PetFox Parser

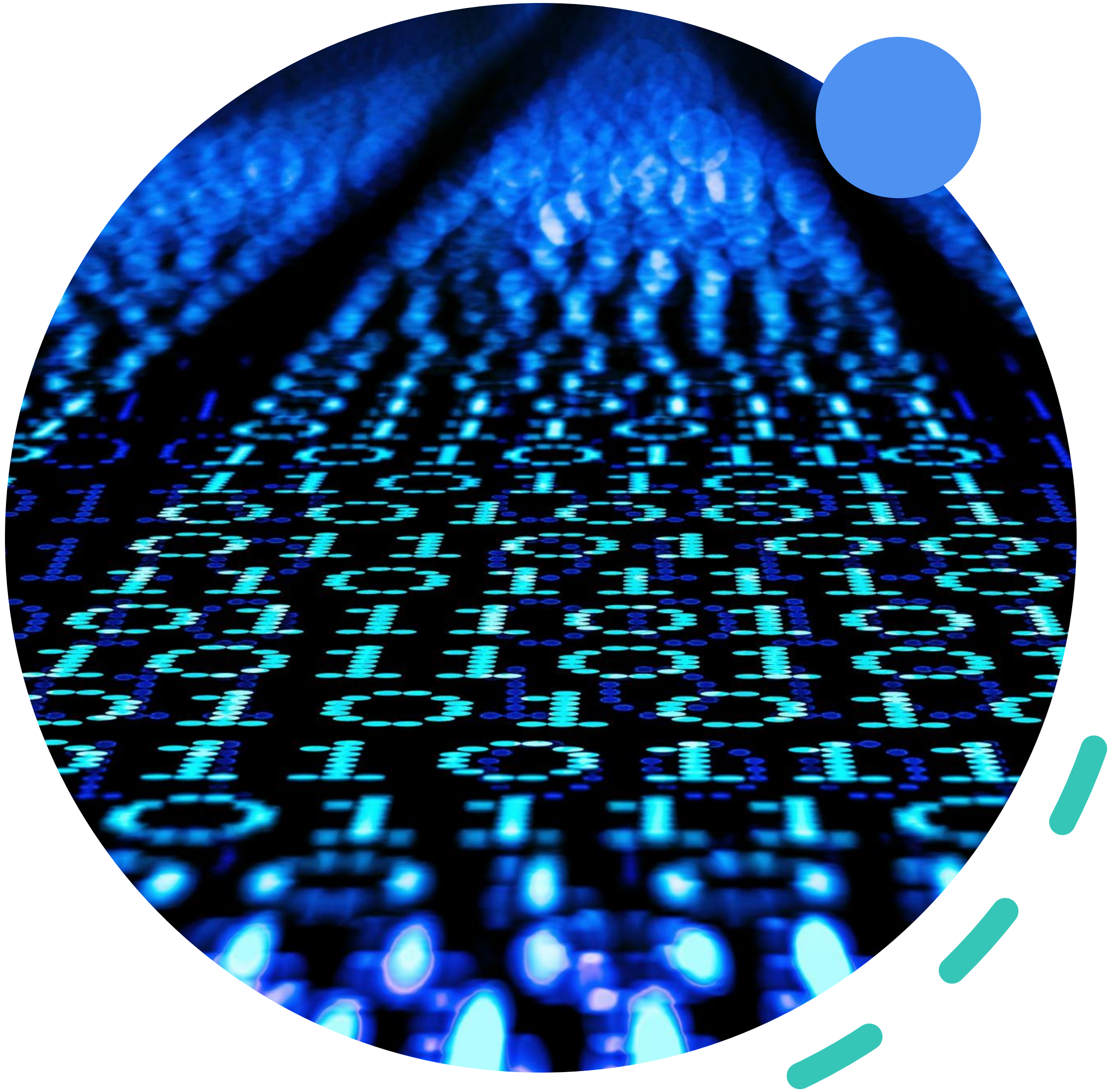
Created by: Alexander Fox & Nolan Pettit

