

# **Kennesaw State University**

College of Computing and Software Engineering

Department of Computer Science

CS4308/W01 Concepts of Programming Languages Spring 2020

Module 5: Second Deliverable – Parser – CPL Project Report

Instructor: Dr. Jose Garrido

**Project Team:**

Zander Bandoly

[abandol1@students.kennesaw.edu](mailto:abandol1@students.kennesaw.edu)

Kindra Hensley

[khensle1@students.kennesaw.edu](mailto:khensle1@students.kennesaw.edu)

John Foster

[jfost103@students.kennesaw.edu](mailto:jfost103@students.kennesaw.edu)

March 16, 2020

**Initial Problem Statement**

### This project consists of developing an interpreter for a minimal form of the BASIC language. The specification of the grammar is given below. The interpreter will process a BASIC program. All tokens in this language are separated by white spaces. The parsing algorithm should detect any syntactical or semantic error. The first such error discovered should cause an appropriate error message to be printed, then terminate the Interpreter. Run-time errors should also be detected with appropriate error messages being printed. In the first deliverable, Module 3, the goal of the assignment was to develop a complete scanner for a minimal form of the BASIC language and write the syntax specification of the BASIC language using an appropriate notation. In this deliverable, Module 5, our goal is to develop a complete parser that executes with the scanner we developed in the first deliverable, Module 3.

### A short report describing the work performed is continued below. It details the execution of the scanner program by using the appropriate input file, table-of-squares.bas and will show a list of the tokens scanned. It also details the execution of the parser program by using relevant source line as input, table-of-squares.bas, and display the corresponding statement being recognized.

**Summary**

The implementation of the Scanner class will open a file specified in its argument, split the file up line by line, and index each character of every line one by one to utilize these to build lexemes/words. The implementation of the Word class holds each lexeme, its token group, and the 'row' and 'col' indexes. The Word class turns the character stream taken in by the Scanner function and turns the stream into lexemes or special characters then displays its relevant information. For testing we will use the table-of-squares.bas test file.

# BASIC Grammar Subset

The grammar we choose for this assignment allows users to create BASIC programs using For Loops, Print, Next and Input statements. We made a BNF with Lexical definitions to create the rules for our subset of the BASIC language grammar allowed. The grammar is defined in the file “BASIC\_Subset.txt”

<Program> -> <Code> end

<Code> -> <Line><Code> | <Line>

<Line> -> <INT\_LIT> <Statement>

<Statement> -> <For\_Statement> | <Print\_Statement> | <Next\_Statement> | <Input\_Statement>

<For\_Statement> -> for <Boolean\_Expression> to <id> <Print\_Statements> <Next\_Statement>

<Print\_Statements> -> <Print\_Statement> | <Print\_Statement> <Print\_Statements>

<Print\_Statement> -> print <id> | print <id>, <id> \* <id> | print <String\_Literal>

<Next\_Statement> -> next <id>

<Input\_statement> -> input <id>

<Boolean\_Expression> -> <id> = <Number>

<String\_Literal> -> " <id> "

<id> -> Letter | Digit | <id><id>

<INT\_LIT> -> <Number>

<Number> -> Digit | <Number>Digit

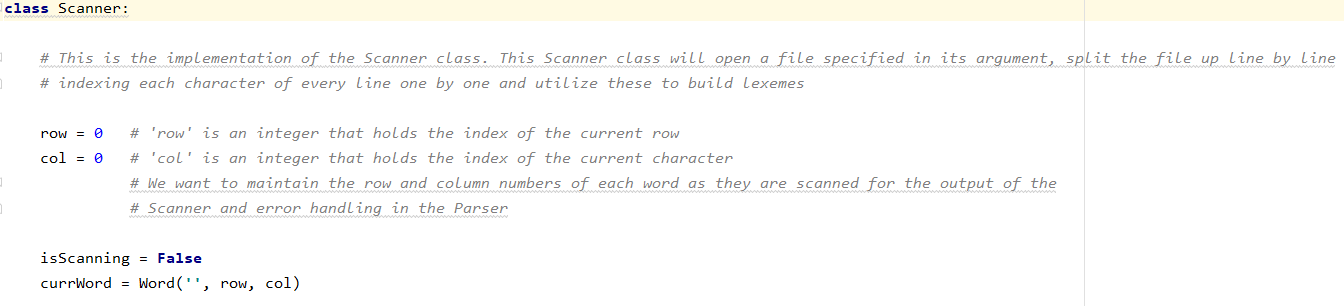
Lexical Analyzer

Letter -> [a-zA-Z]

Digit -> [0-9]

