# Contents



# Exercises for Week 2: Mastering Loops

## Grading Scheme:

Important: This Jupyter notebook needs to be completed and submitted via Blackboard before the due date to receive a non-zero grade.

- 5: Every question is completed and the code returns the correct outputs on all of the sample inputs.
- 4: Almost complete, but certain questions are blank, returns clearly incorrect outputs, or the code there does not run at all.
- 3: This score will not be assigned, as everyone should strive to get 4 or 5.
- 2: Not close to complete, but at least 50% complete.
- 1: At least 10% complete, but less than 50% complete
- 0: Less than 10% complete, or response is identical to someone else's, indicating plagiarism.

A perfect score is 5. Note that your code does not need to be absolutely perfect to receive a 5, but you need to complete every question and ensure that the outputs are correct on all of the sample runs included here. **To ensure that you get 5 out of 5, before you submit, restart the Kernel and run all, and check that all of the outputs are as intended.**

These exercises are intended to be completed in 5-7 hours, including class time. You should budget at least this much time before the due date.

## Name: XXX

## Exercise 2.1: Practicing For-Loops

**a)** Write a for loop which takes in the list of names `['Alice','Bob','Charles']`, and print the number of characters in each name, as below.

**Hint:** you can use the `len` function to find the length of a string, as in

```
len('Alice')
```

```
[14]: # Sample output
```

```
The name Alice has length 5.
The name Bob has length 3.
The name Charles has length 7.
```

```
[ ]: # Write your code here
```

**b)** Write a for loop to print the multiplication table for multiplying by 2, as in the sample output below.

```
[15]: # Sample output
```

```
1 x 2 = 2
2 x 2 = 4
```

```
3 x 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
9 x 2 = 18
```

```
[ ]: # Write your code here
```

**c)** Write one line of code using `range` which generates a `list` object containing odd numbers from 1 to 29 (inclusive).

```
[16]: # Sample output
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
```

```
[ ]: # Write your code here
```

**d)** Print a line that lists the even numbers from 10 to 0 backward, as below.

```
[17]: # Sample output
```

```
10 8 6 4 2 0
```

```
[ ]: # Write your code here
```

**e)** Write a function `squares` with one input parameter `n` (assumed to be a positive integer). The function should print the squares of the first `n` positive integers, as shown below.

```
[18]: # Write your function here
```

```
[19]: # Test code 1
      squares(3)
```

```
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
```

```
[20]: # Test code 2
      squares(5)
```

```
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
4 x 4 = 16
5 x 5 = 25
```

## Exercise 2.2: Practicing List-Comprehension

Execute the following cell to load in the list named "sales":

```
[29]: sales=[10,20,5,25,30,12,18,50,30,20,10,4]
```

**a)** Using list comprehension, write one line to generate the list of elements in "sales" that are at least 20.

```
[30]: # Sample output
```

```
[20, 25, 30, 50, 30, 20]
```

```
[ ]: # Write your code here
```

**b)** Write one line to count the number of elements in "sales" that are at least 20. Hint: find the length of the above list.

[31]: *# Sample output*

```
6
```

[ ]: *# Write your code here*

**c)** Write one line to count the number of elements that are between 5 and 10 (inclusive).

[32]: *# Sample output*

```
3
```

[ ]: *# Write your code here*

**d)** Generate a list of zeros with the same length as the list `sales`.

[33]: *# Sample output*

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[ ]: *# Write your code here*

**e)** Use dictionary comprehension to write one line that generates the following dictionary. The values are from the list `sales` and the keys are the strings "Week 1", "Week 2", etc.

