گزارش فاز دوم پروژه درس طراحی کامپایلر:

:PLY

PLY یک ابزار مناسب برای پیاده سازی بخش مهم کامپایلر lex و yacc به زبان پایتون می باشد. PLY شامل دو ماژول مجزا lex.py و yacc.py است که هردو در پکیج ply موجود هستند. در این فاز ما میخواهیم از ماژول lex.py استفاده نماییم و در ادامه نحوه عملکرد این ماژول را شرح میدهیم.

:LEX

Lex.py در شکستن برنامه ورودی به مجموعه توکن های تعریف شده به صورت عبارت منظم مورد اسفاده قرار میگیرد. به این صورت که کاراکتر های برنامه ورودی را بررسی می نماید و با توکن های معرفی شده به صورت عبارت منظم مقایسه مینماید در صورتی که کاراکتر ها به گونه ای چیده شده و با یکی از توکن ها مطابقت داشته باشند آنگاه آ« دسته کاراکتر را از برنامه ورودی جدا نموده و در گروه توکن مورد نظر قرار داده و به صورت خوروجی برمی گرداند و سپس به سراغ کاراکتر های بعدی رفته و اینگونه تمام برنامه ورودی را به صورت توکن های تعریف شده شسکته و تقسیم بندی مینماید.

:Token List

همانگونه که در بالا اشاره نمودیم، ابزار Lex شامل یک بخش ضروری به نام token list می باشد که شامل لیست تمام توکن هایی است که ممکن است در برنامه شناسایی شوند.

:Specification of tokens

با اعلام لیست توکن ها می بایست هر توکن را جداگانه با استفاده از قوانین عبارت های منظم تعریف نماییم که این عمل توسط ماژولre در پایتون انجام می شود. این شناسایی به دو صورت انجام میشود: اگر توکن مورد نظر ساده باشد کافیست تعریف قانون مربوط به آن توکن به صورت یک string انجام شود. در غیر اینصورت اگر نیاز به انجام عملیات های مختلف در حین تعریف توکن باشد باید تعریف توکن مورد نظر را به صورت یک تابع پیاده سازی کنیم.

