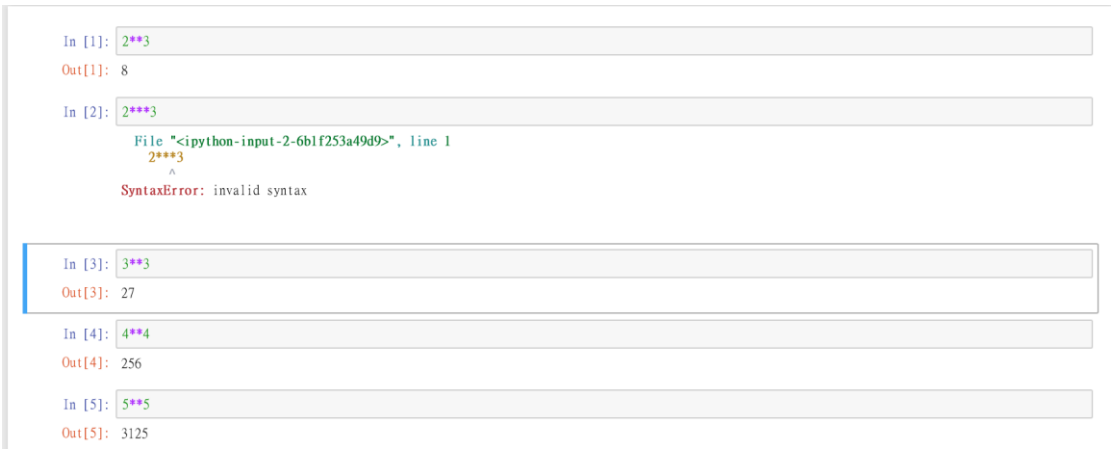
心得:
轉學以來其實一開始都懂程式語言，不曾那麼深刻接觸程式過，以前對於程式的理解就是用電腦打出程式，對於什麼語法其實都不清楚，邏輯方面的我也不在行，所以一開始接觸蔡老師的課時，相當害怕，當時老師推薦了一本演算法圖鑑的書我就立刻買來研究，有好幾次作業我都會先搞懂書中的說明再自己研究如何畫出適合的流程圖。但上了好幾次課程及作業繳交，還是覺得抓不到方向，尤其在研究作業的過程中，我總是要先找很多資料跟影片，花了很長的時間才能理解程式邏輯，再經過參考資料我能漸漸明白程式碼的語意，但是理解後又要自己原創程式碼其實真的不簡單，可能原本懂得那個程式碼意思，但在原創程式碼的時候卻又突然卡關很久，要經過一再尋找資料及觀看影片，才能完成作業，看著程式碼出現錯誤無力感不斷提升，但也只能告訴自己這是訓練我的思考能力，訓練自己如何修改參數及程式碼，讓程式碼成功，因此反覆修改再重新 return 一次，最後程式碼終於成功了，但因為必須加入助教的測資測試自己的程式碼，所以很常繳交完作業後卻沒有如期拿到分數，這讓我挫敗不已。不過這也讓我學習到很多以前沒有體會到的歷程，學習完一學期的課之後，我蠻感謝老師堅持讓我們原創程式碼的作風，讓我們深刻體驗工程師一樣的辛苦，對於程式碼我也從一開始連要怎麼重複印出數字都不太會，到現在能打出不同的 code，雖然還是需要加強很多很多部分，但跟以前的我比起來，我覺得自己成長不少，所以很謝謝老師及助教。希望在畢業以前我可以持續運用各種方式繼續增強實力，在這堂課我發現有幾位老師會提到的同學，他們的能力真的很強，或許都是因為這堂課所以激發出來，希望有天我也能像他們一樣。

(以下是我擷取一部分自己的學習歷程)



In [7]:
```
5%3
```
Out[7]: 2

In [8]:
```
5%5
```
Out[8]: 0

In [9]:
```
55%11
```
Out[9]: 0

In [10]:
```
a=[]
b=[1,2,3]
c=[1,"string",[1,2,3]]
```

In [11]:
```
len(c)
```
Out[11]: 3

In [12]:
```
len(b)
```
Out[12]: 3

In [14]:
```
sum(b)
```
Out[14]: 6

In [15]:
```
x =[0,1,2,3,4]
```

In [16]:
```
x[1]
```

In [17]:
```
x[4]
```
Out[17]: 4

In [18]:
```
x[-1]
```
Out[18]: 4

In [19]:
```
x[-2]
```
Out[19]: 3

In [20]:
```
x[-3]
```
Out[20]: 2

In [21]:
```
x[-1]
```
Out[21]: 4

In [22]:
```
x[3]
```
Out[22]: 3

In [23]:
```
x[0]=-2
```

In [24]:
```
x
```
Out[24]: [-2, 1, 2, 3, 4]

