# 

# Group Names: ID Omar Ahmed 221027530 Abdullah Hani 221027696 Yassin Zakaria 221017794 Abeer Hesham 221017800 Hasna Amer 221017725

# Presented to: Dr Ayman Helmy Eng Habiba Amr

# Documentation

## 1. Introduction

This program is a lexical analyzer (lexer) written in Python that processes user-provided source code into tokens. Tokens are categorized into types such as identifiers, numbers, operators, keywords, strings, parentheses, and semicolons. This tool provides a foundation for compilers or interpreters by breaking source code into its fundamental elements.

## 2. System Design

Core Components:

* - Token Structure: A structure to hold token type and value.  
  - Lexer Class: Encapsulates the logic for parsing source code into tokens. Includes multiple helper functions for token generation.  
  - Error Handling: Handles unexpected input with meaningful runtime error messages.  
  - Console Interface: A main function loop to interact with users for repeated input.

Workflow:

1. 1. The lexer reads the input string character by character.  
   2. It categorizes characters into token types using helper functions.  
   3. Tokens are stored and displayed to the user.  
   4. The process repeats until the user exits.

## 3. Implementation Details

Token Enumeration:

Defines token types such as IDENTIFIER, NUMBER, OPERATOR, KEYWORD, STRING, PARENTHESIS, and ENDING.

Helper Functions:

* - tokenize**\_**number: Recognizes numeric literals.  
  - tokenize**\_**identifier**\_**or**\_**keyword: Differentiates between identifiers and keywords using a predefined keyword list.  
  - tokenize**\_**string**\_**literal: Handles strings enclosed in double quotes.  
  - tokenizeOperator: Detects operators, including single and multi-character ones (e.g., ==, <=).  
  - tokenize**\_**parenthesis: Detects parentheses.  
  - tokenize**\_**ending: Handles semicolons.

Error Handling:

Throws exceptions for unexpected characters and provides meaningful error messages with the problematic character and position.

## 4. User Guide

1. Compilation

1. Compile the program with a text editor that has python compiler(ex: visual studio )
2. In the output:

- Enter source code when prompted.  
- View tokenized output.  
- Decide whether to continue or exit.

## 5. Testing Strategy

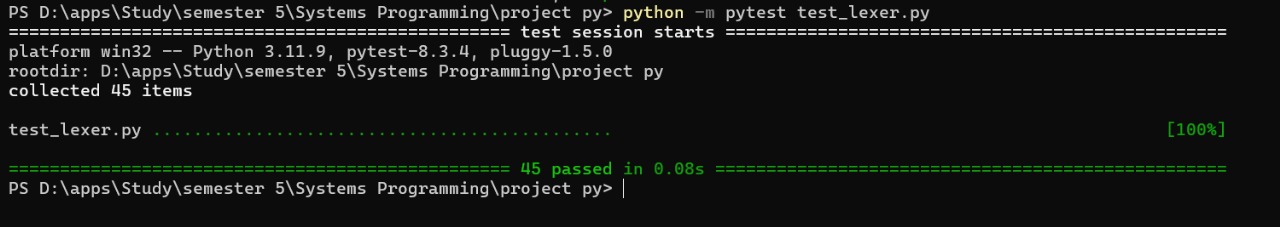
Unit Testing:

* - Validated each helper function for correctness with test cases such as valid numbers, keywords, and strings.  
  - Invalid inputs like unterminated strings or illegal characters were tested to ensure appropriate error messages.

Integration Testing:

* - Verified the entire lexer with complete source code snippets.

Edge Case Testing:

* - Handled edge cases like consecutive operators (++, >=), invalid string literals, and empty input.

