Name: _____   Section: 12:30pm or 2pm

Sample Midterm D

Learning Objective:

- Create Python code to automate a given task.

Instructions:

The midterm tests your mastery of skills taught in Weeks 1-5, which culminates in creating simulation models using Python and algorithmic thinking. There are three questions, worth a total of 24 points. The exam is 80 minutes, and is open notes but closed computer. You can bring paper notes or books of any kind, but no computers, tablets, or cell phones are allowed. Before the exam is graded, do not share the questions or your solutions with anyone else, including students of other sections. When you turn in the exam, you must also hand in all scrap paper that you wrote on. Do not use a cell phone, tablet or computer in the classroom before all of the exams are handed in. **Any violation of academic integrity will result in a zero grade for the midterm for everyone involved.**

As long as you fulfill all the specifications described in the problem description, it doesn't matter how you solve the problem or how efficient is your code. However, if you cannot solve a problem, you may get partial credits for submitting whatever you have, including any parts of the four steps of algorithmic thinking.

Note: All three questions are motivated by the operations of ride hailing platforms such as Uber, Lyft, Didi, or Ola.

Q1. Identifying the Closest Driver (7 Points)

Suppose that a certain ride hailing platform always matches a passenger with the closest driver. This question asks you to develop a tool to help the platform identify the closest driver to a given passenger, and estimate how far the driver needs to travel to complete the trip. For simplicity, every location is represented using its (x, y) coordinates and roads are assumed to be either from north to south or from east to west, so the distance between any two points (x_1, y_1) and (x_2, y_2) can be estimated based on the *Manhattan distance*:

$$|x_1 - x_2| + |y_1 - y_2|,$$

where $|\cdot|$ denotes the absolute value (see the Python function "abs").

Write a function called "closest_dispatch" with the following input arguments:

- **locations**: A dictionary in which each key represents an unique ID of the driver, and the corresponding value is a list of two numbers, representing the coordinates of the driver's current location. For example, `{'A':[3,5],'B':[-5,2.5]}` means that driver A is currently at location $(x_A, y_A) = (3, 5)$ and driver B is at location $(x_B, y_B) = (-5, 2.5)$. You can assume that the dictionary has at least one entry.
- **origin**: A list of two numbers, representing the coordinates of the passenger's current location.
- **distination**: A list of two numbers, representing the the coordinates of the passenger's intended destination.

The function should return two objects:

- The ID of the closest driver to the passenger, based on the Manhattan distance from the driver's location and the passenger's origin. If there is a tie, then it doesn't matter which driver is returned.
- The total travel distance for this driver to complete the trip. This is the sum of the Manhattan distance from the driver's location to the passenger's location, and the Manhattan distance from the passenger's location to the intended destination.

See the sample runs below for illustrations.

```
[2]: # Sample run 1
     locations={'Alice':[10,30],'Bob':[20,5],'Charles':[-10,10]}
     closest,movement=closest_dispatch(locations,[10,13],[10,60])
     print(f'Closest driver: {closest} \tTravel distance: {movement}')

Closest driver: Alice    Travel distance: 64
```

```
[3]: # Sample run 2
     locations={'Alice':[10,30],'Bob':[20,5],'Charles':[-10,10]}
     closest,movement=closest_dispatch(locations,[10,12],[-10,10.5])
     print(f'Closest driver: {closest} \tTravel distance: {movement}')

Closest driver: Bob      Travel distance: 38.5
```

```
[4]: # Sample run 3
     locations={'Alice':[10,30],'Bob':[20,5],'Charles':[-10,10]}
     closest,movement=closest_dispatch(locations,[7.6,11],[20,30])
     print(f'Closest driver: {closest} \tTravel distance: {movement}')

Closest driver: Bob      Travel distance: 49.8
```

```
[5]: # Sample run 4
     locations={'Alice':[10,30],'Bob':[20,5],'Charles':[-10,10]}
     closest,movement=closest_dispatch(locations,[7.4,11],[20,30.1])
     print(f'Closest driver: {closest} \tTravel distance: {movement}')

Closest driver: Charles        Travel distance: 50.1
```

Write your final code below:

Q2. Queuing at Airport Pickup Station (8 Points)

Suppose that a certain ride hailing platform organizes pickups from a certain airport as follows: upon landing, all passengers proceed to a single pickup station, and drivers also line up their cars at the station. Due to capacity limitations at the station, at most k passengers can be picked up each minute. This question asks you to build a tool to estimate the average waiting time for both passengers and drivers, given the number of passengers and drivers who arrive at the station each minute.