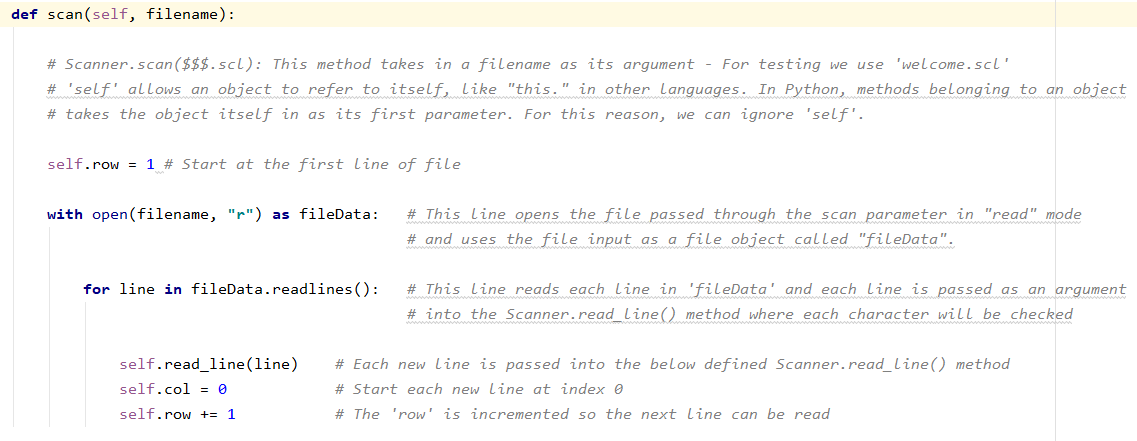
# Class Scanner:

The Scanner class will use variables that monitor row and column integers of the currently scanned row, and the current character, so that the program can maintain track of the words scanned for the output of the scanner, and handle any errors in the parser.



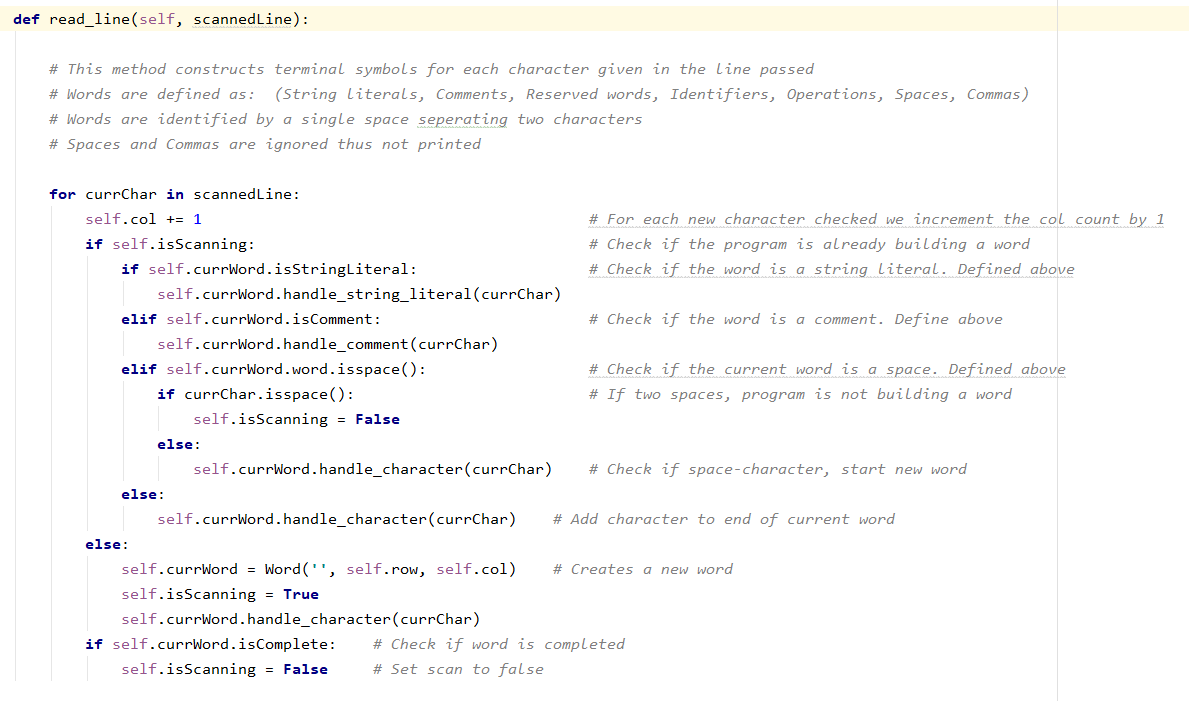
## Scanner.scan()

The scan(filename.bas) method takes in a filename as its argument. For testing we used test file named table-of-squares.bas. ‘self’ allows an object to refer to itself, like "this." in other languages. In Python, methods belonging to an object takes the object itself in, as its first parameter. Thus, allowing the program to ignore ‘self.