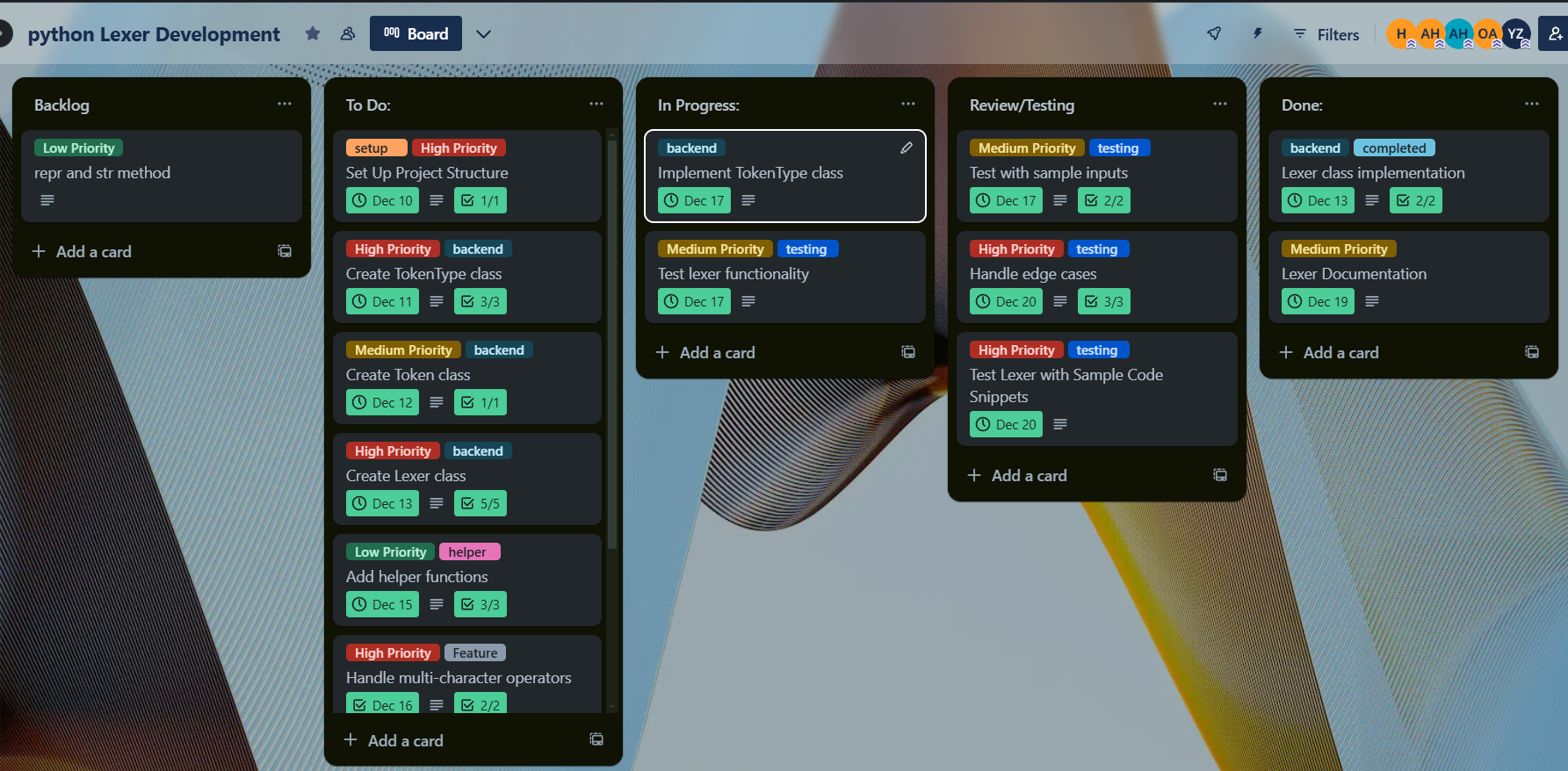
## 6. Tools for Quality of Code

* A computer screen shot of a computer program

  Description automatically generated- Static Analysis: Used tools like **pylint** to enforce standards and detect potential bugs.  
  - Unit Testing Framework: Employed **pylint** to automate and validate tests.

## 7. Trello Board & GitHub: <https://trello.com/b/sVZ7OWlk/c-lexer-development>

Trello Home page:



1. To Do:

A screenshot of a phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

1. In progress:

A screenshot of a phone

Description automatically generated

1. Review/test:

A screenshot of a computer

Description automatically generated

1. Done:

A screenshot of a computer

Description automatically generated

GitHub:

* - Link for github repository: [**https://github.com/Omarahmed1st/Lexer-Program**](https://github.com/Omarahmed1st/Lexer-Program)