

AUTOMATA FORMAL LANGUAGES AND
LOGIC
(UE22CS343A)

ASSIGNMENT
SYNTAX VALIDATION IN PYTHON

AAKANKSH SEELIN(PES2UG22CS003)

ABHINAV SANKARSHANA DASU(PES2UG22CS017)

CONSTRUCT-1

FOR:

```
for.py > ...
1  import ply.lex as lex
2  import ply.yacc as yacc
3
4  # List of reserved words
5  reserved = {
6      'for': 'FOR',
7      'in': 'IN',
8      'range': 'RANGE',
9      'n': 'N',
10 }
11
12 # List of token names
13 tokens = ('FOR', 'ID', 'IN', 'RANGE', 'COLON', 'LPAREN', 'RPAREN', 'COMMA', 'NUMBER', 'N')
14
15 # Regular expression rules for simple tokens
16 t_FOR = r'for'
17 t_IN = r'in'
18 #t_ID = r'[a-zA-Z_][a-zA-Z_0-9]*'
19 t_COLON = r':'
20 t_RANGE = r'range'
21 t_LPAREN = r'\('
22 t_RPAREN = r'\)'
23 t_COMMA = r','
24 t_ignore = ' \t\n'
25
```

```

26 # Regular expression rule for identifiers
27 def t_ID(t):
28     r'[a-zA-Z_][a-zA-Z_0-9]*'
29     t.type = reserved.get(t.value, 'ID') # Check for reserved words
30     return t
31
32 # Regular expression rule for numbers
33 def t_NUMBER(t):
34     r'\d+'
35     t.value = int(t.value)
36     return t
37
38 # Error handling rule
39 def t_error(t):
40     print(f"Illegal character '{t.value[0]}'")
41     t.lexer.skip(1)
42
43 # Build the lexer
44 lexer = lex.lex()
45
46 # Parsing rules
47 def p_for_loop(p):
48     '''for_loop : FOR ID IN RANGE LPAREN NUMBER COMMA NUMBER RPAREN COLON
49     |             | FOR ID IN RANGE LPAREN NUMBER COMMA N RPAREN COLON'''
50     print("For loop syntax is correct.")

```

```

52 def p_error(p):
53     if p:
54         print(f"Syntax error at '{p.value}' on line {p.lineno}")
55     else:
56         print("Syntax error at EOF")
57
58 # Build the parser
59 parser = yacc.yacc()
60
61 # Test it out
62 data = input()
63
64 # Give the data to the lexer
65 lexer.input(data)
66
67 # Tokenize
68 while True:
69     tok = lexer.token()
70     if not tok:
71         break
72     print(tok)
73
74 # Parse the data
75 result = parser.parse(data)

```

CORRECT SYNTAX:

```
PS C:\Users\akaan\OneDrive\Desktop\AFLI> & C:/Python312/python.exe c:/Users/akaan/OneDrive/Desktop/AFLI/for.py
Generating LALR tables
for i in range(0,10):
LexToken(FOR,'for',1,0)
LexToken(ID,'i',1,4)
LexToken(IN,'in',1,6)
LexToken(RANGE,'range',1,9)
LexToken(LPAREN,'(',1,14)
LexToken(NUMBER,0,1,15)
LexToken(COMMA,',',1,16)
LexToken(NUMBER,10,1,17)
LexToken(RPAREN,')',1,19)
LexToken(COLON,':',1,20)
For loop syntax is correct.
```

INCORRECT SYNTAX:

```
PS C:\Users\akaan\OneDrive\Desktop\AFLI> & C:/Python312/python.exe c:/Users/akaan/OneDrive/Desktop/AFLI/for.py
for i in range(1,10)
LexToken(FOR,'for',1,0)
LexToken(ID,'i',1,4)
LexToken(IN,'in',1,6)
LexToken(RANGE,'range',1,9)
LexToken(LPAREN,'(',1,14)
LexToken(NUMBER,1,1,15)
LexToken(COMMA,',',1,16)
LexToken(NUMBER,10,1,17)
LexToken(RPAREN,')',1,19)
Syntax error at EOF
```

