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

DSO-570 Midterm Exam (Fall 2023)

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.

All three problems are motivated by the home improvement industry, which is a 400-800 billion dollar industry in the US but consumer satisfaction the lowest among all industries. This exam encourages you to brainstorm ways in which analytics can improve the industry.

Q1. Ordering Kitchen Cabinets (7 Points)

IKEA offers high-quality kitchen cabinets at low prices, but the cabinets are often out of stock. Cabinet installers require that all cabinets are delivered by the date of installation. You want to begin your kitchen remodel as soon as possible using IKEA cabinets, so you decided to order each cabinet separately as soon as it becomes in stock, and store them in your garage until everything has arrived. Suppose that you know the dates when every cabinet you need will be in stock, as well as the delivery times. These are encoded in data structures with the following format:

```
instock={'A': [1,3,8,9,14,18,30,31,60,61,80,90,100],
         'B': [5,6,7,20,21,40,70,71,100],
         'C': [1,2,9,10,25,50,90,91,100],
         'D': [3,4,10,11,29,35,59,81,100]
        }
delivery_time={'A':3,'B':5,'C':7,'D':4}
today=21
```

In the above, there are four cabinets that you need: A, B, C and D. Cabinet A is in stock on Day 1, Day 3, Day 8, Day 9, Day 14, etc. You can only order a cabinet on a day that it is in stock. Once you order cabinet A, it takes 3 days for it to be delivered. The day that you begin looking for cabinets is Day 21, as represented by the variable `today`. The following table illustrates how you can calculate the earliest date when all the cabinets can be delivered.

	Cabinet A	Cabinet B	Cabinet C	Cabinet D
Earliest order date	30	21	25	29
Earliest delivery date	33	26	32	33

In the above table, the first row gives the earliest date on or after `today` when the cabinets will be in stock. For example, starting from Day 21 (today), Cabinet A will next be in stock on Day 30, Cabinet B on Day 21, Cabinet C on Day 25, and Cabinet D on Day 29. The second row adds the delivery time for each cabinet, as given by the dictionary `delivery_time`. The earliest date when everything can be delivered is Day 33, which is the maximum of the numbers in the second row.

Write a function called `earliest` with three input arguments, with the same format as in the above. The function should return the earliest date on which everything can be delivered, as in the sample runs below. You may assume that all of the input numbers are positive integers. Moreover, all of the lists in the dictionary `instock` are sorted in ascending order, and the last number in every list is always greater than or equal to `today`. In other words, assume that it is always possible to obtain every cabinet within the given lists of dates, so the function can always return a valid answer.

```
[2]: # Sample runs:
     instock={'A': [1,3,8,9,14,18,30,31,60,61,80,90,100],
              'B': [5,6,7,20,21,40,70,71,100],
              'C': [1,2,9,10,25,50,90,91,100],
              'D': [3,4,10,11,29,35,59,81,100]
             }
     delivery_time={'A':3,'B':5,'C':7,'D':4}
     earliest(instock,delivery_time, 21)
```

```
33
```

```
[3]: print(earliest(instock,delivery_time, 1))
```

```
10
```

```
[4]: earliest(instock,delivery_time, 60)
```

```
97
```

Write your final code below:

Q2. Scheduling Contractors (8 Points)

For your kitchen remodel, you need to hire multiple specialized contractors, who requested to come one at a time because space in your kitchen is limited. Suppose that you are given the dates that each contractor is available to work on your project, the sequence in which the contractors requested to come, and the number of days that each contractor needs to work in your kitchen, as below.

```
available={'Cabinet Installers':[3,4,7,8,9,12,21,23],
           'Electrician':[6,7,11,12,15,16,24,25],
           'Plumber':[1,4,7,8,13,17,19,21],
           'Painters':[5,8,9,12,13,14,15,18,19,20,23]}
sequence=['Cabinet Installers','Plumber','Electrician','Painters']
days_needed=[3,2,1,3]
```

In the above, the cabinet installers need to be scheduled first, and they need 3 days, as given by the first element in the lists `sequence` and `days_needed`. The plumber is to be scheduled next, and she needs 2 days, as given by the second element in these two lists. You want to schedule each contractor as early as possible, so you schedule the cabinet installers on Days 3, 4 and 7, and you schedule the Plumber on Days 8 and 13. (Note that the scheduled dates for two distinct contractors cannot overlap.) Next, you schedule the electrician on Day 15, and the painters for Days 18, 19, and 20. Hence, the earliest date by which you can complete the kitchen remodel is Day 20, which is when the painters will be done.

