AFLL ASSIGNMENT 2

Name: Maryam Khan	**SRN: PES2UG21CS283**	**Sem: III**	**Sec: E**
Name: Manasvi Varma	**SRN: PES2UG21CS305**	**Sem: III**	**Sec: E**

Program to evaluate an arithmetic expression and check if its syntax is valid or not

Context Free Grammar:

G={V,T,P,S}
S(start state)=evaluation
V={evaluation, expression, value, number, unsigned, digit, sign, operator}
T={0,1,2,3,4,5,6,7,8,9,+,-,*,/,=,.}

P consists of the following productions:
evaluation -> expression
expression -> value | value operator expression
value -> number | sign number
number -> unsigned | unsigned . unsigned
unsigned -> digit | digit unsigned
digit -> 0| 1| 2| 3| 4| 5| 6| 7| 8| 9
sign -> + | -
operator -> +| -| *| /|(|)

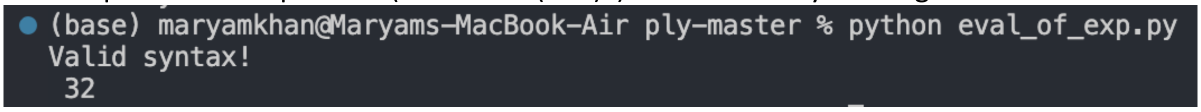
```
from ply import lex
import ply.yacc as yacc

tokens = (
   'PLUS',
   'MINUS',
   'TIMES',
   'DIV',
   'LPAREN',
   'RPAREN',
   'NUMBER',
)

t_ignore = ' \t'
t_PLUS    = r'\+'
t_MINUS  = r'-'
t_TIMES = r'\*'
t_DIV     = r'/'
t_LPAREN = r'\('
t_RPAREN  = r'\)'
```

```
def t_NUMBER( t ) :
    r'[0-9]+'
    t.value = int( t.value )
    return t

def t_newline( t ):
  r'\n+'
  t.lexer.lineno += len( t.value )

def t_error( t ):
  print("Invalid Token:",t.value[0])
  t.lexer.skip( 1 )

lexer = lex.lex()

precedence = (
    ( 'left', 'PLUS', 'MINUS' ),
    ( 'left', 'TIMES', 'DIV' ),
    ( 'nonassoc', 'UMINUS' )
)

def p_add( p ) :
    'expr : expr PLUS expr'
    p[0] = p[1] + p[3]

def p_sub( p ) :
    'expr : expr MINUS expr'
    p[0] = p[1] - p[3]

def p_expr2uminus( p ) :
    'expr : MINUS expr %prec UMINUS'
    p[0] = - p[2]

def p_mult_div( p ) :
    '''expr : expr TIMES expr
            | expr DIV expr'''

    if p[2] == '*' :
        p[0] = p[1] * p[3]
    else :
        if p[3] == 0 :
            print("Can't divide by 0")
            raise ZeroDivisionError('integer division by 0')
        p[0] = p[1] / p[3]
```

```python
def p_expr2NUM( p ) :
   'expr : NUMBER'
   p[0] = p[1]

def p_parens( p ) :
   'expr : LPAREN expr RPAREN'
   p[0] = p[2]

def p_error( p ):
   print("Syntax error in input!")

parser = yacc.yacc()

res = parser.parse("2*3-1+7+(4*5)") # the input
print("Valid syntax!\n",res)
```

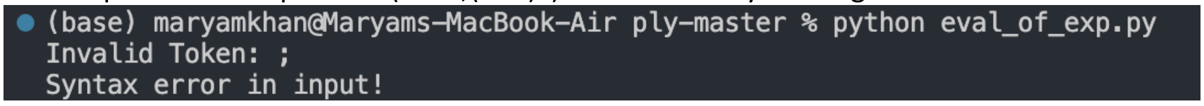
Output:

The output for the expression("2*3-1+7+(4*5)") with correct syntax is given below



```
● (base) maryamkhan@Maryams-MacBook-Air ply-master % python eval_of_exp.py
  Valid syntax!
   32
```

The output for the expression("4+5;(2*3)") with correct syntax is given below



```
● (base) maryamkhan@Maryams-MacBook-Air ply-master % python eval_of_exp.py
  Invalid Token: ;
  Syntax error in input!
```

Program to declare a variable in javaScript

Context Free Grammar:

G={V,T,P,S}
S(start state)=declaration
V={declaration, variable_list, num, exp, symbol, type, NAME, EQUAL}
T={ [[a-zA-Z_0-9],+,-,*,/,=,var,const,let}

declaration→ type variable_list
variable_list→ NAME EQUAL num| NAME EQUAL NAME| NAME EQUAL exp| NAME EQUAL STRING| NAME
variable_list→ variable_list ',' NAME
num→ INT| FLOAT
exp→ exp symbol exp| num| NAME
symbol→ +| -| *| /
type→ var| const| let
NAME→[a-zA-Z_0-9]
EQUAL→=

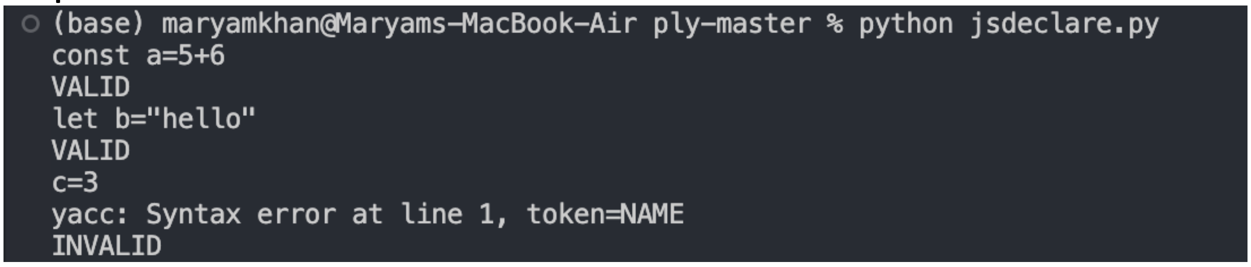
```
#lexer file
import ply.lex as lex
import ply.yacc as yacc

reserved={
    'var':'VAR',
    'const':'CONST',
    'let':'LET'
    }

tokens=[
    'NAME',
    'EQUAL',
    'PLUS',
    'MINUS',
    'MUL',
    'DIV',
    'INT',
    'FLOAT',
    'STRING'
    ] +list(reserved.values())
```

```python
literals=',\"\"'
#const a=7+9
#const a=b+c;
#const a=b+3;
#const a;

t_ignore='; \t'

def t_NAME(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    #print("1",t.value,t.type)
    t.type=reserved.get(t.value,'NAME') #checks for reserved words
    return t

def t_EQUAL(t):
    r'\='
    t.type='EQUAL'
    return t

def t_PLUS(t):
    r'\+'
    t.type='PLUS'
    return t

def t_MINUS(t):
    r'\-'
    t.type='MINUS'
    return t

def t_MUL(t):
    r'\*'
    t.type='MUL'
    return t

def t_DIV(t):
    r'\/'
    t.type='DIV'
    return t

def t_FLOAT(t):
    r'\d+\.\d+'
    t.type='FLOAT'
    return t

def t_INT(t):
    r'\d+'
    t.type='INT'
    return t

def t_STRING(t):
    r'\"([^\\\n]|(\\.))*?\"'
    #print("1",t.value,t.type)
    t.type=reserved.get(t.value,'STRING') #checks for reserved words
    return t


def t_error(t):
    print("Not allowed")
    t.lexer.skip(1)



def p_declaration(p):
    ''' declaration : type variable_list '''
    p[0] = ['DECLARE',p[1], p[2]]

def p_one_variable(p):
    ''' variable_list : NAME EQUAL num
                   | NAME EQUAL NAME
                   | NAME EQUAL exp
                   | NAME EQUAL STRING
                   | NAME '''
    p[0] = [ p[1] ]
```

```
def p_more_variables(p):
    ''' variable_list : variable_list ',' NAME '''
    p[0] = p[1]
    p[0].append(p[3])

def p_num(p):
    ''' num : INT
            | FLOAT '''
    p[0]=p[1]

def p_exp(p):
    ''' exp : exp symbol exp
            | num
            | NAME '''
    p[0]=[p[1]]

def p_symbol(p):
    ''' symbol : PLUS
               | MINUS
               | MUL
               | DIV '''
    p[0]=p[1]



def p_type(p):
    ''' type : VAR
             | CONST
             | LET '''
    p[0]=p[1]

parser=yacc.yacc()
lexer=lex.lex()


while True:
    try:
        s=input()
```

```python
    #print(s)
except EOFError:
    break
result=parser.parse(s)
#print(result)
if(result!=None):
    print("VALID")
else:
    print("INVALID")
    #break
```

Output:



```
○ (base) maryamkhan@Maryams-MacBook-Air ply-master % python jsdeclare.py
  const a=5+6
  VALID
  let b="hello"
  VALID
  c=3
  yacc: Syntax error at line 1, token=NAME
  INVALID
```