Session 5: Algorithmic Thinking II (Solutions Only)

Case 8: Demand Estimation for Substitutable Products

I. Describe:

For each customer, figure out which of the products will he/she purchase, if any. Keep track of the total number of customers purchasing each product.

II. Decompose:

A. Loop through the customers.

B. Figure out which product will a given customer purchase, if any. (Paper coding exercise)

C. Keep track of the total number of customers purchasing each product: define a variable for each product tracking the number of customers purchasing that product so far, and incrementing this by one when needed.

III. Translate:

[2]:
```
# A. Loop through...
values=[[25,15],[18,18],[30,20],[30,30]]
for curVal in values:
    v1=curVal[0]
    v2=curVal[1]
    print(v1,v2)
```

```
25 15
18 18
30 20
30 30
```

[1]:
```
# B. Figure out... (Paper coding exercise solution)
curVal=[25,15]
priceVector=[25,10]

if curVal[0]<priceVector[0] and curVal[1]<priceVector[1]:
    print('Purchase nothing')
elif curVal[0]>=priceVector[0] and curVal[1]<priceVector[1]:
    print('Purchase product 0')
elif curVal[0]<priceVector[0] and curVal[1]>=priceVector[1]:
    print('Purchase product 1')
else:
    diff0=curVal[0]-priceVector[0]
    diff1=curVal[1]-priceVector[1]
    if diff0>=diff1:
        print('Purchase product 0')
    else:
        print('Purchase product 1')
```

```
Purchase product 1
```

[2]:
```
# Alternative logic for B.
curVal=[25,15]
priceVector=[25,10]
diff0=curVal[0]-priceVector[0]
diff1=curVal[1]-priceVector[1]
if diff0<0 and diff1<0:
    print('Purchase nothing')
elif diff0>=diff1:
    print('Purchase product 0')
else:
    print('Purchase product 1')
```

```
Purchase product 1
```

[5]:
```
# C. Keep track...
count=[0,0]
count[0]+=1
count[0]+=1
count[1]+=1
print(count)
```

```
[2, 1]
```

IV. Combine

[3]:
```
# Intermediate version with print outputs and no function encapsulation
values=[[25,15],[18,18],[30,20],[30,30]]
priceVector=[25,20]
count=[0,0]
for curVal in values:
    print('Current value vector:',curVal)
    diff0=curVal[0]-priceVector[0]
    diff1=curVal[1]-priceVector[1]
    print('\tDifference of valuation and prices',diff0,diff1)
    if diff0<0 and diff1<0:
        print('\tPurchase nothing')
        continue
    elif diff0>=diff1:
        print('\tPurchase product 0')
        count[0]+=1
    else:
        print('\tPurchase product 1')
        count[1]+=1
    print('\tCount:',count)
```

```
Current value vector: [25, 15]
        Difference of valuation and prices 0 -5
        Purchase product 0
        Count: [1, 0]
Current value vector: [18, 18]
        Difference of valuation and prices -7 -2
```

```
        Purchase nothing
Current value vector: [30, 20]
        Difference of valuation and prices 5 0
        Purchase product 0
        Count: [2, 0]
Current value vector: [30, 30]
        Difference of valuation and prices 5 10
        Purchase product 1
        Count: [2, 1]
```

[7]:
```
# Final solution
def demand(priceVector, values):
    count=[0,0]
    for curVal in values:
        diff0=curVal[0]-priceVector[0]
        diff1=curVal[1]-priceVector[1]
        if diff0<0 and diff1<0:
            continue
        elif diff0>=diff1:
            count[0]+=1
        else:
            count[1]+=1
    return count

values=[[25,15],[18,18],[30,20],[30,30]]
priceVector=[25,20]
demand(priceVector,values)
```

```
[2, 1]
```

Case 9. Queuing Analysis

I: Describe

Simulate the queue dynamics according to the example table.

II: Decompose

A. Loop throught the customers.

B. Keep track of current queue length. For each row of the table, perform the following operations:

- Customers join queue: add the # of arrivals to the queue.
- Up to k customers are served: # served is minimum of k and queue length.
- Update queue: subtract the served customers from the queue.
- Add queue length at end of minute to running total.

III-IV: Translate and Combine

```
[8]: def queueLength(k,demand):
         T=len(demand)
         curQueue=0
         totalQueue=0
         for arrival in demand:
             curQueue+=arrival
             served=min(curQueue,k)
             curQueue-=served
             totalQueue+=curQueue
         return totalQueue/T
```

Having this function allows the company to run the following analysis:

```
[9]: k=3
     demand=[2,3,6,8,10,2,1,0,1,0]
     print(f'Average queue length is {queueLength(k,demand)} customers.')
```

```
Average queue length is 7.2 customers.
```

```
[10]: import numpy as np
      print(f'Average queuing time is {queueLength(k,demand)/np.average(demand):.1f} minutes.
```

```
Average queuing time is 2.2 minutes.
```

```
[11]: # Find the k needed to keep average waiting time at or below 1.5 minutes.
      demand=[2,3,6,8,10,2,1,0,1,0]
      k=1
      while (queueLength(k,demand)/np.average(demand)>1.5):
          k+=1
      print(f'Service rate needed to keep waiting time below 1.5 minutes: k={k}.')
```

```
Service rate needed to keep waiting time below 1.5 minutes: k=4.
```