AUTOMATA FORMAL LANGUAGES & LOGIC

PROJECT

Topic: Ruby Parser for IF, ELSE, FOR, WHILE & DO WHILE Constructs

IMPORTS

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

TOKENS

```
# Tokenisation of keywords used in the program
tokens = (
    'IF', 'ELSE',
    'FOR', 'IN', 'DO', 'RANGE',
    'WHILE',
    'LOOP', 'BEGIN',
    'END',
    'LPAREN', 'RPAREN', 'LBRACE', 'RBRACE', 'SEMICOLON', 'SINGLEQUOTE',
'DOUBLEQUOTES',
    'IDENTIFIER', 'COMMA', 'STRING', 'NUMBER',
    'EQUALS', 'EQUALS_EQUALS', 'GREATERTHAN', 'LESSTHAN', 'NOT_EQUAL',
'LESSEQUAL', 'GREATEQUAL',
    'PLUS', 'MINUS', 'TIMES', 'DIVIDE',
)

# Defining the tokens to relate each input to a particular token
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_LBRACE = r'\{'
t_RBRACE = r'\}'
t_SEMICOLON = r';'
t_SINGLEQUOTE = r'\''
t_DOUBLEQUOTES = r'"'
t_NUMBER = r'\d+'
t_EQUALS = r'='
t_EQUALS_EQUALS = r'=='
t_GREATERTHAN = r'>'
t_LESSTHAN = r'<'
```

```
t_NOT_EQUAL = r'!='
t_LESSEQUAL = r'<='
t_GREATEQUAL = r'>='
t_PLUS = r'\+'
t_MINUS = r'-'
t_TIMES = r'\*'
t_DIVIDE = r'/'
t_ignore = ' \t'

#using reserved so that these keyword dont fall under the identifier
definition
reserved = {
    'if': 'IF',
    'end': 'END',
    'puts': 'IDENTIFIER',
    'else': 'ELSE',
    'print': 'IDENTIFIER',
    'for': 'FOR',
    'in': 'IN',
    'do': 'DO',
    'range': 'RANGE',
    'while': 'WHILE',
    'loop': 'LOOP',
    'begin': 'BEGIN',
}

def t_IDENTIFIER(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
    t.type = reserved.get(t.value, 'IDENTIFIER')
    return t

#defining strings to be accepted without quotes
def t_STRING(t):
    r'"([^"\\]|\\.|"")*"'
    t.value = t.value[1:-1]
    return t

def t_COMMA(t):
    r','
    return t

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

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

GRAMMAR RULES

```
def p_error(p):
    if p:
        print(f"Syntax error at line {p.lineno}, position {p.lexpos}:
Unexpected token '{p.value}'")
    else:
        print("Syntax error at EOF")

#initialising a parser
parser = yacc.yacc()

while True:
    try:
        s = input('Ruby-code > ')
    except EOFError:
        break
    if not s:
        continue
    result = parser.parse(s)
    print(result)
```

IF CONSTRUCT

```
GRAMMAR

all_constructs: If (condition) {statements} end

condition: < | > | == | <= | >= | !=

statements: statement statements
          | statement

statement: all_constructs
         | functions

functions: print
         | puts

IF CONSTRUCT GRAMMAR RULES

def p_all_constructs(p):
    '''
    all_constructs : IF LPAREN condition RPAREN LBRACE statements RBRACE END
                   | IF LPAREN condition RPAREN LBRACE if_statement RBRACE END
    '''
    p[0] = 'Valid Ruby statement'
```

```
def p_if_statement(p):
    '''
    if_statement : IF LPAREN condition RPAREN LBRACE statements RBRACE END
    '''
    pass

def p_condition(p):
    '''
    condition : expression GREATERTHAN expression
              | expression LESSTHAN expression
              | expression EQUALS_EQUALS expression
              | expression PLUS expression
              | expression MINUS expression
              | expression TIMES expression
              | expression DIVIDE expression
              | expression LESSEQUAL expression
              | expression GREATEQUAL expression
              | expression NOT_EQUAL expression
              | NUMBER
              | expression EQUALS expression
    '''
    pass

def p_expression(p):
    '''
    expression : IDENTIFIER
               | NUMBER
               | STRING
               | function_call
    '''
    pass

def p_statements(p):
    '''
    statements : statement
               | statements statement
    '''
      pass

def p_statement(p):
    '''
    statement : function_call
              | all_constructs
              | condition
    '''
    Pass
```

