Lex

```python
# A simple lexical analyzer for PHP source code using PLY
#import lex from python lex yacc
from ply import lex

# List of PHP keywords
keywords = [
    'PRINT',
    'ECHO',
    'IF',
    'ELSE',
    'WHILE',
]

# List of tokens
tokens = keywords + [
    'PHP_OPEN',
    'PHP_CLOSE',
    'EQUALS',
    'PLUS',
    'DIVIDE',
    'LPAREN',
    'RPAREN',
    'RCURLY',
    'LCURLY',
    'LESSEQUAL',
    'GREATERTHAN',
    'NOTEQUAL',
    'CONCAT',
    'SEMI',
    'MOD',
    'NUMBER',
    'STRING',
    'VAR'
]

t_ignore = ' \t'

t_PHP_OPEN = r'<\?php|<\?'
t_PHP_CLOSE = r'\?>'

t_ignore_COMMENT =r'//.*'

def t_KEYWORD(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
    if t.value.upper() in keywords:
        t.type = t.value.upper()
    return t
def t_VAR(t):
    r'\$[a-zA-Z_][a-zA-Z0-9_]*'
    t.value = t.value[1:]
    return t

def t_STRING(t):
    r'"([^"\\]*(\\.[^"\\]*)*)"'
    t.value = t.value[1:-1]
    t.value = bytes(t.value, "utf-8").decode("unicode_escape")
    return t

t_NUMBER = r'\d+'
t_EQUALS = r'\='
t_PLUS = r'\+'
t_DIVIDE = r'\/'
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_LCURLY = r'\{'
t_RCURLY = r'\}'
t_LESSEQUAL = r'\<='
t_GREATERTHAN = r'\>'
t_CONCAT = r'\.'
t_SEMI = r'\;'
t_NOTEQUAL = r'\!='
t_MOD = r'\%'

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

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

lex.lex(debug=0)
```

Parser

```python
import ply.yacc as yacc
import phplex

tokens = phplex.tokens

precedence = (
    ('left', 'PLUS'),
    ('left', 'DIVIDE'),
)

# Grammar rules
def p_program(p):
    '''program : PHP_OPEN statements PHP_CLOSE'''
    p[0] = ('program', p[2])

def p_statements(p):
    '''statements : statement statements
                  | statement'''
    if len(p) == 3:
        p[0] = [p[1]] + p[2]
    else:
        # If it's the last statement, it shouldn't require a semicolon
        p[0] = [p[1]]

def p_statement_print(p):
    '''statement : PRINT expression SEMI
                 | PRINT expression'''
    p[0] = ('print', p[2])

def p_statement_echo(p):
    '''statement : ECHO expression SEMI
                 | ECHO expression'''
    p[0] = ('echo', p[2])

def p_statement_assign(p):
    '''statement : VAR EQUALS expression SEMI'''
    var_name = p[1]  # Variable name without the '$'
    value = p[3]  # The value being assigned
    p[0] = ('assign', var_name, value)

def p_statement_if_else(p):
    '''statement : IF LPAREN expression RPAREN LCURLY statements RCURLY ELSE LCURLY statements RCURLY'''
    p[0] = ('ifelse', p[3], p[6], p[10])

def p_statement_while(p):
    '''statement : WHILE LPAREN expression RPAREN LCURLY statements RCURLY'''
    p[0] = ('while', p[3], p[6])

def p_statement_if(p):
    '''statement : IF LPAREN expression RPAREN LCURLY statements RCURLY'''
    p[0] = ('if', p[3], p[6])

def p_expression_binop(p):
    '''expression : expression PLUS expression
                  | expression DIVIDE expression
                  | expression MOD expression
                  | expression LESSEQUAL expression
                  | expression GREATERTHAN expression
                  | expression NOTEQUAL expression
                  | expression CONCAT expression'''
    p[0] = ('binop', p[2], p[1], p[3])

def p_expression_group(p):
    '''expression : LPAREN expression RPAREN'''
    p[0] = p[2]

def p_expression_number(p):
    '''expression : NUMBER'''
    p[0] = p[1]

def p_expression_id(p):
    '''expression : VAR'''
    var_name = p[1][1:] if p[1].startswith('$') else p[1]  # Remove '$' from VAR tokens
    p[0] = ('var', var_name)

def p_expression_string(p):
    '''expression : STRING'''
    p[0] = ('string', p[1])

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

def eval_condition(condition):
    if isinstance(condition, tuple):
        left = condition[0]
        operator = condition[1]
        right = condition[2]
        if operator == '>':
            return left > right
        elif operator == '<=':
            return left <= right
    return False

parser = yacc.yacc()
```

Interpreter

```python
from phpparser import parser

variables = {}
class Interpreter:
    def eval(self, node):
        if isinstance(node, tuple):
            if node[0] == 'program':
                for statement in node[1]:
                    self.eval(statement)
            elif node[0] == 'assign':
                var_name = node[1]
                value = self.eval(node[2])
                variables[var_name] = value
            elif node[0] == 'binop':
                left = self.eval(node[2])
                right = self.eval(node[3])
                op = node[1]
                if isinstance(left, str) and left.replace('.', '', 1).isdigit():
                    left = float(left) if '.' in left else int(left)
                if isinstance(right, str) and right.replace('.', '', 1).isdigit():
                    right = float(right) if '.' in right else int(right)
                if op == '+':
                    return left + right
                elif op == '/':
                    result = left / right
                    return int(result) if result.is_integer() else result
                elif op == '%':
                    return left % right
                elif op == '>':
                    return left > right
                elif op == '<=':
                    return left <= right
                elif op == '!=':
                    return left != right
                elif op == '.':
                    return str(left) + str(right)

            elif node[0] == 'number':
                return node[1]
            elif node[0] == 'string':
                return node[1]
            elif node[0] == 'var':
                var_name = node[1]
                if var_name in variables:
                    value = variables[var_name]
                    # Convert to number if possible
                    if isinstance(value, str) and value.replace('.', '',
1).isdigit():            return float(value) if '.' in value else int(value)
                    return value
                else:
                    raise ValueError(f"Undefined variable '{var_name}'")
            elif node[0] == 'print':
                value = self.eval(node[1])
                print(value, end='')
                return value
            elif node[0] == 'echo':
                value = self.eval(node[1])
                print(value, end='')
                return value

            elif node[0] == 'ifelse':
                condition = self.eval(node[1])
                if condition:
                    # Execute the 'if' block if the condition is true
                    for statement in node[2]:
                        self.eval(statement)
                else:
                    for statement in node[3]:
                        self.eval(statement)
            elif node[0] == 'if':
                condition = self.eval(node[1])
                if condition:
                    for statement in node[2]:
                        self.eval(statement)
            elif node[0] == 'while':
                while self.eval(node[1]):
                    for statement in node[2]:
                        self.eval(statement)
        else:
            return node
```

php.py

```
from phplex import lex
from phpparser import parser
from phpinter import Interpreter
import sys

if __name__ == "__main__":
    php_code = open(sys.argv[1]).read()

    # Tokenize the PHP code
    lex.input(php_code)
    tokens = list(lex.token() for _ in
range(len(php_code)))
    # Parse the tokens
    code = parser.parse(php_code)

    # Interpret the parsed code
    interpreter = Interpreter()
    interpreter.eval(code)
```