Update on the Language

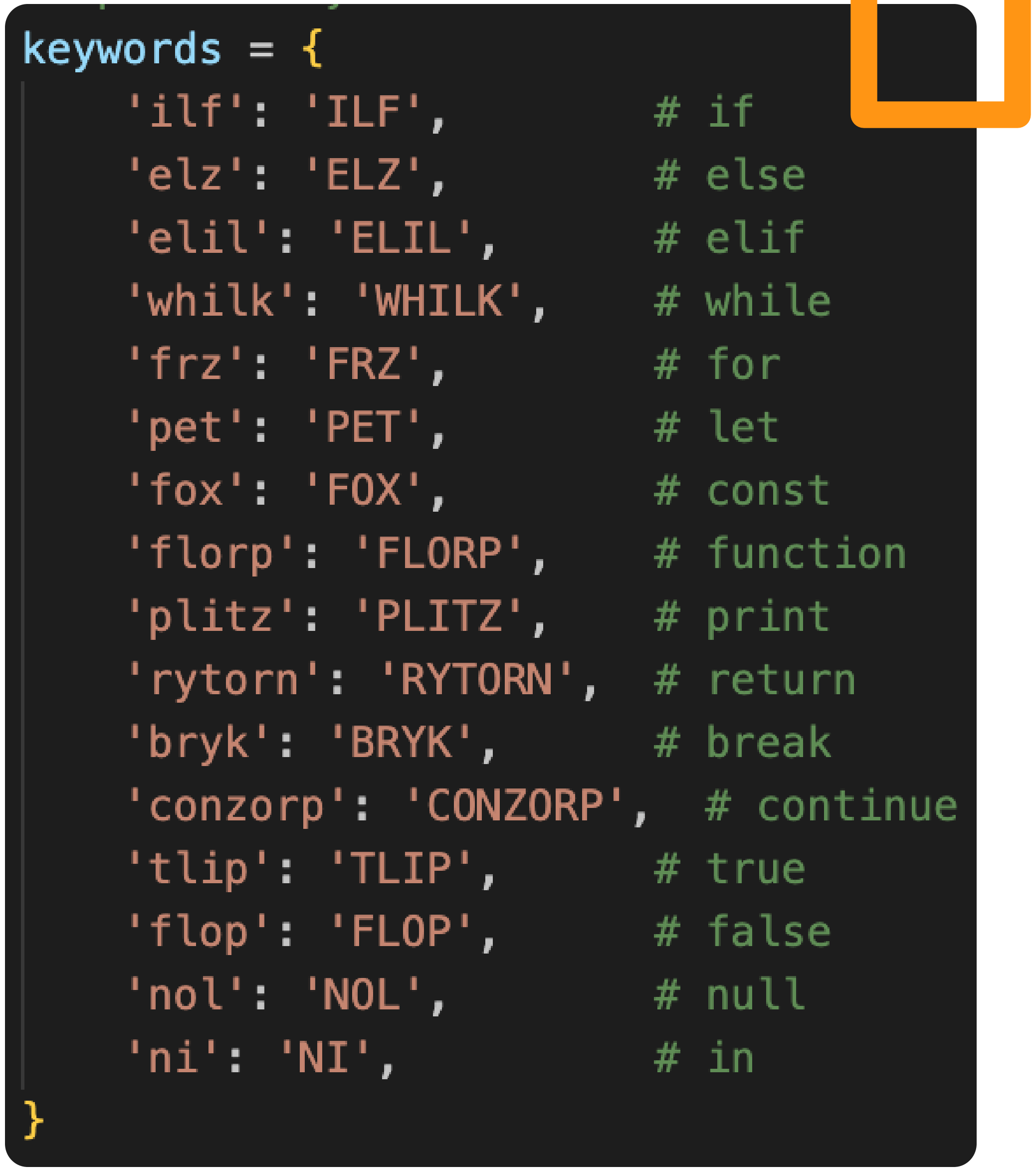
- Added more tokens and keywords to allow for a more developed programming language

