院系	年级专业	姓名	学号	实验日期
计算机学院	2019计科	吴家隆	1915404063	2021.12.13

编程语言：***python3.9***

实验内容

- 利用PLY实现的Python程序的解析

  本次学习的语法是**类语句**，需要注意的是本次使用的语法做了一些改进，不是纯粹的python2语法。

  需要结合上次课四则运算的解析程序

  1.示例程序位于example4/

  2.需要进行解析的文件为学生信息stu.py

  stu.py

  ```
  class Student{
      def __init__(self,name,age,score){
          self.name=name
          self.age=age
          self.score=score
      }

      def add_score(self,score){
          self.score=self.score+score
      }

      def print_info(self){
          print(self.name,self.age)
      }
  }

  a=Student('xiaoming',12,20)
  a.add_score(60)
  a.print_info()
  ```

  3.解析结果以语法树的形式呈现

- 编程实现语法制导翻译

本次课主要需要实现类的解析。主要需要实现：

（1）类的解析

（2）类中变量的翻译

（3）类中函数的翻译

实验步骤

使用lex进行序列标记

在本次实验中要识别的tokens包括以下

```
tokens = ('VARIABLE', 'NUMBER', 'PRINT', 'DEF', 'CLASS', 'SELF', 'STR')

literals = ['=', '+', '-', '*', '(', ')', '{', '}', '<', '>', ',', '.']
```

ply使用"t_"开头的变量来表示规则。如果变量是一个字符串，那么它被解释为一个正则表达式，匹配值是标记的值。 如果变量是函数，则其文档字符串包含模式，并使用匹配的标记调用该函数。该函数可以自由地修改序列或返回一个新的序列来代替它的位置。 如果没有返回任何内容，则忽略匹配。 通常该函数只更改“value”属性，它最初是匹配的文本。

```
def t_NUMBER(t):
    r'[0-9]+'
    return t
def t_STR(t):
    r'\'\w*\''
    return t
def t_PRINT(t):
    r'(?<!\.)print(?=\()'
    return t
def t_DEF(t):
    r'def'
    return t
def t_CLASS(t):
    r'class'
    return t
def t_SELF(t):
    r'self'
    return t
def t_VARIABLE(t):
    r'[a-zA-Z\$_][a-zA-Z\d_]*'
    return t
t_ignore = " \t"
def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)
```

对stu.py进行测试，输出每一个识别到的token

```
from util import clear_text
text=clear_text(open('stu.py','r').read())
lex.input(text)
for tok in iter(lex.token, None):
    print(repr(tok.type), repr(tok.value))
```

util中的clear_text函数为清除每行的空格

```
def clear_text(text):
    lines=[]
    for line in text.split('\n'):
        line=line.strip()
        if len(line)>0:
            lines.append(line)
    return ' '.join(lines)
```

使用yacc进行语法分析

PLY 的解析器适用于lex解析出的序列标记。 它使用 BNF 语法来描述这些标记是如何组装的。

对node进行定义

```
class node:
    def __init__(self, data):
        self._data = data
        self._children = []
        self._value = None
    def getdata(self):
        return self._data
    def setvalue(self, value):
        self._value = value
    def getvalue(self):
        return self._value
    def getchild(self, i):
        return self._children[i]
    def getchildren(self):
        return self._children
    def add(self, node):
        self._children.append(node)
    def print_node(self, prefix):
        print('  ' * prefix, '+', self._data)
        for child in self._children:
            child.print_node(prefix + 1)
def num_node(data):
    t = node(data)
    t.setvalue(float(data))
    return t
```

