Contents

Week 4: Coding and Debugging More Quickly

In this week, we will continue practicing applying algorithmic thinking to tackle more complex problems. The emphasis however will be to go through the process more efficiently, so that you can more quickly write correct code and debug. Mastering these skills will greatly improve your productivity when coding and reduce the pains of debugging.

Session 7: Code Dissection and Print Debugging

Debugging Demonstration

Observe carefully how the instructor debugs the following attempt of Exercise 2.7 using code dissection and intermediate printing.

```
[ ]: def avQueueLength(k,demand):
         total=0
         for minute in range(1,len(demand)+1):
             arrivals=sum(demand[0:minute])
             served = []
             served.append(min(arrivals, 3))
             queue = []
             if arrivals<=3:
                 queue.append(0)
             elif arrivals>3:
                 q=arrivals - sum(served)
                 queue.append(q)
             total+=queue
         return total/len(demand)
```

Replicate bug outside of function

```
[2]: k,demand=3,[2,3,6,8,10,2,1,0,1,0]
     total=0
     for minute in range(1,len(demand)+1):
         arrivals=sum(demand[0:minute])
         served = []
         served.append(min(arrivals, 3))
         queue = []
         if arrivals<=3:
             queue.append(0)
         elif arrivals>3:
```

```
            q=arrivals - sum(served)
            queue.append(q)
        total+=queue
    print( total/len(demand))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-aa174f742e89> in <module>
     11         q=arrivals - sum(served)
     12         queue.append(q)
---> 13     total+=queue
     14 print( total/len(demand))

TypeError: unsupported operand type(s) for +=: 'int' and 'list'
```

Print debugging

```
[1]: k,demand=3,[2,3,6,8,10,2,1,0,1,0]
     total=0
     print('Minute\tArrival\tServed\tQueue')
     for minute in range(1,len(demand)+1):
         arrivals=sum(demand[0:minute])
         served = []
         served.append(min(arrivals, 3))
         queue = []
         if arrivals<=3:
             queue.append(0)
         elif arrivals>3:
             q=arrivals - sum(served)
             queue.append(q)
         #total+=queue
         print(f'{minute}\t{demand[minute]}\t{served}\t{queue}')
     print(queue)
```

```
Minute  Arrival Served  Queue
1       3       [2]     [0]
2       6       [3]     [2]
3       8       [3]     [8]
4       10      [3]     [16]
5       2       [3]     [26]
6       1       [3]     [28]
7       0       [3]     [29]
8       1       [3]     [29]
9       0       [3]     [30]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-1-464ba0de9939> in <module>
     13         queue.append(q)
     14     #total+=queue
---> 15     print(f'{minute}\t{demand[minute]}\t{served}\t{queue}')
     16 print(queue)

IndexError: list index out of range
```

```
[6]: k,demand=3,[2,3,6,8,10,2,1,0,1,0]
     print('Minute\tArrival\tServed\tQueue')
     queue=0
     total=0
     for minute in range(len(demand)):
         arrival=demand[minute]
         served=min(queue+arrival, k)
         queue+=arrival-served
         total+=queue
         print(f'{minute+1}\t{arrival}\t{served}\t{queue}')
     print(total/len(demand))
```

```
Minute  Arrival Served  Queue
1       2       2       0
2       3       3       0
3       6       3       3
4       8       3       8
5       10      3       15
6       2       3       14
7       1       3       12
8       0       3       9
9       1       3       7
10      0       3       4
7.2
```

```
[10]: def avQueueLength(k,demand):
          #print('Minute\tArrival\tServed\tQueue')
          queue=0
          total=0
          for minute in range(len(demand)):
              arrival=demand[minute]
              served=min(queue+arrival, k)
              queue+=arrival-served
              total+=queue
              #print(f'{minute+1}\t{arrival}\t{served}\t{queue}')
          return total/len(demand)
```

Exercise 4.1: Practicing Debugging

In the following, you should try to make the code correct using as few changes as possible, after figuring out the errors using code dissection and print debugging. Do not rewrite the code from scratch or copy/paste a correct solution. Refer to previous weeks' handouts for the instructions to these exercises.

a) Attempt for the "Referral Marketing" Example from Week 2

```
[ ]: # Code to debug
     def buyer(nmonths):

         buyers=[1]
         for n in nmonths:
             buyers = buyers[-1]+buyers[-2]
         return buyers[-1]
```

```
[62]: # Test code
      buyer(12)
```

```
233
```

b) Attempt for the "Epidemic Capacity Planning" Example from Week 3

[ ]:
```
# Code to debug
def capacityNeeded(arrivals):
    maxdemand=[arrivals[0]]
    week=0
    for i in range(0,(len(arrivals))):
        week+=1
        if i>=2:
            demand=arrivals[i]+arrivals[i-1]+arrivals[i-2]
        if i>=1:
            demand=arrivals[i]+arrivals[i-1]
     #   else:
    #      demand=arrivals[i]
        if maxdemand<demand:
            maxdemand=demand
    print(maxdemand)
```

