**Compiler Design Project Report**

**Group Id:** 41

**Group Members:**

Dudekula Suraj: 20114038

Mallamgari Nithin Reddy: 20114053

Thadavarthi Vishnu Sri Sai Sankar: 20114099

**Problem Statement:**

Building an Interpreter in Python language that does multiply. Work with tokens, lexical analyzer and expressions that multiply integers from integers.

**Description of Codes:**

What is Interpreter?

In [computer science](https://en.wikipedia.org/wiki/Computer_science), an interpreter is a [computer program](https://en.wikipedia.org/wiki/Computer_program) that directly [executes](https://en.wikipedia.org/wiki/Execution_(computers)) instructions written in a [programming](https://en.wikipedia.org/wiki/Programming_language) or [scripting language](https://en.wikipedia.org/wiki/Scripting_language), without requiring them previously to have been [compiled](https://en.wikipedia.org/wiki/Compiler) into a [machine language](https://en.wikipedia.org/wiki/Machine_language) program.

The software by which the conversion of the high-level instructions is performed line-by-line to machine level language, other than compiler and assembler, is known as**INTERPRETER**. If an error is found on any line, the execution stops till it is corrected. This process of correcting errors is easier as it gives line-by-line error, but the program takes more time to execute successfully. Interpreters were first used in 1952 to ease programming within the limitations of computers at the time. It translates source code into some efficient intermediate representation and immediately execute this.

In this file, we will deal with each token at a time and perform the syntax analysis at the same time. If there is a syntax error, it is shown here which is the main function of interpreter.

Now the Interpreter that we have build have two stages, they are:

1. Lexical Analysis
2. Interpretation

In Lexical Analysis, we will define the tokens and we will take the whole sample python program as input and categorize the program into tokens through tokenization method. If there is any unrecognized token in the program, Lexical Error will be given.

We used string manipulation to extract the tokens from the sample python program.

The tokens which we are going to recognize in our Interpreter are:

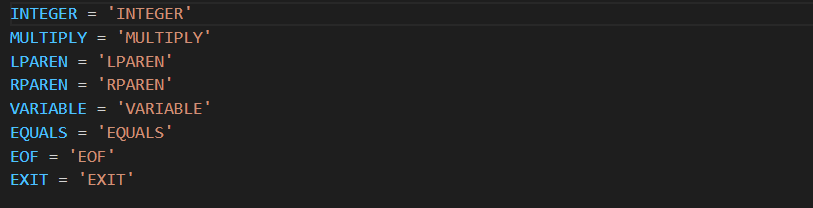
1. [0-9][0-9]\* - INTEGER
2. [a-zA-Z][alnum]\* - VARIABLE
3. ‘/\*’ - MULTIPLY
4. = - EQUALS
5. ‘/(‘ – LPAREN
6. ‘/)’ - RPAREN
7. - EOF

Now let’s discuss about the code and their functionalities below:

**Demonstration of our codes written**:

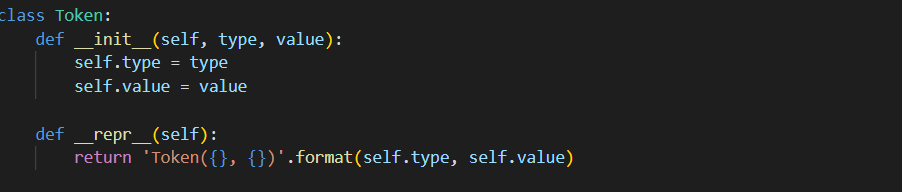
**Lexer.py file:**

Valid Tokens:



These are the tokens that will get recognised as Valid tokens.

Token class:



Each token will be associated with a type and value.

Lexer class and tokenization:



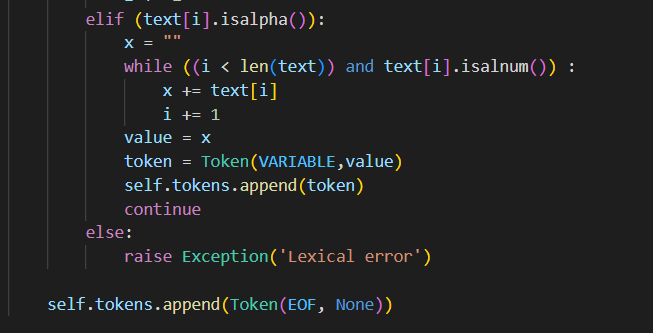
An object of lexer class will be created in Interpreter.py and subsequently we tokenize the input by calling tokenize() method.

If there is a space or a newline, we are just increasing lookahead and continuing the loop.

Else if there is print statement of python, it will be recognized as passed on to print the expression inside which will be done in Interpreter.py file.

Else If there is an Integer, we will be checking whether it is valid integer or not and then returning token of ‘INTEGER’ type with it’s value.

Else if there is multiply operator i.e, ‘\*’ or left parenthesis or right parenthesis or equals operator, appropriate token will be returned.



