Automata Formal Languages and Logic Assignment

UE22CS243A

Syntax Validator of R Programming Language

3rd Semester, Academic Year 2023

Name: Viona Sequeira

SRN: PES2UG22CS665

Name: Vijayalaxmi S.H

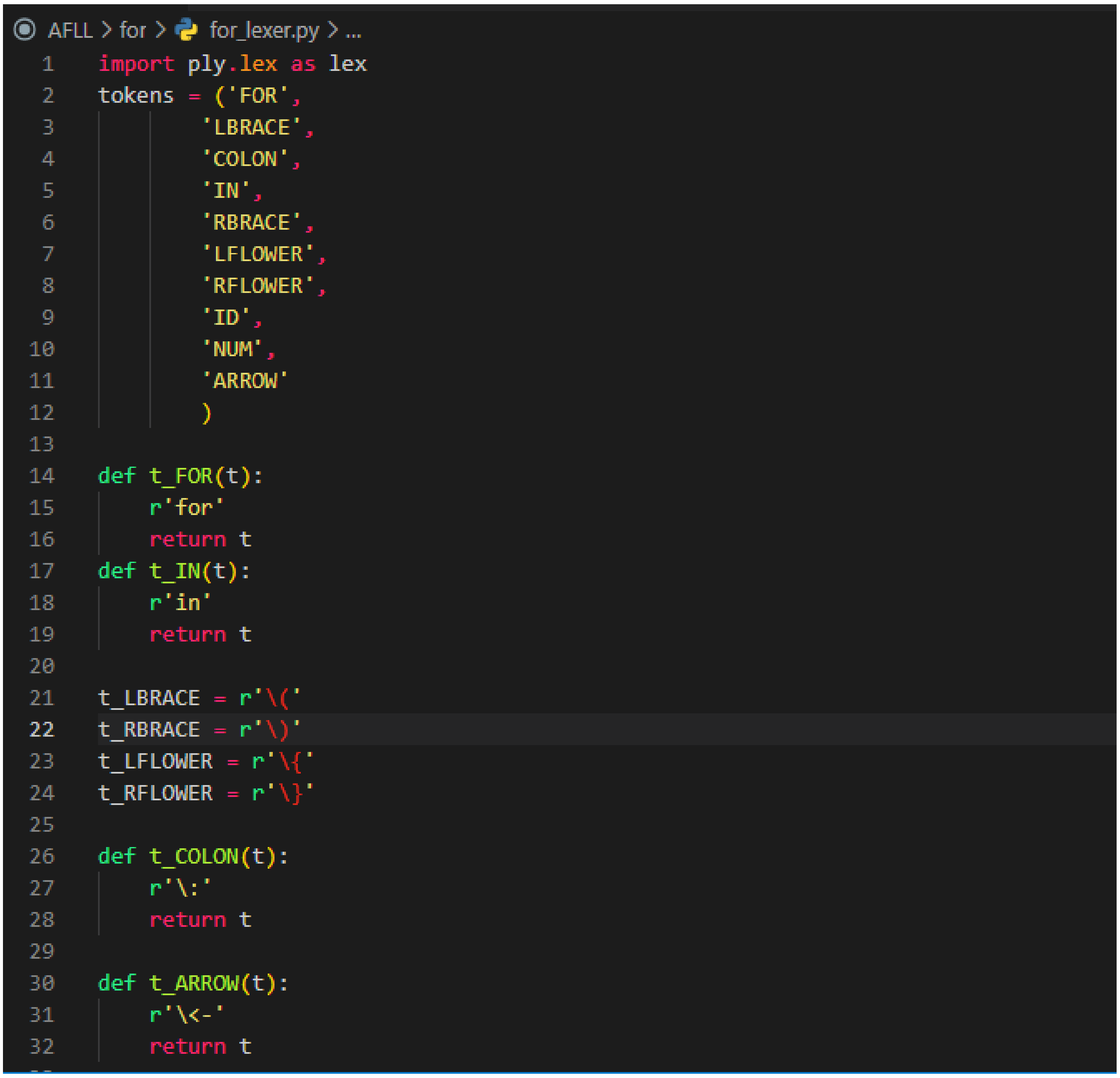
SRN: PES2UG22CS663

SECTION: K

Constructs

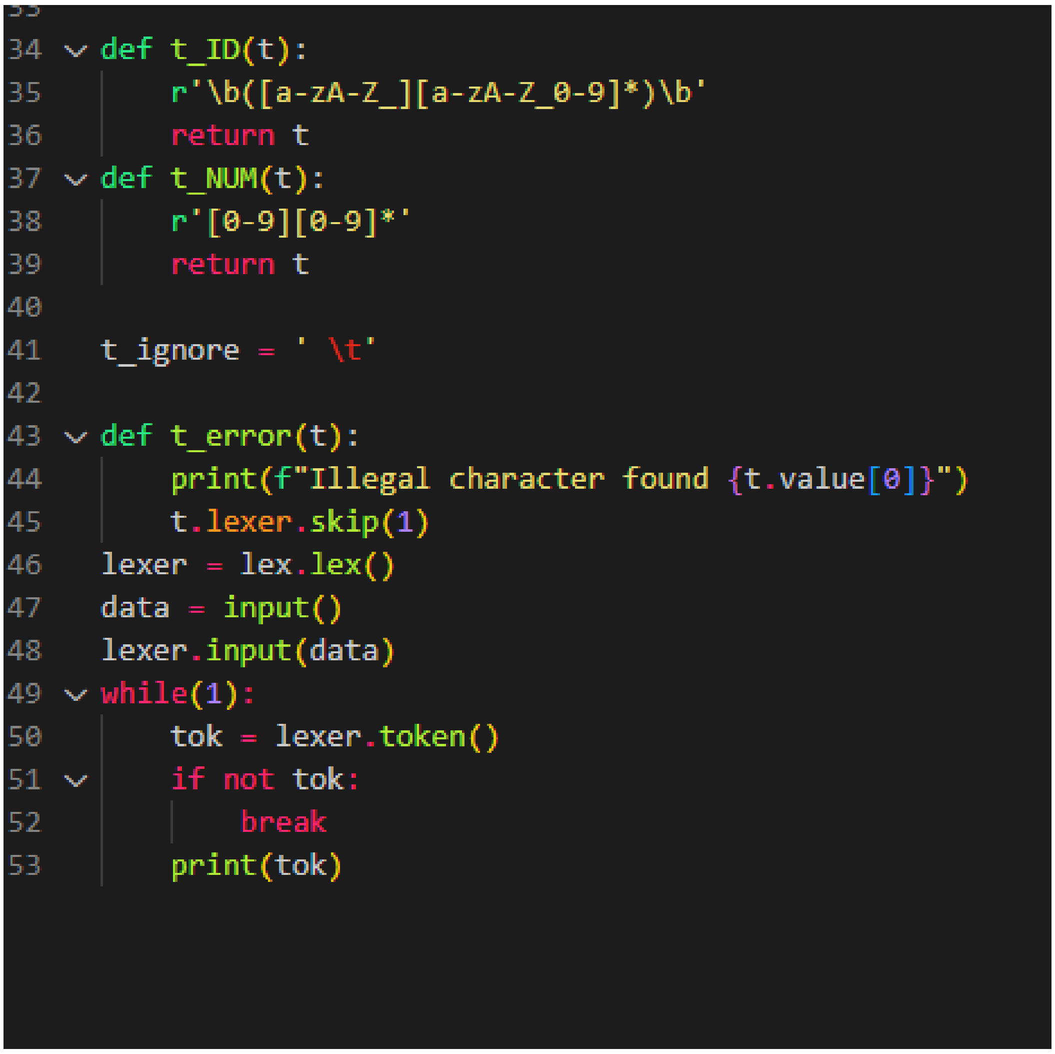
1) For loop

Lex File



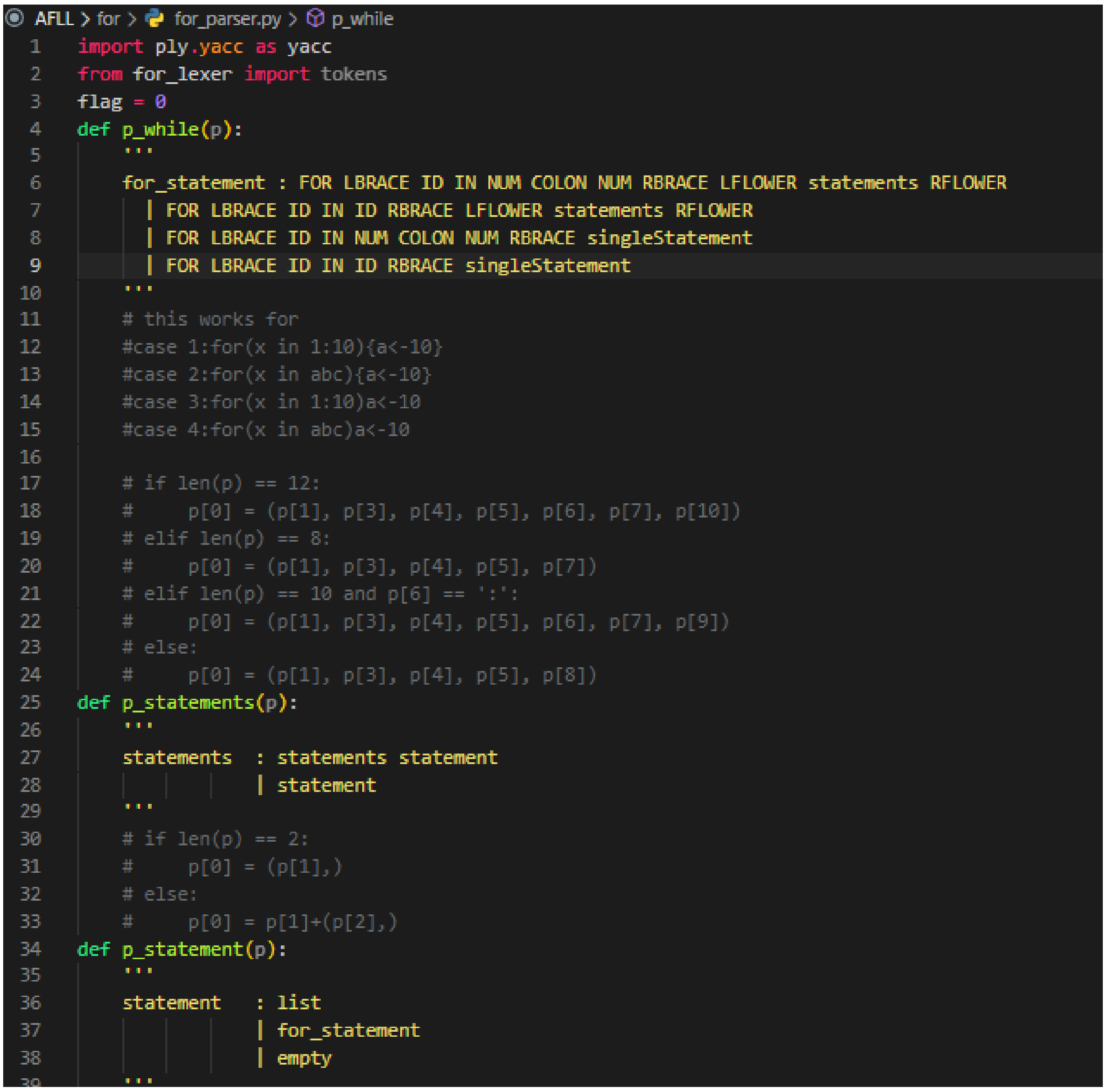
AFLL > for > for_lexer.py > ...

```python
import ply.lex as lex
tokens = ('FOR',
          'LBRACE',
          'COLON',
          'IN',
          'RBRACE',
          'LFLOWER',
          'RFLOWER',
          'ID',
          'NUM',
          'ARROW'
          )

def t_FOR(t):
    r'for'
    return t
def t_IN(t):
    r'in'
    return t

t_LBRACE = r'\('
t_RBRACE = r'\)'
t_LFLOWER = r'\{'
t_RFLOWER = r'\}'

def t_COLON(t):
    r'\:'
    return t

def t_ARROW(t):
    r'\<-'
    return t
```

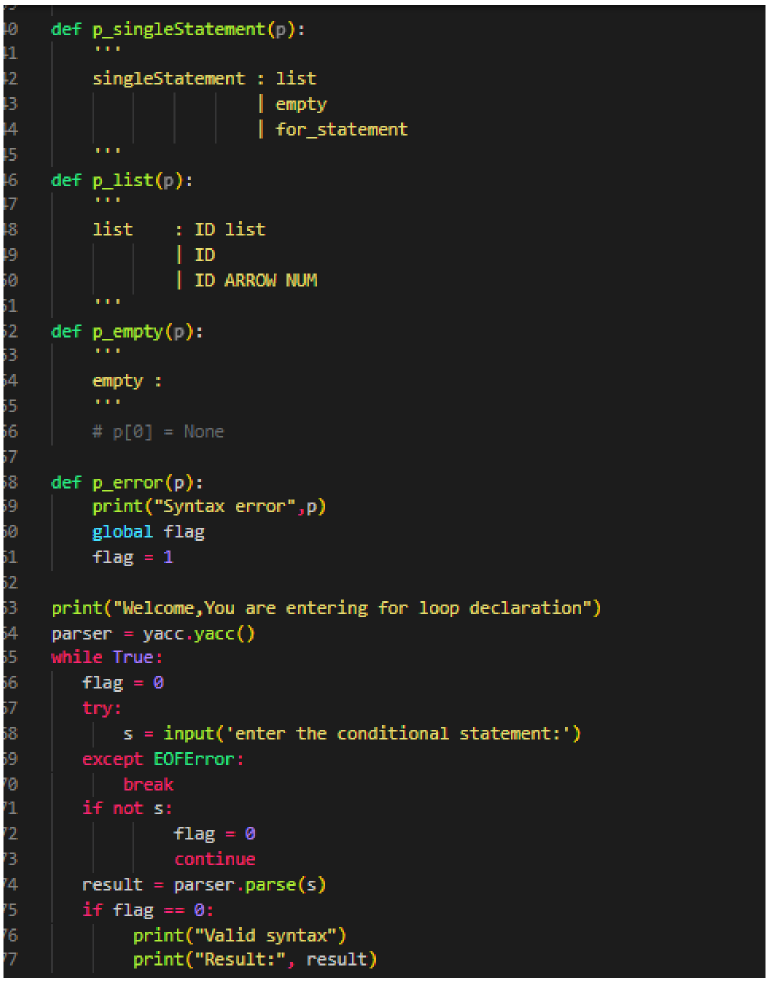


```python
def t_ID(t):
    r'\b([a-zA-Z_][a-zA-Z_0-9]*)\b'
    return t
def t_NUM(t):
    r'[0-9][0-9]*'
    return t

t_ignore = ' \t'

def t_error(t):
    print(f"Illegal character found {t.value[0]}")
    t.lexer.skip(1)
lexer = lex.lex()
data = input()
lexer.input(data)
while(1):
    tok = lexer.token()
    if not tok:
        break
    print(tok)
```

Yacc file

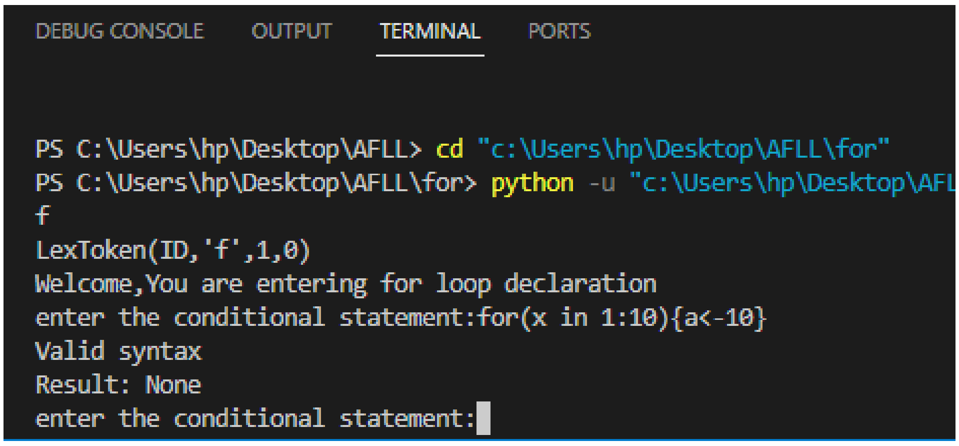


```
import ply.yacc as yacc
from for_lexer import tokens
flag = 0
def p_while(p):
    '''
    for_statement : FOR LBRACE ID IN NUM COLON NUM RBRACE LFLOWER statements RFLOWER
      | FOR LBRACE ID IN ID RBRACE LFLOWER statements RFLOWER
      | FOR LBRACE ID IN NUM COLON NUM RBRACE singleStatement
      | FOR LBRACE ID IN ID RBRACE singleStatement
    '''
    # this works for
    #case 1:for(x in 1:10){a<-10}
    #case 2:for(x in abc){a<-10}
    #case 3:for(x in 1:10)a<-10
    #case 4:for(x in abc)a<-10

    # if len(p) == 12:
    #     p[0] = (p[1], p[3], p[4], p[5], p[6], p[7], p[10])
    # elif len(p) == 8:
    #     p[0] = (p[1], p[3], p[4], p[5], p[7])
    # elif len(p) == 10 and p[6] == ':':
    #     p[0] = (p[1], p[3], p[4], p[5], p[6], p[7], p[9])
    # else:
    #     p[0] = (p[1], p[3], p[4], p[5], p[8])
def p_statements(p):
    '''
    statements  : statements statement
                | statement
    '''
    # if len(p) == 2:
    #     p[0] = (p[1],)
    # else:
    #     p[0] = p[1]+(p[2],)
def p_statement(p):
    '''
    statement   : list
                | for_statement
                | empty
    '''
```



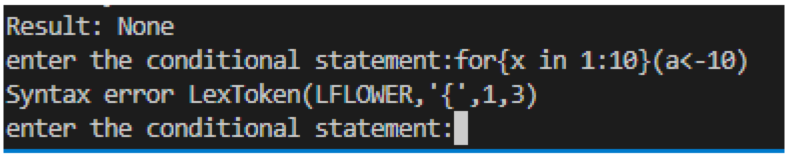
```
def p_singleStatement(p):
    '''
    singleStatement : list
                    | empty
                    | for_statement
    '''
def p_list(p):
    '''
    list    : ID list
            | ID
            | ID ARROW NUM
    '''
def p_empty(p):
    '''
    empty :
    '''
    # p[0] = None

def p_error(p):
    print("Syntax error",p)
    global flag
    flag = 1

print("Welcome,You are entering for loop declaration")
parser = yacc.yacc()
while True:
   flag = 0
   try:
       s = input('enter the conditional statement:')
   except EOFError:
       break
   if not s:
           flag = 0
           continue
   result = parser.parse(s)
   if flag == 0:
        print("Valid syntax")
        print("Result:", result)
```

Valid Test Case



```
PS C:\Users\hp\Desktop\AFLL> cd "c:\Users\hp\Desktop\AFLL\for"
PS C:\Users\hp\Desktop\AFLL\for> python -u "c:\Users\hp\Desktop\AFL
f
LexToken(ID,'f',1,0)
Welcome,You are entering for loop declaration
enter the conditional statement:for(x in 1:10){a<-10}
Valid syntax
Result: None
enter the conditional statement:
```

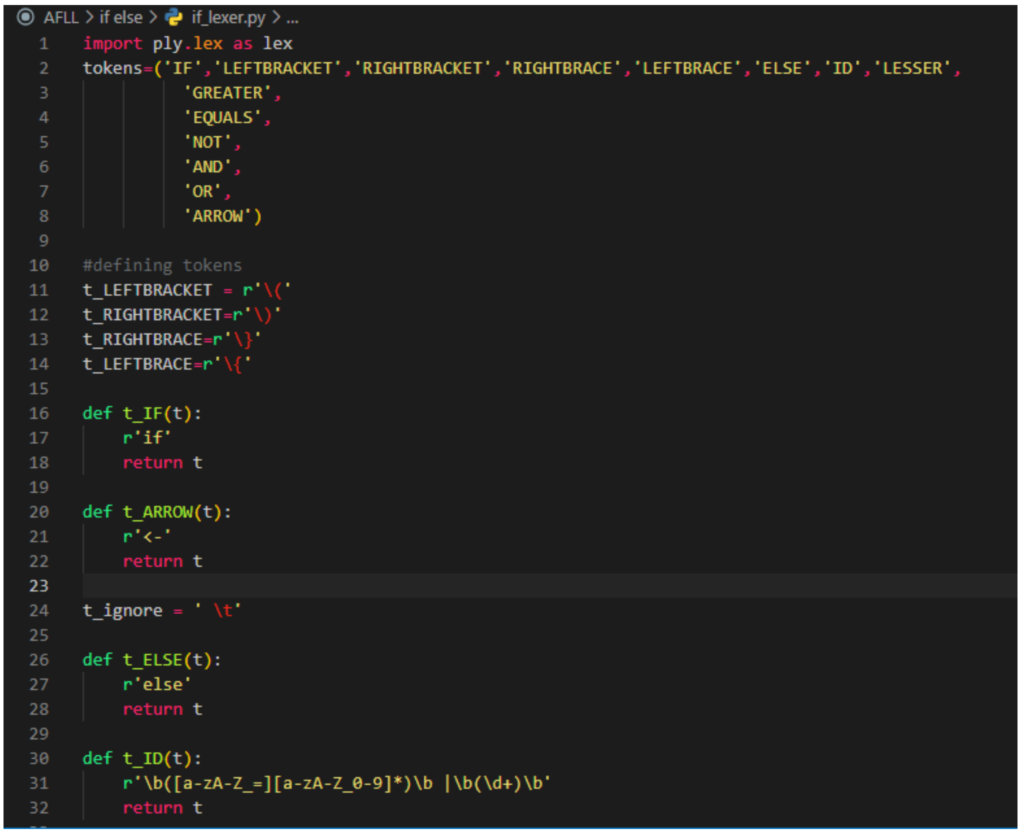
Invalid Test Case



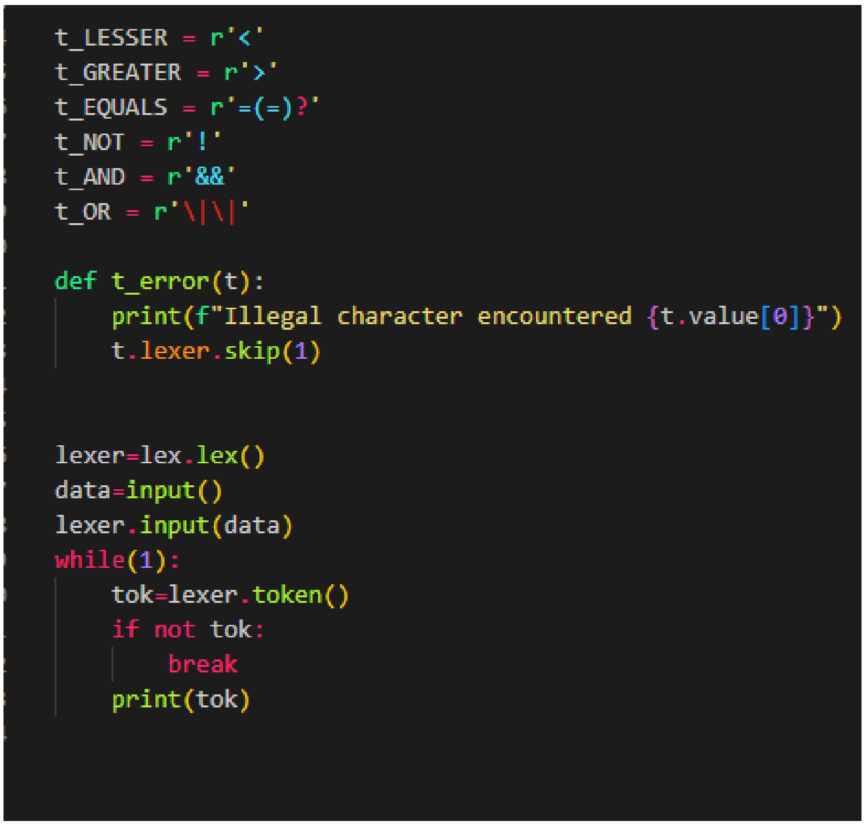
```
Result: None
enter the conditional statement:for{x in 1:10}(a<-10)
Syntax error LexToken(LFLOWER,'{',1,3)
enter the conditional statement:
```

