UE22CS243A : Automata Formal Languages and Logic

PROJECT REPORT

Nagathejas M S
PES1UG22AM098

Malleshappa D Patil
PES1UG22AM090

Problem Statement –

Syntax Validation of a programming language by writing the Context Free Grammar. (PLY Tools).

Language and constructs that we are going to validate are –

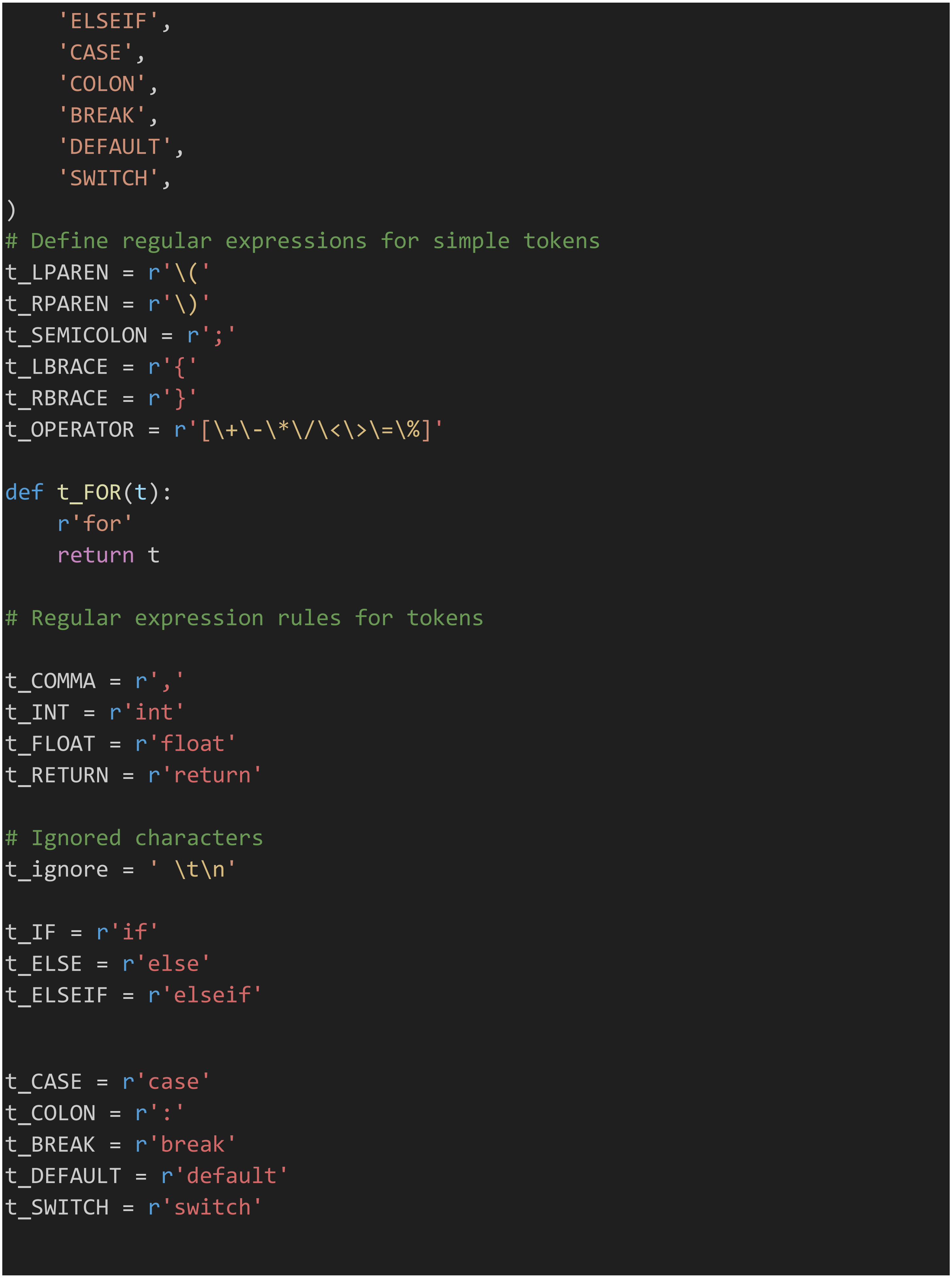
C++

1. For loop
2. Function declaration and definition
3. If-else
4. Nested if-else
5. Switch case

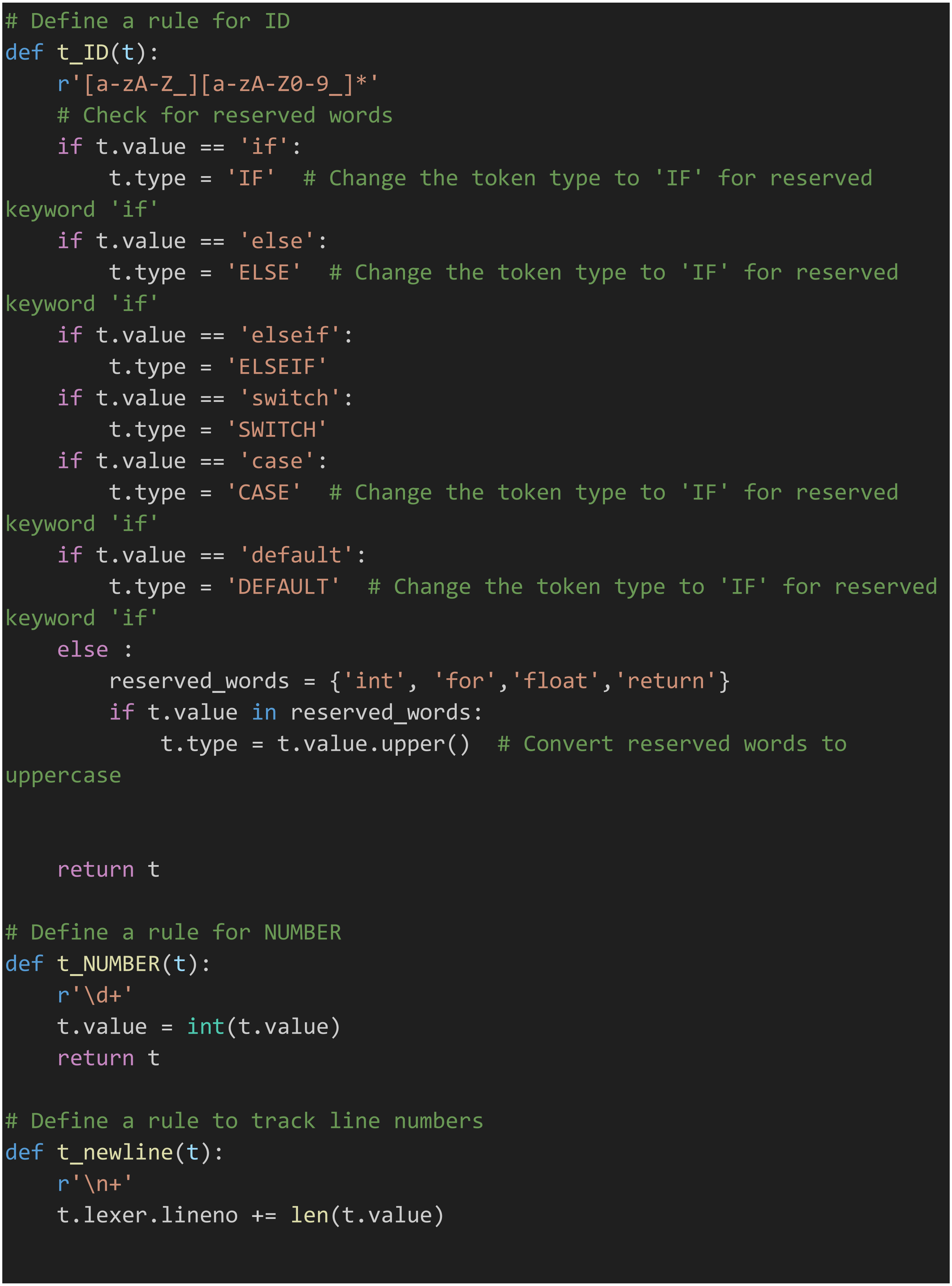
Solution –

Python program –

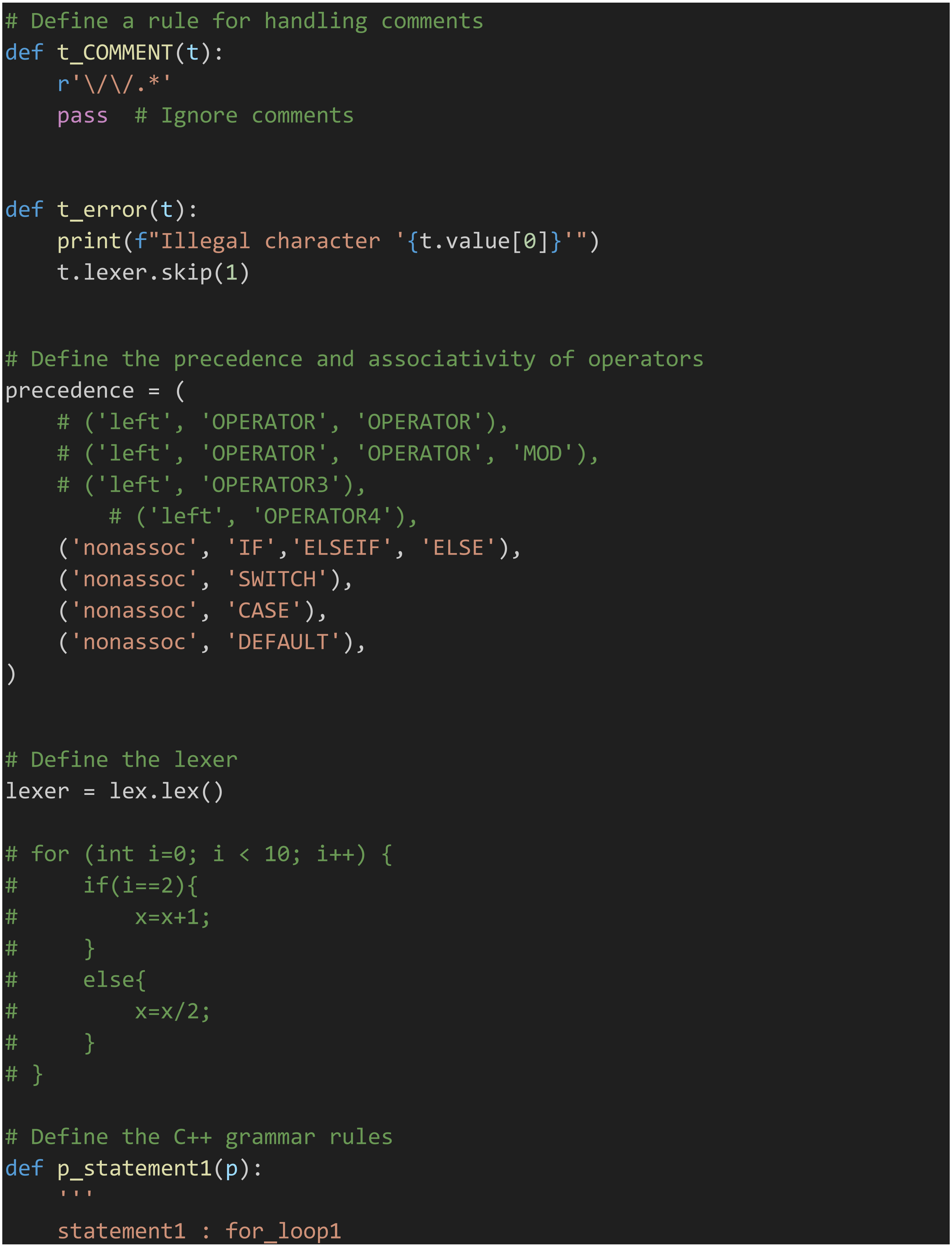
```
#for_loop


#for syntax validation

import ply.yacc as yacc
import ply.lex as lex
# Define the C++ lexer tokens
tokens = (
    'ID', 'LPAREN', 'RPAREN', 'SEMICOLON',
    'INT', 'FOR', 'OPERATOR', 'NUMBER','LBRACE','RBRACE',
