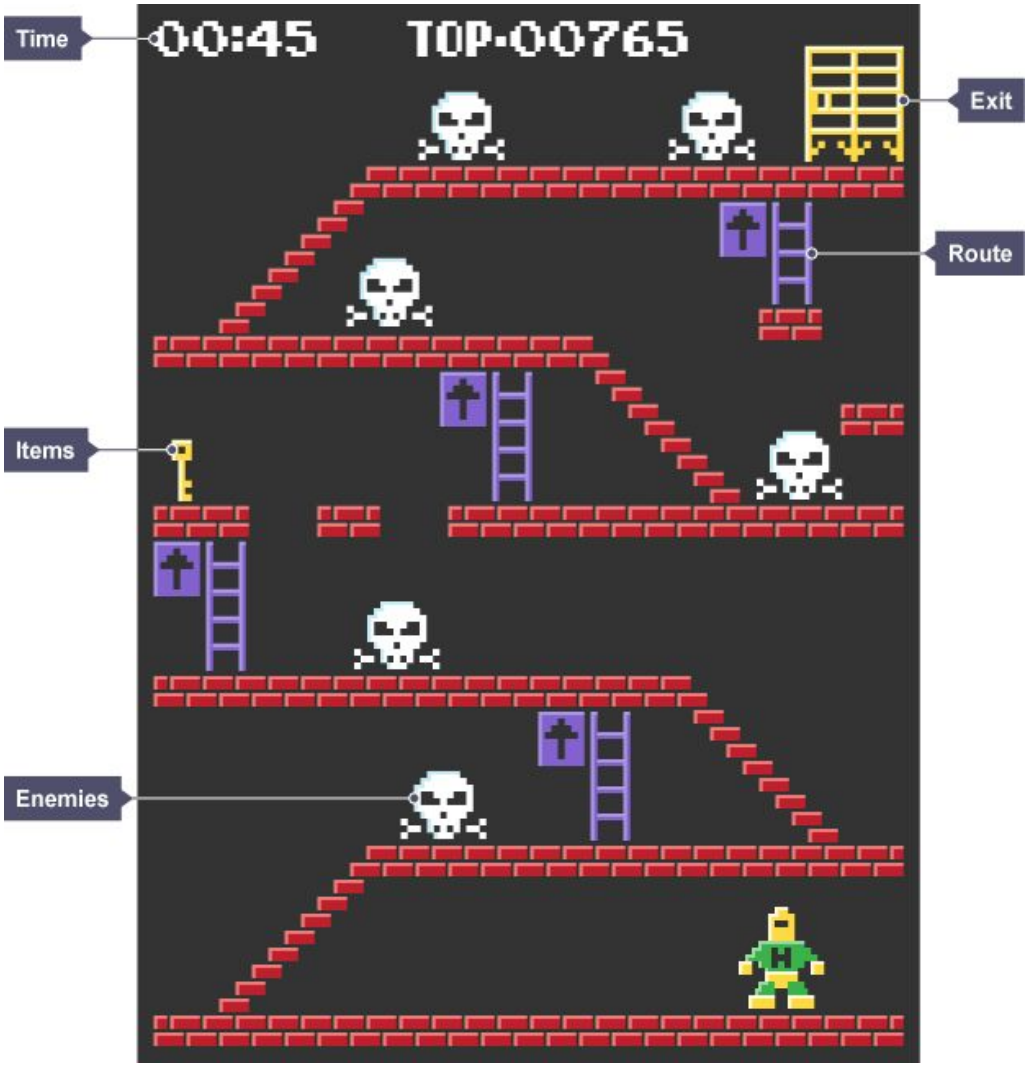
- 你和你的朋友就能更容易去那裏,要做什麼,讓大部份朋友都很開心,你也可以使用電腦搜集,分析資料,找出最適合的方案,現在和未來都可以使用。

For example-玩遊戲過關

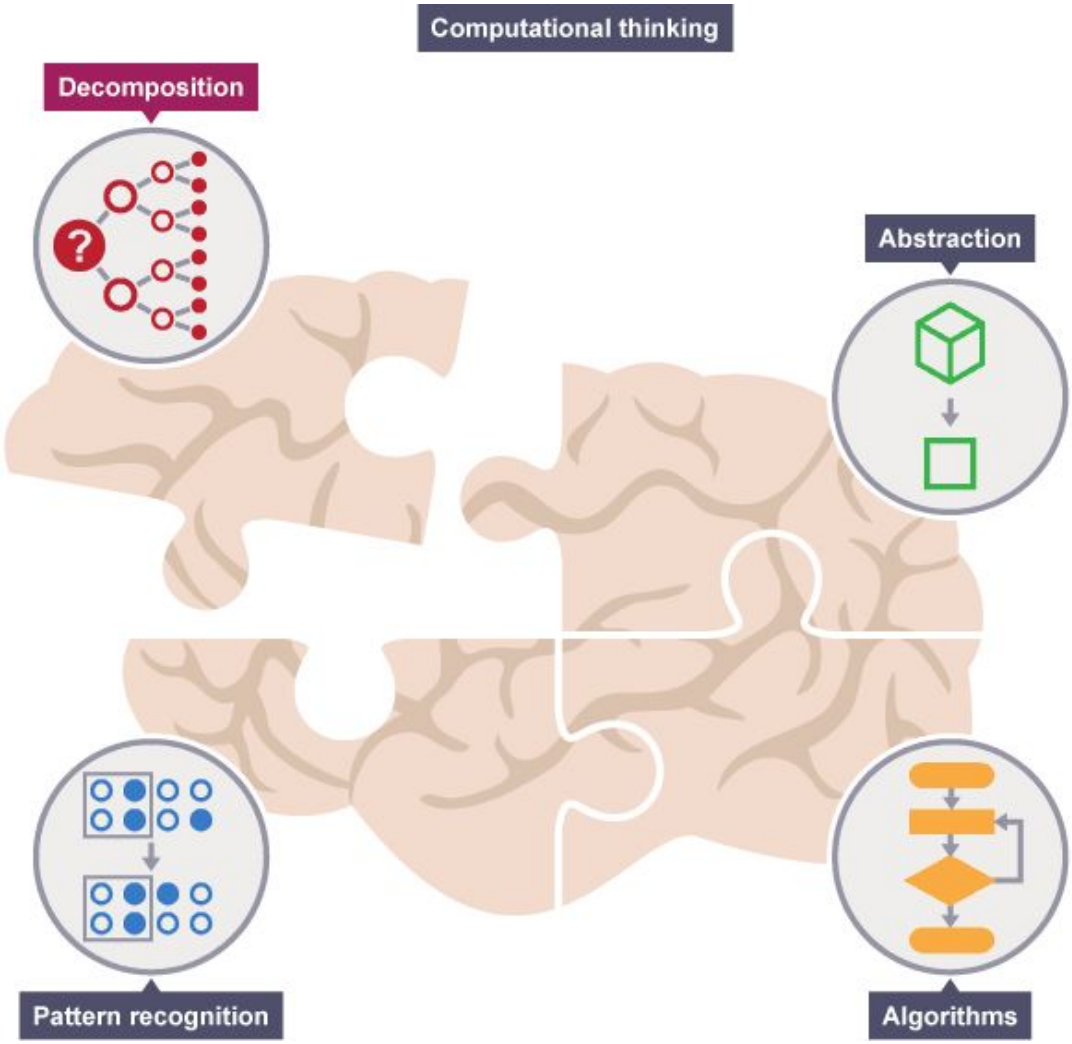
- 為了過關你需要知道
  1. 你需要搜集的道具，如何搜集，要花多久時間。
  2. 出口在那裏和最佳路線
  3. 有那些敵人，他們的弱點是什麼
- 從以上資訊你可以更有效率的想出過關方式
- 如果你要創建自己的電腦遊戲，這些正是你動工前所需要思考並回答的問題。