```
def p_function_call(p):
    '''
    function_call : IDENTIFIER LPAREN arguments RPAREN SEMICOLON
                  | IDENTIFIER arguments SEMICOLON
                  | IDENTIFIER LPAREN arguments RPAREN
                  | IDENTIFIER arguments
                  | IDENTIFIER SINGLEQUOTE arguments SINGLEQUOTE
                  | IDENTIFIER DOUBLEQUOTES arguments DOUBLEQUOTES
    '''
    pass

def p_arguments(p):
    '''
    arguments :
              | expression
              | arguments COMMA expression
    '''
    pass
```

VALID INPUTS

```
Ruby-code > if (x>3) {puts "x greater than 3"} end
Valid Ruby statement
Ruby-code > if (x!=6) {if (x!=5) {print ("x is not 5 or 6")} end } end
Valid Ruby statement
```

INVALID INPUTS

```
Ruby-code > if (x>3) {puts "x greater than 3"}
Syntax error at EOF
None
Ruby-code > if (x!=6) {if (x!=5) {print ("x is not 5 or 6)} end } end
Syntax error at line 1, position 29: Unexpected token '"'
None
```

ELSE CONSTRUCT

```
GRAMMAR

all_constructs: If (condition) {statements} else_statement end

else_statement: else {statements}

condition: < | > | == | <= | >= | !=

statements: statement statements
          | statement

statement: all_constructs
         | functions

functions: print
         | puts

ELSE CONSTRUCT GRAMMAR RULES
def p_all_constructs(p):
    '''
    all_constructs : IF LPAREN condition RPAREN LBRACE statements RBRACE
else_statement END
    '''
    p[0] = 'Valid Ruby statement'

def p_else_statement(p):
    '''
    else_statement : ELSE LBRACE statements RBRACE
    '''
    pass

def p_condition(p):
    '''
    condition : expression GREATERTHAN expression
              | expression LESSTHAN expression
              | expression EQUALS_EQUALS expression
              | expression PLUS expression
              | expression MINUS expression
              | expression TIMES expression
              | expression DIVIDE expression
              | expression LESSEQUAL expression
              | expression GREATEQUAL expression
              | expression NOT_EQUAL expression
              | NUMBER
    '''
    pass
```

```
def p_expression(p):
    '''
    expression : IDENTIFIER
               | NUMBER
               | STRING
               | function_call
    '''
    pass

def p_statements(p):
    '''
    statements : statement
               | statements statement
    '''
    pass

def p_statement(p):
    '''
    statement : function_call
              | all_constructs
    '''
    pass

def p_function_call(p):
    '''
    function_call : IDENTIFIER LPAREN arguments RPAREN SEMICOLON
                  | IDENTIFIER arguments SEMICOLON
                  | IDENTIFIER LPAREN arguments RPAREN
                  | IDENTIFIER arguments
    '''
    pass

def p_arguments(p):
    '''
    arguments :
              | expression
              | arguments COMMA expression
    '''
    pass
```

VALID INPUTS

```
Ruby-code > if (x != 5) { if (x != 6) {puts "not 5 or 6"} else {print "it is 6"} end } else {puts "it is 5"} end
Valid Ruby statement
Ruby-code > if (x != 5) { print "it is not 5"} else {puts "it is 5"} end
Valid Ruby statement
```

INVALID INPUTS

```
Ruby-code > if (x != 5) { if (x != 6) {puts "not 5 or 6"} else {print "it is 6"} } else {puts "it is 5"} end
Syntax error at line 1, position 69: Unexpected token '}'
None
Ruby-code > if (x != 5) { print "it is not 5"} end else {puts "it is 5"}
Syntax error at line 1, position 39: Unexpected token 'else'
None
```

WHILE CONSTRUCT