[38]:
```
# Test code 1
arrivals=[5,8,3,10,7,4,9,5,8]
print('Capacity Needed:',capacityNeeded(arrivals))
```

```
Week    Arrival Demand  Max Demand
1       5       5       5
2       8       13      13
3       3       16      16
4       10      21      21
5       7       20      21
6       4       21      21
7       9       20      21
8       5       18      21
9       8       22      22
Capacity Needed: 22
```

c) Attempt for Exercise 3.2: Demand Estimation for Substitutable Products

[ ]:
```
def demand(priceVector,values):
    for i in values:
        product1=[]
        product2=[]
        diff0=values[0]-priceVector[0]
        diff1=values[1]-priceVector[1]
        if diff0<0 and diff1<0:
            continue
        elif diff0>=diff1:
            product1.append(1)
        else:
            product2.append(1)
        totalproduct=product1+product2
        return totalproduct
```

[53]:
```
# Test code 2
values=[[25,15],[18,18],[25,20],[25,35],[18,20],[26,21]]
priceVector=[25,20]
demand(priceVector,values)
```

```
[3, 2]
```

Exercise 4.2: Grocery Store Restocking

This question asks you to make a tool that helps a grocery store to analyze their policy for restocking shelves for a certain non-perishable item. Write a function called `analyzeScenario` with three input parameters:

- `demandList`: a non-empty list of non-negative integers representing the forecasted daily demand for the item, corresponding to a period of consecutive days. The number of days is `len(demandList)`.
- `stockingLevel`: a positive integer representing the maximum number of units that the store will stock on its shelves at any time.
- `minimumLevel`: a non-negative integer representing the minimum number of units on the shelves that the store can tolerate without restocking.

Assume that the store makes its stocking decision at the end of each day after closing. If the leftover inventory on the shelf at the end of a day is strictly below the "minimumLevel", then the store will restock to a full shelf, and the inventory at the beginning of the next day will be equal to "stockingLevel". If the leftover inventory at the end of a day is greater than or equal to "minimumLevel", then the store will not add anything to the shelf, and the inventory at the beginning of the next day will be the same as the leftover inventory. On the first day, the shelf is full, so the inventory level is equal to "stockingLevel".

Your function should print (not return) the number of times it would decide to restock during the period represented by the input data.

For example, the sample run

```
analyzeScenario([3,4,2,5,15,3,9,3,1,3,9],10,3)
```

should result in exactly the following message printed to screen.

```
The store needs to restock 4 times.
```

The following table illustrates the inventory dynamics.

Beginning Inventory	Demand	Leftover Inventory	Restock?
10	3	7	No
7	4	3	No
3	2	1	Yes
10	5	5	No
5	15	0	Yes
10	3	7	No
7	9	0	Yes
10	3	7	No
7	1	6	No
6	3	3	No
3	9	0	Yes
# of times to restock:			**4**

Note that if demand is greater than the beginning inventory, the leftover inventory is zero. Otherwise, the leftover inventory is equal to the beginning inventory minus the demand. The final answer (the number of times to restock) is equal to the number of Yes's in the last column of the table.

Sample run 2:

```
analyzeScenario([3,4,2,5,15,3,9,3,1,3,9],9,3)
```

The printed message should be exactly as below:

```
The store needs to restock 6 times.
```

Sample run 3:

`analyzeScenario([8,3,2,6,9,3,5,2,9,10],9,5)`

The printed message should be exactly as below:

```
The store needs to restock 7 times.
```

Solve this problem by applying the four steps of algorithmic thinking.

Exercise 4.3: Demand Estimation with n Substitutable Products

This exercise generalizes Exercise 3.3 to n products, where n is any positive integer.

Write a function called `demand` with two input arguments:

- `prices`: a list of n prices, one for each product.
- `values`: a list in which each element represents a customer's valuations for the n products. The valuations is a list of length n, which corresponds to the customer's willingness to pay for each of the n products.

The function should return a list of length n representing the number of each product sold. You should assume that each customer:

- Does not purchase anything if his/her valuation for each product is strictly less than its price.
- Otherwise, purchase the product in which the difference between his/her valuation and the price is the largest. When there is a tie, the customer will purchase the product with the smaller index.

