Extended deadline: April 5, 2019, 11:59pm

Due: Friday, March 29, 11:59PM

We'll be spending the next few assignments implementing a programming language, which will be called SBML. For this assignment, we will be implementing an expression evaluator. This takes an expression as input, evaluates it to a result, and prints the result to standard output.

We will use the parser generator Ply: Python Lex and Yacc

You can install ply with:

pip install ply

OR

1. Download Python Lex and Yacc ply from: http://www.dabeaz.com/ply/

2. Download and unzip.

3. Install with:

python3 setup.py install

4. Read the Web site documentation and try the examples on the Web site.

5. Use the PLY mailing list to ask questions.

Our Language:

\* Data Types: our language has the following data types:

\*\* Numbers: Integer and Real = should be implemented using the Python integer/real types.

\*\* Tuples: finite sequences of arbitrary but fixed length within parentheses, where different components need not be of the same type

\*\* List = can be implemented using a Python list.

\*\* String = sequences of characters within matching single or double quotes in a single line. Strings should be implemented using the equivalent Python String type. When you print a string, you can use the python print statement.

\*\* Boolean.

Each type has a literal representation:

- Integer (positive (no sign) or negative (unary -) decimals). An integer is represented by one or more digits (0-9). For example, 42 and -42 are integer literal.

- Real. A real is represented by zero or more digits (0-9) followed by the decimal point "." and zero or more digits (0-9) (with the exception that  "." (period by itself) is not a real). For example, 3.14159 is a real literal. They may also contain exponents followed by positive or negative integers (like, 1.234e-2 is a real).

- String. A string literal starts with a double or single quotes, may be followed by zero or more non-quote characters, and ends with a matching quote. The value of the string literal should not include the starting and ending quotes. An example string literal is "Hello SeaWolf".

- List. A list literal consists of a square bracket, followed by a comma-separated list of zero or more expressions. For example, [307, "307", 304+3] is a list literal.

\*\* Boolean: True or False (like in python).

Operators: the following operators are listed in precedence order, from highest to lowest (all associative operators are left-associative, except \*\* (power), which is right associative):

Operator          Description

( expression )  A parenthesized expression.

( expression1,  expression2, ... )  creates a tuple.

#i(tuple) returns the argument at index i in the tuple. Indexes start at 1 like in SML.

a[b]      Indexing. B may be any expression.

a \*\* b  Exponent Performs exponential (power) calculation on operators = a to the power b. 2\*\*3\*\*4 = 2\*\*(3\*\*4) = 2417851639229258349412352

a \* b, a / b, a div b, a mod b    Multiplication and division overloaded for integers and reals, quotient division for integers, remainder for integers (divides left hand operand by right hand operand and returns remainder).

a + b, a - b       Addition and subtraction overloaded for integers, reals, strings and lists.

a in b   Evaluates to true if it finds a variable in the specified String or List and false otherwise.

h::t       adds its first argument to the front of the second argument list.

a < b, a <= b, a == b, a <> b, a > b, a >= b     Comparison (works for numbers and strings like in python).

not a    Boolean NOT.

a andalso b      Boolean AND.

a orelse b         Boolean OR.

The operators have the following semantics:

- Indexing: B must be an integer, a must be either a string or a list. If a is a string, returns the b-th character of the string as a string. If a is a list, returns the b-th element of the list (whatever type it has). The index is zero-based, so the first element is at index 0. If the index is out of bounds, it is a semantic error.

- Addition: A and B must be either both numbers, both strings or both lists. If they are integers or reals, then addition (with standard semantics) is performed. If they are both strings, than string concatenation is performed. If they are both lists, then concatenation of the lists is performed.

- Multiplication, Division and Subtraction: A and B must be integers or reals. For division only, B must not be 0. These are performed using standard multiplication semantics.

- Comparison: A and B must be integers, reals or strings. The two values are compared, and the result is True if the comparison is True, and False if the comparison is False.

- Boolean AND, OR, NOT: A and, if present, B must be booleans.

Your program will be called with a single argument. This argument will be a file containing expressions, with one expression per line ending with semicolon. You should process each expression and print one of three possible outputs:

- If the line contains a syntax error, you should print out: SYNTAX ERROR.

- A semantic error occurs when the line does not contain a syntax error, but one of the "semantics" conditions given above is violated when evaluating it. If the line contains a semantic error, you should print out: SEMANTIC ERROR.

- Otherwise, you should evaluate the expression, and print the result.

An example input file might look like:

1 - 2 + 3

1 2

42 + "Red"

1 - (2 + 3)

"Hello" + " " + "SBML."

[[1], 2, 3][0][0] + 40

The output from this file should look like:

2

SYNTAX ERROR

SEMANTIC ERROR

-4

Hello SBML.

41

This will be the first of several projects that uses this code, so it will do you well to make sure your code is reusable (you should use object-oriented programming).

Your program will be run with a command like:

python3 sbml.py inputFileName.txt

Your program must take input from the file specified on the command line and print the results in the console: one output line corresponding to every input line.

You should include your name and ID as the first line (commented out) in your python file. Your program must not produce spurious output.

Submit only the python file archive into a sbml.zip on Blackboard (and nothing else!!). Blackboard does not allow you to upload python files, so you must archive it into a sbml.zip file. You cannot use multiple files for this assignment, so your entire code should be in a single .py file.

Grading: There are a total of 50 points. Your program will be run against several complex use cases using similar expressions to the sample input. If it generates a correct result you will get the points associated for each use case (if the solution is incorrect, you will loose the points for that use case).