Project PL4

Due Date: November 23

Purpose

In this project, you will write a program in Scheme. This will give you a chance to see how to program using a functional language and to think of your data structures in terms of lists.

This is an individual project.

Problem

In this project, you will write a calculator app in Scheme. Quite simply, the program will evaluate a mathematical expression as a calculator would... more or less...

Input

All of the input will be done interactively by the user via the keyboard. As a functional language, you will just call the function `compute` with a list argument representing a mathematical expression (see below). The output will simply be the result.

The math operators, from highest precedence to lowest, are as follows. Operators within a category are evaluated left-to-right.

Operator(s)	meaning
^	exponentiation
*, /	multiplication, division
+, -	addition, subtraction

Except for the list itself, there are **no** parenthesis in the input. (That's one less thing to worry about!)

Output

The output will simply be the result of the mathematical expression, **following the precedence rules above.** However, the *form* of the output can be either integer or floating point, depending on what Scheme does normally; that is, you do not have to do any special numeric conversion. The output will be whatever the input and Scheme specifies.

Input and Output Examples

What follows are input/output examples of the `compute` function in action in Scheme:

```
1 ]=> (compute '(4 * 5 + 3))
;Value: 23

1 ]=> (compute '(4.0 * 5 + 3))
;Value: 23.

1 ]=> (compute '(3 + 4 * 5))
;Value: 23

1 ]=> (compute '(1 + 2 * 3 ^ 2 - 4))
;Value: 15
```

```
1 ]=> (compute '(1 + 2 * 3 ^ -2 - 4))
;Value: -25/9

1 ]=> (compute '(1 + 2 * 3.0 ^ -2 - 4))
;Value: -2.7777777777777777
```

You can test your answer against Scheme's by typing the expression in the proper prefix order:

```
1 ]=> (- (+ 1 (* 2 (expt 3.0 -2))) 4)
;Value: -2.7777777777777777
```

Specifics

- The program must be written in Scheme.
- The program must not have any loops; all repetition should be done with recursion.
- You can have as many functions as you want.
- The result should be returned as a number (not a list; see above).
- You do not have to do any error checking. You may assume that the user will input the expression correctly, and you do not have to check for undefined operations (such as division by zero).

Notes

- As in previous projects, include a comment at the top of the program that includes:
  - the name of the program
  - your name
  - a description of the program's functionality
  - an exact description of valid input
  - an exact description of what is output
- You should have a one-line comment describing each function.
- Much (all?) of this solution can be found online. While you can look up Scheme functions and meanings, copying an (almost) entire program is a violation of the Honor Code. By simply copying, you also won't learn how to program in Scheme, which will have consequences on the final exam.
- Submit your source code via Canvas before 11:59:59 PM on the due date. Name your file *yourLastName*`PL4.scm`, as in `gousiePL4.scm`. Turn in a printed version of your program (it will be pretty short) in class the next day. Include the Honor Code on what you turn in.

The answer is either `m` *or something else.*
– Eva Ma, one of my grad school professors