:Line numbers and positional information

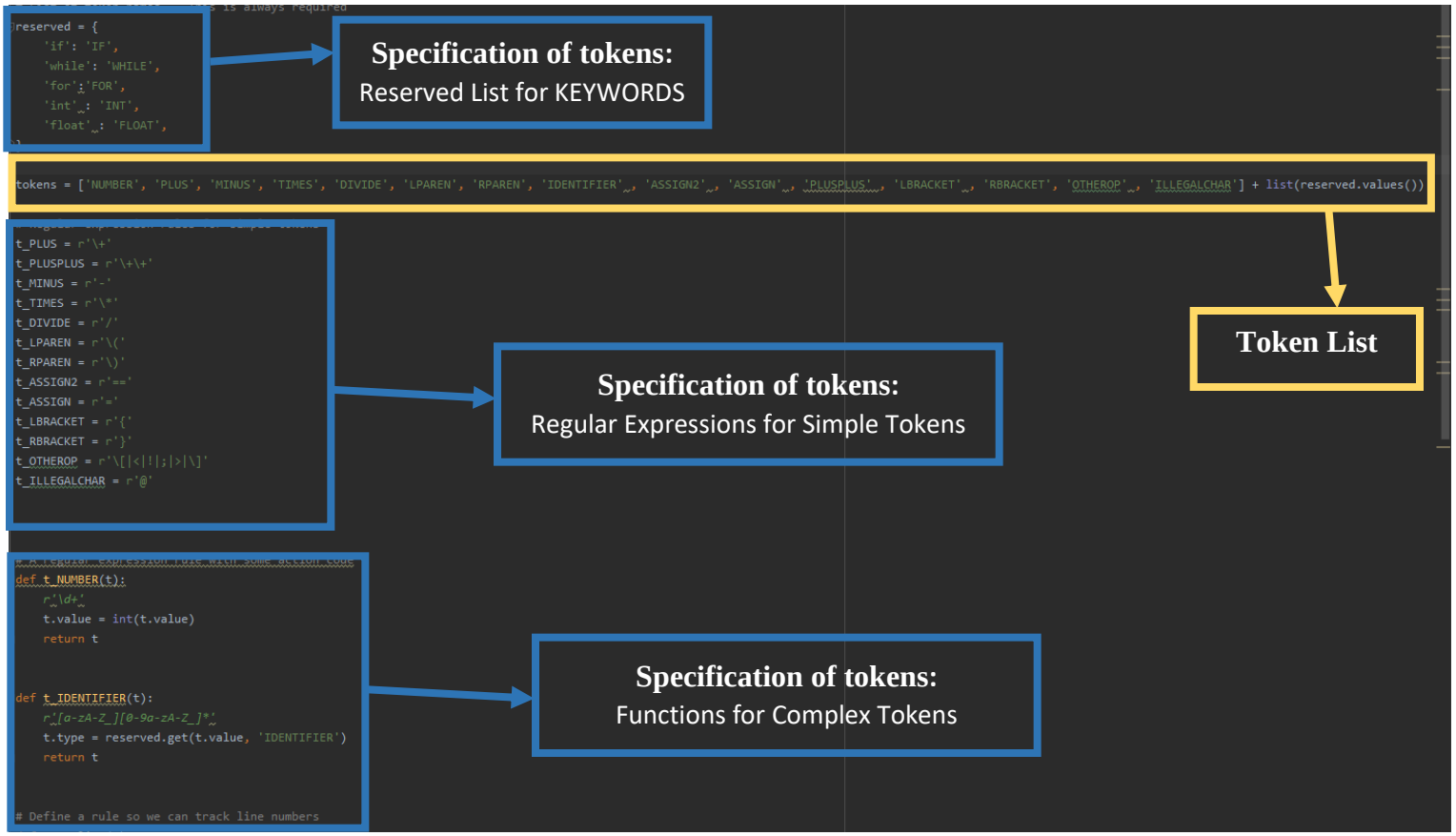
به صورت پیش فرض ابزار lex.py توانایی شمارش خط را ندارد برای این عمل تابعی تحت عنوان t_newline میسازیم که لکسر با خواندن کاراکتر \n توکن newline را شناسایی نموده و به متغیرlineno یک واحد میافزاید. این تابع مقاداری را بازنمیگرداند.