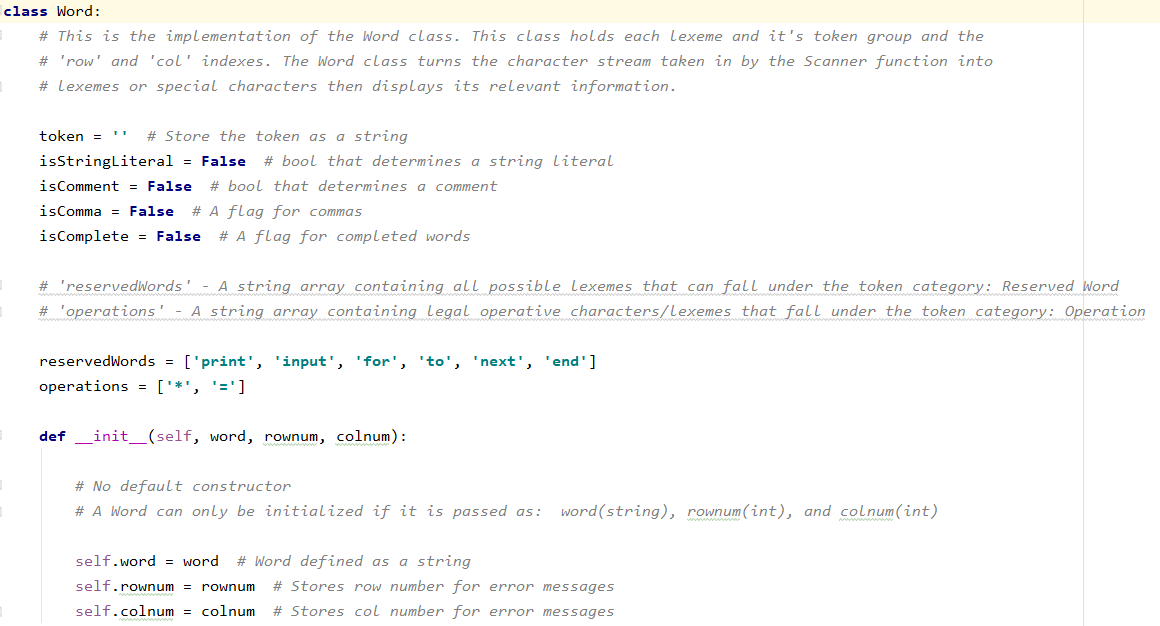
## Scanner.read\_line()

The read\_line method constructs terminal symbols for each character given in the line passed. Words are defined as: (String literals, Comments, Reserved words, Identifiers, Operations, Spaces, Commas). They are identified by a single space separating two characters, where spaces and commas are ignored and not printed.



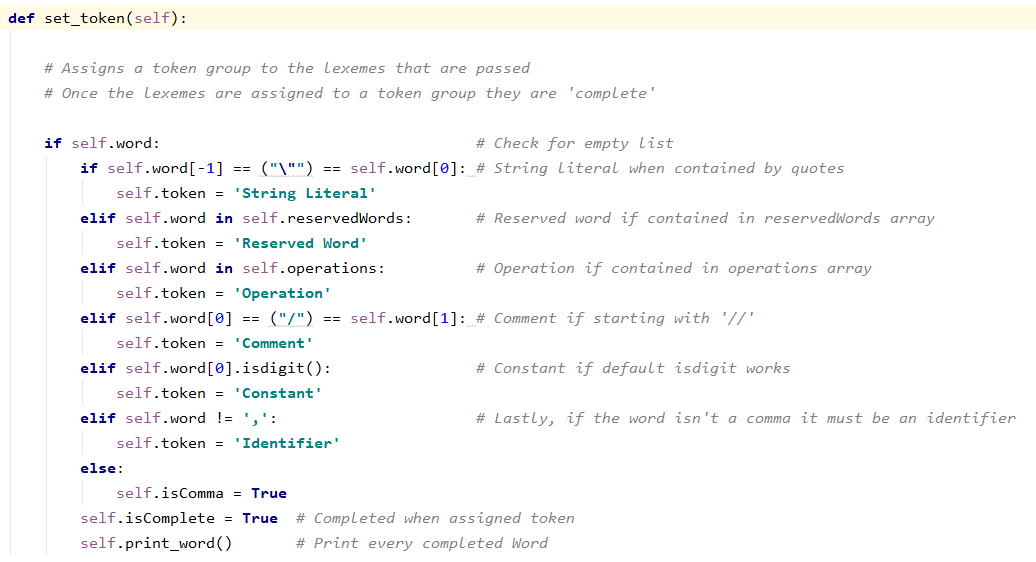
# Class Word:

The Word class holds each lexeme/word for the token group the lexeme is associated with, as well as the row and col indexes passed via the Scanner Class. This class turns the Scanner stream characters into lexemes or special characters and outputs the relevant information. Our Word class holds the subset definitions of the BASIC language that we will be interpreting. The subset is defined in our two arrays called ‘reservedWords’ and ‘operations’, which hold our subset lexemes. The subset can also be found in ‘BASIC\_Subset.txt’ file.