定义文法

```
def simple_node(t, name):
    t[0] = node(name)
    for i in range(1, len(t)):
        t[0].add(node(t[i]))
    return t[0]
def p_program(t):
    '''program : statements'''
    if len(t) == 2:
        t[0] = node('[PROGRAM]')
        t[0].add(t[1])
def p_statements(t):
    '''statements : statements statement
```

```
                  | statement'''
    if len(t) == 3:
        t[0] = node('[STATEMENTS]')
        t[0].add(t[1])
        t[0].add(t[2])
    elif len(t) == 2:
        t[0] = node('[STATEMENTS]')
        t[0].add(t[1])
def p_statement(t):
    ''' statement : assignment
                  | operation
                  | print
                  | function
                  | run_function
                  | class'''
    if len(t) == 2:
        t[0] = node('[STATEMENT]')
        t[0].add(t[1])
def p_assignment(t):
    '''assignment : VARIABLE '=' NUMBER
                  | VARIABLE '[' expression ']' '=' NUMBER
                  | VARIABLE '=' VARIABLE
                  | VARIABLE '=' VARIABLE '[' expression ']'
                  | self '=' VARIABLE
                  | VARIABLE '=' VARIABLE '(' expressions ')' '''
    if len(t) == 4:
        if isinstance(t[1], node):        # self
            t[0] = node('[ASSIGNMENT]')
            t[0].add(t[1])
            t[0].add(node(t[2]))
            t[0].add(node(t[3]))
        elif isinstance(t[3], str):       # NUMBER or VARIABLE
            if ord('0') <= ord(t[3][0]) <= ord('9'):    # NUMBER
                t[0] = node('[ASSIGNMENT]')
                t[0].add(node(t[1]))
                t[0].add(node(t[2]))
                t[0].add(num_node(t[3]))
            else:                                       # VARIABLE
                t[0] = node('[ASSIGNMENT]')
                t[0].add(node(t[1]))
                t[0].add(node(t[2]))
                t[0].add(node(t[3]))
    elif len(t) == 7:
        if t[2] == '[':                         # NUMBER
            t[0] = node('[ASSIGNMENT]')
            t[0].add(node(t[1]))
            t[0].add(t[3])
            t[0].add(node(t[5]))
            t[0].add(num_node(t[6]))
        elif t[5].getdata() == '[EXPRESSION]':  # VARIABLE '[' expression ']'
            t[0] = node('[ASSIGNMENT]')
            t[0].add(node(t[1]))
            t[0].add(node(t[2]))
            t[0].add(node(t[3]))
            t[0].add(t[5])
        elif t[5].getdata() == '[EXPRESSIONS]': # VARIABLE '(' expressions ')'
            t[0] = node('[ASSIGNMENT]')
            t[0].add(node(t[1]))
```

```python
            t[0].add(node(t[2]))
            t[0].add(node(t[3]))
            t[0].add(t[5])
def p_self(t):
    '''self : SELF '.' VARIABLE'''
    if len(t) == 2:
        t[0] = node('[SELF]')
    elif len(t) == 4:
        t[0] = node('[SELF]')
        t[0].add(node(t[3]))
def p_operation(t):
    '''operation : VARIABLE '=' expression
                 | VARIABLE '+' '=' expression
                 | VARIABLE '-' '=' expression
                 | VARIABLE '[' expression ']' '=' expression
                 | self '=' expression'''
    if len(t) == 4:
        if isinstance(t[1], node):  # self '=' expression
            t[0] = node('[OPERATION]')
            t[0].add(t[1])
            t[0].add(node(t[2]))
            t[0].add(t[3])
        else:                           # VARIABLE '=' expression
            t[0] = node('[OPERATION]')
            t[0].add(node(t[1]))
            t[0].add(node(t[2]))
            t[0].add(t[3])
    elif len(t) == 5:
        t[0] = node('[OPERATION]')
        t[0].add(node(t[1]))
        t[0].add(node(t[2] + t[3]))
        t[0].add(t[4])
    elif len(t) == 7:
        t[0] = node('[OPERATION]')
        t[0].add(node(t[1]))
        t[0].add(t[3])
        t[0].add(node(t[5]))
        t[0].add(t[6])
def p_expression(t):
    '''expression : expression '+' term
                  | expression '-' term
                  | term'''
    if len(t) == 4:
        t[0] = node('[EXPRESSION]')
        t[0].add(t[1])
        t[0].add(node(t[2]))
        t[0].add(t[3])
