**EXAMINATION**

**Problem #2:**

You have an image as NxN matrix of pixel (each is number). Write a program to rotate 90 degree that image? Can you do that in place without using any temporary matrix?

\* Program:

import numpy as np  
  
# 2D array  
# Khởi tạo mảng   
arr=np.zeros((10,10))  
for i in range(0,10):  
 for j in range(0,10):  
 arr[i][j]=int(i \* 10 +j)  
print(arr)  
  
#Gía trị của mảng arr2 chính là mảng arr khi xoay sang phải, xuống dưới 90 độ  
arr2=np.zeros((10,10))  
for i in range(0,10):  
 for j in range(0,10):  
 arr2[j][9-i]=arr[i][j]  
print(arr2)

- Sau khi xoay mảng 2 chiều cũ thì ta sẽ có được một mảng mới. Số lượng và giá trị của 2 mảng là giống nhau, nhưng mỗi phần tử sẽ có vị trí thay đổi

- Vị trí của phần tử ở mảng mới sẽ có giá trị của cột bằng với giá trị cột của phần tử ở mảng cũ, vị trí dòng của phần tử ở mảng mới sẽ bằng với chiều dài của mảng trừ đi 1 và trừ đi giá trị cột của phần tử ở mảng cũ

**Problem #1:**

Write an application to simulate elevators in a building. Let say, one building has N floors and M elevators. Each floor can request elevator to go to another floor. Make sure that your output (console log, any GUI or activity) can show that elevators can receive requests and process to respond to those requests. How do you design a system like that? How do you handle concurrency to make sure elevators are running parallel?

Ta khởi tạo Object:

class Evelator(threading.Thread):  
 def \_\_init\_\_(self, floor, Number):  
 threading.Thread.\_\_init\_\_(self)  
 self.floor = floor  
 self.Number = Number  
 self.floorWaiting = []  
  
 def ReceiveRequest(self,floor\_Number):  
 check=False  
 if(self.floor!=floor\_Number):  
 for i in self.floorWaiting:  
 if(i==floor\_Number) : check=True  
 else:  
 print("This floor is here")  
 return False;  
 if(check==False):  
 self.floorWaiting.append(floor\_Number)  
 return True  
 else:  
 return False  
  
 def RunToFloor(self,floorNumber):  
 while self.floor != floorNumber or self.floor != floorNumber:  
 if self.floor > floorNumber:  
 self.floor = self.floor-1  
 else:  
 self.floor = self.floor+1  
 # time.sleep(2)  
 print("Evelator "+ str(self.Number) + " is in floor "+ str(self.floor))  
 # print("Evelator "+ str(self.Number) + " is in floor "+ str(floorNumber))  
  
 def run(self):  
 while(len(self.floorWaiting)!=0):  
 time.sleep(1)  
 self.RunToFloor(self.floorWaiting[0])  
 self.floorWaiting.remove(self.floorWaiting[0])

- Ta khởi tạo đối tượng thang máy là luồng, để có thể chạy đồng thời 4 thang máy cùng 1 lúc

- Đối tượng thang máy có 3 thuộc tính chính :

1. floor: tầng hiện tại thang máy đang ở

2. Number: chính là số thang máy(từ 1 đến N)

3. FloorWaiting chính là mảng chứa danh sách các yêu cầu(là từ các tầng khác nhau)

- Hàm **def ReceiveRequest(self,floor\_Number):** được dùng để đưa yêu cầu vào thang máy, floorNumber chính là tầng yêu cầu thang máy đến

**-** Hàm **def RunToFloor(self,floorNumber):** chính là hàm để chạy đến tầng thứ floorNumber

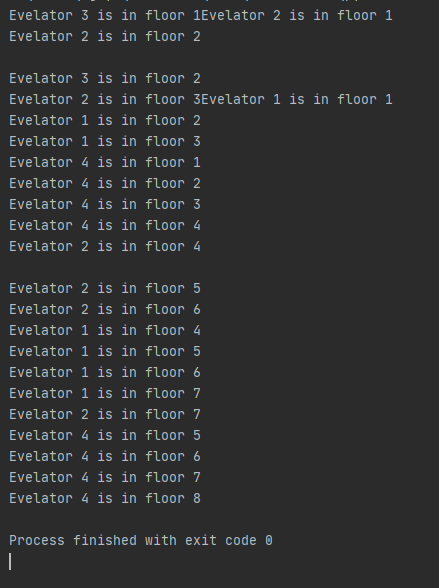
- Hàm def run(self): để chạy hết các yêu cầu từ các tầng trong hàng đợi

Sau khi khởi tạo xong ta bắt đầu chạy chương trình:

#Khởi tạo 4 thang máy ở tầng 0

evelator1=Evelator(0,1)  
evelator2=Evelator(0,2)  
evelator3=Evelator(0,3)  
evelator4=Evelator(0,4)  
  
#Đưa 4 Request vào 4 thang máy  
evelator1.ReceiveRequest(random.randint(1, 10))  
evelator2.ReceiveRequest(random.randint(1, 10))  
evelator3.ReceiveRequest(random.randint(1, 10))  
evelator4.ReceiveRequest(random.randint(1, 10))  
  
#Mỗi thang máy là một luồng chạy riêng, ta chạy 4 thang máy  
evelator1.start()  
evelator2.start()  
evelator3.start()  
evelator4.start()

Ta được kết quả đầu ra



- Chúng ta có thể đưa thêm thuật toán làm tối ưu đường đi cho thang máy nhưng vì chưa đủ khả năng làm trong thời gian giới hạn, tôi xin dừng lại tại đây