運算思維 與
程式語言教學增能

蔡淑玲 *Shalley*
Kids Coding Studio

- Kids Coding Studio 簡介
- 什麼是運算思維？
- 透過 Scratch 學習運算思維
  - ScratchEd 創意運算教材
  - 簡介 Coding for Fun 線上課程
  - 體驗 Coding for Fun
- Scratch 的下一步
- Q & A

關於我

- 學歷
  - 美國德州大學奧斯汀分校 電機與資訊工程學系碩士
  - 台灣交通大學 電機工程系學士
- 經歷
  - 16年業界嵌入式系統軟體開發，歷任經理/軟體工程師
  - 工研院電腦與通訊研究所、資策會
  - 明碁電腦、台達電子、普樺科技

教學經驗

- 北一女中(1995年)
  - 一學期高一電子計算機課程
  - 只有 DOS 的年代
  - 教 BASIC，Gopher
  - 學生只愛玩 BBS

教學經驗

- 新北市桃子腳國中小志工
  (2008~2012年)
  - 國二、中高年級的課輔志工
  - 中高年級的「節能減碳」志工老師
  - 高年級的「早晨Do科學」志工老師

教學經驗

- 兒童程式教學與推廣 (2012~)
- 橘子蘋果程式設計學苑 (2012~2014.2)
  - 共同創辦人/老師
  - Scratch
  - Udacity 課程 (Python) —— Intro to Computer Science：

    Build a Search Engine & a Social Network (計算機概論:建造一個搜尋引擎與社群網路)
- **Kids Coding Studio (2014.2~)**
  - 創辦人/老師/工友
  - 教材開發
  - Scratch: Coding for Fun 線上課程 (2014年底完成)
  - Python 程式設計初級課程（線上教學）

Kids Coding Studio 簡介

Kids Coding Studio 成員

- **Shalley Tsay**
  - 創辦人/老師/工友
  - 設計教材/教學
- **Michael Pan**
  - 高一自學生
  - 程式設計師
  - 網頁前後端程式撰寫
- **Arthur Pan (Ph.D)**
  - 顧問
  - 20多年資訊業界經歷（熱愛寫程式）
  - 交通大學 資訊科學博士
  - 美國德州大學奧斯汀分校 電腦科學系碩士
  - 台灣大學 資訊工程系學士

Kids Coding Studio 目的

- 建立 Young Coder 族群
  - 專注於中小學學子
  - 促進年輕的程式學習者做交流
- 開發程式教材
  - 降低兒童及青少年學習程式的障礙
  - 激發學習興趣
  - 培育程式創作能力
- 與老師家長分享教學經驗
  - 老師培訓
  - 翻轉教室

Kids Coding Studio 目的

- 建立 Young Coder 族群
  - 程式工作坊 (每週一次，共學)
  - 程式俱樂部 (每月一次)
- 開發程式教材
  - Coding for Fun 免費 Scratch 線上課程
  - Python 程式語言初級課程
- 與老師家長分享教學經驗
  - Scratch 師資培訓 (公益性質，開班無需授權)

Kids Coding Studio

- 搜尋：兒童程式設計
- 部落格：http://kidscoding.tw
- FB粉絲團：https://www.facebook.com/kidscodingtw/
- 網站：http://coding4fun.tw
- email：coding4fun.tw@gmail.com

什麼是運算思維？

(Computational Thinking)

簡單的說，就是解決問題的方法

運算思維

- 面對複雜的問題，能夠**理解問題本質**、發展可能的**解決方案**
- 然後使用**電腦**、**人**或**兩者**都可以理解的方式來呈現這些解決方案

運算思維
(Computational Thinking)

- Decomposition (拆解)：將複雜的問題或系統拆解成更小、更易於管理的問題；
- Pattern Recognition (模式識別)：為了讓解決問題更有效率，將每個小問題分別檢視，思考之前是否有解過類似的問題
- Abstraction (抽象)：只注重重要的細節，而忽略不相關的資訊
- Algorithms (演算法)：設計簡單的步驟或規則來解決每個小問題

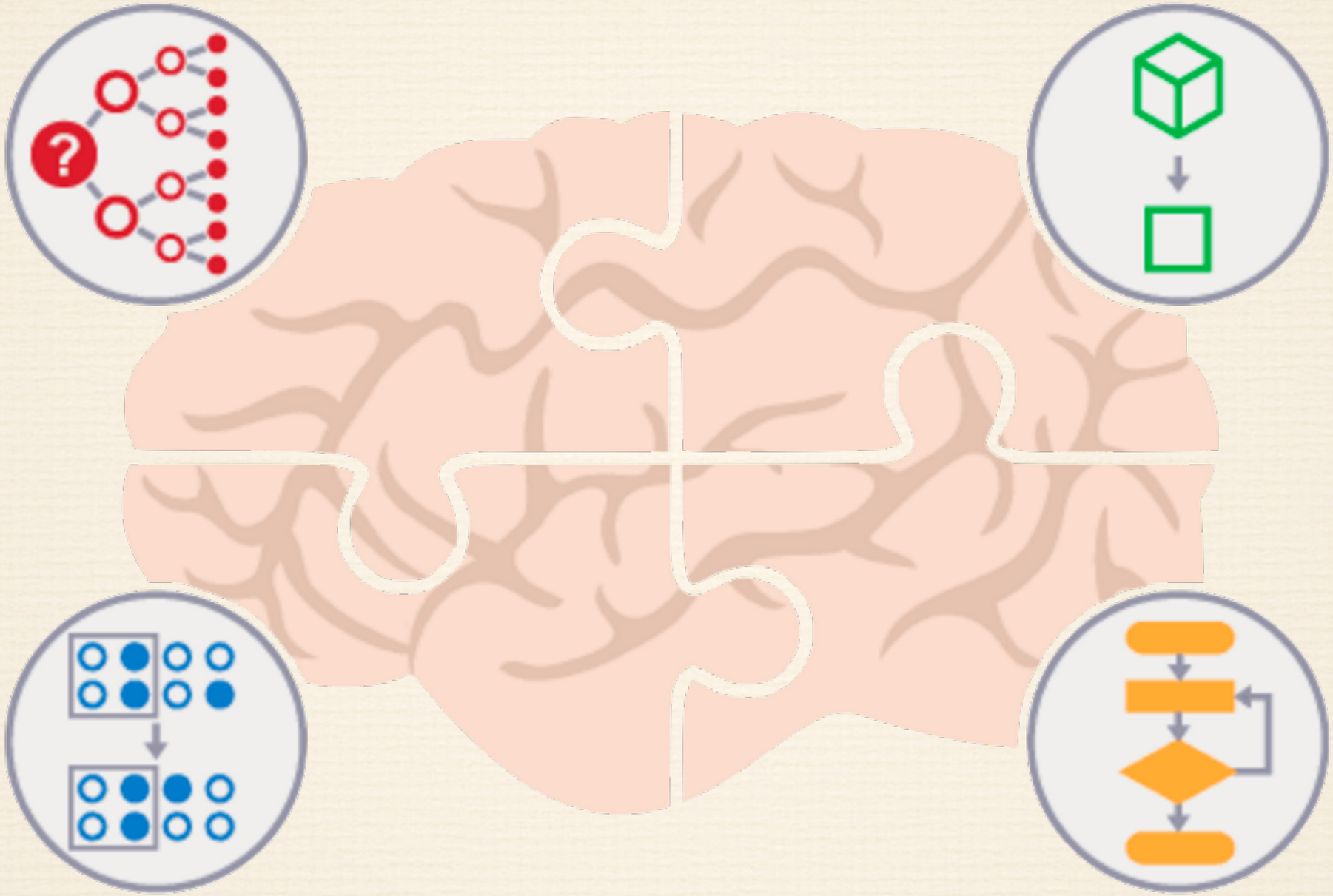
運算思維

- 最後，將這些簡單的步驟或規則寫成程式
- 運算思維並不是寫程式，更不是像電腦一樣思考；因為電腦不能也不會思考
- 程式語言和運算思維無關，只是實現解決問題的<u>工具</u>

拆解

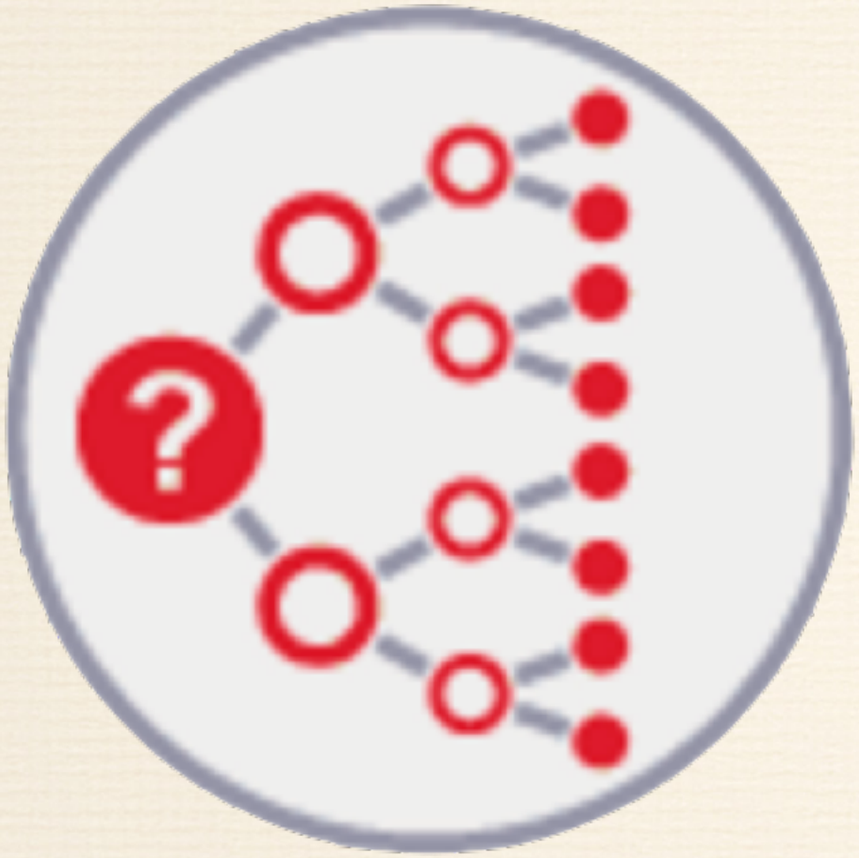
Decomposition

拆解



圖：BBC

為什麼「拆解」很重要？



- 如果不將問題「拆解」，要順利地解決是愈加的困難
- 將問題「拆解」，才能詳細地檢查每個更小的問題
- 使用「拆解」更容易了解一個複雜的系統

範例一：刷牙

每天都做，通常想都沒想.....

