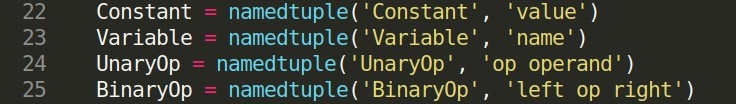


TOKEN\_REGEX : A regular expression that matches optional whitespace followed by a valid token. TOKEN\_END : Represents the end of an input string.

BOOLEAN\_CONSTANTS : The 2 constant values in boolean logic, true or false.

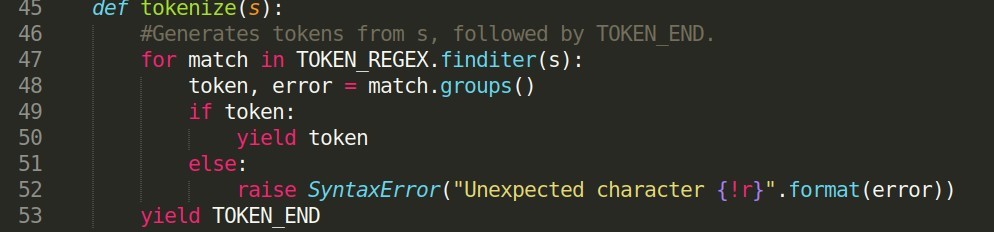
VARIABLES : This defines valid variable names, so all upper case and lowercase characters. UNARY\_OPERATORS: List of all unary operators and what symbols correspond to them. BINARY\_OPERATORS : List of all binary operators and what symbols correspond to them; can change symbols here



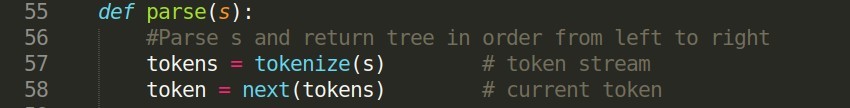
These lines define named tuples for constants, variables, unary operators, and binary operators. A named tuple is used as a stylistic choice instead of using custom objects which can produce the same result.



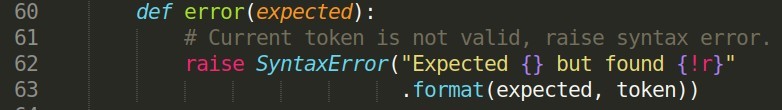
The function checkAmbiguity() takes the entire statement as a parameter and checks to see if a statement could be ambiguous. This is implemented using a finite state machine. This function goes through the input string character by character and counts all the operators and variables within each set of parentheses. If at any point there are more 3 operands and operators inside a single set of parentheses, then the statement is considered to be ambiguous.



This function generates a stream of tokens from a string, and at the end of the stream sends out a newline character.



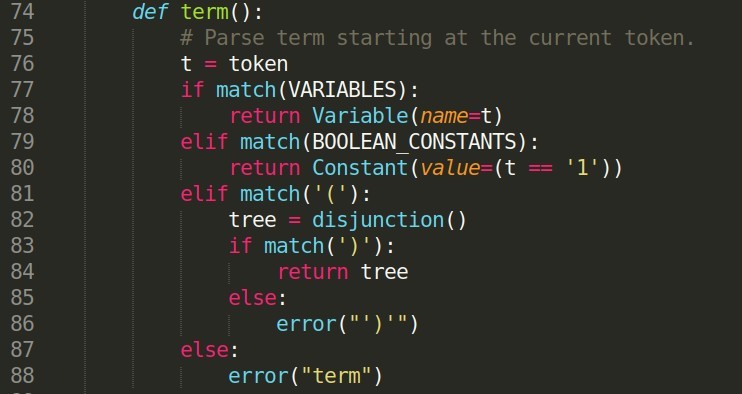
The next several panels will discuss the parse function. This function parses the user entered statement from left to right and generates a tree representation. The token variable keeps track of the overall current state in the parsing, as the work is done by using this value as a ‘global’ variable across the following functions.



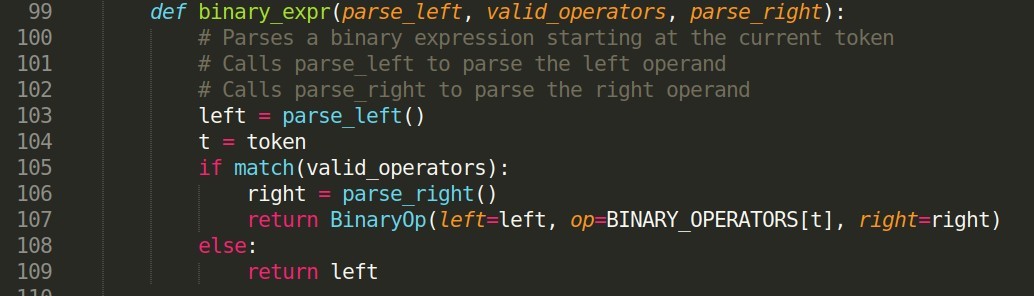
This function is called to raise an error if there is an invalid character in the user inputed statement. The decision to raise an error instead displaying a message is because if this is to be used in another application, according to convention the application should be notified of the error and can be made to handle it appropriately.



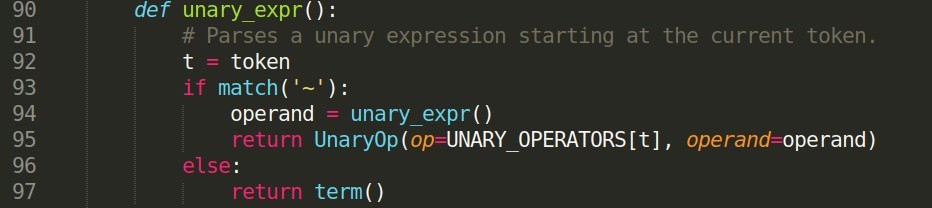
This function is called to check if a token is valid. It also functions to transition to the next token in the strream.



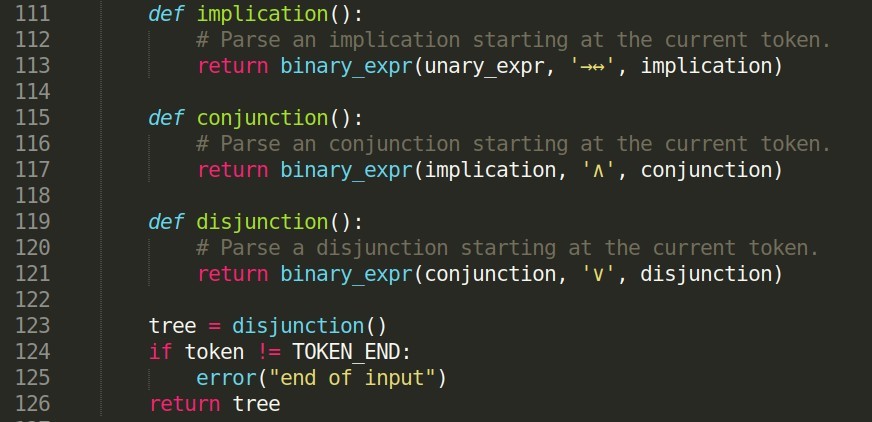
This function is last in the order of descent and is concerned with the most basic of nodes being variables and boolean constants.



This function represents the generic form of an implication, conjunction, and disjunction.



