ACD Lab ASSIGNMENT 4

Name : Aahan Singh Charak

Registration Number :189301024

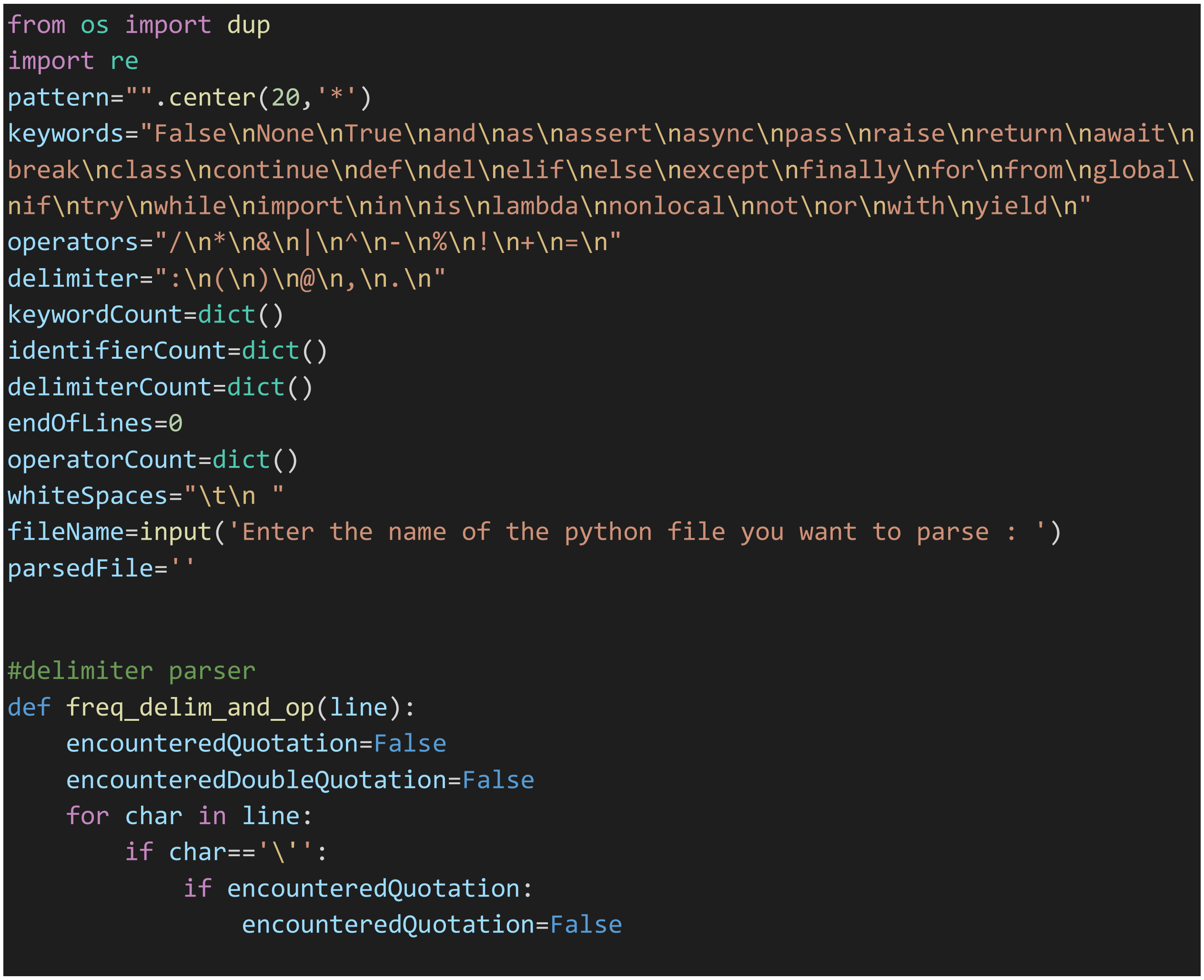
Section : CSE A

Roll no : 5

