

# stanCode

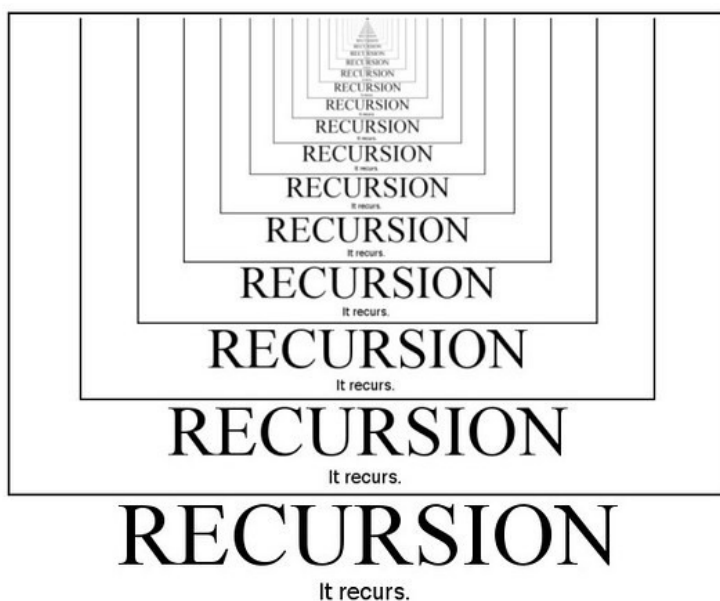
標準程式教育機構

## Assignment 5

This assignment is based on the Assignment 3 and 4 of CS106B at Stanford University

Image Credit: Algodaily.com

點此下載作業檔案



歡迎進到演算法的世界！本份作業改編自 Stanford 進階程式課程 **CS106B**，非常非常具有挑戰性。但若成功征服，您將會有能力，開始準備包含四大科技龍頭 **FAGA**（**Facebook, Apple, Google, Amazon**）在內的各大公司的軟體工程師相關職位的面試。

遞迴 (**recursion**) 是所有軟體工程師面試題最困難的部分。這份作業將提供同學們所有一切解題需要的基本觀念，讓同學在 **LeetCode** 刷題路上可以一路過關斬將！

然而，一定會有不少同學會寫這份作業寫到懷疑人生……

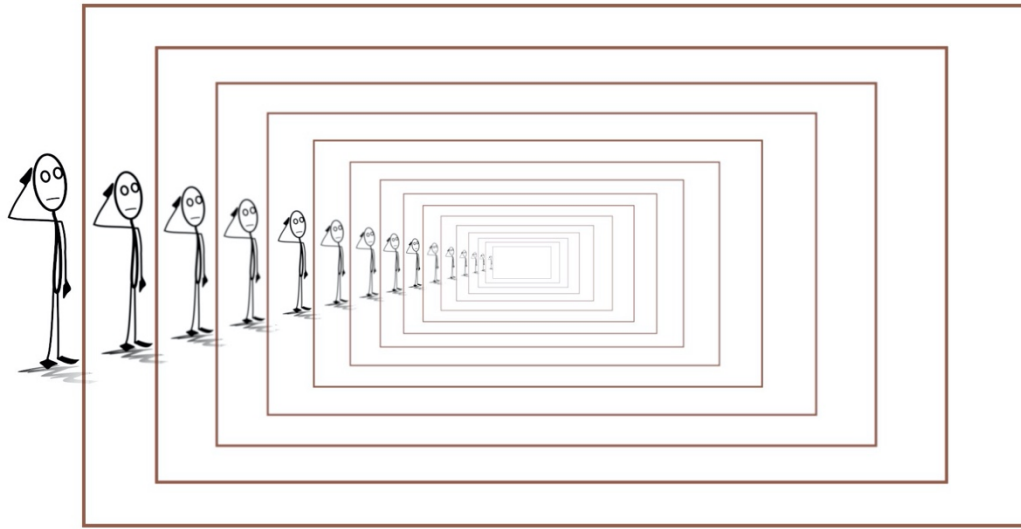


Image Credit: Kulbhushan Singhal

但請同學千萬不要放棄！教學團隊一定會陪你完成不可能的任務，讓你在不論是邏輯思維，還是程式能力，都有爆炸性的突破。

## 本份作業請勿使用 `global variables`

如果過程卡關歡迎各位向助教詢問！也非常鼓勵同學們互相討論作業之概念，但請勿直接把 `code` 分享給同學看，這很可能會剝奪他獨立思考的機會，並讓他的程式碼與你的極度相似，使防抄襲軟體認定有抄襲嫌疑。