    elif len(t) == 2:
        t[0] = node('[EXPRESSION]')
        t[0].add(t[1])
def p_term(t):
    '''term : term '*' factor
            | term '/' factor
            | factor'''
    if len(t) == 4:
        t[0] = node('[TERM]')
        t[0].add(t[1])
        t[0].add(node(t[2]))
        t[0].add(t[3])
    elif len(t) == 2:
        t[0] = node('[TERM]')
        t[0].add(t[1])
def p_factor(t):
    '''factor : NUMBER
              | VARIABLE
              | STR
              | self
              | VARIABLE '[' expression ']'
              | '(' expression ')' '''
    if len(t) == 2:
        if isinstance(t[1], node):                      # self
            t[0] = node('[FACTOR]')
            t[0].add(t[1])
        elif ord('0') <= ord(t[1][0]) <= ord('9'):       # NUMBER
            t[0] = node('[FACTOR]')
            t[0].add(num_node(t[1]))
        elif t[1][0] == '"' and t[1][-1] == '"':        # STR
            t[0] = node('[FACTOR]')
            t[0].add(node(t[1]))
        else:                                           # VARIABLE
            t[0] = node('[FACTOR]')
            t[0].add(node(t[1]))
    elif len(t) == 4:
        t[0] = node('[FACTOR]')
        t[0].add(t[2])
    elif len(t) == 5:
        t[0] = node('[FACTOR]')
        t[0].add(node(t[1]))
        t[0].add(t[3])
def p_print(t):
    '''print : PRINT '(' variables ')' '''
    if len(t) == 5:
        t[0] = node('[PRINT]')
        t[0].add(t[3])
def p_function(t):
    '''function : DEF VARIABLE '(' variables ')' '{' statements '}'
                | DEF VARIABLE '(' SELF ')' '{' statements '}'
                | DEF VARIABLE '(' SELF ',' variables ')' '{' statements '}' '''
    if len(t) == 9:
        if t[4] == 'self':
            t[0] = node('[FUNCTION]')
            t[0].add(node(t[2]))
            t[0].add(node('[SELF]'))
            t[0].add(t[7])
        elif t[4].getdata() == '[VARIABLES]':
            t[0] = node('[FUNCTION]')
            t[0].add(node(t[2]))
            t[0].add(t[4])
            t[0].add(t[6])
            t[0].add(t[9])
    elif len(t) == 11:
        t[0] = node('[FUNCTION]')
        t[0].add(node(t[2]))
        t[0].add(node('[SELF]'))
        t[0].add(t[6])
        t[0].add(t[9])
```

```python
def p_run_function(t):
    '''run_function : VARIABLE '(' expressions ')'
                    | VARIABLE '.' VARIABLE '(' expressions ')' '''
    if len(t) == 5:
        t[0] = node('[RUN_FUNCTION]')
        t[0].add(node(t[1]))
        t[0].add(t[3])
    elif len(t) == 7:
        t[0] = node('[RUN_FUNCTION]')
        t[0].add(node(t[1]))
        t[0].add(node(t[3]))
        t[0].add(t[5])
def p_variables(t):
    '''variables :
                 | VARIABLE
                 | variables ',' VARIABLE
                 | self
                 | variables ',' self'''
    if len(t) == 1:
        t[0] = node('[VARIABLES]')
        t[0].add(node('[NONE]'))
    elif len(t) == 2:
        if isinstance(t[1], node):
            t[0] = node('[VARIABLES]')
            t[0].add(t[1])
        else:
            t[0] = node('[VARIABLES]')
            t[0].add(node(t[1]))
    elif len(t) == 4:
        if isinstance(t[3], node):
            t[0] = node('[VARIABLES]')
            t[0].add(t[1])
            t[0].add(t[3])
        else:
            t[0] = node('[VARIABLES]')
            t[0].add(t[1])
            t[0].add(node(t[3]))
def p_expressions(t):
    '''expressions :
                   | expression
                   | expressions ',' expression'''
    if len(t) == 1:
        t[0] = node('[EXPRESSIONS]')
        t[0].add(node('[NONE]'))
    elif len(t) == 2:
        t[0] = node('[EXPRESSIONS]')
        t[0].add(t[1])
    elif len(t) == 4:
        t[0] = node('[EXPRESSIONS]')
        t[0].add(t[1])
        t[0].add(t[3])
def p_class(t):
    '''class : CLASS VARIABLE '{' statements '}' '''
    if len(t) == 6:
        t[0] = node('[CLASS]')
        t[0].add(node(t[2]))
        t[0].add(t[4])
def p_error(t):
    print("Syntax error at '%s'" % t.value)
yacc.yacc()
```

