# CSC121 Lab 06: Nested Lists and Tuples

## Goals

In this lab assignment, students will demonstrate the ability to:

* Create and use nested lists
* Create and use tuples
* Use list comprehension to create lists
* Install and use the matplotlib package to plot list data

NOTE: This lab document is a copyrighted work of Wake Tech Community College and the course instructor. Any posting of this document outside of Wake Tech is considered a copyright violation. Students who post these documents outside of Wake Tech are subject to academic and possible legal actions.

## Instructions

In this lab, you will demonstrate your mastery of lists and tuples as well as your understanding of the Python package matplotlib.

Follow the instructions in each problem and submit the specified files. All problems will require that Python code be submitted as well as screenshots that prove the programs have been executed in PyCharm.

All problems will consist of programs that you create from scratch that meets the problem specification.

The **main** function: As with Lesson 4 and 5, all programs that you write from scratch must have a main function that contains the top-level logic of your code. Otherwise, you should follow the instructions provided in the lab document.

Remember this for all future labs as well. This reminder is the last one, but as part of the Lab Assignment Guidelines, this applies to the lab problems for the rest of the semester.

## Problems

### Problem 1 – Nested Lists

This problem tests your understanding of nested lists.

Create a file named **Lab06P1.py**.

A college instructor has three students, Ashley, Barb, and Carl, who are taking her class. Each student has taken either 3 to 5 assignments which have been graded.

Write a Python program that performs the following steps. Label each block of code that performs these steps with an appropriate comment (e.g. # Step a):

1. Ask the user to enter 3 assignment scores for Ashley. Store the scores in a list named a\_list. Display the list.
   * NOTE: The assignment scores for all students can only be integers, so convert the input strings to ints.
2. Ask the user to enter 5 assignment scores for Barb. Store the scores in a list named b\_list. Display the list.
3. Ask the user to enter 4 assignment scores for Carl. Store the scores in a list named c\_list. Display the list.
4. Use the following code to create a copy of each list and construct a list of lists from them:

all\_scores = [a\_list[:], b\_list[:], c\_list[:]]

Note: We want the elements of all\_scores to be a copy of the three score lists but not the three score lists themselves. Therefore, the following code does not generate what we want:

all\_scores = [a\_list, b\_list, c\_list] # DO NOT DO THIS!!

After creating the all\_scores list, print that list.

1. The instructor wants to give extra credit to all these students by multiplying each assignment score by 1.05 (5% extra credit) and converting the results to an int. **Use a nested set of for loops** to apply the extra credit to every score in all\_scores. Display all\_scores.
   * NOTE: In order to update the inner elements of a nested list, you must have the for loops generate indexes. With indexes you can then write into each inner element. If you are unclear on this concept, see the explainer at the end of the document.
2. Create a new list called score\_spreads that is the difference between the high and low scores of the extra credit assignment scores of the three students.
   * Use the data from the updated all\_scores to find the maximum and minimum values and compute the difference (maximum - minimum).
     + **You do NOT need nested for loops for this step.**
   * Use a single for loop to access each inner list, then find the maximum and minimum of the inner list. Append that difference between those values to the score\_spreads list.
   * Display score\_spreads.
3. Display the original lists that were created in Steps a, b, and c. This output will demonstrate the original lists were not altered when the all\_scores list was updated.

Sample output:

Please enter Ashley's scores one by one.

Enter a score: 90

Enter a score: 83

Enter a score: 95

Ashley's scores: [90, 83, 95]

Please enter Barb's scores one by one.

Enter a score: 100

Enter a score: 78

Enter a score: 89

Enter a score: 87

Enter a score: 90

Barb's scores: [100, 78, 89, 87, 90]

Please enter Carl's scores one by one.

Enter a score: 57

Enter a score: 82

Enter a score: 85

Enter a score: 89

Carl's scores: [57, 82, 85, 89]

All scores: [[90, 83, 95], [100, 78, 89, 87, 90], [57, 82, 85, 89]]

All scores after extra credit: [[94, 87, 99], [105, 81, 93, 91, 94], [59, 86, 89, 93]]

Score spreads: [12, 24, 34]

Original Scores

Ashley's scores: [90, 83, 95]

Barb's scores: [100, 78, 89, 87, 90]

Carl's scores: [57, 82, 85, 89]

Run this program and take a screenshot with the results. Name the screenshot **Lab06P1-ouput.jpg**.

Submit both files, **Lab06P1.py** and **Lab06P1-output.jpg**, to Blackboard for credit.

### Problem 2 – Tuples

This program tests your understanding of tuples.

Create a file named **Lab06P2.py**. Write a Python program to do the following (note your output should include the steps being performed):

1. Ask the user to enter the number of values that should be in the tuple.
2. Ask the user for the start and end values to use when generating a random number.
3. Use a for loop and a random integer generator to generate the number random integers indicated by the user in the range that was indicated. Store the random integers in a tuple. Display the tuple as shown in the sample output below.
   * Hint: You may want to store the random integers in a list first and then convert the list to a tuple.
4. Create a new tuple. Copy the **first four** elements of the tuple in part (c) to this tuple. Display this tuple as shown below.
5. Create a new tuple. Copy the **last two** elements of the tuple in part (c) to this tuple. Display this tuple as shown below.
6. Concatenate the two tuples in part (e) and part (d). Note your program is putting the step (e) tuple first followed by the step (d) tuple. Display the concatenated tuple as shown below.
7. Add 1 from all the elements in the tuple and display the results. You may need to convert your tuple to a list first, or create a new list, but what you display MUST be a tuple.

Sample output:

Step a:

How many values to put in the tuple? 8

Step b:

What is the start of the range? 5

What is the end of the range? 25

Step c: Tuple of 8 random numbers: (10, 19, 9, 25, 22, 11, 11, 6)

Step d: Tuple of first 4 numbers: (10, 19, 9, 25)

Step e: Tuple of last 2 numbers: (11, 6)

Step f: Two tuples concatenated: (11, 6, 10, 19, 9, 25)

Step g: Two tuples concatenated and incremented: (12, 7, 11, 20, 10, 26)

As shown in the sample output, your output should include which step is being displayed.

Run this program and take a screenshot with the results. Name the screenshot **Lab06P2-ouput.jpg**.

Submit both files, **Lab06P2.py** and **Lab06P2-output.jpg**, to Blackboard for credit.

### Problem 3 – List Comprehensions

This problem tests your understanding of list comprehensions.

Create a file named **Lab06P3.py**. Write a Python program that performs the following steps. In your program output, make sure to indicate which step is being demonstrated with your output (see sample output):

1. Start your program (after the comment header, inside the main function) with this predefined list:

list1 = [4, 5, 8, 2]

list2 = [2, 5, 9]

1. Write a one-line list comprehension that is equivalent to the following code and print list3:

list3 = []

for num in range(6):

list3.append(num \* 2 - 3)

1. Write a one-line list comprehension that is equivalent to the following code and print list4:

list4 = []

for i in range(4):

for j in range(5):

if i % 2 == 1 and j % 2 == 0:

list4.append([i, j])

1. Write a one-line list comprehension that is equivalent to the following code and print list5:

list5 = []

for i in list1:

list5.append(i \*\* 3)

1. Write a list comprehension with list1 only as an input sequence to generate this list:

[12, 15, 24, 6]

1. Write a list comprehension with both list1 and list2 as input sequences to generate this list:

[7, 19, 35, 9, 24, 44, 15, 39, 71, 3, 9, 17]

1. Write a list comprehension with both list1 and list2 as input sequences to generate this list:

['4@2', '4@5', '4@9', '5@2', '5@5', '5@9', '8@2', '8@5', '8@9', '2@2', '2@5', '2@9']

Sample output:

Step b: [-3, -1, 1, 3, 5, 7]

Step c: [[1, 0], [1, 2], [1, 4], [3, 0], [3, 2], [3, 4]]

Step d: [64, 125, 512, 8]

Step e: [12, 15, 24, 6]

Step f: [7, 19, 35, 9, 24, 44, 15, 39, 71, 3, 9, 17]

Step g: ['4@2', '4@5', '4@9', '5@2', '5@5', '5@9', '8@2', '8@5', '8@9', '2@2', '2@5', '2@9']

Run this program and take a screenshot with the results. Name the screenshot **Lab06P3-ouput.jpg**.

Submit both files, **Lab06P3.py** and **Lab06P3-output.jpg**, to Blackboard for credit.

### Problem 4 – Matplotlib

This problem will test your ability to use the Python module matplotlib. For this problem, review the part of Section 7.11 labeled **Plotting a Bar Graph**. Review Programs 7-25 through 7-27 for examples of using matplotlib to create a bar graph. The example code provided with the lecture videos will be very useful as well.

Create a file named **Lab06P4.py**. Write a program that helps a user generate a line graph graphic:

1. As discussed in the book, you will need to use the Python utility **pip** to install matplotlib. Follow the instructions in the book for details.
2. After your program comment header, add the appropriate line for importing the module matplotlib that makes it convenient to work with the pyplot portion of the module:

import matplotlib.pyplot as plt

1. Your program should ask the user to enter a title for a bar graph.
2. The program should ask the user for labels for the x-axis and y-axis.
3. The program should ask the user for the number of data points.
4. Then your program should ask the user for the data point labels and values using a for loop. The labels will be used to label the bar locations on the x-axis. The values will be used to plot the bar heights to the y-axis.
5. After collecting the user's input into a list of labels and a list of values, use the appropriate pyplot functions to do the following:
   1. Create the bar graph.
   2. Add a title for the chart.
   3. Add labels for the x-axis and y-axis.
   4. Display the bar graph.
6. The program terminates after the user closes the window displaying the bar graph.

Sample output:

Enter the bar graph title: Daily Temps

Enter the label for the x-axis: Day of Week

Enter the label for the y-axis: Temperature

Enter the number of data points: 7

Enter the name for tick 1: Mon

Enter the value for tick 1: 66

Enter the name for tick 2: Tue

Enter the value for tick 2: 72.3

Enter the name for tick 3: Wed

Enter the value for tick 3: 76

Enter the name for tick 4: Thu

Enter the value for tick 4: 74.8

Enter the name for tick 5: Fri

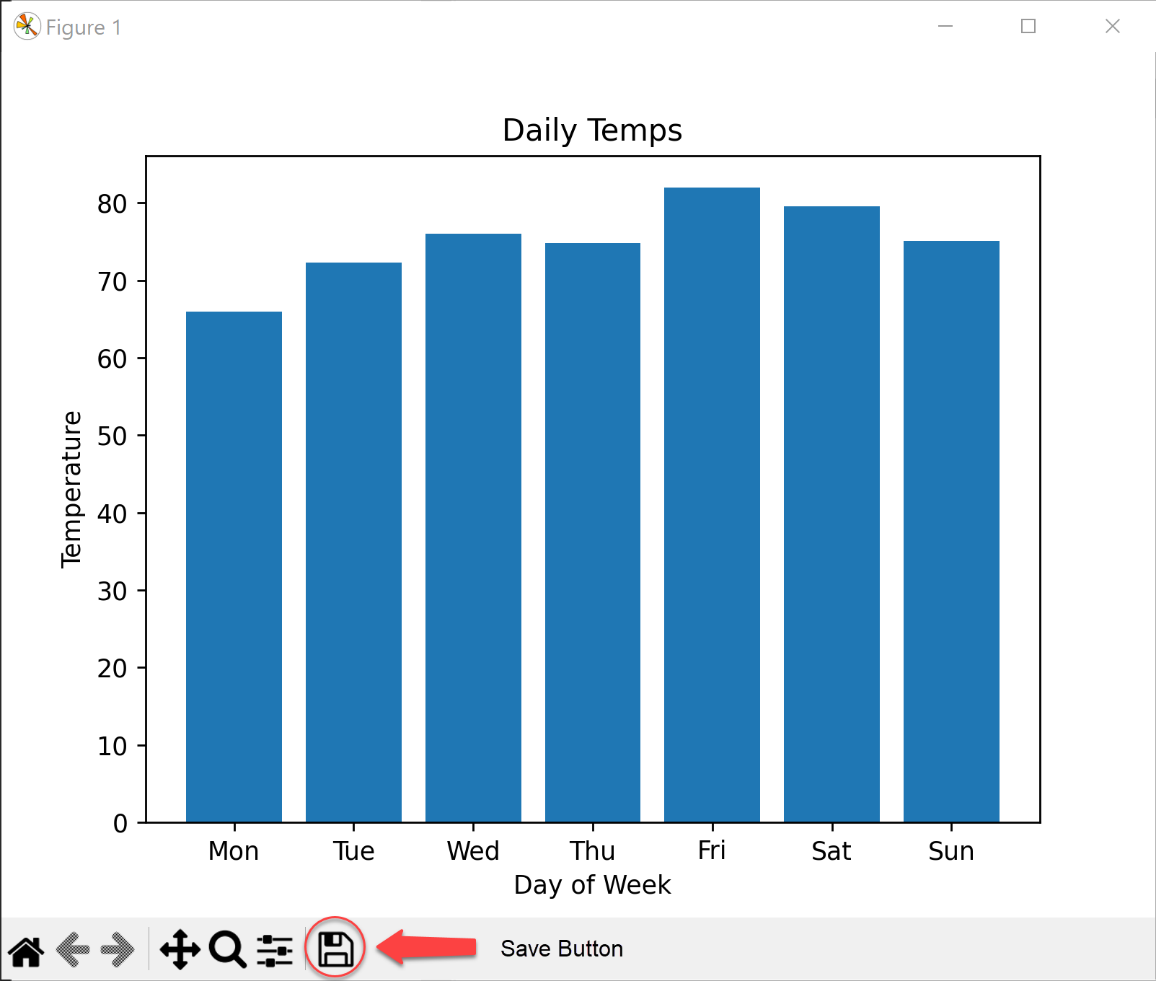
Enter the value for tick 5: 82

Enter the name for tick 6: Sat

Enter the value for tick 6: 79.6

Enter the name for tick 7: Sun

Enter the value for tick 7: 75.1



Run this program. First, try using the test data that you see in the Sample Output.

For a second run, **enter different test data that you come up with on your own**.

When your program displays the bar graph with your test data, **use the save button** to save the graph to a file named **Lab06P4-output.png**. DO NOT TAKE A SCREENSHOT.

Submit both files, **Lab06P4.py** and **Lab06P4-output.png**, to Blackboard for credit.

## Grading Rubric

### Grading rubric for Problem 1 (30 points)

* Program has a well-formatted and correct header [5 points]
* Program does execute correctly and produces correct results [20 points]
* Screenshots demonstrates student executed the faulty program and the corrected program [5 points]

### Grading rubric for Problem 2 (20 points)

* Program has a well-formatted and correct header [5 points]
* Program does execute correctly and produces correct results [10 points]
* Screenshot demonstrates student executed the program [5 points]

### Grading rubric for Problem 3 (25 points)

* Program has a well-formatted and correct header [5 points]
* Program does execute correctly and produces correct results [15 points]
* Screenshot demonstrates student executed the program [5 points]

### Grading rubric for Problem 4 (25 points)

* Program has a well-formatted and correct header [5 points]
* Program does execute correctly and produces correct results [15 points]
* Graphic demonstrates student executed the program using their own test data and knows how to use the save button [5 points]

## Problem 1 Explainer: Understanding Indexes in Lists

In Python, each item in a list has an assigned index value. It is important to note that Python is a zero-indexed based language. This means that the first item in a list has an index of 0.

For example, consider the following list:

my\_list = ['apple', 'banana', 'cherry', 'date']

The item 'apple' has an index of 0, 'banana' has an index of 1, 'cherry' has an index of 2, and 'date' has an index of 3.

### Generating Indexes in a Loop

When you loop over a list using a for loop in Python, you often do it like this:

for fruit in my\_list:

print(fruit)

In this case, fruit is the actual item from the list, not its index. However, sometimes you need to know the index of the current item. This can be achieved using the range() and len() functions, like so:

for i in range(len(my\_list)):

print("The item at index", i, "is", my\_list[i])

Here, i is the index, which we get from the range() function. len(my\_list) gives us the total number of items in the list. So, range(len(my\_list)) generates a sequence of numbers from 0 up to (but not including) the number of items in the list. In each iteration of the loop, i takes on the next value in this sequence.

### Working with Nested Lists

A nested list is a list that contains other lists as its elements. When dealing with nested lists, you have to use nested loops to access the items.

Consider a nested list:

nested\_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

In this list, there are three inner lists. To print each item, we can use a nested loop that generates indices:

for i in range(len(nested\_list)):

for j in range(len(nested\_list[i])):

print("Item at position", (i, j), "is", nested\_list[i][j])

Here, i is the index of the outer list (i.e., the index of each inner list), and j is the index within each inner list.

### Updating Items in Lists and Nested Lists

The reason we often need to generate indices when looping over a list is so we can update the items in the list.

For example, to add 5 to each number in a list, you might try this:

numbers = [1, 2, 3, 4, 5]

for num in numbers:

num += 5

However, this does not actually change the items in the list. To do that, you need to access each item by its index:

for i in range(len(numbers)):

numbers[i] += 5

Now, numbers[i] gives us a reference to the actual item in the list, not just a copy, so when we add 5 to numbers[i], it updates the item in the list.

The same principle applies to nested lists. To update an item in a nested list, you need to access it by its indices:

for i in range(len(nested\_list)):

for j in range(len(nested\_list[i])):

nested\_list[i][j] += 5

This will add 5 to each item in the nested list.

Remember: if you need to update the items in a list (or nested list), generate indices in your loop.