CSCE 110: Programming I

Lab #12 (100 points)

Due: Sunday, November 20 by 11:59pm

1 Please make sure you understand the following.

For this assignment, you are allowed to use anything we have discussed in class. Please make sure to name your files correctly and incorporate user-defined functions. Do not use the `global` keyword in your programs. Otherwise, your programs will be penalized as stated in the grading rubric.

Please label your Python programs `q<num>.py`, where `<num>` is the question number. Take your time and make sure you understand everything in this lab before getting started. Also, make sure your programs match the output EXACTLY as given for each question.

2 Course Evaluations

Please take a few minutes during lab to complete the evaluation for this course. The link to the evaluation form is `http://pica.tamu.edu`. Your evaluations are anonymous. Survey results will not be released to instructors until final grades have been submitted. While surveys do take time to complete, please know that I do appreciate your effort in completing evaluations and surveys related to this course.

3 Lab Questions

1. *Sudoku solution checker.* Write a program (called `q1.py`) that determines whether a Sudoku solution is valid or invalid. In Sudoku, the objective is to fill a 9×9 grid with digits so that each row, each column, and each of the nine 3×3 blocks contains all of the digits from 1 to 9 (see Figure 1 below).

5	4	6	8	9	3	2	1	7
3	9	2	1	7	4	6	8	5
8	1	7	5	2	6	9	3	4
7	6	9	3	4	1	8	5	2
2	3	4	7	8	5	1	9	6
1	8	5	2	6	9	7	4	3
4	5	8	6	1	2	3	7	9
9	2	1	4	3	7	5	6	8
6	7	3	9	5	8	4	2	1

Figure 1: Sudoku solution.

The solution will be stored in a file containing 9 lines, where each line represents the solution for a row. For example, consider the file `soln1.txt` below, which contains the solution for the puzzle shown in Figure 1.

soln1.txt

```
546893217
392174685
817526934
769341852
234785196
185269743
458612379
921437568
673958421
```

Since `soln1.txt` represents a valid solution, your program will output "`Valid solution`." If a solution is invalid, your program will output "`Invalid solution`."

Your objective. Sample code has been provided for you in `q1.py`. The provided code (i) asks the user for a file, (ii) stores the data from the file in a nested tuple of tuples, and (iii) prints the results. Your job is to write the remaining pieces of the code to verify the solution by checking the nine rows, nine columns, and nine 3x3 blocks. Thus, the functions you will write are `valid_rows(soln)`, `valid_cols(soln)`, and `valid_blocks(soln)`, respectively.

Several files have been provided to test your program. But, you should also create your own test files to test the correctness of your code. Also, pay attention to the comments in the provided code.

Programming tips. For each function, print the contents of the variable `soln` (which is a nested tuple of tuples) to see the data you are manipulating. Next, draw a picture of the `soln` data structure before writing your code. Once you understand how the data is organized, you are ready to write the code.

Example #1. The user enters the name of the Sudoku solution file (line 1). The program then determines that the `soln1.txt` file is a valid Sudoku solution (line 3).

```
1 Enter filename: soln1.txt
2
3 Valid solution
```

Example #2. The user enters the name of the Sudoku solution file (line 1). The program then determines that the `soln3.txt` file is an invalid Sudoku solution (line 3).

```
1 Enter filename: soln3.txt
2
3 Invalid solution
```

2. *Credit card interest.* Write a Python program (called `q2.py`) that determines the number of months required to payoff a credit card balance—assuming no other purchases are incurred on the card. You will need to import the `math` module for this program. In particular, you will find the `math.log()` function to be useful.

   Consider the following variables.

   - b = credit card balance
   - n = number of months to payoff balance
   - p = monthly payment (in dollars)
   - r = effective (monthly) interest rate

   Then, the number of months, n to payoff the balance is $n = \frac{-\log(1-\frac{br}{p})}{\log(1+r)}$. As an example, suppose you have a credit card with an 18% annual percentage rate (APR). With the card, you made a purchase costing \$350. As a result, $b = 350, r = 0.18/12$, and assume $p = 10$. Substituting these numbers into the above equation gives $n = 50$. That is, assuming a monthly payment of \$10, it will take over 4 years to pay off a purchase of \$350.

   In your Python program, the user provides the following input: credit card balance (b) and annual percentage rate. Your output will show for each monthly payment (p) between 1% and 25% of the balance b, the number of months required to payoff the credit card.

Note. When computing logarithms, the result is undefined for 0 and negative numbers. For example, log(0) and log (-1.9) are undefined. Therefore, you will need to check for this condition in your program.

Example #1. At the prompt, the user enters a credit card balance of 1000 and an APR of 10.09 (lines 1 and 2). The program then shows the number of months required to payoff the balance for monthly payments between 1% and 25% of the credit card balance (lines 4–31).

```
1  Credit card balance ($): 1000
2  Annual Percentage Rate (%): 10.09
3
4  ----------------------------
5  %    Months to payoff balance
6  ----------------------------
7  1    219.5
8  2    65.1
9  3    39.3
10 4    28.2
11 5    22.0
12 6    18.0
13 7    15.3
14 8    13.3
15 9    11.7
16 10    10.5
17 11    9.5
18 12    8.7
19 13    8.0
20 14    7.4
21 15    6.9
22 16    6.4
23 17    6.1
24 18    5.7
25 19    5.4
26 20    5.1
27 21    4.9
28 22    4.7
29 23    4.4
30 24    4.3
31 25    4.1
```

Example #2. Similar to Example #1, but with a credit card balance of 759.13 and an APR of 18.99.

```
1  Credit card balance ($): 759.13
2  Annual Percentage Rate (%): 18.99
3
4  ----------------------------
```

```
%    Months to payoff balance
----------------------------
1    <Undefined>
2    99.8
3    47.7
4    32.1
5    24.2
6    19.5
7    16.3
8    14.0
9    12.3
10    11.0
11    9.9
12    9.0
13    8.3
14    7.6
15    7.1
16    6.6
17    6.2
18    5.9
19    5.5
20    5.3
21    5.0
22    4.8
23    4.5
24    4.3
25    4.2
```