实现语法制导翻译

定义变量存储类变量

```
c_table = {}
```

定义Tran类进行翻译

将v_table和f_table放到Tran类中，v_table存储变量，f_table存储函数

```
class Tran:
    def __init__(self):
        self.v_table = {}  # variable table
        self.f_table = {}  # function table
```

再定义两个更新函数，更新上述两个表

```
def update_v_table(self, name, value):
    self.v_table[name] = value
def update_f_table(self, name, value):
    self.f_table[name] = value
```

翻译过程

ASSIGENMENT的文法

$$
\begin{gathered}
ASSIGENMENT: \\
VARIABLE = NUMBER \\
|VARIABLE[expression] = NUMBER \\
|VARIABLE = VARIABLE \\
|VARIABLE = VARIABLE[expression] \\
|self = VARIABLE \\
|VARIABLE = VARIABLE(expressions)
\end{gathered}
$$

```
def trans(self, node):
    if node.getdata() == '[ASSIGNMENT]':
        '''assignment : VARIABLE '=' NUMBER
                      | VARIABLE '[' expression ']' '=' NUMBER
                      | VARIABLE '=' VARIABLE
                      | VARIABLE '=' VARIABLE '[' expression ']'
                      | self '=' VARIABLE
                      | VARIABLE '=' VARIABLE '(' expressions ')' '''
        if len(node.getchildren()) == 3:
            if node.getchild(0).getdata() == '[SELF]':                    #
self '=' VARIABLE
                value = self.v_table[node.getchild(2).getdata()]
                # update v_table
                self.trans(node.getchild(0))
                # 注意这里访问类变量名用的是getvalue()
                self.update_v_table(node.getchild(0).getvalue(), value)
            elif ord('0') <= ord(node.getchild(2).getdata()[0]) <= ord('9'):  #
NUMBER
                value = node.getchild(2).getvalue()
                # update v_table
                self.update_v_table(node.getchild(0).getdata(), value)
            else:                                                             #
VARIABLE
                value = self.v_table[node.getchild(2).getdata()]
                # update v_table
                self.update_v_table(node.getchild(0).getdata(), value)
        elif len(node.getchildren()) == 4:
            if node.getchild(2).getdata() == '=':  # NUMBER
                arg = self.v_table[node.getchild(0).getdata()]
                self.trans(node.getchild(1))
                index = int(node.getchild(1).getvalue())
                value = node.getchild(3).getvalue()
                # update VARIABLE
                arg[index] = value
            elif node.getchild(3).getdata() == '[EXPRESSION]':  # VARIABLE '['
expression ']'
                arg1 = self.v_table[node.getchild(2).getdata()]
                self.trans(node.getchild(3))
                index = int(node.getchild(3).getvalue())
                value = arg1[index]
                # update v_table
                self.update_v_table(node.getchild(0).getdata(), value)
            elif node.getchild(3).getdata() == '[EXPRESSIONS]':  # VARIABLE '('
expressions ')'
                variable = node.getchild(0).getdata()
                cname = node.getchild(2).getdata()
                self.trans(node.getchild(3))
                vname1 = node.getchild(3).getvalue()
                c = c_table[cname]
                vname0, fnode = c.f_table['__init__']  # function_name :
(variable_names, function)
                for i in range(len(vname1)):
                    c.v_table[vname0[i]] = vname1[i]
                c.trans(fnode)
                self.update_v_table(variable, (cname, c))
```

SELF

$self : SELF . VARIABLE$

```
elif node.getdata() == '[SELF]':
    '''self : SELF '.' VARIABLE'''
    if len(node.getchildren()) == 0:
        pass
    elif len(node.getchildren()) == 1:
        value = 'self.' + node.getchild(0).getdata()
        node.setvalue(value)
```

OPERATION

$$operation:$$
$$VARIABLE = expression$$
$$|VARIABLE+ = expression$$
$$|VARIABLE- = expression$$
$$|VARIABLE[expression] = expression$$
$$|self = expression$$

```
elif node.getdata() == '[OPERATION]':
    '''operation : VARIABLE '=' expression
                 | VARIABLE '+' '=' expression
                 | VARIABLE '-' '=' expression
                 | VARIABLE '[' expression ']' '=' expression
                 | self '=' expression'''
    if len(node.getchildren()) == 3:
        if node.getchild(0).getdata() == '[SELF]':    # self '=' expression
            self.trans(node.getchild(0))
            self.trans(node.getchild(2))
            value = node.getchild(2).getvalue()
            self.update_v_table(node.getchild(0).getvalue(), value)
        elif node.getchild(1).getdata()[0] == '=':    # VARIABLE '=' expression
            self.trans(node.getchild(2))
            value = node.getchild(2).getvalue()
            self.update_v_table(node.getchild(0).getdata(), value)
        elif node.getchild(1).getdata()[1] == '=':  # '+=' or '-='
            arg1 = self.v_table[node.getchild(0).getdata()]
            self.trans(node.getchild(2))
            arg2 = node.getchild(2).getvalue()
            op = node.getchild(1).getdata()[0]
            if op == '+':
                value = arg1 + arg2
            elif op == '-':
                value = arg1 - arg2
            self.update_v_table(node.getchild(0).getdata(), value)
    elif len(node.getchildren()) == 4:
        arg = self.v_table[node.getchild(0).getdata()]
        self.trans(node.getchild(1))
        index = int(node.getchild(1).getvalue())
        self.trans(node.getchild(3))
        value = node.getchild(3).getvalue()
        arg[index] = value
```

EXPRESSION

$$expr:$$
$$expression + term$$
$$|expression - term$$
$$|term$$

```
elif node.getdata() == '[EXPRESSION]':
    '''expr : expression '+' term
            | expression '-' term
            | term'''
    if len(node.getchildren()) == 3:
        self.trans(node.getchild(0))
        arg0 = node.getchild(0).getvalue()
        self.trans(node.getchild(2))
        arg1 = node.getchild(2).getvalue()
        op = node.getchild(1).getdata()
        if op == '+':
            value = arg0 + arg1
        elif op == '-':
            value = arg0 - arg1
        node.setvalue(value)
    elif len(node.getchildren()) == 1:  # term
        self.trans(node.getchild(0))
        value = node.getchild(0).getvalue()
        node.setvalue(value)
```

TERM

$$term:$$
$$term * factor$$
$$|term/factor$$
$$|factor$$

```
elif node.getdata() == '[TERM]':
    '''term : term '*' factor
            | term '/' factor
            | factor'''
    if len(node.getchildren()) == 3:
        self.trans(node.getchild(0))
        arg0 = node.getchild(0).getvalue()
        self.trans(node.getchild(2))
        arg1 = node.getchild(2).getvalue()
        op = node.getchild(1).getdata()
        if op == '*':
            value = arg0 + arg1
        elif op == '/':
            value = arg0 - arg1
        node.setvalue(value)
    elif len(node.getchildren()) == 1:
        self.trans(node.getchild(0))
        value = node.getchild(0).getvalue()
        node.setvalue(value)
```

FACTOR

$$factor:$$
$$NUMBER$$
$$|VARIABLE$$
$$|STR$$
$$|self$$
$$|VARIABLE[expression]$$
$$|(expression)$$

```
elif node.getdata() == '[FACTOR]':
    '''factor : NUMBER
```

```
                | VARIABLE
                | STR
                | self
                | VARIABLE '[' expression ']'
                | '(' expression ')' '''
    if len(node.getchildren()) == 1:
        if ord('0') <= ord(node.getchild(0).getdata()[0]) <= ord('9'):      #
NUMBER
            value = node.getchild(0).getvalue()
            node.setvalue(value)
        elif node.getchild(0).getdata()[0] == "'":                         # STR
            value = node.getchild(0).getdata()[1:-1]
            node.setvalue(value)
        elif node.getchild(0).getdata() == '[SELF]':                        #
self
            self.trans(node.getchild(0))
            value = self.v_table[node.getchild(0).getvalue()]
            node.setvalue(value)
        elif node.getchild(0).getdata() == '[EXPRESSION]':                  #
'(' expr ')'
            self.trans(node.getchild(0))
            value = node.getchild(0).getvalue()
            node.setvalue(value)
        else:                                                               #
VARIABLE
            value = self.v_table[node.getchild(0).getdata()]
            node.setvalue(value)
    elif len(node.getchildren()) == 2:
        arg = self.v_table[node.getchild(0).getdata()]
        self.trans(node.getchild(1))
        index = int(node.getchild(1).getvalue())
        value = arg[index]
        node.setvalue(value)
```