Else if we get a lexeme of declared type ‘VARIABLE’, it will be returned and if not Lexical error will be raised and it will be handled in try except block of Interpreter.py file.

And if we reach the end of the file, EOF token will be returned.



lookaheadtoken() method:

Based on the token\_idx, appropriate token will be retrieved and returned from tokens array.

get\_next\_token() method:

The value of token\_idx will be increased by 1 and next token will be returned.

reset() method:

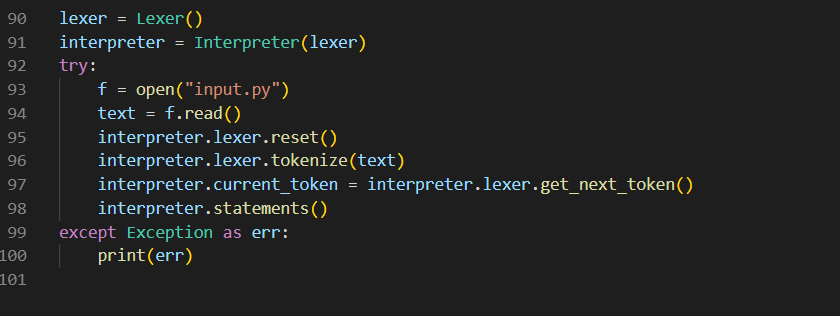
tokens array will be cleared and token\_idx will be set to 0.

**Interpreter.py file:**

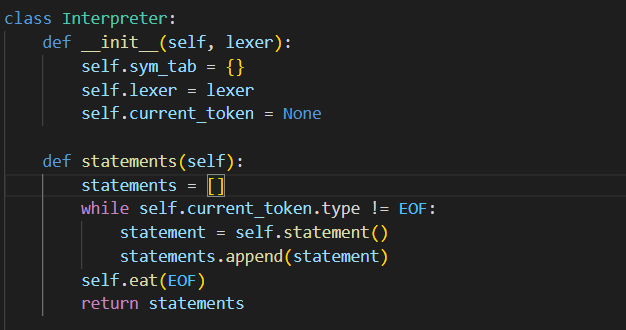
We will be using file handling and Lexer.py file, hence we will be importing them.

****

Taking input and calling appropriate methods for getting started:

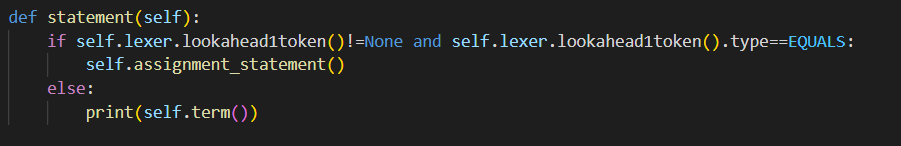
****

Interpreter class:

****

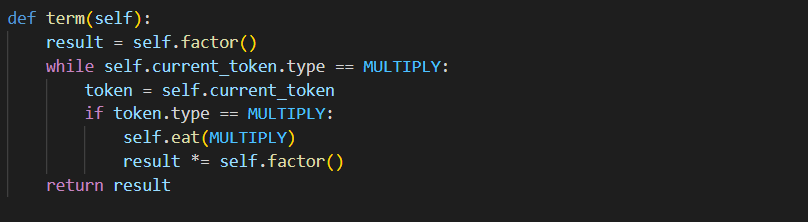
A symbol table named sym\_tab is created which will be used for future references and statements() method will be called which will check whether we reached EOF and call statement() if not in a while loop.

statement() method:



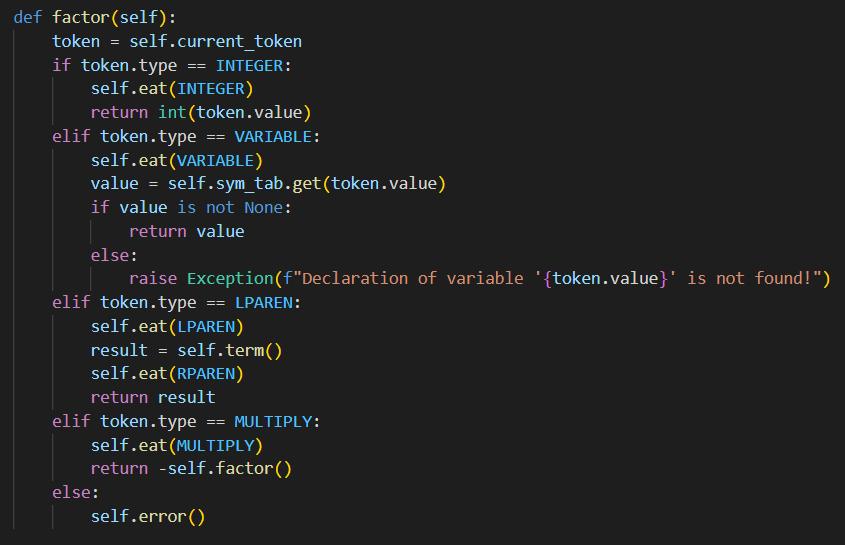
Here we will be checking if there is any assigned operation and if not we will to leaded to calculate the expression which is represented as term() here.

term() method:



We will be evaluating the expression, dividing it into operands and operator which here is MULTIPLY and will be returning the result.

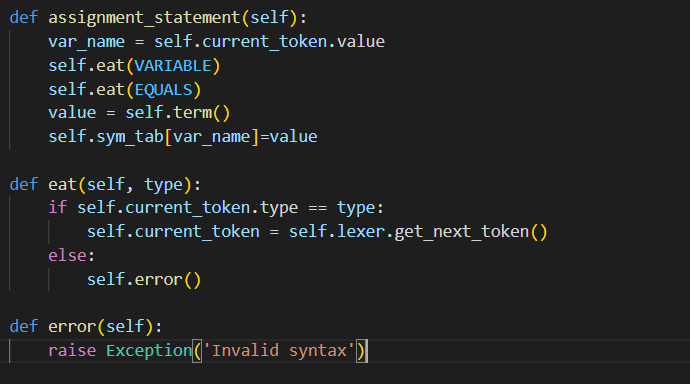
factor() method:



We will be checking whether the current token is INTEGER or VARIABLE or LPAREN or MULTIPLY and if not we will be raising an error.

It is to be noted that if the current token is LPAREN then there should be an expression after it followed by RPAREN.

assignment\_statement() and eat() methods:

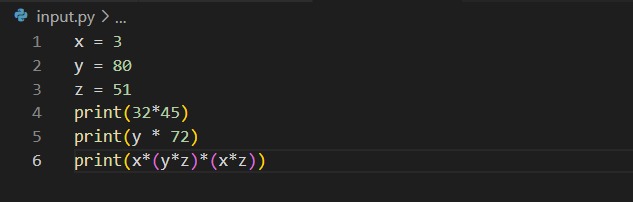


If there is an EQUALS token, we will be calling assignment\_statement() method and if we want to move to next token, we will be initiating current\_token with next token by calling get\_next\_token() method.

If there is any error be it lexical or syntax, error() will be called and an exception will be raised.

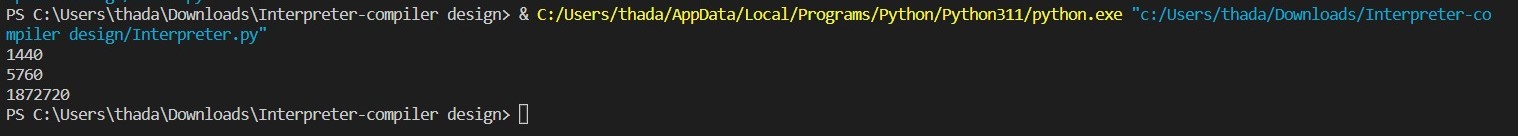
**Test cases:**

Sample input – 1:



We have declared x, y and z initially and subsequently multiplication operations are done. Initially, multiplication of two literals is done and then multiplication of a variable and literal is done. Then, multiplication of multiple operands associated with parentheses is done.

Output:

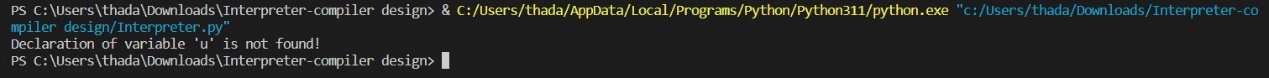


Sample input – 2:

Graphical user interface

Description automatically generated with medium confidence

Output:



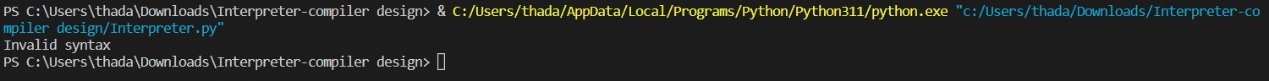
Since variable u is not defined, “declaration of variable u is not defined” is printed.

Sample input – 3:

A picture containing background pattern

Description automatically generated

Output:



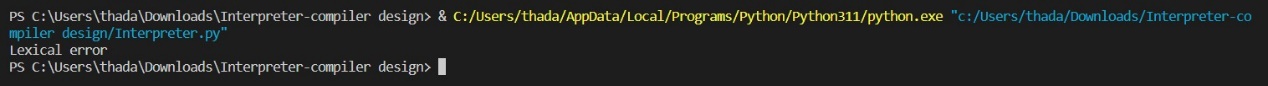
Left and right parentheses are not matched and “Invalid syntax” is printed.

Sample input – 4:

Graphical user interface

Description automatically generated with medium confidence

Output:



Since addition operator is used which is not a valid token, “Lexical error” is printed.

**Conclusion**:

We are taking input and tokenising the given input line by line and all valid tokens are added to an array and symbol table is created which will be used for any future references. Finally, output is given based on the code in input.py file. Thus the Interpreter for Multiplication operation is built in python.