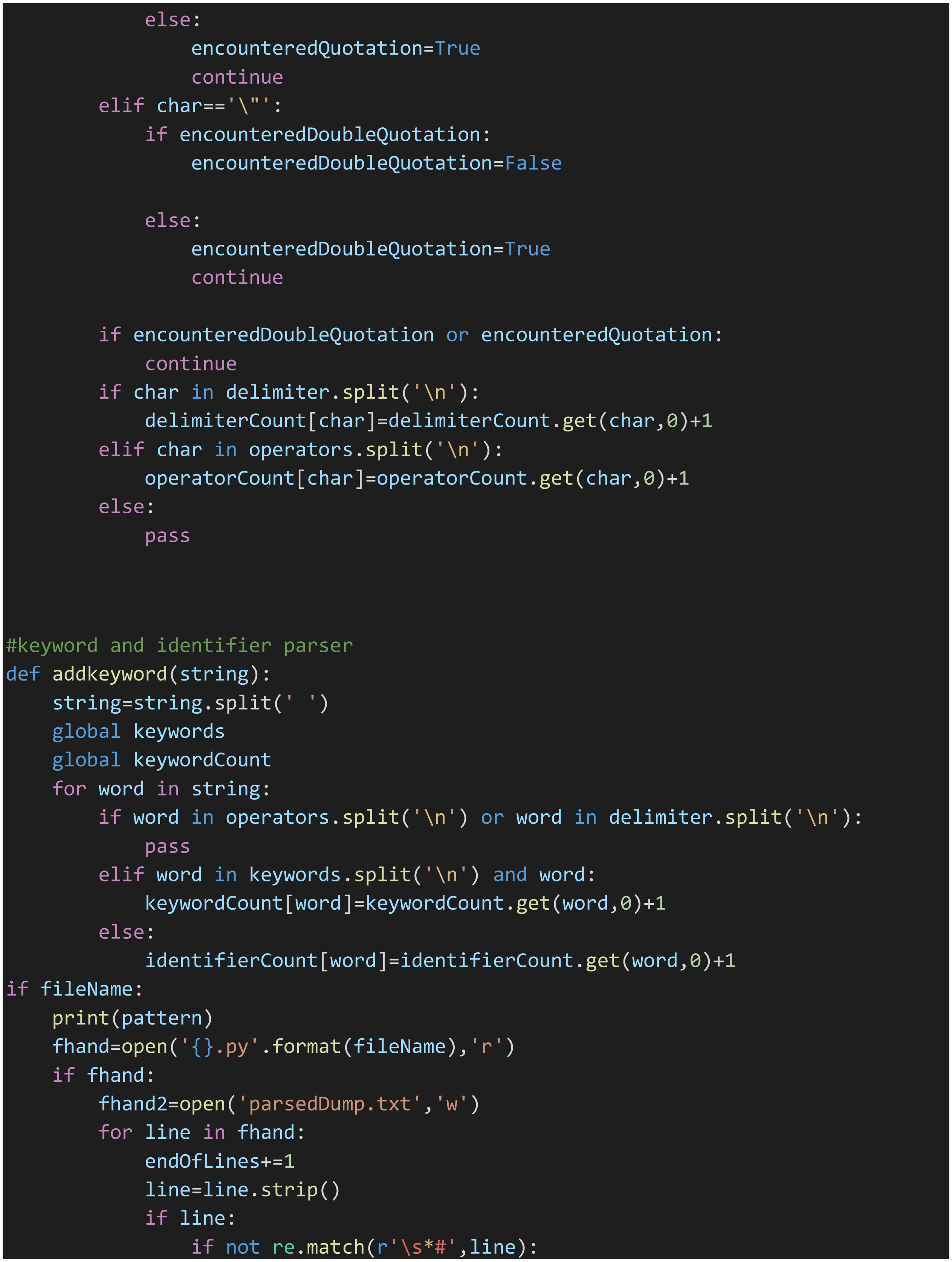
Question : Lexical Analyzer

Solution: Python

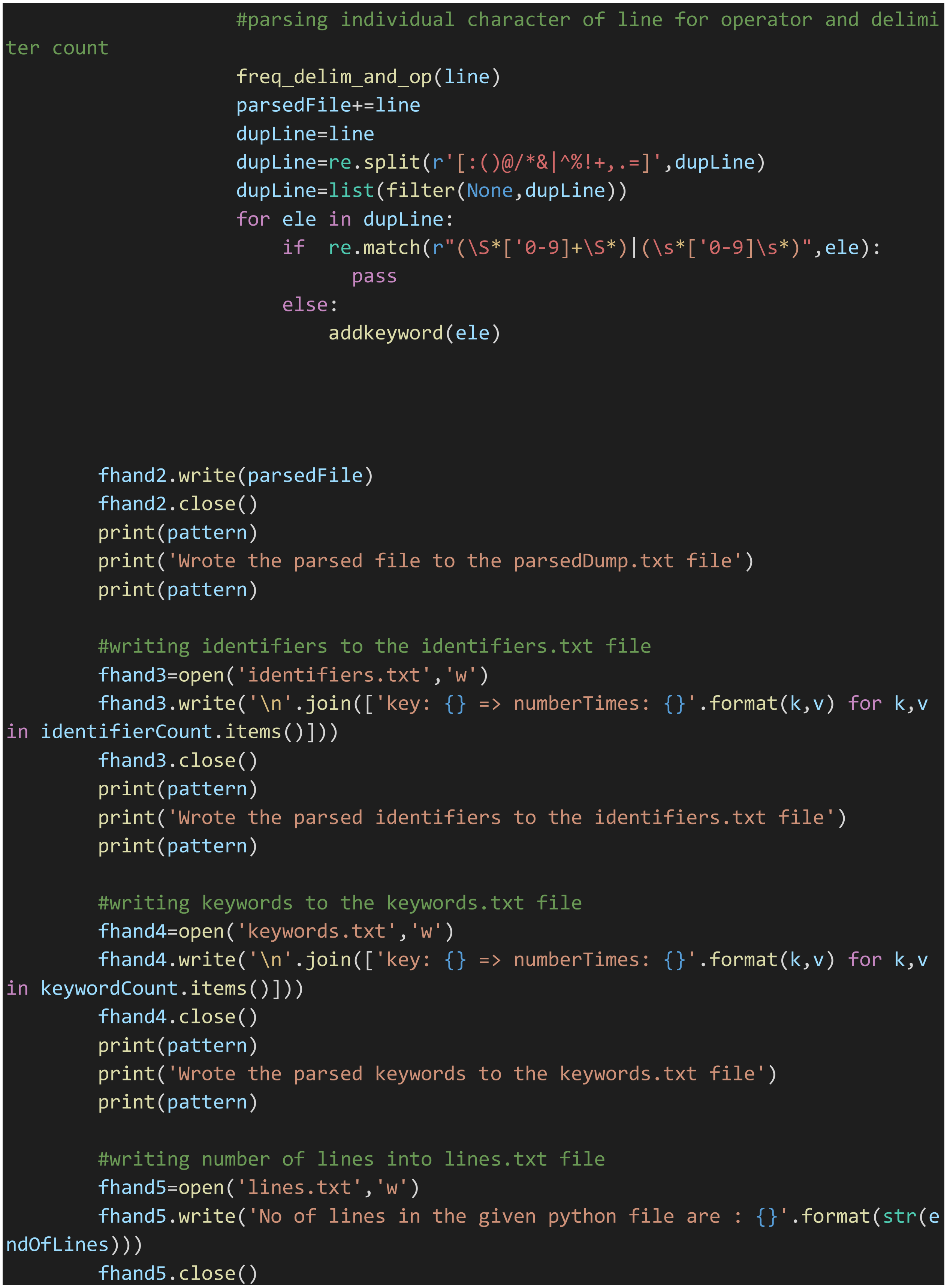
Analyzer Code:



