# Session 4 Handout

## 1. Using `while` loops

The following code assumes that you have a correct `orderQuantity` function from last session in a `session3.py` file in the current directory.

```
[1]: from session3 import orderQuantity
     while True:
         userInput=input('Enter inventory (or done): ')
         if userInput=='done':
             break
         elif userInput=='skip':
             continue
         inventory=int(userInput)
         print ('Order',orderQuantity(inventory),'units.')
```

```
Enter inventory (or done): skip
Enter inventory (or done): 30
Order 70 units.
Enter inventory (or done): 25
Order 75 units.
Enter inventory (or done): done
```

Alternative implementation without using `break` or `continue`.

```
[ ]: userInput=input('Enter inventory (or done): ')
     while userInput!='done':
         if userInput!='skip':
             inventory=int(userInput)
             print ('Order',orderQuantity(inventory),'units.')
         userInput=input('Enter inventory (or done): ')
```

**Q1:** Write a program to repeatedly ask the user to input the number of hours worked, and display the total pay, assuming that the rate for first 40 hours is 10/hour, and the rate for additional hours is 15/hour. The program should terminate whenever the user inputs `done`.

```
[2]:
```

```
Enter hours worked (or done): 38
Pay is 380.0
Enter hours worked (or done): 42
Pay is 430.0
Enter hours worked (or done): done
```

**(optional) Q2**: Rewrite the code in Q1 but do not use `break`.

The following function uses `try` and `except` (see PY4E Chapter 3) for checking whether a certain value is convertable to a `float`.

```
[3]: def isNumber(x):
         try:
            float(x)
            return True
        except:
            return False

    print(isNumber(3))
    print(isNumber('3'))
    print(isNumber('three'))
```

```
True
True
False
```

**Q3:** Modify the first example of this handout so that if the user does not input `done` nor an integer, then the program prints `Invalid input.` and asks for another input. (Hint: first write an `isInteger(x)` function by modifying the above, then use an `if` statement to decide whether to convert the input to an integer, or display `Invalid input.`)

[4]:

```
Enter inventory (or done): 2.5
Invalid input.
Enter inventory (or done): 30
Order 70 units.
Enter inventory (or done): thirty
Invalid input.
Enter inventory (or done): done
```

## 2. Using `for` loops

```
[5]: for i in [0,3,5,2]:
         print(i,end=' ')
```

```
0 3 5 2
```

```
[6]: for i in range(5):
         print(i,end=' ')
```

```
0 1 2 3 4
```

**Q4:** Modify the first example of the handout to use a `for` loop instead of a `while` loop, and limit the number of iterations to at most 5.

The following example illustrates reading and writing to a file using `for` loops. The generated file will be used as an input to case 8c.

### 2.1 Using `for` loops to read files

Type the following code example in your Jupyter notebook, as it will create a data file that we will use later.

```
[7]: import random
     file=open('session4_data.txt','w')
     random.seed(0)
     for t in range(10):
         value=random.randint(10,60)
         print(value,file=file)
     file.close()
```

```
[8]: file=open('session4_data.txt','r')
     for line in file:
         print(int(line),end=' ')
     file.close()
```

```
34 58 36 12 26 42 41 35 60 29
```

## 2.2 Computations Using Loops

```
[9]: l=[0,3,5,2,-2]
     total=0
     largest=-1e100
     smallest=1e100
     for num in l:
         total=total+num
         if num>largest:
             largest=num
         if num<smallest:
             smallest=num
     print('Total:',total)
     print('Average:',total/len(l))
     print('Largest:',largest)
     print('Smallest:',smallest)
```

```
Total: 8
Average: 1.6
Largest: 5
Smallest: -2
```

**Q5:** Modify the above code to apply to the numbers in the file `session4_data.txt` created by the previous code example.

```
[10]:
```

```
Total: 373.0
Average: 74.6
Largest: 60.0
Smallest: 12.0
```

## Case 7: Batch Automation of Payroll Computations

Write a function `calculateAllWages` which takes three input arguments:

- `filename` (default `'session4_data.txt'`): the name of a file in which each line corresponds to the hours worked by an employee.
- `base` (default 10.0): the base hourly rate for hours below 40.
- `bonus` (default 0.5): each hour above 40 obtains a pay of `base*(1+bonus)`.

and display the pay for each employee. (You may use the `calculateWage` function from before.) At the end, the function should display the total and average pay. See the sample output below.

```
[12]: calculateAllWages()
```

```
Employee #1 worked 34.0 hours. Pay: 340.0
Employee #2 worked 58.0 hours. Pay: 670.0
Employee #3 worked 36.0 hours. Pay: 360.0
Employee #4 worked 12.0 hours. Pay: 120.0
Employee #5 worked 26.0 hours. Pay: 260.0
Employee #6 worked 42.0 hours. Pay: 430.0
Employee #7 worked 41.0 hours. Pay: 415.0
Employee #8 worked 35.0 hours. Pay: 350.0
Employee #9 worked 60.0 hours. Pay: 700.0
Employee #10 worked 29.0 hours. Pay: 290.0
Total pay is 3935.0
Average pay is 393.5
```

## Case 8a: Estimating Demand for a Given Price

Write a function `demand` with two input arguments:

- `price`: a proposed price for a certain product.
- `valueFile` (default `session4_data.txt`): a file in which each line contains one number, representing the willingness to pay for the product from a customer (according to a marketing survey).

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

```
[15]: price=20
      demand(price)
```

```
9
```

## Case 8b: Optimal Pricing

Write a function `optPrice` with two input arguments:

- `priceList`: a list of proposed prices.
- `valueFile` (default `session4_data.txt`) as in Case 8a.

The function should iterate through the list of prices, and display the estimated revenue for each price (which equals to the estimated demand from Case 8a multiplied by the price). The function should print a final line giving the best price in the list and the best possible revenue, and return the best price.

```
[17]: priceList=[0,5,10,15,20,25,30,35]
      optPrice(priceList)
```

```
Revenue with price 0 is 0
Revenue with price 5 is 50
Revenue with price 10 is 100
Revenue with price 15 is 135
Revenue with price 20 is 180
Revenue with price 25 is 225
Revenue with price 30 is 210
Revenue with price 35 is 210
Best price is 25 with revenue 225


25
```

**(Optional) Q6:** Modify Case 8 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 the price that maximizes profit.

**(Optional) Q7:** Modify Case 7 and Case 8 so that if the input file has lines that cannot be converted to numbers, then those lines are skipped without the program crashing.

**(Optional) Q8:** Without running the following code, predict its output:

```
l=[6,1,-3,8,3,3]
n=len(l)
for i in range(n):
    for j in range(i+1,n):
        if l[i]>l[j]:
            tmp=l[i]
            l[i]=l[j]
            l[j]=tmp
```