Lab 2

The task is to create a lexical analyser (scanner) for a simple language allowing matrix computation. The lexical analyser should recognise the following lexems:

- type specifiers: `int`, `float`, `string`
- binary operators: +, -, *, /
- matrix element-wise binary operators: .+, .-, .*, ./
- assignment operators: =, +=, -=, *=, /=
- relational operators: <, >, <=, >=, !=, ==
- parentheses: (,), [,], {,}
- range selection: :
- matrix transpose: '
- comma and semicolon: , ;
- reserved keywords: `if`, `else`, `for`, `while`
- instructions `break`, `continue` and `return`
- reserved keywords: `eye`, `zeros` and `ones`
- reserved keyword: `print`
- identifiers
- integer numbers
- float numbers

For recognised lexems created scanned should return:

- corresponding token
- recognised lexem
- line number
- optionally column number can be returned

The following characters should be ignored (discarded):

- white characters: spaces, tabs, newlines
- comments: comments begins with # and spans until end of line

Example.
For the following code:

```
A = zeros(5);  # create 5x5 matrix filled with zeros
B = ones(7);   # create 7x7 matrix filled with ones
I = eye(10);   # create 10x10 matrix filled with ones on diagonal and zeros elsewhere
D1 = A.+B' ; # add element-wise A with transpose of B
D2 -= A.-B' ; # substract element-wise A with transpose of B
D3 *= A.*B' ; # multiply element-wise A with transpose of B
D4 /= A./B' ; # divide element-wise A with transpose of B
```

lexical analyser should return the following sequence of print it on the standard output:

```
(1,1): ID(A)
(1,2): =(=)
(1,4): ZEROS(zeros)
(1,9): ((()
(1,10): INTNUM(5)
(1,11): )())
```

```
(1,12): ;(;)
(2,1): ID(B)
(2,3): =(=)
(2,5): ONES(ones)
(2,9): (((()
(2,10): INTNUM(7)
(2,11): )())
(2,12): ;(;)
(3,1): ID(I)
(3,3): =(=)
(3,5): EYE(eye)
(3,8): (((()
(3,9): INTNUM(10)
(3,11): )())
(3,12): ;(;)
(4,1): ID(D1)
(4,4): =(=)
(4,6): ID(A)
(4,7): DOTADD(.+)
(4,9): ID(B)
(4,10): '('
(4,12): ;(;)
(5,1): ID(D2)
(5,4): SUBASSIGN(-=)
(5,7): ID(A)
(5,8): DOTSUB(.-)
(5,10): ID(B)
(5,11): '('
(5,13): ;(;)
(6,1): ID(D3)
(6,4): MULASSIGN(*=)
(6,7): ID(A)
(6,8): DOTMUL(.*)
(6,10): ID(B)
(6,11): '('
(6,13): ;(;)
(7,1): ID(D4)
(7,4): DIVASSIGN(/=)
(7,7): ID(A)
(7,8): DOTDIV(./)
(7,10): ID(B)
(7,11): '('
(7,13): ;(;)
```

- To solve the task, scanner generator PLY should be used.
- Scanner should detect lexically incorrect input. For incorrect lines, line number along with information that an error ocurred should be printed out.
- To create the scanner, you can use following file main.py.