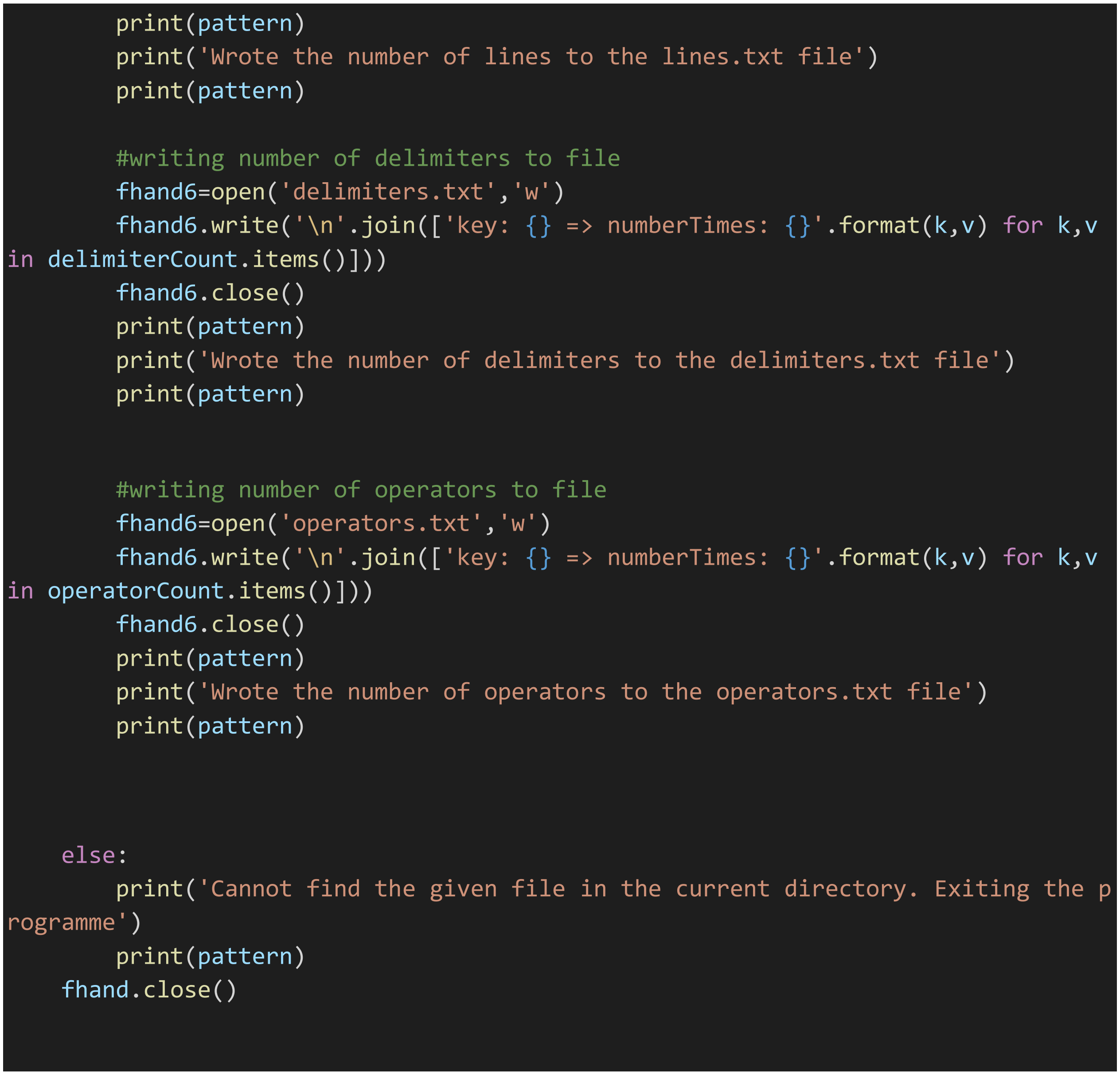
```
from os import dup
import re
pattern="".center(20,'*')
keywords="False\nNone\nTrue\nand\nas\nassert\nasync\npass\nraise\nreturn\nawait\n
break\nclass\ncontinue\ndef\ndel\nelif\nelse\nexcept\nfinally\nfor\nfrom\nglobal\
nif\ntry\nwhile\nimport\nin\nis\nlambda\nnonlocal\nnot\nor\nwith\nyield\n"
operators="/\n*\n&\n|\n^\n-\n%\n!\n+\n=\n"
delimiter=":\n(\n)\n@\n,\n.\n"
keywordCount=dict()
identifierCount=dict()
delimiterCount=dict()
endOfLines=0
operatorCount=dict()
whiteSpaces="\t\n "
fileName=input('Enter the name of the python file you want to parse : ')
parsedFile=''


#delimiter parser
def freq_delim_and_op(line):
    encounteredQuotation=False
    encounteredDoubleQuotation=False
    for char in line:
        if char=='\'':
            if encounteredQuotation:
                encounteredQuotation=False
```



```
            else:
                encounteredQuotation=True
                continue
        elif char=='\"':
            if encounteredDoubleQuotation:
                encounteredDoubleQuotation=False

            else:
                encounteredDoubleQuotation=True
                continue

        if encounteredDoubleQuotation or encounteredQuotation:
            continue
        if char in delimiter.split('\n'):
            delimiterCount[char]=delimiterCount.get(char,0)+1
        elif char in operators.split('\n'):
            operatorCount[char]=operatorCount.get(char,0)+1
        else:
            pass



#keyword and identifier parser
def addkeyword(string):
    string=string.split(' ')
    global keywords
    global keywordCount
    for word in string:
        if word in operators.split('\n') or word in delimiter.split('\n'):
            pass
        elif word in keywords.split('\n') and word:
            keywordCount[word]=keywordCount.get(word,0)+1
        else:
            identifierCount[word]=identifierCount.get(word,0)+1
