lisp-interpreter

This project is a simple Lisp interpreter written in Python, to the spec of New Mexico Tech's 2026 CSE3024 Project 2 assignment.

Running the Interpreter

You may run the interpreter with the commands `make`, `make run`, or `python repl.py`. To supply an alternative location for the output file, specify it as a command-line argument. By default, the output file is `output.txt`.

Completeness

The interpreter's support and completeness is outlined below:

Lisp Construct	Status
Variable reference	Done
Constant literal	Done
Quotation	Done
Conditional	Done
Variable definition	Done
Function call	Done
Assignment	Done
Function definition	Done
Arithmetic operators	Done
Integer type	Done
`car` and `cdr`	Done
`cons` support	Done
sqrt, pow	Done
Comparison expressions	Done
Logical operations	Done
`mapcar` function	Done
Lambda support	Not

Implementation Notes

Data Types

Data types were straightforward to implement, as I opted for a Python-native approach to data representation, with the actual back-end values being simple heterogeneous lists of lists, `int`s, `str`s, `float`s, and `bool`s.

Mathematical Operators

All the built-in functions follow a similar principle, sifting through a match case of built-ins before checking for user defined functions. The mathematical operations, as well as the logical expressions, all simply check for correct parameters, then return the result of the native Python operation on the arguments.

Environment Management

Variables and function definitions make use of the `Lisp` class's `env` property, a simple hashmap relating a symbol name to a value, for variables, or for a simple `Function` struct, for functions. When the function is executed, the `Lisp` instance recursively creates a new interpreter instance so as to offer the passed parameters to the expression within the function.

List Manipulation

List manipulation was simple in this implementation, largely due to Python's loose typing and highly dynamic lists. `car`, `cdr`, and `cons` only needed simple list appending and concatenation.

References

> Make-a-Lisp by `kanaka` on GitHub

This resource proved to be less applicable than I had originally hoped, and in practice I only benefitted from the birds-eye understanding given by the REPL data-flow diagrams, and a starting point for the tokenization regex. The rest of the project was much more thorough than was useful to me, and the rest was through trial-and-error.