2)If else

Lex file

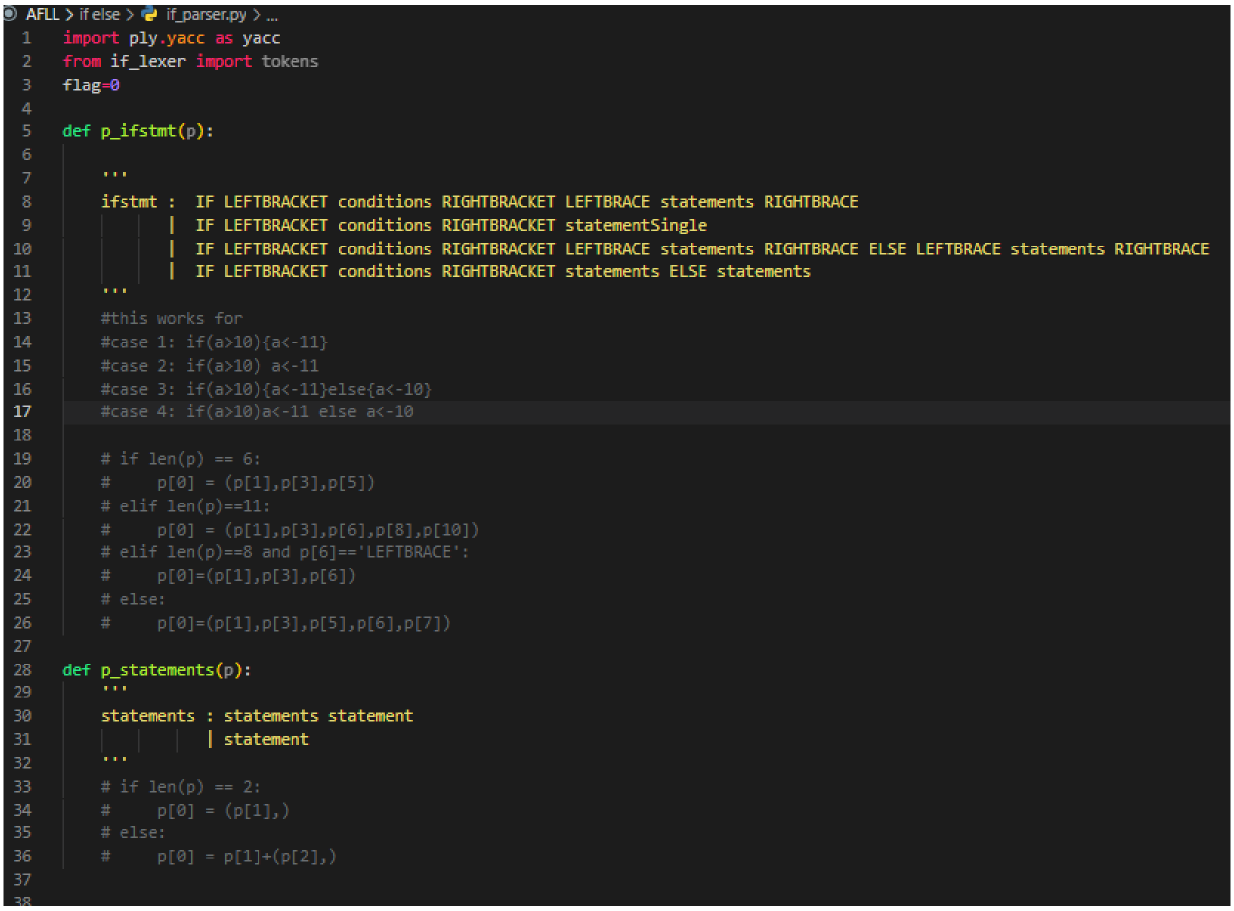


```python
import ply.lex as lex
tokens=('IF','LEFTBRACKET','RIGHTBRACKET','RIGHTBRACE','LEFTBRACE','ELSE','ID','LESSER',
          'GREATER',
          'EQUALS',
          'NOT',
          'AND',
          'OR',
          'ARROW')

#defining tokens
t_LEFTBRACKET = r'\('
t_RIGHTBRACKET=r'\)'
t_RIGHTBRACE=r'\}'
t_LEFTBRACE=r'\{'

def t_IF(t):
    r'if'
    return t

def t_ARROW(t):
    r'<-'
    return t

t_ignore = ' \t'

def t_ELSE(t):
    r'else'
    return t

def t_ID(t):
    r'\b([a-zA-Z_=][a-zA-Z_0-9]*)\b |\b(\d+)\b'
    return t
```

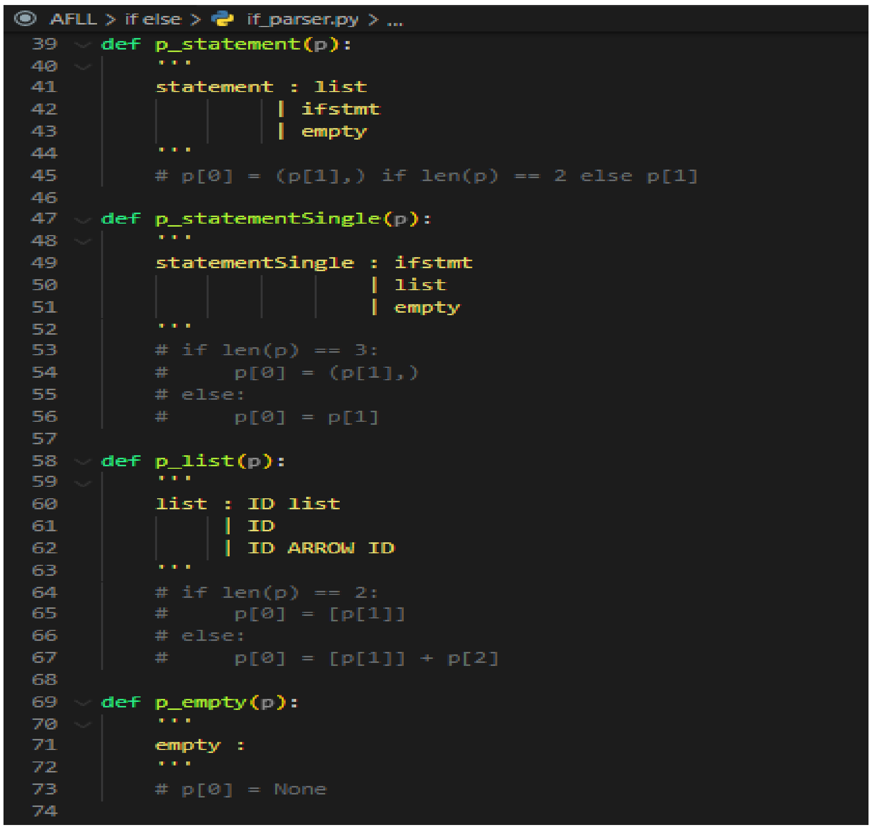


```python
t_LESSER = r'<'
t_GREATER = r'>'
t_EQUALS = r'=(=)?'
t_NOT = r'!'
t_AND = r'&&'
t_OR = r'\|\|'

def t_error(t):
    print(f"Illegal character encountered {t.value[0]}")
    t.lexer.skip(1)


lexer=lex.lex()
data=input()
lexer.input(data)
while(1):
    tok=lexer.token()
    if not tok:
        break
    print(tok)
```

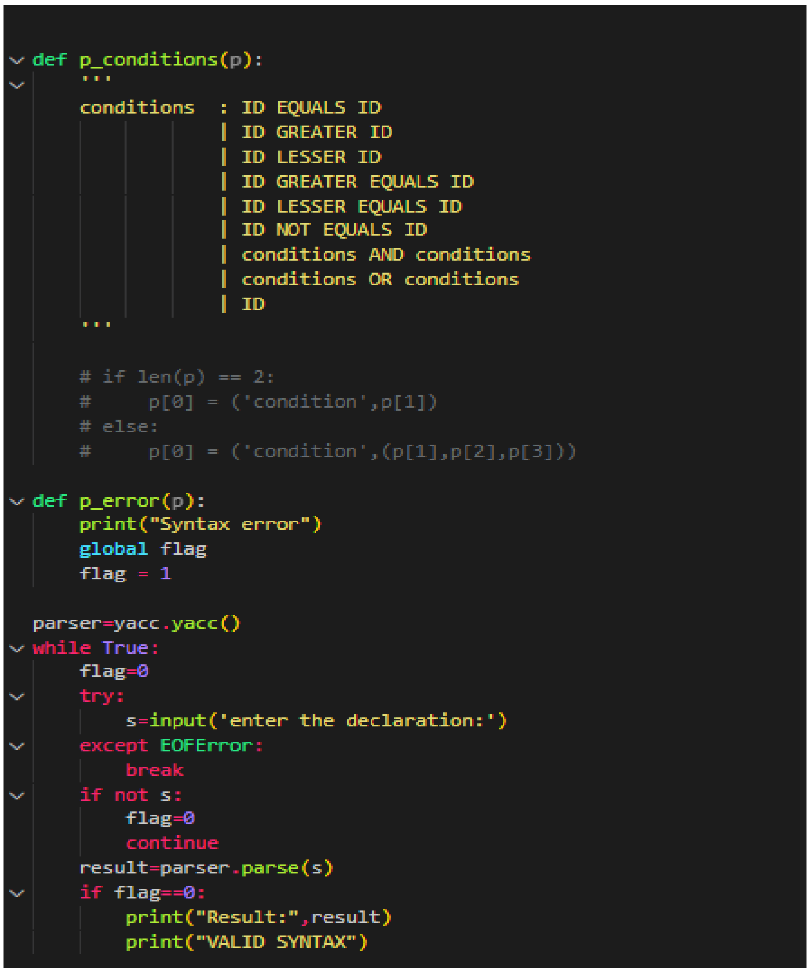
Yacc file



```
import ply.yacc as yacc
from if_lexer import tokens
flag=0

def p_ifstmt(p):

    '''
    ifstmt :  IF LEFTBRACKET conditions RIGHTBRACKET LEFTBRACE statements RIGHTBRACE
           |  IF LEFTBRACKET conditions RIGHTBRACKET statementSingle
           |  IF LEFTBRACKET conditions RIGHTBRACKET LEFTBRACE statements RIGHTBRACE ELSE LEFTBRACE statements RIGHTBRACE
           |  IF LEFTBRACKET conditions RIGHTBRACKET statements ELSE statements
    '''
    #this works for
    #case 1: if(a>10){a<-11}
    #case 2: if(a>10) a<-11
    #case 3: if(a>10){a<-11}else{a<-10}
    #case 4: if(a>10)a<-11 else a<-10

    # if len(p) == 6:
    #     p[0] = (p[1],p[3],p[5])
    # elif len(p)==11:
    #     p[0] = (p[1],p[3],p[6],p[8],p[10])
    # elif len(p)==8 and p[6]=='LEFTBRACE':
    #     p[0]=(p[1],p[3],p[6])
    # else:
    #     p[0]=(p[1],p[3],p[5],p[6],p[7])

def p_statements(p):
    '''
    statements : statements statement
               | statement
    '''
    # if len(p) == 2:
    #     p[0] = (p[1],)
    # else:
    #     p[0] = p[1]+(p[2],)
```

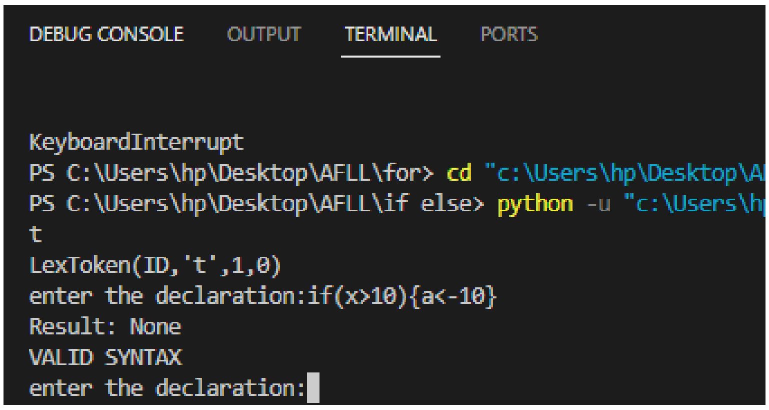


```
def p_statement(p):
    '''
    statement : list
              | ifstmt
              | empty
    '''
    # p[0] = (p[1],) if len(p) == 2 else p[1]

def p_statementSingle(p):
    '''
    statementSingle : ifstmt
                    | list
                    | empty
    '''
    # if len(p) == 3:
    #     p[0] = (p[1],)
    # else:
    #     p[0] = p[1]

def p_list(p):
    '''
    list : ID list
         | ID
         | ID ARROW ID
    '''
    # if len(p) == 2:
    #     p[0] = [p[1]]
    # else:
    #     p[0] = [p[1]] + p[2]

def p_empty(p):
    '''
    empty :
    '''
    # p[0] = None
```



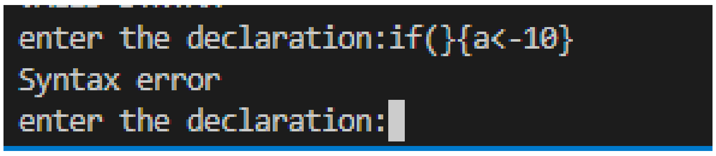
```python
def p_conditions(p):
    '''
    conditions  : ID EQUALS ID
                | ID GREATER ID
                | ID LESSER ID
                | ID GREATER EQUALS ID
                | ID LESSER EQUALS ID
                | ID NOT EQUALS ID
                | conditions AND conditions
                | conditions OR conditions
                | ID
    '''

    # if len(p) == 2:
    #     p[0] = ('condition',p[1])
    # else:
    #     p[0] = ('condition',(p[1],p[2],p[3]))

def p_error(p):
    print("Syntax error")
    global flag
    flag = 1

parser=yacc.yacc()
while True:
    flag=0
    try:
        s=input('enter the declaration:')
    except EOFError:
        break
    if not s:
        flag=0
        continue
    result=parser.parse(s)
    if flag==0:
        print("Result:",result)
        print("VALID SYNTAX")
```

Valid Test Case



```
KeyboardInterrupt
PS C:\Users\hp\Desktop\AFLL\for> cd "c:\Users\hp\Desktop\A
PS C:\Users\hp\Desktop\AFLL\if else> python -u "c:\Users\h
t
LexToken(ID,'t',1,0)
enter the declaration:if(x>10){a<-10}
Result: None
VALID SYNTAX
enter the declaration:
```

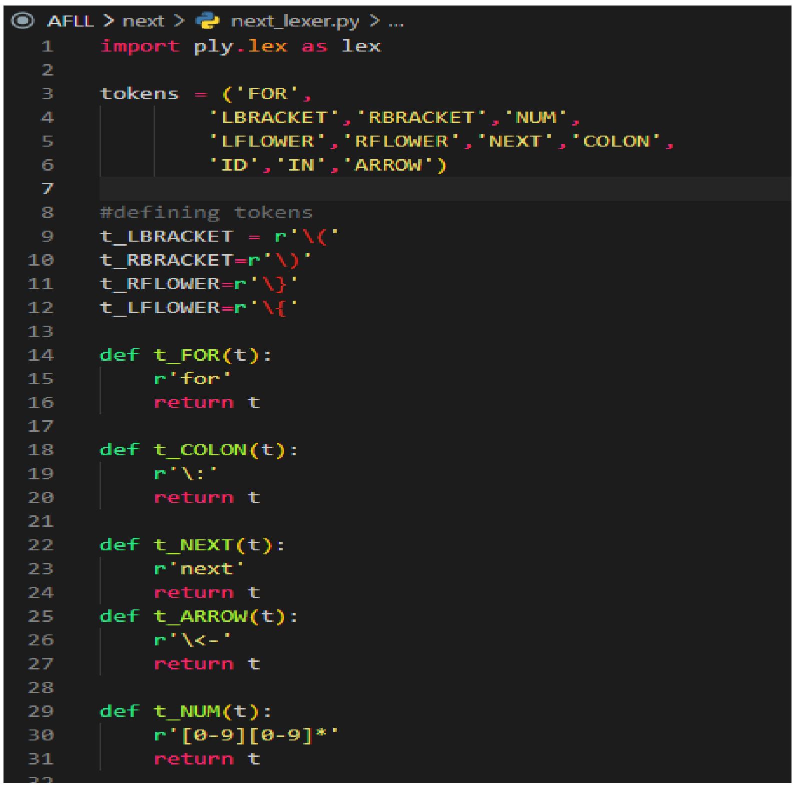
Invalid Test Case



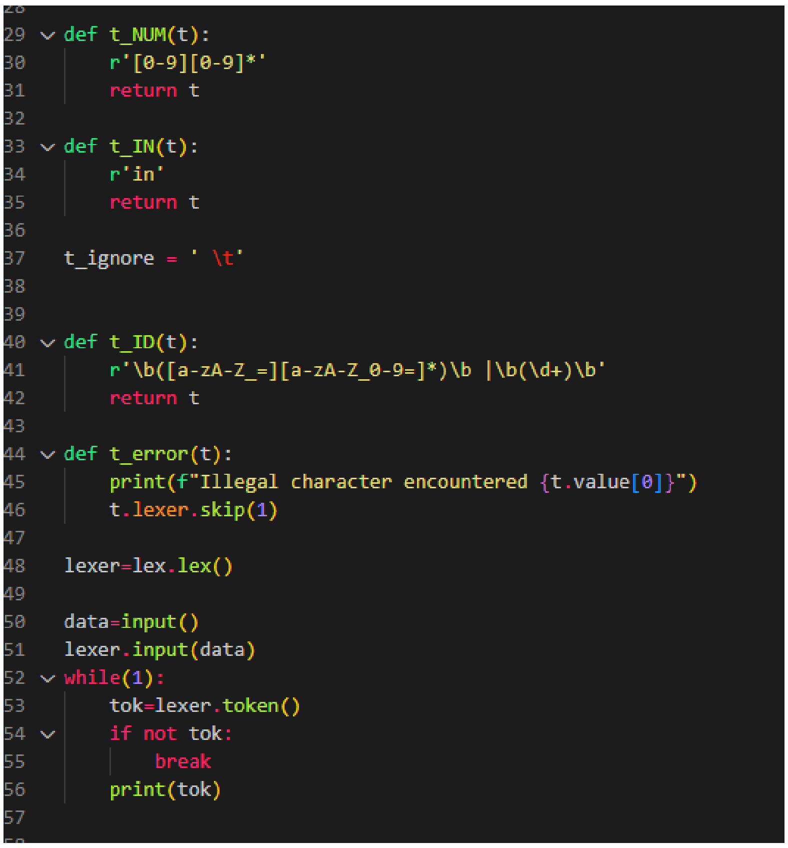
```
enter the declaration:if(){a<-10}
Syntax error
enter the declaration:
```

3) Next

Lex file

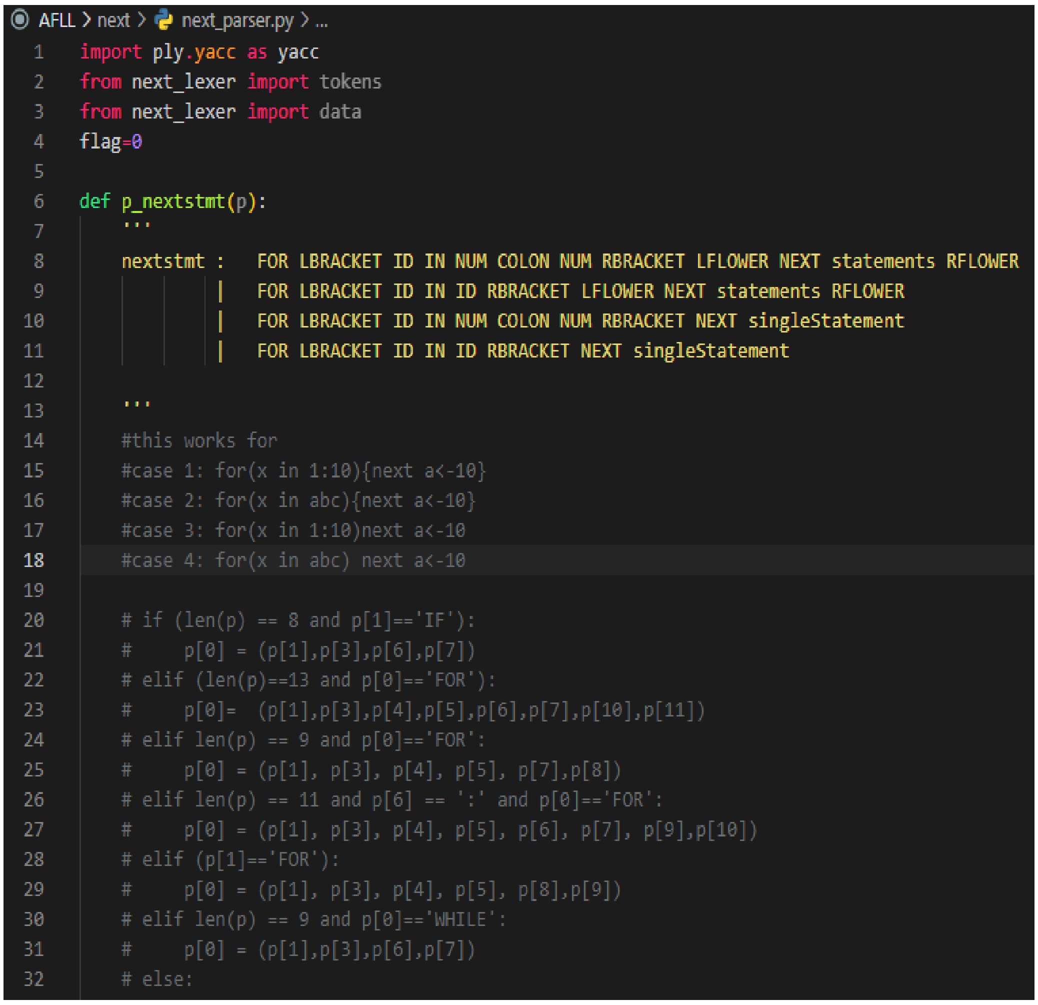


```python
import ply.lex as lex

tokens = ('FOR',
          'LBRACKET','RBRACKET','NUM',
          'LFLOWER','RFLOWER','NEXT','COLON',
          'ID','IN','ARROW')

#defining tokens
t_LBRACKET = r'\('
t_RBRACKET=r'\)'
t_RFLOWER=r'\}'
t_LFLOWER=r'\{'

def t_FOR(t):
    r'for'
    return t

def t_COLON(t):
    r'\:'
    return t

def t_NEXT(t):
    r'next'
    return t
def t_ARROW(t):
    r'\<-'
    return t

def t_NUM(t):
    r'[0-9][0-9]*'
    return t
```

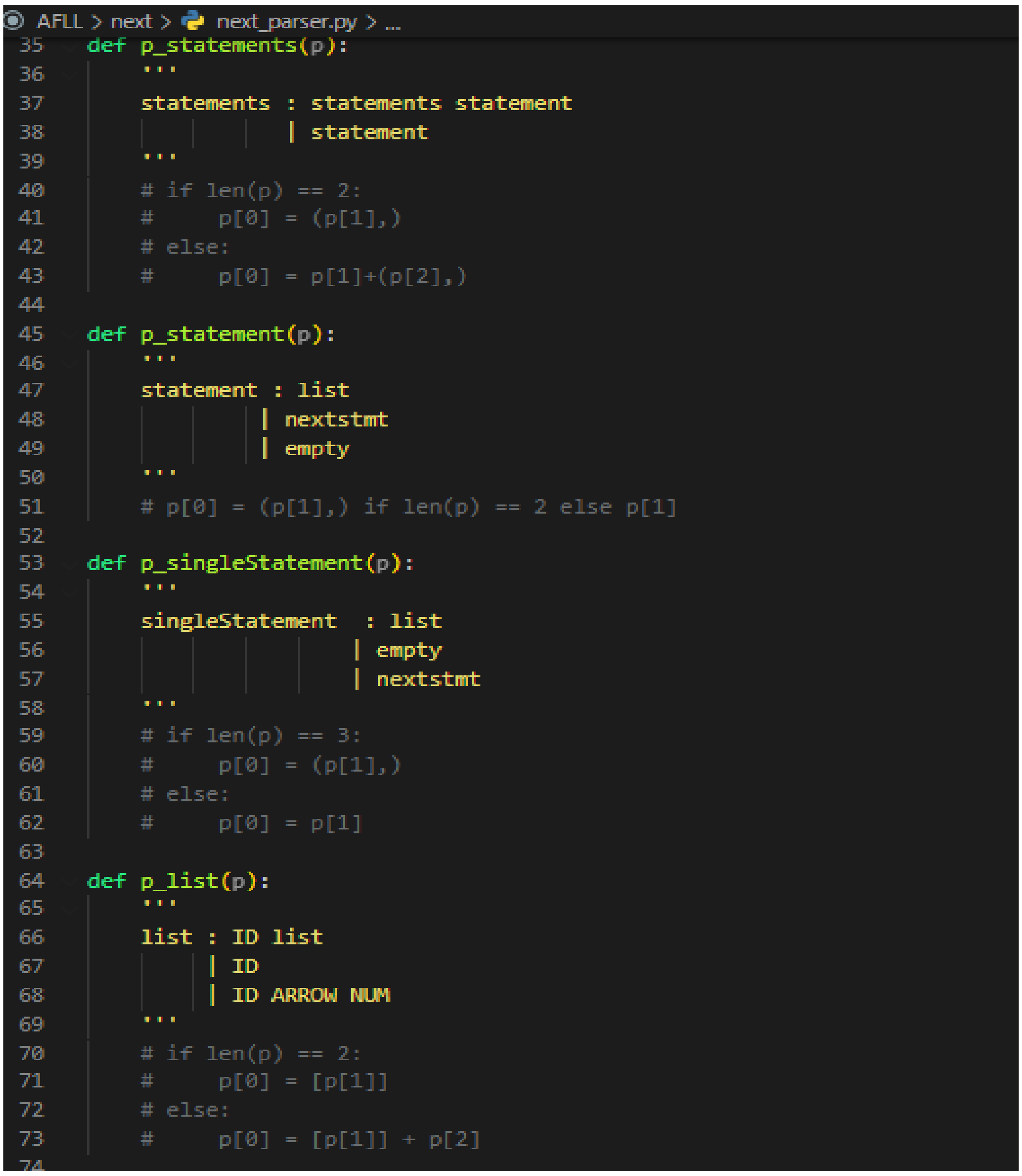


```python
def t_NUM(t):
    r'[0-9][0-9]*'
    return t

def t_IN(t):
    r'in'
    return t

t_ignore = ' \t'


def t_ID(t):
    r'\b([a-zA-Z_=][a-zA-Z_0-9=]*)\b |\b(\d+)\b'
    return t

def t_error(t):
    print(f"Illegal character encountered {t.value[0]}")
    t.lexer.skip(1)

lexer=lex.lex()

data=input()
lexer.input(data)
while(1):
    tok=lexer.token()
    if not tok:
        break
    print(tok)
```