if fileName:
    print(pattern)
    fhand=open('{}.py'.format(fileName),'r')
    if fhand:
        fhand2=open('parsedDump.txt','w')
        for line in fhand:
            endOfLines+=1
            line=line.strip()
            if line:
                if not re.match(r'\s*#',line):
```



```
                    #parsing individual character of line for operator and delimiter count
                    freq_delim_and_op(line)
                    parsedFile+=line
                    dupLine=line
                    dupLine=re.split(r'[:()@/*&|^%!+,.=]',dupLine)
                    dupLine=list(filter(None,dupLine))
                    for ele in dupLine:
                        if  re.match(r"(\S*['0-9]+\S*)|(\s*['0-9]\s*)",ele):
                              pass
                        else:
                            addkeyword(ele)




        fhand2.write(parsedFile)
        fhand2.close()
        print(pattern)
        print('Wrote the parsed file to the parsedDump.txt file')
        print(pattern)

        #writing identifiers to the identifiers.txt file
        fhand3=open('identifiers.txt','w')
        fhand3.write('\n'.join(['key: {} => numberTimes: {}'.format(k,v) for k,v in identifierCount.items()]))
        fhand3.close()
        print(pattern)
        print('Wrote the parsed identifiers to the identifiers.txt file')
        print(pattern)

        #writing keywords to the keywords.txt file
        fhand4=open('keywords.txt','w')
        fhand4.write('\n'.join(['key: {} => numberTimes: {}'.format(k,v) for k,v in keywordCount.items()]))
        fhand4.close()
        print(pattern)
        print('Wrote the parsed keywords to the keywords.txt file')
        print(pattern)

        #writing number of lines into lines.txt file
        fhand5=open('lines.txt','w')
        fhand5.write('No of lines in the given python file are : {}'.format(str(endOfLines)))
        fhand5.close()
```



