You should **submit** an single electronic package that contains the **source** files for your work on the programming questions, by following the submission procedure described in class and on the class discussion forum. *Using the **discussion forum** to clarify details of both the main program and the submission format and procedures is an important part of this homework.*

You are welcome to use any inanimate **resources** (e.g., books, Web sites, publicly available code) to help you with your work. However, *all such help must be clearly noted* in your submissions. Further, no matter what you use, *you must be able to explain, in detail, how it works*. (You may be called upon to explain your homework individually.) Refer to the class policy for details, and ask for clarifications if you are unsure if something is allowed.

Your implementation must use **clean, portable Python 3** that minimizes dependencies on OS, version (beyond 3.x), etc. (If in doubt, please ask.) Packaging and documentation of code are worth a very significant portion of the grade. This homework centers on **extending the** `calc.py` example, which is included with PLY and has been discussed extensively in class, as follows.

1. It should support *div* and *mod* operators with the usual semantics with operator tokens `//` and `%`, respectively. (This extension has been almost fully provided by classroom work.)
2. In addition to scalar variables and literals, it should support lists of scalars. The list constructor uses Python-like syntax. For example, if `x` has value 1, then `(3, 1, 4,)` and `(3, 1, 4)` (note the missing comma) construct the identical lists, composed of the three integers 3, 1, 4 in that order. The motivation for the optional terminal comma is the same as that in Python. In the output, a list should be represented in the second form (including trailing comma) with exactly one space after each internal comma. The calculator uses duck typing like Python, so the same variable may store a scalar and a list at different times.
3. All operators (including those in the earlier extension) and the assignment statement should be extended to lists. A unary scalar operator is extended to a list by applying it element-wise to the list's element. A binary operator is similarly extended to a pair of list operands by applying it to the corresponding (in order) pairs of the lists. You may assume that if one operand of an operator is a list, then so is the other. (Catching and reporting such errors and others is encouraged but not required.)

Input-output: The program should read its input from the *standard input* stream and write its output to the *standard output* stream. Optional diagnostics may be written to the *standard error* stream. It is very important that the program read its input only from the standard in put stream and that it write nothing except the specified output to the standard output stream. In particular, there should be no prompts or informational messages printed to standard output.

The **input** consists of the calculator language of calc.py as discussed in class, extended

[1] Updated 2023-02-21.

to support the above features. The **output** consists of (only) the values of stand-alone expressions (excluding expressions that are part of an assignment statement) in the input. The output for a statement must be produced as soon as the statement appears in the input stream (before waiting for or reading any further input that may appear). There is exactly one (i.e. a unique) output for any given input (but there may be multiple inputs resulting in the same output). If two outputs for the same input differ by even a single character/byte then at least one of them is incorrect. In the output, lists should have a trailing comma only when strictly necessary (i.e., for singleton lists). Each comma except the trailing one (if present/required) should be followed by exactly one space. The following sample input and output illustrates some of these details.

Sample input:

```
2 + 3
hello = (2 + 3) * 7
y = 4 * hello + 3
y
y = (2, 3, hello)
y
hello = 2 * y
hello
hello = hello / y
hello
y = (17 % 5) * y + (6, 0, 2)
y
z = (1) + (2)
z
z = (1,) + (2,)
z
emptylist = ()
emptylist + emptylist
x = ((2) + (1984 % (9 // 4)),)
x
x = ((2) + (1984 % (9 // 4)))
x
x = (x+3*x, (17*x + 3) % (50 // 8), 3*x+1)
x + x * (3, 1, 4)
```

Sample Output: (The symbol ␣ denotes a single space and each line is terminated by a single newline character. There are no blank lines in the output, nor anything else beyond the following.)

```
5
143
(2,␣3,␣35)
(2,␣3,␣35,␣2,␣3,␣35)
(1.0,␣1.0,␣1.0)
(8,␣3,␣37)
3
(3,)
()
(2,)
2
(32,␣2,␣35)
```