CS5001 HW1: Due on Canvas Sun Oct 1 at 11:59pm

You must work **either on your own or with one partner**. If you work with a partner, you and your partner must first register as a group with us then submit your work as a group on Canvas. To register, please send an email to the TAs-- Qing Chen (chen.qing1@northeastern.edu), Kejian Tong (tong.ke@northeastern.edu), Lingyu Hu (hu.lingyu@northeastern.edu), Mingtianfang Li (li.mingt@northeastern.edu).

You may discuss background issues and general solution strategies with others, but the programs you submit must be the work of just you (and your partner). We assume that you are thoroughly familiar with the discussion of academic integrity that is on the course website. Any doubts that you have about "crossing the line" should be discussed with TAs or the instructor before the deadline.

Assignment Objectives. Mastering the assignment statement and conditional execution. Distinguishing between types and values. Practicing with strings and string slicing. Using print, str, int, float, len, import, input, and input. You will get experience bridging the gap from math formula to Python code. You will get practice processing strings according to given rules. Finally, your ability to manipulate files and directories and the Python editor will be enhanced. The assignment is based on Lectures 1,2.

1 The Golden Ratio (50 points)

Given a positive integer n, the n-th Fibonacci number is given by:

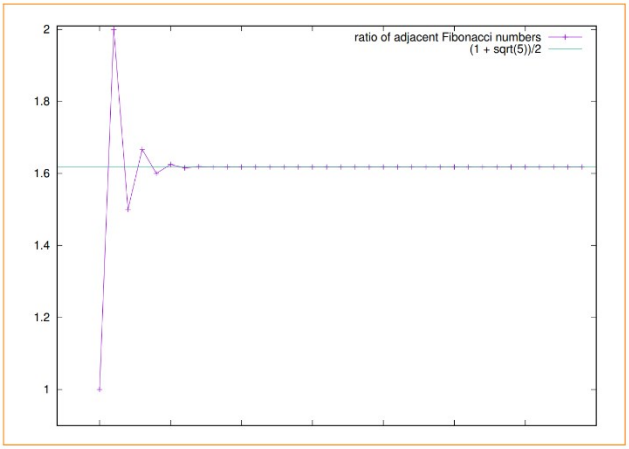
$$f_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right).$$

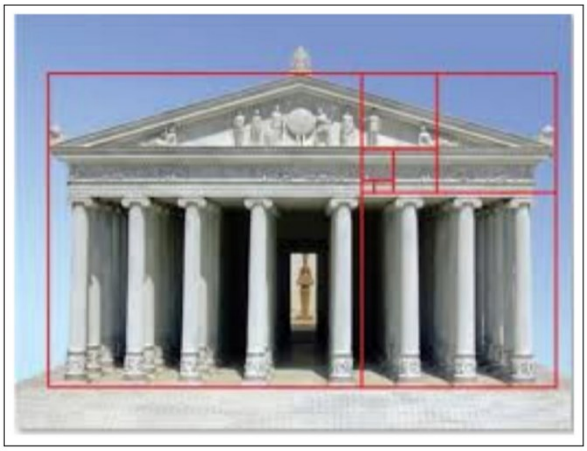
Despite the complexity of the right-hand side, the Fibonacci numbers are integers:

n	1	2	3	4	5	6	7	8	9	10	...	20	21	22	...
f_n	1	1	2	3	5	8	13	21	34	55		6765	10946	17711	

Side note: each Fibonacci number is the sum of its two predecessors, a fact that we will play with when we get to the topic of iteration.

The ratios $r_n = f_{n+1}/f_n$ are interesting, and shown on the left below:





As n increases, the ratios look more and more like the golden ratio ϕ = (1+√5)/2. (It's practically impossible to see, but the y values are still changing in the fifth decimal place halfway through the sequence shown above.) The rectangle whose length-to-width ratio equals ϕ is (according to the Ancient Greeks) the most visually appealing of all rectangles, as hinted at above right.

Lucas numbers: The Lucas sequence has the same recursive relationship as the Fibonacci sequence, where each term is the sum of the two previous terms, but with different starting values. The first few Lucas numbers are:

2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207, 3571, 5778, 9349,…

As with the Fibonacci numbers, each Lucas number is defined to be the sum of its two immediately previous terms, thereby forming a Fibonacci integer sequence. The first two Lucas numbers are $L_0 = 2$, $L_1 = 1$. More information about Lucas numbers is here. The first 200 Lucas numbers are listed here. You can use this to check/test your implementations.

The n^{th} Lucas number can be derived using Fibonacci numbers as follows:

$L_n = 2F_{n+1} - F_n$

Or,
$L_n = F_{n-1} + F_{n+1}$

In the above, F_n represents n^{th} Fibonacci number. In your Python implementation, use any of the above relations to derive the n^{th} Lucas number.