PRINT

$$print : PRINT(variables)$$

```
elif node.getdata() == '[PRINT]':
    '''print : PRINT '(' variables ')' '''
    self.trans(node.getchild(0))
    arg = node.getchild(0).getvalue()
    value = ''
    for i in range(len(arg)):
        value += str(self.v_table[arg[-1 - i]])
        value += ' '
    print(value)
```

FUNCTION

$$function : \\ DEFVARIABLE(variables)\{statements\} \\ |DEFVARIABLE(SELF)\{statements\} \\ |DEFVARIABLE(SELF, variables)\{statements\}$$

```
elif node.getdata() == '[FUNCTION]':
    '''function : DEF VARIABLE '(' variables ')' '{' statements '}'
                | DEF VARIABLE '(' SELF ')' '{' statements '}'
                | DEF VARIABLE '(' SELF ',' variables ')' '{' statements '}' '''
    if node.getchild(1).getdata() == '[VARIABLES]':
        fname = node.getchild(0).getdata()
        self.trans(node.getchild(1))
        vname = node.getchild(1).getvalue()
        self.f_table[fname] = (vname, node.getchild(2))  # function_name :
(variable_names, function)
    elif node.getchild(1).getdata() == '[SELF]':
        if len(node.getchildren()) == 3:
            fname = node.getchild(0).getdata()
            vname = []
            self.f_table[fname] = (vname, node.getchild(2))
        elif len(node.getchildren()) == 4:
            fname = node.getchild(0).getdata()
            self.trans(node.getchild(2))
            vname = node.getchild(2).getvalue()
            self.f_table[fname] = (vname, node.getchild(3))
```

Run_function

$$run_function : VARIABLE(expressions) |VARIABLE.VARIABLE(expressions)$$

```
elif node.getdata() == '[RUN_FUNCTION]':
    '''run_function : VARIABLE '(' expressions ')'
                    | VARIABLE '.' VARIABLE '(' expressions ')' '''
    if len(node.getchildren()) == 2:
        fname = node.getchild(0).getdata()
        self.trans(node.getchild(1))
        vname1 = node.getchild(1).getvalue()
        vname0, fnode = self.f_table[fname]
        t = Tran()
        for i in range(len(vname1)):
            t.v_table[vname0[i]] = vname1[i]
        value = t.trans(fnode)
        if isinstance(value, list):
            node.setvalue(value[1])
        print(t.v_table)
    elif len(node.getchildren()) == 3:
        variable = node.getchild(0).getdata()
        fname = node.getchild(1).getdata()
        self.trans(node.getchild(2))
        vname1 = node.getchild(2).getvalue()
        c = self.v_table[variable][1]
        vname0, fnode = c.f_table[fname]
        for i in range(len(vname1)):
            c.v_table[vname0[i]] = vname1[i]
        value = c.trans(fnode)
        if isinstance(value, list):
            node.setvalue(value[1])
        print(c.v_table)
```

Variables

$$variables : VARIABLE$$
$$|variables, VARIABLE$$
$$|self$$
$$|variables, self$$

```
elif node.getdata() == '[VARIABLES]':
    '''variables :
                 | VARIABLE
                 | variables ',' VARIABLE
                 | self
                 | variables ',' self'''
    if len(node.getchildren()) == 1:
        if node.getchild(0).getdata() == '[NONE]':  # NONE
            value = []
            node.setvalue(value)
        elif node.getchild(0).getdata() == '[SELF]':  # self
            self.trans(node.getchild(0))
            value = [node.getchild(0).getvalue()]
            # value = self.v_table[node.getchild(0).getdata()]
            node.setvalue(value)
        else:                                       # VARIABLE
            value = [node.getchild(0).getdata()]
            # value = self.v_table[node.getchild(0).getdata()]
            node.setvalue(value)
    elif len(node.getchildren()) == 2:
        if node.getchild(1).getdata() == '[SELF]':  # variables ',' self
            self.trans(node.getchild(0))
            value0 = node.getchild(0).getvalue()
            self.trans(node.getchild(1))
            value = [node.getchild(1).getvalue()]
            value.extend(value0)
            node.setvalue(value)
        else:                                       # variables ',' VARIABLE
            self.trans(node.getchild(0))
            value0 = node.getchild(0).getvalue()
            value = [node.getchild(1).getdata()]
            value.extend(value0)
            node.setvalue(value)
```

