-We Imported the os library to be able to open the folder by path and the re library to used regular expression in order to evaluate our user identifiers and their legibility

-**Lists and dictionaries:**

-list of delimiters used to identify delimiters to increment line further down the code.

-The primary function of these lists id to come the nature of the input/lexeme and determine to which category it belongs to be able to tokenize it later

-list of reserved words: reserved\_list

-list of operators: operator\_list

-list of punctuation: punctuation\_list

-dicitionary of all tokens and their respective token number: token\_list\_numbers

-symbol\_table initialized and to store reserved words and to add to it later the user created identifiers : symbol\_table

🡺dictionaries with lexemes and corresponding token

lexim\_token\_reserved

lexim\_token\_opearator

lexim\_token\_punctuation

**matching function:** they check if the lexeme is legal and either return error or token

# check for number:

def check\_num(digit):

# check for operator:

def check\_operator(op):

# check for punctuation:

def check\_punctuation(pun):

# check for identifier + append to symbol table (if the id is not there already) + plus returns either token or error:

def check\_id(id):

# check for function identifier and append it to symbol table if it is not already added and returns either token or error:

def check\_fun\_id(id):

check if reserved word is in reserved word list and if it is it returns its respective token without adding identifier to the symbol table:

def check\_reserved(word):

**function to print to screen and writes into the output file:**

-Its arguments are the output file name, the token and the input that corresponds to that token

def print\_to\_screen\_file(file,line,token,inputy):

it prints to the screen this string:

print("Line " + str(line) + " Token " + "#" +str(number) + ": " + inputy + "\n")

-writes in the file: in case of a user identifier then the id needs to be written along with the token else only the token will be written in the file.

if inputy in id\_list or inputy.isnumeric():

file.write(str(line) + " " + str(number) + " " + str(token) + " " + inputy + "\n")

else:

file.write(str(line) + " " + str(number) + " " + str(token) + "\n")

**main function def main(file)::** where the output and the input file is opened and proceeds with reading character and tokenizing it:

-it takes file path as an input.

-opens the input file and reads from it, it also opens the output file to send it to the previous function.

-This function handles multiple cases of the possible nature/category of the lexeme. It also handles specific cases such as encountering a comment in that case a line must be skipped, in case of a space that needs to be ignored and in case of delimiters in which the line needs to be incremented.

-In each case it calls the following function:

print\_to\_screen\_file(outputfile,line,check\_num(number),number): that will handl both testing the legality of lexemes, tokenizing, outputting and writing to the file.

-The symbole table is then printed to the screen for testing purposed.

-When the loop is broken the output file is closed as well as the input file.

**File and folder:**

-Before calling the main we need to first to provide the code with

directory\_path = "C:\\Users\\benou\\Desktop\\testfolder"

-loops through all files in the folder to find the file that ends with \*.txt and send its path to the main function to read, tokenize and output with the help of other functions.