Your first task. Write a Python script Golden.py that solicits a positive integer *n* using the input command and then produces a two-line table. The first line should display *n*, f_n , r_n, and L_n. The second line should display *n + 1*, f_{n+1}, and r_{n+1}, and L_{n+1}. For example, if the input value for *n* is 35, then this should be displayed:

35	9227465	1.618033988749890	20633239
36	14930352	1.618033988749897	33385282

Please ensure that the displayed data is nicely aligned and informative. Use the print formatting methods discussed in class. The number of blanks in between the columns is not important as long as the all the 8 numbers are neatly displayed. Since square roots have to be computed, your code will have to import that function from the math module.

Problems arise if the input value for *n* is too big. This will be explained later in the course. So to keep things simple in this assignment we require $n \leq 35$. The prompt in the input statement that solicits *n* should make this restriction (and that the input be a positive integer) clear to the user. Remove any print statements that were part of your debugging strategy and submit your implementation of Golden.py to Canvas.

Rubrics

Your code will be tested against several test cases. A correct implementation is expected to answer all the test cases correctly.

1. Correct values of n, n+1 will fetch **1 point each**.
2. Correct values of f_n, f_{n+1} will fetch **8 points each**.
3. Correct values of r_n, r_{n+1} will fetch **8 points each**.
4. Correct values of L_n, L_{n+1} will fetch **8 points each**.

2 LOL? FWM! (50 points)

This problem lets you probe how quickly you super-texters can enter a specified length-3 string. Code from the random module is used to generate a test string. The test string is displayed and code from the datetime module is used to time how long it takes for you to "re-enter" the test string.

To get started, download the skeleton script LOL.py that is available on the assignments page of the Canvas website.

The only thing you need to know about the workings of this script is what it assigns to the variables S1, S2, t1, and t2. This is what happens:

a) the value of S1 is the random length-3 test string.
b) the value of S2 is your response (a string). It can be anything.
c) the value of t1 is the initial timestamp (a string). It encodes the exact time when you are first shown the test string.
d) the value of t2 is the final timestamp (a string). It encodes the exact time when you finish typing in your response.

Play with this script so that you get a sense of what it does. You will first be prompted as follows:

Press the return key when you are ready.

As soon as you do that the test string is generated and displayed and then you are asked to respond:

The test string: tpq
Enter the test string as fast as you can:

At this point the “clock is ticking.” You enter your response and after that, the two timestamp strings are displayed:

The test string: tpq
Enter the test string: tpq

2023-02-01 13:48:44.453000
2023-02-01 13:48:46.965000

The timestamp strings encode a date and time. All we care about is the “seconds slice” which begins in position 17 and extends to the end of the string. In this example, the subtraction

46.965000 − 44.453000 = 2.512000 (1)

reveals that your response took 2.512 seconds. In this problem you are to add code to the end of LOL.py so that it exhibits this behavior:

1) It prints the message “Correct response.” if your response is exactly the same as the test string. **(10 points)**
2) It prints the message “There is a character mismatch in your response.” if your response is length-3 but does not agree with the test string. **(10 points)**
3) It prints the message “Your response has the wrong number of characters.” if your response is not a length-3 string. **(10 points)**
4) It prints the elapsed time (in seconds) to three decimal places using information in the two time stamps. **(20 points)**

As you organize the processing of (1), (2), and (3) keep in mind that the order of operations in computing is very important.

Regarding the elapsed time, it is not always as simple as the subtraction in equation (1) suggests. Suppose we have these two timestamps:

2023-02-01 13:48:44.453000
2023-02-01 13:49:06.965000

Elapsed time is NOT given by 6.965 - 44.452. Figure out how to deal with this situation so that your finished implementation of LOL.py reports the correct elapsed time. Assume that the elapsed time is < 60 seconds. This means that you don’t have to worry about what your program does if someone takes more than a minute to answer.

Here is a sample dialog that indicates how the “correctness” output line and the elapsed time output line should be formatted:

The test string: xwo
Enter the test string as fast as you can: xwoo

2023-02-01 15:05:10.664000

2023-02-01 15:05:14.939000

Entered the wrong number of characters!
Elapsed Time = 4.275 seconds

Be sure to comment your code. Submit the finished version of LOL.py to Canvas. It is important that you remove all print statements that were part of your debugging strategy.

What to submit?

1. Prepare a README.txt file containing the following:

a) Include a quick summary of how you run your program Golden.py.
b) Include a quick summary of how you run your program LOL.py.
c) If any of the programs is not working, include what is the issue in your opinion and how would you fix it if you had more time?

2. Submit your files Golden.py, LOL.py, README.txt inside a single zip file on Canvas.

Additional directions

Following directions apply to both the problems above. Negative numbers indicate the penalty to be applied on your final score for a problem/direction if the directions are not followed.

1. Start your programs with a docstring (comment) summarizing what it is doing, what are the inputs, and the expected output. (-5 points)
2. The docstring at the beginning also needs to include your name (both teammates if applicable), date, and your email ids. (-5 points)
3. For every variable defined in your program, include a brief comment alongside explaining what it stores. (-5 points)
4. README.txt file needs to be submitted as described above. (-10 points)
5. Submit all the files (.py and .txt files) as a single zip folder/file on Canvas. (-5 points)