- Implemented a few more "Martian" terminologies

- Developed a parser to create an Abstract Syntax Tree



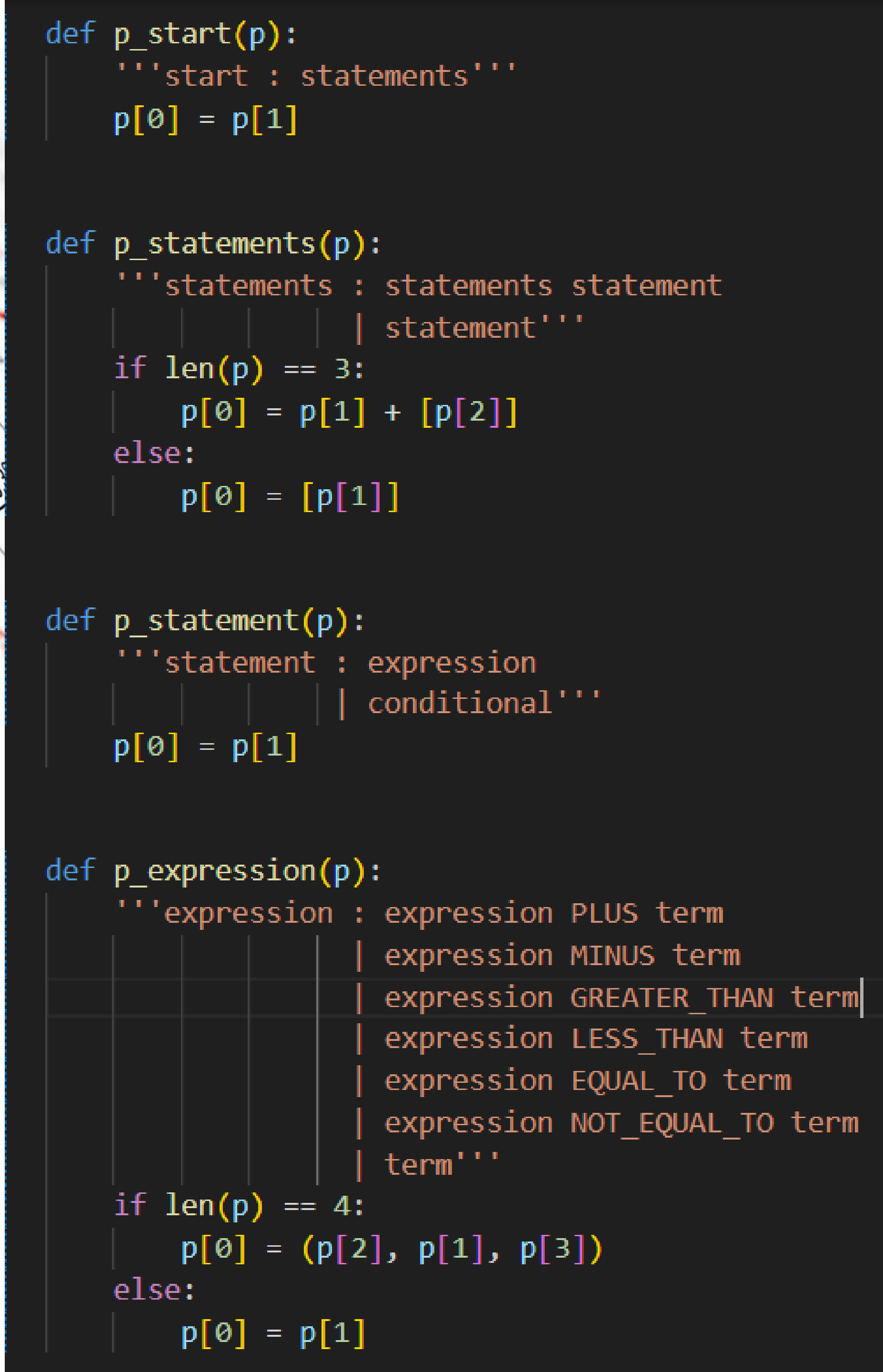
Additional Keywords



```
keywords = {
    'ilf': 'ILF',            # if
    'elz': 'ELZ',            # else
    'elil': 'ELIL',          # elif
    'whilk': 'WHILK',        # while
    'frz': 'FRZ',            # for
    'pet': 'PET',            # let
    'fox': 'FOX',            # const
    'florp': 'FLORP',        # function
    'plitz': 'PLITZ',        # print
    'rytorn': 'RYTORN',      # return
    'bryk': 'BRYK',          # break
    'conzorp': 'CONZORP',  # continue
    'tlip': 'TLIP',          # true
    'flop': 'FLOP',          # false
    'nol': 'NOL',            # null
    'ni': 'NI',              # in
}
```

