A LISP Interpreter in Python

Brady Spencer
New Mexico Institute of Mining and Technology
Department of Computer Science
Socorro NM United States
brady.spencer@student.nmt.edu

1. INTRODUCTION

The LISP interpreter for this project was made in Python. I chose Python for multiple reasons. Python is much less verbose than Java, and the ease with which things can be made dynamic makes it a great choice for interpreting LISP.

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	Done
Car and cdr	Done
Built in cons function	Done
Sqrt, exp	Done
<, >, , ==, <=, >=, !=	Done
Extra Credit	Done
Lambda	

2. Arithmetic

The implementation of basic arithmetic like adding, subtracting, multiplying, and dividing has the potential to be very verbose, which I discovered firsthand on my initial implementation. Having a separate function for each operation is not only tedious, but hurts maintainability as well. Once I utilized a dictionary and the operator library, I was able to combine all arithmetic operations, including comparisons, into a single function, significantly shortening my code. By putting all the operations' syntax into a dictionary with their respective operator functions, I could remove operator syntax from the arithmetic function entirely, replacing it with the module's built-in functionality.

3. Variables

Variables in LISP require a few things to be implemented to function correctly. Defining variables was fairly straightforward. By simply employing a dictionary, I was able to make an entry with the user-defined variable name and the corresponding value, making it easy to search when it is used later on. When taking user input, all that has to be done is check to see if the user input value is contained in the variable dictionary. If it is, the program just has to use the associated value instead.

4. Functions

The implementation of functions is very similar to that of variables. A dictionary is used to store

the function name with its corresponding parameters and body. When a function might be called later, a check is made to see if the contents of the user input is within the functions dictionary. If that is the case, the evaluation function is recursively called to obtain the value from the function when called.

5. Car and Cdr

The implementation of car and cdr are extremely straightforward. Car simply returns the first value in the list that is given with the function. Conversely, cdr returns everything but the first entry in a list. This is easily done in Python with line[1][0] and line[1][1:].

6. Cons

The cons function, much like car and cdr, was very straightforward to implement in Python using a similar method.

7. CONCLUSION

This project has been an incredible learning tool to help better understand the inner workings of a programming language. It has also shown me even more of how powerful Python can be. Talking to fellow classmates, the implementation of some of the most basic features of my interpreter in Python took far more time and code in something like Java. With Python's built-in functionality and dynamic nature, it was very easy to implement many of the features required in a succinct manner. While the project seemed daunting at first, I appreciated the challenge and opportunity to walk away from this project with a lot more knowledge than I had coming in.