[35]: *# Sample output*

```
{'Week 1': 10,
 'Week 2': 20,
 'Week 3': 5,
 'Week 4': 25,
 'Week 5': 30,
 'Week 6': 12,
 'Week 7': 18,
 'Week 8': 50,
 'Week 9': 30,
 'Week 10': 20,
 'Week 11': 10,
 'Week 12': 4}
```

[ ]: *# Write your code here*

**f)** Suppose that the price of the product in each week is given by the following list. Use dictionary comprehension to write one additional line to yield the dictionary mapping the week name to the revenue each week. (Revenue in each week is equal to the sales multiplied by the price.)

```
price=[5,5,10,6,5,5,6,8,8,8,5,10]
```

[36]: *# Sample output*

```
{'Week 1': 50,
 'Week 2': 100,
 'Week 3': 50,
 'Week 4': 150,
 'Week 5': 150,
 'Week 6': 60,
 'Week 7': 108,
 'Week 8': 400,
 'Week 9': 240,
 'Week 10': 160,
 'Week 11': 50,
 'Week 12': 40}
```

```
[ ]: price=[5,5,10,6,5,5,6,8,8,8,5,10]
     # Write your code here
```

## Exercise 2.3: Payroll Calculations

In a certain company, the part-time employees receive a wage based on the number of hours logged each week. Write a function called "payroll" with two input arguments:

- hours: a list of non-negative numbers corresponding to the number of hours an employee logged in successive weeks.
- rate: the hourly wage for the employee.

However, a part-time employee can only be paid for at most 20 hours in any given week, so if the employee logs more than 20 hours in a week, he/she will only be paid for 20 hours.

The program should print (not return) a statement summarizing the total wage as well as the average wage per week.

**Sample run 1:**

```
payroll([5,3,2,4],10)
```

**Sample output 1:**

```
The total wage is $140. Average is $35.0 per week.
```

**Sample run 2:**

```
payroll([25,15,14,30,0],12.5)
```

**Sample output 2:**

```
The total wage is $862.5. Average is $172.5 per week.
```

```
[42]: # Write your function here
```

```
[43]: # Sample run 1
      payroll([5,3,2,4],10)
```

```
The total wage is $140. Average is $35.0 per week.
```

```
[44]: # Sample run 2
      payroll([25,15,14,30,0],12.5)
```

```
The total wage is $862.5. Average is $172.5 per week.
```

## Exercise 2.4: Sentence Analysis

Write a function called "analyzeSentence" with one input argument:

- sentence: a string of words separated by spaces, but without punctuations.

The function should print (not return) a statement with the word count, the average length of the word (rounded to 2 decimal places), and the maximum length of words.

**Hint:** You can use the str.split function, which splits a given string into words by spaces, as illustrated below.

```
'I love programming'.split()
```

Would return the list `['I','love','programming']`.

**Sample run 1:**

```
analyzeSentence('I love programming in Python')
```

**Sample output 1:**

```
This sentence has 5 words.
Average word length: 4.8
Maximum word length: 11
```

**Sample run 2:**

```
analyzeSentence('The quick brown fox jumps over the lazy dog')
```

**Sample output 2:**

```
This sentence has 9 words.
Average word length: 3.89
Maximum word length: 5
```

```
[45]: # Write your function here
```

```
[46]: # Sample run 1
      analyzeSentence('I love programming in Python')
```

```
This sentence has 5 words.
Average word length: 4.8
Maximum word length: 11
```

```
[47]: # Sample run 2
      analyzeSentence('The quick brown fox jumps over the lazy dog')
```

```
This sentence has 9 words.
Average word length: 3.89
Maximum word length: 5
```

## Exercise 2.5: Practicing Break and Continue

**a)** Write a function called `firstOutbreak` with two input arguments:

- **cases**: a list of positive integers corresponding to the number of cases of a certain disease each week. The weeks are labelled starting from 0.
- **threshold:** a positive integer denoting what is an outbreak.

The function should print (not return) one sentence that specifies the first week in which the number of cases is at least equal to the threshold, along with the number of cases that week. If the number of cases never reaches the threshold, the program should print an alternative statement, as in the sample outputs below.

```
[51]: # Write your function here
```

```
[52]: firstOutbreak([10,20,30,15,50],25)
```

```
The first outbreak occured in week 2 with 30 cases
```

```
[53]: firstOutbreak([10,20,30,15,50],51)
```

```
There is no outbreak in the given data.
```

**b)** Write a program that asks the user to type a sentence (in lower case without punctuations) and print the same sentence but removing all occurrences of the following words: “a”, “the”, “of”, “in”, “and”, “i”, “you”, “he”, “she”.

Hint: you can split a sentence into a list of words using

```
sentence.split()
```

assuming that the object `sentence` is a string. You can also print words one by one and separate them by spaces using `print(..., end=' ')`.

```
[54]: # Sample output

Please enter a sentence (in lower case without punctuations):  i love programming in␣
↪python and r

love programming python r

[ ]: # Write your code here
```

## Exercise 2.6: Estimating Time of Selling Out

Suppose that there are limited number of tickets to an event and your goal is to estimate when the tickets will be sold out.

Write a function `sellOutTime` with two input arguments:

- `demand`: a list of positive integers, representing the forecasted demand in each week. Weeks are numbered from 1 (rather than from 0).
- `inventory`: an integer denoting the initial supply of tickets.

The function should return the week in which tickets will sell out. If the tickets never sell out in the given weeks, then the function should return the week after the given horizon.

For example, if `demand=[50,80,89]` (representing a demand of 50 in week 1, 80 in week 2, and 90 in week 3), then

- if initial inventory is 10, the tickets will sell out in week 1. (Function returns 1.)
- if initial inventory is 50, the function should still return 1.
- if initial inventory is 60, function should return 2. (Because supply lasts into week 2.)
- if initial inventory is 300, function should return 4. (Supply lasts even after the 3 weeks are over.)

The following table illustrates what is going on in each iteration of the loop when initial inventory is 500.

```
[55]: # Table

Week    Demand  Remaining Inventory
0               500
1       50      450
2       80      370
3       89      281
4       100     181
5       120     61
6       140     -79

[ ]: # Try to replicate the above table here without function encapsulation

[56]: # Write your function here
```

The following code illustrates the use of the function after you complete it.

```
[57]: demand=[50,80,89,100,120,140,100,80]
      for inventory in [10,50,60,130,220,500,1000]:
          print('With initial inventory',inventory,'supply will last until␣
↪week',sellOutTime(demand,inventory))

With initial inventory 10 supply will last until week 1
With initial inventory 50 supply will last until week 1
With initial inventory 60 supply will last until week 2
With initial inventory 130 supply will last until week 2
With initial inventory 220 supply will last until week 4
With initial inventory 500 supply will last until week 6
With initial inventory 1000 supply will last until week 9
```

## Exercise 2.7: Queue Analysis

A popular fast food restaurant is planning to open a branch in a new location and wants to decide how many servers to hire. To help them, write a function "avQueueLength" with two input parameters:

- k: the number of customers that can be served in a minute. (assumed to be integer)
- demand: a list of integers specifying how many customers arrive each minute. (For simplicity, assume that customers arrive at the beginning of each minute and up to k customers can be served instantly.)

The function should return (not print) the average queue length.

**Sample run:**

```
avQueueLength(3,[2,3,6,8,10,2,1,0,1,0])
```

**Output:**

```
7.2
```

The following table summarizes the evolution of the queue corresponding to the above sample input.

| Minute | # of Arrivals | # Served | Queue Length |
|---|---|---|---|
| 0 | – | – | 0 |
| 1 | 2 | 2 | 0 |
| 2 | 3 | 3 | 0 |
| 3 | 6 | 3 | 3 |
| 4 | 8 | 3 | 8 |
| 5 | 10 | 3 | 15 |
| 6 | 2 | 3 | 14 |
| 7 | 1 | 3 | 12 |
| 8 | 0 | 3 | 9 |
| 9 | 1 | 3 | 7 |
| 10 | 0 | 3 | 4 |
| **Average** | – | – | **7.2** |

**When you code, you should first write your code without a function and print out the above table, to ensure that your logic is correct. At the end, embed your code in a function as the problem specifies.**

```
[60]: # Sample output for the table

Minute  Arrival Served  Queue
1       2       2       0
2       3       3       0
3       6       3       3
4       8       3       8
5       10      3       15
6       2       3       14
7       1       3       12
8       0       3       9
9       1       3       7
10      0       3       4
Average Queue Length: 7.2
```

```
[ ]: # Version of the code without function encapsulation
```

The final code is below. Notice that when we don't need to print the table, we don't need to keep track of the variable "minute".

[61]:
```
# Final code
```

[62]:
```
# Test code 1
avQueueLength(3,[2,3,6,8,10,2,1,0,1,0])
```

```
7.2
```

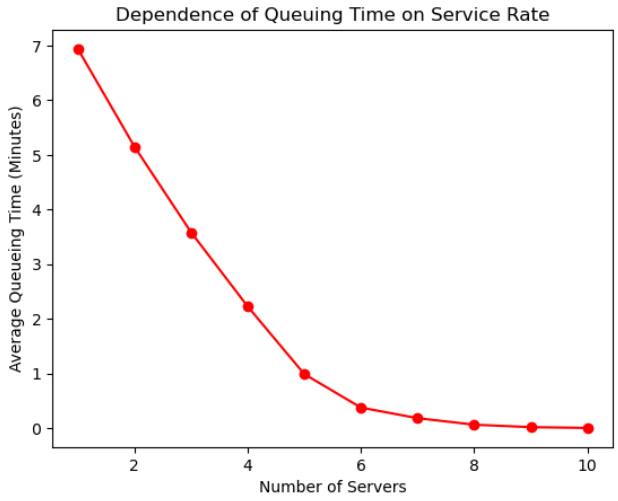
[63]:
```
# Test code 2
demand=[2,3,6,8,10,2,1,0,1,0]
for k in range(1,6):
    print(f'Av. Queue Length with {k} server(s):',avQueueLength(k,demand))
```

```
Av. Queue Length with 1 server(s): 17.2
Av. Queue Length with 2 server(s): 11.7
Av. Queue Length with 3 server(s): 7.2
Av. Queue Length with 4 server(s): 4.0
Av. Queue Length with 5 server(s): 2.2
```

To illustrate why the above function is useful, according to a mathematical result known as Little's Law, the average queuing time of customers is

$$\frac{\text{Average Queue Length}}{\text{Average Arrival Rate}} = \frac{7.2}{3.3} \approx 2.2 \text{ minutes.}$$

[64]:
```
# Test code 3
demand=[2,3,6,8,10,8,4,3,2,6,3,2,1,4,5]
servers=range(1,11)
avArrivalRate=sum(demand)/len(demand)
queuingTime=[avQueueLength(k,demand)/avArrivalRate for k in servers]
import matplotlib.pyplot as plt
plt.plot(servers,queuingTime,'ro-')
plt.xlabel('Number of Servers')
plt.ylabel('Average Queueing Time (Minutes)')
plt.title('Dependence of Queuing Time on Service Rate')
plt.show()
```




## Exercise 2.8: Investment Accounting

This question asks you to create a tool to perform simple accounting for stock trading. Write a function called "accounting" with two input arguments:

- prices: a list of positive numbers corresponding to the price of a stock in successive days.
- changes: a list of integers (positive or negative) corresponding to the change in the number of shares carried. A positive number corresponds to buying shares of the stock and a negative number corresponds to selling. It is possible for you to own net negative shares of the stock.

You may assume that the two lists are of the same length. The function should return (not print) the following two numbers:

- net change in shares: the sum of the numbers in the list "changes".
- net change in cash: the net money spent or earned over the trades in the list "changes". Buying the stock costs money and selling it earns money.

For example, if `prices=[10,12,13,8,7,15]` and `changes=[3,2,-5,3,1,-5]`, the following table illustrates the calculations.

| Price | Change | Cashflow |
|---|---|---|
| 10 | +3 | -30 |
| 12 | +2 | -24 |
| 13 | -5 | 65 |
| 8 | +3 | -24 |
| 7 | +1 | -7 |
| 15 | -5 | 75 |
| **Net** | **-1** | **55** |

**When you code, first write your code without a function and print out the above table. Later, embed your code in a function as the problem specifies.**

**Sample run:**

```
netShares,netCash=accounting([10,12,13,8,7,15],[3,2,-5,3,1,-5])
print(f'Net change in position: {netShares} shares.')
print(f'Net change in cash: {netCash} dollars.')
```

**Sample output:**

```
Net change in position: -1 shares.
Net change in cash: 55 dollars.
```

```
[65]: # Sample output for the table

Price   Change  Cashflow
10      3       -30
12      2       -24
13      -5      65
8       3       -24
7       1       -7
15      -5      75
Net change in position: -1 shares.
Net change in cash: 55 dollars.
```

```
[ ]: # Version of code that prints the table
```

```
[66]: # Final code
```

```
[67]: # Sample run
      netShares,netCash=accounting([10,12,13,8,7,15],[3,2,-5,3,1,-5])
      print(f'Net change in position: {netShares} shares.')
      print(f'Net change in cash: {netCash} dollars.')
```

```
Net change in position: -1 shares.
Net change in cash: 55 dollars.
```

## Exercise 2.9: Appointment Booking

This question asks you to create a tool to help administrators schedule appointments over the phone. Assume that every appointment is for a half-hour slot and that the administrator would like to obtain the next $k$ available slots, where $k$ is a given parameter.

**Write a function called "firstAvailable" with three input arguments:**

- `slots`: a list of strings representing names for each slot, in order from earliest to latest.
- `availability`: a list of True/False with the same length as the above. True corresponds to a slot being available and False corresponds to it being unavailable. The order of entries is the same as in the previous list.
- `k`: a positive integer giving the number of available slots to obtain.

The function should print (not return) the names of next $k$ open slots, counting from earliest to latest (according to the order of the lists). If the total number of available slots is less than $k$, then the function should print all available slots. If there are no available slots, then the function should print that there are no slots. See the sample outputs below.

```
[68]: # Write your function here.
```

```
[2]: # Sample run 1
     slots=['Mon 9am','Mon 9:30am','Mon 10am','Mon 10:30am','Tue 2pm','Tue 2:30pm','Tue␣
 ↪3pm','Tue 3:30pm']
     availability=[False,False,True,False,True,False,False,False]
     firstAvailable(slots,availability,1)
```

```
Available slot 1: Mon 10am
```

```
[3]: # Sample run 2
     slots=['Mon 9am','Mon 9:30am','Mon 10am','Mon 10:30am','Tue 2pm','Tue 2:30pm','Tue␣
 ↪3pm','Tue 3:30pm']
     availability=[False,False,True,False,True,False,False,False]
     firstAvailable(slots,availability,2)
```

```
Available slot 1: Mon 10am
Available slot 2: Tue 2pm
```

```
[4]: # Sample run 3
     slots=['Mon 9am','Mon 9:30am','Mon 10am','Mon 10:30am','Tue 2pm','Tue 2:30pm','Tue␣
 ↪3pm','Tue 3:30pm']
     availability=[False,False,True,False,True,False,False,False]
     firstAvailable(slots,availability,3)
```

```
Available slot 1: Mon 10am
Available slot 2: Tue 2pm
```

```
[5]: # Sample run 4
     slots=['Mon 9am','Mon 9:30am','Mon 10am','Mon 10:30am','Tue 2pm','Tue 2:30pm','Tue␣
 ↪3pm','Tue 3:30pm']
     availability=[False,False,False,False,False,False,False,False]
     firstAvailable(slots,availability,4)
```

```
No slots available!
```