AUTOMATA FORMAL LANGUAGES AND LOGIC
(UE22CS343A)

ASSIGNMENT

SYNTAX VALIDATION IN PYTHON

AAKANKSH SEELIN(PES2UG22CS003)

ABHINAV SANKARSHANA DASU(PES2UG22CS017)

CONSTRUCT-1

FOR:

for.py > ...

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

# List of reserved words
reserved = {
    'for': 'FOR',
    'in': 'IN',
    'range': 'RANGE',
    'n': 'N',
}

# List of token names
tokens = ('FOR','ID','IN','RANGE','COLON','LPAREN','RPAREN','COMMA','NUMBER', 'N')

# Regular expression rules for simple tokens
t_FOR = r'for'
t_IN = r'in'
#t_ID = r'[a-zA-Z_][a-zA-Z_0-9]*'
t_COLON = r':'
t_RANGE = r'range'
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_COMMA = r','
t_ignore = ' \t\n'

```

```
# Regular expression rule for identifiers
def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = reserved.get(t.value, 'ID')  # Check for reserved words
    return t

# Regular expression rule for numbers
def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)
    return t

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

# Build the lexer
lexer = lex.lex()

# Parsing rules
def p_for_loop(p):
    '''for_loop : FOR ID IN RANGE LPAREN NUMBER COMMA NUMBER RPAREN COLON
                | FOR ID IN RANGE LPAREN NUMBER COMMA N RPAREN COLON'''
    print("For loop syntax is correct.")
```

```
def p_error(p):
    if p:
        print(f"Syntax error at '{p.value}' on line {p.lineno}")
    else:
        print("Syntax error at EOF")

# Build the parser
parser = yacc.yacc()

# Test it out
data = input()

# Give the data to the lexer
lexer.input(data)

# Tokenize
while True:
    tok = lexer.token()
    if not tok:
        break
    print(tok)

# Parse the data
result = parser.parse(data)
```

CORRECT SYNTAX:

```
PS C:\Users\aakan\OneDrive\Desktop\AFLL> & C:/Python312/python.exe c:/Users/aakan/OneDrive/Desktop/AFLL/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\aakan\OneDrive\Desktop\AFLL> & C:/Python312/python.exe c:/Users/aakan/OneDrive/Desktop/AFLL/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

```
import ply.lex as lex

reserved = {
    'if' : 'IF'
}
tokens = ('IF', 'ID', 'EQ', 'INT', 'COLON', 'INDENT', 'DEDENT','ASSIGN')

t_IF = r'if'
t_EQ = r'=='
t_COLON = r':'
t_ASSIGN = r'='

def t_ID(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
    t.type = reserved.get(t.value,'ID')    # Check for reserved words
    return t

def t_INT(t):
    r'\d+'
    return t

t_ignore = ' \t'

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

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

lexer = lex.lex()

def lex_analyse(text):
    lexer.input(text)

    while True:
        token = lexer.token()
        if not token:
            break
        print(token)

text=input()
lex_analyse(text)

import ply.yacc as yacc
```

```python
import ply.yacc as yacc
def p_statement_if(p):
    'statement : IF ID EQ INT COLON ID ASSIGN INT'
    p[0] = ('if', p[2], p[4], p[6], p[8])
syntax_error = False
def p_error(p):
    global syntax_error
    syntax_error = True
    print(f"Syntax error at '{p.value}'")

parser = yacc.yacc()

def parse(input):
    global syntax_error
    syntax_error = False
    result = parser.parse(input, lexer=lexer)
    if syntax_error:
        print("Incorrect Syntax")
    else:
        print("Correct Syntax")
    print(result)
parse(text)
```

CORRECT SYNTAX:

```
PS C:\Users\aakan\OneDrive\Desktop\AFLL> & C:/Python312/python.exe c:/Users/aakan/OneDrive/Desktop/AFLL/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\aakan\OneDrive\Desktop\AFLL> & C:/Python312/python.exe c:/Users/aakan/OneDrive/Desktop/AFLL/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

```python
import ply.lex as lex

reserved = {
    'if' : 'IF',
    'else' : 'ELSE'
}
tokens = ('ID', 'EQ', 'INT', 'COLON', 'INDENT', 'DEDENT','ASSIGN')+tuple(reserved.values())

t_IF = r'if'
t_ELSE = r'else'
t_EQ = r'=='
t_COLON = r':'
t_ASSIGN = r'='

def t_ID(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
    t.type = reserved.get(t.value,'ID')    # Check for reserved words
    return t

def t_INT(t):
    r'\d+'
    return t

```

```python
t_ignore = ' \t'

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

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

lexer = lex.lex()

def lex_analyse(text):
    lexer.input(text)

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

```
text=input()
lex_analyse(text)
import ply.yacc as yacc
def p_statement_if(p):
    'statement : IF ID EQ INT COLON ID ASSIGN INT'
    p[0] = ('if', p[2], p[4], p[6], p[8])
def p_statement_if_else(p):
    'statement : IF ID EQ INT COLON ID ASSIGN INT ELSE COLON ID ASSIGN INT'
    p[0] = ('if-else', p[2], p[4], p[6], p[8], p[11], p[1])

syntax_error = False

def p_error(p):
    global syntax_error
    syntax_error = True
    if p:
        print(f"Syntax error at '{p.value}'")
    else:
        print("Syntax error at EOF")

parser = yacc.yacc()

def parse(input):
```

```
def parse(input):
    global syntax_error
    syntax_error = False
    result = parser.parse(input, lexer=lexer)
    if syntax_error:
        print("Incorrect Syntax")
    else:
        print("Correct Syntax")
    print(result)

parse(text)
```

CORRECT SYNTAX:

```
PS C:\Users\aakan\OneDrive\Desktop\AFLL> & C:/Python312/python.exe c:/Users/aakan/OneDrive/Desktop/AFLL/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:

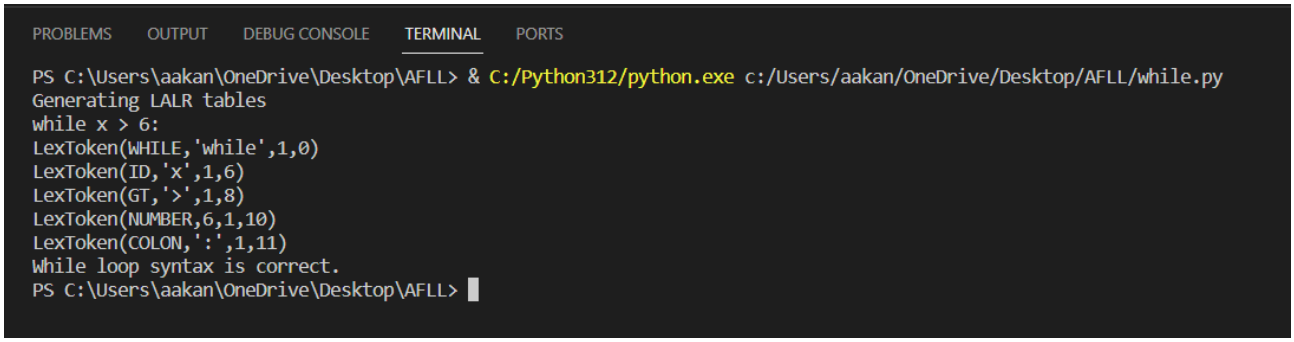
while.py > ...

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

# List of reserved words
reserved = {
    'while': 'WHILE',
    'n': 'N',
}

# List of token names
tokens = ('WHILE', 'ID', 'LT','GT','EQ', 'NUMBER', 'COLON', 'N')

# Regular expression rules for simple tokens
t_WHILE = r'while'
t_LT = r'<'
t_GT = r'>'
t_EQ = r'='
t_COLON = r':'
t_ignore = ' \t\n'

# Regular expression rule for identifiers
def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
```

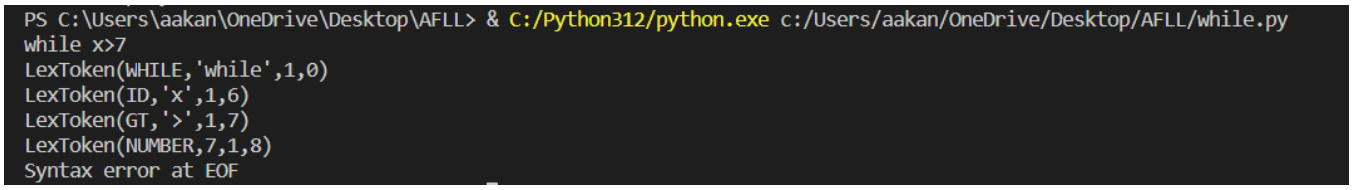
```
    t.type = reserved.get(t.value, 'ID')  # Check for reserved words
    return t
# Regular expression rule for numbers
def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)
    return t
# Error handling rule
def t_error(t):
    print(f"Illegal character '{t.value[0]}'")
    t.lexer.skip(1)
# Build the lexer
lexer = lex.lex()
# Parsing rules
def p_while_loop(p):
    '''while_loop : WHILE ID LT NUMBER COLON
                  | WHILE ID LT N COLON
                  | WHILE ID GT NUMBER COLON
                  | WHILE ID GT N COLON
                  | WHILE ID EQ NUMBER COLON
                  | WHILE ID EQ N COLON
                  | WHILE NUMBER GT ID COLON
                  | WHILE NUMBER LT ID COLON'''
```

```
                  | WHILE NUMBER LT ID COLON'''
    print("While loop syntax is correct.")
def p_error(p):
    if p:
        print(f"Syntax error at '{p.value}' on line {p.lineno}")
    else:
        print("Syntax error at EOF")
# Build the parser
parser = yacc.yacc()
# Test it out
data = input()
# Give the data to the lexer
lexer.input(data)
# Tokenize
while True:
    tok = lexer.token()
    if not tok:
        break
    print(tok)

# Parse the data
result = parser.parse(data)
```

CORRECT SYNTAX:



INCORRECT SYNTAX:



CONSTRUCT-5:

FUNCTION-DEFINITION

func-def.py > t_error

```
import ply.lex as lex
import ply.yacc as yacc
reserved = {
    'def' : 'DEF',
    'if'  : 'IF',
    'else': 'ELSE'
}
# List of token names
tokens = (
    'DEF',
    'ID',
    'LPAREN',
    'RPAREN',
    'COLON',
    'COMMA'
)
# Regular expression rules for simple tokens
t_DEF = r'def'
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_COLON = r':'
t_COMMA = r','
t_ignore = ' \t'
```

```
# A regular expression rule with some action code
def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = reserved.get(t.value,'ID')
    return t
# Error handling rule
def t_error(t):
    print(f"Illegal character '{t.value[0]}'")
    t.lexer.skip(1)
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

# Build the lexer
lexer = lex.lex()

# Parsing rules
def p_function_definition(p):
    'function : DEF ID LPAREN parameter_list RPAREN COLON'
    print("Function definition is syntactically correct.")

def p_parameter_list(p):
    '''
    parameter_list : parameter_list COMMA ID
                   | ID
                   | empty
    '''

def p_empty(p):
    'empty :'
    pass

def p_error(p):
    if p:
        print(f"Syntax error at '{p.value}' on line {p.lineno}")
    else:
        print("Syntax error at EOF")

# Build the parser
parser = yacc.yacc()
# Test it out
data = input()
# Give the data to the lexer
lexer.input(data)
# Tokenize
```

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

# Parse the data
result = parser.parse(data)
```

CORRECT SYNTAX:

```
PS C:\Users\aakan\OneDrive\Desktop\AFLL> & C:/Python312/python.exe c:/Users/aakan/OneDrive/Desktop/AFLL/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\aakan\OneDrive\Desktop\AFLL> & C:/Python312/python.exe c:/Users/aakan/OneDrive/Desktop/AFLL/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
```