**Introduction**

This language is based on the procedural paradigm. The language is intended to be general purpose. This will be a high-level language with a high level of abstraction from the instruction set of the hardware.

**Grammar for Language**

<abn> ::= <comment> | <statement> | <condition>| <expr> | <var\_assign> | <disp> | <endp>

<comment> ::= ‘#’ <string>

<statement> ::= ‘if’ ‘(’ <condition> ‘)’ ‘then’ ‘{’ <expr> ‘}’ ‘else’ ‘{’ <expr> ‘}’

| ‘if’ ‘(’ <condition> ‘)’ ‘then’ ‘{’ <expr> ‘}’ ‘else’ ‘{’ <var\_assign> ‘}’

| ‘if’ ‘(’ <condition> ‘)’ ‘then’ ‘{’ <expr> ‘}’ ‘else’ ‘{’ <disp> ‘}’

| ‘if’ ‘(’ <condition> ‘)’ ‘then’ ‘{’ <var\_assign> ‘}’ ‘else’ ‘{’ <expr> ‘}’

| ‘if’ ‘(’ <condition> ‘)’ ‘then’ ‘{’ <var\_assign> ‘}’ ‘else’ ‘{’ <var\_assign> ‘}’

| ‘if’ ‘(’ <condition> ‘)’ ‘then’ ‘{’ <var\_assign> ‘}’ ‘else’ ‘{’ <disp> ‘}’

| ‘if’ ‘(’ <condition> ‘)’ ‘then’ ‘{’ <disp> ‘}’ ‘else’ ‘{’ <disp> ‘}’

| ‘if’ ‘(’ <condition> ‘)’ ‘then’ ‘{’ <disp> ‘}’ ‘else’ ‘{’ <expr> ‘}’

| ‘if’ ‘(’ <condition> ‘)’ ‘then’ ‘{’ <disp> ‘}’ ‘else’ ‘{’ <var\_assign> ‘}’

| ‘if’ ‘(’ <condition> ‘)’ ‘then’ <expr>

| ‘if’ ‘(’ <condition> ‘)’ ‘then’ <var\_assign>

| ‘if’ ‘(’ <condition> ‘)’ ‘then’ <dis>

<condition> ::= <expr> ‘==’ <expr> | <expr> ‘>’ <expr> | <expr> ‘>=’ <expr>

| <expr> ‘<’ <expr> | <expr> ‘<=’ <expr>

<expr> ::= <expr> ‘+’ <expr> | <expr> ‘-’ <expr> | <expr1> ‘\*’ <expr2> | <expr1> ‘/’ <expr2>

| ‘(’<expr> ‘)’ | <id> | <int> | <float>

<var\_assign> ::= <id> ‘=’ <expr> | <id> ‘=’ <string>

<id> ::= <string> | <id> <string>| <id> <digit>

<int> ::= ‘[- | +]’ <digit> | <int> <digit>

<float> ::= ‘[- | +]’ <int> ‘.’ <int> | ‘.’ <int>

<digit> ::= ‘0’ | ‘1’ | … | ‘9’

<string> ::= ‘ “ ’<char>‘ ” ’|‘ “ ’<digit>‘ ” ’|‘ “ ’<string> <char>‘ ” ’|‘ “ ’<string> <digit>‘ ” ’

<char> ::= ‘a’ | ‘A’ | ‘b’ | ‘B’ | ‘c’ | ‘C’ | … | ‘z’ | ‘Z’ | ‘\_’

<operator> ::= + | - | \* | / | ^ | = | == | > | >= | < | <=

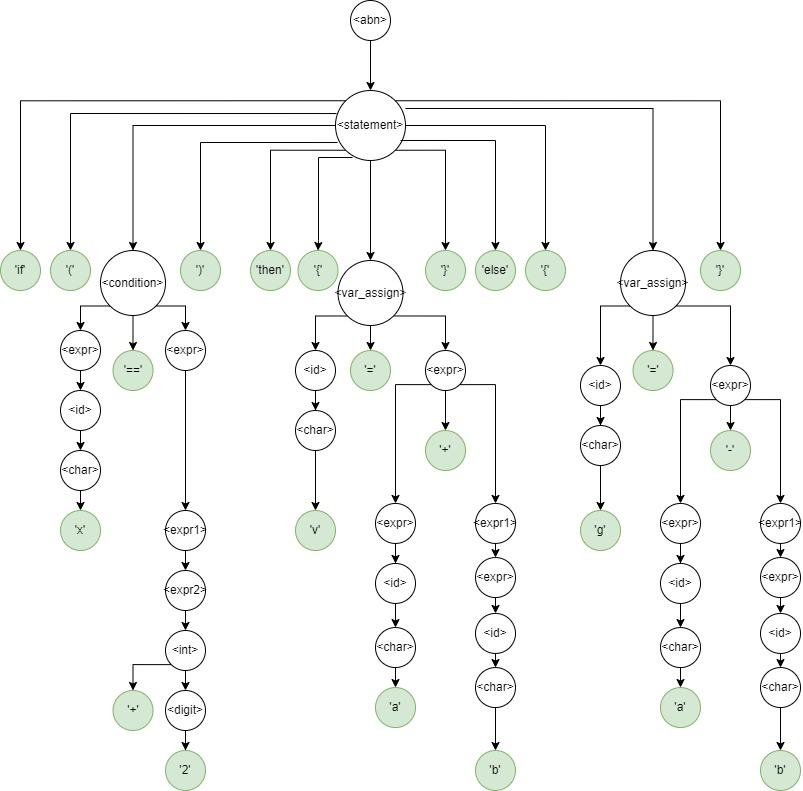
<punctuator> ::= ( | ) | { | }

<disp> ::= <string> <id>

<endp> ::= ‘leave’

**Parse Tree**

*ABN Parse tree for if (x == 2) then {v = a + b} else {g = a - b}*



**Tokens**

Tokens of this language are as follows:

Identifiers

Keywords: if, then, else, leave

Constants: int, float, string

Operators: - + \* / = == > >= < <=

Punctuators: - ( ) { }

**Regular Expressions**

Left Parenthesis = r'\('

Right Parenthesis = r'\)'

Plus = r'\+'

Minus = r'\-'

Times = r'\\*'

Divide = r'\/'

Equals = r'\='

Equality = r'\=='

Greater than = r'\>'

Greater than or equal to = r'\>='

Lesser than = r'\<'

Lesser than or equal to = r'\<='

Left bracket = r'\{'

Right bracket = r'\}'

If = 'if'

Then = 'then'

Else = 'else'

Leave = ‘leave’

Float = r'[-+]?[0-9]\*\.[0-9]+'

Int = r'[-+]?[0-9]+'

Id = r'[a-zA-Z\_][a-zA-Z\_0-9]\*'

String = r' "[^"]\*\" '

Comment = r'\#.\*'

**Compiler Language**

Python is a general purpose, dynamic, high-level, and interpreted programming language. It is processed at runtime by using an interpreter. It is dynamically typed and supports multiple programming patterns. These include imperative, functional, procedural, and object-oriented programming (Grover, 2019). It features dynamic memory allocation.

**Characteristics and Effect on Readability, Writability, Reliability**

Simplicity - Simplicity was a key consideration in creating our language. Using easily understandable keywords and features that conform to our grammar specifications plays a big part in having our language be highly readable, easy to write, and therefore reliable as the concepts are easily understood. Feature multiplicity is addressed in our language by not allowing too many functions for one construct or operator. For example it is only legal to add numbers. String concatenation for this project is not allowed using plus sign. To execute variable assignment type ‘variable name’ equals ‘expression’ or ‘string’.

Data Types - Our language utilizes various data types, categories that define the properties of the data. The data types contribute greatly to the readability, writability and reliability of the language. Choosing the correct data type helps those reading programs written in the language to understand the purpose of the variables, those writing the code to do so efficiently and ensure that it has the desired functionality. Reliability is also increased in choosing correct data types as it reduces the chance of an error in the code. For example, ensuring that the integer data type is only assigned integers, and not floats or strings.

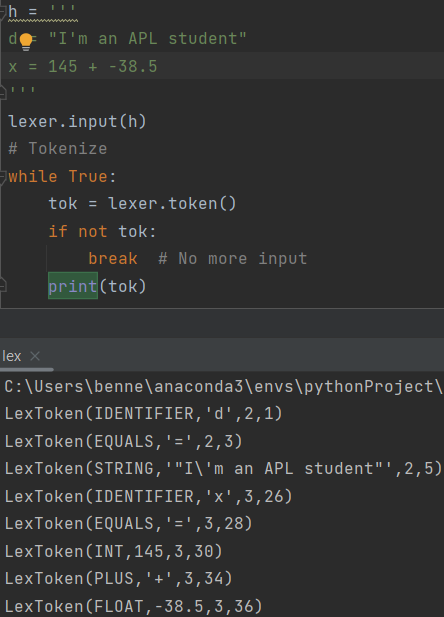


Fig 1. Lexical analyzer identifying string, integer and float data type.

Syntax Design - Syntax can either make or break the readability of a program. Therefore, it is a very important consideration in the language. Good syntax design produces code that is easy to write, and easy to follow when reading. A well-designed syntax can make it easier to write reliable code, as it can provide clear and consistent rules for writing error-free code.

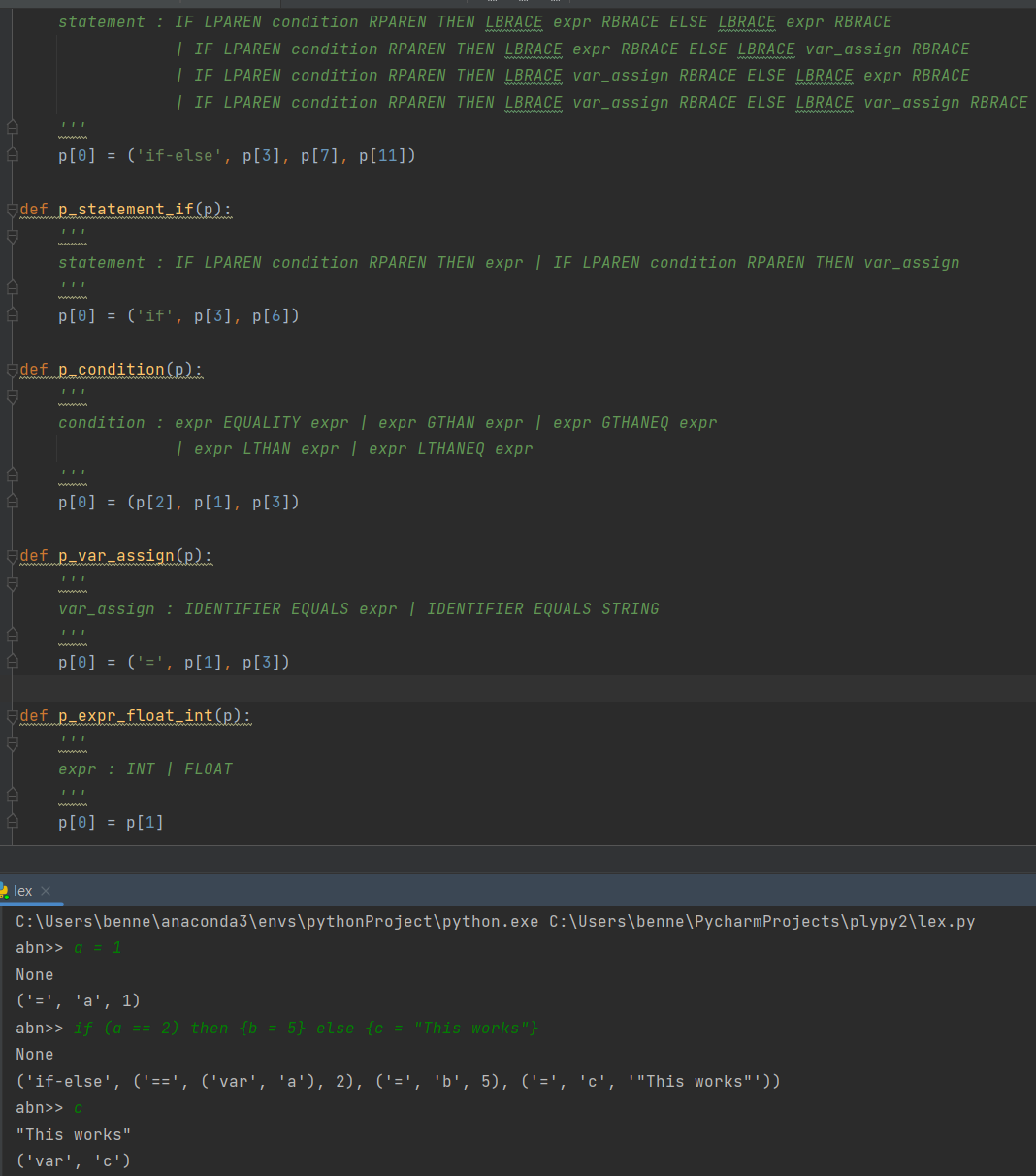


Fig 2. Modeled parse tree showing this if statement follows the syntax.

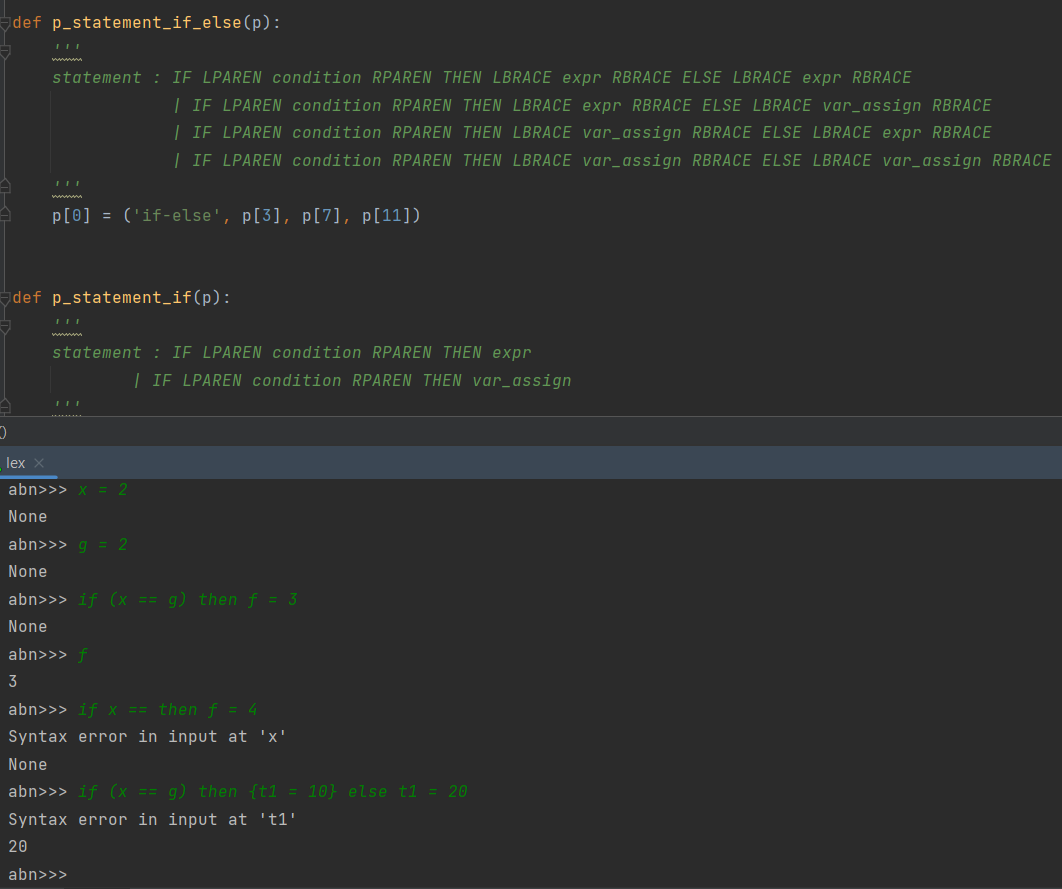


Fig 3. Program identifying errors in the if-then and if-then-else statement based on the grammar

**References**

Beazley, D. M. (n.d.). PLY (Python Lex-Yacc). Ply (python lex-yacc). Retrieved March 31, 2023, from https://my.eng.utah.edu/~cs3100/lectures/l14/ply-3.4/doc/ply.html

Grover, J. (2019). *Perceiving python programming paradigms*. Opensource.com. Retrieved March 31, 2023 from https://opensource.com/article/19/10/python-programming-paradigms