For example-玩遊戲過關

- 每一個複雜的問題分解成若干個小的決定和步驟(例如去哪裡，如何完成的水平- 分解)
- 只有相關的細節都集中在(如天氣，出口的位置- 摘要重點)
- 以前類似問題的知識被用於(模式識別)...
- 用行動步驟計劃，以制定出一個步驟(演算法)



問題分解 Decomposition



什麼是問題分解 Decomposition?

- 分解是計算機科學的四大基石之一。它涉及到打破一個複雜的問題,把問題分割並分類為更小部份，更易於管理的部分。然後，更小的部分可以輕易被檢查和解決，或單獨設計。

- 為什麼分解重要？

- 如果一個問題不分解，這是更難處理。一次處理許多不同階段的間題,遠難於處理分解後的較小問題。分解後的小問題更容易進行詳細的檢查。
- 同樣，試圖了解複雜的系統是如何運作,如此才能更容易解構問題。例如，了解自行車是如何工作的，如果整個自行車分離,成較小的部分，每個部分進行檢查，看看它是如何工作的更詳細更簡單。

生活中的分解

- 生活中的問題分解
- 當在做生活中的瑣事的時候，我們常常並沒有多加思考或是分解(我們每天做很多任務，甚至沒有思考-或分解-他們，比如刷牙。)

- 例1:刷牙
- 分解如何刷牙的問題，我們需要考慮:
  - 用那種牙刷
  - 要刷多久
  - 用多大力道
  - 使用什麼牙膏

生活中的分解範例

- **如何破案**
- 當我們面對一個複雜的問題的時候，我們通常從“細節”開始思考起
- 警官在思考破案問題的時會將複雜問題“分解”成一系列的‘小問題“
  - 是犯了什麼罪
  - 犯罪何時發生
  - 在那裏發生
  - 有什麼樣的證據
  - 是否有任何證人
  - 最近是否有類似的罪行

