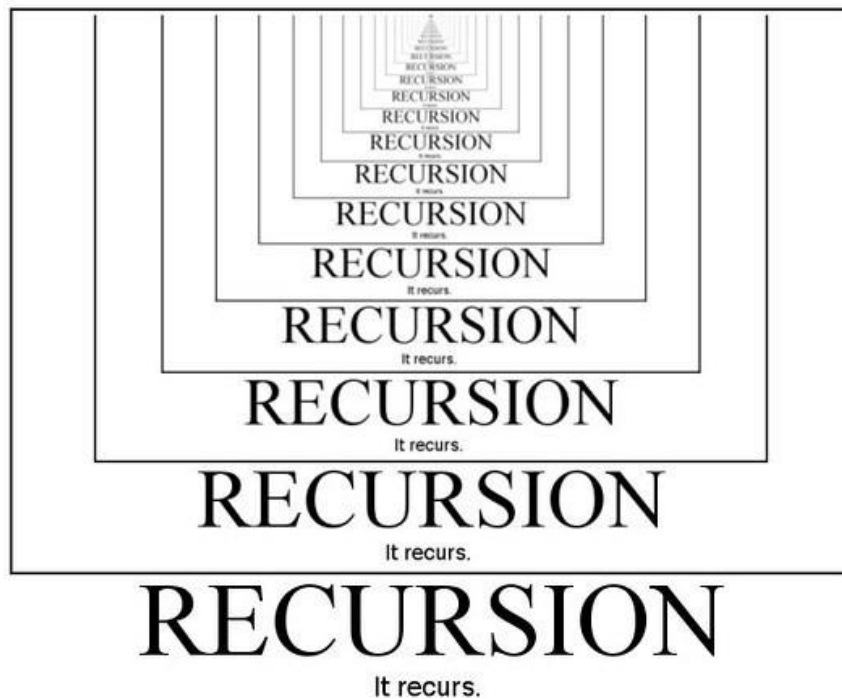


## Assignment 5

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

Image Credit: Algodaily.com



作業檔案下載

歡迎來到 **SC101** 最後一份作業！改編至 **Stanford** 進階程式課程 **CS106B**，本次作業真的非常非常具有挑戰性。然而，若成功征服，您將可以開始準備面試各大公司軟體工程師職位，包含四大科技龍頭 **FAGA – Facebook, Apple, Google, Amazon**。

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

然而，一定會有不少同學會寫這份作業寫到懷疑人生...

