Week 4: Algorithmic Thinking II

Session 7: Writing Correct Code More Quickly

Tips on making your process of coding more efficient:

i. Clarify the underlying logic in English before proceeding to code.
ii. When encountering potentially tricky logic, write code fragments on paper first to capture the essence, before typing on the computer.
iii. Be familiar with Python syntax and take advantage of shortcuts in the language.
iv. Debug using intermediate printing and code dissection. Don't just stare at the code or try random things in hope that it works.

List Comprehension

List comprehension is a shorthand notation for generating a list. The syntax looks like for loops but technically speaking they are entirely distinct concepts.

```
[1]: # Initializing a list with zeros.
     l=[0 for i in range(10)]
     l
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
[2]: l[0]=3
     l[1]=5
     l
```

```
[3, 5, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
[3]: # Quickly generating a list of squares
     [num*num for num in range(1,6)]
```

```
[1, 4, 9, 16, 25]
```

```
[4]: # Adding conditions
     [num*num for num in range(1,6) if num!=3]
```

```
[1, 4, 16, 25]
```

```
[5]: sum([num*num for num in range(1,6)])
```

```
55
```

```
[6]: # Dictionary comprehension
     sales={'USA':3000, 'China':2000, 'India':4000}
     revenue={country:sales[country]*5 for country in sales}
     revenue
```

```
{'USA': 15000, 'China': 10000, 'India': 20000}
```

```
[7]: {country:len(country) for country in sales}
```

```
{'USA': 3, 'China': 5, 'India': 5}
```

```
[8]: {country:len(country) for country in sales if len(country)==5}
```

```
{'China': 5, 'India': 5}
```

Exercise 4.1 Practicing List Comprehension

Download the Jupyter notebook attached to the Blackboard link for this exercise and submit it there after completing it. The notebook contains the following questions.

Execute the following cell to load in the list named “sales”:

```
[9]: sales=[10,20,5,25,30,12,18,50,30,20,10,4]
```

a) Using list comprehension, write one line to generate the list of elements in “sales” that are at least 20.

b) Write one line to count the number of elements in “sales” that are at least 20. Hint: find the length of the above list.

c) Write one line to count the number of elements that are between 5 and 10 (inclusive).

d) Generate a list of zeros with the same length as the list `sales`.

e) Use dictionary comprehension to write one line that generates the following dictionary. The values are from the list `sales` and the keys are the strings “Week 1”, “Week 2”, etc.

```
{'Week 1': 10,
 'Week 2': 20,
 'Week 3': 5,
 'Week 4': 25,
 'Week 5': 30,
 'Week 6': 12,
 'Week 7': 18,
 'Week 8': 50,
 'Week 9': 30,
 'Week 10': 20,
 'Week 11': 10,
 'Week 12': 4}
```

f) Suppose that the price of the product in each week is given by the following list. Use dictionary comprehension to write one line to yield the dictionary mapping the week name to the revenue each week. (Revenue in each week is equal to the sales multiplied by the price.)

```
price=[5,5,10,6,5,5,6,8,8,8,5,10]
```

```
{'Week 1': 50,
 'Week 2': 100,
 'Week 3': 50,
 'Week 4': 150,
 'Week 5': 150,
 'Week 6': 60,
 'Week 7': 108,
 'Week 8': 400,
 'Week 9': 240,
 'Week 10': 160,
 'Week 11': 50,
 'Week 12': 40}
```

Sample Problem: Queue Analysis

A popular fast food restaurant is planning to open a branch in a new location and wants to decide how many servers to hire. To help them, write a function "avQueueLength" with two input parameters:

- k: the number of customers that can be served in a minute. (assumed to be integer)
- demand: a list of integers specifying how many customers arrive each minute. (For simplicity, assume that customers arrive at the beginning of each minute and up to k customers can be served instantly.)

The function should return (not print) the average queue length.
Sample run:

```
avQueueLength(3,[2,3,6,8,10,2,1,0,1,0])
```

Output: 7.2

The following table summarizes the evolution of the queue corresponding to the above sample input.

Minute	# of Arrivals	# Served	Queue Length
0	–	–	0
1	2	2	0
2	3	3	0
3	6	3	30
4	8 +3	3	8 5
5	10+5	3	15 12
6	2 +12	3	14 11
7	1 +11	3	12 9
8	0 +9	3	9 6
9	1 +6	3	7 4
10	0 +4	3	4 1
Average	–	–	**7.2**

In-Class Exercise: Clarifying the Underlying Logic

a) Walk through the above table to figure out the underlying logic. Write brief English descriptions of the logic of the calculations. (This corresponds to Steps 1 and 2 of Algorithmic thinking.)

for every minute, check the number of people arriving and how many of them can be served. If all of them cannot be served, move them over to the next minute.

b) Write code fragments on a piece of paper corresponding to the trickiest parts of the logic described in part a).