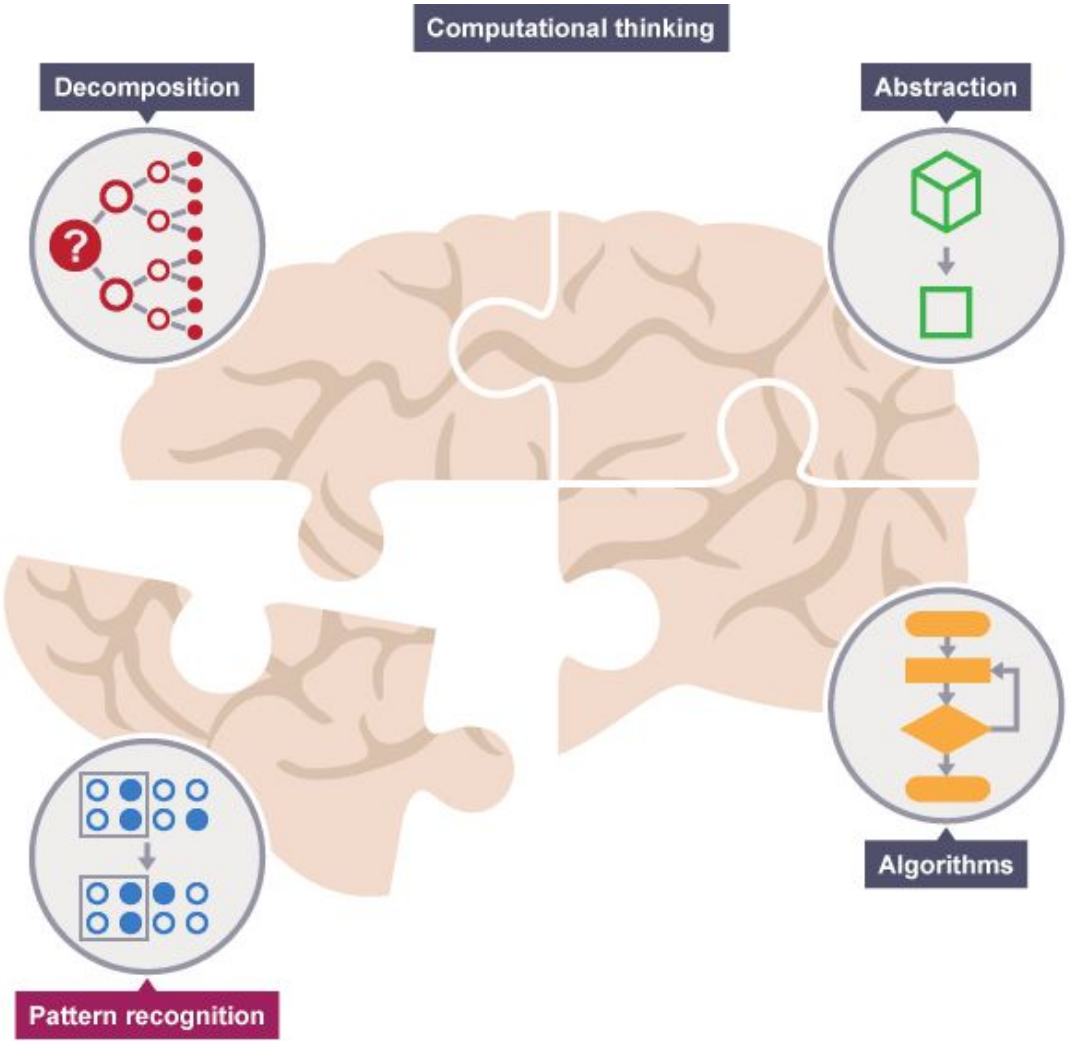
- 一個複雜的問題被分解成好幾個小問題並一一詳細分析

運用分解來製作APP

- 問題:
  怎麼分解製作APP的步驟以及問題
- 答案:
- 為了要進行分解，我們必須先了解這一系列小問題的答案
  - 要做哪一種類型
  - 看起來是什麼樣子
  - 誰是目標客戶
  - 怎麼設計使用者介面
  - 要使用什麼音效
  - 要用什麼軟體來開發
  - 使用者怎麼在移動到不同的頁面
  - 怎麼測試
  - 在哪裡賣

這個列表將製作 APP 這一個很複雜的問題分解成多個簡單的小問題，而每個問題能被逐一擊破。你也可以讓其他人幫助你不同的小部分，譬如說你能讓一個朋友做試用者介面，另一個朋友做測試。

模式識別 Pattern Recongnition



什麼是模式辨識

- 模式辨識是運算思維的四基石之一。模式辨識牽涉找到分解之後小問題的相似之處或是共有屬性(模式)，並用此資訊幫助我們更有效的解決複雜的問題。

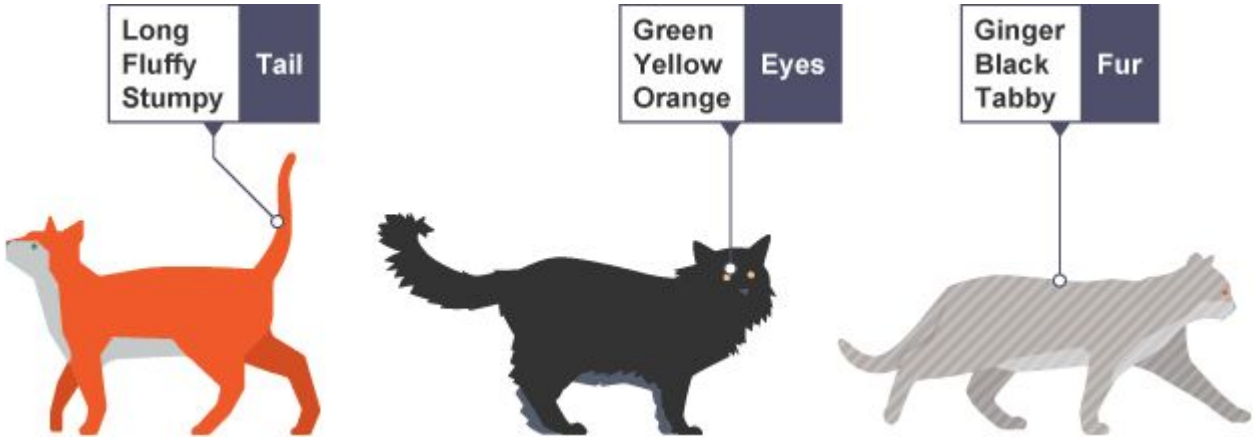
- 當將一個複雜的問題分解之後，我們常常能發現小問題中有機可循，而這個機就是問題中共有的屬性以及相似之處。

什麼是模式

- 想像一下我們想要畫一系列的貓
- 所有的貓都共享一些屬性，譬如說他們都有眼睛，尾巴，毛髮，喜歡吃魚跟喜歡喵喵叫。
- 因為我們知道所有的貓都有這個屬性，當我們想要畫貓的時候便可將這些共有的屬性加入其中讓畫貓這項工作變得更簡單。

畫一系列的貓

- 在運算思維當中，這些屬性稱之為模式。當我們知道怎麼描述一隻貓之後，我們可以依照這些模式輕易地描述其他貓，而唯一不一樣的地方是每隻貓個有的獨特之處（特性）
  - 一隻貓有綠眼睛，長尾巴跟黑毛
  - 另一隻貓有黃眼睛，短尾巴跟條紋的毛色



為什麼我們需要找模式

- 找到模式是非常重要的，模式讓問題簡化，當問題共享特徵時，他們能夠被更簡單的解決，因為當共通模式存在時，我們可以用相同的問題解決法去解決此類問題。
- 當我們發現越來愈多的模式時，解決問題會變得更加容易以及迅速
- 當我們想要畫貓的時候，找到基本畫貓的模式，像是有眼睛，尾巴根毛髮，能讓這項事情變得更簡單跟快速的完成。
- 我們知道所有貓都遵循此一模式，所以我們不用每次畫貓的時候都停下來找出這些屬性，依據我們得出來的模式，我們可以很快地畫出很多隻貓

如果我們不找模式會怎麼樣？

- 假設我們不去找模式，每次我們要畫貓的時候我們就得停下來思考貓長什麼樣子，過程就因此變慢了。
- 我們還是可以畫貓，他們還是會看起來像貓，只是每一隻都要花更久的時間畫，這非常的沒有效率並且是一個不好的方式去解決畫貓這個問題
- 除此之外，如果我們沒有去找模式，我們可能不會知道所有的貓都有眼睛，尾巴跟毛，當我們畫貓的時候也有可能看起來不像貓，這種情況底下因為我們沒有辨別出來模式，而使得我們沒有正確的去解決這個問題。