```
GRAMMAR

all_constructs: while (condition) do {statements} end
              | while (condition) {statements} end

condition: < | > | == | <= | >= | !=

statements: statement statements
          | statement

statement: all_constructs
         | functions

functions: print
         | puts

WHILE CONSTRUCT GRAMMAR RULES
def p_all_constructs(p):
    '''
    all_constructs : WHILE LPAREN condition RPAREN LBRACE statements RBRACE
END
                   | WHILE LPAREN condition RPAREN DO LBRACE statements RBRACE
END
    '''
    p[0] = 'Valid Ruby statement'

def p_condition(p):
    '''
    condition : expression GREATERTHAN expression
              | expression LESSTHAN expression
              | expression EQUALS_EQUALS expression
              | expression PLUS expression
              | expression MINUS expression
              | expression TIMES expression
              | expression DIVIDE expression
              | expression LESSEQUAL expression
              | expression GREATEQUAL expression
```

```
              | expression NOT_EQUAL expression
              | NUMBER
    '''
    pass

def p_expression(p):
    '''
    expression : IDENTIFIER
               | NUMBER
               | STRING
               | function_call
    '''
    pass

def p_statements(p):
    '''
    statements : statement
               | statements statement
    '''
    pass

def p_statement(p):
    '''
    statement : function_call
              | all_constructs
    '''
    pass

def p_function_call(p):
    '''
    function_call : IDENTIFIER LPAREN arguments RPAREN SEMICOLON
                  | IDENTIFIER arguments SEMICOLON
                  | IDENTIFIER LPAREN arguments RPAREN
                  | IDENTIFIER arguments
    '''
    pass

def p_arguments(p):
    '''
    arguments :
              | expression
              | arguments COMMA expression
    '''
    pass
```

VALID INPUTS

```
Ruby-code > while (x < 7) do {print ("iterating");} end
Valid Ruby statement
Ruby-code > while (x < 7) {print "iterating"; while (x != 6) do {puts "still iterating"} end} end
Valid Ruby statement
```

INVALID INPUTS

```
Ruby-code > while ( < 7) {print ("iterating");} end
Syntax error at line 1, position 8: Unexpected token '<'
None
Ruby-code > while (x < 7) {print "iterating"; while (x != 6) do {puts "still iterating"}} end
Syntax error at line 1, position 76: Unexpected token '}'
None
```

FOR CONSTRUCT

```
GRAMMAR

all_constructs: for arguments in range (condition) do {statements} end
              | for arguments in identifier do {statements} end
              | for arguments in range (condition) {statements} end
              | for arguments in identifier {statements} end

arguments: expression
         | arguments , expression

condition: < | > | == | <= | >= | !=

statements: statement statements
          | statement

statement: all_constructs
         | functions

functions: print
         | puts

FOR CONSTRUCT GRAMMAR RULES
def p_all_constructs(p):
    '''
    all_constructs : FOR arguments IN RANGE LPAREN condition RPAREN DO LBRACE
statements RBRACE END
                   | FOR arguments IN IDENTIFIER DO LBRACE statements RBRACE
END
                   | FOR arguments IN IDENTIFIER LBRACE statements RBRACE END
```

```
                  | FOR arguments IN RANGE LPAREN condition RPAREN LBRACE
statements RBRACE END
    '''
    p[0] = 'Valid Ruby statement'

def p_condition(p):
    '''
    condition : expression GREATERTHAN expression
              | expression LESSTHAN expression
              | expression EQUALS_EQUALS expression
              | expression PLUS expression
              | expression MINUS expression
              | expression TIMES expression
              | expression DIVIDE expression
              | expression LESSEQUAL expression
              | expression GREATEQUAL expression
              | expression NOT_EQUAL expression
              | NUMBER
    '''
    pass

def p_expression(p):
    '''
    expression : IDENTIFIER
               | NUMBER
               | STRING
               | function_call
    '''
    pass

def p_statements(p):
    '''
    statements : statement
               | statements statement
    '''
    pass

def p_statement(p):
    '''
    statement : function_call
              | all_constructs
    '''
    pass

def p_function_call(p):
    '''
    function_call : IDENTIFIER LPAREN arguments RPAREN SEMICOLON
                  | IDENTIFIER arguments SEMICOLON
```