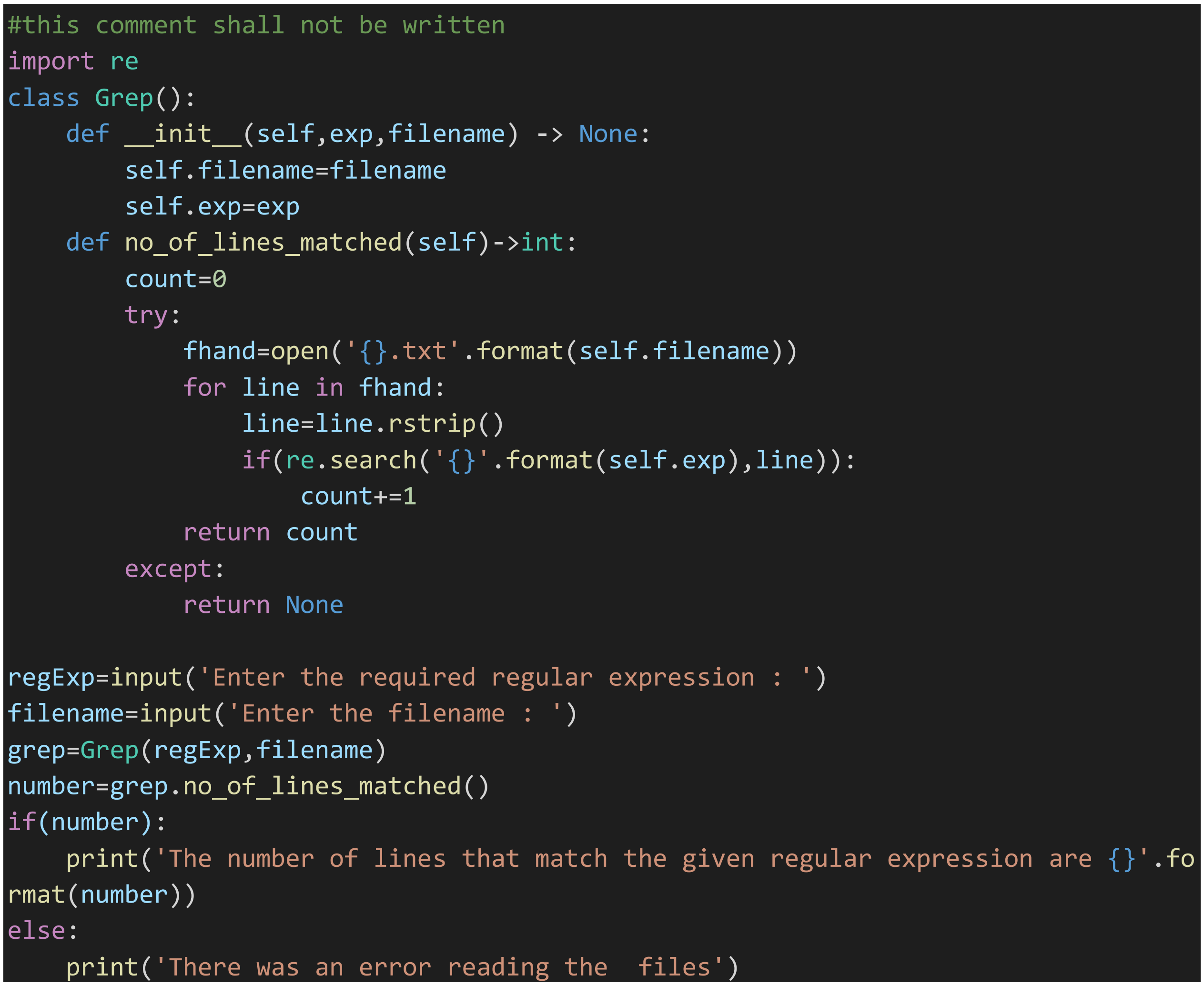
```
        print(pattern)
        print('Wrote the number of lines to the lines.txt file')
        print(pattern)

        #writing number of delimiters to file
        fhand6=open('delimiters.txt','w')
        fhand6.write('\n'.join(['key: {} => numberTimes: {}'.format(k,v) for k,v
in delimiterCount.items()]))
        fhand6.close()
        print(pattern)
        print('Wrote the number of delimiters to the delimiters.txt file')
        print(pattern)


        #writing number of operators to file
        fhand6=open('operators.txt','w')
        fhand6.write('\n'.join(['key: {} => numberTimes: {}'.format(k,v) for k,v
in operatorCount.items()]))
        fhand6.close()
        print(pattern)
        print('Wrote the number of operators to the operators.txt file')
        print(pattern)



    else:
        print('Cannot find the given file in the current directory. Exiting the p
rogramme')
        print(pattern)
    fhand.close()
```