辨別模式

- 我們必須尋找各個問題中的相同以及相似之處來找出問題中的模式，有時也會發現問題之中並沒有共有屬性，但我們還是應該嘗試去尋找。

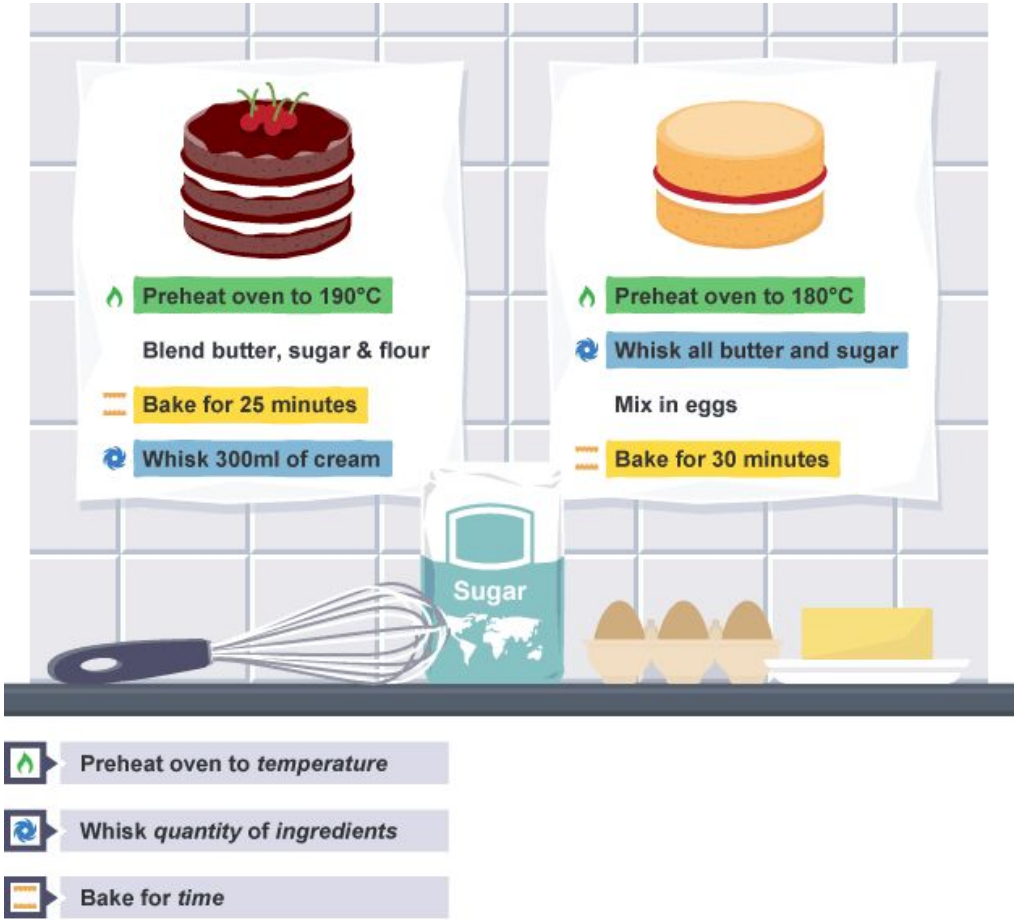
- 模式存在於多個問題之中以及單一個問題裡面，兩個面向都需要觀察。

不同問題中的模式

- 我們在不同問題中尋找他們相同以及相似之處去找出問題中共有的模式
- 譬如說分解烤蛋糕問題需要我們知道一系列小問題的答案
  - 要考什麼樣的蛋糕？
  - 需要麼樣的材料？各需要多少？
  - 這個蛋糕要烤給多少人吃？
  - 烤這個蛋糕需要多少時間？
  - 什麼時候要加什麼材料？
  - 需要什麼器具？
- 當知道怎麼烤某一種蛋糕之後，因為烤蛋糕有模式可循，我們可以了解到烤另一種蛋糕並沒有那麼大的不同。

不同問題中的模式範例

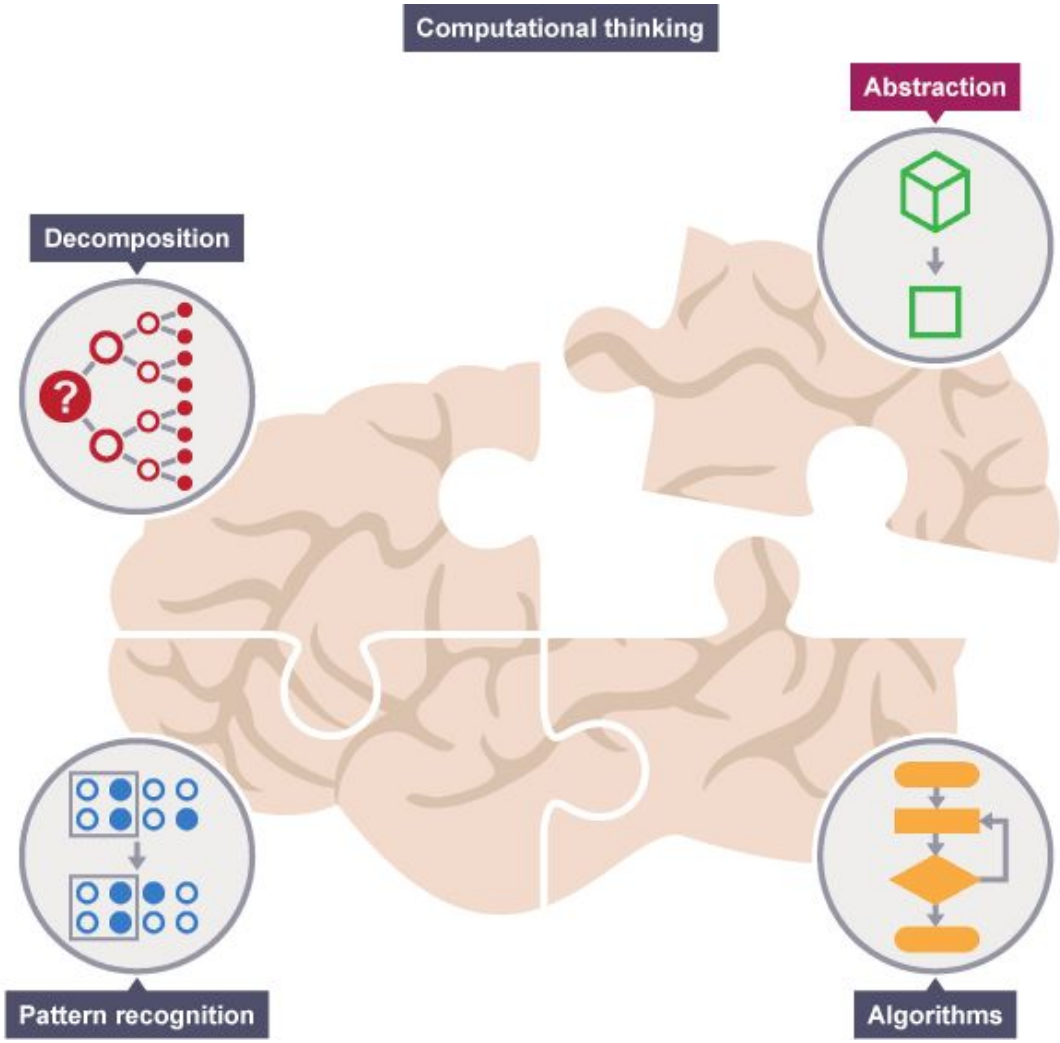
- 模式
  - 每個蛋糕都需要一定份量的材料
  - 材料會在特定時間被加入
  - 每個蛋糕會被烤一定的時間
- 當我們辨別出模式之後，我們便可用此解決辦法來面對不同的問題。