Write a function called `completion_date` with three input arguments, with the same format as in the above. The function should return the earliest date on which the kitchen can be completed, as in the sample runs below. You may assume that all of the input numbers are positive integers. Moreover, assume that all of the lists in the dictionary `available` are sorted in ascending order, and it is always possible to complete the job within the dates given, so the function can always return a valid answer.

```
[7]: # Sample runs:
     available={'Cabinet Installers':[3,4,7,8,9,12,21,23],
                'Electrician':[6,7,11,12,15,16,24,25],
                'Plumber':[1,4,7,8,13,17,19,21],
                'Painters':[5,8,9,12,13,14,15,18,19,20,23]}
     sequence=['Cabinet Installers','Plumber','Electrician','Painters']
     days_needed=[3,2,1,3]
     completion_date(available,sequence,days_needed)

20

[8]: completion_date(available,['Plumber','Painters'],[4,4])

14

[9]: completion_date(available,['Cabinet Installers','Painters'],[4,5])

15

[10]: completion_date(available,['Plumber','Cabinet Installers','Plumber'],[2,3,1])

13
```

Write your final code below:

Q3. Simulating Contractor Availability (9 Points)

One of the inputs in the previous question is a list of days on which a contractor is available to work. This question asks you to implement a simulation model to generate such a list of available days. Suppose that the contractor has already scheduled a number of other jobs, and he is available to work on your project only when he is not occupied with these other jobs. There are T days in which you wish you simulate, which are labelled Day 1, Day 2, $\cdots$, Day T. On each of these T days, the contractor may have already promised other customers to begin other jobs; the number of jobs requested by other customers to begin on each of the T days is Poisson distributed with mean 0.4. The length of each job is drawn independently from the following distribution:

Length (number of days)	1	2	3	5
Probability	.3	.4	.2	.1

In other words, on average 30% of jobs require only one day of work, 40% require two days, 20% require 3 days, and 10% require 5 days. When a contractor receives a job request, he always tries to begin the job on the date that the customer requests. If he cannot do so because he is occupied with another job, then he pushes the job back to the next date on which he is available. Moreover, assume that job requests are scheduled sequentially in ascending order of the requested begin date. For example, suppose that the contractor already has four jobs from other customers:

Job	Requested Begin Date	Length
A	Day 2	3
B	Day 3	2
C	Day 8	1
D	Day 8	1

Here, Job A is requested to begin on Day 2 and will last 3 days. Job B is requested to begin on Day 3 and will lasts 2 days. Job C and D are both requested to begin on Day 8 and will each require one day. After accepting these four jobs, the contractor's calendar on Days 1 through 10 would look like this:

Day	1	2	3	4	5	6	7	8	9	10
Available?	1	0	0	0	0	0	1	0	0	1

This is because Job A is scheduled for Days 2, 3, and 4, since this is the first job request he receives. Job B is pushed back to Days 5 and 6 because he is occupied with Job A on Day 3. Job C is scheduled on Day 8, and Job D is pushed back to Day 9 since he is busy with Job C on Day 8. When you contact him about your job, his list of available dates is `[1,7,10]`, because the second row shows 1 only on Days 1, 7, and 10.

Write a function called `availability` that takes in one input argument, which is the positive integer T. The function should return a list of days in which the contractor is available for your job among days 1 through T. (In the above example, $T = 10$.) Assume that before Day 1, the contractor has not accepted any other jobs that would run over to Day 1.

```
[12]: # Sample runs: (outputs are randomly generated)
      availability(10)
```

```
[1, 7, 10]
```

```
[14]: availability(30)
```

```
[3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 15, 16]
```

Write your final code below: