CS2321 Lab 6

Lab Instructions:
Save the code you write for each exercise in this lab as a *library* -- that is, a textfile with a .py extension containing only executable python code (i.e. no angle-bracket prompts, etc). Name each file according to the exercise number (e.g. ex1.py, ex2.py, etc.) and save them to a directory containing the report file (in PDF), when completed, compress them together in a single zip file to be submitted on D2L.

Each function should have a docstring explaining what the function does.
Any Follow-up Questions and their Answers should be included in a **docstring** following the `main()` function.

e.g. the structure for a Python module should look like:

```
'''
   modulename.py
   Doc-string explaining what this module does
'''
# imports, such as math, random, etc., as needed

# Your code, includes definitions of classes, functions, etc.

def main():
   # Do what is needed.


if __name__ == "__main__":
      main()

'''
   Doc-string answering follow up questions
'''
```

Lab Deliverable: Once all your programs run correctly, collect their code and the results of their test-cases in a nicely-formatted **PDF** file exported from Word Processing document (e.g. MS Word or LibreOffice) to be included in the submission on D2L.

This **report** should consist of each lab exercise, clearly **labeled** in order, consisting of code, then copy/pasted text output, or, for GUI, screen-captured, of its four test-cases.
In this lab, take series of screen captures of your GUIs and insert them into the report.

Paired Programming:
We will work today's lab assignments in pairs -- on a single computer in one partner's account. One person will start out as the *typist*, the other as the *verifier*. These roles will switch.
For each problem, partners should decide upon their proposed algorithm to solve the given

problem *before* the typist begins to type. Sketch it out on a sheet of paper, perhaps.
For **ten** minute periods, the typist will type the code, while the other verifies and suggests corrections (typist has final decision). Under no circumstances may the verifier ever touch the mouse or keyboard. (Note: the *instructor* may not touch your input devices either!)
On the instructor's ten-minute signal, or your own timer, partners will trade responsibilities. This should allow both partners to benefit from each other's strengths. Future paired-programming labs will be with different partners, to spread the gained experience around.
Each partner should post the resulting code in their own D2L folder. You may transmit partnership-generated code to the other partner (only!) by email or thumb-drive.

Exercise

#1: The birthday paradox says that the probability that at least two people in a room will have the same birthday is more than half, provided n, the number of people in the room, is more than 23. This property is not really a paradox, but many people find it surprising.

Design a Python program that can test this paradox by a series of experiments on randomly generated birthdays, which test this paradox for n = 5, 6, 7, …, 50. To achieve a reasonably accurate probability, I suggest you repeat at least 1000 times for each n value. Refer to exercise #2 for generating a csv file to be imported to spreadsheet.

You don’t have to define classes for this exercise. However I expect to see a few functions:

A boolean function to detect same birthdays, a function to calculate the probability of same birthdays by repeated calling the previous boolean function 1000 times with different list of birthdays. Then a main function to go through n values from 5 to 50.

#2: Implement Listing 2.11 from the text (pg 75) to determine pop() and pop(0) timings for lists of different sizes. Modify the book's code to test list-sizes from one million to (20 million + 1), increasing in steps of one million (this should give you twenty lines of output).

If you follow the book's code exactly, you may end up with 100 lines of output. Either way, you should also write the output values to a comma-delimited text-file to make importing results into your spreadsheet program easier!

```
f = open("times.csv", "w")
then in a loop:
        f.write("%d, %f\n"%(size, time))
f.close()
```

Give a quick summary of your code. Explain what you are about to do, and hypothesize as to what the result will be.

Graph your results in an Excel file. Place list-size on the x-axis (horizontal), and time on the y-axis (vertical). It should then contain two lines (series).

What is the rate of growth of each of the lines as n increases?

Copy and paste the resulting graph into your final report. Make sure both axes and your legend are helpfully-labeled.

Provide an explanation for why the two series in your graph look as they do.