    'COMMA',
    'FLOAT',
    'RETURN',
    'IF',
    'ELSE',
```



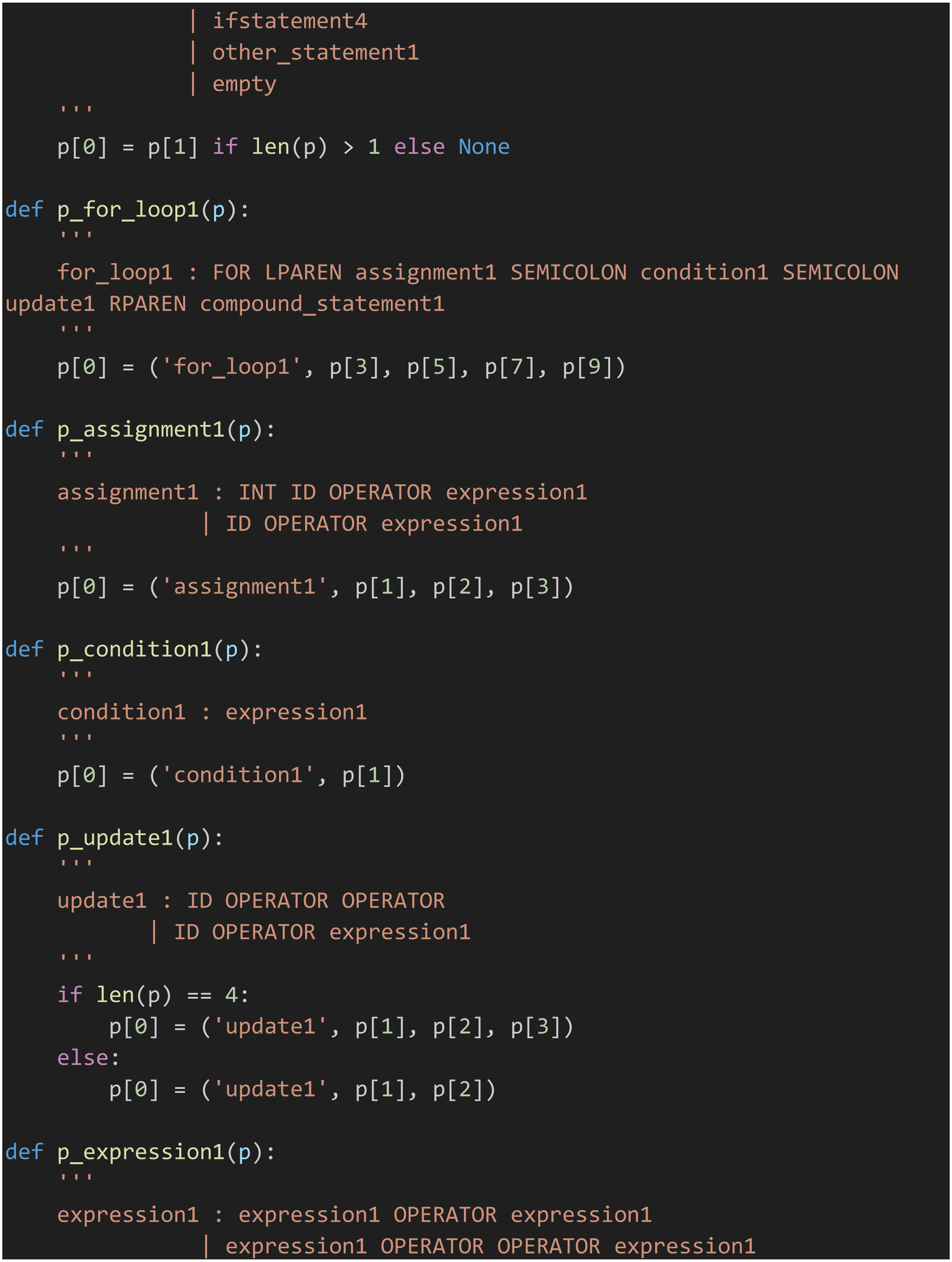
```
    'ELSEIF',
    'CASE',
    'COLON',
    'BREAK',
    'DEFAULT',
    'SWITCH',
)
# Define regular expressions for simple tokens
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_SEMICOLON = r';'
t_LBRACE = r'{'
t_RBRACE = r'}'
t_OPERATOR = r'[\+\-\*\/\<\>\=\%]'

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

# Regular expression rules for tokens

t_COMMA = r','
t_INT = r'int'
t_FLOAT = r'float'
t_RETURN = r'return'

# Ignored characters
t_ignore = ' \t\n'

t_IF = r'if'
t_ELSE = r'else'
t_ELSEIF = r'elseif'


t_CASE = r'case'
t_COLON = r':'
t_BREAK = r'break'
t_DEFAULT = r'default'
t_SWITCH = r'switch'
```