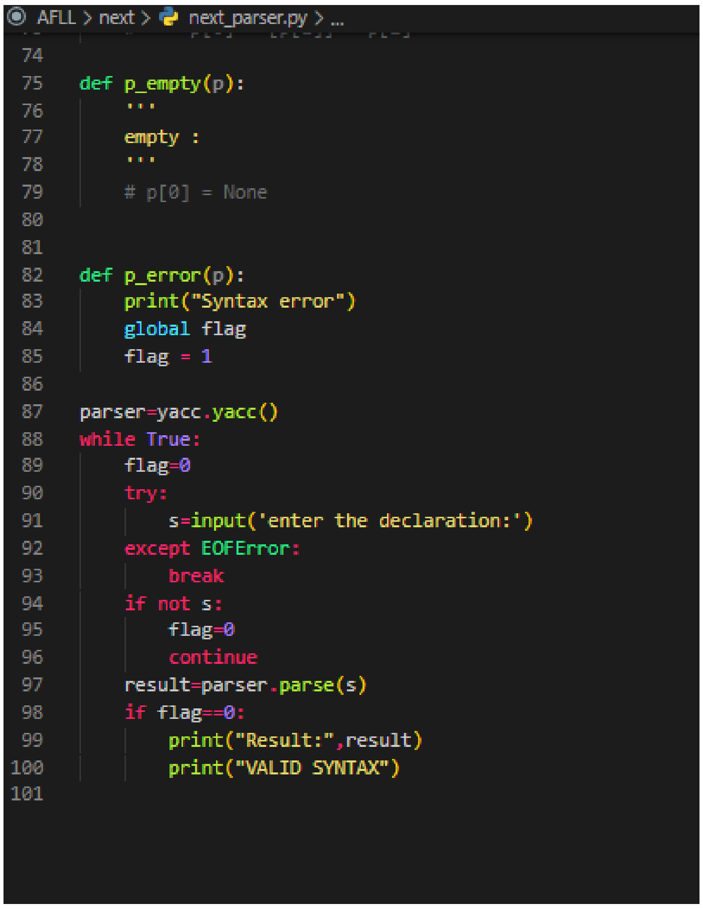
Yacc file


AFLL > next > next_parser.py > ...

```python
import ply.yacc as yacc
from next_lexer import tokens
from next_lexer import data
flag=0

def p_nextstmt(p):
    '''
    nextstmt :   FOR LBRACKET ID IN NUM COLON NUM RBRACKET LFLOWER NEXT statements RFLOWER
             |   FOR LBRACKET ID IN ID RBRACKET LFLOWER NEXT statements RFLOWER
             |   FOR LBRACKET ID IN NUM COLON NUM RBRACKET NEXT singleStatement
             |   FOR LBRACKET ID IN ID RBRACKET NEXT singleStatement

    '''
    #this works for
    #case 1: for(x in 1:10){next a<-10}
    #case 2: for(x in abc){next a<-10}
    #case 3: for(x in 1:10)next a<-10
    #case 4: for(x in abc) next a<-10

    # if (len(p) == 8 and p[1]=='IF'):
    #     p[0] = (p[1],p[3],p[6],p[7])
    # elif (len(p)==13 and p[0]=='FOR'):
    #     p[0]=  (p[1],p[3],p[4],p[5],p[6],p[7],p[10],p[11])
    # elif len(p) == 9 and p[0]=='FOR':
    #     p[0] = (p[1], p[3], p[4], p[5], p[7],p[8])
    # elif len(p) == 11 and p[6] == ':' and p[0]=='FOR':
    #     p[0] = (p[1], p[3], p[4], p[5], p[6], p[7], p[9],p[10])
    # elif (p[1]=='FOR'):
    #     p[0] = (p[1], p[3], p[4], p[5], p[8],p[9])
    # elif len(p) == 9 and p[0]=='WHILE':
    #     p[0] = (p[1],p[3],p[6],p[7])
    # else:
```



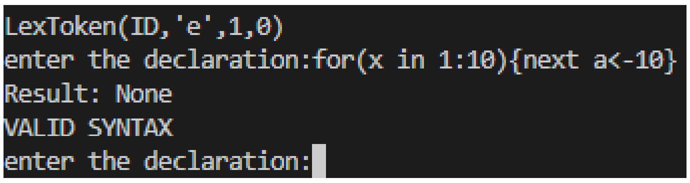
```
def p_statements(p):
    '''
    statements : statements statement
               | statement
    '''
    # if len(p) == 2:
    #     p[0] = (p[1],)
    # else:
    #     p[0] = p[1]+(p[2],)

def p_statement(p):
    '''
    statement : list
              | nextstmt
              | empty
    '''
    # p[0] = (p[1],) if len(p) == 2 else p[1]

def p_singleStatement(p):
    '''
    singleStatement  : list
                     | empty
                     | nextstmt
    '''
    # if len(p) == 3:
    #     p[0] = (p[1],)
    # else:
    #     p[0] = p[1]

def p_list(p):
    '''
    list : ID list
         | ID
         | ID ARROW NUM
    '''
    # if len(p) == 2:
    #     p[0] = [p[1]]
    # else:
    #     p[0] = [p[1]] + p[2]
```



AFLL > next > next_parser.py > ...

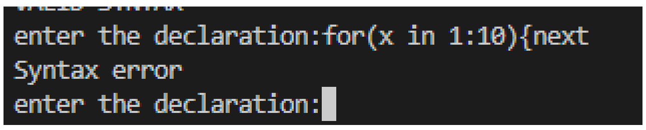
```python

def p_empty(p):
    '''
    empty :
    '''
    # p[0] = None


def p_error(p):
    print("Syntax error")
    global flag
    flag = 1

parser=yacc.yacc()
while True:
    flag=0
    try:
        s=input('enter the declaration:')
    except EOFError:
        break
    if not s:
        flag=0
        continue
    result=parser.parse(s)
    if flag==0:
        print("Result:",result)
        print("VALID SYNTAX")

```

Valid Test Case



```
LexToken(ID,'e',1,0)
enter the declaration:for(x in 1:10){next a<-10}
Result: None
VALID SYNTAX
enter the declaration:
```

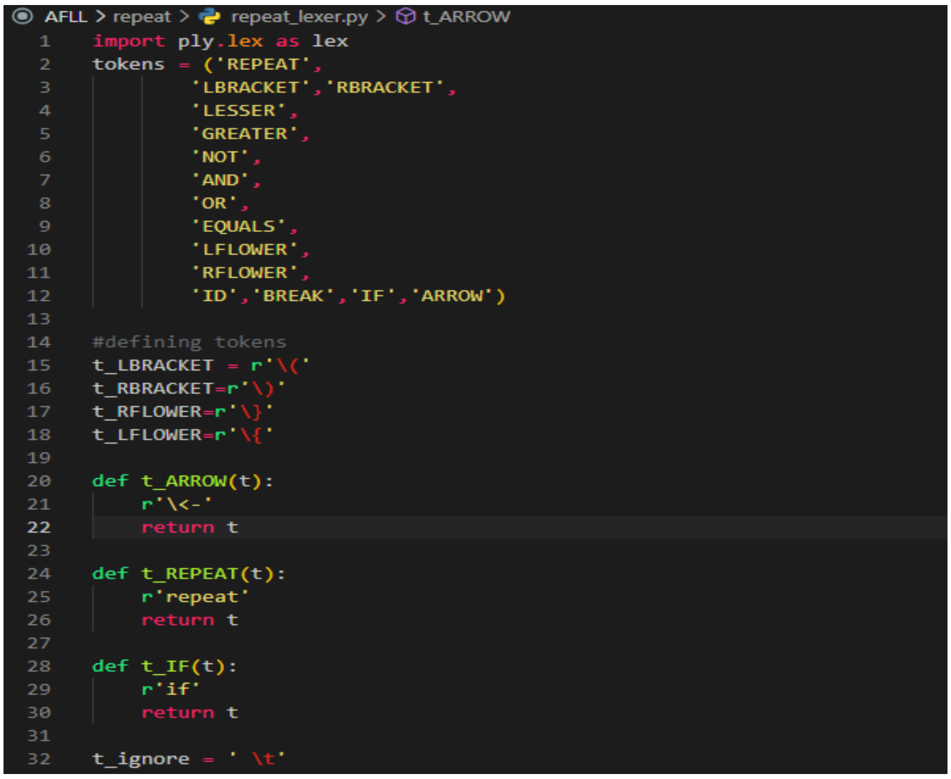
Invalid Test Case



```
enter the declaration:for(x in 1:10){next
Syntax error
enter the declaration:
```

