DSCI 503 – HW 02 Instructions

General Instructions

Create a new notebook named **`HW_02_YourLastName.ipynb`** and complete problems 1 – 8 described below.

Any set of instructions you see in this document with an orange bar to the left will indicate a place where you should create a markdown cell. For each new problem, create a markdown cell that indicates the title of that problem as a level 2 header.

Any set of instructions you see with a blue bar to the left will provide instructions for creating a single code cell.

Read the instructions for each problem carefully. Each problem is worth 6 points. An additional 2 points are allocated for formatting and following general instructions.

The assignment should be completed using only base Python commands. Do not use any external packages.

Assignment Header

Create a markdown cell with a level 1 header that reads: "DSCI 503 - Homework 02". Add your name below that as a level 3 header.

Problem 1: Altering Lists

In this problem, you will create and update lists containing information about employees within a particular department at a company. The table below provides the names of the employees in the department, as well as the number of years of service they have at the company:

Employee	Years of Service
Alex	5
Beth	15
Chad	2
Drew	8
Emma	4
Fred	11

Create two lists. One should be named **`employees`** and should contain the names of the employees as strings. The second should be named **`years`** and should contain the number of years of service for each employee, stored as integers.

The lists should be created in "parallel" so that the values in the two lists at a particular index refer to the same person. Furthermore, the lists should be ordered in decreasing order of service. In other words, the person with the greatest number of years of service should appear first in the list, and the person with the fewest years of service should appear last in the list.) You should perform this sorting manually when creating the lists.

You do not need to print anything in this cell.

Suppose that two employees from other departments transfer to this department. The new members are named Gina and Herb. Gina has 12 years of service at the company and Herb has 3 years of experience.

In addition, the company hire a new employee, Iris, to join the department. Iris has 0 years of experience with the company.

Update **`employees`** and **`years`** to include these new members. The updates should be performed so that they are still ordered by years of service, in decreasing order.

Print both lists.

Fred accepts a position at another company. Remove his information from the lists.

It is discovered that due to a clerical error, Emma's years of service was recorded incorrectly. She actually has 6 years of experience with the company rather than 4. Update the lists to reflect this new information. The updates should be performed so that they are still ordered by years of service, in decreasing order.

Print both lists.

Problem 2: Sorting and Slicing Lists

In this problem, you will get practice with sorting and slicing lists, and with using list functions such as **`max()`** and **`min()`**.

Create a code cell and copy the following code into it (without the indentations).

```
import random
random.seed(12)
random_list = random.sample(range(500), 99)
```

This code will create a list named **`random_list`** that contains 99 randomly generated integers. We will discuss how this code works later in the course.

Print the length of **`random_list`**. Then print the first 10 elements of this list, all on the same line. Format your output as follows:

```
Length of random_list: xxxx
First 10 Elements of random_list: xxxx
```

Use the **`sum()`** function to calculate the sum of the elements in random_list. Print the result in the format shown below. Match the formatting exactly, including the period at the end of the sentence. There should be no space between the sum and the period.

```
The sum of the elements in random list is xxxx.
```

Create a list named **`sorted_list`** that contains the same elements of **`random_list`**, but sorted in increasing order. The order of the elements in **`random_list`** should not be changed. Print the first ten elements of both lists on separate lines. Format your output as follows:

```
First 10 Elements of sorted_list: xxxx
First 10 Elements of random_list: xxxx
```

Create lists named **`bot_slice`**, **`mid_slice`**, and **`top_slice`** as follows:

- **`bot_slice`** should contain the smallest 33 elements in **`sorted_list`**.
- **`mid_slice`** should contain the middle 33 elements in **`sorted_list`**.
- **`top_slice`** should contain the largest 33 elements in **`sorted_list`**.

Print the six messages shown below, with the appropriate numerical values inserted in place of the xxxx symbols. Include the periods at the end of the sentences, and include blank lines between each pair of sentences, as shown.

```
The smallest element of bot_slice is xxxx.
The largest element of bot_slice is xxxx.

The smallest element of mid_slice is xxxx.
The largest element of mid_slice is xxxx.

The smallest element of top_slice is xxxx.
The largest element of top_slice is xxxx.
```

Problem 3: Calculating Sum of Squared Errors

Suppose that a biologist is studying the relationship between the length and weight of yellow-bellied marmots. Based on prior work, she believes that the length and weight of adult marmots can be approximately modeled by a relationship of the following form: $weight = 0.6 * length - 5.2$

To test this model, the biologist collects a sample of 20 adult marmots. The lengths and weights of the marmots are as follows:

```
length = [22.7, 22.4, 25.8, 21.3, 20.1, 22.1, 21.1, 25.3, 26.9, 26.9,
          23.0, 23.8, 26.2, 20.4, 23.0, 21.9, 23.5, 27.8, 25.3, 25.9]

weight = [9.2, 8.8, 10.7,  8.3, 6.2, 8.6, 7.2, 11.2, 10.5, 11.3,
          9.6, 9.9, 10.9,  5.9, 9.5, 9.1, 9.7, 11.6, 10.2, 10.5]
```

Create these lists, as the appear here. You do not need to print anything in this cell.

Create a list named **`pred_weight`** that contains the predicted weight for each marmot, according to the biologist's model.

You should begin by creating **`pred_weight`** as an empty list. Then loop over the elements of **`length`**. Each iteration of the loop should calculate a new predicted weight using the formula provided above, and rounded to 2 decimal places. The round value for each predicted weight should be appended to the list **`pred_weight`**.**`pred_weight`**.

Print **`pred_weight`**.

The biologist wishes to score her model using the sum of squared errors (**SSE**) metric. This metric is calculated as follows:

1. For each observation, calculate the difference between the true weight and the predicted weight.
2. Square this difference.
3. Sum the squared differences over all observations

Use a for loop to calculate the model's **SSE** score for this sample of marmots.

There are multiple ways to perform this task, but for the sake of efficiency, I would like for you to do it **without creating any new lists** and using **only one loop**.

Print the value of **SSE**, rounded to four decimal places. Note that the value stored in **SSE** should not be rounded, only the value displayed.

Problem 4: Calculating Exam Scores

In this problem, you will use a loop to calculate scores for two students who have taking a multiple-choice exam with 25 questions. Assume that the correct answers on the exam are as follows:

Correct Answers: `D, B, C, A, C, D, A, C, C, B, D, A, B, D, C, D, C, D, C, A, B, D, C, B, A`

The answers submitted by the two students are provided below.

Student 1 Answers: `A, B, C, A, B, D, A, A, C, B, D, A, D, C, C, B, C, D, B, A, D, D, C, C, A`
Student 2 Answers: `D, A, C, A, B, D, A, C, C, B, D, A, B, D, A, D, C, D, C, A, B, C, C, B, A`

Perform the following steps in a single code cell:

1. Create lists of named **`correct`**, **`answers1`**, and **`answers2`** to store the correct answers for the exam, as well as the answers submitted by the two students. The answers should be stored as strings of the form `'A'`, `'B'`, `'C'`, and `'D'`.
2. Create variables named **`count1`** and **`count2`** to store the number of correct answers for each student.
3. Use a single loop to simultaneously loop over all three lists. As you do so, count the number of correct answers for student 1 and student 2.
4. When the loop is finished executing, divide each count of correct answers by the total number of questions and multiply by 100 to obtain the grades for both students. Store the results **as integers** in variables named **`grade1`** and **`grade2`**.
5. Print the results with descriptive messages as shown below, with the **xxxx** symbols replaced with the appropriate values. Match the format and spacing exactly.

```
Student 1 Grade: xxxx%
Student 2 Grade: xxxx%
```

No lists should be created for this problem other than the three described in Step 1. Only one loop should be used in this problem.

Problem 5: Determining Monthly Payments

In this problem we will be determining the monthly payment that would be required to repay a mortgage on a home in 30 years for a range of different house prices.

Assume that a loan collects interest at a monthly interest rate of `i`, and let n denote the number of monthly payments (for a 30-year loan, then $n = 360$). The size of the loan (denoted by L) and the required monthly payment (denoted by PMT) are related according to the following equation:

$$L = PMT \cdot \frac{1 - (1 + i)^{-n}}{i}$$

Assume that you are considering purchasing a house and have been approved for a 30-year loan with a 0.4% monthly interest rate. We will calculate the monthly payment required for a several house prices between $100,000 and $200,000. Specifically, we will consider a sequence of prices starting at $100,000 and increasing in increments of $10,000 up to (and including) a price of $200,000.

Perform the following steps in a single code cell.