Write a function called "airport_queue" with the following input arguments:

- **passengers**: a list representing the number of passengers who arrive at the station each minute. You may assume that each entry of the list is a non-negative integer, and that the sum of entries in the list is strictly positive.
- **drivers**: a list representing the number of drivers who arrive at the station each minute. You may assume that each entry is a non-negative integer, and that the sum of entries is strictly positive. Moreover, the lengths of the two lists are the same.
- **k**: a positive integer representing the maximum number of pickups per minute.

Assume that initially, both the passenger and driver queues are empty. The following table illustrates the queueing dynamic under the following parameters:

```
passengers=[5,10,20,5,0,0,0,0,15,0]
drivers=[20,0,0,0,0,30,40,20,0,0]
k=10
```

Minute	Passenger Arrivals	Driver Arrivals	Passenger Queue	Driver Queue	Pickups
0	5	20	5	20	5
1	10	0	10	15	10
2	20	0	20	5	5
3	5	0	20	0	0
4	0	0	20	0	0
5	0	30	20	30	10
6	0	40	10	60	10
7	0	20	0	70	0
8	15	0	15	70	10
9	0	0	5	60	5
Total	**55**	**110**	**125**	**330**	

Note that the columns "Passenger Arrivals" and "Driver Arrivals" are from the two input lists. In each row except the first one, the "Passenger Queue" is equal to the previous row's value plus the number of passenger arrivals in this row, minus the number of pickups in the previous row. An analogous relationship holds for the "Driver Queue" column. In each row, "Pickups" is equal to the minimum of k, "Passenger Queue" and "Driver Queue".

The function should return two objects:

- **average waiting time for passengers**: this is the total of the "Passenger Queue" column divided by the total of the "Passenger Arrivals" column. In the above table, this is $125/55 \approx 2.27$ minutes.
- **average waiting time for drivers**: this is the total of the "Driver Queue" column divided by the total of the "Driver Arrivals" column. In the above table, this is $330/110 = 3$ minutes. These formula are based on a mathematical relationship between waiting time and queue length called "Little's Law," which assumes that both queues instantly clear after the given time period.

See the sample runs below for illustrations.

```
[7]: # Sample run 1
     passengers=[5,10,20,5,0,0,0,0,15,0]
```

```
    drivers=[20,0,0,0,0,30,40,20,0,0]
    k=10
    passenger_wait,driver_wait=airport_queue(passengers,drivers,k)
    print(f'Av. waiting time for passengers is {round(passenger_wait,2)} minutes.')
    print(f'Av. waiting time for drivers is {round(driver_wait,2)} minutes.')

Av. waiting time for passengers is 2.27 minutes.
Av. waiting time for drivers is 3.0 minutes.
```

Write your final code below:

Q3. Simulating a Driver's Movements (9 Points)

This question asks you to simulate a driver's movements and cumulative earnings across a given work day under following simplifying assumptions. There is only a finite number of possible locations, each encoded by a certain locationID. Assume that the driver arrives at a given initial location at time $t = 0$, at which point he begins accepting dispatches from the platform. Each dispatch tells him to drive a passenger to a given destination. The driver accepts every dispatch and does not move locations unless he is fulfilling a dispatch. Given the driver's current location, the destination of his next dispatch is independently drawn from a given distribution. If the driver's current location is i and the destination of his next dispatch is j, then the time elapsed between his arrival at i and his arrival at j is independently drawn from a uniform distribution between certain parameters a_{ij} and b_{ij}, where $a_{ij} \leq b_{ij}$. (This time elapsed already includes the idle time waiting for the dispatch, the time needed to pickup the passenger, and the travel time for completing the dispatch.) The driver ends his work day and stops accepting dispatches when he has completed n dispatches, or if the cumulative time t after completing a dispatch is greater than or equal to a certain threshold T.