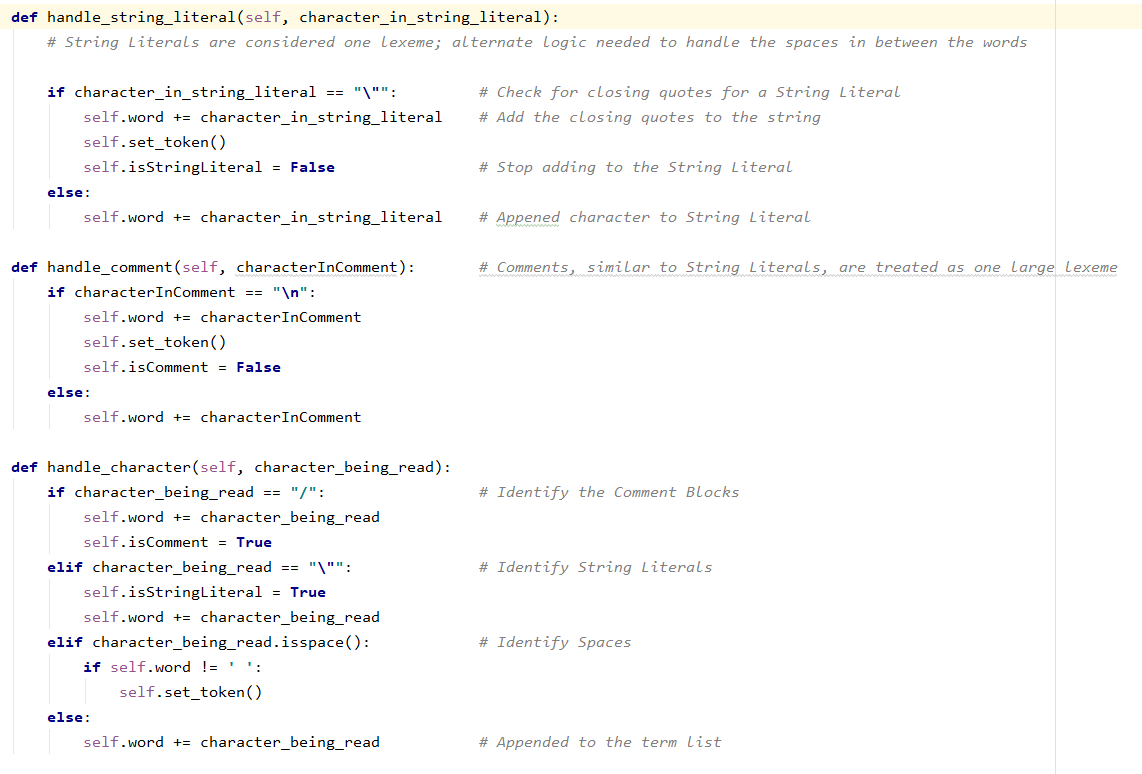
## Word.set\_token()

The set\_token method assigns a token group to the lexeme through a series of checks against our defined subset, comments and identifiers. This method will then output the token assigned, and the word passed.



## Word.handle\_string\_literal() - Word.handle\_comment() - Word.handle\_character()

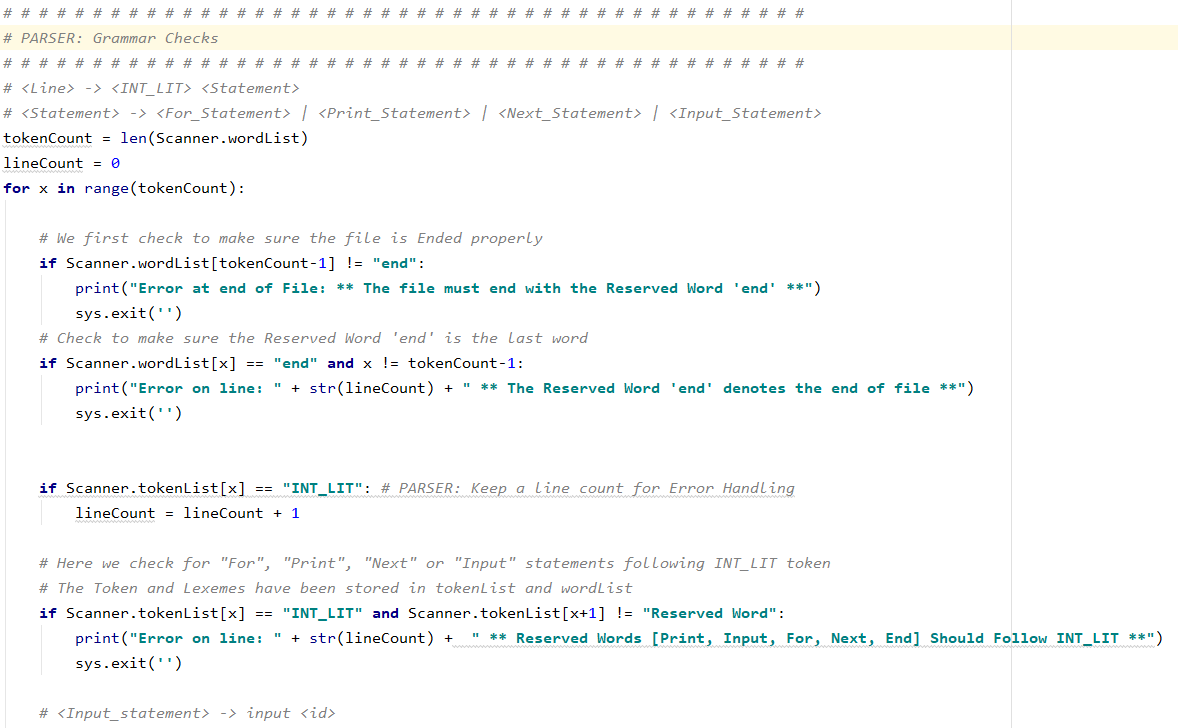
The ‘handle\_string\_literal’, ‘handle\_comment’ and ‘handle\_character’ are if/else methods checks are used to check the word for their respective names.

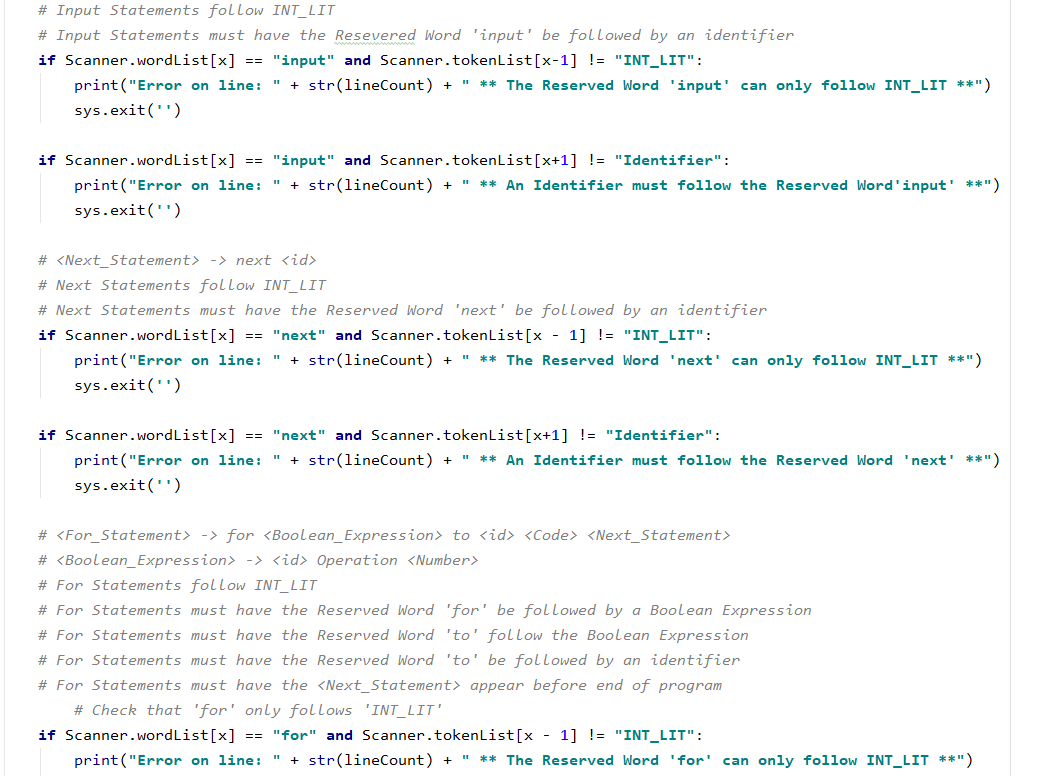


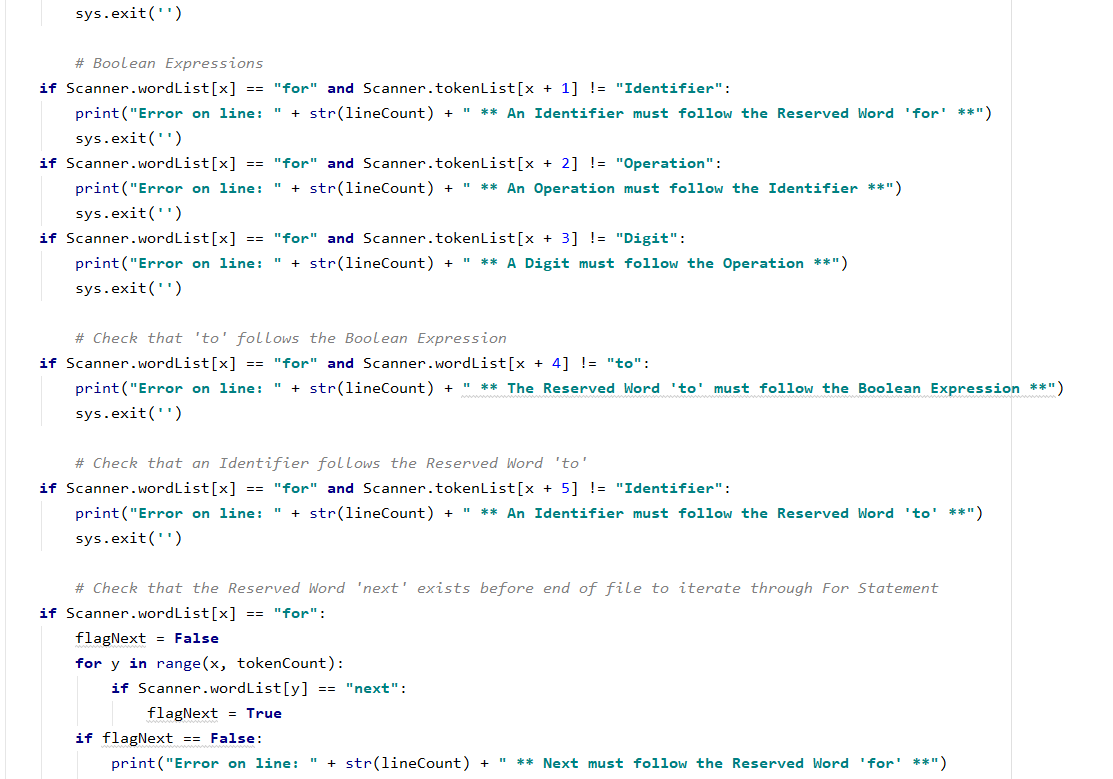
# Class Parser:

The Parser portion of the code handles grammar checks defined in the BASIC\_subset.txt file. The parser checks the token and word object lists created by the scanner function. After each line is created, the Parser checks for grammatical errors within the tokens and words. It is imperative to ensure that the tokens follow the defined grammar and when errors are found, the program prints out the line number with the necessary grammar to follow the token in question. This also stops the program to allow for quick error handling. *See Error Handling on page 12.*

Along with the error handling portion, we have the Parser program print out the tokens and subsequent words followed by the correct grammar needed for the next token. The Parser loops through the token and word list allowing the user to see the various decision trees that each token allows. Our Parser print function ends with the end of the tokens and words list which have been checked to make sure that the last token is the Reserved Word ‘end’.

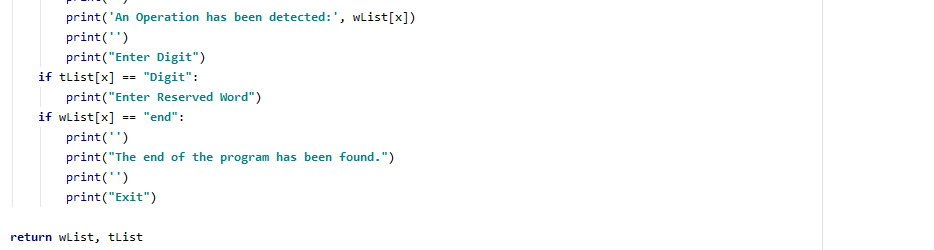






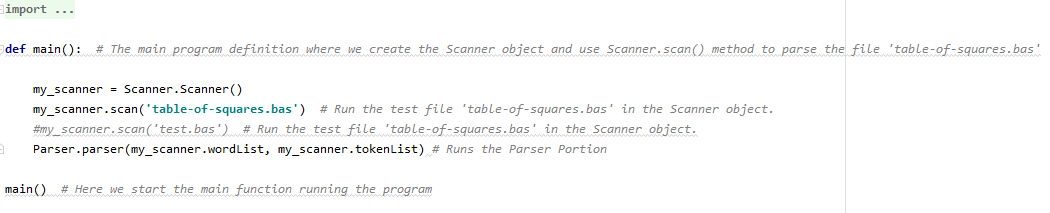






# main

The main program definition where we create the Scanner object and use Scanner.scan() method to parse the file 'table-of-squares.bas'. my\_scanner.scan('table-of-squares.bas') will run the test file ' table-of-squares.bas' in the Scanner object.



# Class Interpreter:

The Interpreter class of the code package

The Parser portion of the code handles grammar checks defined in the BASIC\_subset.txt file. The parser checks the token and word object lists created by the scanner function. After each line is created, the Parser checks for grammatical errors within the tokens and words. It is imperative to ensure that the tokens follow the defined grammar and when errors are found, the program prints out the line number with the necessary grammar to follow the token in question. This also stops the program to allow for quick error handling. *See Error Handling on page 12.*

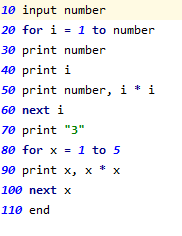
Along with the error handling portion, we have the Parser program print out the tokens and subsequent words followed by the correct grammar needed for the next token. The Parser loops through the token and word list allowing the user to see the various decision trees that each token allows. Our Parser print function ends with the end of the tokens and words list which have been checked to make sure that the last token is the Reserved Word ‘end’.

