Iasam KaKaKhail

2020490

Design Decisions:

Use of RE (Regular Expressions):

Regular Expressions are used for tokenizing the Language, ensuring an effective approach to recognition of different token patterns.

Class-Based Design:

The Language Scanner is implemented as a class to encapsulate the scanning functionality, for code organization and reuse.

Token Representation:

The representation of tokens as tuples remains unchanged, providing a standard way to represent tokens.

Error Handling:

Lexical errors continue to be handled by printing error message with details about the line number and token.

Reading Source Code:

Code is read from a TXT file specified as command-line argument.

Scanner Structure:

Initialization:

The Scanner is initialized with the Language source code and sets up data structures.

Tokenization:

The Scanner reads the code line by line, ignores comments, and uses regular expressions to find and classify tokens.

Token Types:

Token Types are determined based on Regular Expression matches and the defined Language specifications.

Error Handling:

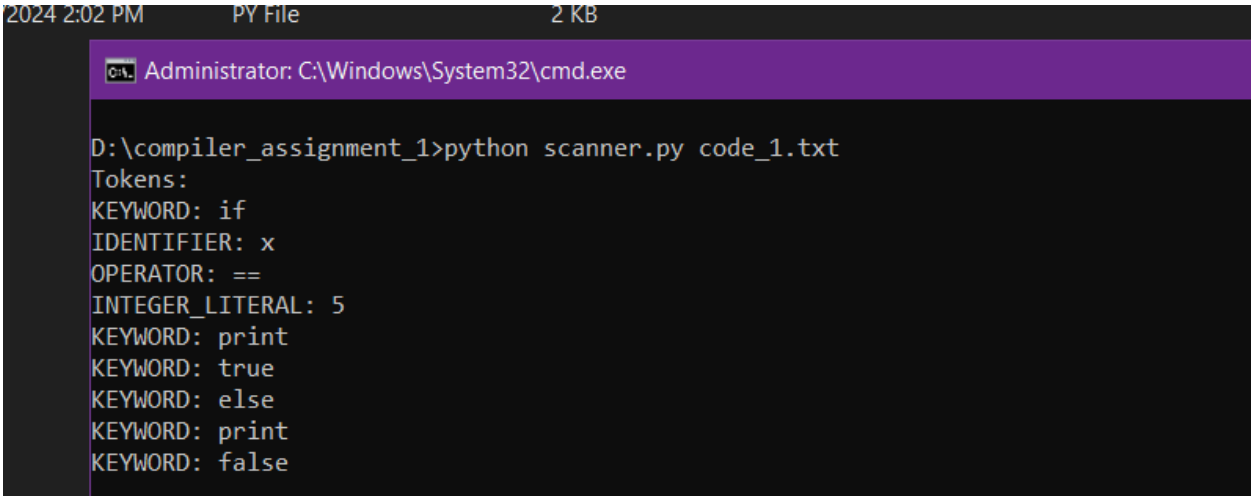
Lex Errors are handled by printing error messages when unrecognized token is encountered.

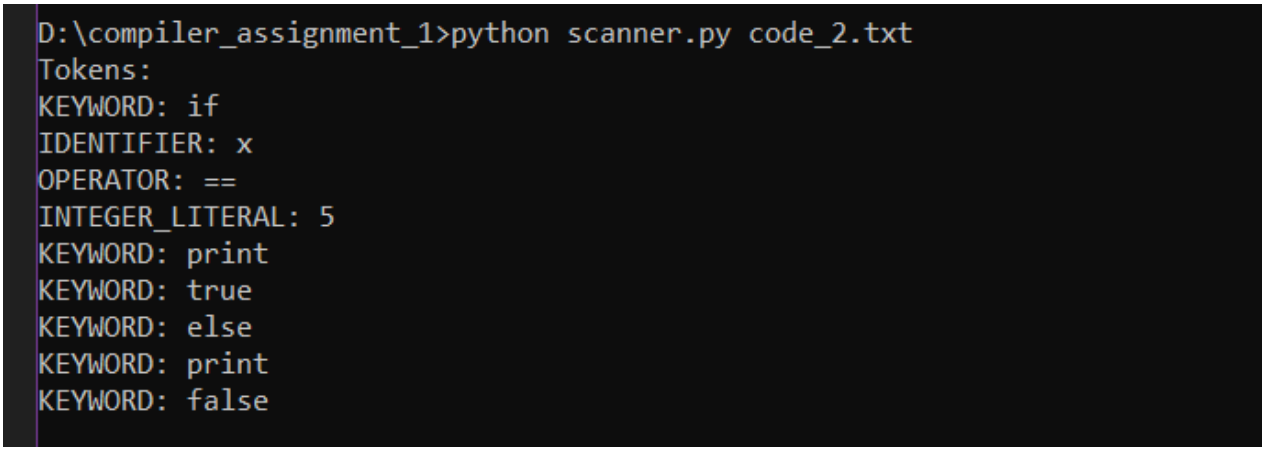
How to Execute Program:

The Program can be executed via command line by using the following command:

Python scanner.py code*.txt

Test Cases:





```
KEYWORD: false

D:\compiler_assignment_1>python scanner.py code_3.txt
Tokens:
IDENTIFIER: result
OPERATOR: =
INTEGER_LITERAL: 3
OPERATOR: +
OPERATOR: (
IDENTIFIER: y
OPERATOR: *
INTEGER_LITERAL: 2
OPERATOR: )
KEYWORD: print
IDENTIFIER: result
```

```
D:\compiler_assignment_1>python scanner.py code_4.txt
Tokens:
IDENTIFIER: a
OPERATOR: =
INTEGER_LITERAL: 10
IDENTIFIER: b
OPERATOR: =
KEYWORD: false
KEYWORD: print
IDENTIFIER: a
OPERATOR: +
IDENTIFIER: b
```