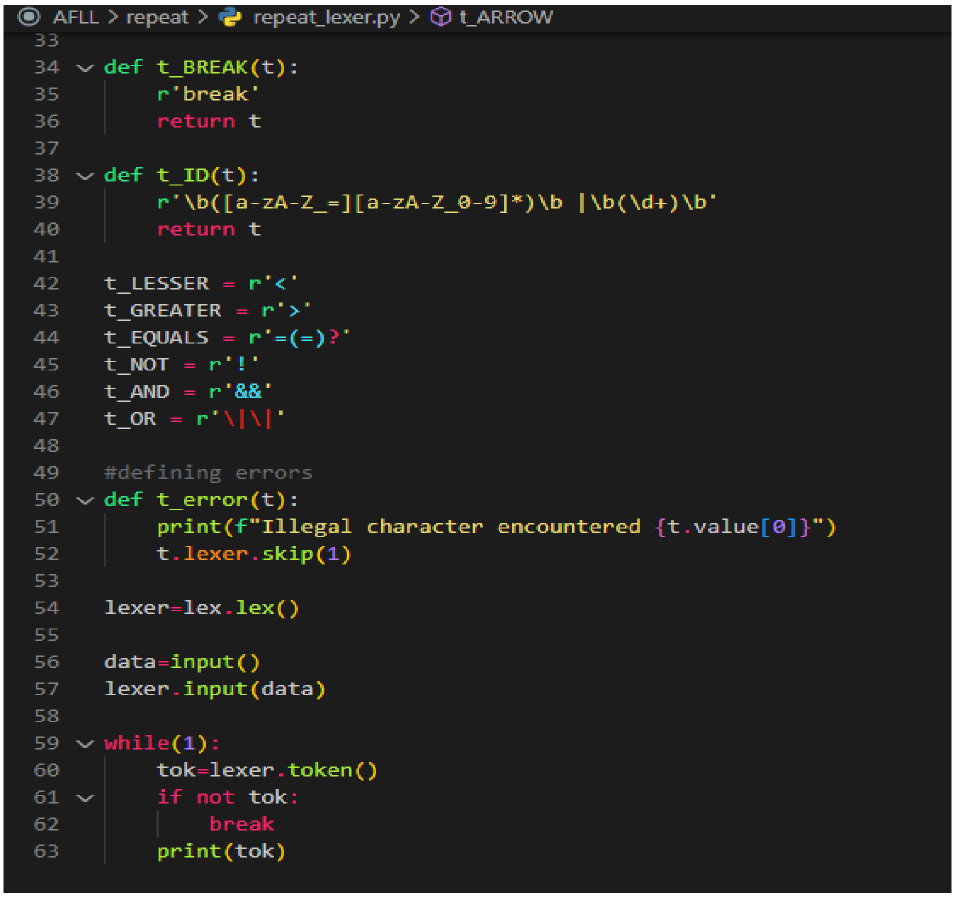
4) Repeat

Lex file



AFLL > repeat > repeat_lexer.py > t_ARROW

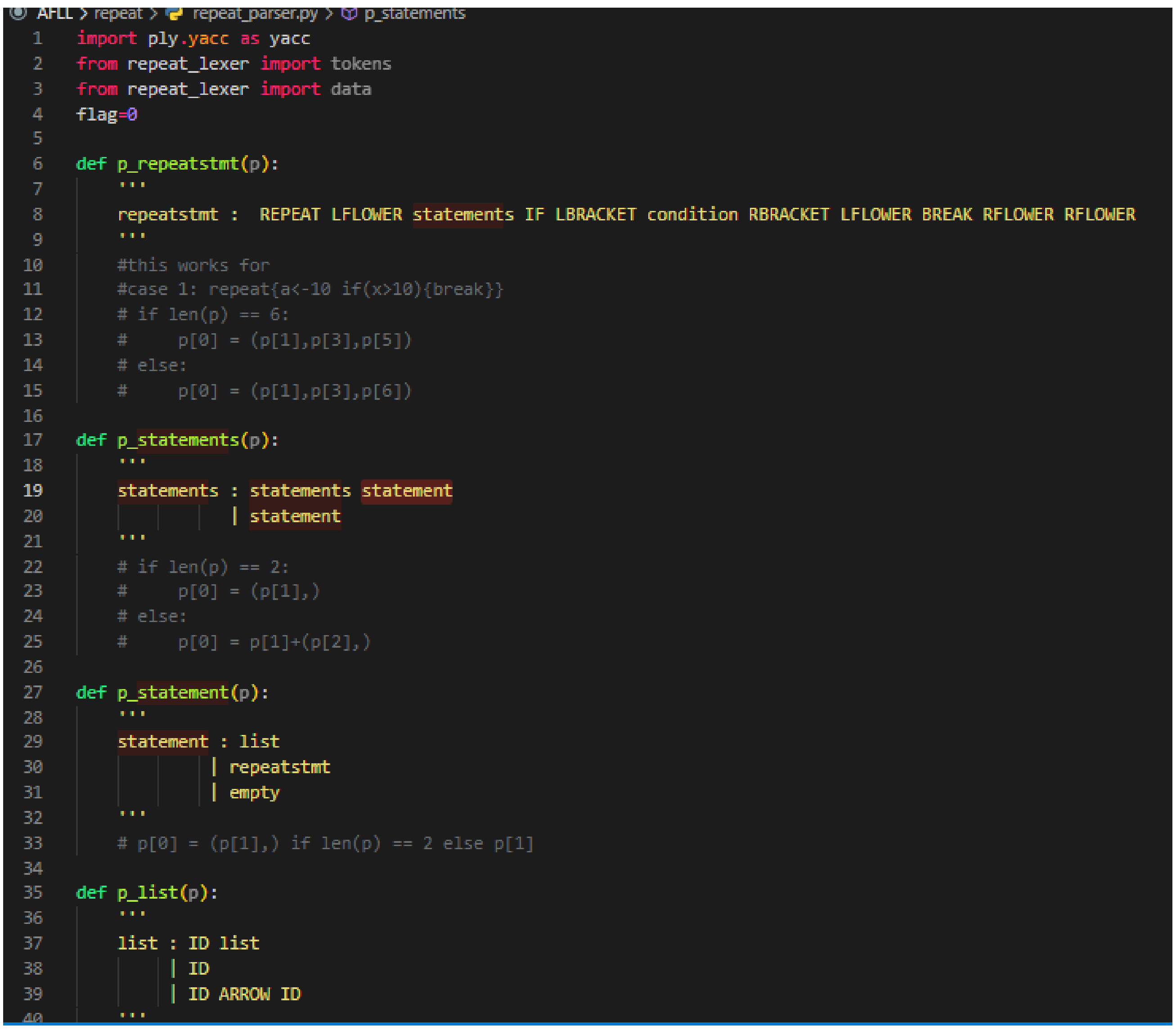
```python
import ply.lex as lex
tokens = ('REPEAT',
          'LBRACKET','RBRACKET',
          'LESSER',
          'GREATER',
          'NOT',
          'AND',
          'OR',
          'EQUALS',
          'LFLOWER',
          'RFLOWER',
          'ID','BREAK','IF','ARROW')

#defining tokens
t_LBRACKET = r'\('
t_RBRACKET=r'\)'
t_RFLOWER=r'\}'
t_LFLOWER=r'\{'

def t_ARROW(t):
    r'\<-'
    return t

def t_REPEAT(t):
    r'repeat'
    return t

def t_IF(t):
    r'if'
    return t

t_ignore = ' \t'
```



AFLL > repeat > repeat_lexer.py > t_ARROW

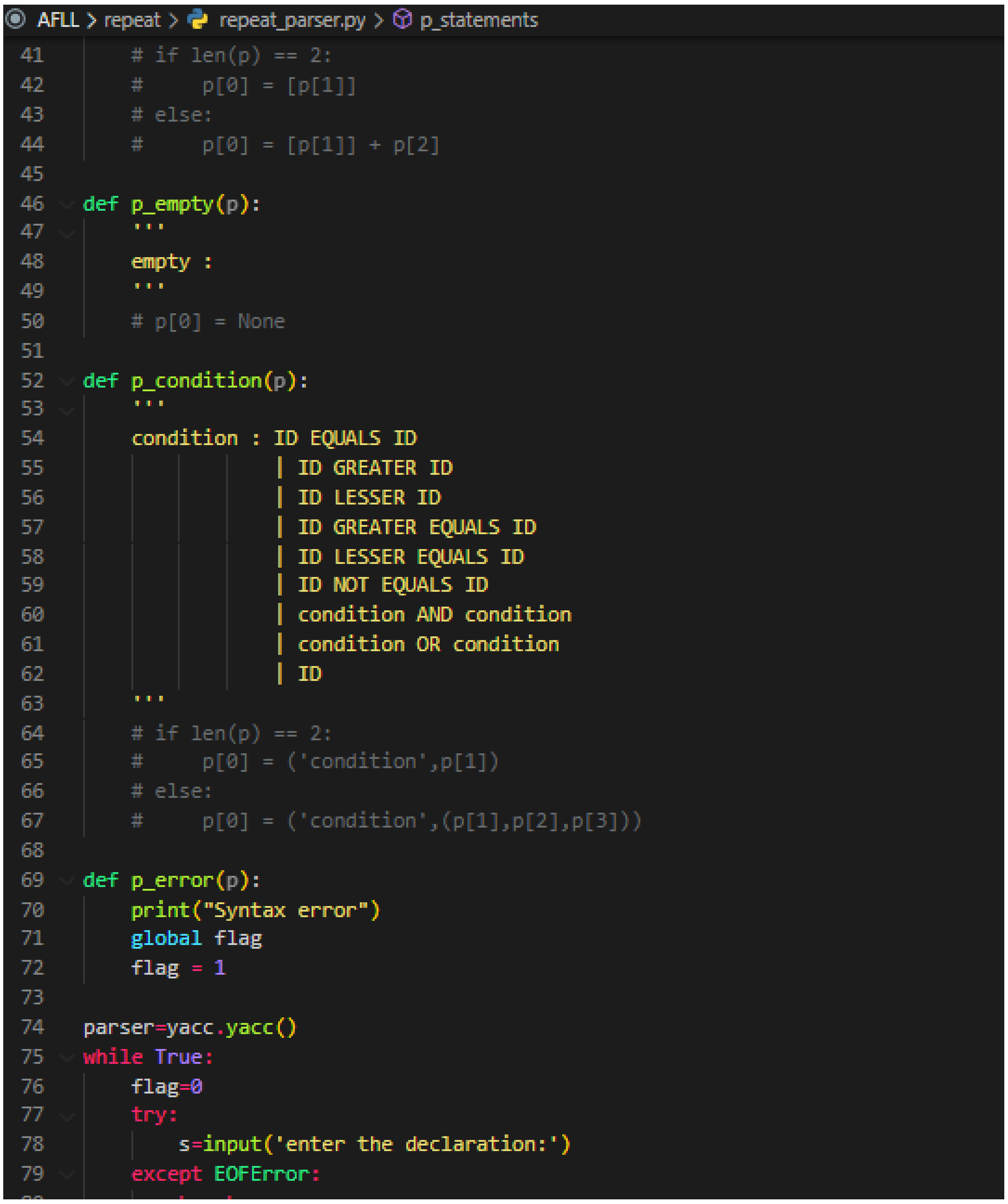
```python

def t_BREAK(t):
    r'break'
    return t

def t_ID(t):
    r'\b([a-zA-Z_=][a-zA-Z_0-9]*)\b |\b(\d+)\b'
    return t

t_LESSER = r'<'
t_GREATER = r'>'
t_EQUALS = r'=(=)?'
t_NOT = r'!'
t_AND = r'&&'
t_OR = r'\|\|'

#defining errors
def t_error(t):
    print(f"Illegal character encountered {t.value[0]}")
    t.lexer.skip(1)

lexer=lex.lex()

data=input()
lexer.input(data)

while(1):
    tok=lexer.token()
    if not tok:
        break
    print(tok)
```

Yacc file

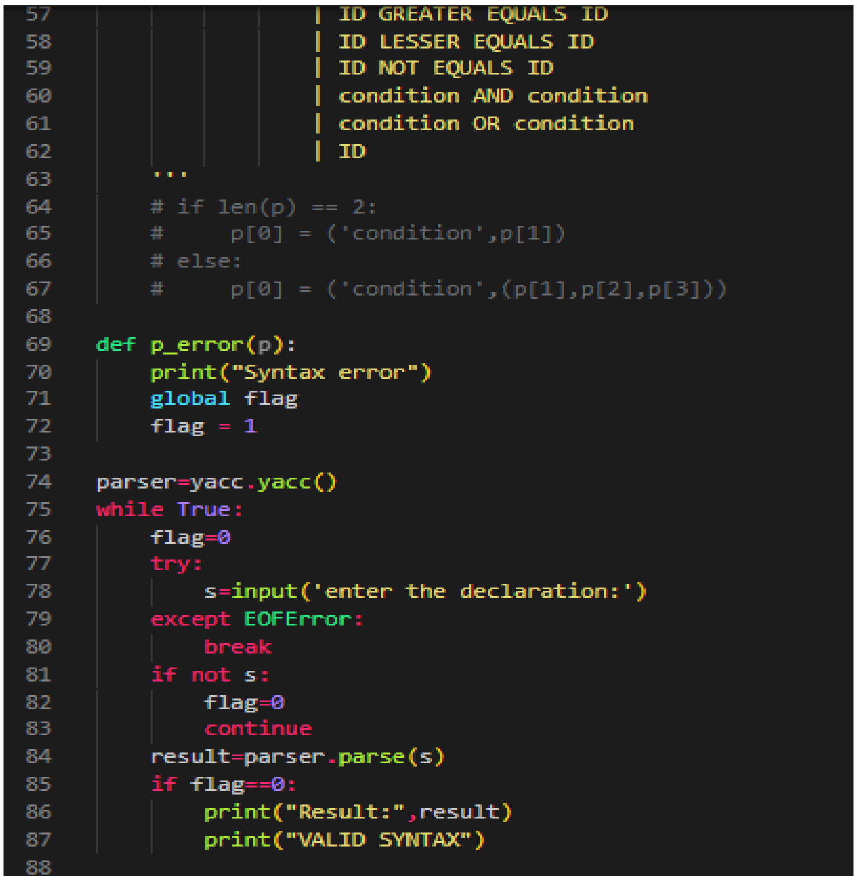


AFLL > repeat > repeat_parser.py > p_statements

```
import ply.yacc as yacc
from repeat_lexer import tokens
from repeat_lexer import data
flag=0

def p_repeatstmt(p):
    '''
    repeatstmt :  REPEAT LFLOWER statements IF LBRACKET condition RBRACKET LFLOWER BREAK RFLOWER RFLOWER
    '''
    #this works for
    #case 1: repeat{a<-10 if(x>10){break}}
    # if len(p) == 6:
    #     p[0] = (p[1],p[3],p[5])
    # else:
    #     p[0] = (p[1],p[3],p[6])

def p_statements(p):
    '''
    statements : statements statement
               | statement
    '''
    # if len(p) == 2:
    #     p[0] = (p[1],)
    # else:
    #     p[0] = p[1]+(p[2],)

def p_statement(p):
    '''
    statement : list
              | repeatstmt
              | empty
    '''
    # p[0] = (p[1],) if len(p) == 2 else p[1]

def p_list(p):
    '''
    list : ID list
         | ID
         | ID ARROW ID
    '''
```

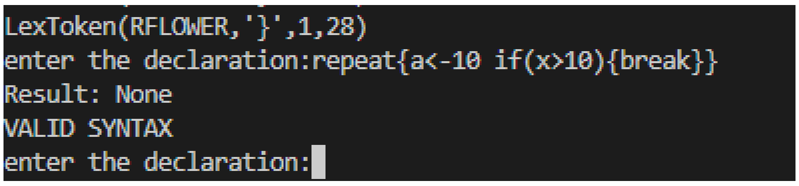


```python
    # if len(p) == 2:
    #     p[0] = [p[1]]
    # else:
    #     p[0] = [p[1]] + p[2]

def p_empty(p):
    '''
    empty :
    '''
    # p[0] = None

def p_condition(p):
    '''
    condition : ID EQUALS ID
              | ID GREATER ID
              | ID LESSER ID
              | ID GREATER EQUALS ID
              | ID LESSER EQUALS ID
              | ID NOT EQUALS ID
              | condition AND condition
              | condition OR condition
              | ID
    '''
    # if len(p) == 2:
    #     p[0] = ('condition',p[1])
    # else:
    #     p[0] = ('condition',(p[1],p[2],p[3]))

def p_error(p):
    print("Syntax error")
    global flag
    flag = 1

parser=yacc.yacc()
while True:
    flag=0
    try:
        s=input('enter the declaration:')
    except EOFError:
```



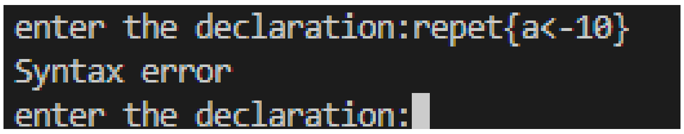
```
                    | ID GREATER EQUALS ID
                    | ID LESSER EQUALS ID
                    | ID NOT EQUALS ID
                    | condition AND condition
                    | condition OR condition
                    | ID
    '''
    # if len(p) == 2:
    #     p[0] = ('condition',p[1])
    # else:
    #     p[0] = ('condition',(p[1],p[2],p[3]))

def p_error(p):
    print("Syntax error")
    global flag
    flag = 1

parser=yacc.yacc()
while True:
    flag=0
    try:
        s=input('enter the declaration:')
    except EOFError:
        break
    if not s:
        flag=0
        continue
    result=parser.parse(s)
    if flag==0:
        print("Result:",result)
        print("VALID SYNTAX")
```

Valid Test Case



```
LexToken(RFLOWER,'}',1,28)
enter the declaration:repeat{a<-10 if(x>10){break}}
Result: None
VALID SYNTAX
enter the declaration:
```

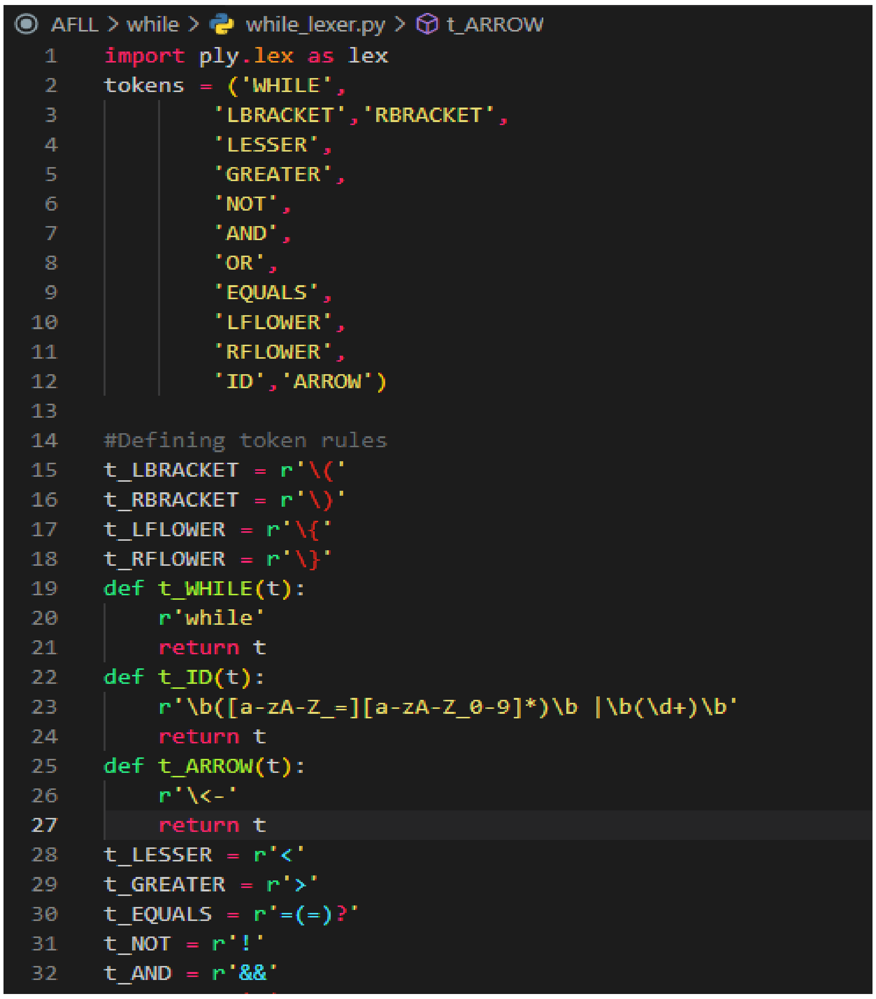
Invalid Test Case



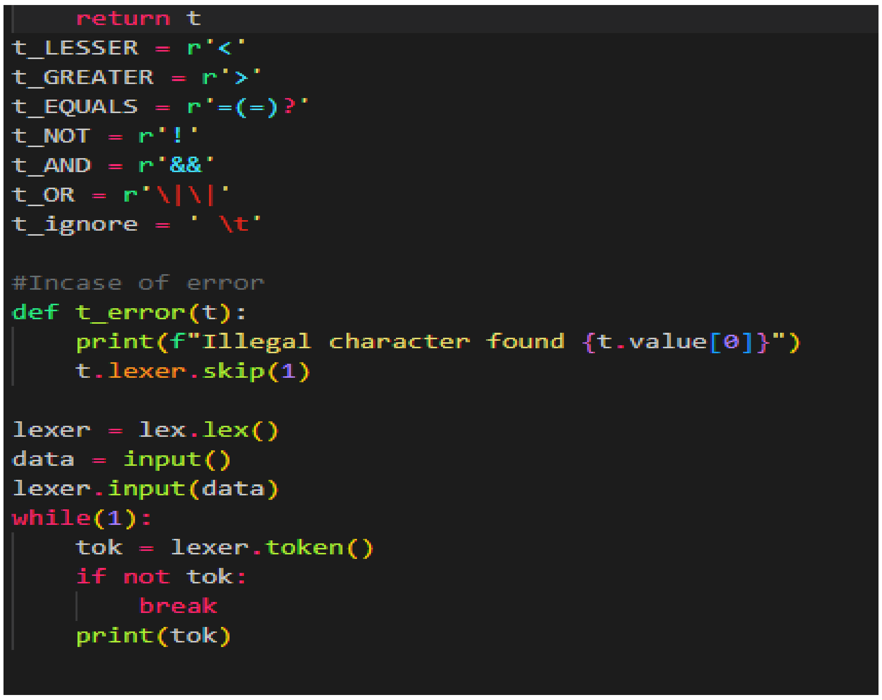
```
enter the declaration:repet{a<-10}
Syntax error
enter the declaration:
```

5) While loop

Lex file



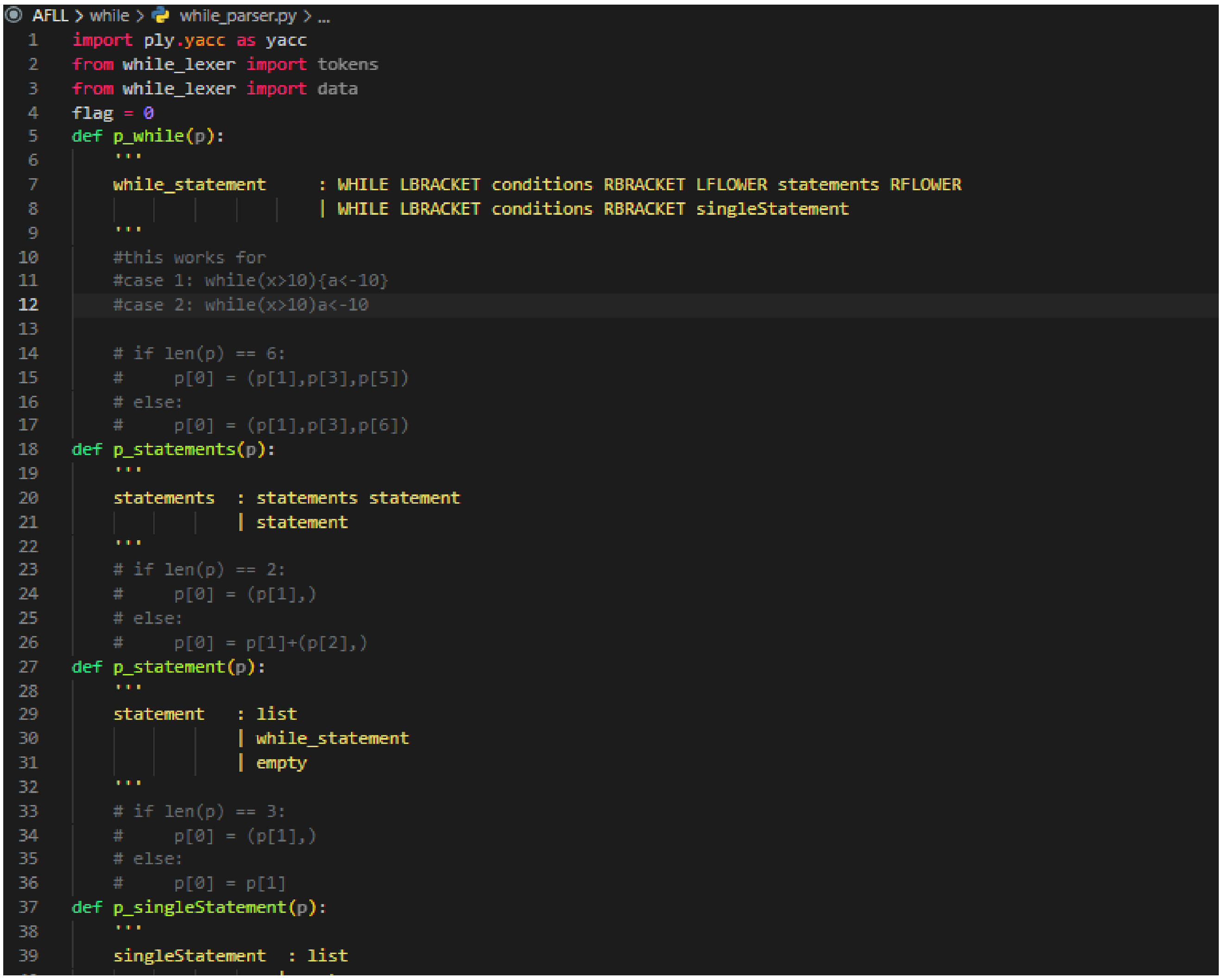
```python
import ply.lex as lex
tokens = ('WHILE',
          'LBRACKET','RBRACKET',
          'LESSER',
          'GREATER',
          'NOT',
          'AND',
          'OR',
          'EQUALS',
          'LFLOWER',
          'RFLOWER',
          'ID','ARROW')

#Defining token rules
t_LBRACKET = r'\('
t_RBRACKET = r'\)'
t_LFLOWER = r'\{'
t_RFLOWER = r'\}'
def t_WHILE(t):
    r'while'
    return t
def t_ID(t):
    r'\b([a-zA-Z_=][a-zA-Z_0-9]*)\b |\b(\d+)\b'
    return t
def t_ARROW(t):
    r'\<-'
    return t
t_LESSER = r'<'
t_GREATER = r'>'
t_EQUALS = r'=(=)?'
t_NOT = r'!'
t_AND = r'&&'
```