**How to run the program:**

You can run the source code through an IDE. This program was developed in PyCharm. You will want to make sure the table-of-squares.bas file is under the Project folder to ensure the program can find and reads the correct file. Make sure to have the python interpreter installed and the path to the interpreter properly set to run Python commands. To configure a Python interpreter in PyCharm, click on the blue link at the top right corner of the IDE that reads ‘Configure Python Interpreter’. From the dropdown menu, select Python 3.7, or the latest version. Follow the prompts to configure the interpreter. Once configured, click on the green arrow to run the program. The output should display the text mirrored in the input test file. See the screenshots below of both the input file and the expected output.

**Input:**

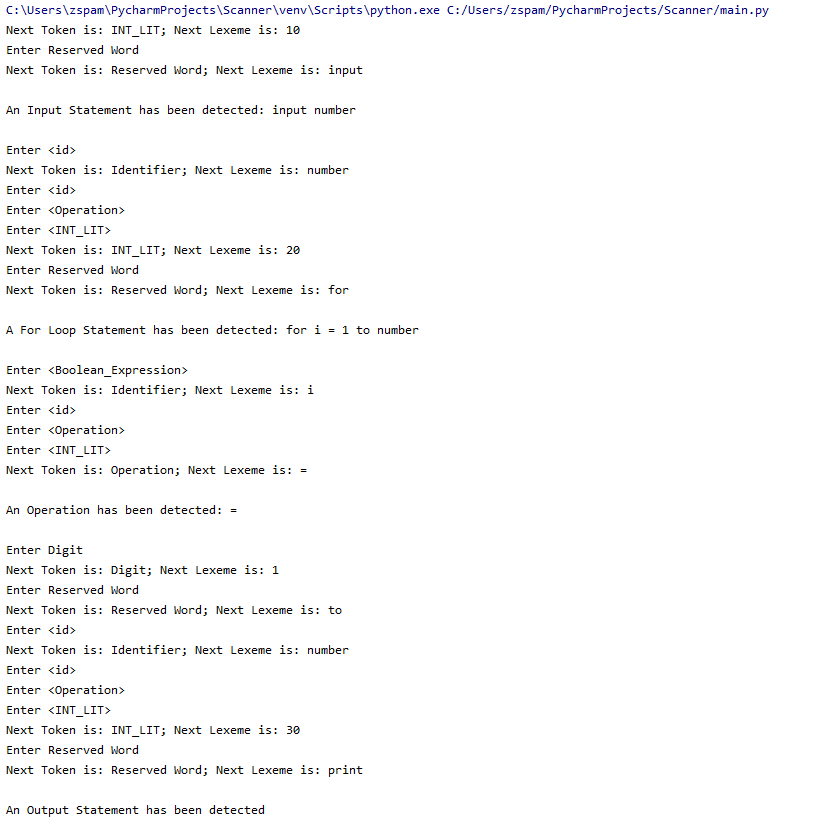
The input file is the test file, table-of-squares.bas. The file creates an output table listing numbers and their squares based on the amount of values the user would like to see. This file will be read by the Scanner with the Words/Lexicon and displayed as the output in the program.

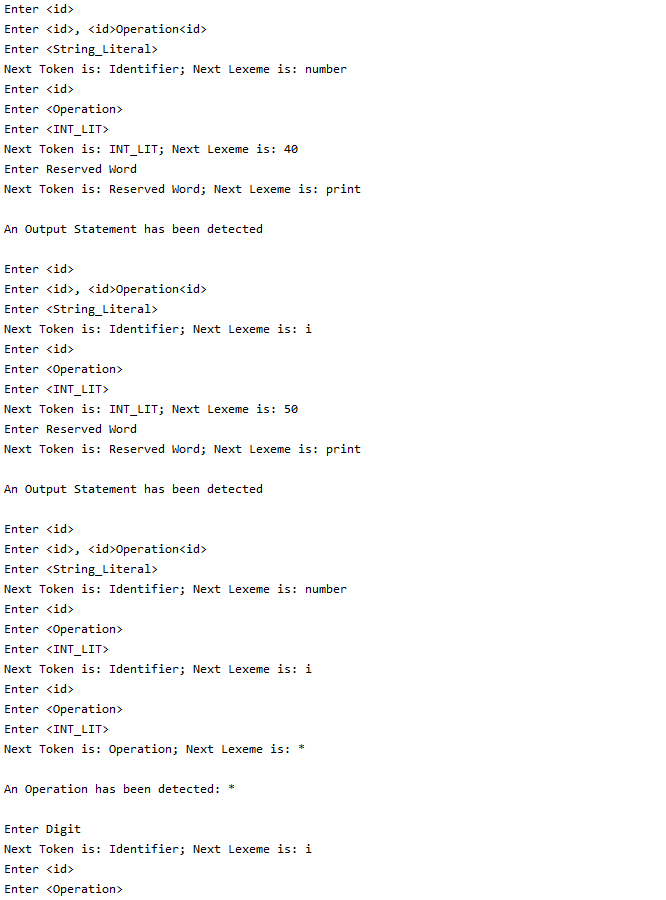


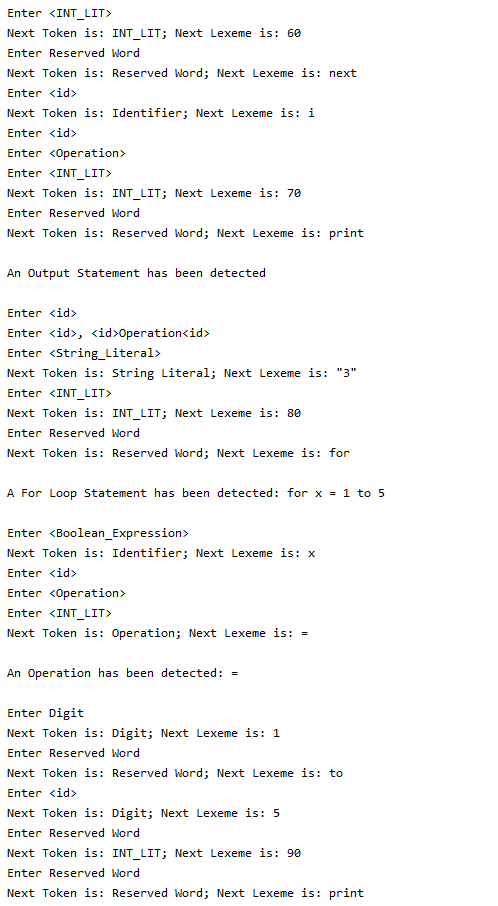
**Output:**

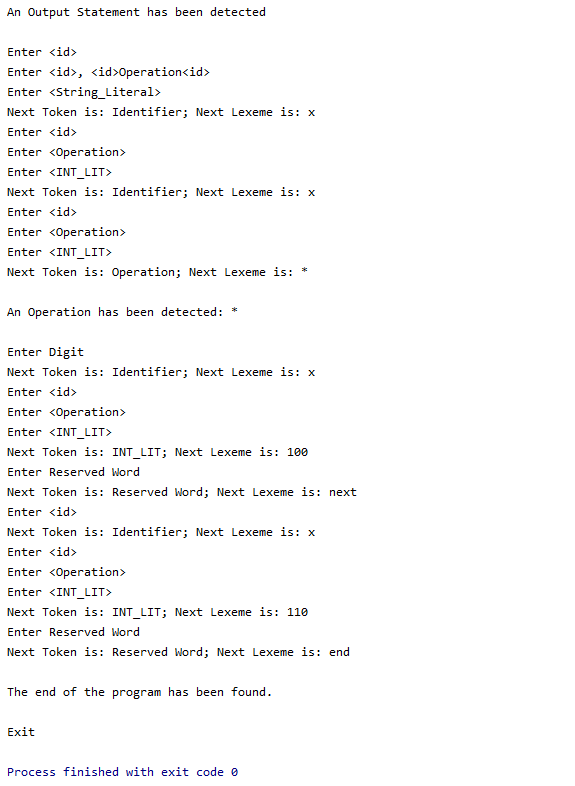
The output displays the scanned data that is contained within the table-of-squares.bas file. The program finds the table-of-squares.bas file, scans it and tests each word assigning it a token. The program then displays each word, it’s column and row (line) along with the words associated token. The syntax analyzer determines whether the given programs are syntactically

Correct. It scans the table-of-squares.bas file line by line, to output the tokens and lexemes.









**Error Handling**

The parser is designed to handle errors. When errors are found within the program, it will output the line number with the necessary grammar to follow the token in question. The program stops to allow error handling. In the example below, the input file was edited to reflect intentional errors. On line 30, the Reserved Word ‘print’ was added to throw an error and on line 40, an operator, =, was added before an Identifier. The program identifies the errors in the table-of-squares.bas file and prints the errors and their specified line within the file.

Input

A screenshot of a cell phone

Description automatically generated

Output

A screenshot of a cell phone

Description automatically generated

**Comments and Conclusion:**

This scanner and tokenizer are a fairly basic reader script, with ample comments to help with later bug testing during parser functionality implementation. The program works without issue in the current environment and the construction of our code should make it easy to update if issues arise when implementing the parser section of the interpreter project. Any possible issues that may arise later on would most likely be the result of bad token conversion, and that is where we will look to bug test in the future.

**References:**

Kerr, Kenny. “Compiler Basics, Part 2: Building the Scanner.” Visual Studio Magazine. N.p., 20 June 2014. Web. 12 Feb. 2020.

Leeward Community College. Streams of binary matrix code. *University of Hawai’i Community Colleges*, Department of Information and Computer Science. Image. 2020

Sebesta, Robert W. Concepts of Programming Languages. Pearson, 2012.