Session 4: Algorithmic Thinking

1. Steps for solving a complex programming problem

I. Describe in English the task in precise language.

II. Decompose the description into well-defined components. For each component, give a step by step recipe of the logic, so that a computer can follow.

III. Translate the description of each component into runnable code, and test each component.

IV. Combine the code together into one coherent program and test the entire program.

Case 6: Optimal Pricing

Write a function `optPrice` with two input arguments:

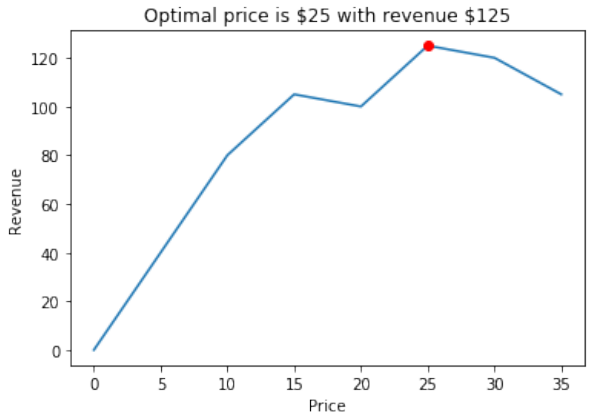
- `priceList`: a list of proposed prices.
- `valueList`: a list of numbers. Each number represents the willingness to pay for the product from a particular customer.

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 profit 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. See the following code for an example of how one would use the function.

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

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



I. Describe in English the task in precise language.

II. Decompose the description into well-defined components. For each component, give a step by step recipe that a computer can follow.

III. Translate the description of each component into runnable code, and test each component.

IV. Combine your code from the above steps, but for ease of debugging, do not package the code in the optPrice function yet and print intermediate results.

[23]:

```
Price: 0 Revenue: 0 Dictionary {0: 0}
Price: 5 Revenue: 40 Dictionary {0: 0, 5: 40}
Updated bestPrice: 5 bestRevenue: 40
Price: 10 Revenue: 80 Dictionary {0: 0, 5: 40, 10: 80}
Updated bestPrice: 10 bestRevenue: 80
Price: 15 Revenue: 105 Dictionary {0: 0, 5: 40, 10: 80, 15: 105}
Updated bestPrice: 15 bestRevenue: 105
Price: 20 Revenue: 100 Dictionary {0: 0, 5: 40, 10: 80, 15: 105, 20: 100}
Price: 25 Revenue: 125 Dictionary {0: 0, 5: 40, 10: 80, 15: 105, 20: 100, 25: 125}
Updated bestPrice: 25 bestRevenue: 125
Price: 30 Revenue: 120 Dictionary {0: 0, 5: 40, 10: 80, 15: 105, 20: 100, 25: 125, 30: 120}
Price: 35 Revenue: 105 Dictionary {0: 0, 5: 40, 10: 80, 15: 105, 20: 100, 25: 125, 30: 120, 3
Final bestPrice: 25 bestRevenue 125
```

After you have checked that everything works, remove the intermediate print statements and reorganize the code if needed to make it more readable. Enclose the final code in the optPrice function according to the prompt.

```
[26]: priceList=[0,5,10,15,20,25,30,35]
      values=[32,10,15,18,25,40,50,43]
      bestPrice,result=optPrice(priceList,values)
      print('Best price:',bestPrice)
      print('Profit for each price:',result)
```

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

Case 7. Optimal Wage Contract

Write a function `optimalContract` with two input arguments:

- `hours`: the number of hours you would like to work.
- `contracts`: a dictionary mapping the name of a contract to a list of two numbers. The first number is the hourly rate for the first 40 hours. The second number is the bonus for overtime hours, as a proportion of the hourly rate.

The function should return two objects. The first is the best possible pay under the specified number of hours worked, and the second is a list of the names of all contracts resulting in the best pay. (If one contract is better than all the rest, then the list has one element. If two or more contracts are tied for the best pay, then the list contains all of the names of the optimal contracts.)

```
    contracts={'A':[10,.8],'B':[12,0],'C':[12,.1]}
    optimalContract(38,contracts)
```

```
(456, ['B', 'C'])
```

```
    optimalContract(42,contracts)
```

```
(506.4, ['C'])
```

```
    optimalContract(60,contracts)
```

```
(760.0, ['A'])
```

I. Describe in English the task in precise language.

II. Decompose the description into well-defined components. For each component, give a step by step recipe that a computer can follow.

III. Translate the description of each component into runnable code, and test each component.

IV. Combine the code together into one coherent program and test the entire program. (First code directly in a notebook cell and print intermediate results for ease of debugging.)

[29]:

```
Processing contract A Base: 10 Bonus: 0.5 Pay: 380
        bestPay: 380 bestContracts: ['A']
Processing contract B Base: 12 Bonus: 0 Pay: 456
        bestPay: 456 bestContracts: ['B']
Processing contract C Base: 12 Bonus: 0.1 Pay: 456
        bestPay: 456 bestContracts: ['B', 'C']
```

Final Solution: