PES University, RR Campus, Bangalore - 560085

Automata Formal Languages and Logic

UE23CS242A

Mini Project

3rd Semester, Academic Year 2024

Date: 12/11/2024

To describe and write Grammar for the constraints of a programming language

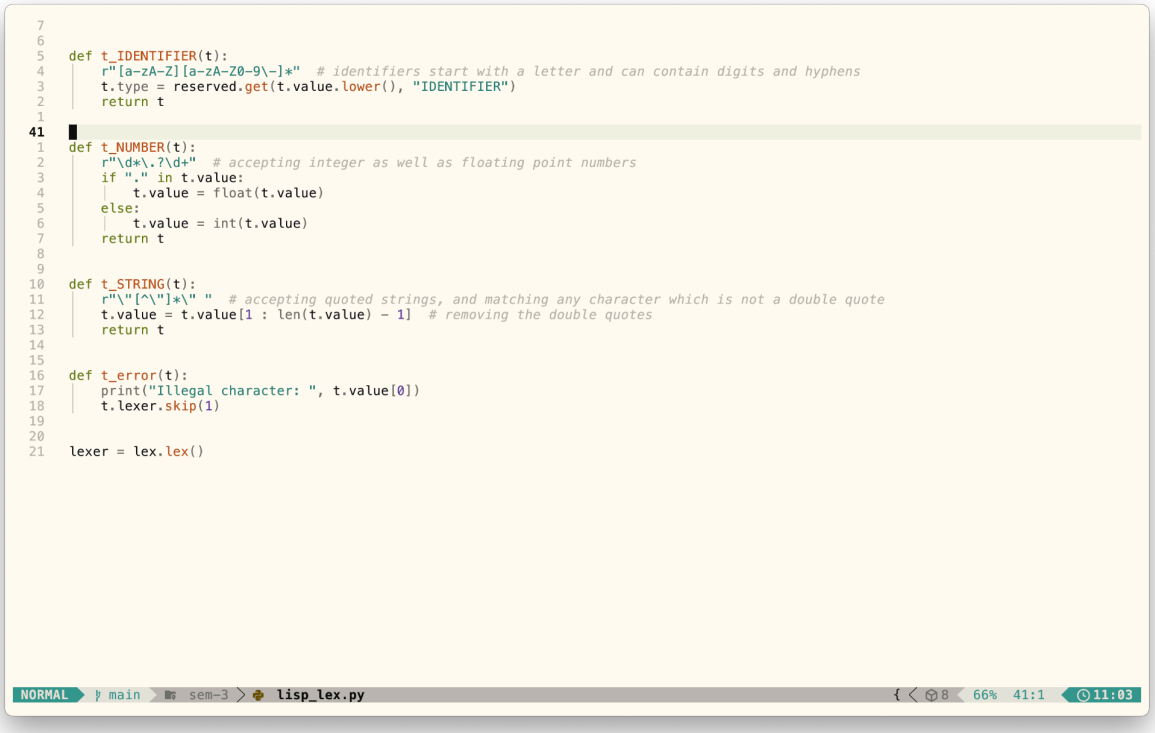
Language : Lisp

Constrains : Variable assignment, if-else conditions and function declaration

Name of team member:	SRN of team member:	Section :
Anirudh Sudhir	PES1UG23CS917	B

Lexer

```
from ply import lex

reserved = {"if": "IF", "defun": "DEFUN", "setq": "SETQ", "t": "T", "nil": "NIL"}
tokens = (
    "LPAREN",
    "RPAREN",
    "IDENTIFIER",
    "NUMBER",
    "STRING",
    "EQUALS",
    "PLUS",
    "MINUS",
    "MULTIPLY",
    "DIVIDE",
    "GREATER",
    "LESS",
    "GREATEREQUAL",
    "LESSEQUAL",
) + tuple(reserved.values())

t_LPAREN = r"\("
t_RPAREN = r"\)"
t_EQUALS = r"="
t_PLUS = r"\+"
t_MINUS = r"-"
t_MULTIPLY = r"\*"
t_DIVIDE = r"/"
t_GREATER = r">"
t_LESS = r"<"
t_GREATEREQUAL = r">="
t_LESSEQUAL = r"<="

t_ignore = " \t\n"


def t_IDENTIFIER(t):
    r"[a-zA-Z][a-zA-Z0-9\-]*"  # identifiers start with a letter and can contain digits and hyphens
    t.type = reserved.get(t.value.lower(), "IDENTIFIER")
    return t


def t_NUMBER(t):
    r"\d*\.?\d+"  # accepting integer as well as floating point numbers
    if "." in t.value:
NORMAL  main  sem-3  lisp_lex.py                                  { < 8  Top  1:1  11:02
```



```
def t_IDENTIFIER(t):
    r"[a-zA-Z][a-zA-Z0-9\-]*"  # identifiers start with a letter and can contain digits and hyphens
    t.type = reserved.get(t.value.lower(), "IDENTIFIER")
    return t

def t_NUMBER(t):
    r"\d*\.?\d+"  # accepting integer as well as floating point numbers
    if "." in t.value:
        t.value = float(t.value)
    else:
        t.value = int(t.value)
    return t


def t_STRING(t):
    r"\"[^\"]*\" "  # accepting quoted strings, and matching any character which is not a double quote
    t.value = t.value[1 : len(t.value) - 1]  # removing the double quotes
    return t


def t_error(t):
    print("Illegal character: ", t.value[0])
    t.lexer.skip(1)


lexer = lex.lex()
```

NORMAL main sem-3 lisp_lex.py 8 66% 41:1 11:03

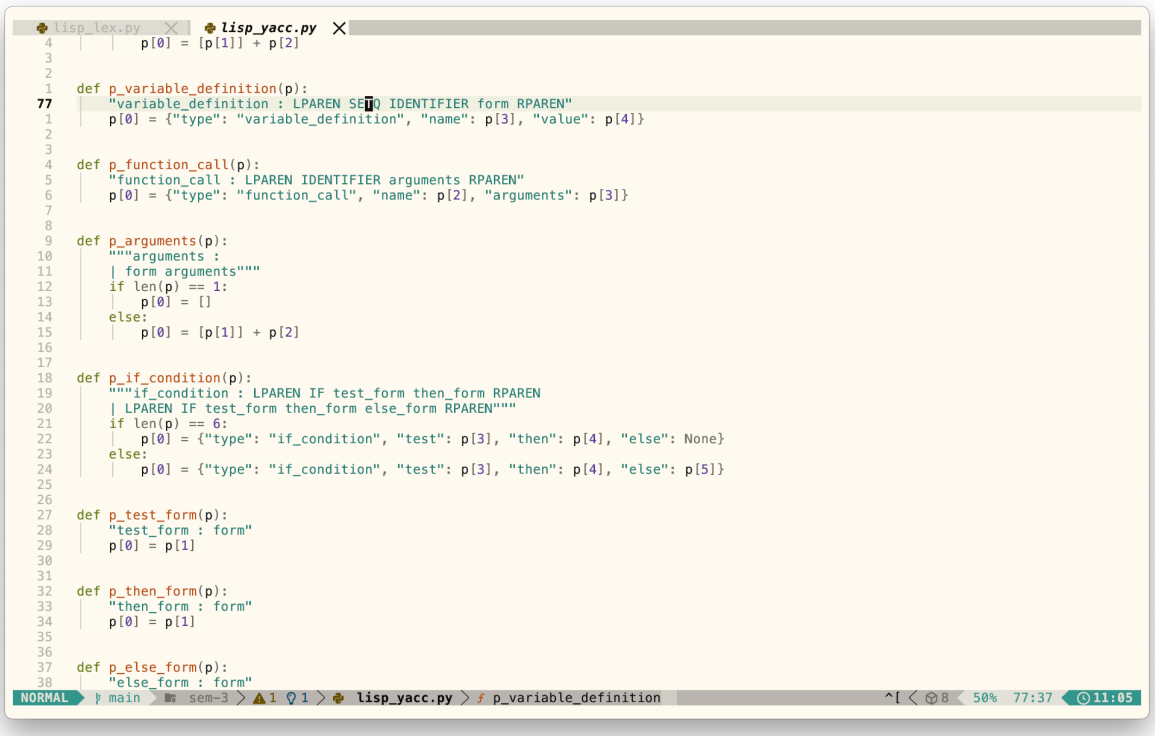
Parser

lisp_lex.py lisp_yacc.py

```
from ply import yacc

from lisp_lex import tokens


def p_program(p):
    """program : form
    | program form"""
    if len(p) == 2:
        p[0] = [p[1]]
    else:
        p[0] = p[1] + [p[2]]


def p_form(p):
    """form : function_definition
    | variable_definition
    | function_call
    | if_condition
    | literal
    | operation"""
    p[0] = p[1]


def p_operation(p):
    """operation : LPAREN operator arguments RPAREN"""
    p[0] = {"type": "operation", "operator": p[2], "arguments": p[3]}


def p_operator(p):
    """operator : EQUALS
    | PLUS
    | MINUS
    | MULTIPLY
    | DIVIDE
    | GREATER
    | LESS
    | GREATEREQUAL
    | LESSEQUAL"""
    p[0] = p[1]


def p_function_definition(p):
```

NORMAL main sem-3 1 1 lisp_yacc.py ^[8 1% 2:1 11:05


lisp_lex.py
lisp_yacc.py
| LESS
| GREATEREQUAL
| LESSEQUAL"""
p[0] = p[1]
def p_function_definition(p):
"function_definition : LPAREN DEFUN IDENTIFIER parameter_list body RPAREN"
p[0] = {
"type": "function_definition",
"name": p[3],
"parameters": p[4],
"body": p[5],
}
def p_parameter_list(p):
"parameter_list : LPAREN parameters RPAREN"
p[0] = p[2]
def p_parameters(p):
"""parameters :
| IDENTIFIER parameters"""
if len(p) == 1:
p[0] = []
else:
p[0] = [p[1]] + p[2]
def p_body(p):
"""body : form
| form body"""
if len(p) == 2:
p[0] = [p[1]]
else:
p[0] = [p[1]] + p[2]
def p_variable_definition(p):
"variable_definition : LPAREN SETQ IDENTIFIER form RPAREN"
p[0] = {"type": "variable_definition", "name": p[3], "value": p[4]}
NORMAL
main
sem-3
1
1
lisp_yacc.py
11:05


lisp_lex.py
lisp_yacc.py
p[0] = [p[1]] + p[2]
def p_variable_definition(p):
"variable_definition : LPAREN SETQ IDENTIFIER form RPAREN"
p[0] = {"type": "variable_definition", "name": p[3], "value": p[4]}
def p_function_call(p):
"function_call : LPAREN IDENTIFIER arguments RPAREN"
p[0] = {"type": "function_call", "name": p[2], "arguments": p[3]}
def p_arguments(p):
"""arguments :
| form arguments"""
if len(p) == 1:
p[0] = []
else:
p[0] = [p[1]] + p[2]
def p_if_condition(p):
"""if_condition : LPAREN IF test_form then_form RPAREN
| LPAREN IF test_form then_form else_form RPAREN"""
if len(p) == 6:
p[0] = {"type": "if_condition", "test": p[3], "then": p[4], "else": None}
else:
p[0] = {"type": "if_condition", "test": p[3], "then": p[4], "else": p[5]}
def p_test_form(p):
"test_form : form"
p[0] = p[1]
def p_then_form(p):
"then_form : form"
p[0] = p[1]
def p_else_form(p):
"else_form : form"
NORMAL
main
sem-3
1
1
lisp_yacc.py
p_variable_definition
11:05

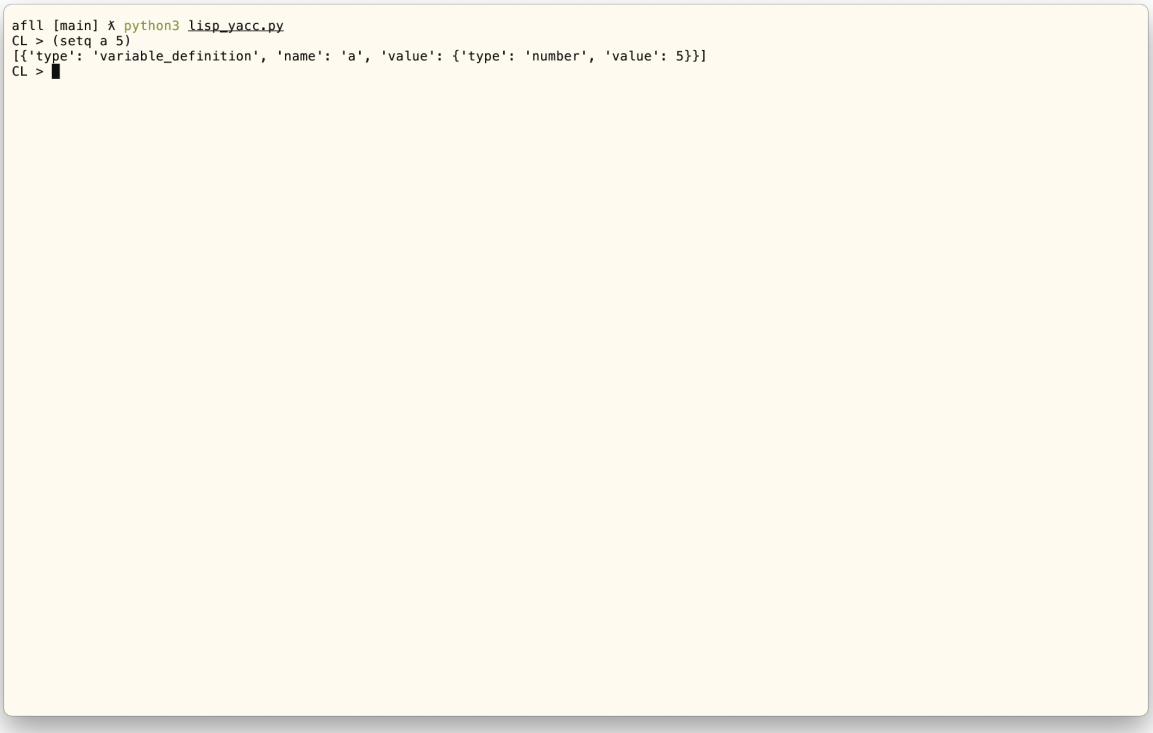

lisp_lex.py
lisp_yacc.py
p[0] = p[1]
def p_then_form(p):
"then_form : form"
p[0] = p[1]
def p_else_form(p):
"else_form : form"
p[0] = p[1]
def p_literal(p):
"""literal : NUMBER
| STRING
| T
| NIL
| IDENTIFIER"""
if isinstance(p[1], (int, float)):
p[0] = {"type": "number", "value": p[1]}
elif p[1] == "t":
p[0] = {"type": "boolean", "value": True}
elif p[1] == "nil":
p[0] = {"type": "boolean", "value": False}
elif isinstance(p[1], str):
if p.slice[1].type == "STRING":
p[0] = {"type": "string", "value": p[1]}
else:
p[0] = {"type": "identifier", "value": p[1]}
def p_error(p):
if p:
print("Syntax error at: ", p.value)
else:
print("Syntax error at EOF")
parser = yacc.yacc()
while True:
try:
s = input("CL > ")
NORMAL main sem-3 1 1 lisp_yacc.py p_else_form
^[8 75% 116:15 11:05


lisp_lex.py
lisp_yacc.py
| IDENTIFIER"""
if isinstance(p[1], (int, float)):
p[0] = {"type": "number", "value": p[1]}
elif p[1] == "t":
p[0] = {"type": "boolean", "value": True}
elif p[1] == "nil":
p[0] = {"type": "boolean", "value": False}
elif isinstance(p[1], str):
if p.slice[1].type == "STRING":
p[0] = {"type": "string", "value": p[1]}
else:
p[0] = {"type": "identifier", "value": p[1]}
def p_error(p):
if p:
print("Syntax error at: ", p.value)
else:
print("Syntax error at EOF")
parser = yacc.yacc()
while True:
try:
s = input("CL > ")
except EOFError:
break
if not s:
continue
result = parser.parse(s)
print(result)
NORMAL main sem-3 1 1 lisp_yacc.py p_literal
^[8 83% 128:37 11:06

Output

Variable assignment

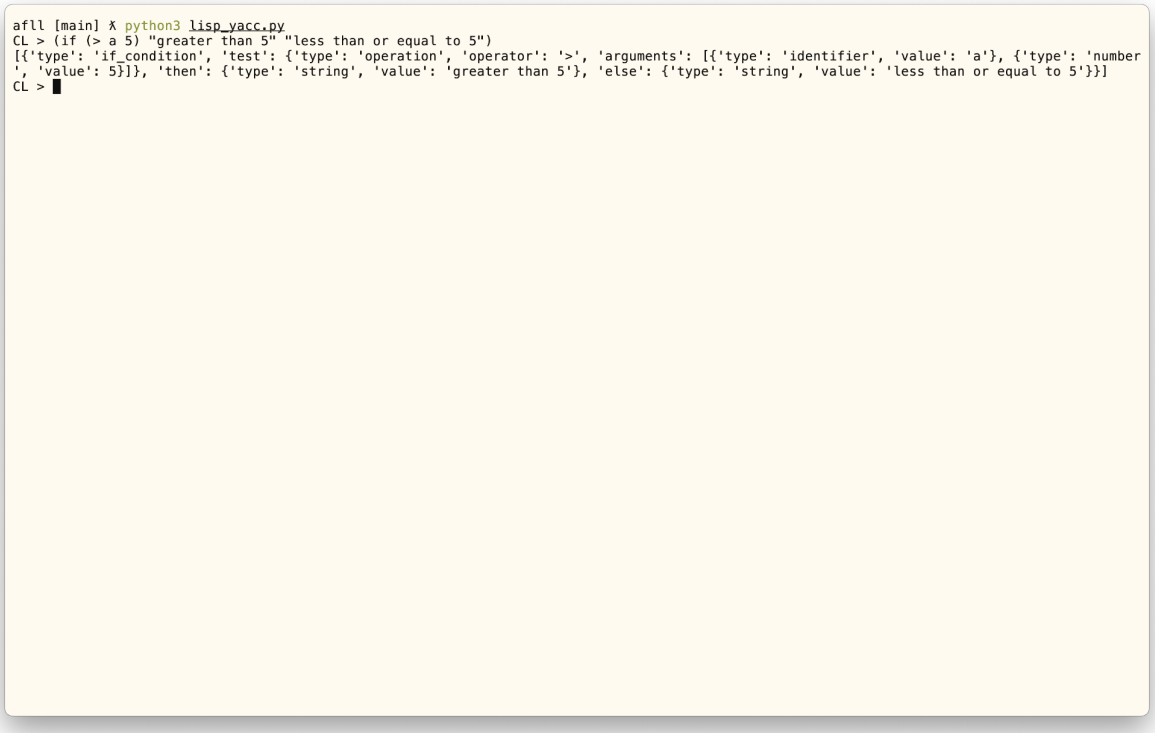
- Valid syntax



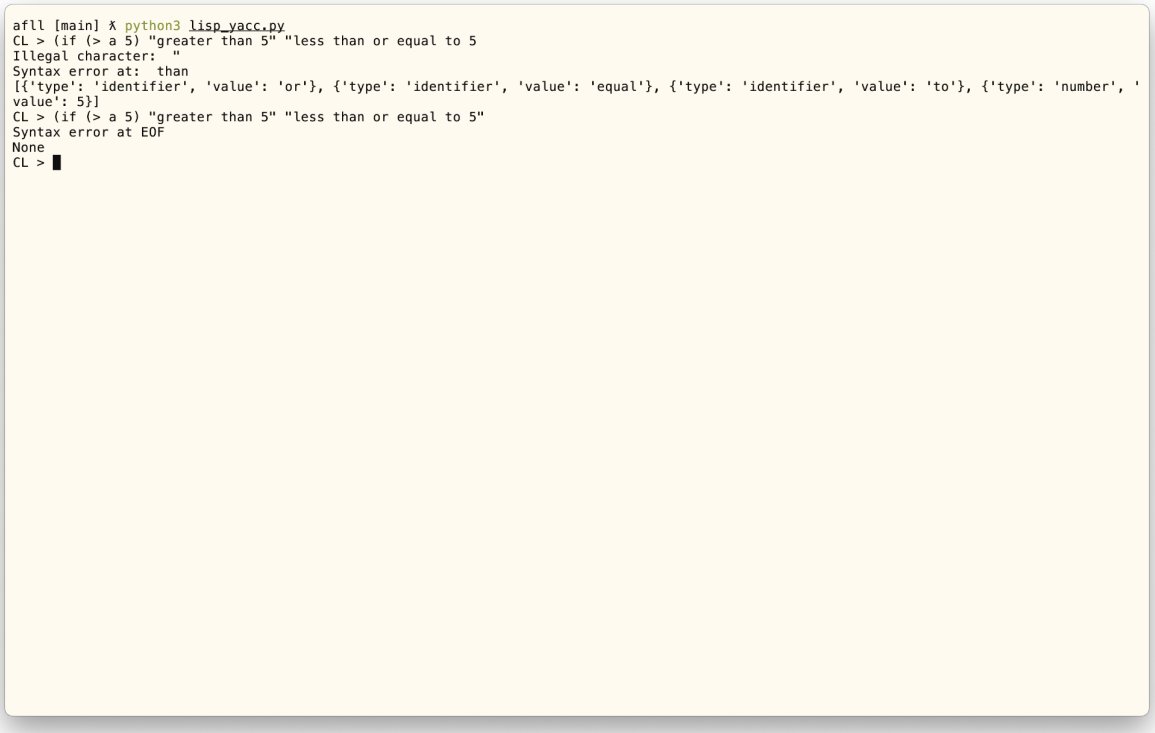
- Invalid syntax (error)

If condition

- Valid syntax



- Invalid syntax (error)



Function declaration

- ## Valid syntax

```
afll [main] ✗ python3 lisp_yacc.py
CL > (defun foo (a) ( if (> a 5) "greater than 5" "less than or equal to 5") )
[{'type': 'function_definition', 'name': 'foo', 'parameters': ['a'], 'body': [{'type': 'if_condition', 'test': {'type': 'operation', 'operato
r': '>', 'arguments': [{'type': 'identifier', 'value': 'a'}, {'type': 'number', 'value': 5}]}, 'then': {'type': 'string', 'value': 'greater t
han 5'}, 'else': {'type': 'string', 'value': 'less than or equal to 5'}}]}]
CL >
```

- ## Invalid syntax (error)

```
afll [main] ✗ python3 lisp_yacc.py
CL > (defun foo (a) ( if (> a 5) "greater than 5" "less than or equal to
Illegal character:  "
Syntax error at:  than
[{'type': 'identifier', 'value': 'or'}, {'type': 'identifier', 'value': 'equal'}, {'type': 'identifier', 'value': 'to'}]
CL > (defun foo (a) ( if (> a 5) "greater than 5" "less than or equal to 5")
Syntax error at EOF
None
CL > defun foo (a) ( if (> a 5) "greater than 5" "less than or equal to 5") )
Syntax error at:  defun
Syntax error at:  )
None
CL >
```