Homework #3

NOTE: Read the instructions CAREFULLY. They should provide almost everything you need to write the code. Be sure to look at examples posted on Canvas, as well as examples in the slides and book. We have seen most of what you will need to write these programs.

NOTE: You may notice that some problems require you to use a loop that has an index variable/loop variable. These are loops such as a while loop, or using a for loop with a range function. This is to get you somewhat familiar with using an index to access list elements, because NumPy uses similar principles.

Problem 1:

In this problem, we are going to compute the dot product of two sequences. If you do not know how to compute a dot product, do not worry. The instructions go into detail as to how to do this.

You do not need to write any functions for this program. The data we start with will be stored in a list and a tuple.

- Name the list cList. The list values at the beginning of the program will be: 9, 4, 7, 3, 8
- Name the tuple cTuple. The tuple values at the beginning of the program will be: 5, 4, 3, 5, 2
- We will also have a third list, productList. This should be set to an empty list at the start.

To produce a dot product, write code that will:

1. Determine the length of cList. Note that cList and cTuple will be the same length for this program. In a more realistic situation we would have to check both, but we know they are the same length so you only have to check one. Do NOT hard-code the length, e.g. use a literal. Assessing the length can be done very easily (if it's difficult at all, reconsider your work). Save the list length to a variable, listLen.
2. Run a **while** loop to iterate over both cList and cTuple, so that it processes every element in both Use a loop variable named **loopIndex**. USE THE CONTENTS OF listLen TO DETERMINE HOW LONG TO ITERATE THE LOOP. **DO NOT USE A LITERAL VALUE OF "5" FOR THIS LOOP.** In this loop:
    a. Use loopIndex as the index to access the correct value of cList and cTuple.
    b. Multiply both of these values and bind the result to a variable named productOfPair.
    c. Append productOfPair to productList.
    d. Increment loopIndex
3. Set a variable named total to 0.
4. Iterate over productList in a manner similar to how you iterated over cList and cTuple, using a while loop and an index variable. Add up the value of every list element in productList, continuously adding each element to total. USE THE CONTENTS OF listLen TO DETERMINE HOW LONG TO ITERATE THE LOOP. **DO NOT USE A LITERAL VALUE OF 5 FOR THIS LOOP.**
5. Print the value in total.

The finished contents of productList should be: 45, 16, 21, 15, 16

The end result of the program should be 113.

Problem 2:

Write a function, buildMultTable(), to create a multiplication table in a 2D list, using factors 0 through 9. This will require two nested while loops.

Within buildMulTable(), we will start by preparing a loop variable, named outerLoop. Set it to 0. We will also make a variable named mulTable, and assign it to an empty list.

Start writing the outer loop. It should loop from 0 through 9, using outerLoop. Note that, unlike with Problem 1, you can use the literal values of 0 and 9 (or 10, depending on your loop condition). You can use a for loop (*only* if you use the range function) *or* a while loop. **You MUST use a loop with an *index variable. outerLoop will be your index variable*.**

Within the outer loop:

1. Set a variable named innerLoop to 0. We must do this every time the outer loop iterates, because the inner loop has to run several times for every iteration of the outer loop.
2. Set a variable named currentLine to hold an empty list. Similarly we must do this to prepare a list for the inner loop so that it can be filled with data.
3. Start the inner loop, so that it iterates from 0 to 9 using innerLoop, using the **exact same type of loop** you used for the outer loop, and using innerLoop as the looping and index variable. Within the inner loop:
    a. Multiply innerLoop by outerLoop.
    b. Store the result in currentLine. Use innerLoop as your index.
    c. Increase innerLoop, if you are using a while loop.
4. Now that we are back in the outer loop, we have built a full line for the multiplication table. Append this to the mulTable list.
5. Increase outerLoop, if you are using a while loop.

Now that the outer loop is done, mulTable should contain our full multiplication table. Return mulTable.

At the global level, you should call buildMulTable() and bind the return value to a variable named mt. Print mt.

HINT: Look at the program in the slides that demonstrates a two-dimensional loop and use it as a basis to build your program. The code on the slides can act as a guide. This problem may require a lot of text to describe it, but it is not as difficult as it may seem.

Problem 3:

In this program, we are going to take a list that is input to a function, add up the middle elements (in other words, everything but the first and last element), and then change the list into a three

element list containing the first element, the sum of the middle elements, and the last element. We will use **slicing** to accomplish this.

Start by writing a function named sumMiddle(). It takes one parameter, a list named dataList.

Within sumMiddle(), write code that does the following:

1. First, the function should determine the length of dataList. We will assume the list length is greater than or equal to 3 since this is an exercise, so we will not have error checking. There is a simple way to determine the length of the list. Put this in a variable named lengthOfList.
2. Now, use slicing to extract the middle of the list. Use lengthOfList to calculate where the ending of the list is. You must use slicing to access the entire list, EXCEPT for the very first and very last elements. Bind the slice to another variable, sliceToSum. While you can use a literal value for the starting index for the slicing, **you MUST use lengthOfList to calculate the ending index**. Do not simply insert a literal value in there.
3. Sum up the elements extracted using the sum() function in the Python standard library (you should not need to import anything). Store it in a variable named middleSum.
4. **Replace** the elements you extracted with middleSum. **Use slicing to replace the section of the list with the sum.** You can use the original slicing specifications you used to extract the section of the list in order to replace it, e.g. calculate it from lengthOfList. As you are replacing the middle elements with a single element using slicing, **the list will have a length of 3 at the end**.

You do not need to return anything, because the list object that was passed was changed due to pass-by-value.

Then, write a small function called program(). It takes no parameters.

1. First, in program(), initialize a list. Name it myList. It should contain the following values to begin with: 2, 4, 6, 8, 10
2. Then, print myList.
3. After this, call sumMiddle(), and give it myList as its only parameter.
4. After this, print myList again.
5. Then exit the function.

Outside of the function, you should only have one line of code. It calls program() with no parameters.

The first list output should be exactly the list we started with. The last should be a list with the values: 2, 18, 10.

This is because the purpose of the function sumMiddle() is to add up all of the elements, except for the first and the last, and replace those elements with the sum.