單一問題中的模式

- 當我們把問題分解成小問題之後，有時小問題之中也有模式可循。
- 拿烤蛋糕舉例，我們也可以找到小問題之中的共有屬性。譬如說每個蛋糕都需要一定份量的材料，每一種材料分別需要被
  - 識別
  - 量測
- 當我們知道怎麼處理第一種材料之後，我們可以用相同的方法來辨別其他的模式。

重點摘要 Abstraction



什麼是重點摘要

- 重點摘要是運算思維的四基石之一。重點摘要在於“過濾”以及“忽略”掉不必要的特徵，讓我們可以集中在重要的特徵上，他也包含過濾掉“特定”的細節。

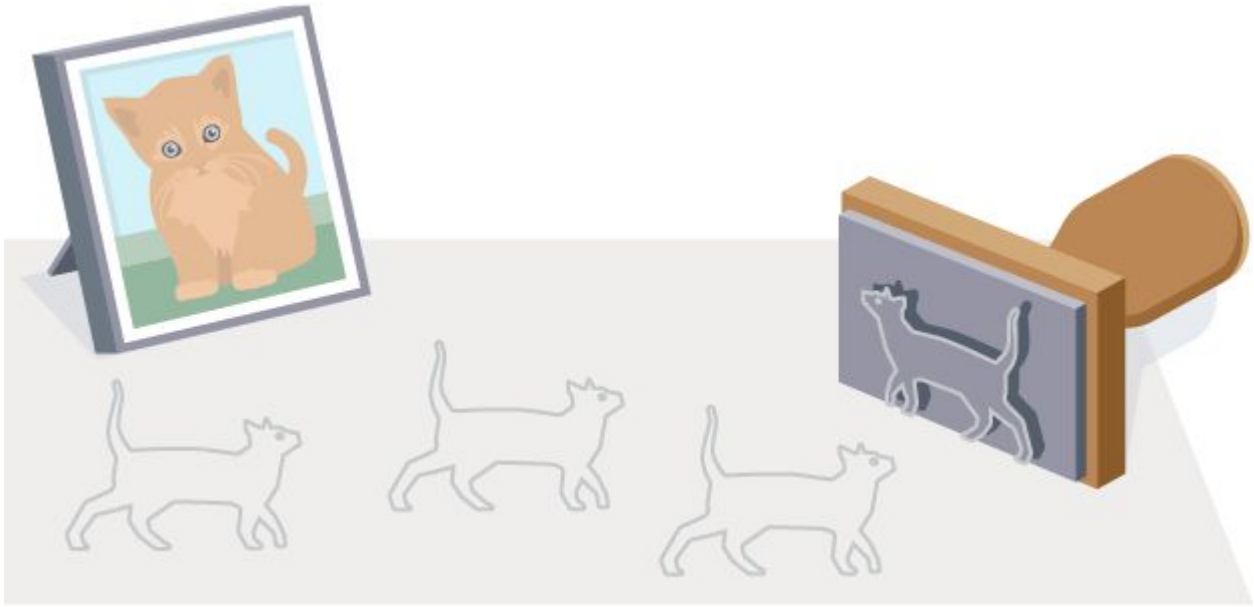
- 在這過濾過後的資訊之上我們建立起我們想要解決的問題重點摘要。

什麼是特性？什麼是共同屬性？

- 用畫一系列貓咪的例子。我們知道所有的貓都有**共同屬性**，像是眼睛，尾巴，毛，喜歡吃魚跟會喵喵叫。除此之外每一隻貓有自己的個別屬性像是**黑色**的毛，**長**尾巴，**綠**眼睛，喜歡吃**鮭魚**以及**大聲**喵喵叫。這些個別屬性我們稱之為**特性**。
- 我們必須知道共同的屬性(有眼睛，尾巴，毛)去畫出**基本**的貓咪，這些屬性是相關的，共通的。而我們不需要知道特性(黑色的毛，長的尾巴)去畫出基本的貓咪，這些與基本屬性並不相關，可以以後再解決。所以這些**特性**應該被**過濾**掉。
- 共同的屬性就是摘要出來的重點
  特性則是過濾掉的資訊

什麼是特性？什麼是共同屬性？

- 從共同屬性出發，像是眼睛尾巴跟毛法，我們可以建立一個貓咪初步的形象，並藉由此形象畫出基本的貓咪。
- 再用每隻貓咪的特性來畫出個別的貓咪。



為什麼重點摘要很重要

- 重點摘要讓我們建立一個通用的問題以及怎麼解決他。這個過程需要我們把特性以及無法幫入解決問題的模式去掉。這幫助我們將問題具體化，而這個具體化的問題則稱之為“模板(model)”。
- 如果我們不做重點摘要的話，我們很有可能找出一個錯誤的答案。以畫貓咪為例子，如果我們不做重點摘要，畫貓咪的模板可能會被設定為所有的貓都有長尾巴跟短毛，做完重點摘要後我們知道雖然貓咪有尾巴跟毛，但是並非所有都是長尾巴或是短毛。此過程幫入我們建立正確的基本貓咪的模板。