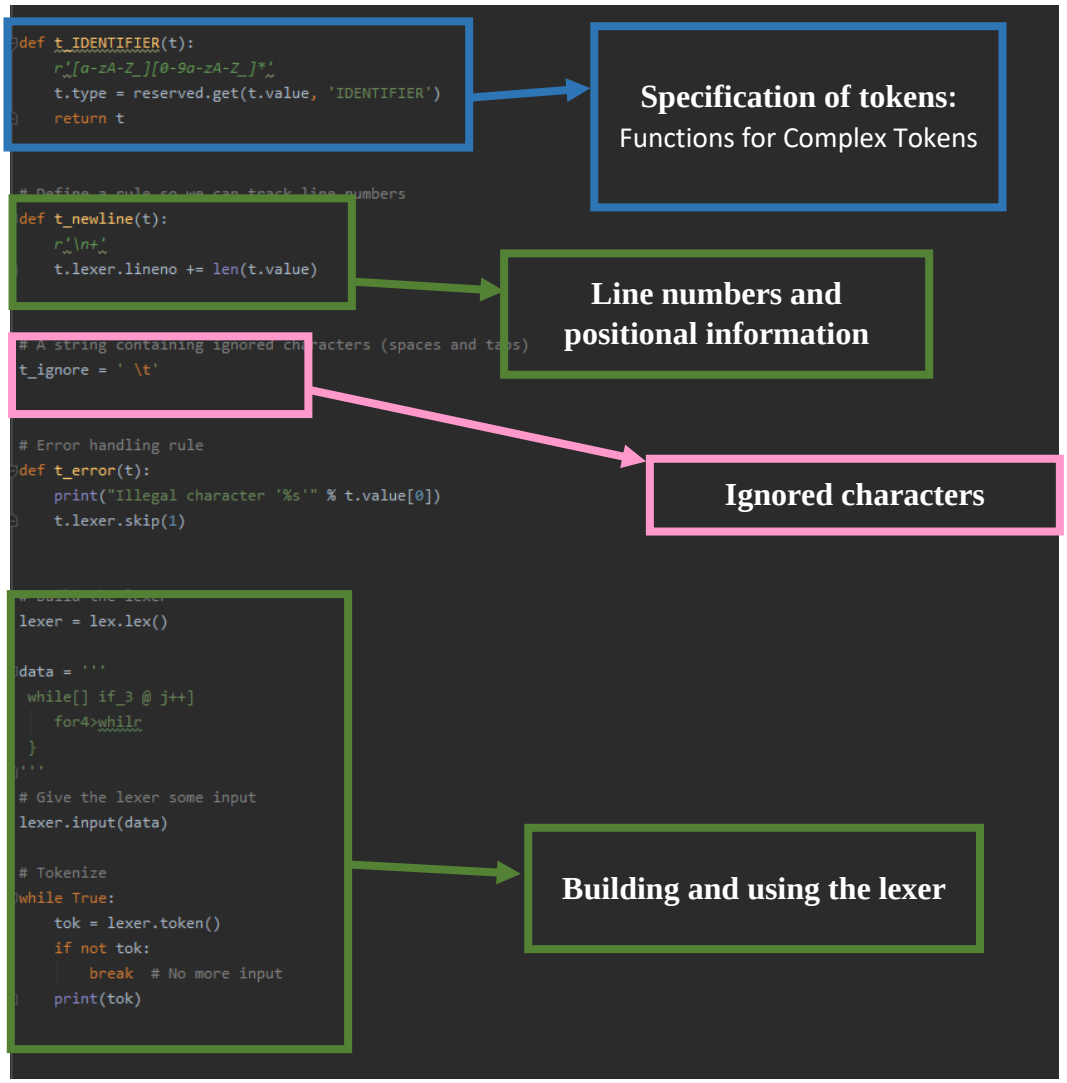
:Ignored characters

قانون t_ignore برای lex.py رزرو شده است تا کاراکتر هایی که باید در برنامه ورودی نادیده گرفته شوند را مورد بررسی قرار ندهد مانند فاصله ها و tabها

