1. Lexical analysis is a phase of compilation process when the source code is taken and modified into list of tokens. All white spaces and comments are ignored by compiler and removed. This is the first stage of compilation.
2. On this stage the used PLY module is lex.py. Tokenizer splits input string into separate tokens. To work with it properly several things should be defined. List of token names with the constant name “tokens”.

   tokens = (“NUMBER”, “MINUS”, “DOT”...)

   Then, for each token it should be described with a reserved name t_TOKENNAME. There are two ways to describe it: first - simple tokens are described as a regular expressions, second - they can also be described as a function with a similar reserved name.

```
def t_NUMBER(t):
     r'\d+'
     t.value = int(t.value)
     return t

t_DIVIDE  = r'/'
```

   There are also several reserved names for special occasions, for instance t_ignore and t_error.

3. The way of recognition special cases:
   a. Keywords - are stored in the list “reserved” and all potential variables, function names etc are checked to be in this list. Also all keywords has there own tokens with names t_KEYWORD.
   b. Comments are defined with a regular expression {.*} stored in a reserved token `t_ignore_COMMENT`.
   c. White Spaces between tokens are also defined with regular expression and stored in a reserved token `t_ignore  = '\n \r\t'`.
   d. All operators are stored in a list of regular tokens and defined with regular expressions.
   e. Integer literals are defined with a function which recognize them using regexp and then return integer as a token value.
   f. String literals are also defined as a function which recognize them with a complicated regexp.
   g. Same as for string literals (with different rule).
   h. Same as for string literals (with different rule).

4. The way to distinguish between:
   a. Function names - constant names. There is a rule related to it: constant names could only have upper case letters, while function names should start with [A-Z] and then have one or more symbol from this list [a-z][0-9]{_}.
   b. Keywords - variable names. All keywords are stored initially in a special list “reserved” so the program just check whether the word in it or not.
   c. Minus ‘-’ and Right arrow ‘->’ it is done with regexp: `t_RARROW = r'->', t_MINUS = r'-'.`
   d. String literals & variable names also have different rules of construction: string literal is anything inside the quotation marks, while variable name must start with lowercase letter and be followed by at least one letter from [a-z][A-Z][0-9]{_}.
   e. Comments & other code. Comments are defined in a reserved token t_ignore_token = {.*} and then they are totally ignored while collecting tokens.
   f. Tuple names & two compared variables. Tuple names are defined with regexp `r'<[a-z]+>'.`

5. No extras.
6. I think it was really nice assignment. PLY is really useful and it was great to find out how to use it. It also wasn’t really difficult to implement this functionality. However, I think it would much and much more complicated to implement such code “manually from start to end” without using special libraries.