CONSTRUCT-2

IF:

```
if.py > lex_analyse
1  import ply.lex as lex
2
3  reserved = {
4      'if' : 'IF'
5  }
6  tokens = ('IF', 'ID', 'EQ', 'INT', 'COLON', 'INDENT', 'DEDENT', 'ASSIGN')
7
8  t_IF = r'if'
9  t_EQ = r'=='
10 t_COLON = r':'
11 t_ASSIGN = r'='
12
13 def t_ID(t):
14     r'[a-zA-Z_][a-zA-Z0-9_]*'
15     t.type = reserved.get(t.value, 'ID')    # Check for reserved words
16     return t
17
18 def t_INT(t):
19     r'\d+'
20     return t
21
22 t_ignore = ' \t'
23
```

```
24 def t_newline(t):
25     r'\n+'
26     t.lexer.lineno += len(t.value)
27
28 def t_error(t):
29     print(f"illegal character '{t.value[0]}'")
30     t.lexer.skip(1)
31
32 lexer = lex.lex()
33
34 def lex_analyse(text):
35     lexer.input(text)
36
37     while True:
38         token = lexer.token()
39         if not token:
40             break
41         print(token)
42
43 text=input()
44 lex_analyse(text)
45
46 import ply.yacc as yacc
```

```

46 import ply.yacc as yacc
47 def p_statement_if(p):
48     'statement : IF ID EQ INT COLON ID ASSIGN INT'
49     p[0] = ('if', p[2], p[4], p[6], p[8])
50 syntax_error = False
51 def p_error(p):
52     global syntax_error
53     syntax_error = True
54     print(f"Syntax error at '{p.value}'")
55
56 parser = yacc.yacc()
57
58 def parse(input):
59     global syntax_error
60     syntax_error = False
61     result = parser.parse(input, lexer=lexer)
62     if syntax_error:
63         print("Incorrect Syntax")
64     else:
65         print("Correct Syntax")
66     print(result)
67 parse(text)

```

CORRECT SYNTAX:

```

PS C:\Users\akaan\OneDrive\Desktop\AFLI> & C:/Python312/python.exe c:/Users/akaan/OneDrive/Desktop/AFLI/if.py
if x=5:y=4
LexToken(IF,'if',1,0)
LexToken(ID,'x',1,3)
LexToken(EQ,'=',1,4)
LexToken(INT,'5',1,6)
LexToken(COLON,':',1,7)
LexToken(ID,'y',1,8)
LexToken(ASSIGN,'=',1,9)
LexToken(INT,'4',1,10)
Correct Syntax
('if', 'x', '5', 'y', '4')

```

INCORRECT SYNTAX:

```

PS C:\Users\akaan\OneDrive\Desktop\AFLI> & C:/Python312/python.exe c:/Users/akaan/OneDrive/Desktop/AFLI/if.py
if x=5:y=5
LexToken(IF,'if',1,0)
LexToken(ID,'x',1,3)
LexToken(ASSIGN,'=',1,4)
LexToken(INT,'5',1,5)
LexToken(COLON,':',1,6)
LexToken(ID,'y',1,7)
LexToken(ASSIGN,'=',1,8)
LexToken(INT,'5',1,9)
Syntax error at '='
Incorrect Syntax

```

CONSTRUCT-3

IF-ELSE

```
if-else.py > lex_analyse
1  import ply.lex as lex
2
3  reserved = {
4      'if' : 'IF',
5      'else' : 'ELSE'
6  }
7  tokens = ('ID', 'EQ', 'INT', 'COLON', 'INDENT', 'DEDENT', 'ASSIGN')+tuple(reserved.values())
8
9  t_IF = r'if'
10 t_ELSE = r'else'
11 t_EQ = r'=='
12 t_COLON = r':'
13 t_ASSIGN = r'='
14
15 def t_ID(t):
16     r'[a-zA-Z_][a-zA-Z0-9_]*'
17     t.type = reserved.get(t.value, 'ID')    # Check for reserved words
18     return t
19
20 def t_INT(t):
21     r'\d+'
22     return t
23
```

```
24 t_ignore = ' \t'
25
26 def t_newline(t):
27     r'\n+'
28     t.lexer.lineno += len(t.value)
29
30 def t_error(t):
31     print(f"Illegal character '{t.value[0]}'")
32     t.lexer.skip(1)
33
34 lexer = lex.lex()
35
36 def lex_analyse(text):
37     lexer.input(text)
38
39     while True:
40         token = lexer.token()
41         if not token:
42             break
43         print(token)
```

```

46 text=input()
47 lex_analyse(text)
48 import ply.yacc as yacc
49 def p_statement_if(p):
50     'statement : IF ID EQ INT COLON ID ASSIGN INT'
51     p[0] = ('if', p[2], p[4], p[6], p[8])
52 def p_statement_if_else(p):
53     'statement : IF ID EQ INT COLON ID ASSIGN INT ELSE COLON ID ASSIGN INT'
54     p[0] = ('if-else', p[2], p[4], p[6], p[8], p[11], p[1])
55
56 syntax_error = False
57
58 def p_error(p):
59     global syntax_error
60     syntax_error = True
61     if p:
62         print(f"Syntax error at '{p.value}'")
63     else:
64         print("Syntax error at EOF")
65
66 parser = yacc.yacc()
67
68 def parse(input):