Write a function called "driver_movement" with the following input parameters:

- **locations**: a list of possible locationIDs. You may assume that each entry is a string. Denote the length of this list by m. You may assume that $m \geq 1$.
- **initial**: the driver's locationID at time $t = 0$.
- **transition_probs**: a dictionary with m entries in which each key corresponds to a locationID and each value is a list of length m representing the probability of moving to the corresponding locationID in the list "locations", assuming that the current location is given by the dictionary key.
- **df_a**: a DataFrame in which the row and column labels are both given by the list "locations", and each value `df_a.loc[i,j]` corresponds to the parameter a_{ij}.
- **df_b**: analogous in format to "df_a", except that `df_b.loc[i,j]` corresponds to the parameter b_{ij}.
- **df_earnings**: analogous in format to "df_a", except that `df_earnings.loc[i,j]` corresponds to the earnings from moving from location i to the next location j.
- **n**: a positive integer corresponding to the maximum number of dispatches the driver is willing to fulfill.
- **T**: a positive number corresponding to the time after which the driver will no longer accept new dispatches.

The function should return three objects:

- **timeList**: a list in which the first entry is 0, and each subsequent entry corresponds to the time that the driver completes a dispatch.
- **locationList**: a list in which the first entry is the same as **initial**, which is the locationID at time 0. Each subsequent entry corresponds to the location of the driver upon completing the corresponding dispatch.
- **earningsList**: a list in which the first entry is 0, and each subsequent entry stores the cumultative earnings for the driver after completing a dispatch.

See the sample outputs for illustrations. (In the sample outputs, whether a whole number is displayed as int or float does not matter.)

```
[12]: # Sample run 1  (deterministic example)
      import pandas as pd
      locations=['Downtown', 'Airport']
      initial='Airport'
      transition_probs={'Downtown':[0,1],'Airport':[1,0]}
      df_a=df_b=pd.DataFrame([[0,50],[70,0]],index=locations,columns=locations)
      df_earnings=pd.DataFrame([[0,30],[40,0]],index=locations,columns=locations)
      t,l,e=driver_movement(locations,initial,transition_probs,df_a,df_b,df_earnings,10,190)
      print('Time\tLocation\tCumulative Earnings')
      for i in range(len(t)):
          print(f'{t[i]}\t{l[i]:10}\t{e[i]}')

      Time    Location        Cumulative Earnings
```

```
0        Airport        0
70.0     Downtown       40
120.0    Airport        70
190.0    Downtown       110
```

```python
[13]: # Sample run 2  (randomized example)
      import pandas as pd
      locations=['Downtown', 'Airport']
      initial='Airport'
      transition_probs={'Downtown':[0.7,0.3],'Airport':[1,0]}
      df_a=df_b=pd.DataFrame([[5,50],[70,0]],index=locations,columns=locations)
      df_earnings=pd.DataFrame([[10,30],[40,0]],index=locations,columns=locations)
      t,l,e=driver_movement(locations,initial,transition_probs,df_a,df_b,df_earnings,10,400)
      print('Time\tLocation\tCumulative Earnings')
      for i in range(len(t)):
          print(f'{t[i]}\t{l[i]:10}\t{e[i]}')
```

```
Time     Location        Cumulative Earnings
0        Airport         0
70.0     Downtown        40
75.0     Downtown        50
80.0     Downtown        60
130.0    Airport         90
200.0    Downtown        130
205.0    Downtown        140
210.0    Downtown        150
260.0    Airport         180
330.0    Downtown        220
335.0    Downtown        230
```

Write your final code below (use the back if needed):