For example, if `prices=[10,8,12]`, then

- A customer with valuations `[9,7,11]` purchases nothing.
- A customer with valuations `[10,8,12]` purchases product 1.
- A customer with valuations `[9,8,12]` purchases product 2.
- A customer with valuations `[9,8,13]` purchases product 3.

Sample run 1:

```
prices=[10,8,12]
values=[[9,7,11],[10,8,12],[9,8,12],[9,8,13]]
ans=demand(prices,values)
for i in range(len(prices)):
    print(f'Demand for product {i+1}:',ans[i])
```

Correct output:

```
Demand for product 1: 1
Demand for product 2: 1
Demand for product 3: 1
```

Sample run 2:

```
prices=[20,15,30]
values=[[30,30,20],[40,10,15],[18,13,29],[40,30,50],[10,30,50],[10,10,10],[20,15,30]]
ans=demand(prices,values)
for i in range(len(prices)):
    print(f'Demand for product {i+1}:',ans[i])
```

Correct output:

```
Demand for product 1: 3
Demand for product 2: 1
Demand for product 3: 1
```

Solve this problem by applying the four steps of algorithmic thinking.

Session 8: 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 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 code dissection and print debugging. Don't just stare at the code or try random things in hope that it works.

Exercise 4.4: Simulating Availabilities of Hospital Beds

One challenge in health care operations is to forecast the number of hospital beds that are available at a given time, since patients admitted in the past may stay for several days and the number of beds are limited. If no more beds are available, then incoming patients may need to be turned away.

Write a function called `admissionSimulation` with three input arguments:

- `demandList`: a list of positive integers representing the number of incoming patients desiring a hospital bed in each day. (The first number corresponds to day 0, the second number to day 1, and so on.)
- `beds`: a positive integer representing the total number of hospital beds available.
- `stay`: a positive integer representing the number of days each admitted patient will stay. If `stay=1`, then every admitted patient leaves before any incoming patients arrive the next day. If `stay=2`, then each patient admitted on day t will occupy a bed also for day $t+1$, and leave before incoming patients arrive on day $t+2$.

The function should return a list `admissionRecord`, corresponding to the number of incoming patients admitted on each day.

Sample run 1:

```
demandList=[1,2,1,0,2,3]
beds=2
stay=2
admissionRecord=admissionSimulation(demandList,beds,stay)
print(f'Day\tDemand\tAdmitted')
for i in range(len(demandList)):
    print(f'{i}\t{demandList[i]}\t{admissionRecord[i]}')
```

Correct output:

```
Day   Demand   Admitted
0       1       1
1       2       1
2       1       1
3       0       0
4       2       2
5       3       0
```

Sample run 2:

```
demandList=[5,8,6,8,4,4,8,6,1]
beds=7
stay=3
admissionRecord=admissionSimulation(demandList,beds,stay)
print(f'Day\tDemand\tAdmitted')
for i in range(len(demandList)):
    print(f'{i}\t{demandList[i]}\t{admissionRecord[i]}')
```

Correct output:

```
Day   Demand   Admitted
0       5       5
1       8       2
2       6       0
3       8       5
4       4       2
5       4       0
6       8       5
7       6       2
8       1       0
```

Sample run 3:

```
# Sample run 3
demandList=[5,8,6,4,4,4,8,1,3]
beds=7
stay=3
admissionRecord=admissionSimulation(demandList,beds,stay)
print(f'Day\tDemand\tAdmitted')
for i in range(len(demandList)):
    print(f'{i}\t{demandList[i]}\t{admissionRecord[i]}')
```

Correct output:

```
Day      Demand   Admitted
0       5       5
1       8       2
2       6       0
3       4       4
4       4       3
5       4       0
6       8       4
7       1       1
8       3       2
```

Hint: In Step 2, you want to create a table. The above tables are insufficient to carry through the logic, as you also need to keep track of the number of discharges at the beginning of each day, as well as the number of available beds before new patients arrive.

Solve this problem by applying the four steps of algorithmic thinking.

Step 1. Understand

Step 2. Decompose

Step 3. Analyze

Step 4. Synthesize

Exercise 4.5: Health Insurance Selection

This question asks you to create a function to help individuals select the most cost effective health insurance given the terms of each plan and the individual's estimated health expenditures.

Write a function called "bestPlan" with two input arguments:

- `planInfo`: a dictionary in which the index is the name of a plan and the value is a list of 3 non-negative numbers. The first number is the annual premium of this plan, the second number is the annual deductible, and the third number is the proportion of expenditures (beyond the deductible) reimbursed by the plan.
- `expenditure`: a non-negative number representing the individual's total health care spending without any health insurance.