怎麼做重點摘要

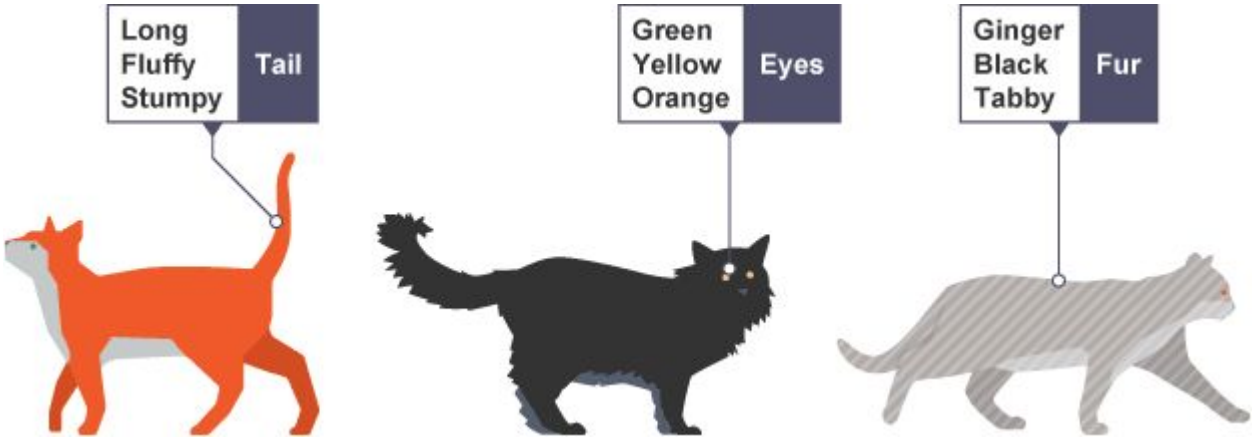
- 重點摘要是收集所需的資料，過濾掉不必要的資訊和個性，留下相關以及重要的共同屬性的過程。
- 以烤蛋糕為例子，以下是烤蛋糕的共同屬性
  - 烤蛋糕需要素材
  - 每個素材需要一定的量
  - 烤蛋糕需要計時

怎麼做重點摘要

共同屬性	特性
我們需要知道烤蛋糕需要素材	我們不必知道是具體是哪一種素材
我們知道每一種素材需要一定的量	我們不必知道這個具體的量是多少
我們知道烤蛋糕需要一定時間	我們不必知道具體的時間是多少

建立模板

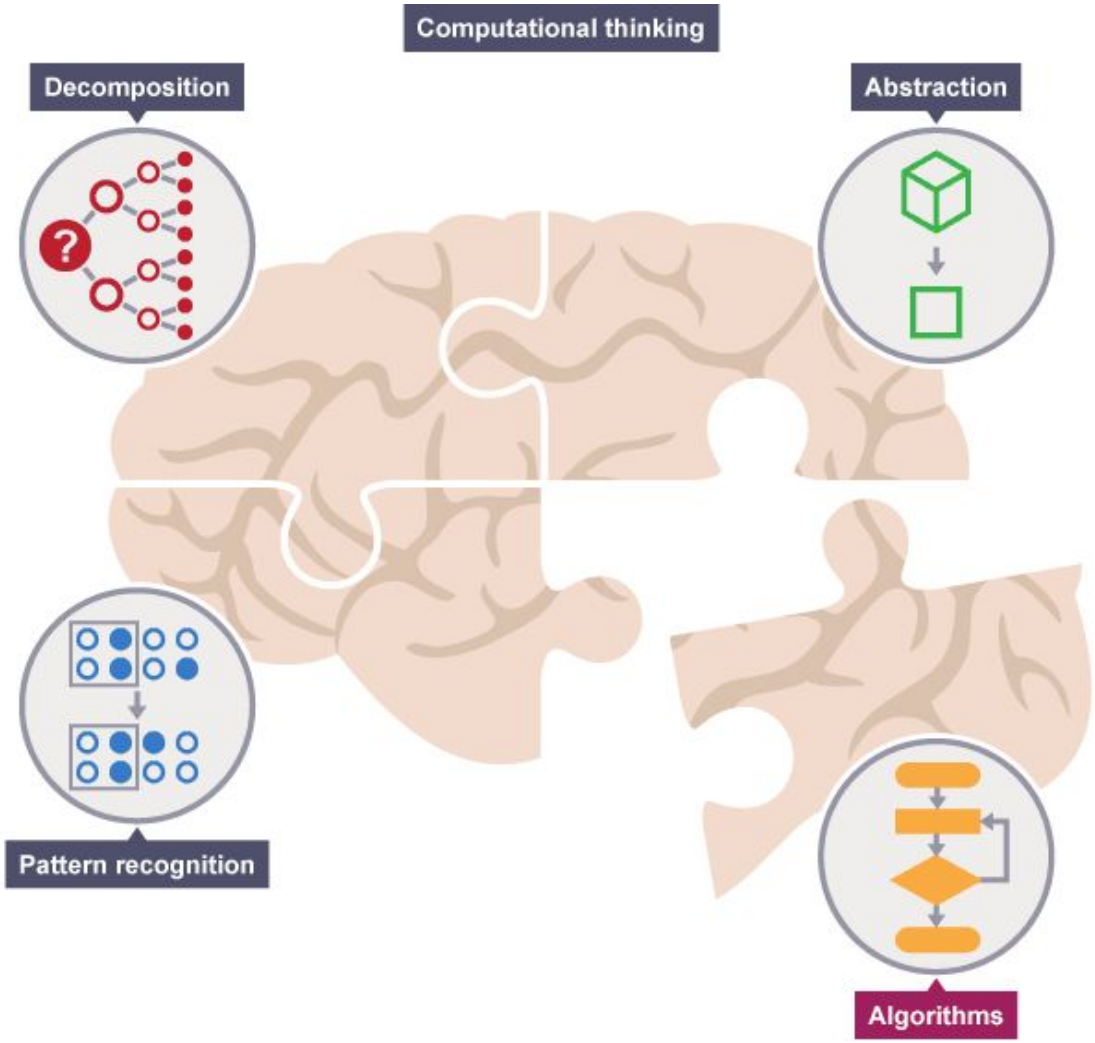
- 模板是一個我們想要解決的問題的大致理念
- 譬如說一個所有貓咪都可以套用的模板並不是一個有長尾巴短毛的貓咪模板，這麼模板要能套用到所有的貓咪身上。從我們的模板出發我們可以知道所有的貓咪共享哪些屬性



建立模板

- 當烤蛋糕時，烤蛋糕的模板並不是考某一種特定的蛋糕笑是海綿蛋糕或是水果蛋糕，這個烤蛋糕模板應該要能套用到所有烤蛋糕的過程上面，從這個模板我們可以知道如何考任何蛋糕。
- 當有了一個問題的模板，我們就可以開始設計演算法去解決這個問題。

演算法 Algorithm



什麼是演算法

- 演算法式運算思維的四基石的最後一個，演算法就是計劃，這個計畫裡面包含解決問題的每一個步驟跟指示，如果你知道怎麼繫鞋帶，怎麼泡茶，怎麼煮飯你已經知道如何遵從演算法來完成任務了。
- 在演算法當中，每一個指示以及順序都是經過計畫過的，演算法常常被使用為設計電腦程式的第一步，常見的演算法表示方式包含流程圖以及偽代碼(pseudocode)。

什麼是演算法

- 如果我們想要讓電腦做某些事情，我們必須寫個電腦程式並告訴電腦每一個步驟，我們想要做什麼以及我們想要怎麼做。這一個遵循每一個步驟的程式是需要經過規劃的，而演算法則是用來規劃的工具。
- 電腦能做出來的事情最多只能跟他們被給予的演算法一樣好，如果你給一個很糟糕的演算法，那麼就會得到很糟糕的結果。
- 演算法被廣泛用在很多不同的領域，包含計算，資料處理以及自動化。

什麼是演算法

- Algorithms are used for many different things including calculations, data processing and automation.