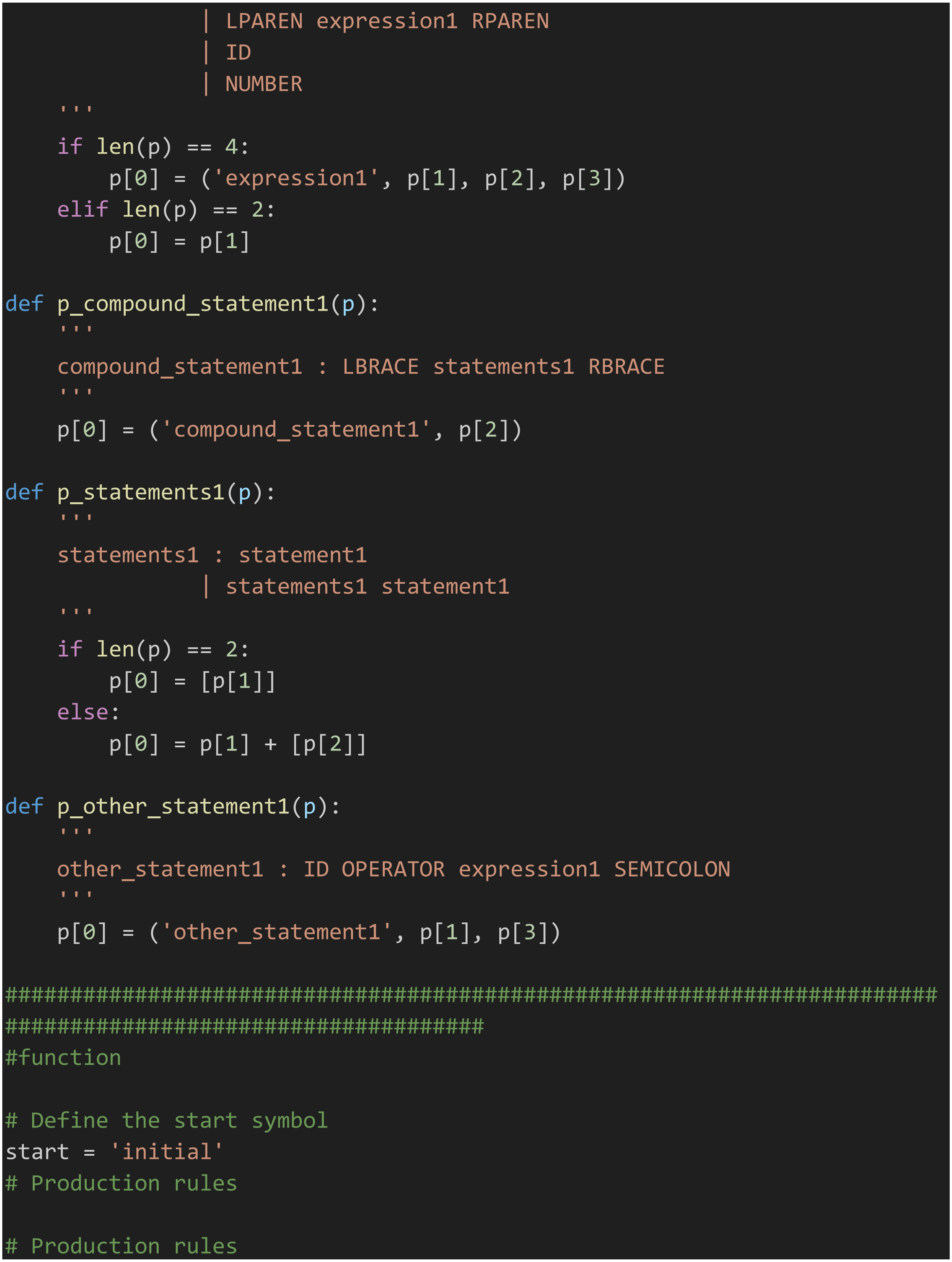
```python
# Define a rule for ID
def t_ID(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
    # Check for reserved words
    if t.value == 'if':
        t.type = 'IF'  # Change the token type to 'IF' for reserved keyword 'if'
    if t.value == 'else':
        t.type = 'ELSE'  # Change the token type to 'IF' for reserved keyword 'if'
    if t.value == 'elseif':
        t.type = 'ELSEIF'
    if t.value == 'switch':
        t.type = 'SWITCH'
    if t.value == 'case':
        t.type = 'CASE'  # Change the token type to 'IF' for reserved keyword 'if'
    if t.value == 'default':
        t.type = 'DEFAULT'  # Change the token type to 'IF' for reserved keyword 'if'
    else :
        reserved_words = {'int', 'for','float','return'}
        if t.value in reserved_words:
            t.type = t.value.upper()  # Convert reserved words to uppercase


    return t

# Define a rule for NUMBER
def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)
    return t

# Define a rule to track line numbers
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)
```



```
# Define a rule for handling comments
def t_COMMENT(t):
    r'\/\/.*'
    pass  # Ignore comments


def t_error(t):
    print(f"Illegal character '{t.value[0]}'")
    t.lexer.skip(1)


# Define the precedence and associativity of operators
precedence = (
    # ('left', 'OPERATOR', 'OPERATOR'),
    # ('left', 'OPERATOR', 'OPERATOR', 'MOD'),
    # ('left', 'OPERATOR3'),
        # ('left', 'OPERATOR4'),
    ('nonassoc', 'IF','ELSEIF', 'ELSE'),
    ('nonassoc', 'SWITCH'),
    ('nonassoc', 'CASE'),
    ('nonassoc', 'DEFAULT'),
)


# Define the lexer
lexer = lex.lex()

# for (int i=0; i < 10; i++) {
#     if(i==2){
#         x=x+1;
#     }
#     else{
#         x=x/2;
#     }
# }

# Define the C++ grammar rules
def p_statement1(p):
    '''
    statement1 : for_loop1
```