The function should return (not print) the name of the best plan for the individual, where best is defined as having the lowest total cost, as given by

$$\text{Total cost} = \text{Premium} + \text{Expenditure} - \text{Reimbursements},$$

where Reimbursements is equal to the proportion reimbursed multiplied by the reimbursable amount, and the reimbursable amount is equal to the amount by which the expenditures exceeds the deductible.

For example, if Premium = 2000, Proportion reimbursed = 0.9, Deductible = 200, and Expenditure = 100, then the reimbursable amount is 0, and Total cost $= 2000 + 100 - 0 = 2100$.

However, if Expenditure = 10000, then the reimbursable amount is $10000 - 200 = 9800$, and Total cost $= 2000 + 10000 - (9800) \times 0.9 = 3180$. If there are multiple plans tied for the lowest total cost, then the function can return any of them.

For example, if

```
planInfo={'BlueCross PPO':[3000,0,0.95],'Anthem HMO':[1200,100,0.8],'Cigna EPO':[2400,1000,0.98],'No Insurance':[0,0,0]}
```

Then,

- `bestPlan(planInfo,1500)` should return `'No Insurance'`.
- `bestPlan(planInfo,1700)` should return `'Anthem HMO'`.
- `bestPlan(planInfo,11000)` should return `'Anthem HMO'`.
- `bestPlan(planInfo,12000)` should return `'BlueCross PPO'`.
- `bestPlan(planInfo,13000)` should return `'Cigna EPO'`.
- `bestPlan(planInfo,100000)` should return `'Cigna EPO'`.

Apply the four steps of algorithmic thinking to solve this problem.

Step 1. Understand

Step 2. Decompose

Step 3. Analyze

Step 4. Synthesize

Exercise 4.6: Meeting Scheduling

This question asks you to create a tool to obtain the best times to schedule a meeting given the availabilities of all people interested. The tool outputs the time slot(s) that the most number of people can make.

Write a function called `getBestTimes` with one input argument:

- `availabilities`: a dictionary in which the key is the name of a person and the value is the list of all time slots that the person can make. Each time slot is represented by a string containing the date and the starting time. You can assume that each time slot has the same length so that only the start time matters. Moreover, the strings encoding the start times are formatted in a consistent way so that different strings represent different time slots, while equal strings represent the same time slot.

The function should return (not print) a dictionary where the keys are the best time slots (i.e. the slots the most number of people can make) and the value is the list of people who can make that time slot. In other words, if there are two time slots, say A and B, that 4 people can make, and for each of the remaining time slots, at most 3 people can make it, then the dictionary should contain the time slots A and B as keys and the values should be the people who can make each time slot.

Your returned dictionary must contain all of the optimal time slots. However, the order in which time slots appear in the dictionary does not matter. Moreover, the order names appear in the list of people who can make each time slot does not matter. See the sample outputs for illustrations.

```
[37]: # Sample run 1
      best=getBestTimes({'Alice':['Tue 9am','Tue 9:30am', 'Tue 2pm','Wed 11am'],\
                  'Bob':['Tue 11am','Tue 11:30am','Tue 1:30pm','Tue 2pm']})
      best

{'Tue 2pm': ['Alice', 'Bob']}

[38]: # Sample run 2
      getBestTimes({'Alice':['Tue 9am','Tue 9:30am', 'Tue 2pm','Wed 11am'],\
                  'Bob':['Tue 11am','Tue 11:30am','Tue 1:30pm','Tue 2pm', 'Wed 11am']})

{'Tue 2pm': ['Alice', 'Bob'], 'Wed 11am': ['Alice', 'Bob']}

[39]: # Sample run 3
      availabilities={}
      availabilities['Alice']=['Tue 9am','Tue 9:30am', 'Tue 2pm','Wed 11am']
      availabilities['Bob']=['Tue 11am','Tue 11:30am','Tue 1:30pm','Tue 2pm', 'Wed 11am']
      availabilities['Charlie']=['Wed 11am','Thu 11am','Fri 11am']
      availabilities['Dylan']=['Mon 9am','Mon 2pm','Tue 9am','Tue 2pm','Wed 9am']
      getBestTimes(availabilities)

{'Tue 2pm': ['Alice', 'Bob', 'Dylan'], 'Wed 11am': ['Alice', 'Bob', 'Charlie']}

[40]: # Sample run 4
      d={}
      d['Jack']=['5/12 9am','5/12 11am','5/12 12pm','5/13 2pm','5/13 3pm']
      d['Jill']=['5/12 8am','5/12 9am','5/12 10am','5/13 8am','5/13 9am', '5/13 10am']
      d['Jason']=['5/12 2pm','5/12 3pm','5/12 4pm','5/13 10am','5/13 2pm']
      d['Jenna']=['5/11 9pm','5/12 9am','5/12 3pm','5/12 9pm','5/13 12pm']
      d['Juan']=['5/12 2pm','5/12 3pm','5/13 2pm','5/13 3pm']
      best=getBestTimes(d)
      print('Best time slots:')
      for time in best:
          print(f'\t{time}\tAvailable:',best[time])

Best time slots:
        5/12 9am        Available: ['Jack', 'Jill', 'Jenna']
        5/13 2pm        Available: ['Jack', 'Jason', 'Juan']
        5/12 3pm        Available: ['Jason', 'Jenna', 'Juan']
```