File to be lexically analyzed:

I have made an analyzer that analyzes python file

Name file as script.py in order to run correctly and it should be in the same folder as the above driver python file

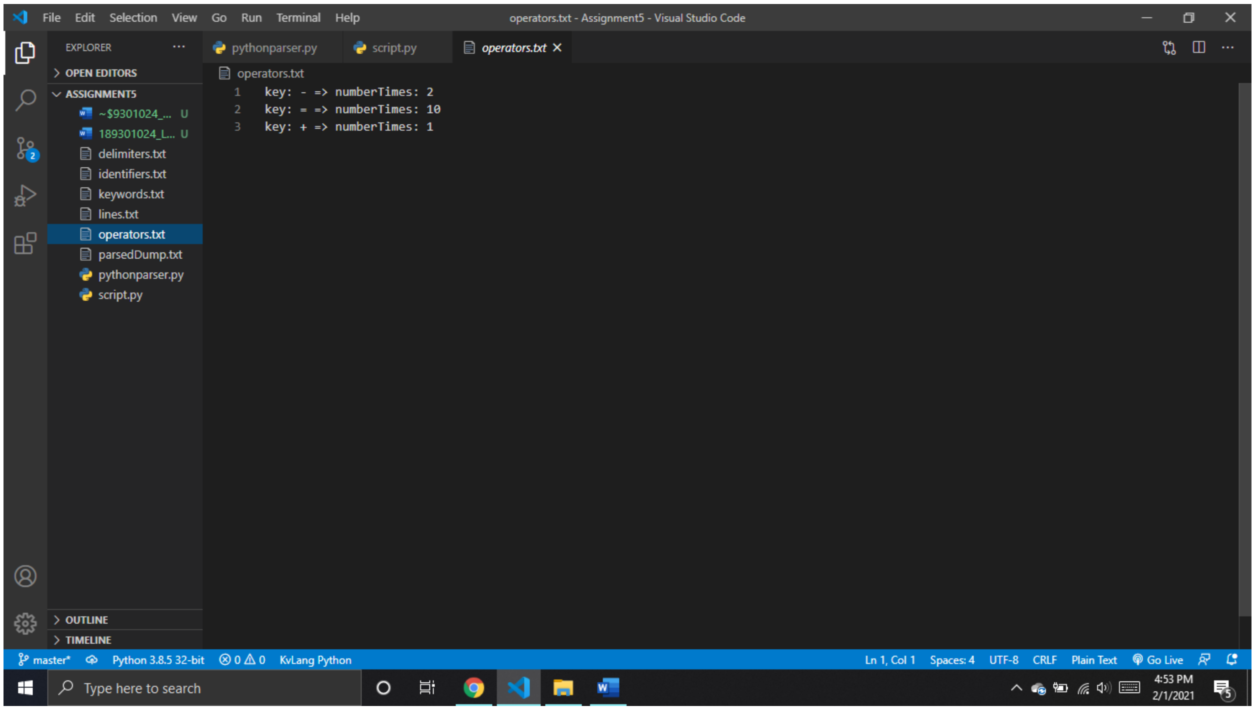
Script.py:



```
#this comment shall not be written
import re
class Grep():
    def __init__(self,exp,filename) -> None:
        self.filename=filename
        self.exp=exp
    def no_of_lines_matched(self)->int:
        count=0
        try:
            fhand=open('{}.txt'.format(self.filename))
            for line in fhand:
                line=line.rstrip()
                if(re.search('{}'.format(self.exp),line)):
                    count+=1
            return count
        except:
            return None

regExp=input('Enter the required regular expression : ')
filename=input('Enter the filename : ')
grep=Grep(regExp,filename)
number=grep.no_of_lines_matched()
if(number):
    print('The number of lines that match the given regular expression are {}'.fo
rmat(number))
else:
    print('There was an error reading the  files')
```

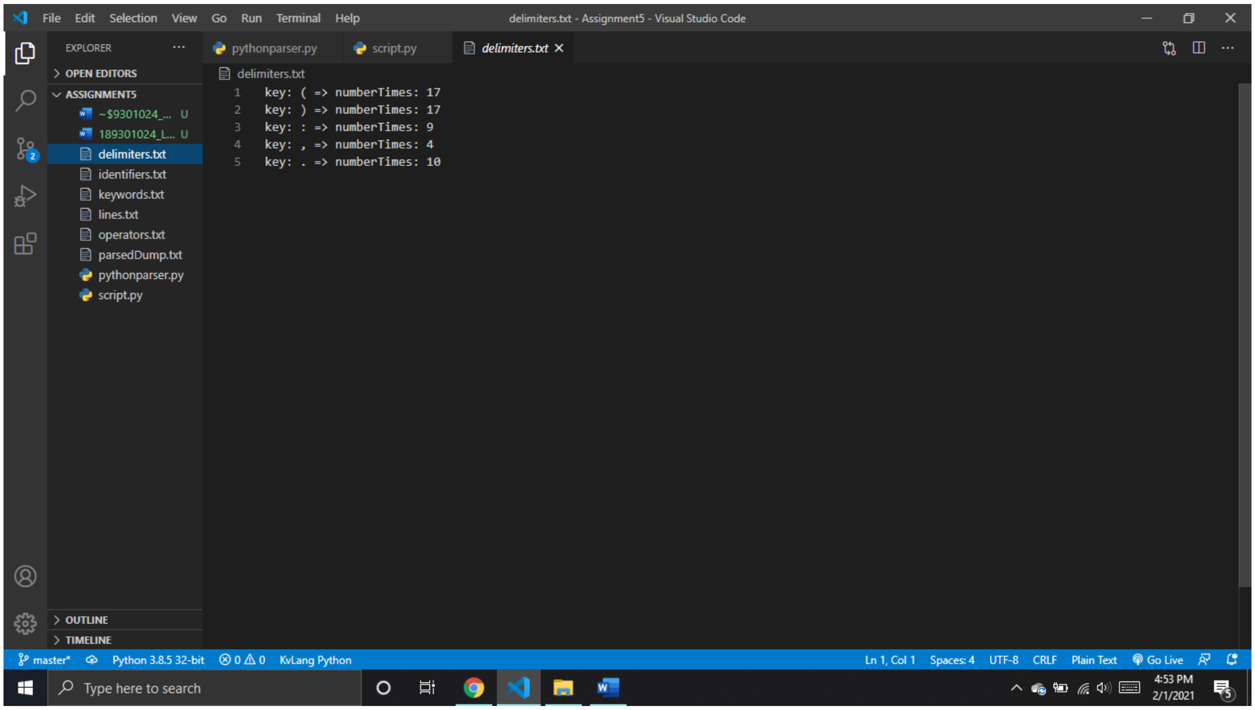
Output:

Operators.txt



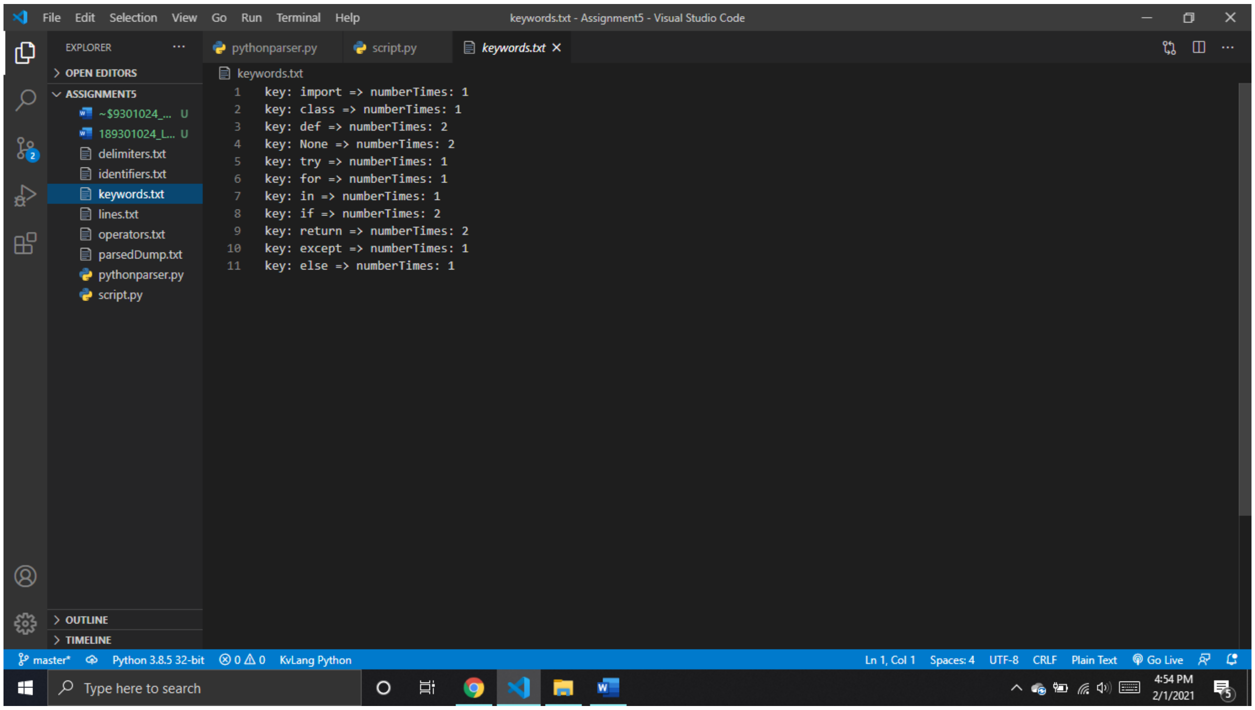
```
key: - => numberTimes: 2
key: = => numberTimes: 10
key: + => numberTimes: 1
```

Delimiters.txt



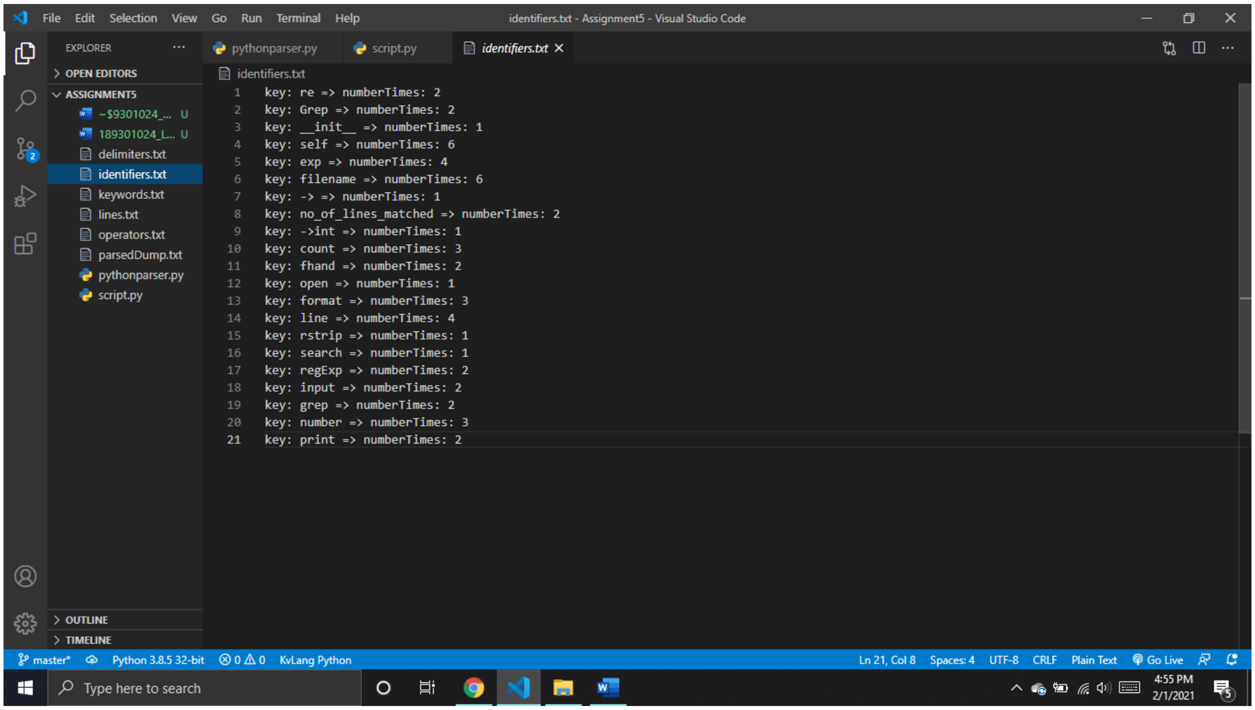
```
key: ( => numberTimes: 17
key: ) => numberTimes: 17
key: : => numberTimes: 9
key: , => numberTimes: 4
key: . => numberTimes: 10
```