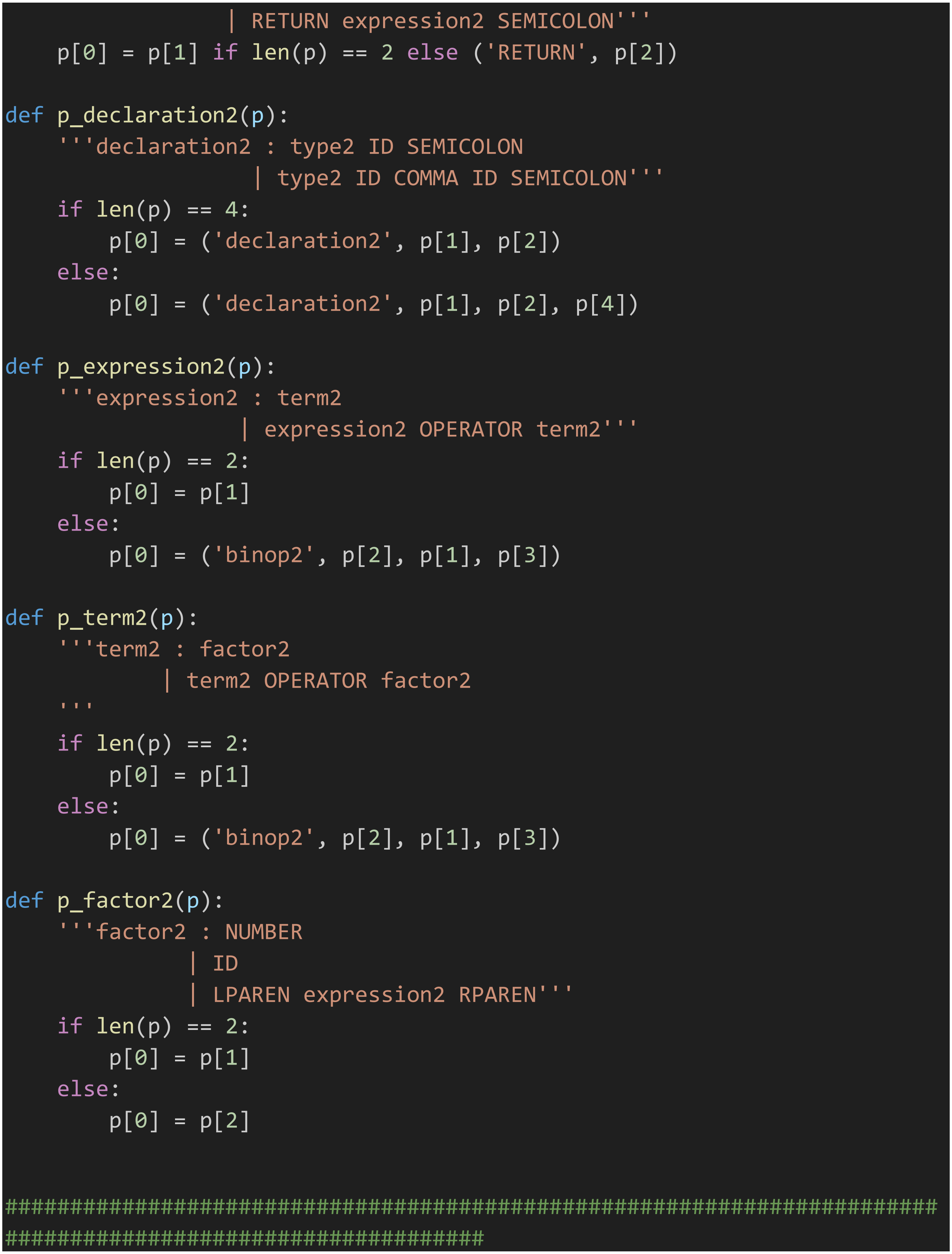


```
              | ifstatement4
              | other_statement1
              | empty
    '''
    p[0] = p[1] if len(p) > 1 else None

def p_for_loop1(p):
    '''
    for_loop1 : FOR LPAREN assignment1 SEMICOLON condition1 SEMICOLON
update1 RPAREN compound_statement1
    '''
    p[0] = ('for_loop1', p[3], p[5], p[7], p[9])

def p_assignment1(p):
    '''
    assignment1 : INT ID OPERATOR expression1
               | ID OPERATOR expression1
    '''
    p[0] = ('assignment1', p[1], p[2], p[3])

def p_condition1(p):
    '''
    condition1 : expression1
    '''
    p[0] = ('condition1', p[1])

def p_update1(p):
    '''
    update1 : ID OPERATOR OPERATOR
           | ID OPERATOR expression1
    '''
    if len(p) == 4:
        p[0] = ('update1', p[1], p[2], p[3])
    else:
        p[0] = ('update1', p[1], p[2])

def p_expression1(p):
    '''
    expression1 : expression1 OPERATOR expression1
               | expression1 OPERATOR OPERATOR expression1
```



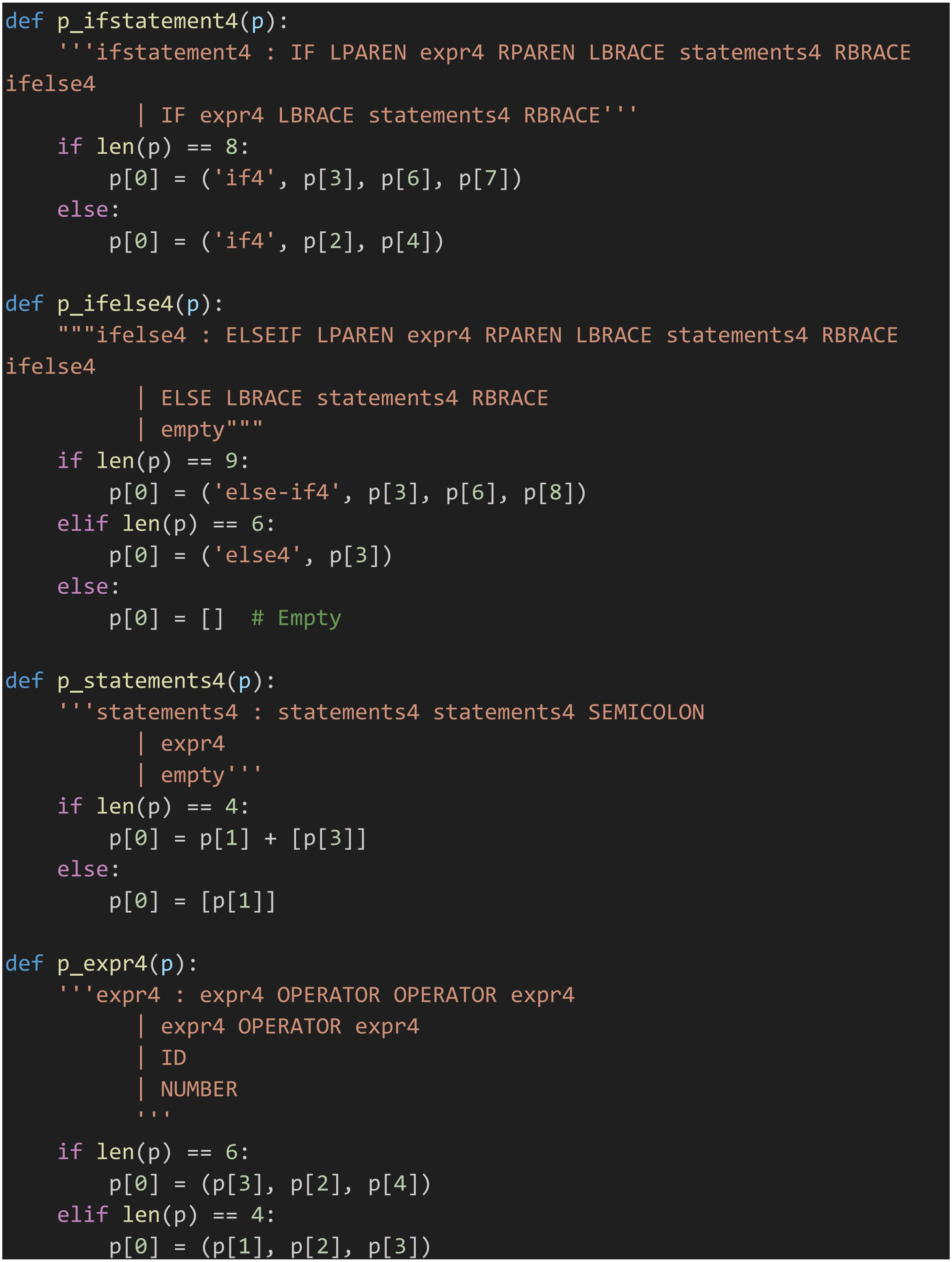
```
               | LPAREN expression1 RPAREN
               | ID
               | NUMBER
    '''
    if len(p) == 4:
        p[0] = ('expression1', p[1], p[2], p[3])
    elif len(p) == 2:
        p[0] = p[1]

def p_compound_statement1(p):
    '''
    compound_statement1 : LBRACE statements1 RBRACE
    '''
    p[0] = ('compound_statement1', p[2])

def p_statements1(p):
    '''
    statements1 : statement1
               | statements1 statement1
    '''
    if len(p) == 2:
        p[0] = [p[1]]
    else:
        p[0] = p[1] + [p[2]]

def p_other_statement1(p):
    '''
    other_statement1 : ID OPERATOR expression1 SEMICOLON
    '''
    p[0] = ('other_statement1', p[1], p[3])

########################################################################
#####################################
#function

# Define the start symbol
start = 'initial'
# Production rules

# Production rules
```

```
def p_function2(p):
    '''function2 : type2 ID LPAREN params2 RPAREN LBRACE statements2
RBRACE function2
                | type2 ID LPAREN RPAREN LBRACE statements2 RBRACE
function2
                | empty'''
    if len(p) == 2:
        p[0] = p[1]
    elif len(p) == 8:
        p[0] = ('function2', p[1], p[2], p[4], p[7]) + p[8]
    else:
        p[0] = ('function2', p[1], p[2], p[4], p[7])
def p_params2(p):
    '''params2 : param2
              | params2 COMMA param2'''
    if len(p) == 2:
        p[0] = [p[1]]
    else:
        p[0] = p[1] + [p[3]]

def p_param2(p):
    '''param2 : type2 ID'''
    p[0] = ('param2', p[1], p[2])

def p_type2(p):
    '''type2 : INT
            | FLOAT'''
    p[0] = p[1]

def p_statements2(p):
    '''statements2 : statement2
                  | statements2 statement2'''
    if len(p) == 2:
        p[0] = [p[1]]
    else:
        p[0] = p[1] + [p[2]]

def p_statement2(p):
    '''statement2 : declaration2
                 | expression2 SEMICOLON
```