- 要拆解「刷牙」，需要考慮：
  - 用哪支牙刷
  - 刷多久
  - 刷牙要多用力
  - 使用哪個牙膏

範例二：偵辦犯罪

- 警官要搜集一系列的(小)答案：

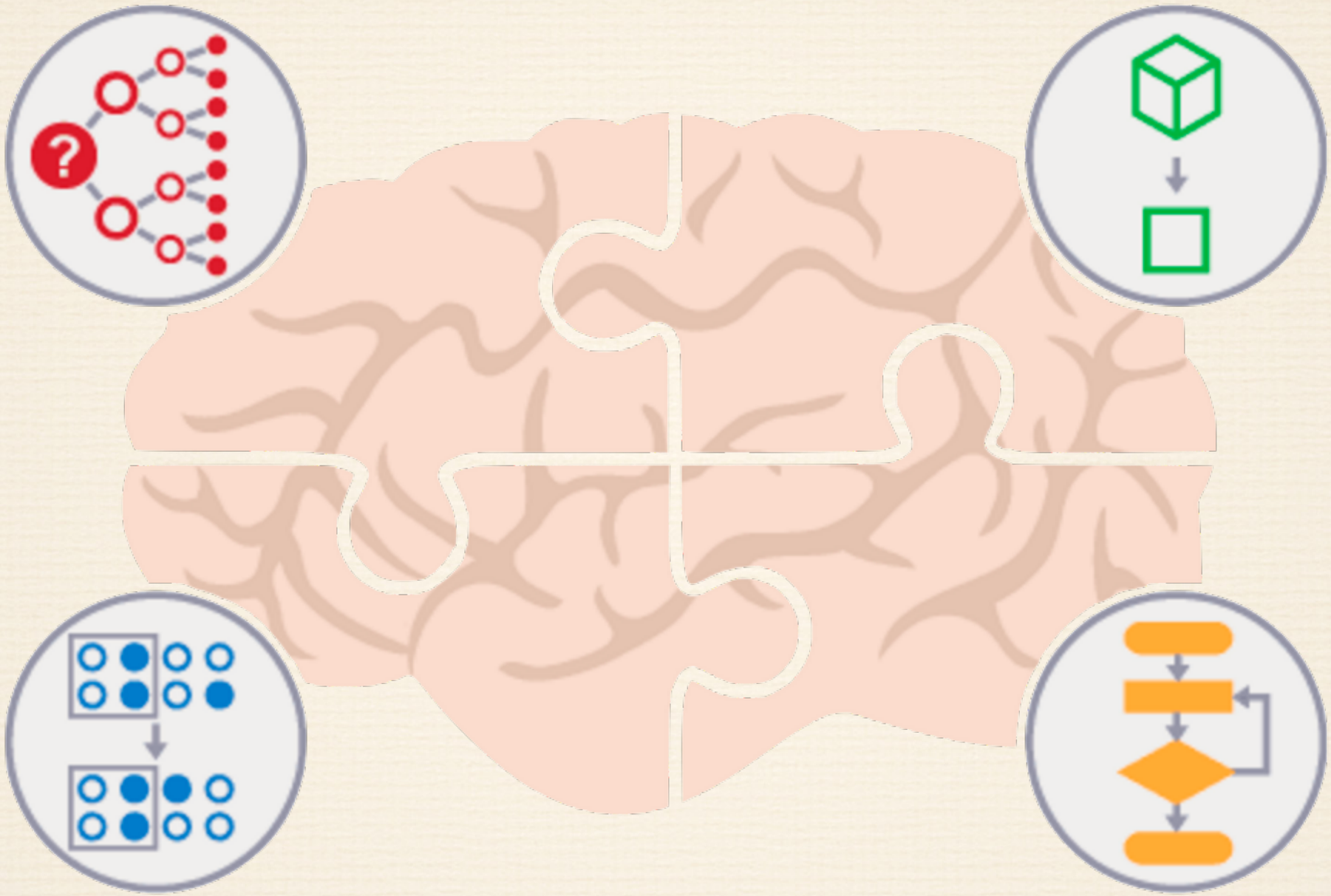
圖：BBC

範例三：做個 app

- 哪種類型的 app
- app 看起來像什麼樣子
- 使用對象
- 使用的圖形
- 使用的聲音特效
- 使用的程式語言或工具
- 使用者會如何使用 app
- 如何測試 app
- 在哪裡賣 app

模式識別

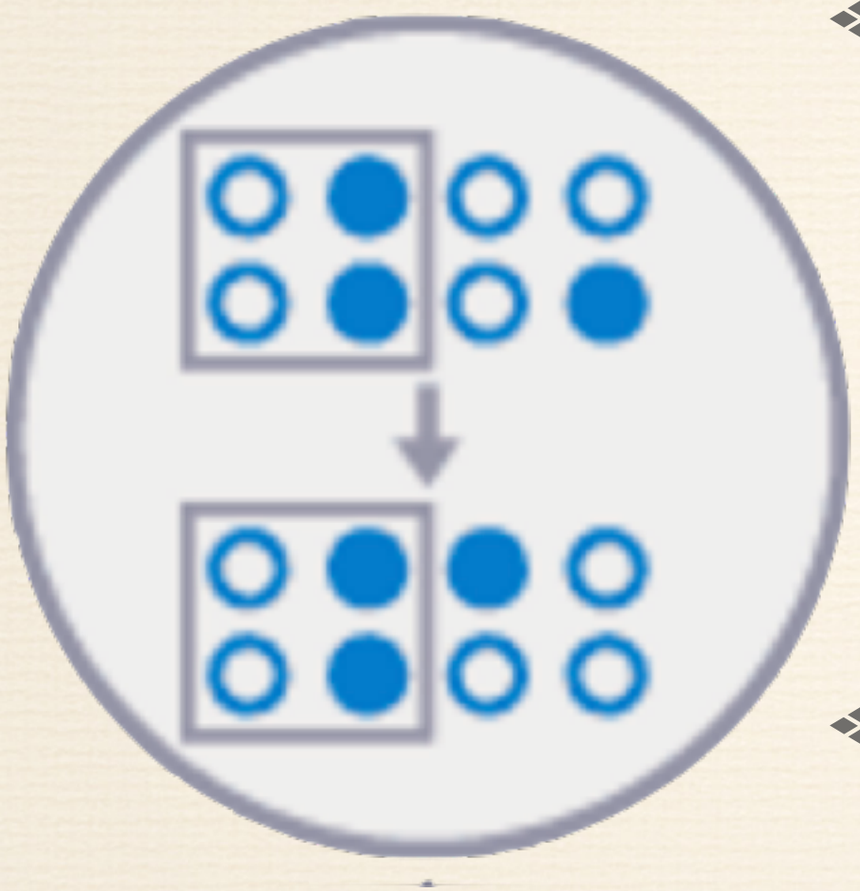
Pattern Recognition



模式識別

圖：BBC

什麼是「模式識別」？



- 將複雜的問題拆解後，經常會發現小問題之間有一些模式
  - 模式是問題之間的<u>相似處</u>，或是<u>特性</u>
- 找到這些模式，可以讓我們更<u>有效率</u>地解決複雜的問題

什麼是模式 (pattern)？

- 貓有共同的特徵
  - 眼睛、 尾巴和毛皮
  - 喜歡吃魚和發出喵喵聲
- 有了這些共同的特徵，就能試圖描述一隻貓
- 在運算思維，這些特徵被稱為模式
- 一旦我們知道了如何描述一隻貓，按照相同的模式，就可以描述其他的貓

❖ 不同之處可能是：

圖：BBC

為什麼要尋找「模式」？

- 模式讓任務變得簡單
- 找到了模式，可以使用相同的解決方案
- 找到愈多模式，解決整個問題會更容易、更快

圖：BBC

如果沒有找到「模式」...

- 如果沒有找到貓的「模式」
  - 每次畫貓時，必須停下來思考貓的樣子
  - 仍然可以畫出貓，它們也像貓，但每隻貓會花掉許多時間
  - 效率很差
- 此外，我們可能不會注意到貓有眼睛、尾巴、和毛皮
  - 畫出來的貓，可能不像貓
  - 這種情況下，如果沒有辨識出模式，可能就會將問題解錯

識別「模式」

- 在問題之間，試著找出**相同的模式**，或**非常相似的模式**
  - 即使可能不存在共同的特徵，仍應嘗試看看
- 模式存在於**不同問題之間**，也可能存在**單一問題**裡
  - 兩者都應尋找

不同問題之間的模式

不同問題之間的模式

- **烤蛋糕**可以拆解成這些小問題：
  - 什麼蛋糕
  - 成分是什麼，需要的量有多少
  - 幾人份
  - 要烤多久
  - 每種成分在哪個時間點加進去
  - 需要什麼設備

不同問題之間的模式

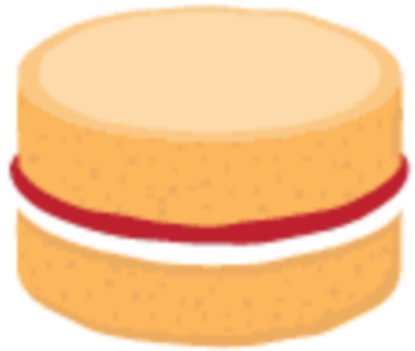
- 知道如何烤某一類型的蛋糕，要烤出另一類型的蛋糕就變簡單。因為存在模式
- **烤蛋糕的模式**：
  - 每種蛋糕的成分必須精確
  - 在特定的時間添加成分
  - 每種蛋糕有其烘焙的特定時間
- 有了模式，不同的問題可以使用共同的解決方案

預熱烤箱到 190 度C

混合奶油、糖、和麵粉

烤 25 分鐘

攪拌 300 cc 的鮮奶油



預熱烤箱到 180 度C

攪拌所有的奶油與糖

混合雞蛋

烤 30 分鐘

圖：BBC

單一問題裡的模式

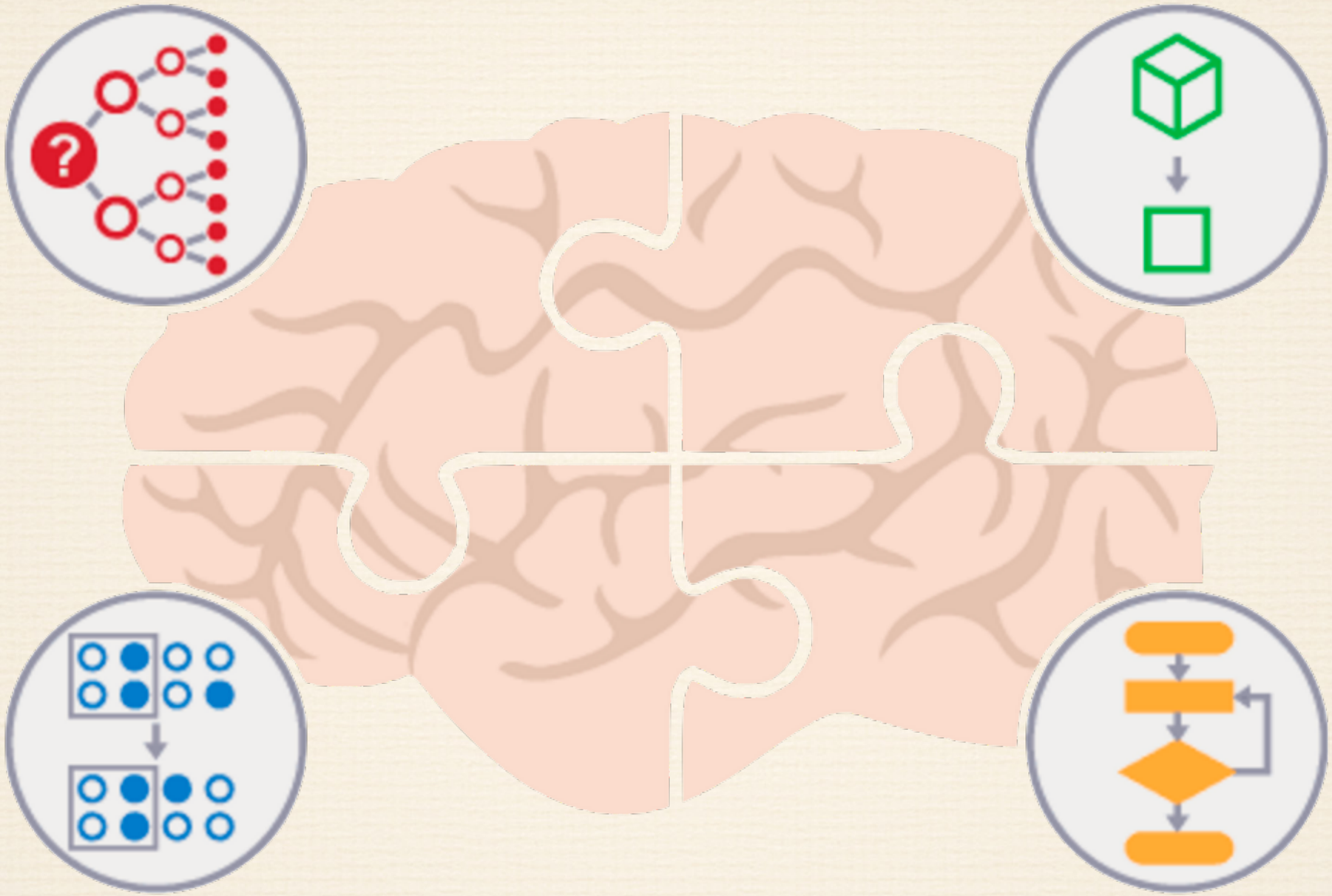
單一問題裡的模式

- 對於烤蛋糕，小問題之間也能找到模式
- **蛋糕的成分必須有精確的**
  - **數量**
  - **測量方法**
- 一旦確認成分和它的測量方法，就能將這種模式應用於所有成分

抽象

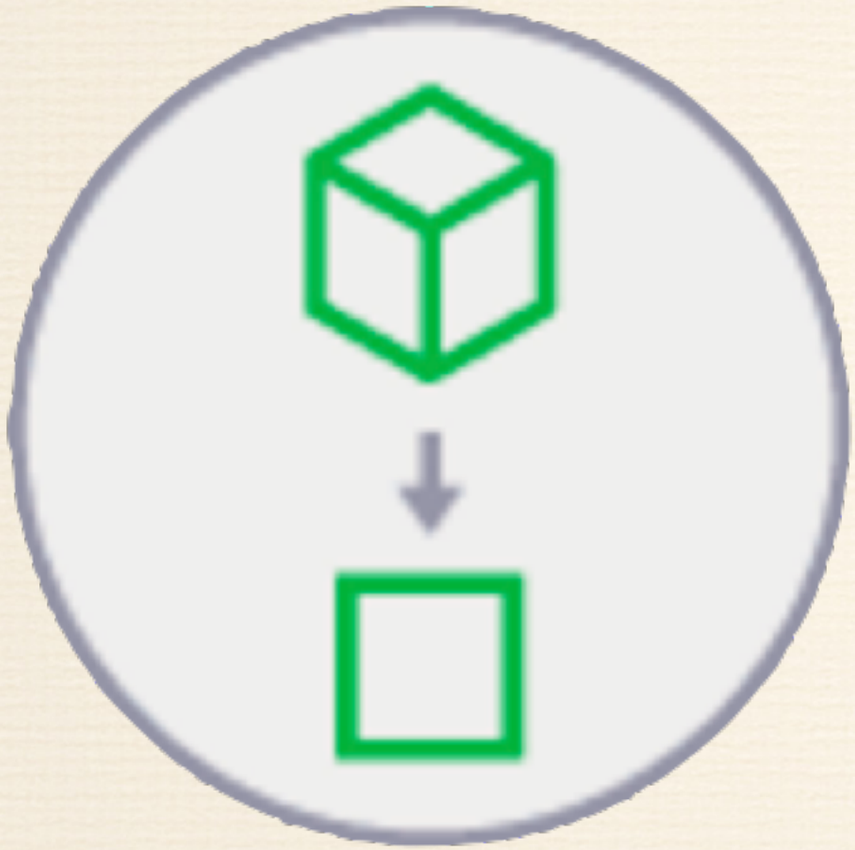
Abstraction

抽象



圖：BBC

什麼是「抽象」？



- 化繁為簡
- 聚焦在重要的特徵，忽略不必要的資訊
- 同時也濾掉具體細節

什麼是「特點」和「細節」？

- 所有的貓都有的特點：
  - 眼睛、尾巴、皮毛、喜歡吃魚和發出喵喵聲
- 每隻貓有其具體的特點
  - 如黑色皮毛、 長尾巴，綠眼睛，愛三文魚，叫聲響亮等
  - 稱為<u>細節</u>

- 要**繪製**基本的貓
  - **只要知道**它有尾巴，毛皮和眼睛；**這些特點是相關的**
  - **不需要知道**貓發出什麼聲音，或是它喜歡吃哪種魚；**這些特點是無關緊要的**
  - 貓有尾巴，皮毛和眼睛，但不需要知道大小和顏色；**這些細節可以過濾掉**

- 從貓的一般特徵，我們可以建立一隻貓的形象
- 可以畫出貓基本的樣子

為什麼「抽象」很重要？

- 抽象讓我們對於問題的本質與如何解決問題，能夠建立大略的想法
  - 這個過程引導我們**刪除所有的細節**，或是**移除對解決問題毫無用處的模式**
  - 這有助於我們形成對問題的想法。這種想法稱為**模型(model)**
- 不將問題抽象化，**可能會放錯重點**，得到錯誤的解決方案

為什麼「抽象」很重要？

- 以貓的例子來看
  - 如果缺少抽象的過程，我們可能認為所有的貓都有長長的尾巴和短毛
  - 有了抽象的過程，我們知道貓有尾巴和毛皮，但不是所有的貓尾巴都很長，並不是所有的毛是短的
  - 抽象幫助我們對「貓」塑造了清晰的模型

如何「抽象」化？

- 烤蛋糕時，不同的蛋糕都有一般特性

一般相關的模式 General patterns	具體細節 Specific details
蛋糕需要成分	不需要知道需要哪些成分
每種成分都需要指定數量	不需要知道數量是多少
蛋糕需要計時	不需要知道時間長短為何

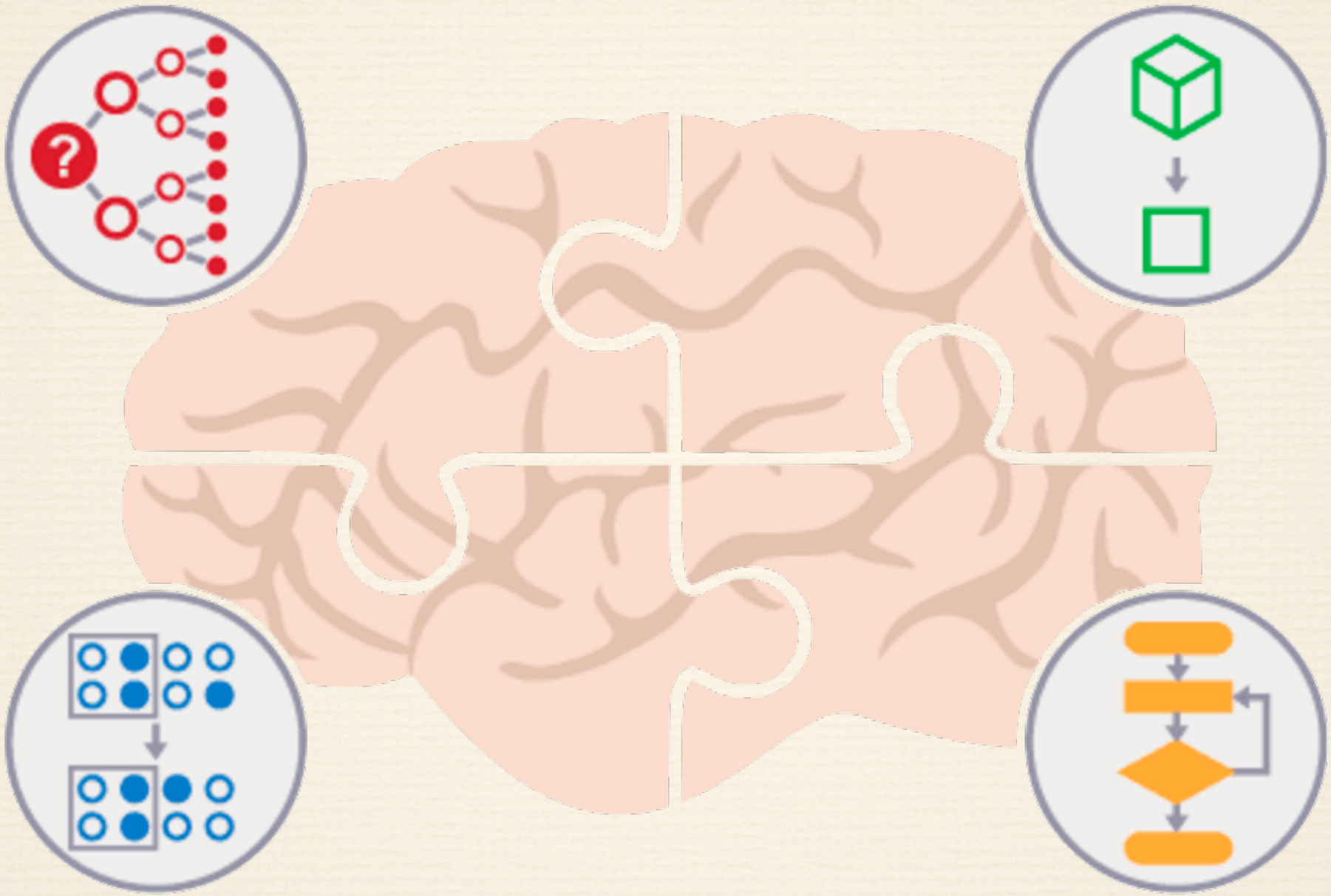
- 刪除具體細節，保留一般相關的模式

創立模型

- 模型是我們解決問題的整體思路
- 模型貓會是任何一隻貓
  - 不是某隻有長尾巴與短毛皮的貓，模型表達所有貓的形象
  - 從貓的模型，根據貓共有的模式，可以知道貓的外貌
- 烤蛋糕時，模型蛋糕不是特定的蛋糕
  - 不是海綿蛋糕或水果蛋糕
  - 從「烤蛋糕」的模型，根據烤蛋糕的共有模式，可以知道如何烤蛋糕
- 有了問題的**模型**，可以設計演算法來解決它

演算法

Algorithms



演算法

圖：BBC

什麼是「演算法」？

- 演算法是一組計畫
- 規劃解決問題的詳細步驟

泡茶

解魔術方塊

繫鞋帶

摺衣服

圖：BBC

- 解決問題需要規劃，使用**演算法**來確認每個指令以及執行的先後順序
- 如果要讓電腦來解決問題，必須寫程式來告訴電腦，**做些什麼**以及**如何做**，按部就班 (step-by-step)
  - 給電腦差的演算法，就會得到差的結果
  - 演算法應用於很多地方，包括計算、 資料處理和自動化

演算法

- 演算法必須明確
  - 要有起始點、終點、以及之間的明確指令
- 演算法通常是寫作程式的起始點
  - 虛擬程式碼 (pseudo code)
  - 流程圖 (flow chart)

虛擬程式碼 Pseudo Code

- 類似程式碼，但沒有嚴苛的程式語法
- 重點在「流程」與「內容」的描述

- 作為程式開發的藍圖，提高程式的可讀性
- 撰寫時要能夠簡潔地描述每一個執行步驟

Pseudo Code 範例一

- 與流程描述相關的項目直接寫在虛擬程式碼中
  - 邏輯運算子(and, or, not, …)
  - 條件運算子(==,!=, >, …)
  - 條件判斷(if, else, switch, …)
  - 重複執行(for, while, do while, ...)

- 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!'

Pseudo Code 範例二

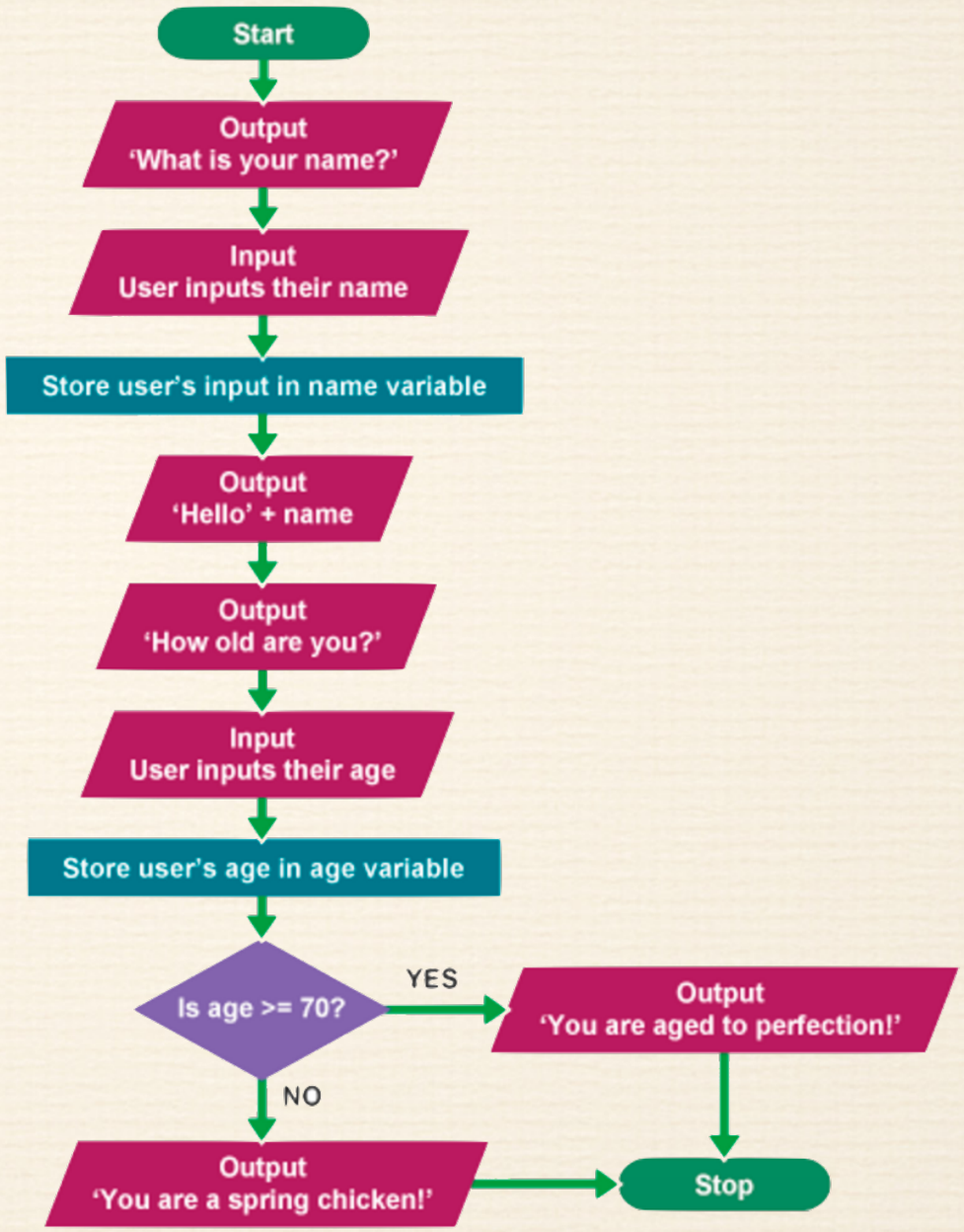
- 也可以直接以口語化的中文或英文撰寫要執行的工作內容

- 求兩個自然數的最大公因數(GCD)

1. 輸入兩個自然數 A 與 B。
2. A 除以 B 餘數為 R。
3. 如果 R 為零，則跳至第 5。
4. AB (B 的值給 A)，BR(R的值給 B)，跳至 2。
5. B 即為 GCD。