Keywords.txt



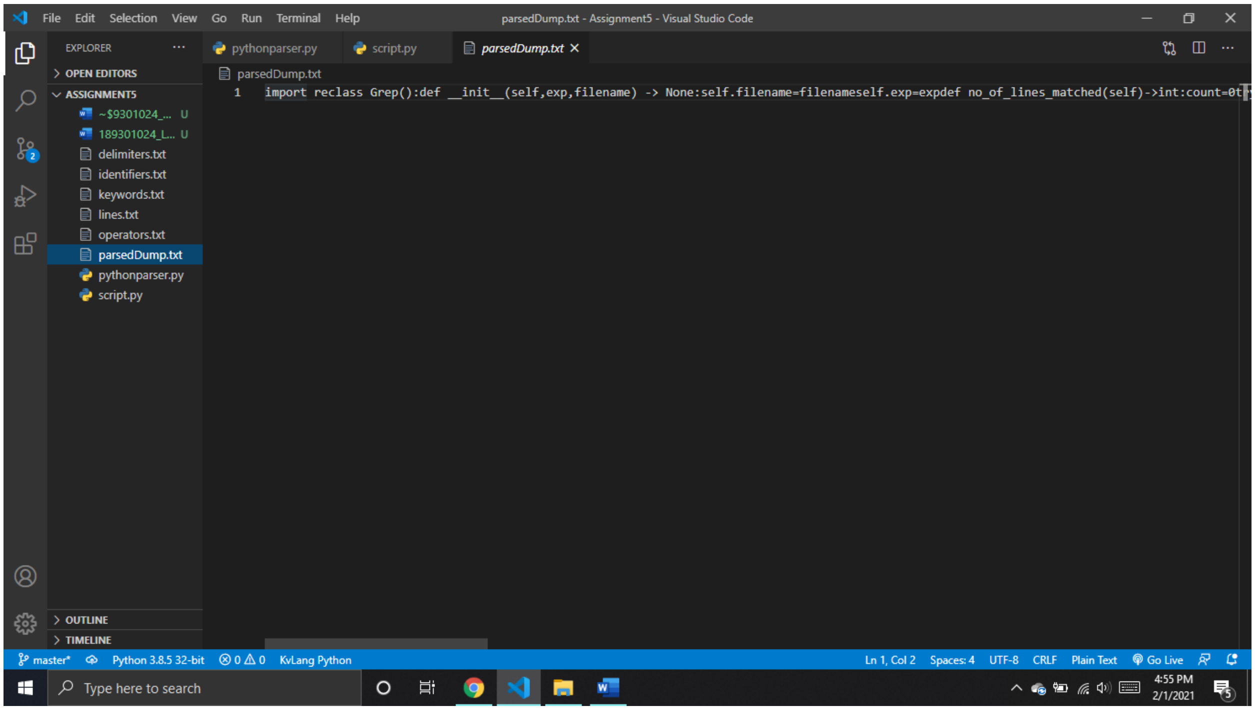
```
key: import => numberTimes: 1
key: class => numberTimes: 1
key: def => numberTimes: 2
key: None => numberTimes: 2
key: try => numberTimes: 1
key: for => numberTimes: 1
key: in => numberTimes: 1
key: if => numberTimes: 2
key: return => numberTimes: 2
key: except => numberTimes: 1
key: else => numberTimes: 1
```

Identifiers.txt



```
key: re => numberTimes: 2
key: Grep => numberTimes: 2
key: __init__ => numberTimes: 1
key: self => numberTimes: 6
key: exp => numberTimes: 4
key: filename => numberTimes: 6
key: -> => numberTimes: 1
key: no_of_lines_matched => numberTimes: 2
key: ->int => numberTimes: 1
key: count => numberTimes: 3
key: fhand => numberTimes: 2
key: open => numberTimes: 1
key: format => numberTimes: 3
key: line => numberTimes: 4
key: rstrip => numberTimes: 1
key: search => numberTimes: 1
key: regExp => numberTimes: 2
key: input => numberTimes: 2
key: grep => numberTimes: 2
key: number => numberTimes: 3
key: print => numberTimes: 2
```

File which includes code without preceding whitespaces and tabs Named as parsedDump.txt



```
import reclass Grep():def __init__(self,exp,filename) -> None:self.filename=filenameself.exp=expdef no_of_lines_matched(self)->int:count=0t
```