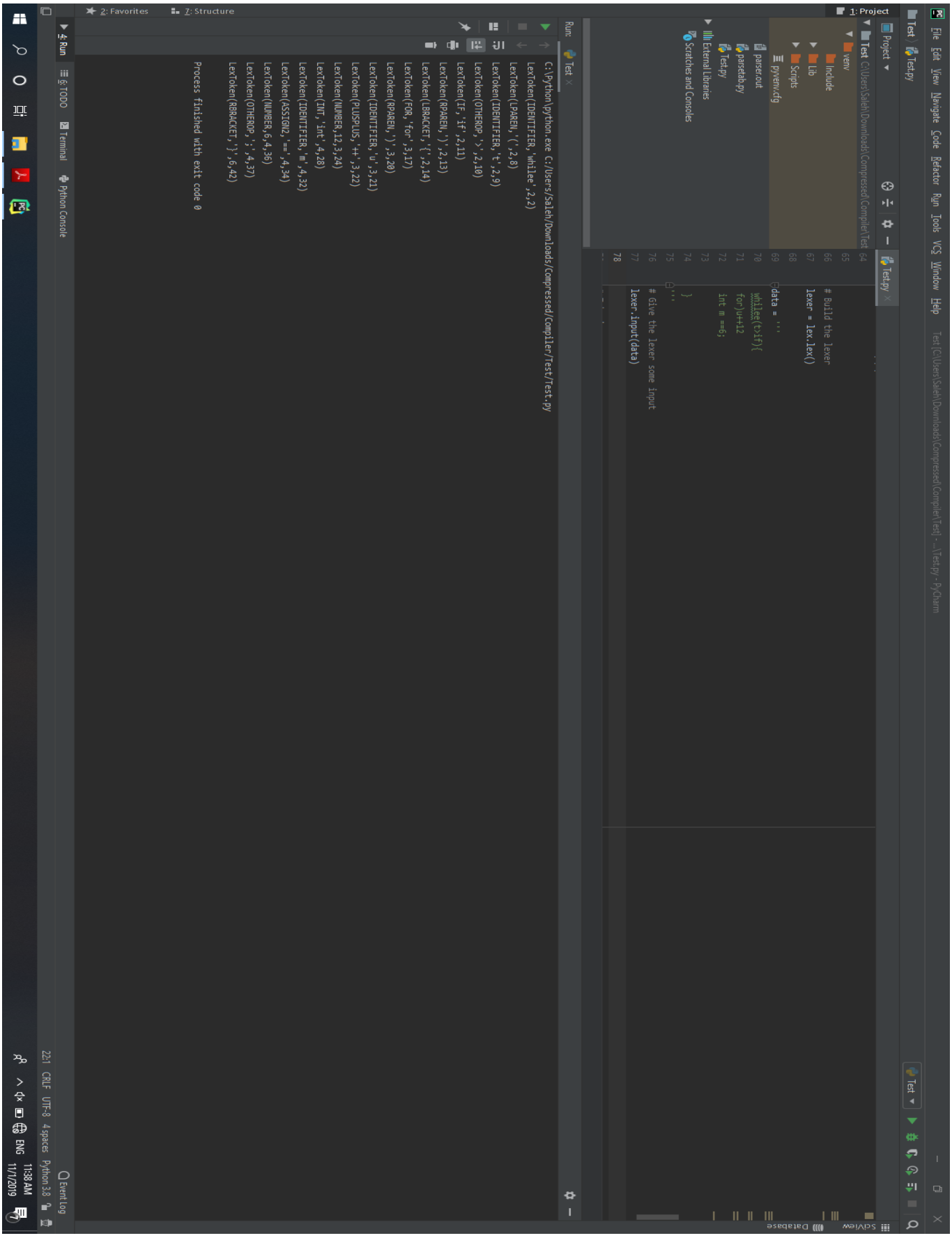
:Building and using the lexer

در نهایت برای استفاده از یک لکسر و مشخص کردن توکن های فایل ورودی ابتدا با استفاده از تابع lex.lex() یک لکسر ایجاد میکنیم مثل: Lexer = lex.lex(). پس از ایجاد لکسر مورد نظر میتوانیم از دو متد lexer.input(data) برای دادن فایل برای مشخص نمودن توکن های آن و همچنین lexer.token() برای بازگرداندن توکن یافت شده از فایل ورودی، استفاده نماییم.

