Lab 01 – CSCI 112

Due Date: Monday, August 31 at 5:00pm EST

Information

- This lab is intended to be completed **individually.**
- The files must be submitted with the exact file name provided in this file. If the file names do not match you will receive **zero** points for that file.
- Before you submit, make sure that your code runs. Any code which does not run without errors will receive **zero** points.
- Do not share your work with anyone other than Professor Khan. You may discuss algorithms, approaches, ideas, but **NOT** exact code.
- No late work is accepted. Make sure you are aware of the due date. A second past the due date **WILL** be locked out from submission.

Review

This lab is designed to review the basics of python as well as introduce you to the concept of coding efficiency. Read the following categories thoroughly, even if you are familiar with the concepts from CSCI 111.

Print method in Python: The `print()` method displays text to the terminal. Given no extra parameters beyond the desired print string, the method will buffer the output (it will wait to display to the screen until it has enough data). The print method also defaults to adding a newline after each print method invocation. To change these, named parameters can be used. The `end` parameter indicates what character to apply after the string, defaulted to `"\n"`. To change this, simply add `end="desiredCharacter"` to the print method invocation. To force the print method to be non-buffered, the `flush` parameter can be set to `True: flush=True` inside the print method invocation. Calling print by itself with no parameters will print a single new line character, moving the cursor down one line in the terminal.

String and Number Formatting: Python has a formatting style available to make numbers/strings look "pretty" while printing. This requires the `format()` method invoked on a string. The string to be formatted will indicate the space in which the formatted variable should go via `{}` and a letter indicating the type of data to be formatted.

An example of a string which formats a string, an integer and a floating point number could be:

```
"{:s} {:d} {:f}".format(name, age, weight)
```

This string can also be printed in the same line of code:

```
print("{:s} {:d} {:f}".format(name, age, weight))
```

These formatted variables can be padded with white space to left or right align the information as well as indicate how many decimal points to show. An example of right-aligning the variables in the previous code, which will pad the age variable to 10 spaces, the weight variable to 20 spaces, and limit the weight variable to 5 decimal points.

```
print("{:s} {:>10d} {:>20.5f}".format(name, age, weight))
```

There are more formatting options available with python which can be found in most python documentation sources.

Assignment

In this lab you will need to modify a single file, `main.py` which has been provided in the zip archive **lab_1.zip**. Inside this file will be code to demonstrate the differences between two possible approaches to creating a list of random numbers. Your assignment is to analyze the code and use critical thinking about the two approaches used. Run the code to see the output of the two approaches (note, this may take a while, sometimes longer than a full minute, to complete).

Task 1 - Random Number Lists [5 points]

First, take a close look at the two approaches provided to you, `randomNumbers` and `randomIndexes`. These functions attempt to create lists containing the numbers 0 through n-1, given an integer as a parameter, in random order. In a plain text file named **analysis.txt** (no word doc or otherwise), write a brief description of what each function is doing in English. Try to explain the approach as if someone did not understand coding terms. Also explain why these approaches are **inefficient** ways to code. Have one paragraph (or more) per approach, don't combine the analysis of the two into one paragraph. It's okay to get a bit redundant in your writing here. Include your name at the top of the file.

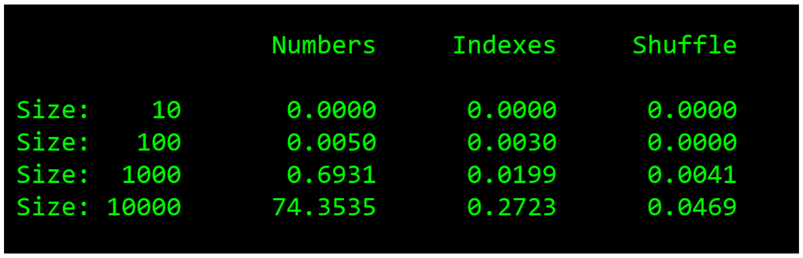
Task 2 - RandomShuffle() [5 points]

Add another function to main.py named `randomShuffle()`. This function will take a more nuanced approach to creating a random list by first creating an in-order list of numbers and then shuffling these numbers amongst themselves. The approach follows the following algorithm:

1. Create a list of numbers in order from 0 to n-1.
    a. This is easy with list comprehension, try to create this list with one line of code
2. For each index i:
    a. Calculate a random number between i and n-1
    b. Swap the values at the indexes i and this random number

This approach is guaranteed to give a random ordering as each individual number as an equal probability of being in any given index.

Use `showList()` to test your new function to be reasonably confident that the numbers are in a random order. Add `randomShuffle()` to the `main()` function to result in a printout similar to the following image:



```
                   Numbers      Indexes      Shuffle

Size:     10        0.0000       0.0000       0.0000
Size:    100        0.0050       0.0030       0.0000
Size:   1000        0.6931       0.0199       0.0041
Size:  10000       74.3535       0.2723       0.0469
```

Note that your runtimes may vary depending on your computer and random values. In a separate paragraph in **analysis.txt**, add a description of what `randomShuffle()` is doing (again, explain to a non-coder), and why it is better than the previous two approaches.

Task 3 – Dictionary Comparisons [5 points]

The files cities.csv contains a list of US cities and their zip codes. These cities are listed in an ascending order. In this task you are required to load the cities into a regular dictionary as well as an ordered dictionary and observe in which order elements are iterated over each.

a. First, without running the code, explain what you expect to see in your comparison? Add a few lines at the end of **analysis.txt**

b. Next, modify the code in **dict_comparison.py** to add the cities and their zip codes to a regular dictionary and an ordered dictionary. Use the city name as the key and the zip code as the value e.g. {"Nashville": 37201}. The current version of the code reads the file for you line by line and load the values into a list.

c. Finally add code to **dict_comparison.py** that iterates over each dictionary one by one. Are the elements for both visited in the same of different order? Explain in a few lines how this compares to your answer in part in **analysis.txt**.

What To Turn In

Create a zip file named **lab1_<your W&L ID>.zip**. Inside this zip archive should be **main.py, dict_comparison.py** and **analysis.txt**. No other files are needed.