Contents

Exercises for Week 1: Review of Python Syntax

Grading scheme for the weekly exercises:

Download the Jupyter notebook containing all the exercises from Blackboard and submit it there before the due date. The grading scheme on all the weekly exercises are provided below.

- **3 (Essentially perfect)**: All required questions have a complete solution and all outputs are essentially correct based on the sample outputs.
- **2 (Solid effort)**: Several required questions may be incomplete or the outputs incorrect, but at least two thirds has a correct solution.
- **1 (Some effort)**: Between one third and two thirds of the required questions have a correct solution.
- **0 (No submission or essentially blank)**: No submission on Brightspace by the deadline, or less than one third of the required questions have a correct solution.

Every question is required unless it is marked with "(optional)" (which are usually the last few questions). **To ensure that you get 3 out of 3, before you submit, restart the Kernel and run all, and check that all of the outputs match the sample outputs from the PDF version of the exercises. You need to develop the habit of meticulously checking your outputs in order to ensure the best grade.** I will not count minor formating, rounding, or importing issues, but otherwise your output should match the sample outputs exactly when rerun. The weekly exercises are intended to be completed in 4-5 hours, excluding class time. You should budget at least this much time per week for these exercises.

Exercise 1.1: Automating a Calculation

a) Write a program that asks the user for the quantity sold and the price per unit and multiply them to calculate the total revenue, as in the sample run below. (Hint: you need to convert from string to float before multiplying.)

[33]:
```
# Sample run
```

```
Input quantity sold:  4
Input the price per unit:  3.5
```

```
Total revenue is $14.0.
```

[1]:
```
# Write your code for Exercise 1.1 a) here.
```

b) Write a program that calculates the present value of a certain investment, which will pay off a cash value of C in n years. The program should ask the user to input the final cash value C, the annual interest rate r (in percentage points), and the number of years n. The formula for present value V is

$$V = \frac{C}{(1 + \frac{r}{100})^n}$$

You should round the final answer to two decimal places, as in the sample run below.

```
[34]: # Sample run

Input final cash value:  300000
Input annual interest rate in percent:  5.5
Input number of years:  30

The present value is $60193.2.
```

```
[ ]: # Write your code for Exercise 1.1 b) here.
```

Exercise 1.2: Automating Simple Logic

a) Basestock Policy in Inventory Management

Write a program that asks the user for the current inventory level.

- If the inventory is at least equal to the basestock level of 100, then output "Sufficient inventory. No need to order."
- Otherwise, output "Order x units", where x is the difference between the basestock level and the current inventory.

```
[39]: # Sample run

Current inventory:  75

Order 25 units.
```

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

b) Payroll Calculator

Write a program that asks the user to input the number of hours worked this week, and calculates the total pay based on the following contract: for the first 40 hours, the hourly pay is 10. Each hour worked over 40 is compensated at an increased rate of 15 per hour.

```
[40]: # Sample run

Hours worked this week:  42.5

Total pay is $437.5.
```

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

c): Grade Conversion

Write a program that asks the user to input a numerical score, and convert it into a letter grade based on the following table:

Score	Grade
At least 90	A
At least 80 but less than 90	B
At least 70 but less than 80	C
At least 60 but less than 70	D
Less than 60	F

```
[41]: # Sample run

Enter numerical score:  95

Letter grade: A
```

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

Exercise 1.3: Temperature Converter

Write a program that asks for the user to input a temperature in Fahrenheit and convert it to Celcius, using the formula

$$T_{celcius} = (T_{fahrenheit} - 32) \times \frac{5}{9}.$$

The program should elicit the user to input a number using the prompt "Enter temperature in Fahrenheit:". Afterward, the program should print a line giving the temperature in celcius, rounded to two decimal places.

Sample run 1:

```
Enter temperature in Fahrenheit: 77
Equivalent temperature in celcius: 25.0
```

Sample run 2:

```
Enter temperature in Fahrenheit: 77.5
Equivalent temperature in celcius: 25.28
```

```
[42]: # Sample run

Enter temperature in Fahrenheit:  77

Equivalent temperature in celcius: 25.0
```

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

Exercise 1.4: BMI Calculator

Write a program which asks users to input their weight (in lbs) and height (in inches), and compute their body mass index (BMI), which is defind as $BMI = \frac{W}{H^2}$, where W is their weight converted to kg, and H is their height converted to m. (Assume 1 kg=2.205 lbs, and 1 m = 39.37 inches.) The program can assume that the weight and height the user inputs are positive numbers.

Classification	BMI (kg/m^2)
Underweight:	<18.5
Normal:	18.5 to 25 (>=18.5 but <25)
Overweight:	25 to 30 (>=25 but <30)
Obese:	>=30

The program should print a message classifying the BMI according to the above chart. The message must round the outputed BMI to 2 decimal places, and be formated exactly as the sample outputs below.

Sample run 1:

```
Please input your weight (in lbs): 145
Please input your height (in inches): 71
Your BMI is 20.22, which is classified as Normal.
```

Sample run 2:

```
Please input your weight (in lbs): 180.5
Please input your height (in inches): 60.5
Your BMI is 34.66, which is classified as Obese.
```

[43]:
```
# Sample run
```

```
Please input your weight (in lbs):  145
Please input your height (in inches):  71

Your BMI is 20.22, which is classified as Normal.
```

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

Exercise 1.5: Making your Code for 1.2 Re-Usable

a) Write a function called `orderQuantity` with two input arguments

- **inventory** (default value 0): number of items on hand.
- **basestock** (default value 100): the target inventory level.

If inventory is at least equal to basestock, then return 0. Otherwise, return how many items you should order to meet the target. Include an appropriate docstring to explain what the function does.

[61]:
```
# Write your function here
```

[62]:
```
# Code to test your function
print('Result with no inputs:', orderQuantity())
print(f'Order {orderQuantity(25)} when inventory is 25.')
print(f'Order {orderQuantity(51,50)} when inventory is 51 and basestock is 50.')
```

```
Result with no inputs: 100
Order 75 when inventory is 25.
Order 0 when inventory is 51 and basestock is 50.
```

[63]:
```
orderQuantity(basestock=200)
```

```
200
```

[64]:
```
orderQuantity(inventory=80)
```

```
20
```

[65]:
```
help(orderQuantity)
```

```
Help on function orderQuantity in module __main__:

orderQuantity(inventory=0, basestock=100)
    Calculates order quantity given current inventory and basestock level
    - inventory: number of items on hand
    - basetock: the target inventory level
```

[66]:
```
# Tests using assert
# Test case 1: inventory is greater than or equal to basestock
assert orderQuantity(100, 100) == 0
assert orderQuantity(200, 100) == 0

# Test case 2: inventory is less than basestock
assert orderQuantity(60, 100) == 40
assert orderQuantity(0, 100) == 100
```

b) Write a function called `calculateWage` with three input arguments:

- **hours**: the number of hours worked.
- **base** (default value 10): the pay for each of the first 40 hours worked.
- **bonus** (default value 0.5): the proportional bonus for each hour worked above 40.

The function should return the total pay.

[67]:
```
# Write your function here
```

[68]:
```
# Testing code
print('Pay for 42 hours with default base and bonus:',calculateWage(42))
print('Pay for 42 hours with base 12/hour and default bonus:',calculateWage(42,12))
print('Pay for 42 hours with base 12/hour and bonus 60%:', calculateWage(42,12,.6))
print('Pay for 42 hours with default base and bonus 60%:',calculateWage(42,bonus=0.
↪6))
```

```
Pay for 42 hours with default base and bonus: 430.0
Pay for 42 hours with base 12/hour and default bonus: 516.0
Pay for 42 hours with base 12/hour and bonus 60%: 518.4
Pay for 42 hours with default base and bonus 60%: 432.0
```

c) Write a function called `letterGrade` with one input argument:

- **score**: the numerical score.

The function should return a string denoting the letter grade.

[69]:
```
# Write your function here
```

[70]:
```
# Code to test
print(f'Grade corresponding to 95 is {letterGrade(95)}.')
print(f'Grade corresponding to 80 is {letterGrade(80)}.')
print(f'Grade corresponding to 60 is {letterGrade(60)}.')
```

```
Grade corresponding to 95 is A.
Grade corresponding to 80 is B.
Grade corresponding to 60 is D.
```

[71]:
```
# More extensive tests
assert letterGrade(90)=='A'
assert letterGrade(89)=='B'
assert letterGrade(80)=='B'
assert letterGrade(79)=='C'
assert letterGrade(70)=='C'
assert letterGrade(69)=='D'
assert letterGrade(60)=='D'
assert letterGrade(59)=='F'
assert letterGrade(0)=='F'
```

Exercise 1.6: Practicing List Syntax

a) Create an empty list called "l".

[19]:
```
# Write your code for a) here
```

b) Append each of the following objects to the list: `30`, `4.0`, `[3,4]`, `"Hello"`. The final value of the list should be as below.

[20]:
```
# Write your code for b) here
```

[21]:
```
# Value of l after running your code for parts a) and b) in order
l
```

```
[30, 4.0, [3, 4], 'Hello']
```

c) Obtain the type of each element of the list.

[ ]:

d) Obtain a slice of the list from `30` to `4.0`.

[26]: *# Write your code here*

```
[30, 4.0]
```

e) Obtain the third element `[3,4]` in two different ways, using positive and negative indexing respectively.

[27]:

```
[3, 4]
```

[28]:

```
[3, 4]
```

f) Find the length of the third element (which is a list as well).

[29]: *# Write your code here*

```
2
```

g) Check if the integer `4` is in the list. Do the same for the string `'4'`.

[30]: *# Check if 4 is in the list*

```
True
```

[31]: *# Check if '4' is in the list*

```
False
```

Exercise 1.7: Practice with Dictionary

a) Create an empty dictionary called "english".

[24]: *# Code for part a)*

b) Add the keys '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' to this dictionary, and each number should maps to its name in English in lowercase. (i.e. the key '3' should map the the value 'three'). You should do this one key-value pair at a time.

[25]: *# Code for part b)*

c) Ask the user to input a digit. If the input is one of the keys of the above dictionary, then print "You entered XXX." where XXX is the corresponding value. For any other input, print "I cannot read your input."

[26]: *# Sample run 1*

```
Please enter a digit: 3
You entered three.
```

[4]: *# Sample run 2*

```
Please enter a digit:  9
```

```
You entered nine.
```

[5]: *# Sample run 3*

```
Please enter a digit:  ff
```

```
I cannot read your input
```

[ ]: *# Write your code here*

Exercise 1.8: Practice with Series

a) Create a Series named `tuition` with the labels being the year and the values being the cost, as given by the two lists below.

```
[1]: years = [1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996,␣
 ↪1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,␣
 ↪2011, 2012, 2013, 2014, 2015, 2016, 2017]
     costs = [10000, 10750, 11900, 13300, 14250, 15350, 16400, 17500, 18550, 19750,␣
 ↪21000, 22700, 23840, 25000, 26260, 27250, 28500, 30050, 31800, 33650, 35600, 37500,␣
 ↪39600, 41900, 43800, 46150, 48600, 51200, 53500, 56175, 58875, 61225, 63675]
```

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

b) Display the first five entries as well as the last seven entries.

```
[3]: # Sample output for first five entries
```

```
1985    10000
1986    10750
1987    11900
1988    13300
1989    14250
dtype: int64
```

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

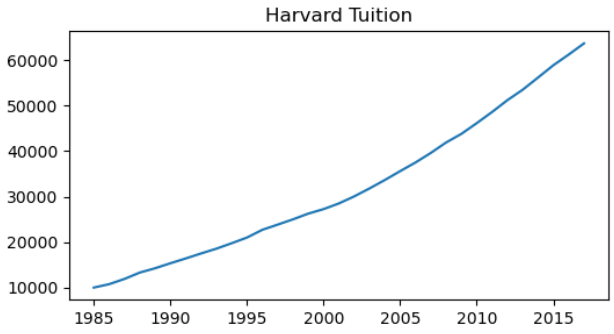
```
[4]: # Sample output for last seven entries
```

```
2011    48600
2012    51200
2013    53500
2014    56175
2015    58875
2016    61225
2017    63675
dtype: int64
```

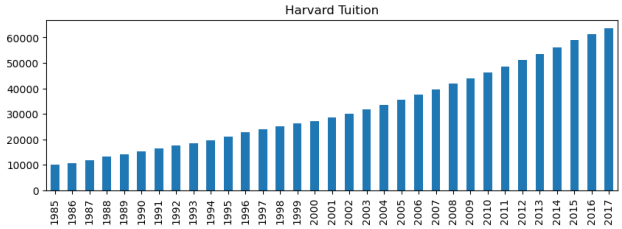
```
[ ]: # Write your code here
```

c) Make a line plot of the tuition, as well as a vertical bar plot.

```
[27]: # Sample output
```




```
[26]: # Sample output
```




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

```
[ ]: # and here
```

d) Create another Series called `normalized_tuition` which divides Series `tuition` by the tuition in the year 2000.

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

```
[8]: # Testing your code
     normalized_tuition.tail()
```

```
2013    1.963303
2014    2.061468
2015    2.160550
2016    2.246789
2017    2.336697
dtype: float64
```

e) What is the average tuition overall? What is the median tuition? Which year has the maximum tuition?

```
[9]: # Average tuition
```

```
31987.878787878788
```

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

```
[10]: # Median tuition
```

```
28500.0
```

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

```
[11]: # Year with maximum tuition
```

```
2017
```

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

f) What is the average tuition in the last 10 years of the data? What is the average tuition in the first 10 years? What is the average tuition from the Year 2001 to 2010 inclusive?

```
[12]: # Average tuition in last 10 years
```

```
52510.0
```

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

```
[13]: # Average tuition in first 10 years
```

```
14775.0
```

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

```
[14]: # Average tuition from 2001 to 2010 (inclusive)
```

```
36855.0
```

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

g) How many years in the dataset had tuition over between 30000 and 40000 (inclusive)?

```
[15]: # Correct output
```

```
6
```

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

h) Filter the Series and include only the years in which tuition is between 30000 and 40000.

```
[16]: # Correct output
```

```
2002    30050
2003    31800
2004    33650
2005    35600
2006    37500
2007    39600
dtype: int64
```

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

Exercise 1.9: Practice with DataFrames

This question asks you to perform simple operations based on the following DataFrame.

```
[37]: import pandas as pd
      insurance=pd.DataFrame([[1000,20,0.8],[2000,25,0.9],[1500,30,0.85]],\
                             index=['A','B','C'],\
                             columns=['Deductible','Copay','Coverage'])
      insurance
```

```
   Deductible  Copay  Coverage
A        1000     20      0.80
B        2000     25      0.90
C        1500     30      0.85
```

a) Obtain the coverage of plan B in two ways, one using `.loc` and one using `.iloc`.

```
[39]: # Obtaining the entry using .loc
```

```
0.9
```

```
[40]: # Obtaining the entry using .iloc
```

```
0.9
```

b) Obtain a DataFrame that sorts the plans by increasing order of deductible, as follows.

```
[41]: # Correct output
```

```
   Deductible  Copay  Coverage
A        1000     20      0.80
C        1500     30      0.85
B        2000     25      0.90
```

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

c) Obtain a Series which represents the copay column.

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

```
A    20
B    25
C    30
Name: Copay, dtype: int64
```

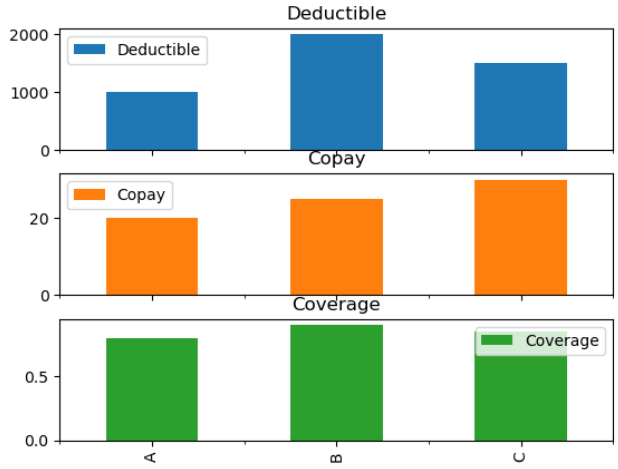
d) Obtain a Series which represents the row for plan B.

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

```
Deductible    2000.0
Copay           25.0
Coverage         0.9
Name: B, dtype: float64
```

e) Generate bar plots comparing the three plans, as below.

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



(Optional) Exercise 1.10: BMI Calculator v2

This question asks you to package the code from Exercise 1.4 into a function. In general, whenever you are asked to code a function, you should follow the steps in this exercise to first develop a version without the function as in Exercise 1.4, and package the code into a function only once everything is correct. To package a segment of code into a function, copy and paste the code segment into a new cell, highlight everything and press Tab to indent everything, then write the function declaration at the beginning, and make minor edits to make sure that the code segment is using the inputs from the function.

Write a function called `calculateBMI` with two input arguments:

- `weight`: the person's weight in lbs.
- `height`: the person's height in inches.

The function should return two objects:

- `bmi`: the person's BMI (not rounded, unlike in Exercise 1.4).
- `classification`: whether the person is "Underweight", "Normal", "Overweight", or "Obese", as in Exercise 1.4.

Sample input 1:

```
bmi,classification=calculateBMI(145,71)
print('BMI:',bmi)
print('classification:',classification)
```

Correct output:

```
BMI: 20.21964566293356
classification: Normal
```

Sample input 2:

```
calculateBMI(160,60)
```

Correct output:

```
(31.24206399596875, 'Obese')
```

Sample input 3:

```
weight=float(input('Enter weight (lbs): '))
height=float(input('Enter height (inches): '))
bmi,classification=calculateBMI(145,71)
print(f"The person's BMI is {round(bmi,2)}, which is {classification}.")
```

Sample output (as in Exercise 1.4, output depends on what user inputs):

```
Enter weight (lbs): 145
Enter height (inches): 71
The person's BMI is 20.22, which is Normal.
```

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

```
[8]: # Sample run 1
     bmi,classification=calculateBMI(145,71)
     print('BMI:',bmi)
     print('classification:',classification)
```

```
BMI: 20.21964566293356
classification: Normal
```

```
[9]: # Sample run 2
     calculateBMI(160,60)
```

```
(31.24206399596875, 'Obese')
```

```
[10]: # Sample run 3
      weight=float(input('Enter weight (lbs): '))
      height=float(input('Enter height (inches): '))
      bmi,classification=calculateBMI(weight,height)
      print(f"The person's BMI is {round(bmi,2)}, which is {classification}.")
```

```
Enter weight (lbs): 145
Enter height (inches): 71
The person's BMI is 20.22, which is Normal.
```

(Optional) Exercise 1.11: Blood Sugar Checker

Write a function called `bloodSugarCheck` with two input arguments:

- `hours`: the number hours the patient has fasted.
- `level`: the patient's blood sugar level.

The function should print a message based on the following logic:

- If they have fasted strictly less than 2 hours, then print "You need to fast at least 2 hours to perform this test."
- If they fasted at least 2 hours but strictly less than 8 hours, then print "Your blood sugar level is high" if `level` is strictly more than 140, and "Your blood sugar level is normal" otherwise.

- If they fasted for at least 8 hours, then print "Your blood sugar level is high" if `level` is strictly more than 100, and "Your blood sugar level is normal" otherwise.

Include the doc-string "Function for triaging a patient's blood sugar level based on hours of fasting and current level."

Sample run 1:

```
bloodSugarCheck(1,100)
```

Correct output:

```
You need to fast at least 2 hours to perform this test.
```

Sample run 2:

```
bloodSugarCheck(2,110)
```

Correct output:

```
Your blood suguar level is normal.
```

Sample run 3:

```
bloodSugarCheck(2,141)
```

Correct output:

```
Your blood sugar level is high.
```

Sample run 4:

```
bloodSugarCheck(8,110)
```

Correct output:

```
Your blood sugar level is high.
```

Sample run 5:

```
help(bloodSugarCheck)
```

Correct output:

```
Help on function bloodSugarCheck in module __main__:

bloodSugarCheck(hours, level)
    Function for triaging a patient's blood sugar level based on hours of fasting and current level.
```

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

```
[12]: bloodSugarCheck(1,100)
```

```
You need to fast at least 2 hours to perform this test.
```

```
[13]: bloodSugarCheck(2,110)
```

```
Your blood suguar level is normal.
```

```
[14]: bloodSugarCheck(2,141)
```

```
Your blood sugar level is high.
```

```
[15]: bloodSugarCheck(8,110)
```

```
Your blood sugar level is high.
```

```
[16]: help(bloodSugarCheck)
```

```
Help on function bloodSugarCheck in module __main__:

bloodSugarCheck(hours, level)
    Function for triaging a patient's blood sugar level based on hours of fasting and␣
 ↪current level.
```