```

```

68     def parse(input):
69         global syntax_error
70         syntax_error = False
71         result = parser.parse(input, lexer=lexer)
72         if syntax_error:
73             print("Incorrect Syntax")
74         else:
75             print("Correct Syntax")
76         print(result)
77
78     parse(text)

```


CORRECT SYNTAX:

```
PS C:\Users\akaan\OneDrive\Desktop\AFLI> & C:/Python312/python.exe c:/Users/akaan/OneDrive/Desktop/AFLI/if-else.py
if x ==5: y=6 else:y=7
LexToken(IF,'if',1,0)
LexToken(ID,'x',1,3)
LexToken(EQ,'==',1,5)
LexToken(INT,'5',1,7)
LexToken(COLON,':',1,8)
LexToken(ID,'y',1,10)
LexToken(ASSIGN,'=',1,11)
LexToken(INT,'6',1,12)
LexToken(ELSE,'else',1,14)
LexToken(COLON,':',1,18)
LexToken(ID,'y',1,19)
LexToken(ASSIGN,'=',1,20)
LexToken(INT,'7',1,21)
WARNING: Token 'DEDENT' defined, but not used
WARNING: Token 'INDENT' defined, but not used
WARNING: There are 2 unused tokens
Generating LALR tables
Correct Syntax
('if-else', 'x', '5', 'y', '6', 'y', 'if')
```

INCORRECT SYNTAX:

```
if x= 5: y==6 else:y=7
LexToken(IF,'if',1,0)
LexToken(ID,'x',1,3)
LexToken(ASSIGN,'=',1,4)
LexToken(INT,'5',1,6)
LexToken(COLON,':',1,7)
LexToken(ID,'y',1,9)
LexToken(EQ,'==',1,10)
LexToken(INT,'6',1,12)
LexToken(ELSE,'else',1,14)
LexToken(COLON,':',1,18)
LexToken(ID,'y',1,19)
LexToken(ASSIGN,'=',1,20)
LexToken(INT,'7',1,21)
Syntax error at '='
Incorrect Syntax
```

CONSTRUCT-4

WHILE:

```
while.py > ...
1  import ply.lex as lex
2  import ply.yacc as yacc
3
4  # List of reserved words
5  reserved = {
6      'while': 'WHILE',
7      'n': 'N',
8  }
9
10 # List of token names
11 tokens = ('WHILE', 'ID', 'LT', 'GT', 'EQ', 'NUMBER', 'COLON', 'N')
12
13 # Regular expression rules for simple tokens
14 t_WHILE = r'while'
15 t_LT = r'<'
16 t_GT = r'>'
17 t_EQ = r'='
18 t_COLON = r':'
19 t_ignore = ' \t\n'
20
21 # Regular expression rule for identifiers
22 def t_ID(t):
23     r'[a-zA-Z_][a-zA-Z_0-9]*'
```

```

24     t.type = reserved.get(t.value, 'ID') # Check for reserved words
25     return t
26 # Regular expression rule for numbers
27 def t_NUMBER(t):
28     r'\d+'
29     t.value = int(t.value)
30     return t
31 # Error handling rule
32 def t_error(t):
33     print(f"Illegal character '{t.value[0]}'")
34     t.lexer.skip(1)
35 # Build the lexer
36 lexer = lex.lex()
37 # Parsing rules
38 def p_while_loop(p):
39     '''while_loop : WHILE ID LT NUMBER COLON
40                   | WHILE ID LT N COLON
41                   | WHILE ID GT NUMBER COLON
42                   | WHILE ID GT N COLON
43                   | WHILE ID EQ NUMBER COLON
44                   | WHILE ID EQ N COLON
45                   | WHILE NUMBER GT ID COLON
46                   | WHILE NUMBER LT ID COLON'''

```

```

46                   | WHILE NUMBER LT ID COLON'''
47     print("While loop syntax is correct.")
48 def p_error(p):
49     if p:
50         print(f"Syntax error at '{p.value}' on line {p.lineno}")
51     else:
52         print("Syntax error at EOF")
53 # Build the parser
54 parser = yacc.yacc()
55 # Test it out
56 data = input()
57 # Give the data to the lexer
58 lexer.input(data)
59 # Tokenize
60 while True:
61     tok = lexer.token()
62     if not tok:
63         break
64     print(tok)
65
66 # Parse the data
67 result = parser.parse(data)

```

CORRECT SYNTAX:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\akaan\OneDrive\Desktop\AFLI> & C:/Python312/python.exe c:/Users/akaan/OneDrive/Desktop/AFLI/while.py
Generating LALR tables
while x > 6:
LexToken(WHILE,'while',1,0)
LexToken(ID,'x',1,6)
LexToken(GT,'>',1,8)
LexToken(NUMBER,6,1,10)
LexToken(COLON,':',1,11)
while loop syntax is correct.
PS C:\Users\akaan\OneDrive\Desktop\AFLI>
```

INCORRECT SYNTAX:

```
PS C:\Users\akaan\OneDrive\Desktop\AFLI> & C:/Python312/python.exe c:/Users/akaan/OneDrive/Desktop/AFLI/while.py
while x>7
LexToken(WHILE,'while',1,0)
LexToken(ID,'x',1,6)
LexToken(GT,'>',1,7)
LexToken(NUMBER,7,1,8)
Syntax error at EOF
```

CONSTRUCT-5:

FUNCTION-DEFINITION

```
func-def.py > t_error
1  import ply.lex as lex
2  import ply.yacc as yacc
3  reserved = {
4      'def' : 'DEF',
5      'if'  : 'IF',
6      'else': 'ELSE'
7  }
8  # List of token names
9  tokens = (
10     'DEF',
11     'ID',
12     'LPAREN',
13     'RPAREN',
14     'COLON',
15     'COMMA'
16 )
17 # Regular expression rules for simple tokens
18 t_DEF = r'def'
19 t_LPAREN = r'\('
20 t_RPAREN = r'\)'
21 t_COLON = r':'
22 t_COMMA = r','
23 t_ignore = ' \t'
```

```

24 # A regular expression rule with some action code
25 def t_ID(t):
26     r'[a-zA-Z_][a-zA-Z_0-9]*'
27     t.type = reserved.get(t.value, 'ID')
28     return t
29 # Error handling rule
30 def t_error(t):
31     print(f"Illegal character '{t.value[0]}'")
32     t.lexer.skip(1)
33 def t_newline(t):
34     r'\n+'
35     t.lexer.lineno += len(t.value)
36
37 # Build the lexer
38 lexer = lex.lex()
39
40 # Parsing rules
41 def p_function_definition(p):
42     'function : DEF ID LPAREN parameter_list RPAREN COLON'
43     print("Function definition is syntactically correct.")
44
45 def p_parameter_list(p):
46     '''
47     parameter_list : parameter_list COMMA ID
48                   | ID
49                   | empty
50     '''
51
52 def p_empty(p):
53     'empty :'
54     pass
55
56 def p_error(p):
57     if p:
58         print(f"Syntax error at '{p.value}' on line {p.lineno}")
59     else:
60         print("Syntax error at EOF")
61
62 # Build the parser
63 parser = yacc.yacc()
64 # Test it out
65 data = input()
66 # Give the data to the lexer
67 lexer.input(data)
68 # Tokenize

```

```

69 while True:
70     tok = lexer.token()
71     if not tok:
72         break
73     print(tok)
74
75 # Parse the data
76 result = parser.parse(data)

```

CORRECT SYNTAX:

```

PS C:\Users\akaan\OneDrive\Desktop\AFLI> & C:/Python312/python.exe c:/Users/akaan/OneDrive/Desktop/AFLI/func-def.py
Generating LALR tables
def func(child1):
LexToken(DEF,'def',1,0)
LexToken(ID,'func',1,4)
LexToken(LPAREN,'(',1,8)
LexToken(ID,'child1',1,9)
LexToken(RPAREN,')',1,15)
LexToken(COLON,':',1,16)
Function definition is syntactically correct.

```

INCORRECT SYNTAX:

```

PS C:\Users\akaan\OneDrive\Desktop\AFLI> & C:/Python312/python.exe c:/Users/akaan/OneDrive/Desktop/AFLI/func-def.py
DEF func(child1):
LexToken(ID,'DEF',1,0)
LexToken(ID,'func',1,4)
LexToken(LPAREN,'(',1,8)
LexToken(ID,'child1',1,9)
LexToken(RPAREN,')',1,15)
LexToken(COLON,':',1,16)
Syntax error at 'DEF' on line 1

```