Handout for Session 5 (with Solutions)

1. Basic Syntax for Python Lists

1.1 Indexing and slicing

```
[1]: l=[10,20,'Python',int]
     len(l)
```

```
4
```

```
[2]: l[1]
```

```
20
```

```
[3]: l[-1]
```

```
int
```

```
[4]: l[0:3]
```

```
[10, 20, 'Python']
```

```
[5]: l[1:]
```

```
[20, 'Python', int]
```

```
[65]: l[4]
```

```
        ---------------------------------------------------------------------------
        IndexError                                Traceback (most recent call last)

        <ipython-input-65-32713d2680e6> in <module>
    ----> 1 l[4]


        IndexError: list index out of range
```

```
[6]: 10 in l
```

```
True
```

```
[7]: 50 in l
```

```
False
```

Q1-a: Create a list of the following objects: `30`, `4.0`, `[3,4]`, `"Hi"`.
Q1-b: Obtain the type of each element of the list.

```
[8]: t=[30,4.0,[3,4],"Hi"]
     type(t[0])
```

```
int
```

```
[9]: type(t[1])
```

```
float
```

```
[10]: type(t[2])
```

```
list
```

```
[11]: type(t[3])
```

```
str
```

Q1-c: Obtain a slice of the list from `30` to `4.0`.

```
[12]: t[0:2]
```

```
[30, 4.0]
```

Q1-d: Obtain the third element `[3,4]` in two different ways, using positive and negative indexing respectively.

```
[13]: t[2]
```

```
[3, 4]
```

```
[14]: t[-2]
```

```
[3, 4]
```

Q1-e: Find the length of the third element (which is a list as well).

```
[15]: len(t[2])
```

```
2
```

Q1-f: Check if the integer `4` is in the list. Do the same for the string `'4'`.

```
[16]: 4 in t
```

```
True
```

```
[17]: '4' in t
```

```
False
```

1.2 Modifying lists

```
[18]: l=[10,20,'Python',int]
      l+[50,60]
```

```
[10, 20, 'Python', int, 50, 60]
```

```
[19]: l
```

```
[10, 20, 'Python', int]
```

```
[20]: l=l+[50,60]
      l
```

```
[10, 20, 'Python', int, 50, 60]
```

```
[21]: l.append(40)
      l
```

```
[10, 20, 'Python', int, 50, 60, 40]
```

```
[22]: del l[4:]
      l
```

```
[10, 20, 'Python', int]
```

```
[23]: l[2]=10
      l
```

```
[10, 20, 10, int]
```

Q2-a: Define list `m=[5,4,3]`. Find two ways of adding the elements `2` and `1` to `m`, using `append` and `+` respectively.

```
[24]: m=[5,4,3]
      l.append(2)
      l.append(1)
      l
```

```
[10, 20, 10, int, 2, 1]
```

```
[25]: m=[5,4,3]
      m=m+[2,1]
      m
```

```
[5, 4, 3, 2, 1]
```

Q2-b: Modify the list so that the element 2 becomes a string `"Two"`.

```
[26]: m[3]="Two"
      m
```

```
[5, 4, 3, 'Two', 1]
```

Q2-c: Remove the last two elements from the list `m`.

```
[27]: del m[-2:]
      m
```

```
[5, 4, 3]
```

1.3 For loops and list comprehension

```
[28]: # Iterating through a list
      l=[0,2,4,6,8]
      for item in l:
          print(item)
```

```
0
2
4
6
8
```

```
[29]: # Creating a list iteratively
      l=[]
      for i in range(5):
          l.append(i*2)
      l
```

```
[0, 2, 4, 6, 8]
```

```
[30]: # List comprehension
      l=[2*i for i in range(5)]
      l
```

```
[0, 2, 4, 6, 8]
```

Q3: Write a function called `squares` with one input parameter `n`. The function should return a list of the first n squares, including 0. You should write two versions of this function, one by building from an empty list, and the other by using list comprehension.

```
[31]: def squares(n):
          l=[]
          for i in range(n):
              l.append(i*i)
          return l

      squares(5)
```

```
[0, 1, 4, 9, 16]
```

```
[32]: def squares(n):
          return [i*i for i in range(n)]

      squares(5)
```

```
[0, 1, 4, 9, 16]
```