In [25]:
```
x[:2]
```
Out[25]: [-2, 1]

In [26]:
```
x[0]=-4
```

In [27]:
```
x
```
Out[27]: [-4, 1, 2, 3, 4]

In [28]:
```
x[:3]
```
Out[28]: [-4, 1, 2]

In [31]:
```
x[4:]
```
Out[31]: [4]

In [32]:
```
x[-3:]
```
Out[32]: [2, 3, 4]

In [37]:
```
x=[1,2,3]
x.extend([4,5,6])
print(x)
```
[1, 2, 3, 4, 5, 6]

In [38]:
```
x
```
Out[38]: [1, 2, 3, 4, 5, 6]

In [39]:
```
x
```
Out[39]: [1, 2, 3, 4, 5, 6]

In [43]:
```
x=[1,2,3]
y=x+[8,7,6]
print(y)
```

```
[1, 2, 3, 8, 7, 6]
```

In [41]:
```
y
```

Out[41]:
```
[1, 2, 3, 8, 7, 6]
```

In [1]:
```
def quick_sort(list[],m,n):
{
 if(m < n)then
      {
          i = m,j = n+1,k = list[m]:
          Repeat
          {
              repeat
                i = i+1:
              until list[i] >= k:
              repeat
                j = j-1:
              until list[j] <= k:
              if(i<j)then Swap(list,[i],list[j]):
          }until(i>= j):
          Swap(list[m],list[j]):
          quick_sort(list,m,j-1):
          quick_sort(list,j+1,n):
      }:
}
```

In [3]:
```
def quick_sort(array):
    Smaller=[]
    Equal=[]
    Bigger=[]      #定義大中小數值
    i = smaller - 1
    for j in range(Smaller, Bigger):
        if Equal[j] <= arr[Bigger]:
            i = i + 1
            Equal[i], Equal[j] = arr[j], arr[i]
    arr[i + 1], arr[hight] = arr[hight], arr[i + 1]
    return i

def quick_sort(l, low, hight):
    if low < hight:
        key_Index = partition(l, low, hight)
        quick_sort(l, low, key_Index)
        quick_sort(l, key_Index + 1, hight)
    else:
        return

l = [1,3,5,7,9,2,4,6,8,10]
quick_sort(l, 0, len(l) - 1)
print(l)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-e6dc0d5b6ee8> in <module>
     20
     21 l = [1,3,5,7,9,2,4,6,8,10]
---> 22 quick_sort(l, 0, len(l) - 1)
     23 print(l)
```

In [4]:
```
def partition(arr, low, hight):   #
    i = low - 1
    for j in range(low, hight):
        if arr[j] <= arr[hight]:
            i = i + 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[hight] = arr[hight], arr[i + 1]
    return i

def quick_sort(l, low, hight):
    if low < hight:
        key_Index = partition(l, low, hight)
        quick_sort(l, low, key_Index)
        quick_sort(l, key_Index + 1, hight)
    else:
        return

l = [1,3,5,7,9,2,4,6,8,10]
quick_sort(l, 0, len(l) - 1)
print(l)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [5]:
```
list = [1,3,5,7,9,2,4,6,8,10]   #設定list為括號內數字
def quicksort(list, left, right):    #定義快速排序法
    if left >= right :
        return

    i = left                        #設定為左右
    j = right
    key = list[left]

    while i != j:
        while list[j] > key and i < j:

            j -= 1

        while list[i] <= key and i < j:

            i += 1

        if i < j:
            list[i], list[j] = list[j], list[i]




    list[left],list[i] = list[i],list[left]
    quicksort(list, left, i-1)
    quicksort(list, j+1, right)

print(list)
quicksort(list, 0, len(list)-1)
print(list)
```

```
[1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [6]:
```
from collections import defaultdict
class Graph:

    def __init__(self):
        self.graph = defaultdict(list)

    def add_new_edge(self,u,v):
        self.graph[u].append(v)

    def BFS(self, stack):
        visited_before = [False] * (len(self.graph))
        queue = []
        queue.append(stack)
        visited_before[stack] = True

        while queue:
            stack = queue.pop(0)
            print(stack, end = " ")
            for index in self.graph[stack]:
                if visited_before[index] == False:
                    queue.append(index)

                    visited_before[index] = True
```