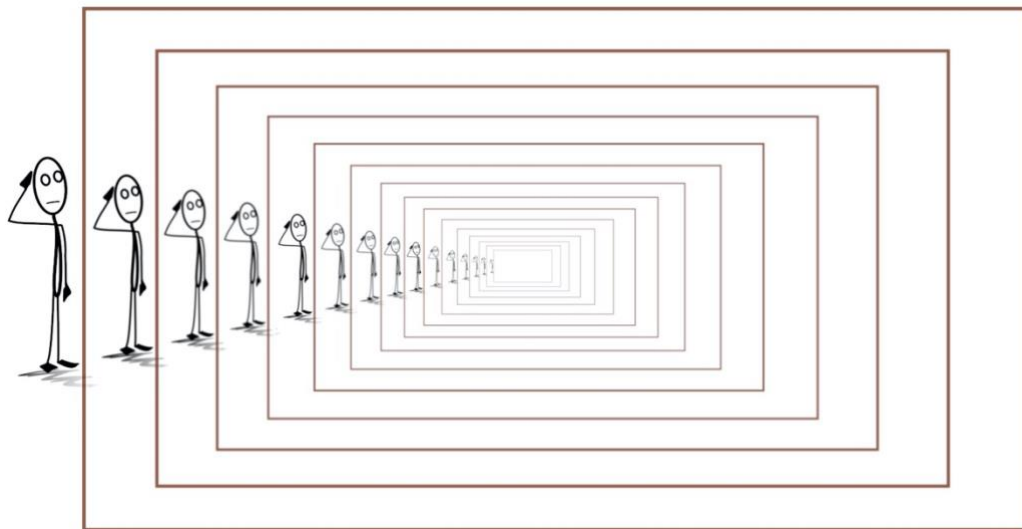


Image Credit: Kulbhushan Singhal

但請同學千萬不要放棄！教學團隊一定會陪各位同學完成不可能的任務，讓各位不管是邏輯思維還是程式能力都有爆炸性的突破

**作業預計時間：20 小時**

如果作業卡關**歡迎各位到社團提問**，也非常鼓勵同學們互相討論作業之**概念**，但請**勿把 code 給任何人看**（也不要將程式碼貼在社團裡）分享妳/你的 **code** 會剝奪其他學生獨立思考的機會，也會讓其他學生的程式碼與你/妳的極度相似，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順序。請模仿 Jerry 上課時在白板追蹤遞迴程式的過程追蹤下圖遞迴程式碼，並在紙上寫下「**每一個 stack frames 的變數數值與其執行順序**」

寫完之後，請拍下紙上的筆跡、將照片放到 Assignment5 作業資料夾 "Problem 1" 中，並回答下列問題：

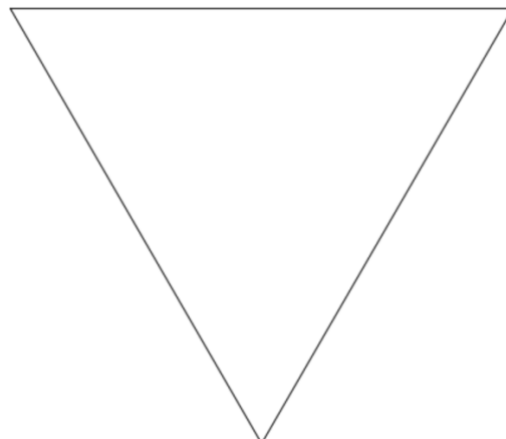
- (1.) 請問執行這段程式碼會在 Console 上看到幾遍 **"Base Case !"**
- (2.) 變數 **num** 數值是多少？

```
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()
```

## Problem 2 – sierpinski.py

Fractal 是用遞迴概念 – self similarity – 畫出來的圖形。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

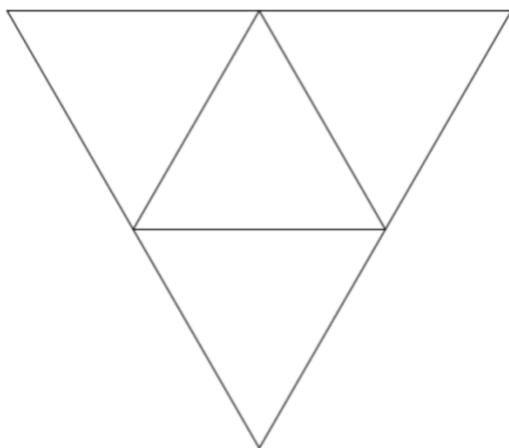
如果今天我們畫 order 1 的 Sierpinski Triangle，您會得到一個正三角形：



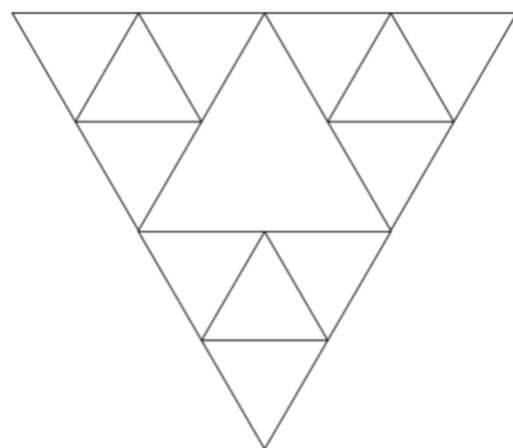
*Order 1 Sierpinski triangle*

為了完成一個 order K 的 Sierpinski Triangle，我們必須先畫三個邊長為原長一半的 order K-1 三角形，且這三個 order K-1 的三角形都在 order K 的三個頂點。