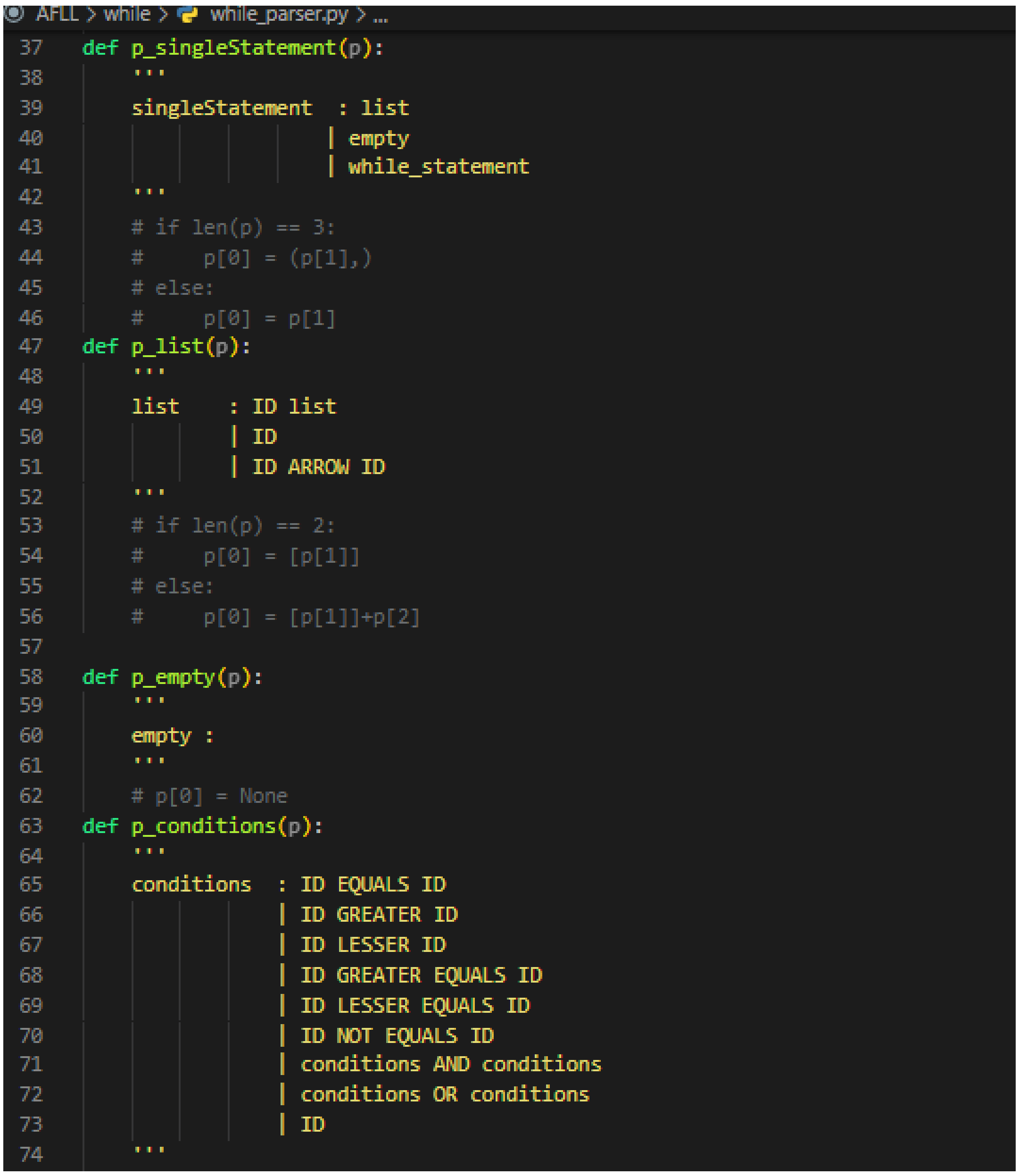
```python
    return t
t_LESSER = r'<'
t_GREATER = r'>'
t_EQUALS = r'=(=)?'
t_NOT = r'!'
t_AND = r'&&'
```

```python
t_OR = r'\|\|'
t_ignore = ' \t'

#Incase of error
def t_error(t):
    print(f"Illegal character found {t.value[0]}")
    t.lexer.skip(1)

lexer = lex.lex()
data = input()
lexer.input(data)
while(1):
    tok = lexer.token()
    if not tok:
        break
    print(tok)
```

Yacc file

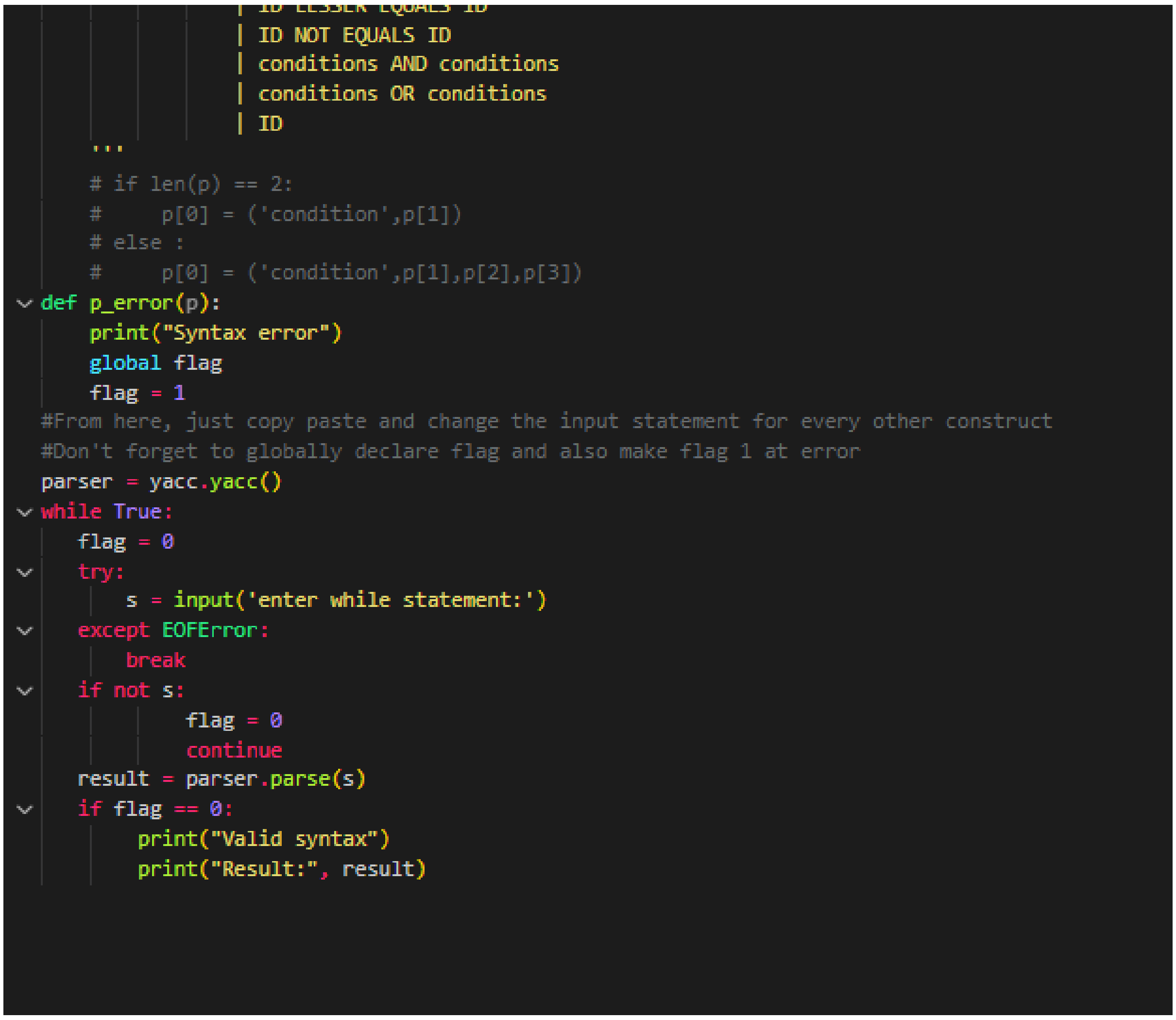


```python
import ply.yacc as yacc
from while_lexer import tokens
from while_lexer import data
flag = 0
def p_while(p):
    '''
    while_statement     : WHILE LBRACKET conditions RBRACKET LFLOWER statements RFLOWER
                        | WHILE LBRACKET conditions RBRACKET singleStatement
    '''
    #this works for
    #case 1: while(x>10){a<-10}
    #case 2: while(x>10)a<-10

    # if len(p) == 6:
    #     p[0] = (p[1],p[3],p[5])
    # else:
    #     p[0] = (p[1],p[3],p[6])
def p_statements(p):
    '''
    statements  : statements statement
                | statement
    '''
    # if len(p) == 2:
    #     p[0] = (p[1],)
    # else:
    #     p[0] = p[1]+(p[2],)
def p_statement(p):
    '''
    statement   : list
                | while_statement
                | empty
    '''
    # if len(p) == 3:
    #     p[0] = (p[1],)
    # else:
    #     p[0] = p[1]
def p_singleStatement(p):
    '''
    singleStatement  : list
```



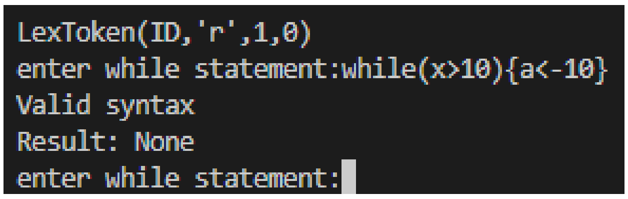
AFLL > while > while_parser.py > ...

```python
def p_singleStatement(p):
    '''
    singleStatement  : list
                     | empty
                     | while_statement
    '''
    # if len(p) == 3:
    #     p[0] = (p[1],)
    # else:
    #     p[0] = p[1]
def p_list(p):
    '''
    list     : ID list
             | ID
             | ID ARROW ID
    '''
    # if len(p) == 2:
    #     p[0] = [p[1]]
    # else:
    #     p[0] = [p[1]]+p[2]

def p_empty(p):
    '''
    empty :
    '''
    # p[0] = None
def p_conditions(p):
    '''
    conditions   : ID EQUALS ID
                 | ID GREATER ID
                 | ID LESSER ID
                 | ID GREATER EQUALS ID
                 | ID LESSER EQUALS ID
                 | ID NOT EQUALS ID
                 | conditions AND conditions
                 | conditions OR conditions
                 | ID
    '''
```



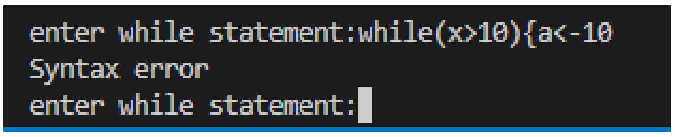
```
                  | ID LESSER EQUALS ID
                  | ID NOT EQUALS ID
                  | conditions AND conditions
                  | conditions OR conditions
                  | ID
    '''
    # if len(p) == 2:
    #     p[0] = ('condition',p[1])
    # else :
    #     p[0] = ('condition',p[1],p[2],p[3])
def p_error(p):
    print("Syntax error")
    global flag
    flag = 1
#From here, just copy paste and change the input statement for every other construct
#Don't forget to globally declare flag and also make flag 1 at error
parser = yacc.yacc()
while True:
   flag = 0
   try:
       s = input('enter while statement:')
   except EOFError:
       break
   if not s:
            flag = 0
            continue
   result = parser.parse(s)
   if flag == 0:
        print("Valid syntax")
        print("Result:", result)
```

Valid Test Case



```
LexToken(ID,'r',1,0)
enter while statement:while(x>10){a<-10}
Valid syntax
Result: None
enter while statement:
```

Invalid Test Case



```
enter while statement:while(x>10){a<-10
Syntax error
enter while statement:
```