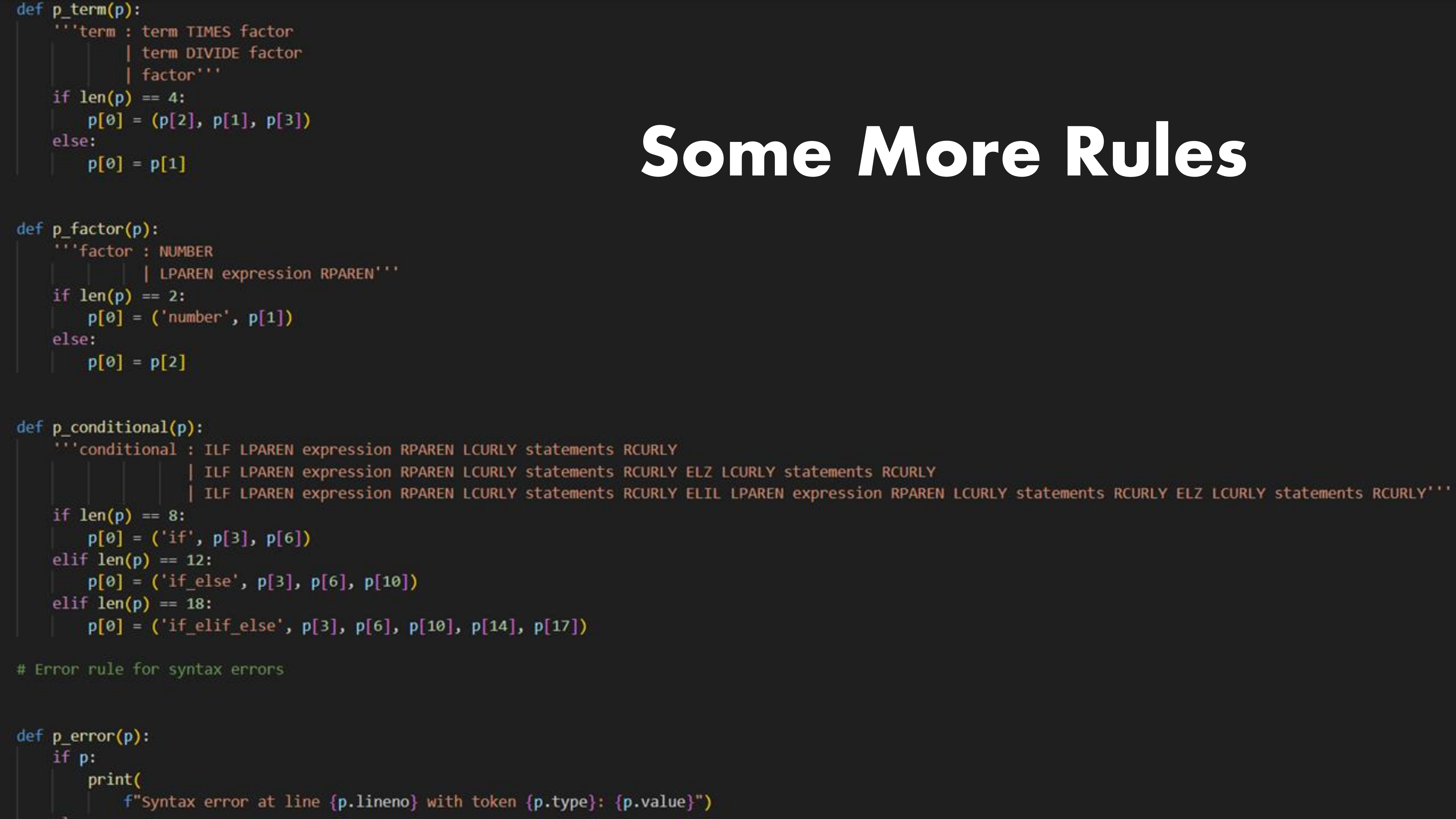
Grammar Rules

- **P_start ->** starting point of grammar
- **P_statements ->** defines the construction of a series of statements
- **P_statement ->** defines what makes up a single statement
- **P_expression ->** handles binary operations through tuples