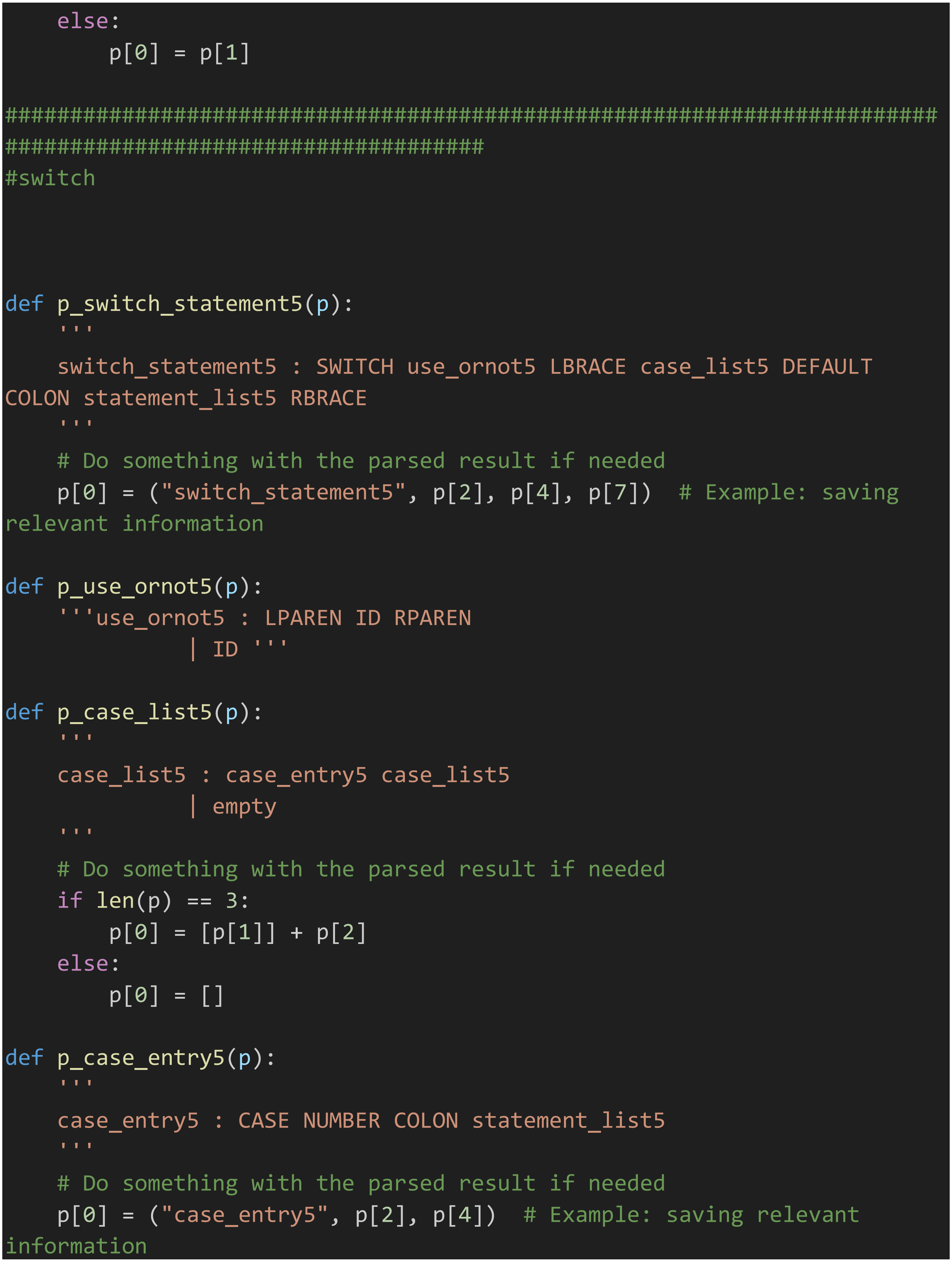


```
                 | RETURN expression2 SEMICOLON'''
    p[0] = p[1] if len(p) == 2 else ('RETURN', p[2])

def p_declaration2(p):
    '''declaration2 : type2 ID SEMICOLON
                   | type2 ID COMMA ID SEMICOLON'''
    if len(p) == 4:
        p[0] = ('declaration2', p[1], p[2])
    else:
        p[0] = ('declaration2', p[1], p[2], p[4])

def p_expression2(p):
    '''expression2 : term2
                  | expression2 OPERATOR term2'''
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[0] = ('binop2', p[2], p[1], p[3])

def p_term2(p):
    '''term2 : factor2
            | term2 OPERATOR factor2
    '''
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[0] = ('binop2', p[2], p[1], p[3])

def p_factor2(p):
    '''factor2 : NUMBER
              | ID
              | LPAREN expression2 RPAREN'''
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[0] = p[2]


########################################################################
#####################################
```

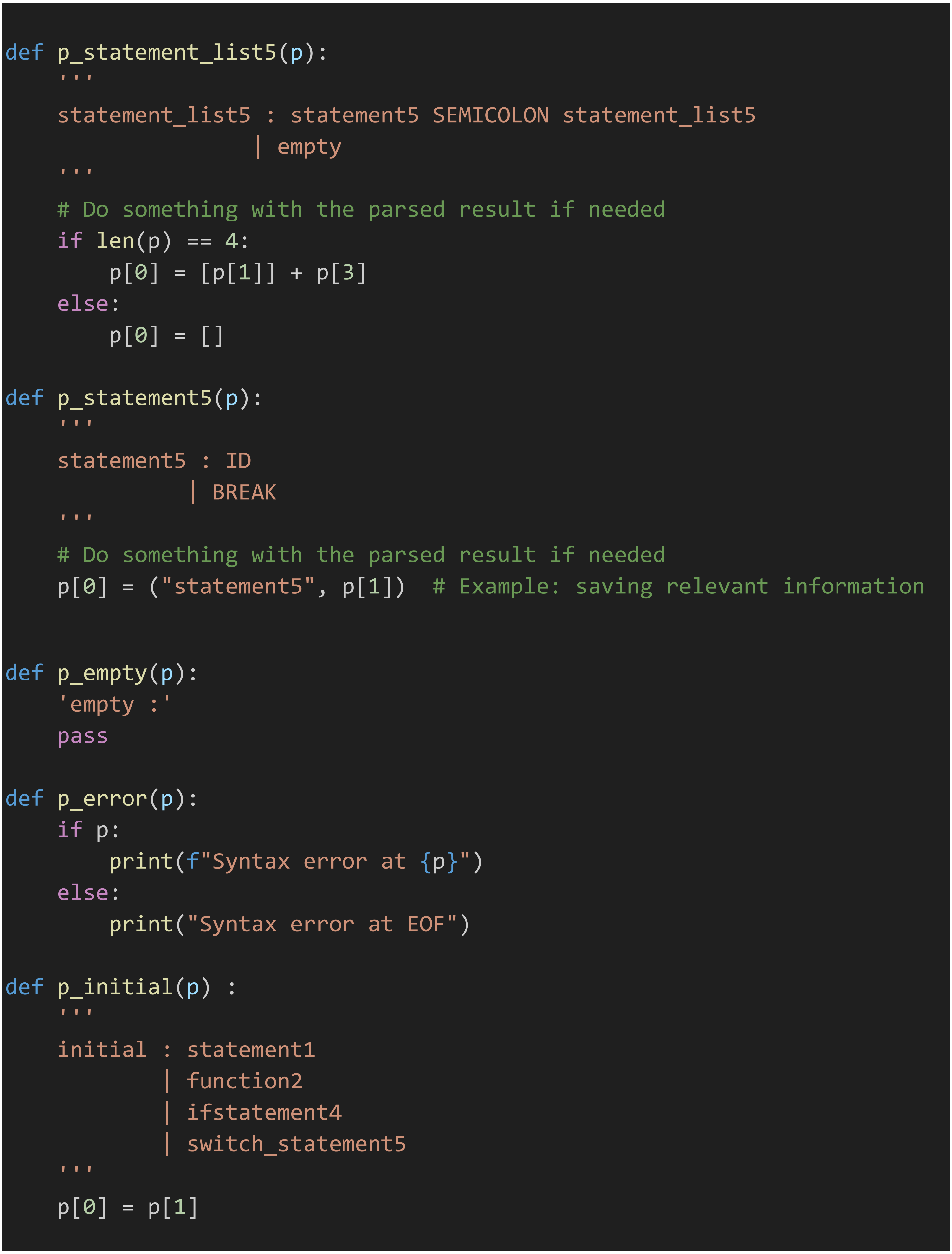
```
# ifelse


# yacc

# def p_ifstatement3(p):
#     '''ifstatement3 : IF LPAREN expr3 RPAREN LBRACE statements3 RBRACE
ELSE LBRACE statements3 RBRACE
#           | IF expr3 LBRACE statements3 RBRACE'''
#     if len(p) == 11:
#         p[0] = ('if-else3', p[3], p[6], p[10])
#     else:
#         p[0] = ('if3', p[3], p[6])

# def p_statements3(p):
#     '''statements3 : statements3 statements3 SEMICOLON
#           | expr3
#           | empty'''
#     if len(p) == 4:
#         p[0] = p[1] + [p[3]]
#     else:
#         p[0] = [p[1]]

# def p_expr3(p):
#     '''expr3 : expr3 OPERATOR OPERATOR expr3
#           | expr3 OPERATOR expr3
#           | ID
#           | NUMBER
#           '''
#     if len(p) == 6:
#         p[0] = (p[3], p[2], p[4])
#     elif len(p) == 4:
#         p[0] = (p[1], p[2], p[3])
#     else:
#         p[0] = p[1]

########################################################################
#####################################
#ifelse_full
```



