**Language:** Python

**Constructs:** Lambda Function Definition, Dictionary Declaration, List Comprehension, Exception Handling (try-except), Generator Function with yield

|  |  |  |
| --- | --- | --- |
| **Name of the team members** | **Team members’ SRN** | **Section** |
| Rakesh R K | PES2UG23CS464 | H |
| Rahul Thushar Machikoppa | PES2UG23CS463 | H |

**Grammer: 1. Lambda Function Definition:**

S -> 'lambda' PARAMETERS ':' EXPRESSION

PARAMETERS -> ID | ID ',' PARAMETERS

ID -> letter (letter | digit)\* letter -> [a-z A-Z\_]

digit -> [0-9]

1. **Dictionary Declaration:** 
   1. -> '{' PAIRS '}'

PAIRS -> PAIR | PAIR ',' PAIRS

PAIR -> STRING ':' EXPRESSION

STRING -> '"' letter\* '"'

1. **List Comprehension:** 
   1. -> '[' EXPRESSION 'for' ID 'in' EXPRESSION ']' ID -> letter (letter | digit) \*
2. **Exception Handling (try-except):** S -> 'try' ':' BLOCK 'except' ':' BLOCK BLOCK -> STATEMENT | STATEMENT BLOCK

STATEMENT -> statement **5. Generator Function with yield:**

* 1. -> 'def' ID '(' PARAMETERS ')' ':' YIELD\_BLOCK

YIELD\_BLOCK -> YIELD\_STATEMENT | YIELD\_STATEMENT

BLOCK

BLOCK -> STATEMENT | STATEMENT BLOCK

YIELD\_STATEMENT -> 'yield' EXPRESSION STATEMENT -> statement

**Lexer and Parser Program:**

import ply.lex as lex

import ply.yacc as yacc

# List of token names tokens = (

'LAMBDA', 'COLON', 'COMMA', 'LBRACE', 'RBRACE', 'LBRACKET',

'RBRACKET',

'LPAREN', 'RPAREN', 'FOR', 'IN', 'TRY', 'EXCEPT', 'DEF', 'YIELD',

'PASS', 'IDENTIFIER', 'STRING', 'NUMBER', 'PLUS'

)

# Regular expression rules for simple tokens

t\_LAMBDA = r'lambda' t\_COLON = r':' t\_COMMA = r',' t\_LBRACE = r'\{' t\_RBRACE = r'\}' t\_LBRACKET = r'\[' t\_RBRACKET = r'\]' t\_LPAREN = r'\(' t\_RPAREN = r'\)' t\_FOR = r'for' t\_IN = r'in' t\_TRY = r'try' t\_EXCEPT = r'except' t\_DEF = r'def' t\_YIELD = r'yield' t\_PASS = r'pass' t\_PLUS = r'\+'

t\_STRING = r'\"[a-zA-Z\_][a-zA-Z0-9\_]\*\"'

# Reserved words reserved = {

'lambda': 'LAMBDA',

'for': 'FOR',

'in': 'IN',

'try': 'TRY',

'except': 'EXCEPT',

'def': 'DEF',

'yield': 'YIELD',

'pass': 'PASS'

}

# IDENTIFIER rule, with reserved word check def t\_IDENTIFIER(t):

*r'[a-zA-Z\_][a-zA-Z0-9\_]\*'*

t.type = reserved.get(t.value, 'IDENTIFIER') # Check for reserved words

return t

t\_NUMBER = r'[0-9]+'

# Ignored characters (whitespace) t\_ignore = " \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\_start(p):

*'''start : lambda\_func*

*| dict\_decl*

*| list\_comprehension*

*| try\_except | generator\_func'''* pass

def p\_lambda\_func(p):

*'''lambda\_func : LAMBDA params COLON expression'''* print("Lambda function recognized")

def p\_params(p):

*'''params : IDENTIFIER*

*| IDENTIFIER COMMA params'''* pass

def p\_dict\_decl(p):

*'''dict\_decl : LBRACE pairs RBRACE'''*

print("Dictionary recognized")

def p\_pairs(p):

*'''pairs : pair*

*| pair COMMA pairs'''* pass

def p\_pair(p):

*'''pair : STRING COLON expression'''* pass

def p\_list\_comprehension(p):

*'''list\_comprehension : LBRACKET expression FOR IDENTIFIER IN expression RBRACKET'''* print("List comprehension recognized")

def p\_try\_except(p):

*'''try\_except : TRY COLON block EXCEPT COLON block'''* print("Try-except block recognized")

def p\_block(p):

*'''block : statement | statement block'''* pass

def p\_generator\_func(p):

*'''generator\_func : DEF IDENTIFIER LPAREN RPAREN COLON yield\_block'''*

print("Generator function recognized") def p\_yield\_block(p):

*'''yield\_block : yield\_statement*

*| yield\_statement block'''* pass def p\_yield\_statement(p):

*'''yield\_statement : YIELD expression'''* pass def p\_statement(p):

*'''statement : YIELD expression*

*| PASS'''* pass

# Define operator precedence precedence = (

('left', 'PLUS'),

)

def p\_expression(p):

*'''expression : IDENTIFIER*

*| NUMBER*

*| expression PLUS expression*

*| IDENTIFIER LPAREN expression RPAREN'''* pass

# Error rule for syntax errors def p\_error(p):

print(p) if p: print(f"Syntax error at '{p.value}'") else:

print("Syntax error at EOF")

# Build the parser

parser = yacc.yacc()

# Test cases for the constructs

test\_cases = [

'try: pass except: pass', # Try-except block

'{"key": value, "name": John}', # Dictionary declaration

'[x for x in range(10)]', # List comprehension

'def gen(): yield x', # Generator function with yield

'lambda x, y: x + y', # Lambda function

'lambda a: a + 5', # Simple lambda with one parameter

'def fibonacci(): yield 1', # Generator function yielding a constant

'{"age": 30, "height": 180}', # Dictionary with multiple key-value pairs

]

# Parse each test case for test in test\_cases: print(f"\nTesting: {test}") result = parser.parse(test, lexer=lexer) if result is None: print("Parsed successfully") else:

print("Parsing failed")

**Output:**