製作計畫

- 計畫如何得出問題的答案是很重要的並且能確保得出來的結果是正確的。運用運算思維以及問題分解我們可以把複雜的問題分解成很多的小問題，然後我們在計畫如何把每一個小問題的答案用正確的順序拼在一起得出最後的答案
- 這個順序可以被表示成為演算法，而必須包含以下幾個要素
  - 起始點
  - 清楚的中間步驟
  - 終結點

表達演算法:偽代碼

- 表達演算法的方式主要有兩種:流程圖和偽代碼
- 大部份的程式都是用程式語言撰寫出來的，這些語言有特定的語法。偽代碼不是一個真實的程式語言，而是一個簡單的方式去表達一系列的指示，而這只是並不需要遵從特定的語法。
- 撰寫偽代碼很像撰寫真實的程式語言，每一個步驟都被描述在所屬的順序的那一行。一般來說指示用大寫書寫，變數用小寫書寫而訊息則用一般方式書寫。

表達演算法:偽代碼

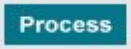
- 在偽代碼中，input 表達問一個問題，output 表達顯示在螢幕
- 一個簡單的程式可以簡單地詢問使用者的姓名以及年紀並且依據使用者的回答給予不同的回應。這個簡單的程式偽代碼如下:

```
OUTPUT 'What is your name?'
INPUT user inputs their name
STORE the user's input in the name variable
OUTPUT 'Hello' + name
OUTPUT 'How old are you?'
INPUT user inputs their age
STORE the user's input in the age variable
IF age >= 70 THEN
    OUTPUT 'You are aged to perfection!'
ELSE
    OUTPUT 'You are a spring chicken!'
```

表達演算法:流程圖

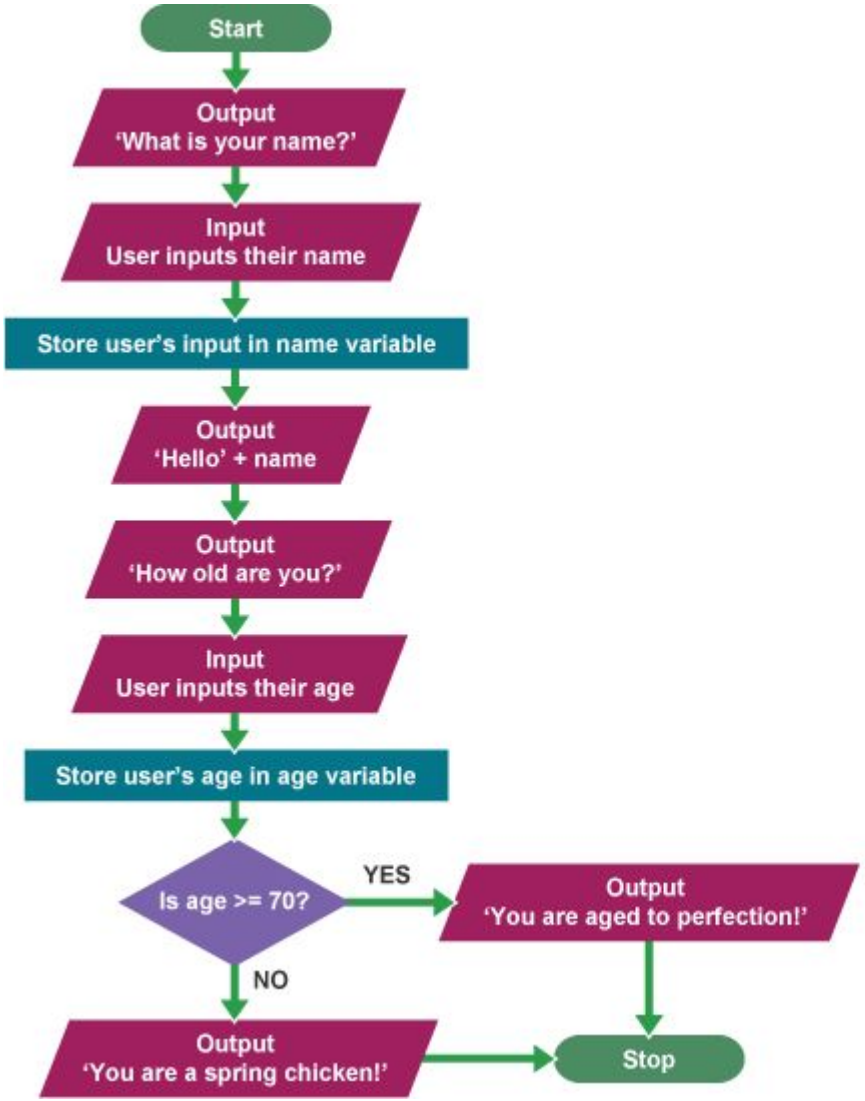
- 流程圖是一種表達一系列指示的方式，流程圖一般用不同的圖示來表達不同的指示，沒有什麼規定去規範流程圖應該要多詳細，有些流程圖被分解成很多個小步驟去提供很多的細節，而有些則很簡化，將幾個步驟合成一個步驟。

流程圖指示

名字	圖示	用途
開始 / 停止		一系列指示的開始或結束
流程		指示或是命令
決定		決定:是或否
輸入 / 輸出		從電腦接收出入 從電腦發出訊息
連結點		從一個點跳到另外一個點
移動方向		將符號連結起來，箭頭指向下一個進行的指示

