Week 4: Algorithmic Thinking II

Session 8: Efficient Debugging via Code Dissection and Intermediate Printing

Debugging Demonstration

Observe carefully how the instructor debugs the following attempt of Exercise 4.2 using code dissection and intermediate printing.

tips for debugging:
① Replicate error outside of functions.
② Use code dissection for Syntax error.
③ Use 'intermediate printing' for logical errors.

```
[ ]: 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)
```

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

```
[12]: # 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))
```

Exercise 4.4 Debugging Exercise

Download the Jupyter notebook attached to the Blackboard link for this exercise and submit it there after completing it. The notebook asks you to fix all bugs in the following student attempts of session exercises from Week 3. **You should try to make the code correct using as few changes as possible, after figuring out the errors using code dissection and intermediate printing.** 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 Exercise 3.1 Python Code for Referral Marketing

```
[ ]: # 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 Exercise 3.2 Python Code for Epidemic Capacity Planning

```
[ ]: # 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))
```

```
Capacity Needed: 22
```

```
[39]: # Test code 2
      capacityNeeded([5,8,3,10,6,11,9,12,15,9,5,7])
```

```
36
```

c) Attempt for Exercise 3.3 Python Code for Demand Estimation

```
[ ]: 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
```

```
[52]: # Test code 1
      values=[[25,15],[18,18],[30,20],[30,30]]
      priceVector=[25,20]
      demand(priceVector,values)
```

```
[2, 1]
```

```
[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]
```

Sample Problem: Optimal Pricing

Write a function "optPrice" with two input arguments:

- `priceList`: a list of proposed prices.
- `valueList`: a list of numbers. Each number represents the willingness to pay for the product from a particular customer.

For a given price, the demand is equal to the number of customers with willingness to pay greater than or equal to the price. The function should iterate through the list of prices, and compute the estimated revenue for each price, which is equal to the price times the demand.

The function should return two objects: the first is the best price found. The second object is a dictionary mapping each price to the estimated revenue for that price.

Sample run:

```
priceList=range(0,36,5)
valueList=[32,10,15,18,25,40,50,43]
bestPrice,revenueDict=optPrice(priceList,valueList)
print('Best price:',bestPrice)
print('Revenue dictionary:',revenueDict)
```

Correct output:

```
Best price: 25
Revenue dictionary: {0: 0, 5: 40, 10: 80, 15: 105, 20: 100, 25: 125, 30: 120, 35: 105}
```

Exercise 4.5 Applying Algorithmic Thinking to Optimal Pricing

Download the Jupyter notebook attached to the Blackboard link for this exercise and submit it there after completing it. The notebook asks you to practice applying the four steps of algorithmic thinking to the above problem.

[60]:
```
# Graphical display
priceList=list(range(0,40,5))
valueList=[32,10,15,18,25,40,50,43]
bestPrice,revenueDict=optPrice(priceList,valueList)

import matplotlib.pyplot as plt
revenueList=[revenueDict[price] for price in priceList]
plt.plot(priceList,revenueList)
plt.plot([bestPrice],[revenueDict[bestPrice]],'ro')
plt.xlabel('Price')
plt.ylabel('Revenue')
plt.title(f'Optimal price is \${bestPrice} with revenue \${revenueDict[bestPrice]}')
plt.show()
```