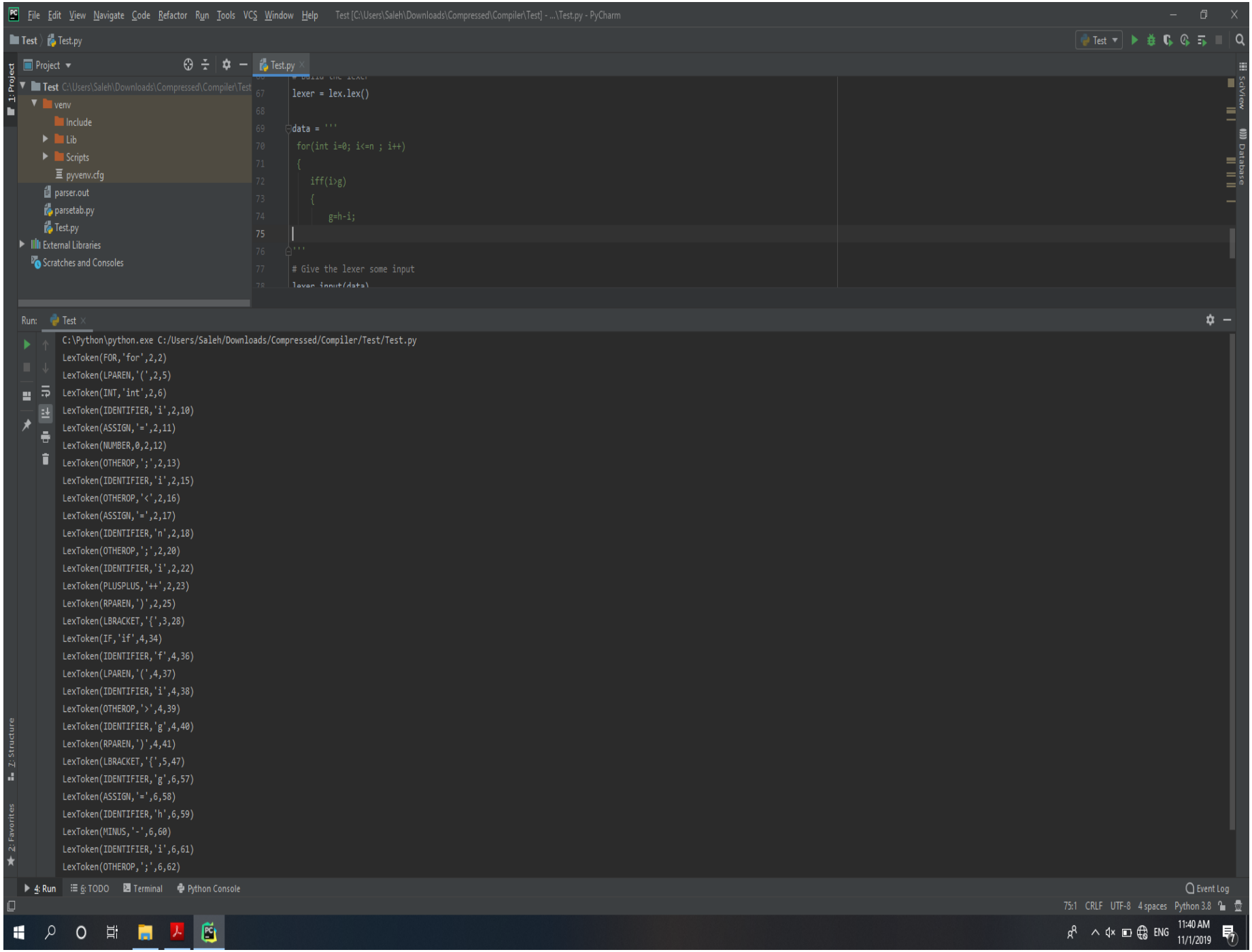

reserved = {
'if': 'IF',
'while': 'WHILE',
'for':'FOR',
'int': 'INT',
'float': 'FLOAT',
Specification of tokens:
Reserved List for KEYWORDS
tokens = ['NUMBER', 'PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'LPAREN', 'RPAREN', 'IDENTIFIER', 'ASSIGN2', 'ASSIGN', 'PLUSPLUS', 'LBRACKET', 'RBRACKET', 'OTHEROP', 'ILLEGALCHAR'] + list(reserved.values())
Token List
t_PLUS = r'\+'
t_PLUSPLUS = r'\+\+'
t_MINUS = r'-'
t_TIMES = r'*'
t_DIVIDE = r'/'
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_ASSIGN2 = r'=='
t_ASSIGN = r'='
t_LBRACKET = r'{'
t_RBRACKET = r'}'
t_OTHEROP = r'\[|<|!|;|>|\]'
t_ILLEGALCHAR = r'@'
Specification of tokens:
Regular Expressions for Simple Tokens
def t_NUMBER(t):
r'\d+'
t.value = int(t.value)
return t
def t_IDENTIFIER(t):
r'[a-zA-Z_][0-9a-zA-Z_]*'
t.type = reserved.get(t.value, 'IDENTIFIER')
return t
Define a rule so we can track line numbers
Specification of tokens:
Functions for Complex Tokens


def t_IDENTIFIER(t):
r'[a-zA-Z_][0-9a-zA-Z_]*'
t.type = reserved.get(t.value, 'IDENTIFIER')
return t
Specification of tokens:
Functions for Complex Tokens
def t_newline(t):
r'\n+'
t.lexer.lineno += len(t.value)
Line numbers and
positional information
t_ignore = ' \t'
# Error handling rule
def t_error(t):
print("Illegal character '%s'" % t.value[0])
t.lexer.skip(1)
Ignored characters
lexer = lex.lex()
data = '''
while[] if_3 @ j++]
for4>whilr
}
'''
# Give the lexer some input
lexer.input(data)
# Tokenize
while True:
tok = lexer.token()
if not tok:
break # No more input
print(tok)
Building and using the lexer

برنامه های تست شده به همراه خروجی آن ها:



برنامه های تست شده به همراه خروجی آن ها:



برنامه های تست شده به همراه خروجی آن ها: