CSCE 120 Assignment 1 Fall 2017

Name(s) ____

Instructions

1. Collaboration is highly encouraged. Though you may work on this assignment alone, you are encouraged to complete this assignment with a single partner according to the following guidelines:
   - You must work on *all* problems *together*. You must both put in an equivalent effort.
   - You are encouraged to discuss (at a high level) any problems or issues with other individuals or pairs, but you may not directly share code with them.
   - To foster true collaboration, you should use one computer: the person typing is the "driver" while the other person is the "navigator" and should direct the driver. Trade off these roles every 15 minutes or at least every other exercise.
2. Hand in all your source code files through webhandin
   - For convenience, place each program into its own folder with appropriate file name(s)
   - Place all of your files into one folder/directory and zip it up. Turn in the ZIP archive file
3. Be sure to rigorously test your programs with sufficient input examples and test cases.
4. Be sure to thoroughly read this rubric and understand how we will evaluate your program.
5. Printout and hand in a hardcopy of this rubric (first page is sufficient) with you name(s) on it.

Question	Points	Score
1	25	
2	25	
3	25	
Total:	75	

Programs

1. (25 points) In sports, the *magic number* is a number that indicates the combination of the number of games that a leader in a division must win and/or the 2nd place team must lose for the leader to clinch the division. The magic number can be computed using the following formula:
$$G + 1 - W_A - L_B$$
where G is the number of games played in the season, W_A is the number of wins the leader currently has and L_B is the number of losses the 2nd place team currently has.

   Write a program that prompts the user to enter:

   - G, the total number of games in the season
   - W_A the number of wins of the leading team
   - L_A the number of losses of the leading team
   - W_B the number of wins of the second place team
   - L_B the number of losses of the second place

   Your program will then output the current win percentages of both teams, the magic number of the leading team as well as the percentage of the remaining games that must go in team A's favor to satisfy the magic number (for this, we will assume A and B do not play each other).

   For example, if a user enters the values 162, 96, 58, 93, 62, the output should communicate the following information.

```
Team Wins Loss Perc   Magic Number
A    96   58   62.34% 5
B    93   62   60.00%
With 15 total games remaining, 33.33% must go in Team A's favor
```

2. (25 points) Radioactive isotopes decay into other isotopes at a rate that is measured by a half-life, H. For example, Strontium-90 has a half-life of 28.79 years. If you started with 10 kilograms of Strontium-90, 28.79 years later you would have only 5 kilograms (with the remaining 5 kilograms being Yttrium-90 and Zirconium-90, Strontium-90's decay products).

   Given a mass m of an isotope with half-life H we can determine how much of the isotope remains after y years using the formula,
$$r = m \cdot \left(\frac{1}{2}\right)^{(y/H)}$$
   For example, if we have $m = 10$ kilograms of Strontium-90 with $H = 28.79$, after $y = 2$ years we would have
$$r = 10 \cdot \left(\frac{1}{2}\right)^{(2/28.79)} = 9.5298$$

kilograms of Strontium-90 left.

Write a program that prompts the user for an amount m (mass, in kilograms) of an isotope and its half-life H as well as a number of years y and outputs the amount of the isotope remaining after y years. For the example above your output should look something like the following.

```
Starting with 10.00kg of an isotope with half-life
28.79 years, after 2.00 years you would have
9.5298 kilograms left.
```

3. (25 points) When taking out a loan for a given *principle*, a customer pays a certain percentage of interest (APR) over a certain *term* (number of months). These determine a fixed payment that the customer will pay on a monthly basis. In each monthly payment, a certain amount is applied to the accrued interest and the rest toward the principle.

   The monthly payment amount can be determined by the following formula:

$$m = p\left(\frac{r}{1-(1+r)^{-t}}\right)$$

   where

   - m is the resulting monthly payment
   - p is the original principle amount borrowed
   - t is the term (in number of months)
   - r is the monthly interest rate which is the APR divided by 1200 (the APR is on the scale $[0, 100]$ so we divide it by 100 then divide by 12 to get the monthly rate)

   In addition, we can compute the *true* cost of a loan which is the amount that the customer pays in interest over the life of the loan. This is simply the total amount of all payments less the original principle.

   Write a JavaScript program that prompts the user for the necessary inputs. It should then compute the monthly payment amount and the true cost of the loan. Output a summary of the loan to the console or in an alert box. Take care: if the user enters invalid data, instead of a summary, you should output an appropriate error message instead.

   For example, if a user inputs $p = \$10,000$, $t = 60$ (5 years), and $r = 4.9\%$, the output could look *something* like the following.

```
Loan Amount:     $10000
Months:           60
APR:              4.9%
Monthly Payment: $188.25
Total Cost:      $11295.00
True Cost:       $1295.00
```

Rubric

When we grade your assignment, we will be assessing it based on the items below.

Following Instructions

- All required soft-copy files handed in via webhandin
- Correct file name(s) and organization
- Any required hardcopies (this rubric) handed in
- Programs successfully execute and there are no formatting or syntax errors

Style

- Appropriate variable and function/method identifiers
- Style and naming conventions are consistent
- Good use of whitespace; proper indentation
- Clean, readable code
- Code is well-organized

Documentation

- Well written comments that clearly explain the purpose of each non-trivial piece of code
- Comments explain the "what" and "why"
- Comments are not overly verbose or overly terse
- Code itself is "self-documenting"; explains the "how"

Program Design

- Code is well-organized and efficient
- Code is modular; substantial pieces of it could be reused; few redundancies
- Code is easily understood and maintainable
- It is clear that sufficient testing has been performed
- Corner cases and bad input have been anticipated and handled appropriately

Program Correctness

- Source code compiles, executes as expected
- Program runs as specified: correctly reads any input; correctly formatted output
- Test cases successfully execute