Order 2 與 order 3 的 Sierpinski Triangle 如下圖所示：



*Order-2*

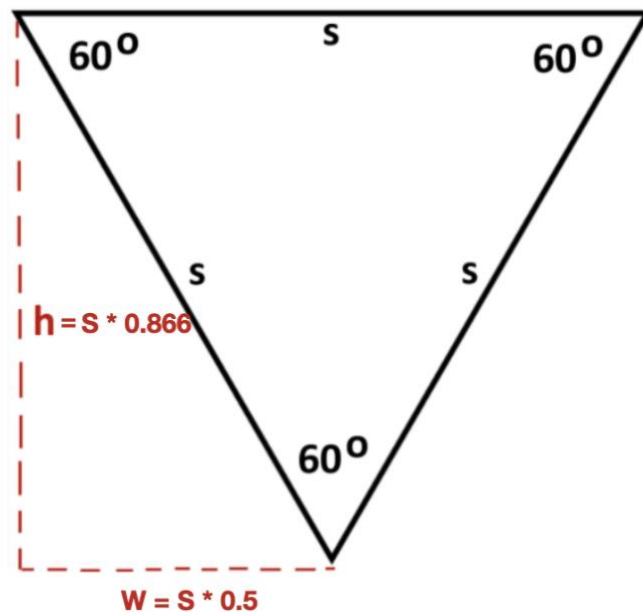


*Order-3*

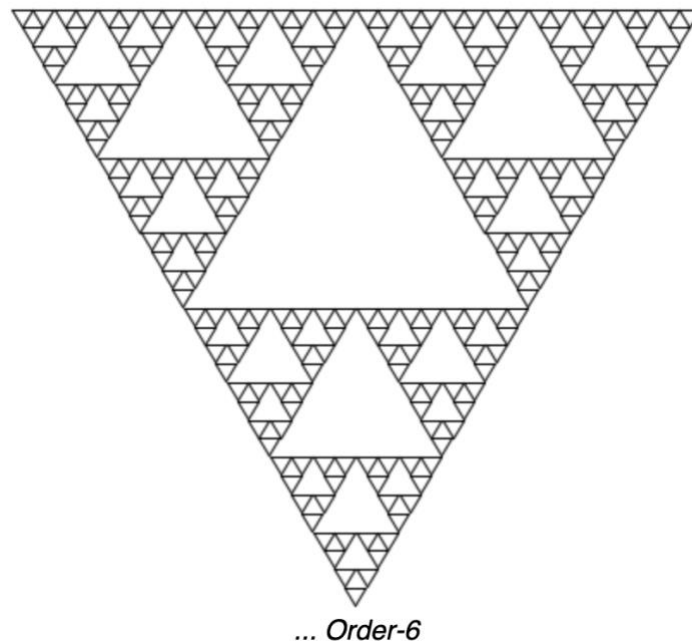
請在 window 上畫出由常數 ORDER 控制的 Sierpinski Triangle。我們已經在 main( ) 呼叫了您即將要撰寫的 **def sierpinski\_triangle(order, length, upper\_left\_x, upper\_left\_y)**. 這邊要注意的是，upper\_left\_x 與 upper\_left\_y 分別代表三角形左上方頂點的 x 座標與 y 座標，且可以假設 order 一定大於零

有些學生會認為：每一個 Sierpinski Triangle 都應該要畫出兩種不同的三角形：「往上指的三角形」還有「往下指的三角形」。但這種想法是錯的，因為遞迴應該要尋找 **self similarity**。也就是說，在 order 2 畫出的所有三角形都應該跟 order 1 畫出的三角形相似

在畫出每一個三角形的過程，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，應該會在執行 `sierpinski.py` 出現與下圖一模一樣的圖樣：



### Problem 3 – largest\_digit.py

第二題要請同學用遞迴的方式找出任意整數裡最大的位數數值！我們在 `main()` 裡面放了五組不同的整數，有正也有負，在您即將撰寫的 `find_largest_digit` 函式。若您的程式正確無誤，應該會在 `Console` 看到 5, 8, 6, 1, 9 這五個數字（換行印）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
```

為了通過我們的測試，您的程式不僅要可以正確印出 5, 8, 6, 1, 9 這五個數字，還要符合下列三個限制：

1. 請勿使用任何資料結構 (list, tuple, ...) 或是文字 (str)
2. 您只能使用 `recursion`，禁止使用迴圈 (for, while, ...)
3. 您的程式效率應該要在  $O(N)$ ， $N$  是輸入整數的位數個數

## Probleme 4 - anagram.py

Anagram 的中文解釋為「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，每一個字彙都只使用了四個字母：'o', 'p', 's', 't'。有趣的是，並非所有 'o', 'p', 's', 't' 的排列組合出來的英文字彙都存在於英文字典裡！舉例來說，字典裡並不存在 opst 這個字、亦不存在 ptso。因此，我們將寫出一個程式，接收使用者輸入的單字，並使用 backtracking 找出所有的 anagrams！

程式一開始會先印出 Welcome to stanCode "Anagram Generator" (or -1 to quit) 的字樣。與 SC001 第二份作業的 weather 一樣，我們需要印出雙引號，也要使用一個常數 (-1) 控制離開的條件（如下圖所示）。現在回想兩個月前的 SC001，希望同學此刻有明顯地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** 並按下鍵盤上的 Enter，您的程式將開始搜尋定義在常數 FILE 裡的文字檔（就是一部字典），查看 'o', 'p', 's', 't' 這四個字母排列組合出來的文字是否為存在於字典裡。詳細程式執行畫面如下圖所示：

```
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
```

以下六個重點提醒：

1. 請編輯名為 **def read\_dictionary()** 的函示，讀取整本字典並將讀到的所有字彙（記得去除換行字元）儲存在 Python list（可以使用 Global Variable）
2. 請編輯名為 **def find\_anagrams(s)** 的函示，讓 **s** 接收使用者輸入的單字，並使用 **backtracking** 搜尋所有的 **anagrams**
3. 為了縮短搜尋時間，我們通常會提前將搜尋程序終止！為了做到這點，請同學編輯一個名為 **def has\_prefix(sub\_s)** 的函示，告訴使用者「字典裡是否存在由 **sub\_s** 開頭的字彙？」

Python 文字都有內建一個名叫 **startswith(sub\_s)** 的 method，可以檢查任意文字是否以 **sub\_s** 開頭。若我們使用 **'coding'.startswith('co')** 會得到 **True**；而若我們使用 **'standard'.startswith('stam')** 就會得到 **False**

舉例來說，假設今天我們要搜尋橡皮擦“eraser”的所有 **anagrams**。若排列組合的過程中若碰到了“rr”，程式就不應該繼續找下去了，因為 **has\_prefix("rr")** 搜尋整本字典後發現沒有任何字彙是以“rr”開頭的。因此 **has\_prefix("rr")** 會 **return False**，告訴程式我們沒有繼續排列組合下去的必要。同理，**has\_prefix("srr")**, **has\_prefix("sre")** 都應該 **return False**。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！

4. 這個程式在處理上會花很多時間。這時，適當在 **Console** 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:** 的字樣，並將找到的字彙緊接在後印出
5. 最後，請將所有您搜尋到的 **anagrams** 儲存在一個 **list**，並在結束遞迴搜尋後印出該 **list** 的內容。舉例來說。若搜尋手臂的英文單字 **'arm'** 搜尋到的結果為 **['arm', 'ram', 'mar']**，遞迴結束前的搜尋結果應該如下：  
**3 anagrams: ['arm', 'ram', 'mar']**
6. 請勿重複加入字彙。答案展示出來的單字應該都是獨一無二的。同學們可以使用 **contains** 這個英文單字來測試。若程式這卻無誤，應該可以看到下圖的每一行文字：