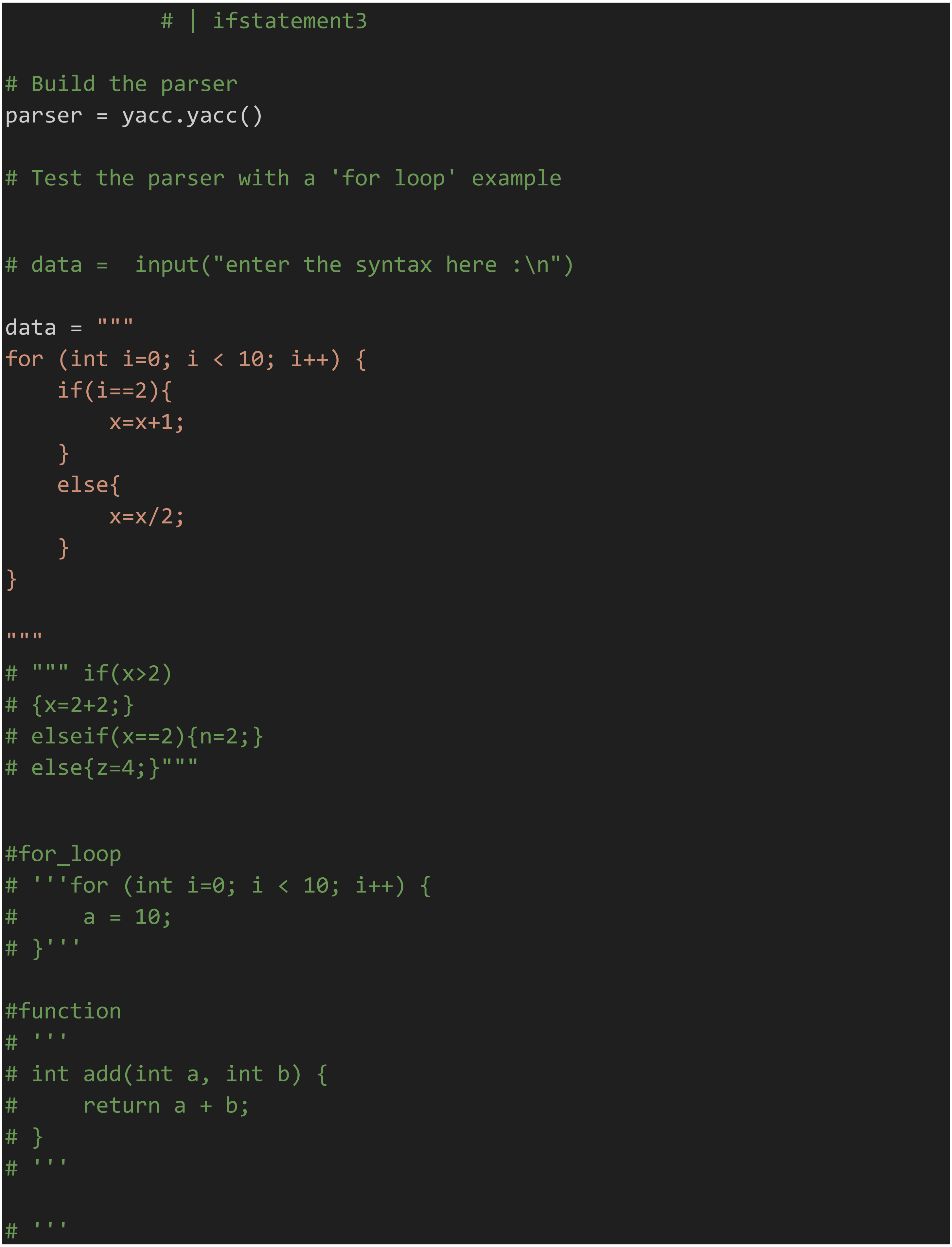
```
def p_ifstatement4(p):
    '''ifstatement4 : IF LPAREN expr4 RPAREN LBRACE statements4 RBRACE
ifelse4
          | IF expr4 LBRACE statements4 RBRACE'''
    if len(p) == 8:
        p[0] = ('if4', p[3], p[6], p[7])
    else:
        p[0] = ('if4', p[2], p[4])

def p_ifelse4(p):
    """ifelse4 : ELSEIF LPAREN expr4 RPAREN LBRACE statements4 RBRACE
ifelse4
          | ELSE LBRACE statements4 RBRACE
          | empty"""
    if len(p) == 9:
        p[0] = ('else-if4', p[3], p[6], p[8])
    elif len(p) == 6:
        p[0] = ('else4', p[3])
    else:
        p[0] = []  # Empty

def p_statements4(p):
    '''statements4 : statements4 statements4 SEMICOLON
          | expr4
          | empty'''
    if len(p) == 4:
        p[0] = p[1] + [p[3]]
    else:
        p[0] = [p[1]]

def p_expr4(p):
    '''expr4 : expr4 OPERATOR OPERATOR expr4
          | expr4 OPERATOR expr4
          | ID
          | NUMBER
          '''
    if len(p) == 6:
        p[0] = (p[3], p[2], p[4])
    elif len(p) == 4:
        p[0] = (p[1], p[2], p[3])
```



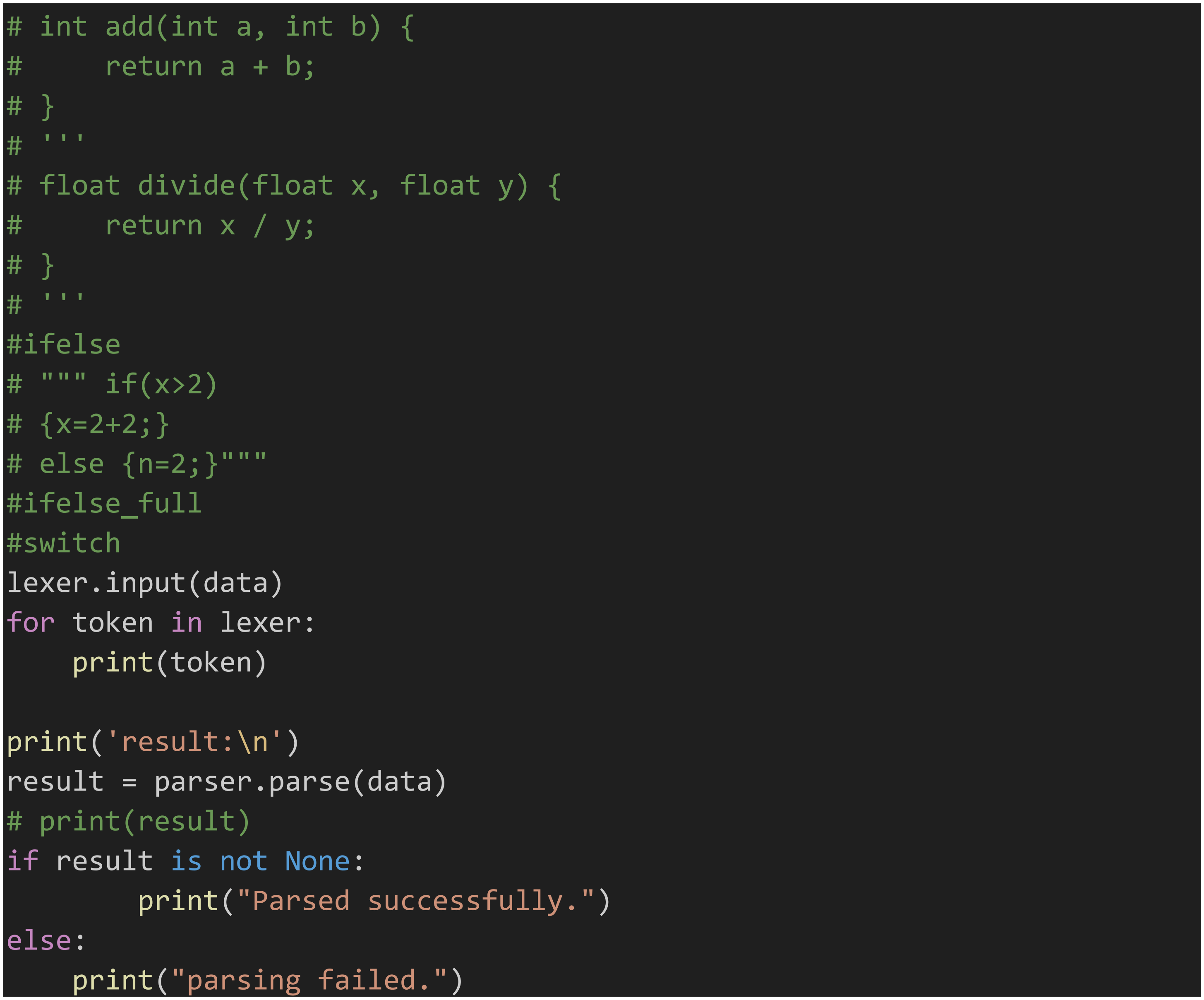
```
    else:
        p[0] = p[1]

########################################################################
#####################################
#switch



def p_switch_statement5(p):
    '''
    switch_statement5 : SWITCH use_ornot5 LBRACE case_list5 DEFAULT
COLON statement_list5 RBRACE
    '''
    # Do something with the parsed result if needed
    p[0] = ("switch_statement5", p[2], p[4], p[7])  # Example: saving
relevant information

def p_use_ornot5(p):
    '''use_ornot5 : LPAREN ID RPAREN
              | ID '''

def p_case_list5(p):
    '''
    case_list5 : case_entry5 case_list5
              | empty
    '''
    # Do something with the parsed result if needed
    if len(p) == 3:
        p[0] = [p[1]] + p[2]
    else:
        p[0] = []

def p_case_entry5(p):
    '''
    case_entry5 : CASE NUMBER COLON statement_list5
    '''
    # Do something with the parsed result if needed
    p[0] = ("case_entry5", p[2], p[4])  # Example: saving relevant
information
```



```python
def p_statement_list5(p):
    '''
    statement_list5 : statement5 SEMICOLON statement_list5
                    | empty
    '''
    # Do something with the parsed result if needed
    if len(p) == 4:
        p[0] = [p[1]] + p[3]
    else:
        p[0] = []

def p_statement5(p):
    '''
    statement5 : ID
               | BREAK
    '''
    # Do something with the parsed result if needed
    p[0] = ("statement5", p[1])  # Example: saving relevant information


def p_empty(p):
    'empty :'
    pass

def p_error(p):
    if p:
        print(f"Syntax error at {p}")
    else:
        print("Syntax error at EOF")

def p_initial(p) :
    '''
    initial : statement1
            | function2
            | ifstatement4
            | switch_statement5
    '''
    p[0] = p[1]
```



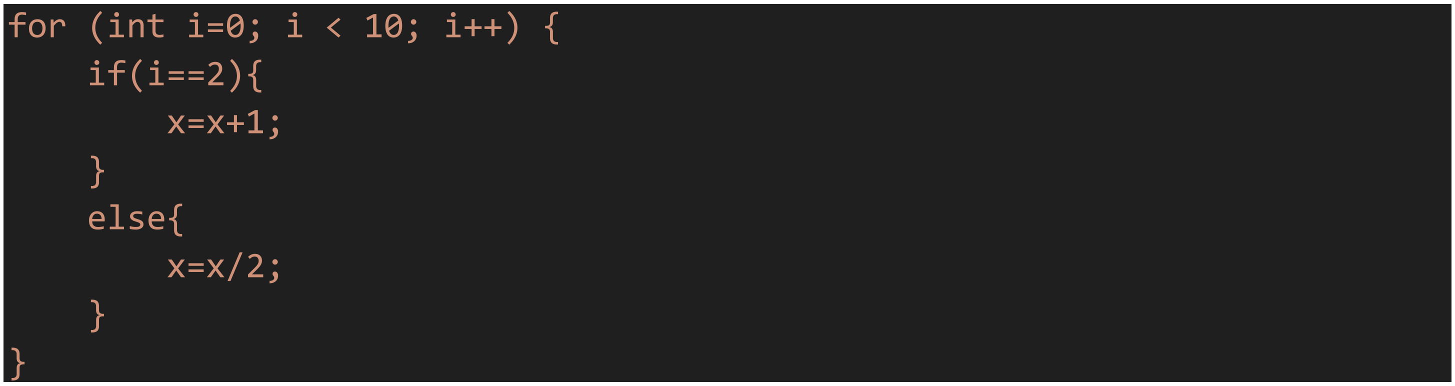
```
            # | ifstatement3

# Build the parser
parser = yacc.yacc()

# Test the parser with a 'for loop' example


# data =  input("enter the syntax here :\n")

data = """
for (int i=0; i < 10; i++) {
    if(i==2){
        x=x+1;
    }
    else{
        x=x/2;
    }
}

"""
# """ if(x>2)
# {x=2+2;}
# elseif(x==2){n=2;}
# else{z=4;}"""


#for_loop
# '''for (int i=0; i < 10; i++) {
#     a = 10;
# }'''

#function
# '''
# int add(int a, int b) {
#     return a + b;
# }
# '''

# '''
```



```
# int add(int a, int b) {
#     return a + b;
# }
# '''
# float divide(float x, float y) {
#     return x / y;
# }
# '''
#ifelse
# """ if(x>2)
# {x=2+2;}
# else {n=2;}"""
#ifelse_full
#switch
lexer.input(data)
for token in lexer:
    print(token)

print('result:\n')
result = parser.parse(data)
# print(result)
if result is not None:
        print("Parsed successfully.")
else:
    print("parsing failed.")
```