```
                | IDENTIFIER LPAREN arguments RPAREN
                | IDENTIFIER arguments
    '''
    pass

def p_arguments(p):
    '''
    arguments :
              | expression
              | arguments COMMA expression
    '''
    pass
```

VALID INPUTS

```
Ruby-code > for i in range(10) do {puts "iterating"} end
Valid Ruby statement
Ruby-code > for i in range(15) {print "iterating"} end
Valid Ruby statement
Ruby-code > for a,b in numbers do {print "iterating"} end
Valid Ruby statement
Ruby-code > for a,b in numbers {puts ("iterating");} end
Valid Ruby statement
```

INVALID INPUTS

```
Ruby-code > for i in rane(10) do {puts "iterating"} end
Syntax error at line 1, position 13: Unexpected token '('
None
Ruby-code > for i range(15) {print "iterating"} end
Syntax error at line 1, position 6: Unexpected token 'range'
None
Ruby-code > for a,b in numbers do print "iterating"} end
Syntax error at line 1, position 22: Unexpected token 'print'
None
Ruby-code > for a,b in numbers {puts ("iterating");}
Syntax error at EOF
None
```

DO WHILE CONSTRUCT

```
GRAMMAR

all_constructs: begin {statements} end while_statement
              | loop do {statements} end

while_statement: while (condition)
               | while condition

condition: < | > | == | <= | >= | !=

statements: statement statements
          | statement

statement: all_constructs
         | functions

functions: print
         | puts

WHILE DO CONSTRUCT GRAMMAR RULES
def p_all_constructs(p):
    '''
    all_constructs : BEGIN LBRACE statements RBRACE END while_statement
                   | LOOP DO LBRACE statements RBRACE END
    '''
    p[0] = 'Valid Ruby statement'

def p_while_statement(p):
    '''
    while_statement : WHILE LPAREN condition RPAREN
                    | WHILE condition
    '''
    pass

def p_condition(p):
    '''
    condition : expression GREATERTHAN expression
              | expression LESSTHAN expression
              | expression EQUALS_EQUALS expression
              | expression PLUS expression
              | expression MINUS expression
              | expression TIMES expression
              | expression DIVIDE expression
              | expression LESSEQUAL expression
              | expression GREATEQUAL expression
              | expression NOT_EQUAL expression
```

```
              | NUMBER
    '''
    pass

def p_expression(p):
    '''
    expression : IDENTIFIER
               | NUMBER
               | STRING
               | function_call
    '''
    pass

def p_statements(p):
    '''
    statements : statement
               | statements statement
    '''
    pass

def p_statement(p):
    '''
    statement : function_call
              | all_constructs
    '''
    pass

def p_function_call(p):
    '''
    function_call : IDENTIFIER LPAREN arguments RPAREN SEMICOLON
                  | IDENTIFIER arguments SEMICOLON
                  | IDENTIFIER LPAREN arguments RPAREN
                  | IDENTIFIER arguments
    '''
    pass

def p_arguments(p):
    '''
    arguments :
              | expression
              | arguments COMMA expression
    '''
    pass
```

VALID INPUTS

```
Ruby-code > begin {print ("iterating");} end while(x<6)
Valid Ruby statement
Ruby-code > loop do { puts "hello"; break; } end
Valid Ruby statement
Ruby-code > begin {print ("iterating");} end while x<=6
Valid Ruby statement
```

INVALID INPUTS

```
Ruby-code > begin {print ("iterating");} end while(x<<6)
Syntax error at line 1, position 41: Unexpected token '<'
None
Ruby-code > begin {print ("iterating");}  while x<=6
Syntax error at line 1, position 30: Unexpected token 'while'
None
Ruby-code > loop  { puts "hello"; break; } end
Syntax error at line 1, position 6: Unexpected token '{'
None
```

OVERVIEW OF THE PROJECT

The Ruby Syntax Analyzer verifies the structure of Ruby-like code segments, detecting syntax errors in if-else, for, while, and do-while statements. It breaks down the code into tokens, checks constructs' syntax accuracy, and precisely identifies errors by their line number and position in the code. It supports nested if-else statements and validates loop structures. This analyser provides foundational syntax validation for Ruby-like code.