In [24]:
```
class TreeNode:

    def __init__(me,x):
        me.val = x
        me.left = None
        me.right = None

    def insert(root,node):
        if root == None:
            root = node
        else:
            if root.val < node.val:
                if root.right == None:
                    root.right = node
                else:
                    insert(root.right, node)
            else:
                if root.left == None:
                    root.left = node
                else:insert(root.left, node)
def PrintTreenode(me):
        if me.left:
            me.left.PrintTreenode()
        print(me,x),
        if me.right:
            me.right.PrintTreenode()
```

In [7]:
```
tree = ['red-1', 'gray-2', 'pink-3', 'green-4', 'yellow-5', 'purple-6']
edgeList = [(0,0), (0,1), (1,0) , (1,3) , (2,0) , (2,4) , (2,5) , (3,1), (3,5),(4,2) , (4,5), (5,2)]
graphs = [tree, edgeList]

def dfs(graph, s):
    tree, edgeList = graph
    visited = []
    stack = [s]
    adjacencyList = [[] for vertice in tree]


        while stack:
        current = stack.pop()
        for neighbor in adjacencyList[current]:
            if not neighbor in visited:
                stack.append(neighbor)
        visited.append(current)
    return tree

print(dfs(graphs, 0))
```

```
  File "<ipython-input-7-8a50c23bfebe>", line 12
    while stack:
    ^
IndentationError: unexpected indent
```

In [8]:
```
#dfs
verticeList = ['0', '1', '2', '3', '4', '5', '6']
edgeList = [(0,1), (0,2), (1,0) , (1,3) , (2,0) , (2,4) , (2,5) , (3,1), (4,2) , (4,6), (5,2), (6,4)]
graphs = (verticeList, edgeList)

def DFS(graph, start):
    verticeList, edgeList = graph
    visitedVertex = []
    stack = [start]
    adjacencyList = [[] for vertex in vertexList]

    for edge in edgeList:
        adjacencyList[edge[0]].append(edge[1])

    while stack:
        current = stack.pop()
        for neighbor in adjacencyList[current]:
            if not neighbor in visitedVertex:
                stack.append(neighbor)
        visitedVertex.append(current)
    return visitedVertex

print(dfs)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-c6614813d411> in <module>
     21     return visitedVertex
     22
---> 23 print(dfs)

NameError: name 'dfs' is not defined
```

In [9]:
```
vertexList = ['red-1', 'orange-2', 'lime-3', 'green-4', 'yellow-5', 'blue-6']
edgeList = [(0,0), (0,1), (1,0) , (1,3) , (2,0) , (2,4) , (2,5) , (3,1), (3,5),(4,2) , (4,5), (5,2)]
graphs = [vertexList, edgeList]

def dfs(graph, s):
    vertexList, edgeList = graph
    visitedVertex = []
    stack = [s]
    adjacencyList = [[] for vertex in vertexList]

    for edge in edgeList:
        adjacencyList[edge[0]].append(edge[1])

    while stack:
        current = stack.pop()
        for neighbor in adjacencyList[current]:
            if not neighbor in visitedVertex:
                stack.append(neighbor)
        visitedVertex.append(current)
    return visitedVertex

print(dfs(graphs, 0))
```

```
[0, 1, 3, 5, 2, 4, 0]
```

In [10]:
```
tree = ['red-1', 'gray-2', 'pink-3', 'green-4', 'yellow-5', 'purple-6']
edgeList = [(0,0), (0,1), (1,0) , (1,3) , (2,0) , (2,4) , (2,5) , (3,1), (3,5),(4,2) , (4,5), (5,2)]
graphs = [tree, edgeList]
```

In [11]:
```
tree = ['red-1', 'gray-2', 'pink-3', 'green-4', 'yellow-5', 'purple-6']
edgeList = [(0,0), (0,1), (1,0) , (1,3) , (2,0) , (2,4) , (2,5) , (3,1), (3,5),(4,2) , (4,5), (5,2)]
graphs = [tree, edgeList]

def dfs(graph, s):
    tree, edgeList = graph
    visited = []
    stack = [s]
    adjacencyList = [[] for vertex in tree]


    while stack:
        current = stack.pop()
        for neighbor in adjacencyList[current]:
            if not neighbor in visited:
                stack.append(neighbor)
        visited.append(current)
    return tree

print(dfs(graphs, 0))
```

```
['red-1', 'gray-2', 'pink-3', 'green-4', 'yellow-5', 'purple-6']
```

In [12]:
```
tree = ['red-1', 'gray-2', 'pink-3', 'green-4', 'yellow-5', 'purple-6']
edgeList = [(0,0), (0,1), (1,0) , (1,3) , (2,0) , (2,4) , (2,5) , (3,1), (3,5),(4,2) , (4,5), (5,2)]
graphs = [tree, edgeList]

def dfs(graph, s):
    tree, edgeList = graph
    visited = []
    stack = [s]
    adjacencyList = [[] for verteice in tree]

    for edge in edgeList:
        adjacencyList[edge[0]].append(edge[1])

    while stack:
        current = stack.pop()
        for neighbor in adjacencyList[current]:
            if not neighbor in visited:
                stack.append(neighbor)
        visited.append(current)
    return tree

print(dfs(graphs, 0))
```

```
['red-1', 'gray-2', 'pink-3', 'green-4', 'yellow-5', 'purple-6']
```

In [13]:
```
class HashTable:

    def __init__(self):
        self.size = 256
        self.hashmap = [[] for _ in range(0, self.size)]
        # print(self.hashmap)

    def hash_func(self, key):
        hashed_key = hash(key) % self.size
        return hashed_key

    def set(self, key, value):
        hash_key = self.hash_func(key)
        key_exists = False
        slot = self.hashmap[hash_key]
        for i, kv in enumerate(slot):
            k, v = kv
            if key == k:
                key_exists = True
                break

        if key_exists:
            slot[i] = ((key, value))
        else:
            slot.append((key, value))

    def get(self, key):
        hash_key = self.hash_func(key)
        slot = self.hashmap[hash_key]
        for kv in slot:
            k, v = kv
            if key == k:
                return v
```

In [15]:
```
from Cryptodome.Hash import MD5
class ListNode:
    def __init__(self, data):
        self.data = data
        self.link = None
class MyHashSet:
    def __init__(self, storage=200):
        self.storage = storage
        self.contents = [None] * storage
    def add(self, key):
        hashtable = MD5.new()
        hashtable.update(key.encode("utf-8"))
        hashtable = hashtable.hexdigest()
        bucket = int(hashtable , 16)%self.storage
        if self.contents[bucket] == None:
            self.contents[bucket] = ListNode(hashtable)
        else:
            new = ListNode(hashtable)
            now = self.contents[bucket]
            while now.link != None:
                now = now.link
            now.link = new
    def remove(self, key):
        hashtable = MD5.new()
        hashtable.update(key.encode("utf-8"))
        hashtable = hashtable.hexdigest()
        bucket = int(hashtable , 16)%self.storage
        deletebucket = self.contents[bucket]
        if deletebucket != None:
            if deletebucket.link != None:
                if deletebucket.data == hashtable:
                    self.contents[bucket] = deletebucket.link
                else:
                    while deletebucket.link:
                        while deletebucket.link:
                            if deletebucket.data == hashtable:
                                prev.link = deletebucket.link
                                deletebucket = deletebucket.link
                            else:
                                prev = deletebucket
                                deletebucket = deletebucket.link
                        if deletebucket.link == None:
                            if deletebucket.data == hashtable:
                                prev.link = deletebucket.link
            else:
                if deletebucket.data == hashtable:
                    self.contents[bucket] = None
        if self.contains(key) == True:
            self.remove(key)
    def contains(self, key):
        hashtable = MD5.new()
        hashtable.update(key.encode("utf-8"))
        hashtable = hashtable.hexdigest()
        bucket = int(hashtable, 16)%self.storage
        if self.contents[bucket] != None:
            node = self.contents[bucket]
            if node.data == hashtable:
                return True
            else:
                while node.link != None:
                    node = node.link
                    if node.data == hashtable:
                        break
                if node.data == hashtable:
                    return True
                else:
                    return False
        else:
            return False
```