```
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
```

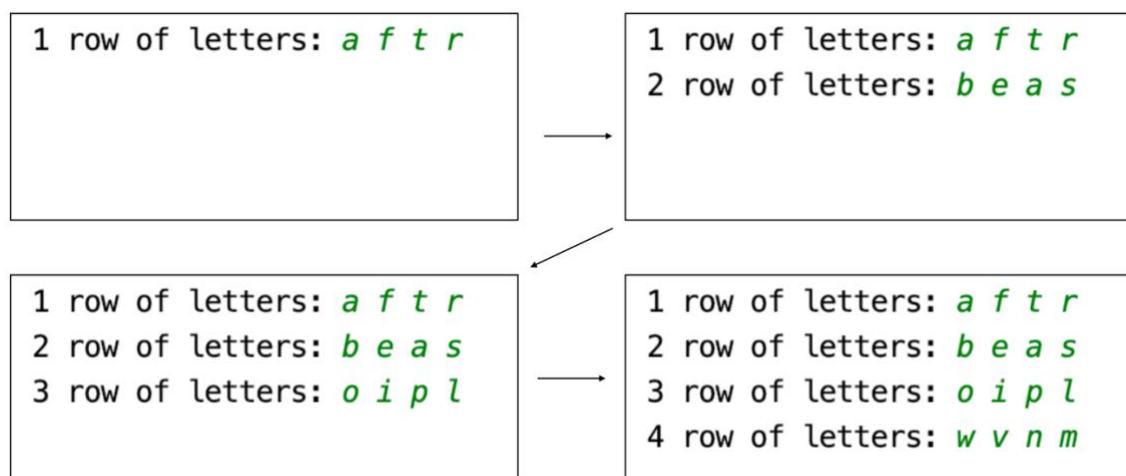
## Problem 5 - boggle.py



Boggle 是美國家喻戶曉的桌遊！遊戲一開始會先排出一個 4 x 4 的方形字母拼盤，接著玩家開始串連在字母盤上相連的字母，去找出存在於這個 4 x 4 的方形字母拼盤的所有英文單字。沒錯，SC101 作業的最後一題您將使用 **backtracking** 完成這個遊戲搜尋的程式，來為課程劃下一個完美的句點

在串連字母的過程，我們只能串連字母塊的「鄰居」－ 上、下、左、右、左上、左下、右上、右下 － 且一個字母塊只能經過一次

遊戲一開始玩家會被要求輸入 4 排以空白隔開的 4 個英文單字（如下圖所示）



若使用者的輸入不符合規定，程式就會**終止執行**。換句話說，我們強迫使用者輸入 16 個英文字母且每一排的字母間一定要用空白隔開

```
1 row of letters: a b v c
2 row of letters: aa n u i
Illegal input
```

假設使用者正確輸入了下圖所示的 16 個字母：

```
1 row of letters: f y c l
2 row of letters: i o m g
3 row of letters: o r i l
4 row of letters: h j h u
```

這時您的程式就會開始搜尋這個 4x4 字母盤上**所有**可以被串連起來的英文單字

若您的程式撰寫正確應該能在 **Console** 完美重現下圖中的每一行文字：

```
1 row of letters: f y c l
2 row of letters: i o m g
3 row of letters: o r i l
4 row of letters: h j h u
Found "firm"
Found "form"
Found "foil"
Found "coif"
Found "coir"
Found "corm"
Found "coil"
Found "moor"
Found "moil"
Found "miri"
Found "giro"
Found "glim"
Found "roil"
Found "roof"
Found "room"
Found "roomy"
Found "rimy"
Found "iglu"
Found "limy"
Found "limo"
Found "hoof"
There are 21 words in total.
```

以下八個重點提醒：

1. 起始字母可以任意選擇。然而，一旦選擇了，請遵守上述串連法則
2. 程式應該為 **Case insensitive**
3. 程式只搜尋長度大於等於 **4** 的英文單字
4. 編輯名為 **def read\_dictionary()** 的函式，功能與 **Problem 4** 一樣
5. 編輯名為 **def has\_prefix(sub\_s)** 的函式，功能與 **Problem 4** 一樣
6. 當您撰寫的遞迴程式找到一個單字時，請印出 **Found** 並緊接著印出“找到的文字”
7. 這題最困難的地方莫過於上圖中“**room**”與“**roomy**”：該如何在找到一個解答後，繼續要求我們的遞迴程式往下搜尋更長的單字呢？
8. 最後，請印出在該 **4 x 4** 字母盤上找到了幾個單字的總數

## Assignment Overflow – DFS and BFS

**Backtracking** 扮演著許多演算法的根基。其中，各大公司最注重的 Depth First Search (DFS) 以及 Breadth First Search (BFS) 更是常見的面試題目

### Depth First Search – DFS

**DFS** 是一個以「深度」為主軸的搜尋演算法。換句話說，在探索每一個資料點的過程中，DFS 都會以最遠的幾個資料點開始搜尋。如果遇到死路，再使用我們上課提過的 **Backtracking** 去探索其他路徑。下方影片提供一個非常好的範例：

<https://www.youtube.com/watch?v=NUgMa5coCoE>

如影片所示，**黃色圓圈**（最新探索選擇）的探索會越來越深；**紅色圓圈**（過去的選擇）會因此而越變越長。然而，當紅色路徑到死胡同、沒路往下走的時候，同學可以發現 **藍色圓圈** 就會執行 **backtracking** – 退到一個有其他選擇的叉路，再繼續往下繼續探索

### Breadth First Search – BFS

**BFS** 是一個以「廣度」為主軸的搜尋演算法。換句話說，在探索每一個資料點的過程中，BFS 會優先以鄰居的資料點開始搜尋。下方影片提供一個非常好的範例：<https://www.youtube.com/watch?v=x-VTfcmrLEQ>

為了讓 **BFS** 可以記得所有鄰居，演算法都會搭配一個名為 **Queue** 的資料結構。若同學還記得 **Week 3** 的上課內容，Jerry 曾經提過 **Queue** 是一個 **First-In-First-Out (FIFO)** 的資料結構。雖然 **BFS** 跟我們上課所學比較不一樣，但同學們只要稍微上網看一些資料一定就可以了解細節了！

若這份作業做完還意猶未盡的同學，歡迎觀賞下方連結中，一位韓國人清楚講解 **BFS** 概念的影片：<https://www.youtube.com/watch?v=QkEOGoUar3g&t=167s>

## 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迴圈（**infinite loop**）之中。

**Style** - 如同我們在課堂上所說，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 **main()**程式概要、**function comments** 和單行註解。

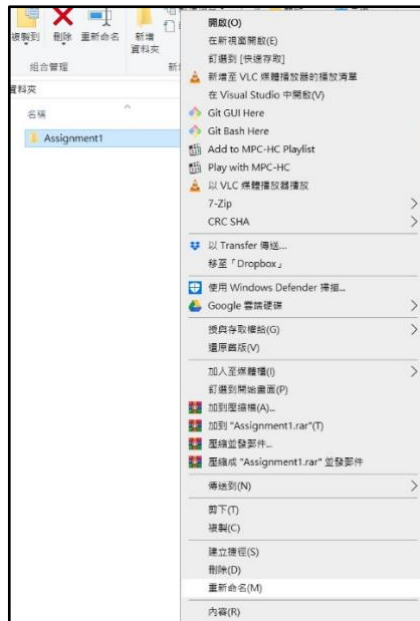
## 作業繳交

恭喜您完成 **Assignment 5**！大家應該要對自己的成就感到驕傲，因為這份作業跟史丹佛大學的學生作業非常相似，代表您跟世界各國的菁英一樣厲害了。

