Contents

Week 2: Mastering Loops

Loops often cause confusion for those learning programming for the first time. By the end of this week, you will be able to trace through the logic of loops step by step, as well as to write loops to implement simple simulations. Mastering the logic of loops will greatly enhance your ability as a business analyst and equip you to create more complex simulation models in coming weeks.

Session 3: Understanding Loops

2.1 Review of For-Loops

Basic Syntax

For-loops are used to iterate through a given collection of items in order.

```
[1]: for i in [5,4,3,2,1]:
         print('Count down:',i)
     print('Take off!')
```

```
Count down: 5
Count down: 4
Count down: 3
Count down: 2
Count down: 1
Take off!
```

```
[2]: for name in ['Alice','Bob','Charles']:
         print(f'Hi {name}!')
```

```
Hi Alice!
Hi Bob!
Hi Charles!
```

Iterating through Dictionaries, Series, or DataFrame

In the above example, the collection of items is a list. For-loops can also iterate through dictionaries, Series, and DataFrames, as follows.

```
[3]: d={'Apple':3, 'Banana':8, 'Orange':2}
     for key in d:
         value=d[key]
         print(f'{key} -> {value}')
```

```
Apple -> 3
Banana -> 8
Orange -> 2
```

[4]:
```
import pandas as pd
s = pd.Series(d)
# Iterating through the values
for v in s:
    print(v)
```

```
3
8
2
```

[5]:
```
# Iterating through the index
for key in s.index:
    value=s[key]
    print(f'{key} -> {value}')
```

```
Apple -> 3
Banana -> 8
Orange -> 2
```

[6]:
```
df=pd.DataFrame(index=s.index)
df['Value']=s
df['Price']=[2,1,3]
df
```

	Value	Price
Apple	3	2
Banana	8	1
Orange	2	3

[7]:
```
for fruit in df.index:
    value=df.loc[fruit,'Value']
    price=df.loc[fruit,'Price']
    print(f'{fruit}: value={value} price={price}')
```

```
Apple: value=3 price=2
Banana: value=8 price=1
Orange: value=2 price=3
```

Using Range

Instead of explicitly giving a collection of items, you can also specific a range, as follows.

[8]:
```
list(range(5,0,-1))
```

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

[9]:
```
for i in range(5,0,-1):
    print('Count down:',i)
print('Take off!')
```

```
Count down: 5
Count down: 4
Count down: 3
Count down: 2
Count down: 1
Take off!
```

[10]:
```
list(range(1,5))
```

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

[11]:
```
list(range(5))
```

```
[0, 1, 2, 3, 4]
```

[12]:
```
for i in range(5):
    print('Repeating five times.')
```

```
Repeating five times.
Repeating five times.
Repeating five times.
Repeating five times.
Repeating five times.
```

[13]:
```
# Listing the even numbers from 0 to 10.
for i in range(0,11,2):
    print(i,end=' ')
```

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

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]:

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

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

[15]:

```
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
```

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

[16]:

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

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

[17]:

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

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.

[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
```

2.2 Review of List-Comprehension

List-comprehension is a shorthand notation for generating a list. The syntax looks like for-loops but technically speaking they are distinct concepts.

[21]:
```
# Initializing a list with zeros.
l=[0 for i in range(10)]
l
```

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

[22]:
```
l[0]=3
l[1]=5
l
```

```
[3, 5, 0, 0, 0, 0, 0, 0, 0, 0]
```

[23]:
```
# Quickly generating a list of squares
[num*num for num in range(1,6)]
```

```
[1, 4, 9, 16, 25]
```

[24]:
```
# Adding conditions
[num*num for num in range(1,6) if num!=3]
```

```
[1, 4, 16, 25]
```

[25]:
```
sum([num*num for num in range(1,6)])
```

```
55
```

[26]:
```
# Dictionary comprehension
sales={'USA':3000, 'China':2000, 'India':4000}
revenue={country:sales[country]*5 for country in sales}
revenue
```

```
{'USA': 15000, 'China': 10000, 'India': 20000}
```

[27]:
```
{country:len(country) for country in sales}
```

```
{'USA': 3, 'China': 5, 'India': 5}
```

[28]:
```
{country:len(country) for country in sales if len(country)==5}
```

```
{'China': 5, 'India': 5}
```

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.

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.

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

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

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.

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]
```

2.3 Tracing Loop Logic using a Table

```
[37]: # Example 1
      sales=[10, 25, 5, 15, 20]
      total=0
      for value in sales:
          total+=value    # shorthand for total=total+value
      total
```

```
75
```

In-class Exercise: Tracing Code

Do the following on paper WITHOUT a computer. (Use a table to "walk through" the following code line by line.)

a) What is the final value of `total`?

```
[38]: # Example 2
      sales=[10, 25, 5, 15, 20]
      total=0
      for value in sales:
          if value>=20:
              total+=value
```

b) What are the final values of `maxValue` and `maxIndex`?

```
[40]: # Example 3
      sales=[10, 25, 5, 15, 20]
      maxValue=-1
      maxIndex=-1
      for i in range(len(sales)):
          value=sales[i]
          if value>maxValue:
              maxValue=value
              maxIndex=i
```

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.
```

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
```

2.4 Break and Continue

Sometimes you want to modify what is being iterated over in a dynamic way while the program is running:

- `break` can be used to terminate a loop.
- `continue` can be used to skip the rest of the current iteration, and continue to the next iteration.

[48]:
```
for i in [5,4,3,2,1]:
    if i==2:
        break
    print('Count down:',i)
print('Take off!')
```

```
Count down: 5
Count down: 4
Count down: 3
Take off!
```

[49]:
```
for i in [5,4,3,2,1]:
    if i==2:
        continue
    print('Count down:',i)
print('Take off!')
```

```
Count down: 5
Count down: 4
Count down: 3
Count down: 1
Take off!
```

[50]:
```
# Find first week in which sales drops below 10.
sales=[10, 25, 5, 15, 20]
week=0
found=False
for num in sales:
    if num<10:
        found=True
        break
    week+=1
if found:
    print(f'Sales first drops below 10 in Week {week} with {num} units.')
```

```
Sales first drops below 10 in Week 2 with 5 units.
```

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.

[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=' ')`.

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

love programming python r
```

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.

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

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
```

Session 4: Replicating Tables with Loops

2.5 Example: Referral Marketing

A start-up's marketing team is trying to predict the growth of a new product launch. Assuming that each person who buys the product will refer a friend (who hasn't seen the product before) in the next month, as well as two months after buying, but will stop referring after that. Each referred friend will buy the product upon hearing about it and follow the same pattern of referral. Assume that

- each customer only buys the product once;
- there is one person who buys the product in month zero;
- the only buyers of the product are those referred.

a) How many people would buy the product in month 12?

b) Instead of calculating by hand, write Python code to compute the number of buyers in month 12. When doing so, try to print out the table as in the sample solutions to part a).

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**

In-Class Exercise:

Before you code, first walk through the above table to figure out the underlying logic. For each column, try to state a simple formula of how each entry is calculated, and write down code fragments representing this.

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.

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

```
7.2
```

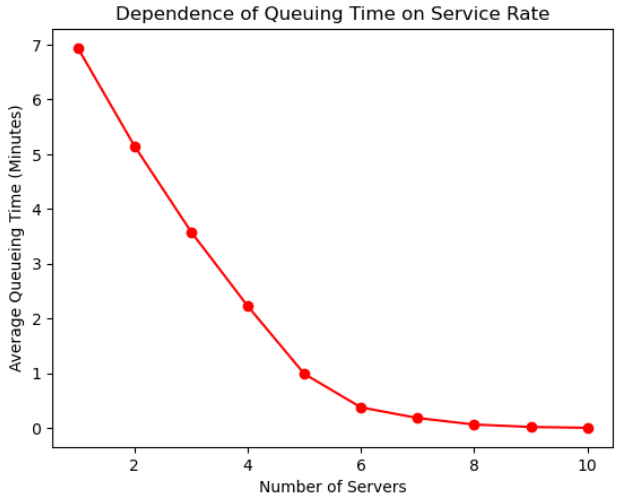
[63]:
```
# Test code 2
demand=[2,3,6,8,10,2,1,0,1,0]
for k in range(1,6):
    print(f'Av. Queue Length with {k} server(s):',avQueueLength(k,demand))
```

```
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.
```

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.

```
[69]: # 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
```

```
[70]: # 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
```

```
[71]: # 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
```

```
[72]: # 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!
```

Instructions for Quiz 1

Quiz 1 will take place on Thursday next week, in the first fifteen minutes of class. **The quiz will be open-notes but closed-computer. It will ask you to predict the outputs of certain code fragments that will be assigned to you.** To prepare for the quiz, you should practice manually tracing through the logic of Python code line by line. Becoming proficient in manually tracing code will help you better understand what the computer is doing and catch bugs more quickly.

As specified in the syllabus, the quiz is worth 4 percent of your final grade. If you miss the quiz, the weight will automatically be transferred to the midterm. After you are done, do not share the quiz questions or your answers with other students, including those in other sections. There will be many versions of the quiz, so don't be tempted to look at your neighbors' answers.

If you finish the quiz early, you can quietly stay in your seats or leave the room. **Do not use a cell phone, tablet or computer in the classroom before all the quizzes are handed in.**

Sample Quiz 1

Name: _____ Section: 12:30pm or 2pm

For each of the following three code cells, manually trace the code and write down the final output. There are no partial credits for incorrect answers.

1) Output: _____

```
[1]: x=5
     y='Hello'
     z=len(y)
     if x+z>10:
         x='DSO'
     else:
         x='MSBA'
     print(len(x+y))
```

2) Output: _____

```
[2]: y=2
     z=[0]
     i=0
     for x in [1,3,5,2,7,3]:
         z.append(max(y-x,0))
         y+=z[i]
         i+=1
     print(sum(z))
```

3) Output: _____

```
[3]: b=0
     d=5
     for a in [7,2,4,9,5,4]:
         b=a+d
         if b>=10:
             c=1
         else:
             c=0
         d=b-c*10
     print(d)
```