### Problem 1 - Tracing Recursive Code

第一題我們先來建立基本遞迴觀念，並了解遞迴程式執行的順序。  
請試著模仿 Jerry 上課時，在白板追蹤遞迴程式的過程。追蹤下圖之遞迴程式碼，並在紙上寫下：

「每個 `stack frame` 執行的順序及其變數存的數值」

```
def recursion():
    num = b(5, 2)
    print(num)

def b(n, k):
    if k == 0 or k == n:
        print('Base Case!')
        return 2
    else:
        return b(n-1, k-1) + b(n-1, k)

if __name__ == '__main__':
    recursion()
```

寫完之後，請在紙上空白處回答下列問題：

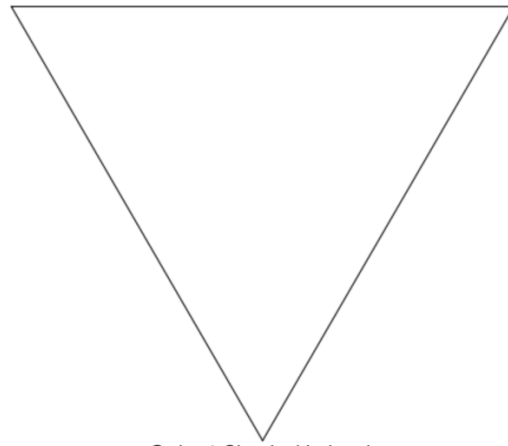
1. 請問執行這段程式碼會在 console 上看到幾遍「**Base Case !!**」
2. 變數 num 數值是多少？

請拍照後將圖檔放到作業資料夾內「Problem 1」的資料夾中。

## Problem 2 - sierpinski.py

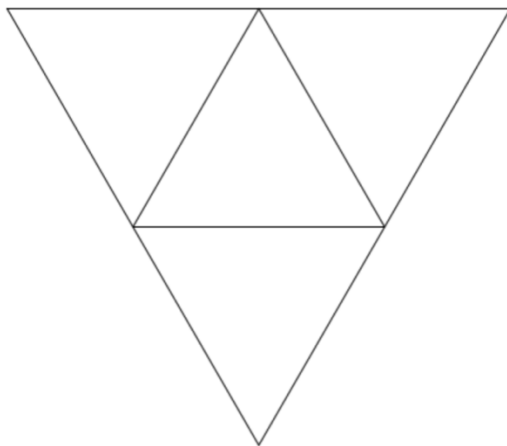
Fractal 是一種用遞迴概念「self similarity（自相似）」所畫出來的幾何形狀。其中一個最有名的例子，就是波蘭數學家 Waclaw Sierpinski 發明的三角形：Sierpinski Triangle。

這種三角形會依照不同的 **order**（**階級**）而有相對應的變化。如果今天我們要畫一個 order 1 的 Sierpinski Triangle，會是一個單純的 **倒立正三角形**：

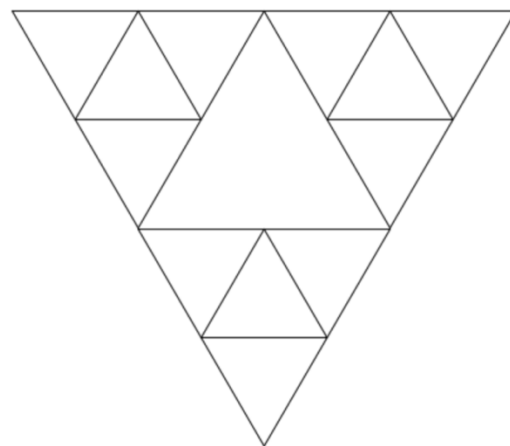


*Order 1 Sierpinski triangle*

而 order 2 & order 3 的 Sierpinski Triangle 會長得像這樣：



*Order-2*



*Order-3*

規則是這樣的：

1. Order K-1 的 Sierpinski Triangle（以下簡稱三角形）邊長為 order K 的三角形邊長的一半。
2. 一個 order K 的三角形會由三個 order K-1 的三角形組成。
3. 這三個 order K-1 的三角形都位在 order K 的三個角落，中間留下一個正三角形的空間。

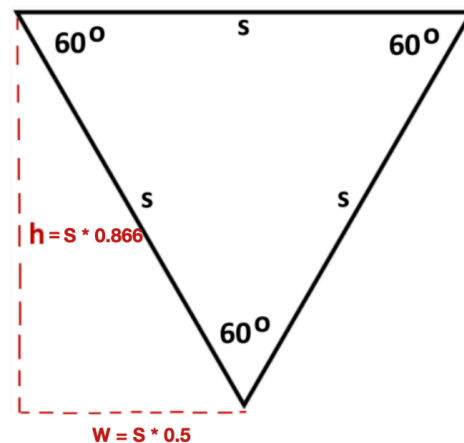
請在 window 上畫出不同 order 的 Sierpinski Triangle，程式當次執行的 order 由常數 ORDER 的值決定。我們已經在 `main()` 呼叫了您即將要撰寫的 `def sierpinski_triangle(order, length, upper_left_x, upper_left_y)`。這邊要注意的是，`upper_left_x` 與 `upper_left_y` 分別代表三角形左上方頂點的 x 座標與 y 座標，且您可以假設 order 為大於零的正整數。

**請特別留意：**

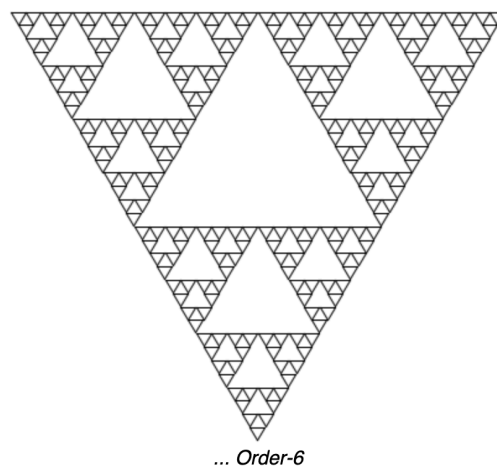
有些同學可能會認為，Sierpinski Triangle 是由「往上指的三角形」和「往下指的三角形」共同組成的。但請避免使用這個想法來編寫您的程式。因為遞迴應該是要尋找 self similarity，並使用可以「重複」的規則來組合成複雜的圖案。過程中可能有許多圖型或線條堆疊在一起，而這是沒有關係的。

另外，每個三角形請使用「三條GLine()」來完成，避免使用GPolygon()。

在製作三角形的過程中，您會發現下圖的三角形數學公式非常有幫助：



若您的程式撰寫正確，在執行 `sierpinski.py` 時會出現與下圖一模一樣的圖案：



### Problem 3 - largest\_digit.py

這題要請同學用遞迴的方式找出「一個任意整數中最大的數字」！

在 `main()` 中，我們丟了五個整數進去 `find_largest_digit()`。若您的程式撰寫正確在執行 `largest_digit.py` 後，會在 console 上看到 5、8、6、1、9 這五個數字。

輸入的整數及其對應的答案，如下圖所示：

```
def main():  
    print(find_largest_digit(12345))      # 5  
    print(find_largest_digit(281))        # 8  
    print(find_largest_digit(6))          # 6  
    print(find_largest_digit(-111))       # 1  
    print(find_largest_digit(-9453))      # 9
```

除了可以印出 5、8、6、1、9 這五個數字以外，還需要符合下列三個條件：

1. 請勿使用任何資料結構 (list、tuple、...) 及文字 (str)
2. 請勿使用迴圈 (for loop & while loop)，只能使用遞迴 (recursion) 函數
3. 程式效率應為  $O(N)$ ， $N$ 代表輸入整數的位數個數

## Problem 4 - anagram.py

Anagram 的意思是「同字母異序字」。

舉例來說，**stop** 這個單字總共有 6 個 **anagrams**：

**stop**、**spot**、**tops**、**opts**、**post**、**pots**。

我們可以發現，每一個 anagram 都是使用了 stop 中的每個字母組成的。不過，並非所有排列組合的結果是一個英文單字，像 opst 或 ptso，就在字典裡找不到。我們要寫出一個程式，可以在接收使用者輸入的單字後，使用 backtracking 找出所有正確的 anagrams！

程式一開始會先印出「Welcome to stanCode "Anagram Generator" (or -1 to quit)」的字樣（如下圖所示）。有上過基礎班的同學，還記得 SC001 Assignment 2 的 weather master 題目嗎？我們當時是如何印出雙引號，並使用「常數（EXIT）」所存的值，來控制離開條件的。現在回想幾個月前，在上 SC001 的自己，希望此刻的你有感受到自己的成長與蛻變！

```
Welcome to stanCode "Anagram Generator" (or -1 to quit)
Find anagrams for: |
```

假設使用者輸入單字 stop 後，按下鍵盤上的 Enter(return)，你的程式將開始嘗試各種單字組合，並判斷組合出來的單字是否存在於字典裡。程式執行畫面如下圖：

```
Welcome to stanCode "Anagram Generator" (or -1 to quit)
Find anagrams for: stop
Searching...
Found: stop
Searching...
Found: spot
Searching...
Found: tops
Searching...
Found: opts
Searching...
Found: post
Searching...
Found: pots
Searching...
6 anagrams: ['stop', 'spot', 'tops', 'opts', 'post', 'pots']
Find anagrams for: -1

Process finished with exit code 0
```

以下七個重點提醒：

1. 請編輯名為 `def read_dictionary()` 的函式，使其可以讀取常數 `FILE` 所存的字典文字檔，並將其中的所有英文單字（記得去除換行字元）儲存在一個 Python list 中。
2. 請編輯名為 `def find_anagrams(s)` 的函式，使用 `backtracking` 的方式搜尋所有 anagrams（`s` 為使用者輸入的單字）。
3. 為了縮短搜尋時間，我們通常會提前將搜尋程序終止 (**early stopping**)！請同學編輯 `def has_prefix(sub_s)`，並 return「字典裡是否存在由 `sub_s` 開頭的字彙？」的 boolean 結果。

要做到這個功能，我們會使用到一個叫做 `startswith()` 的 Python String（文字）的內建 method。這個 method 可以判斷任意一個 String 是否是以另一個 String 開頭，並 return Boolean 結果。例如，當我們呼叫 `'coding'.startswith('co')`，會 return True；而當我們呼叫 `'standard'.startswith('stat')`，會 return False。

設今天我們是要搜尋 "eraser" 的所有 anagrams。若排列組合的過程中出現了 "rr"，`has_prefix("rr")` 會將 "rr" 比對過整本字典後，發現沒有任何單字是以 "rr" 開頭的，告訴我們不需要繼續往下排列組合了。同理，`has_prefix("srr")`, `has_prefix("sre")` 都應該 return False，停止「explore」的過程。如此，我們程式的遞迴搜尋次數就可以大大減少，也大大加快我們程式的搜尋速度！（然而，可能有些同學會發現在使用 `has_prefix()` 後，發現跑的時間跟預期的有些落差。他的力量，會在下一份作業才會更明顯！這邊可以先忽略這個問題，完成下一份作業後最後再回來思考。）

4. 這個程式在處理上會花很多時間，適時在 console 上印出文字就顯得格外重要！（不然使用者會以為電腦當機了）。因此，在您的遞迴開始搜尋前，請先印出「**Searching...**」的字樣，告訴使用者我們已經開始搜尋。除此之外，每當我們找到一個 anagram，請印出「**Found: + (找到的單字)**」，來增加即時感！
5. 最後，請將所有您搜尋到的 anagrams 儲存在一個 Python list，並在結束遞迴搜尋後印出該 list。舉例來說，若 arm 的 anagrams 為 arm、ram、mar，那麼遞迴結束後應該在最後印出：

**3 anagrams: ['arm', 'ram', 'mar']**



6. 展示出來的 anagrams 應該都是獨一無二的，請勿加入重複的單字。同學們可以使用 contains 這個英文單字來測試。若程式正確無誤，應該可以看到如下圖的畫面：

```
Welcome to stanCode "Anagram Generator" (or -1 to quit)
Find anagrams for: contains
Searching...
Found: contains
Searching...
Found: canonist
Searching...
Found: actinons
Searching...
Found: sonantic
Searching...
Found: sanction
Searching...
5 anagrams: ['contains', 'canonist', 'actinons', 'sonantic', 'sanction']
Find anagrams for: -1

Process finished with exit code 0
```

## 7. 演算法之王！

def main() 裡面的 start = time.time() 以及 end = time.time() 可以幫我們記下演算法開始及結束的時間。這邊要請你將：

1. start = time.time() 移動到使用者輸入(input) 完後的位置
2. end = time.time() 移動到印出所有結果後的位置
3. 找出 'contains' 之 anagrams 所花費的時間為競賽標準

```
def main():
    start = time.time()
    #####
    #                #
    #      TODO:      #
    #                #
    #####
    end = time.time()
    print('-----')
    print(f'The speed of your anagram algorithm: {end-start} seconds.')
```

助教會比較全班每一位同學的 anagram 演算法，看看誰的速度最快。stanCode 也將頒發 500 元獎學金給我們班的冠軍，大家加油！

## 評分標準

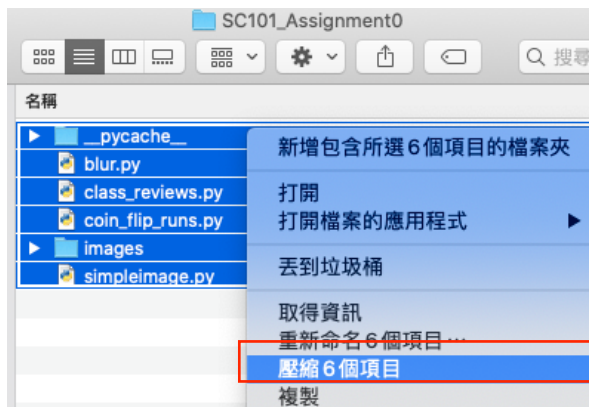
**Functionality** - 程式是否有通過我們的基本要求？程式必須沒有 bug 、能順利完成指定的任務、並確保程式沒有卡在任何的無限環圈（Infinite loop）之中。

**Style** - 好的程式要有好的使用說明，也要讓人一目瞭然，這樣全世界的人才能使用各位的 code 去建造更多更巨大更有趣的程式。因此請大家寫精簡扼要的使用說明、function敘述、單行註解。

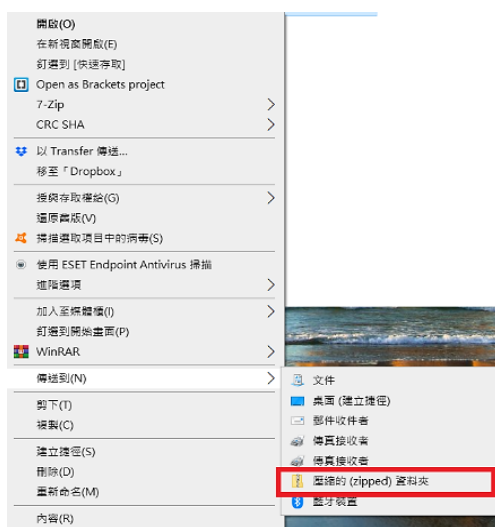
## 作業繳交

1. 以滑鼠「全選」作業資料夾內的所有檔案，並壓縮檔案。請見下圖說明。

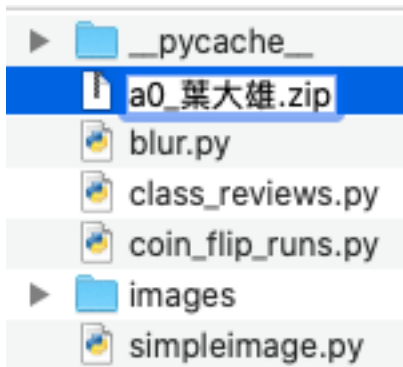
macOS：按右鍵選擇「壓縮n個項目」



Windows：按右鍵選擇「傳送到」→「壓縮的(zipped)資料夾」



2. 將壓縮檔(.zip)重新命名為「a(n)\_中文姓名」。如：  
assignment 0命名為a0\_中文姓名;  
assignment 1命名為a1\_中文姓名;...



3. 將命名好的壓縮檔(.zip)上傳至Google Drive（或任何雲端空間）
  - 1) 搜尋「google drive」
  - 2) 登入後，點選左上角「新增」→「檔案上傳」→選擇作業壓縮檔(.zip)
4. 開啟連結共用設定，並複製下載連結
  - 1) 對檔案按右鍵，點選「共用」
  - 2) 點擊「變更任何知道這個連結的使用者權限」後，權限會變為「可檢視」
  - 3) 點選「複製連結」



5. 將連結上傳至臉書社團的作業貼文提供的「作業提交表單」