請同學於**作業繳交期限前**，依照下圖將您完成的作業的**下載連結**上傳至社團提供的**作業繳交表單**。



找到作業資料夾，按右鍵



點選重新命名

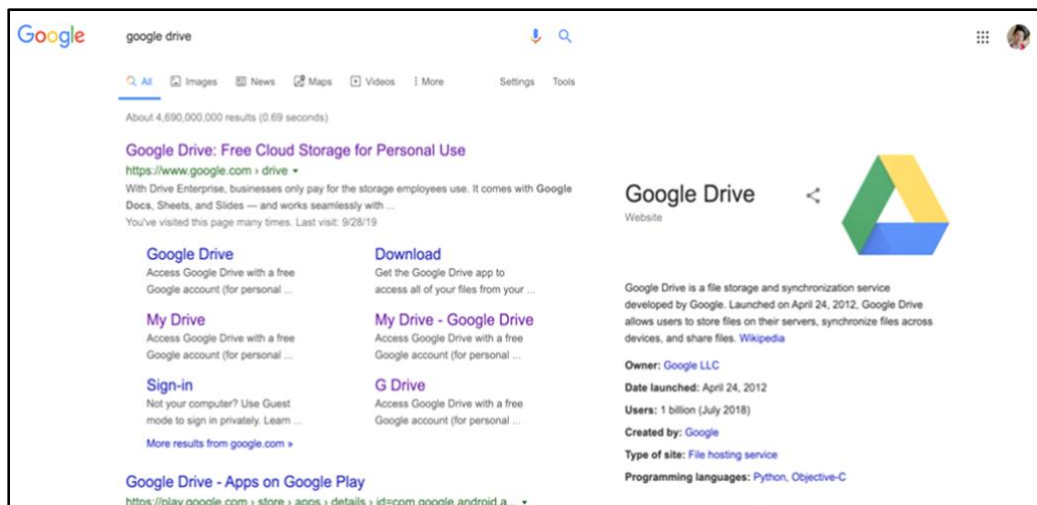
名稱	修改日期	類型	大小
Assignment7_羅志祥	2020/7/1 下午 02:28	檔案資料夾	

請命名成「**Assignment5\_中文姓名**」的格式

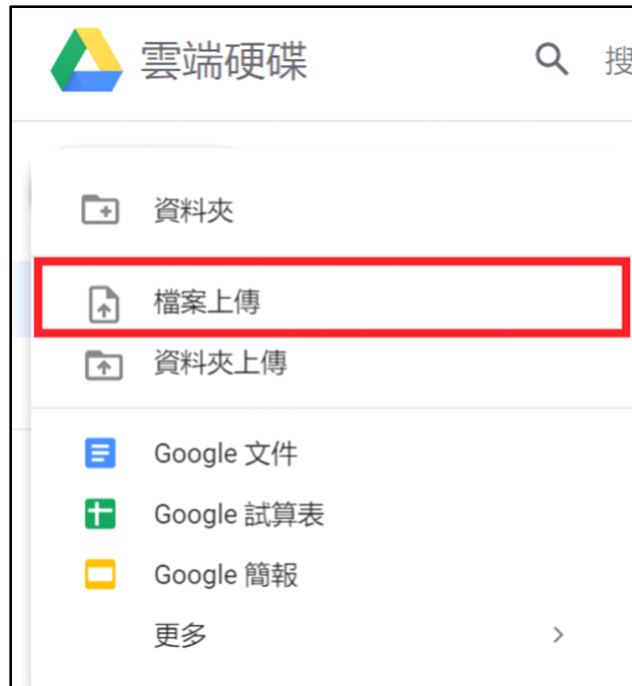


**Windows 請點選「傳送到」->「壓縮的(zipped)資料夾」**

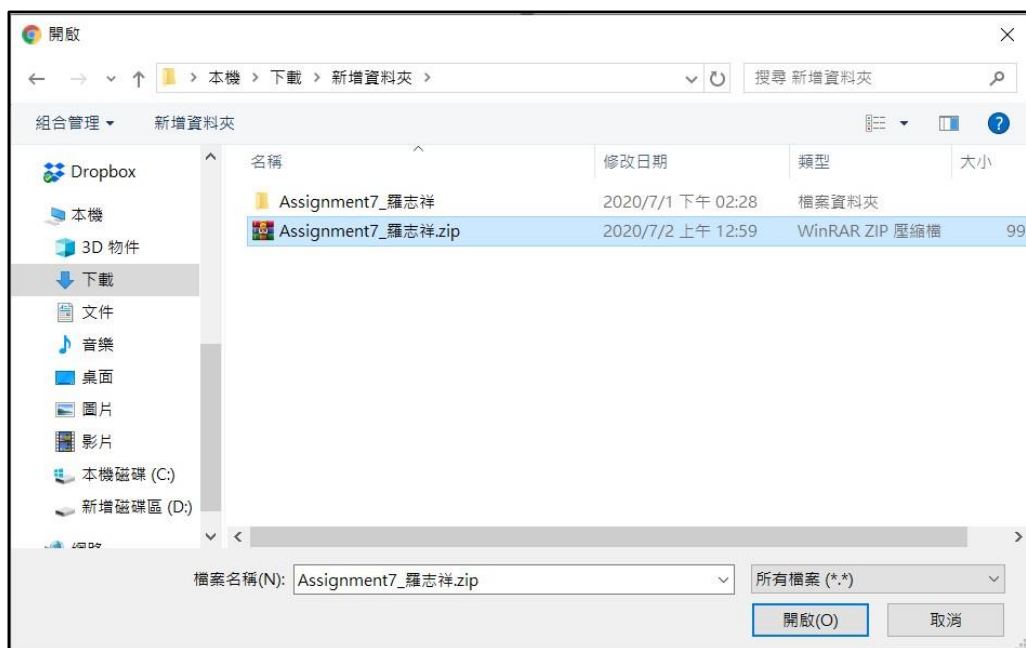
**Mac 請點選 Compress "Assignment5\_中文姓名"**



**上網搜尋 Google Drive**

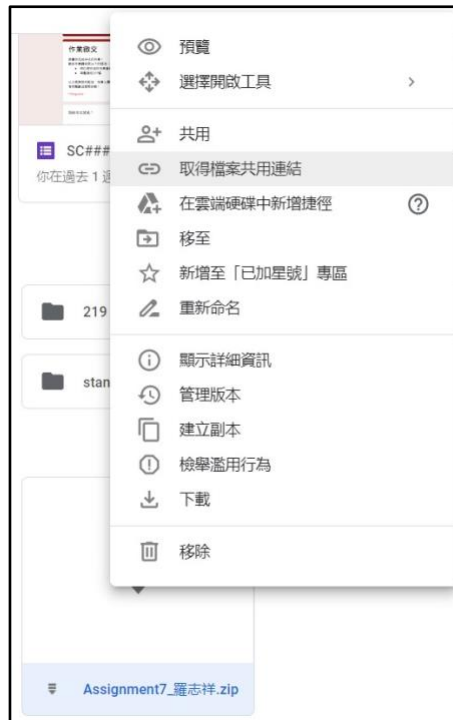


點選 檔案上傳(File Upload)



找到剛剛壓縮的 Assignment5 檔案後，點選 開啟(Open)





上傳完成後，對檔案按右鍵，選擇 取得檔案共用連結(Get Shareable Link)



將「限制(Restricted)」改成「知道連結的使用者(Anyone With the Link)」

再按下 複製連結

點開社團提供的作業繳交表單：



stanCode

### 作業繳交

恭喜您完成本次的作業！

繳交作業請依照以下的規定：

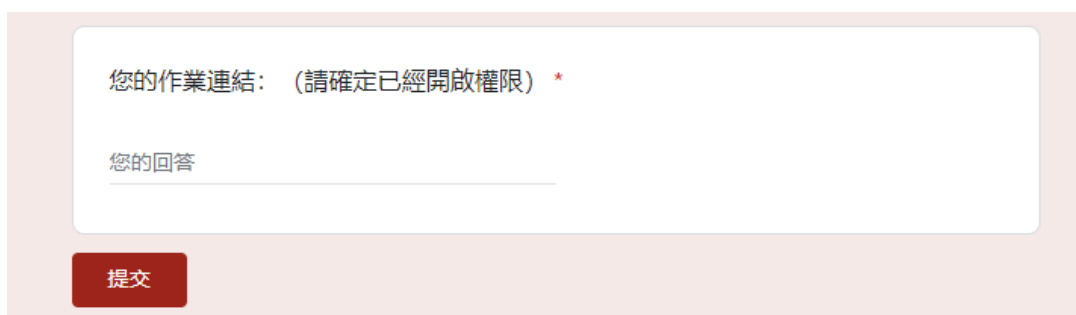
- 將已經完成的作業資料夾命名成以下格式「Assignment#\_Name」
- 再壓縮成ZIP檔

作業以最後繳交的版本計算成績。  
若有提交新版本，請通知各自的助教，以免遺漏。  
以上，謝謝配合～

\*必填

您的中文姓名： \*

您的回答



您的作業連結：（請確定已經開啟權限） \*

您的回答

提交

填入相關資訊，並在最後附上剛剛複製的連結

確認無誤後按下提交

stanCode - 標準程式教育機構



stanCode

Should you have any idea or questions, please feel free to contact.