流程圖

- 使用圖來表示流程
- 沒有規定細節的程度
  - 有時提供大量細節
  - 有時簡化

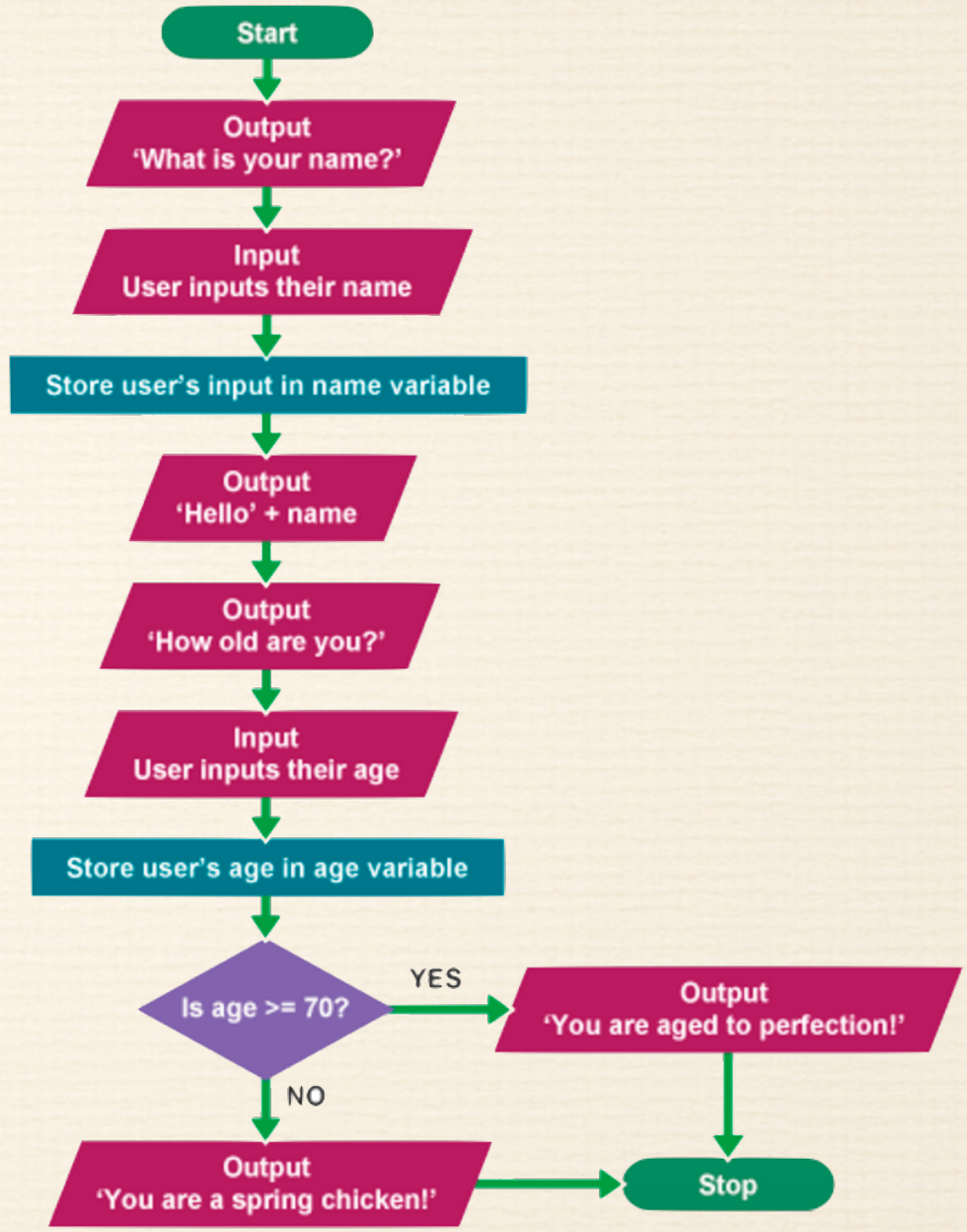


圖：BBC

Pseudo Code

```
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!'
```

流程圖

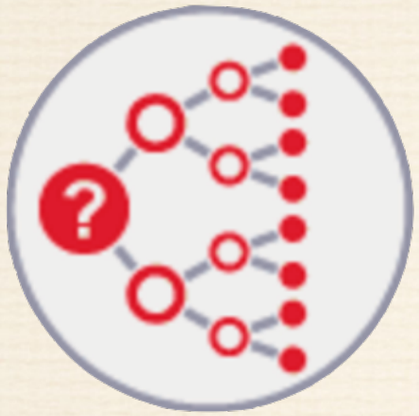


圖：BBC

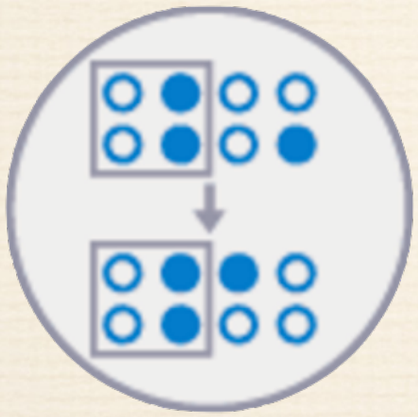
~ Break ~

問題

拆解



一群小問題



尋找

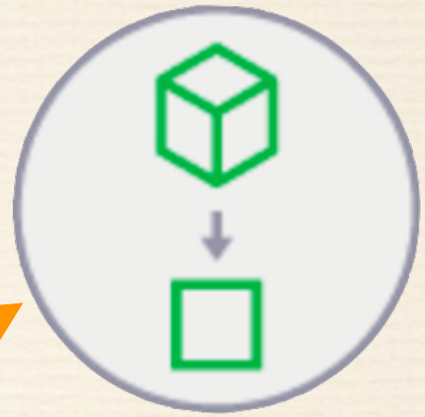
.已存問題有相似模式

.小問題之間的模式

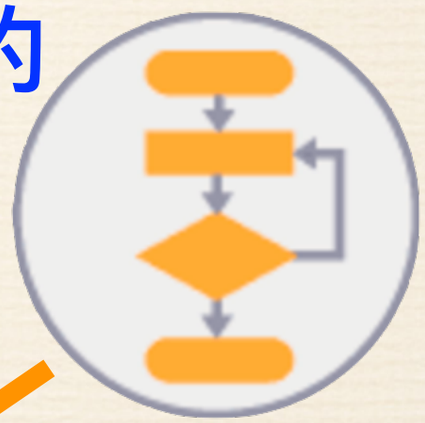
.找出重要資訊

.並過濾細節

.建立模型



解決問題的詳細步驟



.計劃執行

.程式

圖：BBC

透過 Scratch 學習運算思維

什麼是 Scratch？

- 程式積木
- 以玩樂高的方式來組合程式
  - 可以嘗試、重組、把積木散亂地擺放著

Scratch ?

- 每種程式語言都包含了五種基本指令
  - 輸入：從鍵盤、檔案或其他設備得到資料
  - 輸出：在螢幕上顯示，或是將資料傳送到檔案或其他設備
  - 數學：執行數學運算，像是「加減乘除」
  - 條件：檢查特定的條件並執行適合的程式
  - 重複：重複執行某些動作，通常會有一些差異

Scratch

- 美國麻省理工學院 媒體實驗室 為 8 歲以上孩子設計的程式語言
- 具三個特色
  - low floor （低門檻，易學）
  - high ceiling（可以建構複雜的專案）
  - wide walls（支援廣泛、多樣性的專案）

優勢

- 圖像式的表達，具程式設計的雛形
- 隱藏細節，將複雜的程式模組化
- 適合「運算思維」的學習
  - 比一般程式語言入手快
  - 短時間就能學會並應用
  - 可以做動畫、遊戲與模擬

經典遊戲

https://scratch.mit.edu/projects/2345919/#fullscreen

https://scratch.mit.edu/projects/687332/#fullscreen

學生製作的動畫

https://www.youtube.com/watch?v=VlJgGF0gn8I

https://www.youtube.com/watch?v=MHsciHYlUUU

ScratchEd 教材

ScratchEd

- 在 2011 年
  - 創意運算：以**設計**為出發點來介紹**運算思維**
- 在 2014 年 8 月
  - 創意運算
- 從三個面向來培養運算思維
  - 運算觀念
  - 運算實作
  - 運算觀點

運算觀念

觀念	描述
序列（sequence）	對某一個任務，確定一系列的步驟
迴圈（loops）	重複執行相同的序列
平行（parallelism）	在同一時間讓許多事同時發生
事件（events）	一件事引發另一件事的發生
條件（conditionals）	根據條件做決定
運算子（operators）	支援數學與邏輯的表達
資料（data）	儲存資料、讀取資料與更新資料

運算實作

實作	描述
反覆地增加（being iteratively and incrementally）	發展了一點點，試試看，然後再發展一些
測試與除錯（testing and debugging）	確保事情能執行，發現錯誤並解決問題
重複使用與混合（reusing and remixing）	在現有的專案或想法上做點什麼
抽象與模組化（abstracting and modularizing）	探索整個問題與其他部分的連結

運算觀點

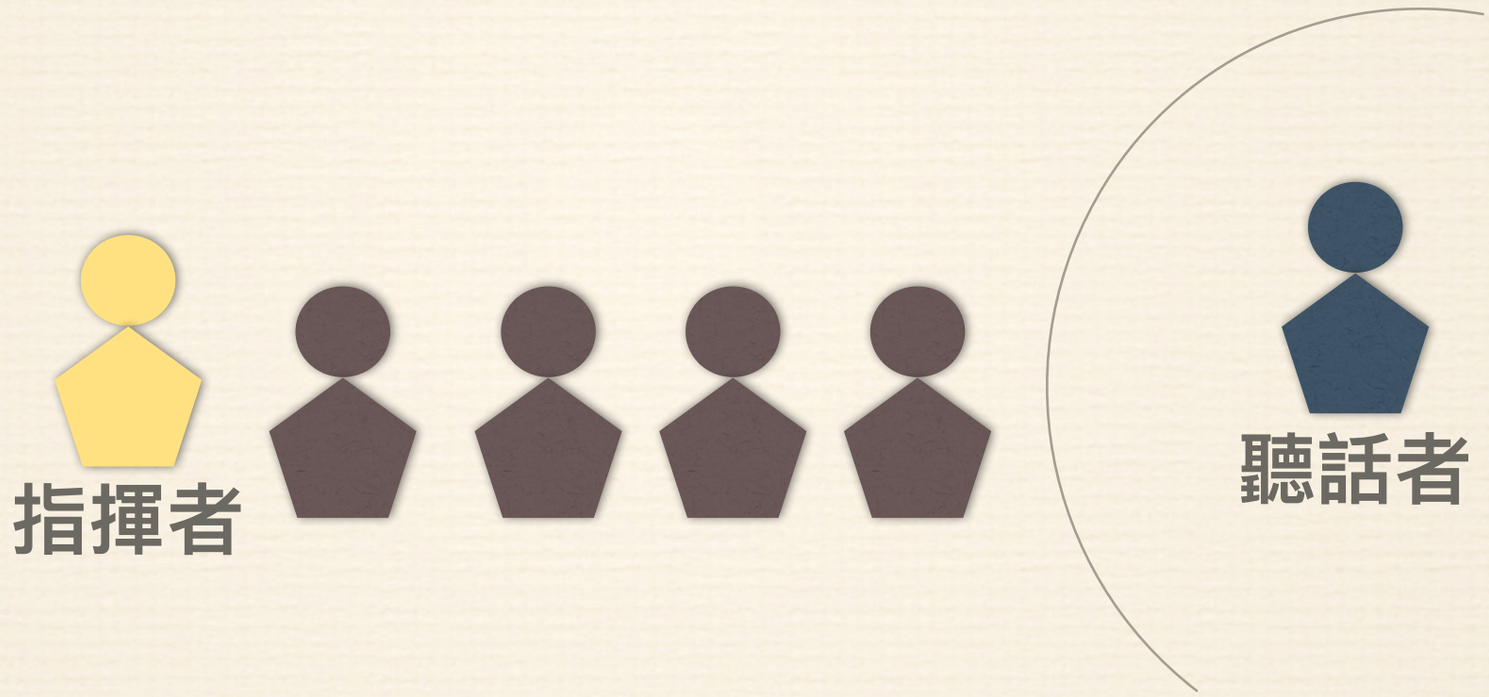
觀點	描述
表達 (expressing)	了解「電腦運算」是一種創作的媒介，「我可以創造」
連接 (connecting)	體認到「與他人一起創造」和「為他人創造」的力量， 「當我有機會接觸他人，我可以做不一樣的事！」
質疑 (questioning)	有能力對世界提出問題， 「我可以（運用電腦運算）提問，並理解（電腦運算）世界！」

教材特點

- 強調**創意**，學生是積極的參與者
  - 以**創作自己的專案**作為孩子寫程式的動機
  - Code to learn，寫程式引導學習，而不是學習寫程式
- 以活動帶領學生感受「為什麼要這樣？」
  - 步調似乎很慢，卻是內化、深刻的學習
  - 例如以「跳舞」讓孩子體驗該如何寫 instruction

來跳舞吧～

投影幕



- 聽話者背對投影幕，指揮者及其他人面對投影幕
- 指揮者和其他人看影片，但聽話者不能看影片

- 指揮者向聽話者形容，如何演出影片中一連串的舞蹈動作
- 只能以口頭說明，愈簡要愈好，不可以有肢體動作

- 當指揮者的容易之處和困難之處有哪些？
- 當聽話者的容易之處和困難之處有哪些？
- 當觀眾的容易之處和困難之處有哪些？
- 這項活動和我們從事 Scratch 有什麼關聯？

如何給出好的指令呢？

教材特點

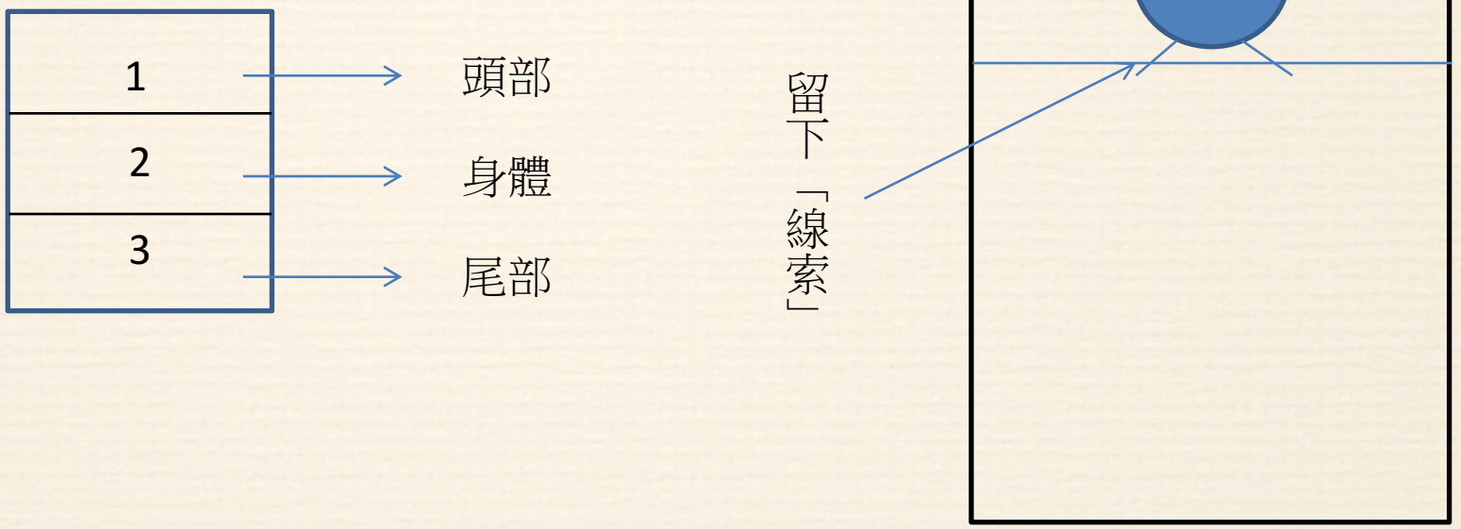
- 強調**探索**
  - 不像一般的程式教學去解釋每個指令
  - 藉著創作的過程，學習更多的程式積木
- 強調**反思**
  - 設計筆記
  - 在活動前/後，讓孩子比較學習成果

教材特點

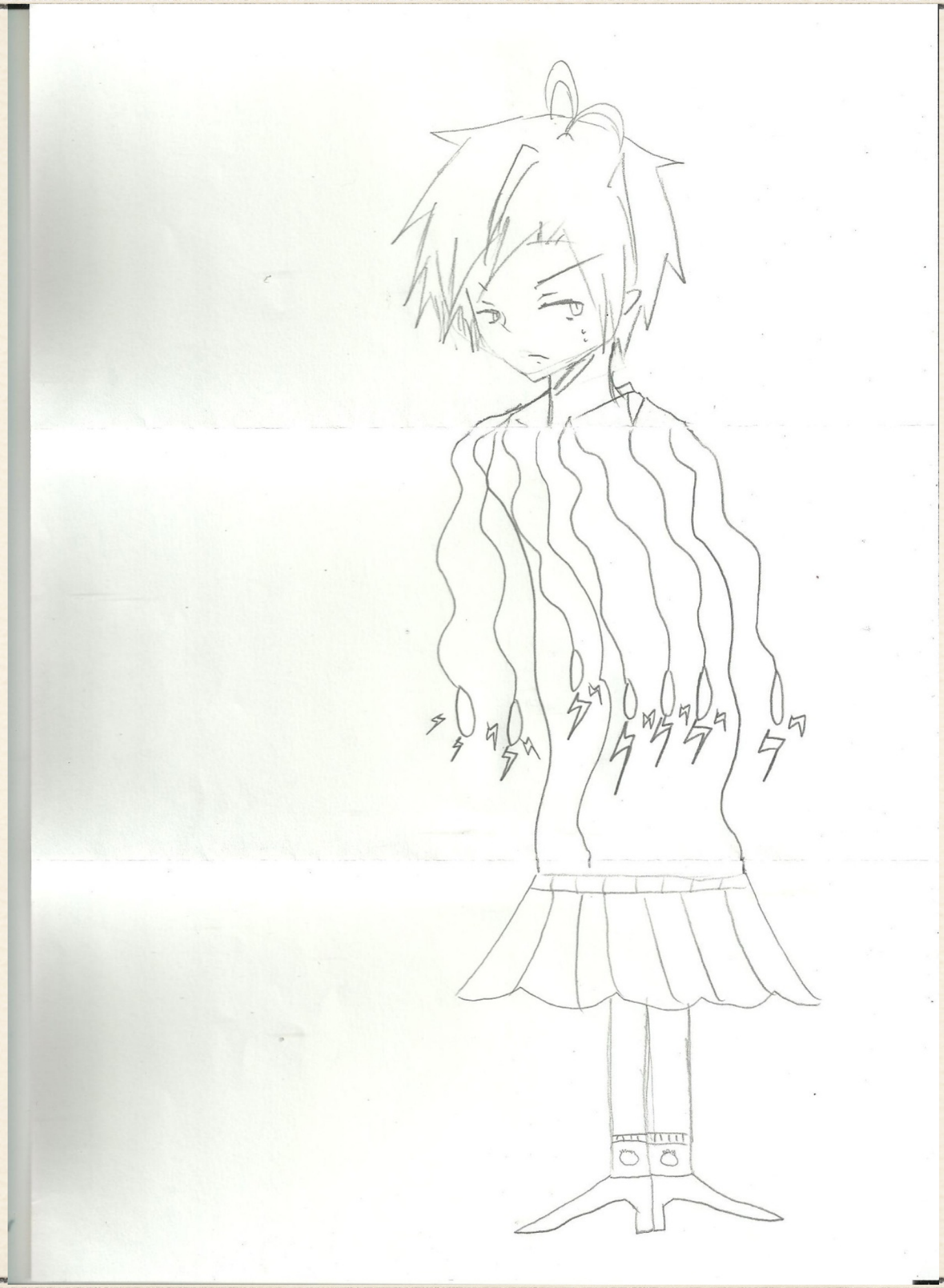
- 小組合作
  - 如何循著別人既有的程式往下發展
  - 如何留線索給後續的人
  - 活動「生命的創造」
  - 活動「傳下去」

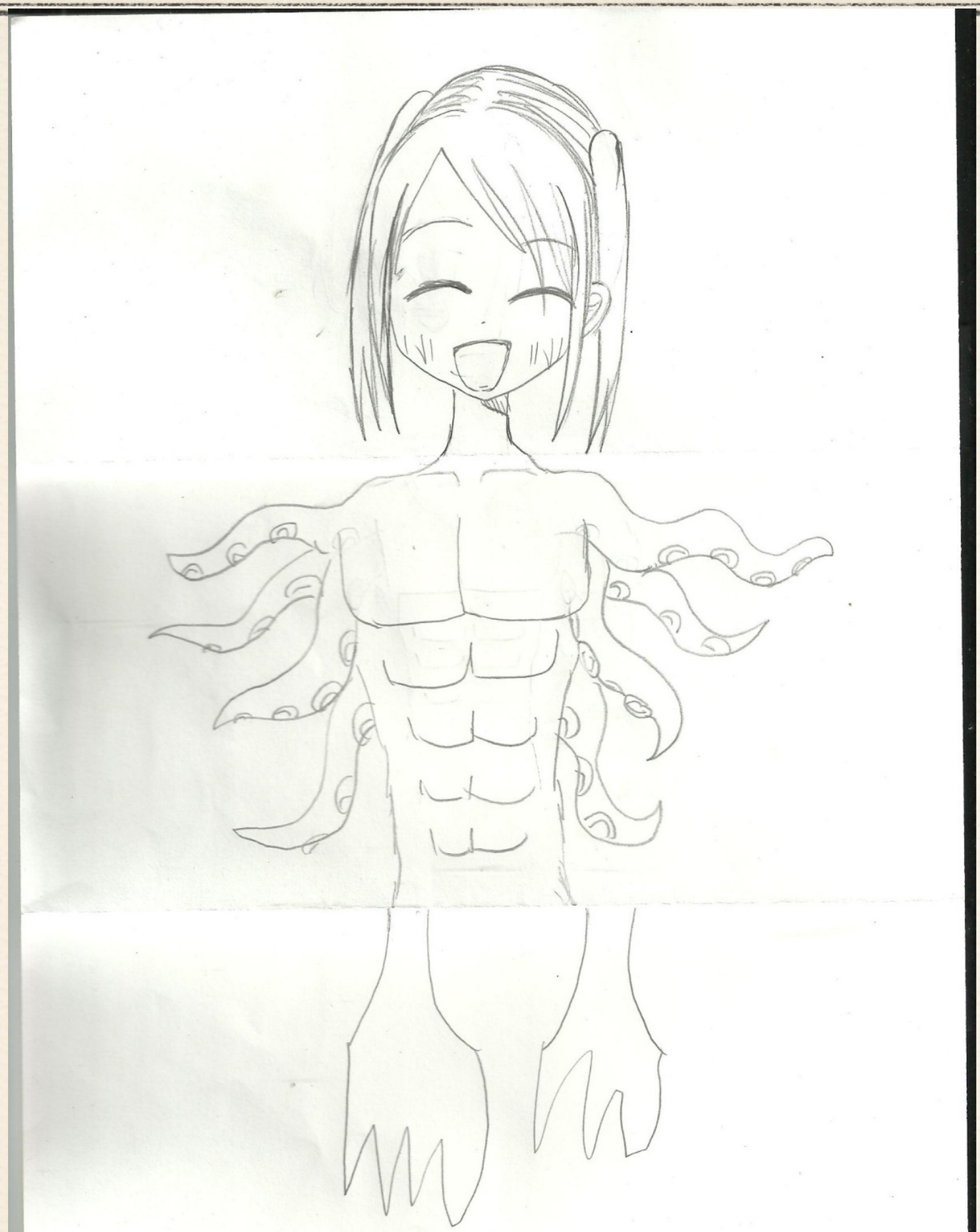
生命的創造

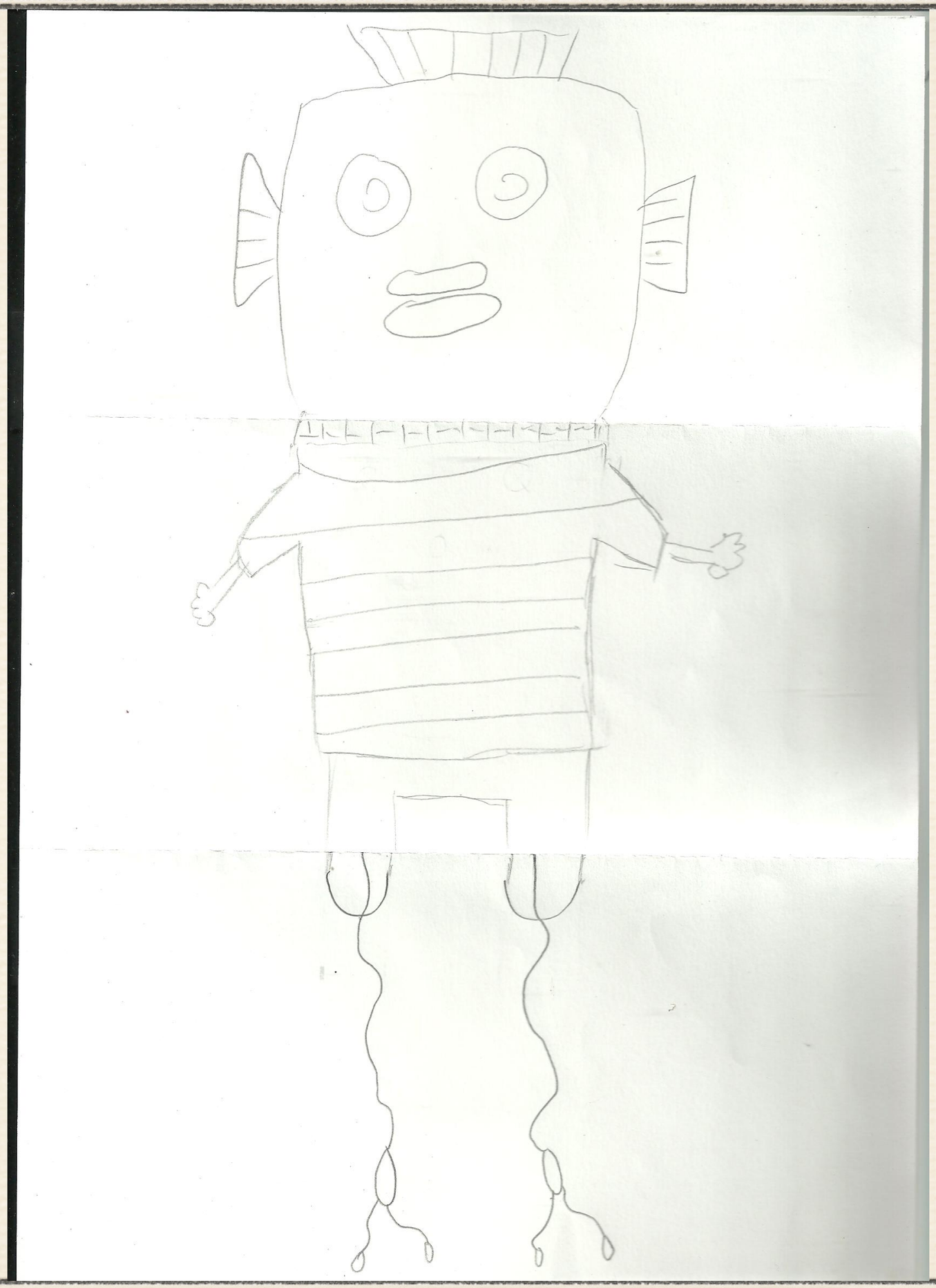
- 白紙摺成三等分
- 每人負責畫生物的<u>三分之一</u>

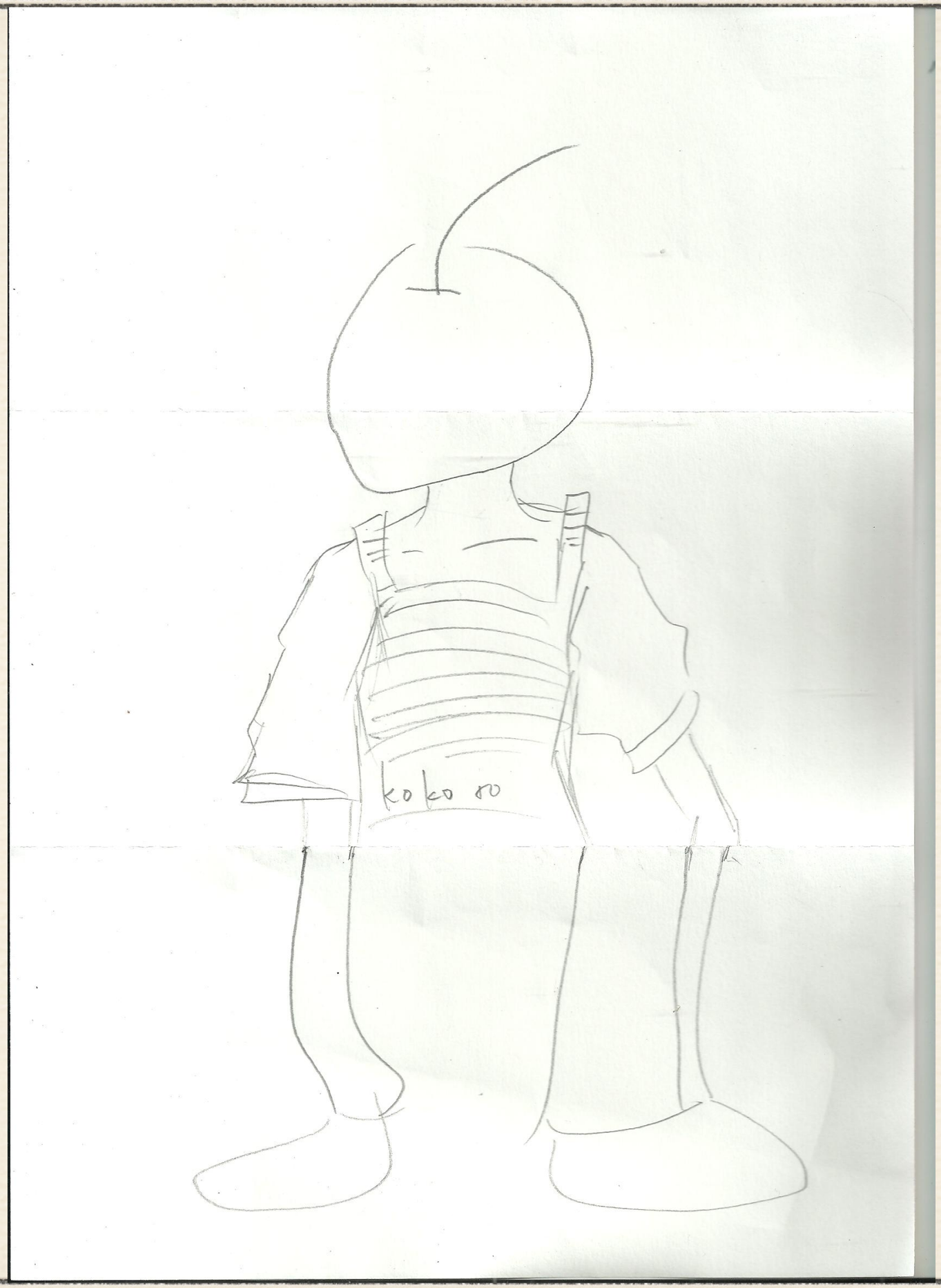


- 不要讓別人看見你畫了什麼
- 留點線條在邊界作為線索








kokoro

傳下去

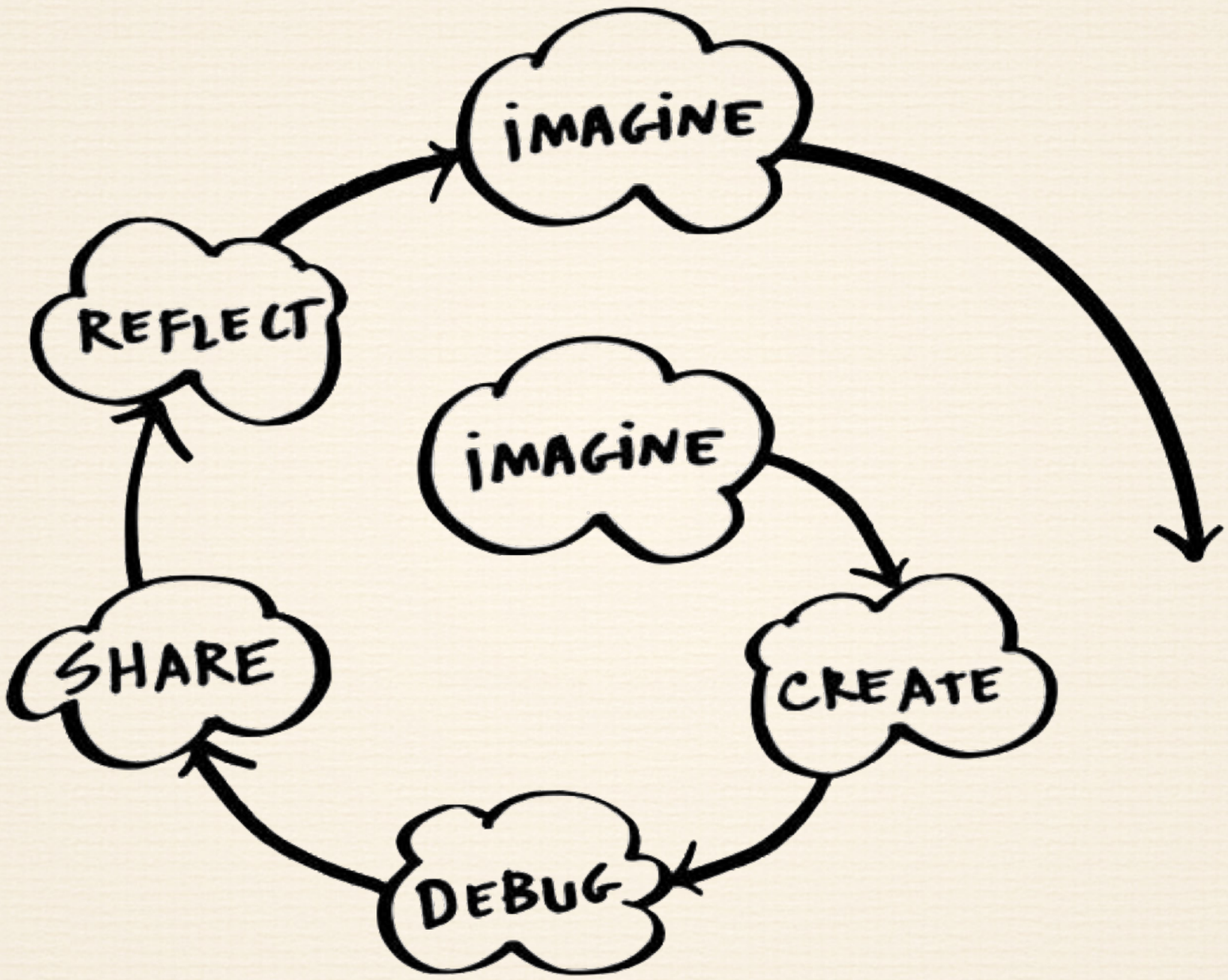
- Remix
- 兩人一組
- 每個專案由 3 個小組接力完成
- 做延伸或是重新設計
- 各10分鐘
- 從任何角色，背景開始.....

教材特點

- 同儕分享
- 走動式分享
  - 厲害的同學有成就感
  - 不知如何下手的同學可以從中學習

教材特點

- 2014.8 新內容
- 提供更多除錯（debug）練習
  - 讓學生設計 debug 題目
- 提供 Scratch 2.0 新積木的學習課程
- 如何評估運算思維的運用程度



Lifelong Kindergarten Group, MIT Media Lab

教學檢討

- 開放式的學習，台灣環境有些水土不服
  - ★學生人數多（20 人已超越極限）
  - ★學生自我學習能力低
  - ★學生稍微分神，後面的嘗試就做不了
  - ★跟上的學生，往往要等落後的同學
  - ★為了讓每個學生都學得到，**老師疲於奔命**

Coding for Fun 是怎麼來的？

結合 ScratchEd 教材與 MOOC

什麼是 MOOC？

大規模開放式線上課程

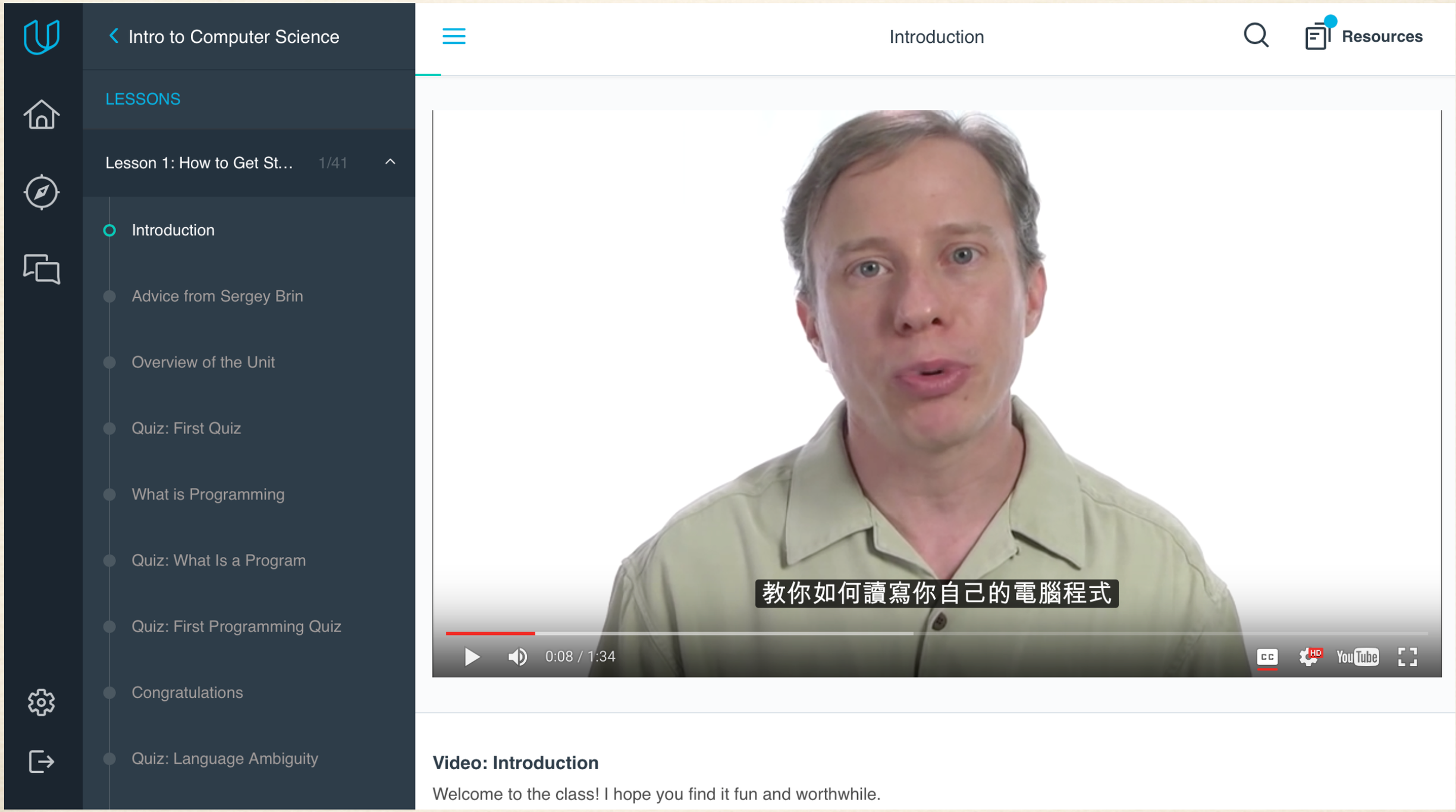
- MOOC (Massive Open Online Course)
  - 磨課師 (台灣)
  - 慕課 (中國)
- 是一種針對於大眾人群的線上課程，人們可以透過網路來學習線上課程
- MOOC 沒有既定的定義，但有兩個顯著的特點：
  - Open access/開放共享：不必是在學生，不需學費，大家共享
  - Scalability/可擴張性：傳統課堂一位老師對應一小群學生，MOOC 裡的「大規模」則提供大量的參與者

與電腦資訊相關的MOOCs

- Udacity (英文，部分有中文翻譯)
- Codecademy (英文)
- 泡面吧 (簡體中文)
- Khan Academy (可汗學院，英文)
- edX
- COURSERA
- code.org

Udacity

Udacity CS101



```
# Write Python code to print out how far light travels
# in centimeters in one nanosecond.  Use the values
# defined below.
# speed_of_light = 299792458   meters per second
# centimeters = 100            one meter is 100 centimeters
# nanosecond = 1.0/1000000000  one billionth of a second
speed_of_light = 299792458
centimeters = 100
nanosecond = 1.0/1000000000


```

VIEW INTRO

RESET QUIZ | TEST RUN | VIEW ANSWER | SUBMIT ANSWER

Udactiy 特色

- 提供了豐富的電腦科學課程
  - 根據職場需求，課程不斷地增加
  - 補習班無法提供如此多元化的課程
- 影片教學，每段影片解釋一個小概念，或是做一個小測驗
  - 學生較能專注，邊學邊動手
  - 即時批改作業
  - 不會，也有解說
  - 像打電動一樣不斷地闖關

如何收費

- 大多數是免費課程
- Nanodegree 付費課程
  - Nanodegree (**$199/月**)：在 12 個月內完成課程，學費退一半
  - Nanodegree Plus (**$299/月**) 就業保證：畢業生保證在六個月內得到工作，否則學費全數退還

Udactiy 人力資源公司？

- 課程來源：
  - Udacity自製
  - 企業/公司：Google、Facebook、at&t、GitHub、Hack Reactor、mongoDB、Zipfian Academy、Amazon……
  - 大學：Georgia Tech (合作開碩士課程) 、San Jose State University
- 介聘人力

Codecademy

Codecademy

- Codecademy 是以程式設計的教學為主，提供「文字」的互動教學，引導學生如何做一步一步地學習
  - HTML/CSS
  - Javascript
  - jquery
  - Python
  - Ruby
  - PHP

MOOC 使用經驗

- Udactiy 是非常好的資源
  - 動態說明，學生在較短時間能抓到重點
  - 不懂，可以再看一次影片
  - 課程廣度、深度俱佳
  - 重視實際應用
- Codecademy 較適合對程式很有興趣，或是已有程式經驗的人
  - 純文字的敘述，不適合剛入門者
  - 專注於程式語言的學習
  - 目前只有英文版，中國有山寨版

Udacity 模式

- 學生對 Udacity 課程的教學適應得很好
  - 國、高中生
- 但是 Udacity 的 Python 入門（計算機概論）對一般學生仍有些難度。課程漏掉了一些基礎的細節
- 所以 2014 年製作了 Python 程式設計初級課程，採用 Udacity 模式
  - 每個影片都很短，一次講一個小概念，或是一個小嘗試
  - 缺點：無法線上批改
- 結果：學生上課不覺得累

Coding for Fun 是怎麼來的？

結合 ScratchEd 教材與 MOOC

Coding for Fun 的歷史

2014.12 ~

Coding for Fun 的歷史...

- 2014 年 8 月 某天清晨 4 點
  - 突然醒來，「Coding for Fun」這個詞浮現腦海
  - 申請網域名稱
- 2014 年 9 月
  - 家長要求我開班，但招生不順
  - 開放式的教學其實很累
  - 同年 4 - 6 月製作 Python 線上課程，上課變省力
  - 開始構思 Scratch 線上課程

Coding for Fun 的歷史...

- 2014 年 10 月開始試用課程，12 月完成課程
  - 老師變輕鬆了
  - 只要解答疑難雜症，學生的問題其實很少
- 2014 年 11 月 ~ 2015 年 1 月
  - FlyingV 群眾募資平台
  - 將課程送進偏鄉
  - 募資並未成功；開始提供台灣地區老師單機版教材
  - 接受網友以買教材的形式贊助 FB 廣告費

Coding for Fun 的歷史...

- 2015 年 3 - 6 月以程式工作坊的形式做實驗
  - 沒有指導老師，由志工帶領，學生自己看影片學
  - 小五以上自學，小四以下親子共學
  - 只有解答問題的助教
  - 家長第一堂課疑問：「沒有老師主講，小孩真的能學嗎？」
  - 家長結束時建議：「這門課不僅教孩子程式設計，也讓孩子學會自學。」

Coding for Fun 的歷史...

- 2015 年 11 月 ~ 2016 年 3 月
  - 製作師資培訓影片
  - 在 Facebook 以 10,000 個讚做號召，公開師資培訓影片
  - http://coding4fun.tw/product
- 2016 年 6 月下旬 ~ 2016 年 7 月上旬
  - 在台北、新竹、台中、台南、高雄舉辦 7 場師資培訓課程。100+人。
  - 對象：學校老師、家長、程式設計師、大學以上在學生

Coding for Fun 教材

Coding for Fun 主旨

- 孩子學習「程式設計」的第一哩路
- 無論城市或偏鄉，每個孩子都有機會，無壓力、快樂地學會新世紀的溝通語言「程式語言」

Coding for Fun 服務對象

- 對電腦操作(熟悉小畫家)，以及文字理解具備基礎能力的學生
  - 建議國小五年級以上自學
  - 國小四年級以下親子共學
- 老師班級教學與經營

協助老師進行教學

- 多數老師並不具備「程式能力」，要如何恰如其分地傳達「程式設計」的概念呢？
  - 交給 Coding for Fun
  - 讓學生去體會、去感受程式設計的魅力，啟動學習意願
  - 按照個人理解能力，自我引導學習
  - 網路學習已是世界趨勢，讓學生建立自學能力
- 補課容易

Coding for Fun 學習目標

- 透過這份教材，學生可以完整地了解積木式程式語言 Scratch
- 學生對於「電腦運算」具備基本的概念，並有能力應用**程式基礎概念**，呈現**創意與想法**

傳統的 Learn to Code

Boring

如果傳統的程式教學方式會成功，
為什麼許多資訊相關科系畢業生選擇轉行？

大學生不能？
小孩的機會當然更小？

Coding for Fun 課程概念一

Code to Learn

在探索中
學習

在創作中
學習

在解決問題中
學習

一步一步「依樣畫葫蘆」地寫程式

Boring

按照課本一個個步驟地寫，
跟學 WORD 等應用程式沒有不同

無法建立程式設計的概念

Coding for Fun 課程概念二

以專案為導向
加入學生的創意與想法

自我介紹

建立樂團

音樂影片

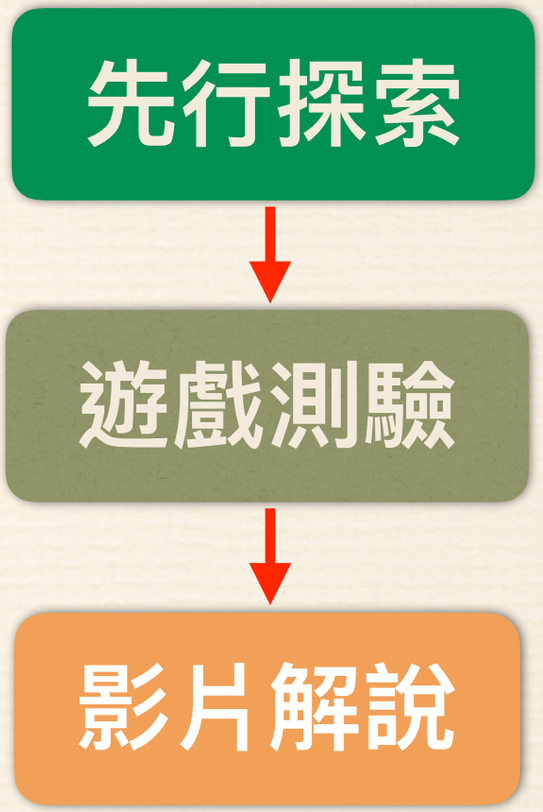
短篇故事

遊戲

Coding for Fun 課程概念三

工欲善其事
必先利其器

了解積木的運作方式
和傳統的教法是倒過來的



Coding for Fun 課程概念四

啟動學習動機

走動式分享

- 沒想法的
  吸取想法
- 有想法的
  有成就感
  成為領頭羊

Coding for Fun 課程概念五

開放教材

修改 Scratch 工作室網址
製作更好玩的遊戲測驗

1 2 3 4 5 6 7 8 9 10

圖形編輯器

動作積木

事件積木

聲音積木

外觀積木

畫筆

積木

內功

條

指令

迴圈

事件

序列

平行

運算觀念

變數

列表

算子

角色舞台

與程式互動

平面座標

音樂元素

其他元素

製造分身

debug

解決問題

debug

Water Slide

自我介紹

建立樂團

音樂影片

作業

遊戲

進階遊戲

自由創作

Coding for Fun 上課需求

- 桌上型電腦或筆記型電腦
  - Windows/OS X/Linux
  - 麥克風
  - 喇叭/耳機
  - 攝影機
  - 可以連上網際網路

~ Break ~

體驗 Coding for Fun

- 網站：http://coding4fun.tw
- 課程：http://coding4fun.tw/scratch
- 教學資源：http://coding4fun.tw/product
- 單機版下載網址：http://goo.gl/JCFDxG

~ Break ~

Coding for Fun 特色

- 一招半式闖天下：武林秘笈
  - ★引發興趣
  - ★體驗控制遊戲規則的能力，superpower
  - ★從繪圖編輯器開始，假設學生大多熟悉「小畫家」，降低對「程式設計」的距離感
- 內功：程式積木
  - ★在課程/作業中不斷地累積
  - ★自行探索，漸漸學會如何自學
  - ★小測驗，滿分有音樂，激勵孩子自我成長
  - ★控制積木分散在許多單元

	1	2	3	4	5	6	7	8	9	10
內功	圖形編輯器	動作積木	事件積木 聲音積木	外觀積木	更多積木	畫筆積木 運算積木				
運算觀念		指令 序列	迴圈 事件 平行		remix true/false	件 運算子			變數 列表	
其他元素	角色舞台	與程式互動 平面座標	音樂元素	動畫	傳遞訊息	讀程式	遊戲	製造分身		
解決問題			debug	debug	debug					debug
作業	Water Slide	自我介紹	建立樂團	音樂影片	短篇故事	不設限的專案	遊戲	進階遊戲	自由創作	

Coding for Fun 特色

- 到 Scratch 網站註冊為會員

  ★建立工作室，加入自己的作品

  ★拓展學生**全球視野**

  ★老師建立工作室，集結學生的作品

  ★登記學生的帳號做管理

  ★**師資訓練工作室**

  https://scratch.mit.edu/studios/1959814/

Coding for Fun 特色

- 從實作裡體驗，不解釋「難解釋的名詞」

  ★Scratch 舞台 (平面座標)

  ★蘿蔔蹲 (訊息傳遞)

  ★生日快樂歌 (分辨 播放聲音 喵 / 播放聲音 喵 直到播放完畢)

  ★製作積木 (functions)

  ★Boolean值 (True/False)，使用遊戲來理解

  ★以變數/列表的應用，引導學生放入程式中

Coding for Fun 特色

- 以影片做細節的介紹，學生反覆看影片，隨時暫停、重看，以個別的速度做學習
- 老師把時間留給真的需要協助的學生
- 也可以運用已完成任務的「小老師」，降低老師的負擔

Coding for Fun 特色

- 練習偵錯 (debug)

  ★學生了解程式積木的行為？

  ★理解文字，對症下藥，而不是創造出另一番風景

  ★耐心讀程式，分離出問題點

  ★邏輯是否清楚

Coding for Fun 特色

- 偵錯 (debug) 進行方式
  - ★單人或小組進行，視學生程度而定
  - ★題目有些難，鼓勵學生再想一下、再試試看
  - ★真的不理解或不會，偷看答案無妨
  - ★讓完成任務的同學擔任「小老師」
  - ★解題方式並非唯一，萬一學生繞圈圈解問題，仍應讚賞他/她完成了任務
  - ★讓同學分享不同的解題方式，老師不需評論誰優誰劣

Coding for Fun 特色

- 做作業

  ★學生為自己的創作加入各種元素

  ★願意投入時間創作的學生，會得到較多的進步

- 作業分享

  (1) 自學：

  ★透過 Scratch 工作室，可以看到別人的作品

  (2) 老師指導：

  ★將作品放到最大，做走動分享

  ★厲害的學生有成就感，有了精益求精的動力

  ★不足的學生見賢思齊，模仿也是一種學習

不同的創意會被激發出來

- 具備程式設計能力的學生：

  ★邏輯思考能力強，願意接受挑戰、克服困難

- 說故事、做動畫的學生：

  ★結合文學能力、繪畫能力、或是音樂能力，
  可以將故事做情意上的表達，打動人心

Coding for Fun 未來的支援

- 提供討論網站

  ★需求： (1) 學生不太會描述問題

  (2) 老師無法解答問題

  (3) 師資不足

  ★目標： 讓學生很容易問問題的網站

  ★誰來回答：(1) 學生 (2) 志工

Scratch 的下一步

精進 Scratch

- 精進 Scratch
  - ★學習運算思維，需要一個可以立即上手的程式工具
  - ★可以在閱讀、文學、音樂、美術課融入 Scratch
- 將 Scratch 當作多媒體的創作工具，像是
  - ★我們這一班
  - ★我的好朋友
  - ★我最快樂的事
- 舉辦班際觀摩，激發孩子的創作潛能

閱讀


史達柯頓
美女還是老虎？
繪圖／麥克努雪夫
譯寫／郝廣才
FIAT·JUSTITIA·RUAT·CAELUM·BREVEM·ESSE·DOLEAT

Scratch 運算思維評估系統

- Dr. Scratch
- http://drscratch.programamos.es
- 雖然只是測試版，但可以給老師/學生協助

Dr. Scratch

輸入在 Scratch 網站的作品網頁

或是

上傳 Scratch 作品

特點

- 非常容易使用，適合初學者與專業的程式設計師
- 提供分析報告
- 不斷地提升寫程式的技巧，在家裡也可以
- 藉由圖表和統計，追蹤進步的狀況

分析報告

Dr.Scratch
Analyze your Scratch projects here!

HELP | DR. SCRATCH(BETA VERSION)

Score: 17/21 Tweet

The level of your project is...

MASTER!

You're the master of the universe!!!

Come back to your Scratch project.

Best practice

- 1 duplicated scripts.
- 0 sprite naming.
- 0 dead code.
- 4 sprite attributes.

Project certificate

https://scratch.mit.edu/projects/69387338/

Download

Level up	Level
Flow control	2/3
Data representation	2/3
Abstraction	2/3
User interactivity	2/3
★ Synchronization	3/3
★ Parallelism	3/3
★ Logic	3/3

證書


Dr.Scratch
Analyze your Scratch projects here!
The Dr. Scratch team
has the honour of presenting this
CERTIFICATE
to the project https://scratch.mit.edu/projects/69387338/
because it has obtained a score of
17/21
This project has been analyzing with Dr. Scratch (www.drscratch.org).
Dr. Scratch aims to provide a means of learning and feedback on the quality of the projects in Scratch.
Universidad Rey Juan Carlos
Programamos
Videojuegos y "apps"
FECYT

正式的程式語言

- Python 是首選
- 根據學術刊物 Communications of ACM 2014年7月的調查顯示，Python 超越 Java，成為美國大學課程中入門程式語言的主流：

★Python 相較於 Java 和 C++，Python 語法簡單多了，學習者幾乎可以立刻上手，而且 Python 也適用於商業應用

★10 所美國頂尖的電腦科學系，有 8 所採用 Python 作為入門語言

★39 所頂尖大學中也有 27 所，使用 Python 作為入門語言

★3 個熱門的線上課程提供者：Coursera、edX 和 Udacity，都提供 Python 課程

誰使用 Python？

- Google
- Facebook
- Yahoo
- YouTube
- Dropbox
- NASA

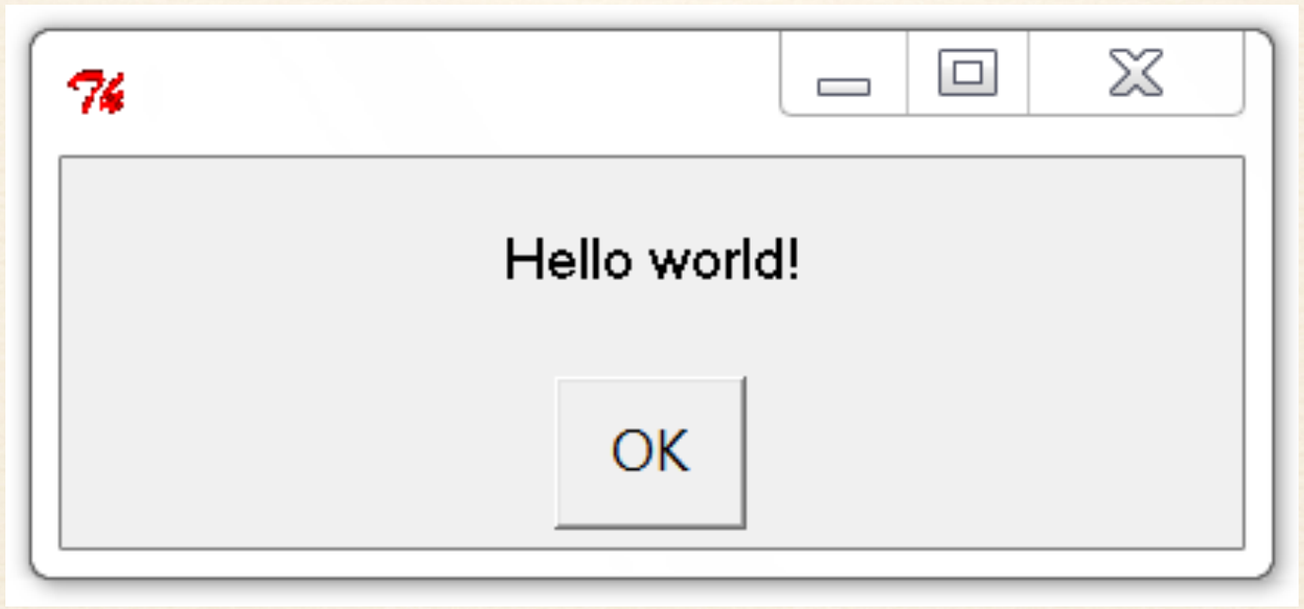
小學生也能學

- `print("Hello world!")`

```
Hello world!
```

- `import easygui`

`easygui.msgbox("Hello world!")`

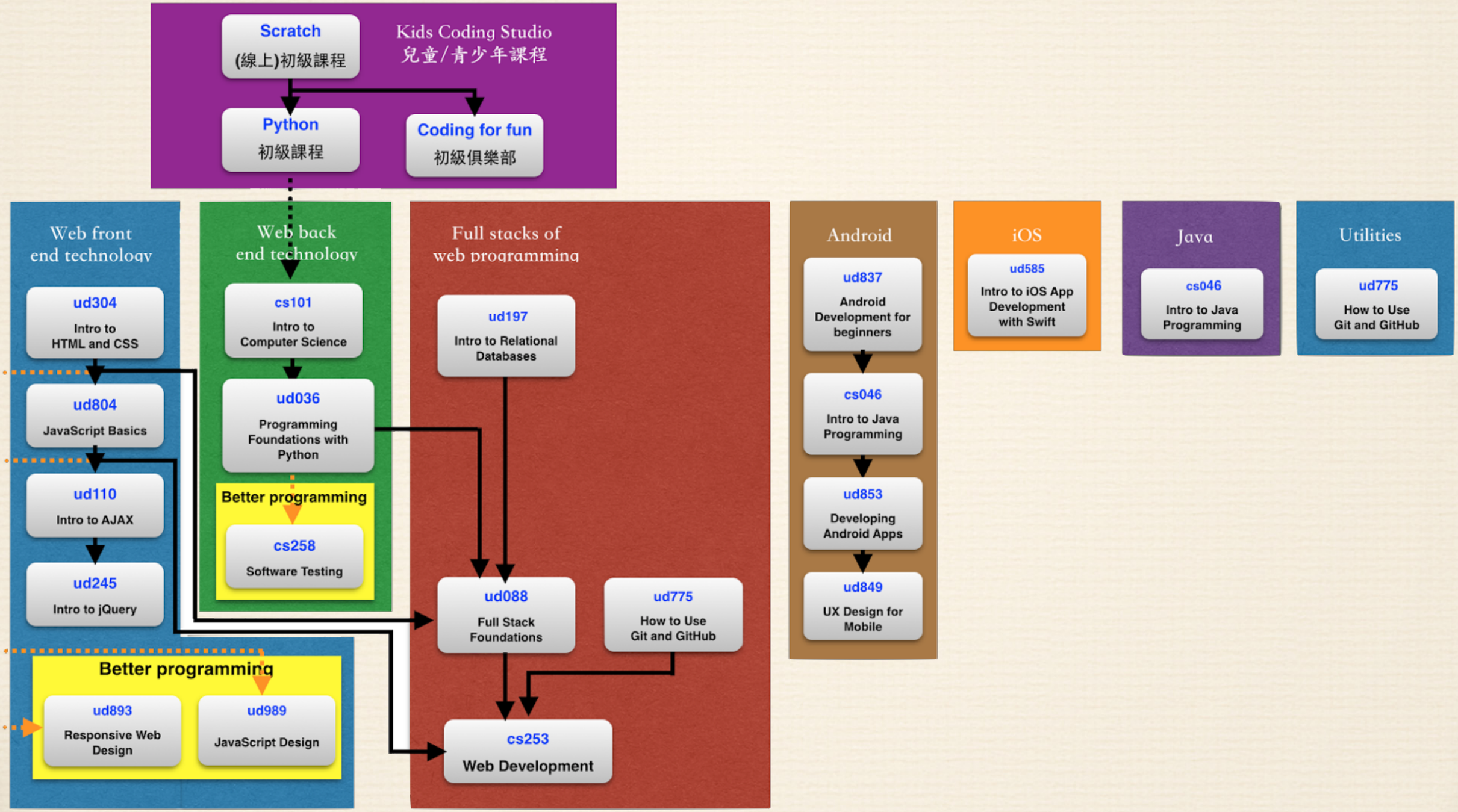


為什麼第一個程式語言不應學 C？

誠懇的建議

- 軟體吃掉全世界，各行各業都需要投資軟體開發。推動程式教育的學習，可以為下一代奠定良好的根基
- 這些產業並不限於科技產業，而那些難以學習的程式語言 (C, C++)，90%以上的機率是碰不到的

程式設計學習地圖


Kids Coding Studio
兒童/青少年課程
Scratch
(線上)初級課程
Python
初級課程
Coding for fun
初級俱樂部
Web front end technology
ud304
Intro to HTML and CSS
ud804
JavaScript Basics
ud110
Intro to AJAX
ud245
Intro to jQuery
Better programming
ud893
Responsive Web Design
ud989
JavaScript Design
Web back end technology
cs101
Intro to Computer Science
ud036
Programming Foundations with Python
Better programming
cs258
Software Testing
Full stacks of web programming
ud197
Intro to Relational Databases
ud088
Full Stack Foundations
ud775
How to Use Git and GitHub
cs253
Web Development
Android
ud837
Android Development for beginners
cs046
Intro to Java Programming
ud853
Developing Android Apps
ud849
UX Design for Mobile
iOS
ud585
Intro to iOS App Development with Swift
Java
cs046
Intro to Java Programming
Utilities
ud775
How to Use Git and GitHub

Q & A

Kids Coding Studio

- 搜尋：兒童程式設計
- 部落格：http://kidscoding.tw
- FB粉絲團：https://www.facebook.com/kidscodingtw/
- 網站：http://coding4fun.tw
- email：coding4fun.tw@gmail.com


Coding for Fun