Exercise 4.2 Python Code for Queue Analysis

Download the Jupyter notebook attached to this exercise on Blackboard and submit it there after completing it. The notebook asks you to carry out Steps 3 and 4 of algorithmic thinking to complete the sample problem.

```
[21]: # Test code 1
      avQueueLength(3,[2,3,6,8,10,2,1,0,1,0])
```

```
7.2
```

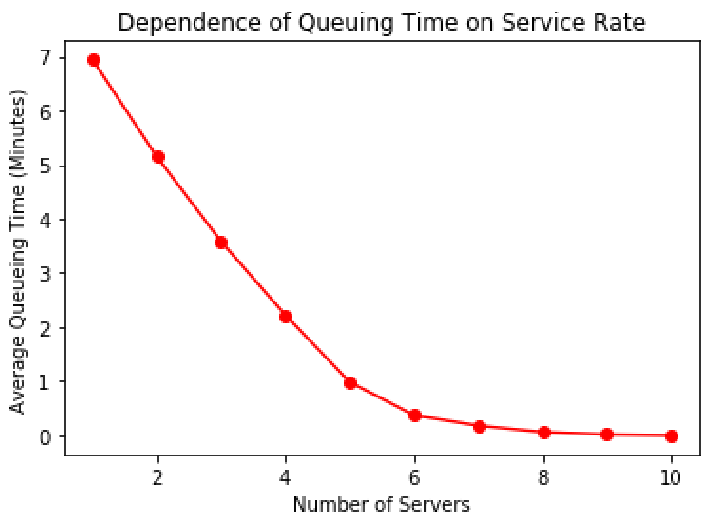
```
[22]: # Test code 2
      demand=[2,3,6,8,10,2,1,0,1,0]
      for k in range(1,6):
          print(f'Av. Queue Length with {k} server(s):',avQueueLength(k,demand))
```

```
Av. Queue Length with 1 server(s): 17.2
Av. Queue Length with 2 server(s): 11.7
Av. Queue Length with 3 server(s): 7.2
Av. Queue Length with 4 server(s): 4.0
Av. Queue Length with 5 server(s): 2.2
```

To illustrate why the above function is useful, according to a mathematical result known as Little's Law, the average queuing time of customers is

$$\frac{\text{Average Queue Length}}{\text{Average Arrival Rate}} = \frac{7.2}{3.3} \approx 2.2 \text{ minutes.}$$

```
[33]: # Test code 3
      demand=[2,3,6,8,10,8,4,3,2,6,3,2,1,4,5]
      servers=range(1,11)
      avArrivalRate=sum(demand)/len(demand)
      queuingTime=[avQueueLength(k,demand)/avArrivalRate for k in servers]
      import matplotlib.pyplot as plt
      plt.plot(servers,queuingTime,'ro-')
      plt.xlabel('Number of Servers')
      plt.ylabel('Average Queueing Time (Minutes)')
      plt.title('Dependence of Queuing Time on Service Rate')
      plt.show()
```



Exercise 4.3 Forecasting Harvard Tuition

Complete the Jupyter notebook attached to the Blackboard link for this exercise and submit it there after completing it. The notebook asks you to apply all four steps of algorithmic thinking to solve the following problem.

An **exponential smoothing** forecast is given by

$$F_1 = A_0,$$

$$F_t = \alpha A_{t-1} + (1-\alpha)F_{t-1} \quad \text{for } t > 1,$$

where - $\alpha \in (0,1]$ is the smoothing constant chosen by the user. - A_t is the actual value for time period $t = 0, 1, ...,$ - F_t is the forecast for time period $t = 1, ...$ (Note that there is no F_0 because no forecast is available for the initial period $t = 0$.)

Write a program that applies exponential smoothing to forecast the tuition of Harvard's MBA program in 2018 using $\alpha = 0.8$, and calculates the mean absolute deviation of the forecast from 2000-2017 (inclusive), as defined by the average value of $|F_t - A_t|$ for these years. The outputed print statement must exactly match the sample output below, and the final outputs should be rounded to 2 decimal places. The input data is given in the code cell below.

Sample output:

```
Tuition forecast for 2018: 63064.18
Mean error since 2000: 2577.42
```

The following table illustrates the logic in the first few years.

Year	Tuition	Forecast	Error
1985	10000	———	———
1986	10750	10000	$\|10750 - 10000\| = 750$
1987	11900	$0.8(10750) + 0.2(10000) = 10600$	$\|11900 - 10600\| = 1300$
1988	13300	$0.8(11900) + 0.2(10600) = 11640$	$\|13300 - 11640\| = 1660$
1989	14250	$0.8(13300) + 0.2(11640) = 12968$	$\|14250 - 12968\| = 1282$

...

Hints: Absolute values can be calculated using the `abs(...)` functions, see `help(abs)`.

```
[24]: # Data
      years = [1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997,
      tuitions = [10000, 10750, 11900, 13300, 14250, 15350, 16400, 17500, 18550, 19750, 21000
```