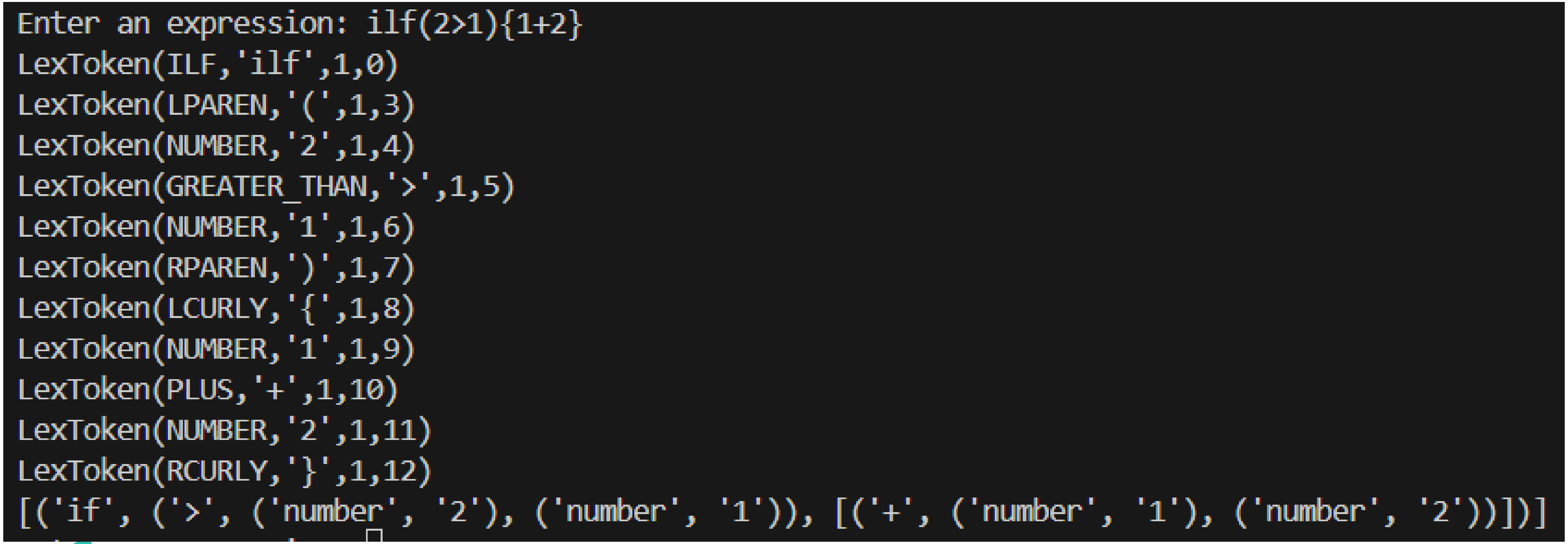
```python
def p_start(p):
    '''start : statements'''
    p[0] = p[1]


def p_statements(p):
    '''statements : statements statement
                  | statement'''
    if len(p) == 3:
        p[0] = p[1] + [p[2]]
    else:
        p[0] = [p[1]]


def p_statement(p):
    '''statement : expression
                 | conditional'''
    p[0] = p[1]


def p_expression(p):
    '''expression : expression PLUS term
                  | expression MINUS term
                  | expression GREATER_THAN term
                  | expression LESS_THAN term
                  | expression EQUAL_TO term
                  | expression NOT_EQUAL_TO term
                  | term'''
    if len(p) == 4:
        p[0] = (p[2], p[1], p[3])
    else:
        p[0] = p[1]
```

Some More Rules



```
def p_term(p):
    '''term : term TIMES factor
            | term DIVIDE factor
            | factor'''
    if len(p) == 4:
        p[0] = (p[2], p[1], p[3])
    else:
        p[0] = p[1]


def p_factor(p):
    '''factor : NUMBER
              | LPAREN expression RPAREN'''
    if len(p) == 2:
        p[0] = ('number', p[1])
    else:
        p[0] = p[2]


def p_conditional(p):
    '''conditional : ILF LPAREN expression RPAREN LCURLY statements RCURLY
                   | ILF LPAREN expression RPAREN LCURLY statements RCURLY ELZ LCURLY statements RCURLY
                   | ILF LPAREN expression RPAREN LCURLY statements RCURLY ELIL LPAREN expression RPAREN LCURLY statements RCURLY ELZ LCURLY statements RCURLY'''
    if len(p) == 8:
        p[0] = ('if', p[3], p[6])
    elif len(p) == 12:
        p[0] = ('if_else', p[3], p[6], p[10])
    elif len(p) == 18:
        p[0] = ('if_elif_else', p[3], p[6], p[10], p[14], p[17])

# Error rule for syntax errors


def p_error(p):
    if p:
        print(
            f"Syntax error at line {p.lineno} with token {p.type}: {p.value}")
```

Example Usage



```
Enter an expression: ilf(2>1){1+2}
LexToken(ILF,'ilf',1,0)
LexToken(LPAREN,'(',1,3)
LexToken(NUMBER,'2',1,4)
LexToken(GREATER_THAN,'>',1,5)
LexToken(NUMBER,'1',1,6)
LexToken(RPAREN,')',1,7)
LexToken(LCURLY,'{',1,8)
LexToken(NUMBER,'1',1,9)
LexToken(PLUS,'+',1,10)
LexToken(NUMBER,'2',1,11)
LexToken(RCURLY,'}',1,12)
[('if', ('>', ('number', '2'), ('number', '1')), [('+', ('number', '1'), ('number', '2'))])]
```