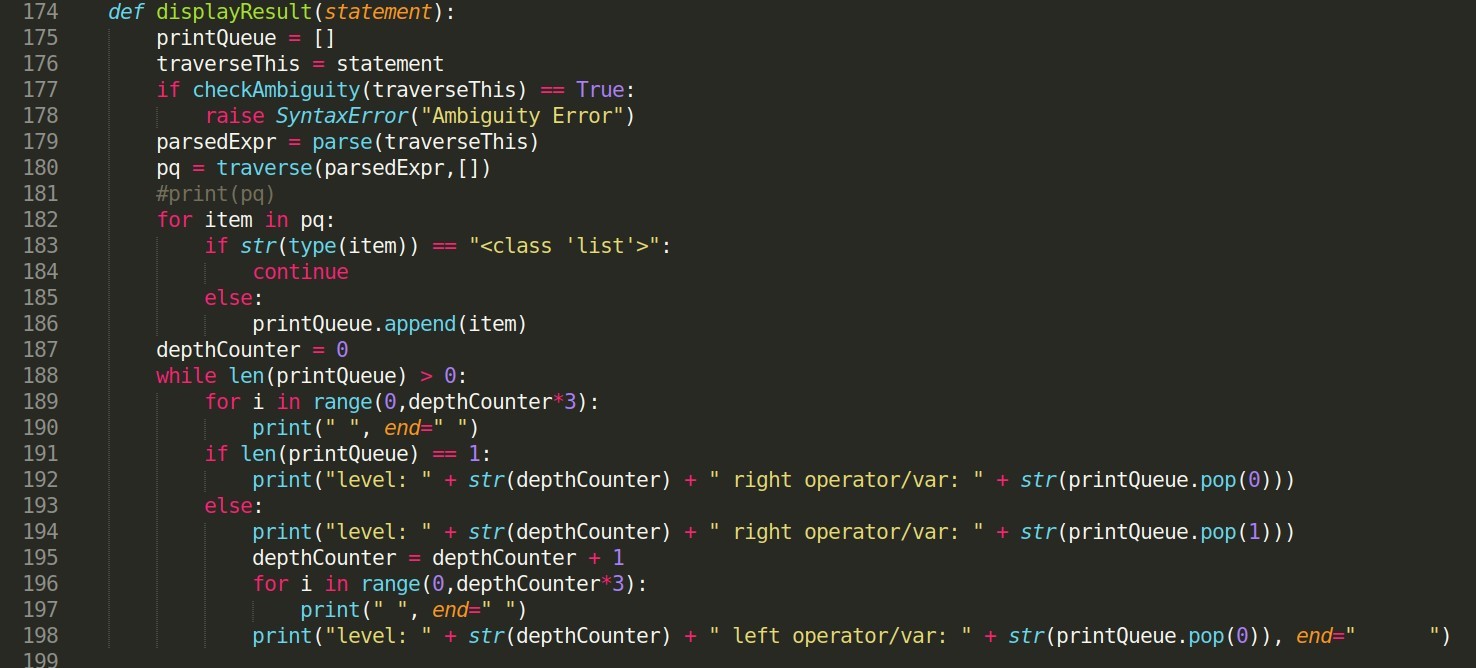
This function is called to parse a unary expresson. The only one being NOT. This also operates using recursion in the case of several NOTs in a row.



This entire parse function is implemented as a recursive descent parser. The order of descent is from disjunction to conjunction to implication to unary expression to term. Binary expressions continue to be parsed until broken down to operators and variables.



The traverse() function takes the output from the parse() function and constructs an array in a specific order for printing out the tree. The traversal is done by means of recursion. Starting at the leftmost object in the parse() function output, the function will continue to traverse it’s children nodes and add to the printQueue in order from left to right, because lines 136-138 go in that order. The end cases are when a variable or operator is run into. At the end the formatted array representation of the string will be returned. This format is a stylistic choice and is not mandatory across all implementations.



This is the main part of the program and the wrapper that brings everything together. First line 177 checks to see if the expression entered by the user is ambiguous. If it is, an error is raised. Next if the statement is not ambiguous, the expression is converted into tree form on line 179, and on line 180 that tree is traversed and arranged into the array for formatted printing to console. Lines 182-186 are there to mask a bug where sometimes unwanted data results from the traversal. Lines 187-198 print the tree to the console. Variable numbers of spaces are used to delineate levels as shown in lines 189,190,186, and 197.

Helpful Sites

[https://regex101.com/ - use this to test and develop regular expressions](https://regex101.com/) [https://www.geeksforgeeks.org/recursive-descent-parser/ : reference on recursive descent parsers](https://www.geeksforgeeks.org/recursive-descent-parser/)