Exercise 4.7: Longest Period without Rain

This question asks you to make a tool to help a roofing company plan its work given a forecast of whether it will rain in a period of consecutive days. Write a function called `analyzeForecast` with one input argument:

- `forecast`: a list of 1's and 0's indicating whether it will rain on a given day. Each 1 indicates that it will rain on that day and 0 indicates that it will not. The first entry of the list corresponds to Day 0, the second entry to Day 1, etc.

The function should return two numbers:

- `maxDryPeriod`: the longest number of consecutive days without rain in this given period.
- `startDay`: the first position in the list that begins a consecutive sequence of 0's of length `maxDryPeriod`. If `maxDryPeriod` is 0, then it doesn't matter what you return for `startDay`.

Sample run 1:

```
maxDryPeriod,startDay=analyzeForecast([1,1,0,0,0,0,1,1,0,0,1,0,0,0])
print(f'maxDryPeriod={maxDryPeriod} startDay={startDay}')
```

Correct output:

```
maxDryPeriod=4 startDay=2
```

Sample run 2:

```
maxDryPeriod,startDay=analyzeForecast([0,0,0,0,1,1,1,0,0,0,0,0,0,0])
print(f'maxDryPeriod={maxDryPeriod} startDay={startDay}')
```

Correct output:

```
maxDryPeriod=7 startDay=7
```

Sample run 3:

```
maxDryPeriod,startDay=analyzeForecast([0,1,0,0,1,1,1,0,0,0,1,0,0,0])
print(f'maxDryPeriod={maxDryPeriod} startDay={startDay}')
```

Correct output

```
maxDryPeriod=3 startDay=7
```

Instructions for Quiz 2

Quiz 2 will take place on Thursday next week, in the first fifteen minutes of class. The quiz will be open-notes but closed-computer. **You will be asked to write code that will print certain tables that will be given to you.** This is similar to what you did when solving Exercises 2.6, 2.7, 2.8, 4.2 and 4.7. Practicing this skill will sharpen your understanding of loops and speed up your development of simulation models, as many simulation models can be thought of as replicating a certain table.

As specified in the syllabus, the quiz is worth 4 percent of your final grade. If you miss the quiz, the weight will automatically be transferred to the midterm. After you are done, do not share the quiz questions or your answers with other students, including those in other sections. There will be many versions of the quiz, so don't be tempted to look at your neighbors' answers.

If you finish the quiz early, you can quietly stay in your seats or leave the room. **Do not use a cell phone, tablet or computer in the classroom before all the quizzes are handed in.**

Sample Quiz 2

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

Write a function named `process` which takes in two lists and prints a tab-delimited table, as shown in the sample outputs below. The headers and formatting should match exactly the sample outputs. Partial credits are given based on how close the solution is from being correct. Explanation of columns:

- The first column x takes values from the first list.
- The third column z takes values from the second list. (You can assume that both lists have equal lengths.)
- The columns y, a and b are equal to 0 in the first row. For subsequent rows:
  - The column y is equal to the sum of the column x and the previous row's column b.
  - The column a is 1 if the column y is greater than or equal to z, otherwise it is 0.
  - If the column a is 1, then the column b is equal to y-z, otherwise it is equal to y.

```
[47]: # Sample run 1
      process([6,8,4,6,10],[9,9,9,9,9])
```

```
x       y       z       a       b
-       0       -       0       0
6       6       9       0       6
8       14      9       1       5
4       9       9       1       0
6       6       9       0       6
10      16      9       1       7
```

```
[48]: # Sample run 2
      process([5,6,9,8,7,8,4],
      [10,10,15,15,10,10,10])
```

```
x       y       z       a       b
-       0       -       0       0
5       5       10      0       5
6       11      10      1       1
9       10      15      0       10
8       18      15      1       3
7       10      10      1       0
8       8       10      0       8
4       12      10      1       2
```