In [17]:
```
def heap(k, n, i):  #堆積
    large = i  #假設i為最大值的樹根
    left = 2 * i + 1      #分成左右子樹
    right = 2 * i + 2

    if left < n and k[i] < k[left]:  #如果左子樹比原先假設的樹根(i)大 則讓左子樹代替原樹根i
        large = left

    if right < n and k[large] < k[right]: #如果右子樹比樹根大 則讓右再替代樹根
        large = right

    if large != i:  #如果原先假設的i不等於最大值 則交換位置 讓最大值成為樹根
        k[i],k[large] = k[large],k[i]  #換位置

def heapSort(k):
    n = len(k) #n為k字元數

    for i in range(n, -1, -1): #設定範圍從n個開始一個一個處理到0(因python是從0開始數所以end設定-1)
                heapify(k, n, i)

    for i in range(n-1, 0, -1):
        k[i], k[0] = k[0], k[i]   # 換位置
        heapify(k, i, 0)

k = [12, 5, 6, 7, 18, 2]
heapSort(k)
n = len(k)
for i in range(n):
    print("%d" %k[i])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-2fef23271b9c> in <module>
     24
     25 k = [12, 5, 6, 7, 18, 2]
---> 26 heapSort(k)
     27 n = len(k)
     28 for i in range(n):

<ipython-input-17-2fef23271b9c> in heapSort(k)
     17
     18     for i in range(n, -1, -1): #設定範圍從n個開始一個一個處理到0(因python是從0開始數所以end設定-1)
---> 19                 heapify(k, n, i)
     20
     21     for i in range(n-1, 0, -1):

NameError: name 'heapify' is not defined
```

In [20]:
```
def heap(k, n, i):  #堆積
    large = i  #假設i為最大值的樹根
    left = 2 * i + 1      #分成左右子樹
    right = 2 * i + 2

    if left < n and k[i] < k[left]:  #如果左子樹比原先假設的樹根(i)大 則讓左子樹代替原樹根i
        large = left

    if right < n and k[large] < k[right]: #如果右子樹比樹根大 則讓右再替代樹根
        large = right

    if large != i:  #如果原先假設的i不等於最大值 則交換位置 讓最大值成為樹根
        k[i],k[large] = k[large],k[i]  #換位置
def heapSort(k):
    n = len(k) #n為k字元數

    for i in range(n, -1, -1): #設定範圍從n個開始一個一個處理到0(因python是從0開始數所以end設定-1)
        heapify(k, n, i)

    for i in range(n-1, 0, -1):
        k[i], k[0] = k[0], k[i]   # 換位置
        heapify(k, i, 0)
```

```
  File "<ipython-input-20-cd81ed7f837f>", line 26
    for i in range(n)
                    ^
SyntaxError: invalid syntax
```

In [21]:
```
def mergesort(listneedtosort):
    index1 = index2 = index3 = 0
    if len(listneedtosort) > 1:
        middlenumber = len(listneedtosort)//2
        leftsort = listneedtosort[:middlenumber]
        rightsort = listneedtosort[middlenumber:]
        mergesort(leftsort)
        mergesort(rightsort)

        while index1 < len(leftsort) and  index2 < len(rightsort):
            if(leftsort[index1] < rightsort[index2]):
                listneedtosort[index3] = leftsort[index1]
                index1 = index1 + 1
            elif(leftsort[index1] >= rightsort[index2]):
                    listneedtosort[index3] = rightsort[index2]
                    index2 = index2 + 1
            index3 = index3 + 1

        while index1 < len(leftsort):
            listneedtosort[index3] = leftsort[index1]
            index3 = index3 + 1

        while index2 < len(rightsort):
            listneedtosort[index3] = rightsort[index2]
            index2 = index2 + 1
            index3 = index3 + 1
```

In [25]:
```
class DisjointSet(dict):
    '''不相交集'''

    def __init__(self, dict):
        pass

    def add(self, item):
        self[item] = item

    def find(self, item):
        if self[item] != item:
            self[item] = self.find(self[item])
        return self[item]

    def unionset(self, item1, item2):
        self[item2] = self[item1]

def Kruskal(nodes, edges):
    all_nodes = nodes  # set(nodes)
    used_nodes = set()
    MST = []
    edges = sorted(edges, key=lambda element: element[2], reverse=True)
    # 對所有的邊按權重升序排列
    while used_nodes != all_nodes and edges:
        element = edges.pop(-1)
        if element[0] in used_nodes and element[1] in used_nodes:
            continue
        MST.append(element)
        used_nodes.update(element[:2])
        # print(used_nodes)
    return MST
```