

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Inginerie Software și Automatică**

**Tabanschi Nichita FAF-222**

**Report**

*Laboratory work n.3*

***of Limbaje Formale și Automate***

Checked by:

**Dumitru Cretu**, *university assistant*

DISA, FCIM, UTM

**Chișinău – 2024**

**Topic: Lexer**

## Overview

    The term lexer comes from lexical analysis which, in turn, represents the process of extracting lexical tokens from a string of characters. There are several alternative names for the mechanism called lexer, for example tokenizer or scanner. The lexical analysis is one of the first stages used in a compiler/interpreter when dealing with programming, markup or other types of languages.     The tokens are identified based on some rules of the language and the products that the lexer gives are called lexemes. So basically the lexer is a stream of lexemes. Now in case it is not clear what's the difference between lexemes and tokens, there is a big one. The lexeme is just the byproduct of splitting based on delimiters, for example spaces, but the tokens give names or categories to each lexeme. So the tokens don't retain necessarily the actual value of the lexeme, but rather the type of it and maybe some metadata.

## Objectives:

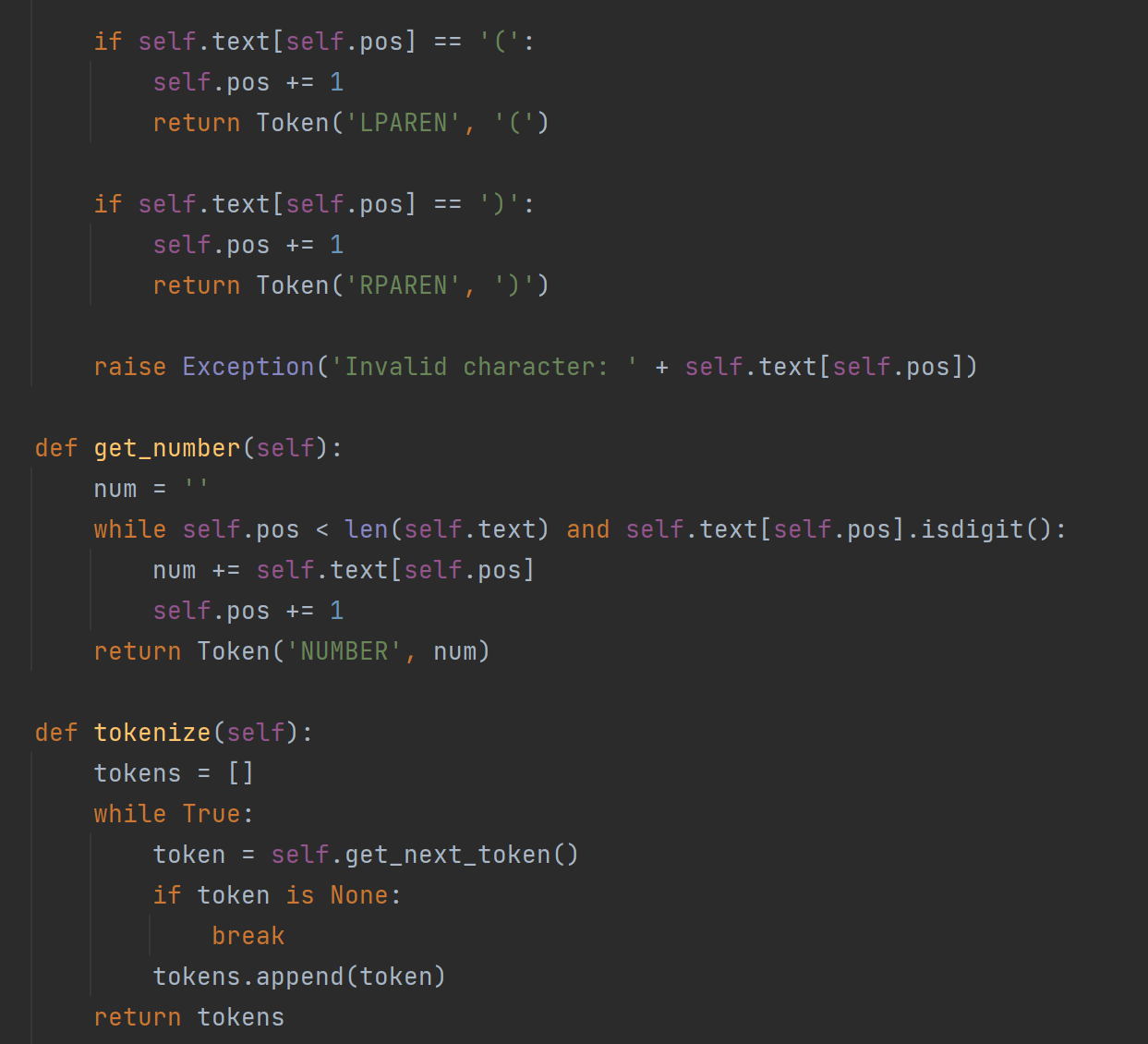
1. Understand what lexical analysis [1] is.
2. Get familiar with the inner workings of a lexer/scanner/tokenizer.
3. Implement a sample lexer and show how it works.

## Implementation tips:

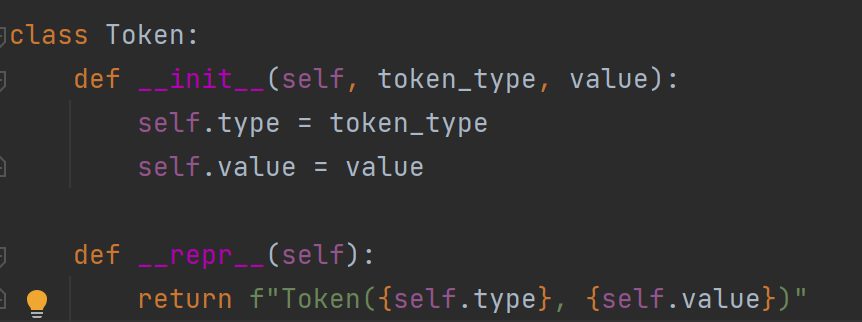
1. You can find implementation tips on the reference [2].
2. Please take into consideration the indicated structure of the project. Find a suitable place for the lexer implementation and for the new report of course.



This structure separates concerns by having a module (**token.py**) dedicated to token representation and another module (**lexer.py**) dedicated to lexical analysis. It follows the principles of encapsulation and modularity, making the codebase easier to understand, maintain, and extend.



In the **lexer.py** module, we define the **Lexer** class responsible for tokenizing input text. It contains methods **get\_next\_token**, **get\_number**, and **tokenize**, as in the previous implementation. We import the **Token** class from the **token** module to create token objects.



In this module, we define a **Token** class representing individual tokens. Each token has a type (**token\_type**) and a value (**value**). We implement a **\_\_repr\_\_** method to provide a string representation of the token for debugging purposes.

**Output:**

****