2. Basic Syntax for Python Dictionaries

2.1 Indexing

```
[33]: d={'apple':1,'orange':2,'grape':3}
      d
```

```
{'apple': 1, 'orange': 2, 'grape': 3}

[34]: len(d)

3

[35]: d['apple']

1

[66]: d[1]

        ---------------------------------------------------------------------------
        KeyError                                  Traceback (most recent call last)

        <ipython-input-66-abe283337115> in <module>
    ----> 1 d[1]


        KeyError: 1


[36]: 'apple' in d

True

[37]: 3 in d

False

[38]: 3 in d.values()

True

[39]: d.get('apple',0)

1

[40]: d.get('rice',0)

0
```

Q4-a: Create a dictionary `name` mapping the numbers from 0 to 3 to their English name. (i.e. 0 should map to `"zero"` and 1 should map to one and so on.)

```
[41]: name={0:'zero',1:'one',2:'two',3:'three'}
      name

{0: 'zero', 1: 'one', 2: 'two', 3: 'three'}
```

Q4-b: Find the number of elements in the dictionary using `len`.

```
[42]: len(name)
```

```
4
```

Q4-c: Check if 2 is a key is in the dictionary. Check if it is a value.

```
[43]: 2 in name
```

```
True
```

```
[44]: 2 in name.values()
```

```
False
```

2.2 Modifying

```
[45]: d={'apple':1,'orange':2,'grape':3}
      d
```

```
{'apple': 1, 'orange': 2, 'grape': 3}
```

```
[46]: d['carrots']=4
      d
```

```
{'apple': 1, 'orange': 2, 'grape': 3, 'carrots': 4}
```

```
[47]: d['apple']=5
      d
```

```
{'apple': 5, 'orange': 2, 'grape': 3, 'carrots': 4}
```

```
[48]: d['apple']+=1
      d
```

```
{'apple': 6, 'orange': 2, 'grape': 3, 'carrots': 4}
```

```
[49]: d['rice']=d.get('rice',0)+1
      d
```

```
{'apple': 6, 'orange': 2, 'grape': 3, 'carrots': 4, 'rice': 1}
```

```
[50]: d['rice']=d.get('rice',0)+1
      d
```

```
{'apple': 6, 'orange': 2, 'grape': 3, 'carrots': 4, 'rice': 2}
```

Q5: Write a function `histogram` with one input argument `values`, which is a list. The function should return a dictionary mapping each unique element in the list to the number of occurences. (Hint: use a `for` loop to go through each `item` in the list and add 1 to the corresponding dictionary value each time.)

```
[51]: def histogram(values):
          ans={}
          for item in values:
              ans[item]=ans.get(item,0)+1
          return ans
      histogram([3,2,1,2,2,2])
```

```
{3: 1, 2: 4, 1: 1}
```

2.3 For loops and dictionary comprehension

```
[52]: # Iterating through a dictionary
      d={'apple':5,'rice':4,'broccoli':8}
      for key in d:
          value=d[key]
          print(key,value)
```

```
apple 5
rice 4
broccoli 8
```

```
[53]: # Building a dictionary iteratively
      l=['apple','rice','broccoli']
      d={}
      for item in l:
          d[item]=len(item)
      d
```

```
{'apple': 5, 'rice': 4, 'broccoli': 8}
```

```
[54]: # Dictionary comprehension
      l=['apple','rice','broccoli']
      d={item:len(item) for item in l}
      d
```

```
{'apple': 5, 'rice': 4, 'broccoli': 8}
```

Q6: Modify the function `squares` so that instead of returning a list, return a dictionary mapping each number between 0 and $n-1$ to the number squared. You should write two versions of this function, one by building from an empty dictionary, and the other by using dictionary comprehension.

```
[55]: def squares(n):
          d={}
          for i in range(n):
              d[i]=i*i
          return d

      squares(5)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

```
[56]: def squares(n):
          return {i:i*i for i in range(n)}

      squares(5)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Case 7a: Estimating Demand for a Given Price

Write a function `demand` with two input arguments:

- `price`: a proposed price for a certain product.
- `values`: a list of numbers. Each number represents the willingness to pay for the product from a particular customer.

The function should return the number of consumers willing to buy the product at the given price. (Hint: count the number of values greater than or equal to `price`.)

```
[57]: def demand(price,values):
          count=0
          for value in values:
              if value>=price:
                  count+=1
          return count
```

```
[58]: price=20
      values=[32,10,15,18,25,40,50,43]
      demand(price,values)
```

```
5
```

Digression: Multiple return values

```
[59]: def f(a,b):
          return a+b,a*b

      s,p=f(5,3)
      print('sum is',s)
      print('product is',p)
```

```
sum is 8
product is 15
```

Case 7b: Optimal Pricing

Write a function `optPrice` with two input arguments:

- `priceList`: a list of proposed prices.
- `values`: as in 7a.

The function should iterate through the list of prices, and compute the estimated profit for each price (which equals to the estimated demand from Case 7a multiplied by the price).

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 profit for that price.

```
[60]: def optPrice(priceList,values):
          bestProfit=0
          bestPrice=0
          dic={}
          for price in priceList:
```

```
            profit=demand(price,values)*price
            dic[price]=profit
            if profit>bestProfit:
                bestProfit=profit
                bestPrice=price
        return bestPrice, dic
```

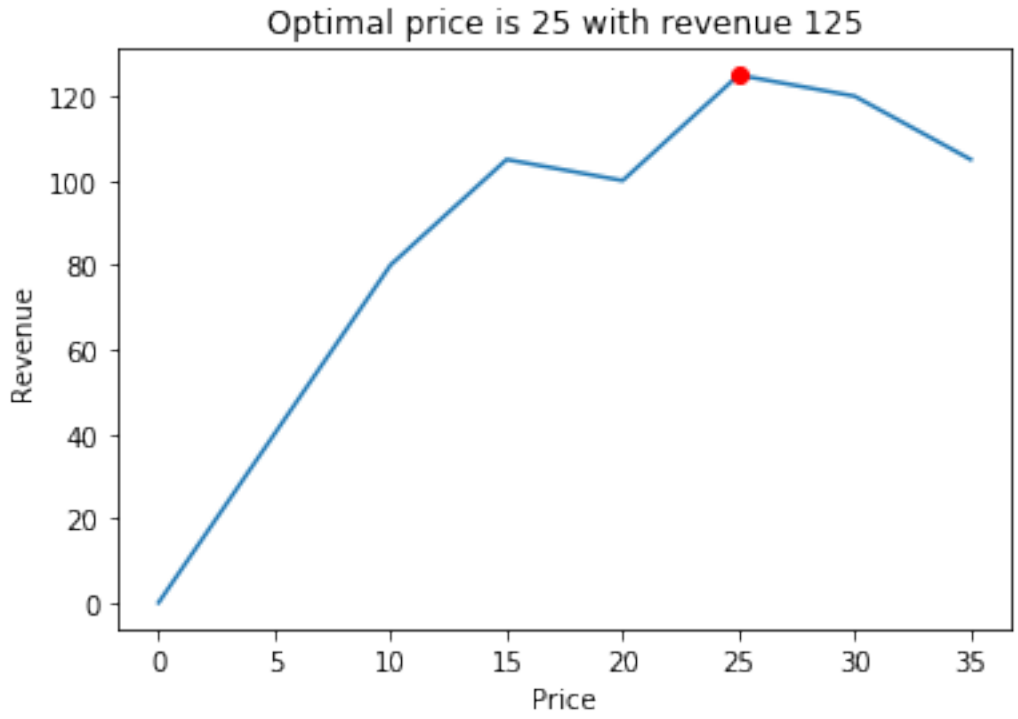
[61]:
```
priceList=list(range(0,40,5))
bestPrice,profit=optPrice(priceList,values)
print('Best price is',bestPrice)
profit
```

```
Best price is 25
```

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

After you successfully implement the function, you should be able to use the following code to graph the output.

[64]:
```
import matplotlib.pyplot as plt
bestPrice,revenue=optPrice(priceList,values)
priceList=sorted(priceList)
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('Optimal price is '+str(bestPrice)+' with revenue '+str(round(revenue[bestPri
plt.show()
```



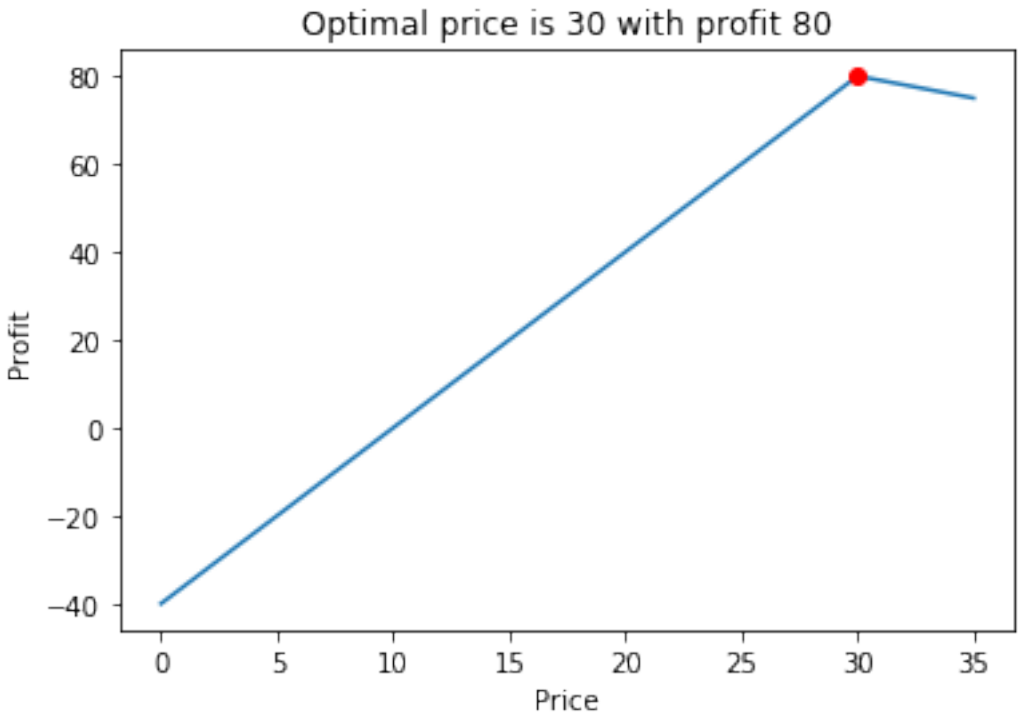
(Optional): Modify Case 7b) so that `optPrice` maximizes profit instead of revenue and takes into account production capacity. The function should take in two additional parameters:

- `cost` (default 0): the production cost for one unit of the product.
- `capacity` (default `1e100`): the maximum number of units that can be sold.

The function should evaluate profit as $(price - cost) \times \min(demand, capacity)$, and return two objects. The first one being the optimal price found, and the second one being a dictionary mapping each price to each profit.

[63]:
```
def optPrice(priceList,values,cost=0,capacity=1e100):
    bestProfit=0
    bestPrice=0
    profit={}
    for price in priceList:
        profit[price]=min(capacity,demand(price,values))*(price-cost)
        if profit[price]>bestProfit:
            bestProfit=profit[price]
            bestPrice=price
    return bestPrice,profit

import matplotlib.pyplot as plt
bestPrice,profit=optPrice(priceList,values,cost=10,capacity=4)
priceList.sort()
revenueList=[profit[price] for price in priceList]
plt.plot(priceList,revenueList)
plt.plot([bestPrice],[profit[bestPrice]],'ro')
plt.xlabel('Price')
plt.ylabel('Profit')
plt.title('Optimal price is '+str(bestPrice)+' with profit '+str(round(profit[bestPrice
plt.show()
```


Optimal price is 30 with profit 80
Profit
Price
80
60
40
20
0
−20
−40
0
5
10
15
20
25
30
35