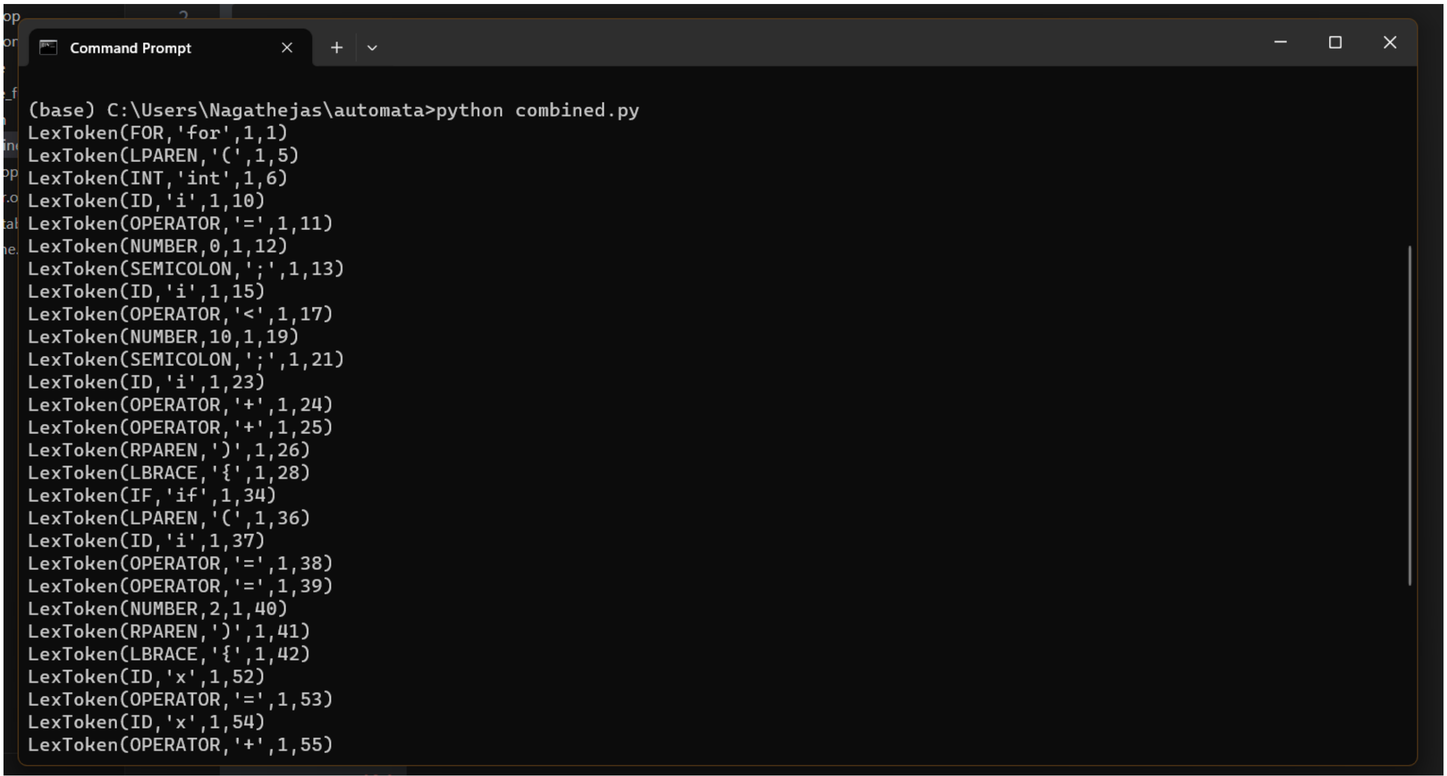
Output screenshot –

Input –

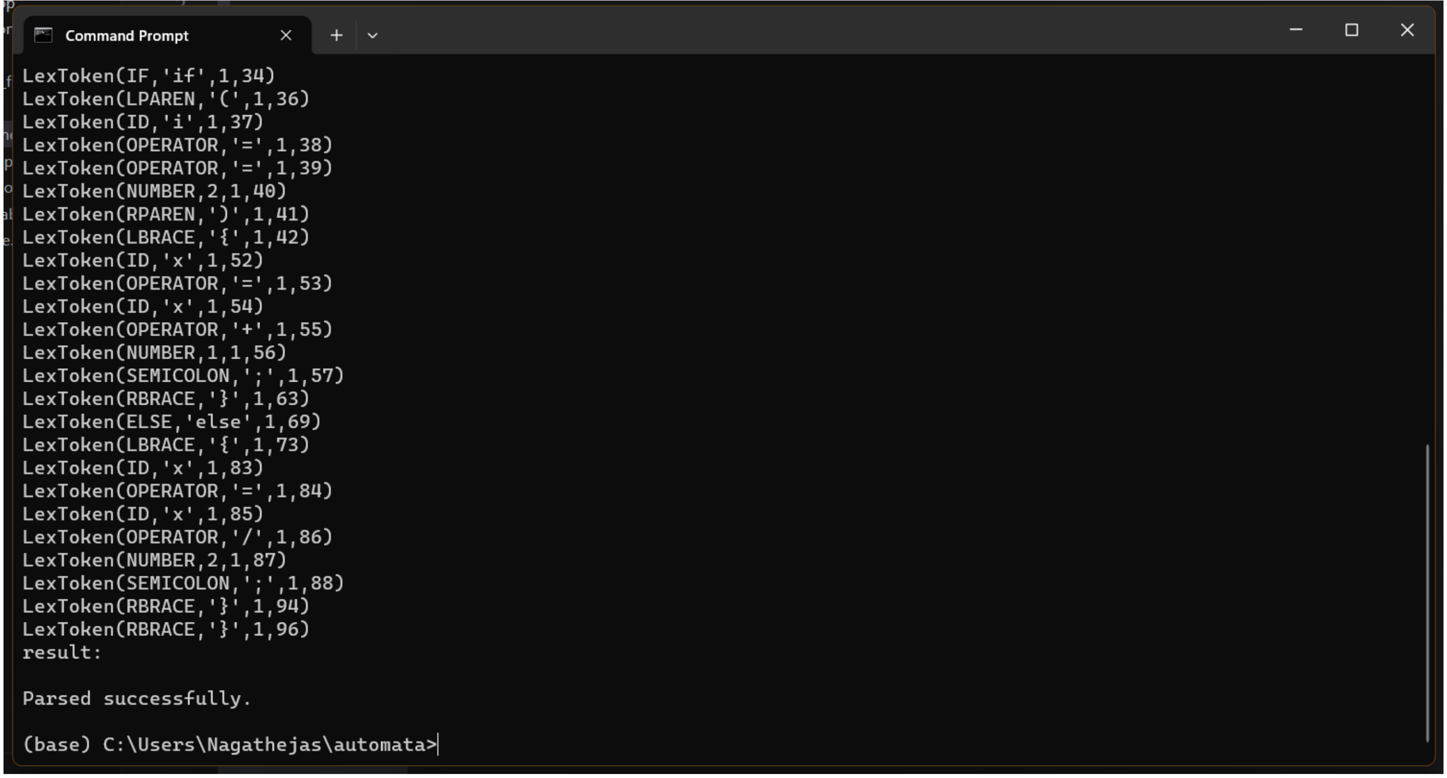


```
for (int i=0; i < 10; i++) {
    if(i==2){
        x=x+1;
    }
    else{
        x=x/2;
    }
}
```

Console-



```
(base) C:\Users\Nagathejas\automata>python combined.py
LexToken(FOR,'for',1,1)
LexToken(LPAREN,'(',1,5)
LexToken(INT,'int',1,6)
LexToken(ID,'i',1,10)
LexToken(OPERATOR,'=',1,11)
LexToken(NUMBER,0,1,12)
LexToken(SEMICOLON,';',1,13)
LexToken(ID,'i',1,15)
LexToken(OPERATOR,'<',1,17)
LexToken(NUMBER,10,1,19)
LexToken(SEMICOLON,';',1,21)
LexToken(ID,'i',1,23)
LexToken(OPERATOR,'+',1,24)
LexToken(OPERATOR,'+',1,25)
LexToken(RPAREN,')',1,26)
LexToken(LBRACE,'{',1,28)
LexToken(IF,'if',1,34)
LexToken(LPAREN,'(',1,36)
LexToken(ID,'i',1,37)
LexToken(OPERATOR,'=',1,38)
LexToken(OPERATOR,'=',1,39)
LexToken(NUMBER,2,1,40)
LexToken(RPAREN,')',1,41)
LexToken(LBRACE,'{',1,42)
LexToken(ID,'x',1,52)
LexToken(OPERATOR,'=',1,53)
LexToken(ID,'x',1,54)
LexToken(OPERATOR,'+',1,55)
```



```
LexToken(IF,'if',1,34)
LexToken(LPAREN,'(',1,36)
LexToken(ID,'i',1,37)
LexToken(OPERATOR,'=',1,38)
LexToken(OPERATOR,'=',1,39)
LexToken(NUMBER,2,1,40)
LexToken(RPAREN,')',1,41)
LexToken(LBRACE,'{',1,42)
LexToken(ID,'x',1,52)
LexToken(OPERATOR,'=',1,53)
LexToken(ID,'x',1,54)
LexToken(OPERATOR,'+',1,55)
LexToken(NUMBER,1,1,56)
LexToken(SEMICOLON,';',1,57)
LexToken(RBRACE,'}',1,63)
LexToken(ELSE,'else',1,69)
LexToken(LBRACE,'{',1,73)
LexToken(ID,'x',1,83)
LexToken(OPERATOR,'=',1,84)
LexToken(ID,'x',1,85)
LexToken(OPERATOR,'/',1,86)
LexToken(NUMBER,2,1,87)
LexToken(SEMICOLON,';',1,88)
LexToken(RBRACE,'}',1,94)
LexToken(RBRACE,'}',1,96)
result:

Parsed successfully.

(base) C:\Users\Nagathejas\automata>
```