1. Create variables named **`i`** and **`n`** to store the interest rate rate and the number of monthly payments. These values will remain constant throughout this problem.
2. Create a variable name **`start`** and set it equal to 100,000. Create another variable named **`increment`** and set it equal to 10,000.
3. Use a loop to perform the tasks below for each of the house prices that we wish to consider. Note that since we are using **`i`** to denote the interest rate, you will need to select a different name for the loop counter.
    - Calculate the new loan amount to be considered, storing the result in a variable.
    - Calculate the require monthly payment for this loan amount, rounded to 2 decimal places.
    - Print the result in the format shown below. Match the format exactly, including the inclusion of the period at the end of the sentence. There should be no spaces following the dollar signs.

```
A loan of $xxxx would require monthly payments of $xxxx.
```

No lists should be created for this problem. Only one loop should be used in this problem.

Problem 6: Creating an Amortization Schedule

Assume that you have decided to borrow $160,000 to purchase a house. Suppose that the loan has a monthly interest rate of 0.4% and that you will repay it over a period of 30 years (360 months). According to your results in the previous problem, you should see that this will result in monthly payments of $839.46. In this problem, we will use a loop to determine how much will still be owed on the loan at the end of each year.

Perform the following steps in a single code cell.

1. Create variables named **`balance`**, **`i`** and **`pmt`** to store the initial loan amount, interest rate, and monthly payment amount mentioned in the paragraph above.
2. Use a loop to perform the following tasks 360 times. As in Problem 5, you will need to use a name other than **`i`** for the loop counter. For convenience, I recommend starting the loop counter at 1 rather than 0.
    - Calculate the new balance. This will be done by multiplying the current balance by **`1+i`**, and then subtracting the payment amount. Store the result back into **`balance`**, rounded to 2 decimal places.
    - If current payment marks the end of a year of payments (i.e. if the loop counter is a multiple of 12), then print a message in the format shown below. Match the format exactly, including the inclusion of the period at the end of the sentence. Do not include extra spaces.

      ```
      The balance at the end of xxxx months will be $xxxx.
      ```

No lists should be created for this problem, and only one loop should be used.

Problem 7: Calculating Price of a Bulk Order

WidgCo is a company that sells widgets. The company uses the bulk pricing policy described below.

- If 100 or fewer widgets are ordered, then each widget costs $350.
- If 200 or fewer widgets are ordered, the first 100 widgets each cost $350, and each additional widget costs $300.
- If more than 200 widgets are ordered, then the first 100 each cost $350, the next 100 each cost $300, and each widget beyond the first 200 will cost $250.

In this problem, you will determine the price for orders containing the following numbers of widgets:

```
84, 100, 126, 150, 186, 200, 216, 248
```

Perform the following steps in a single code cell:

1. Create a list named **`quantities`** to store the order quantities mentioned above.
2. Loop over this list. Each time the loop executes, perform the following steps:
    - Calculate the price of the order.
    - Print a message in the format shown below, with the xxxx symbols replaced by the appropriate values. Match the format exactly, including the period at the end of the sentence. Do not include extra spaces.

      ```
      The cost for an order of xxxx widgets is $xxxx.
      ```

Only one list should be created for this problem, and only one loop should be used. Your loop should contain only one print statement.

Problem 8: Adding Matrices

In this problem, you will be asked to calculate the sum of two 3x5 matrices. Each matrix will be stored as a list of lists, with each sub-list representing a single row of the matrix.

Consider the following two matrices:

$$A = \begin{bmatrix} 13 & 43 & 28 & 22 & 41 \\ 17 & 39 & 46 & 16 & 21 \\ 41 & 34 & 31 & 25 & 14 \end{bmatrix} \qquad B = \begin{bmatrix} 35 & 29 & 43 & 21 & 31 \\ 48 & 26 & 19 & 17 & 23 \\ 32 & 34 & 24 & 16 & 27 \end{bmatrix}$$

Create two lists named **A** and **B** to represent the matrices above. Each of these lists should contain three lists, with each sub-list containing 5 integers representing a row in the matrix. For example, the first element of **A** should be the list **`[13, 43, 28, 22, 41]`**.

We will now use loops to calculate the sum of the matrices A and B.

Create an empty list named **`AplusB`**. Write a nested loop that adds the elements of **A** and **B**, storing the result in **`AplusB`**. Each time the outer loop executes, it should calculate a new row of the sum and should finish by adding this row to **`AplusB`**. The final structure of **`AplusB`** should be the same as **A** and **B**.

Print the rows of **`AplusB`**, with each row appearing on its own line.

Submission Instructions

When you are done, click **Kernel > Restart and Run All**. If any cell produces an error, then manually run every cell after that one, in order. Save your notebook, and then export the notebook as an HTML file. Upload the HTML file to Canvas and upload the IPYNB file to CoCalc, placing it in the **Homework/HW 02** folder.