流程圖圖示

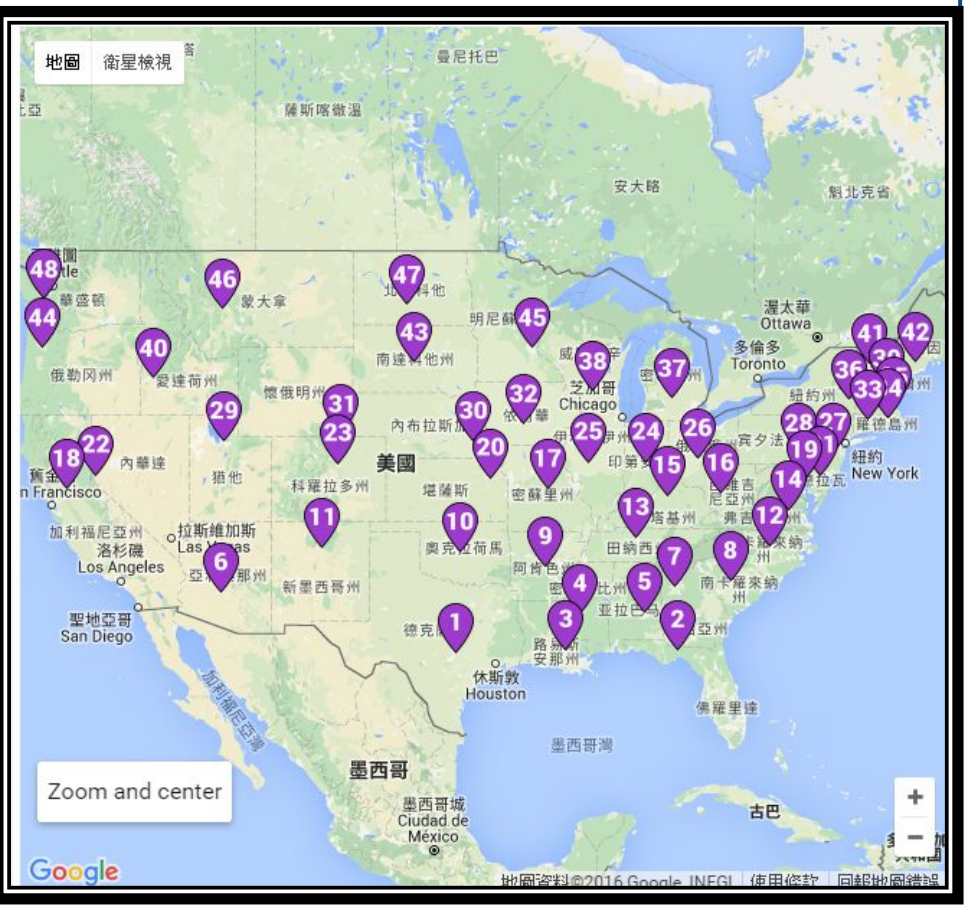
- 一個簡單的程式範例(根據使用者的姓名以及年齡給予回答)可以被表達成右方的流程圖

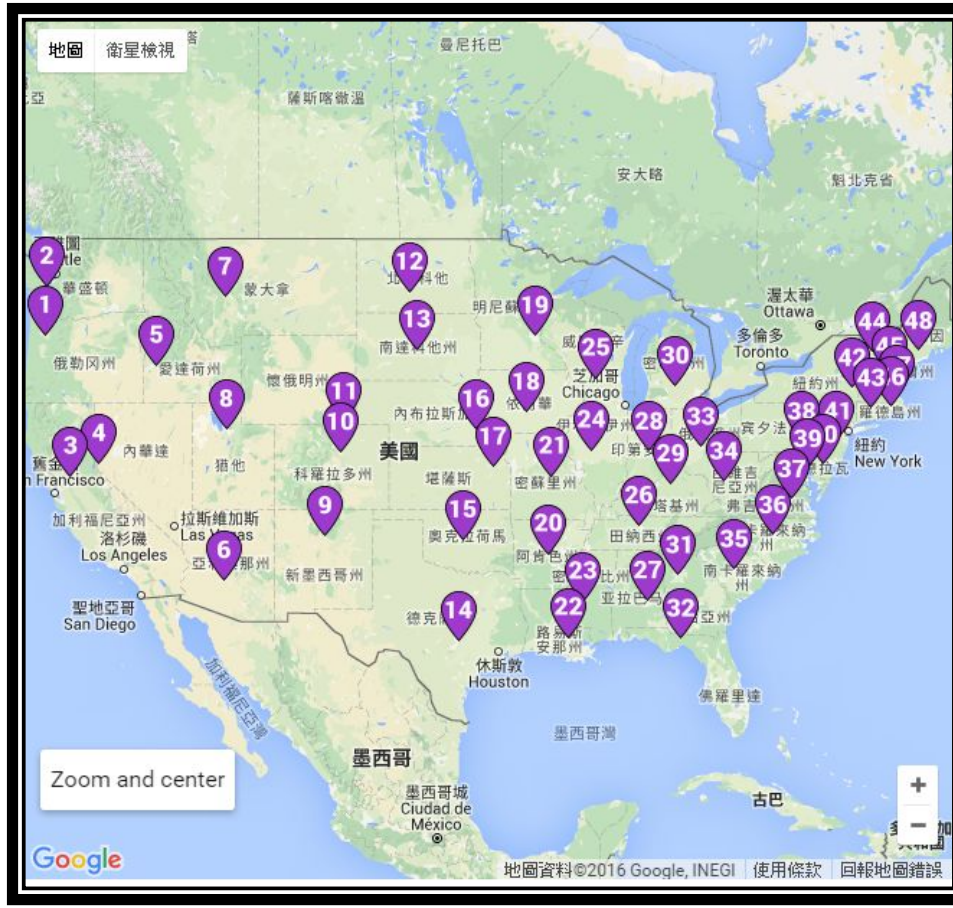


Traveling(旅遊)

- 您和您的家人正計劃通過所有在該地區首府客場之旅，你必須搞清楚的最短路線可以節省燃料費用的工作。
- 嘗試通過選擇運行示例客場之旅的一個緯度，經度，或隨機從下拉菜單中。看到結果後，嘗試切換到可拖動和完善城市為了一個你認為最適合。
- 課程練習網址
- https://computationalthinkingcourse.withgoogle.com/unit?unit=2&lesson=3

旅遊路線安排


地圖
衛星檢視
Zoom and center
Google
地圖資料©2016 Google, INEGI
使用條款
回報地圖錯誤


地圖
衛星檢視
Zoom and center
Google
地圖資料©2016 Google, INEGI
使用條款
回報地圖錯誤

旅遊路線問題考量

- 你從開始哪個城市？
- 請問你的策略用於開發路線(縱向，橫向等)，適用於任何國家或它依賴？嘗試切換到另一個國家，看看它的工作原理，以及在那裡。
- 哪些原因這個問題很難？