Expressions

$$expressions : expression$$
$$|expressions, expression$$

```
elif node.getdata() == '[EXPRESSIONS]':
    '''expressions :
                   | expression
                   | expressions ',' expression'''
    if len(node.getchildren()) == 1:
        if node.getchild(0).getdata() == '[NONE]':
            value = []
            node.setvalue(value)
        else:
            self.trans(node.getchild(0))
            value = [node.getchild(0).getvalue()]
            node.setvalue(value)
    elif len(node.getchildren()) == 2:
        self.trans(node.getchild(0))
        value0 = node.getchild(0).getvalue()
        self.trans(node.getchild(1))
        value = [node.getchild(1).getvalue()]
        value.extend(value0)
        node.setvalue(value)
```

Class

$class : CLASSVARIABLE(statesments)$

```
elif node.getdata() == '[CLASS]':
    '''class : CLASS VARIABLE '{' statements '}' '''
    if len(node.getchildren()) == 2:
        cname = node.getchild(0).getdata()
        t = Tran()
        t.trans(node.getchild(1))
        c_table[cname] = t
else:
    for c in node.getchildren():
        self.trans(c)
return node.getvalue()
```

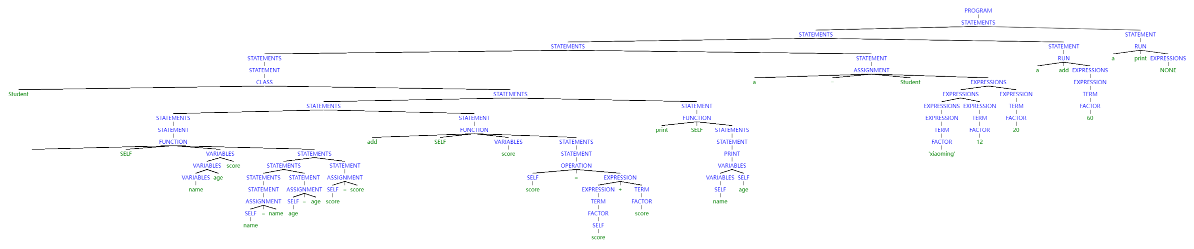
实验结果

主程序代码

```
from py_yacc import yacc
from util import clear_text
from translation import Tran
def translation(filename):
    text = clear_text(open(filename, 'r').read())
    def put2str(node):
        global res
        if node:
            data = str(node._data)
            data = data.replace("[", "").replace("]", "").replace("/'", "")
            res += data
        if node._children:
            for i in node._children:
                res += "["
                put2str(i)
                res += "]"
    # syntax parse
    root = yacc.parse(text)
    root.print_node(0)
    # translation
    t = Tran()
    t.trans(root)
    print(t.v_table)
    put2str(root)
    print("["+res+"]")
if __name__ == '__main__':
    res = ""
    translation("stu.py")
```

结果

字符串形式的语法树



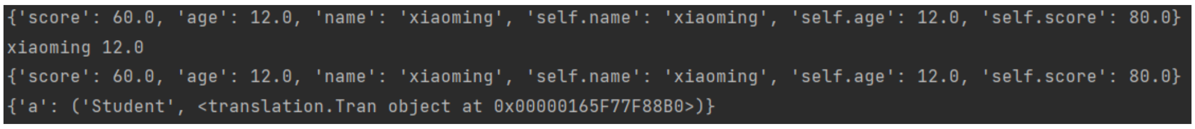
输出

```
+ [PROGRAM]
  + [STATEMENTS]
    + [STATEMENTS]
      + [STATEMENTS]
        + [STATEMENTS]
          + [STATEMENT]
            + [CLASS]
              + Student
              + [STATEMENTS]
                + [STATEMENTS]
                  + [STATEMENTS]
                    + [STATEMENT]
                      + [FUNCTION]
                        + __init__
                        + [SELF]
                        + [VARIABLES]
                          + [VARIABLES]
                            + [VARIABLES]
                              + name
                            + age
                          + score
                        + [STATEMENTS]
                          + [STATEMENTS]
                            + [STATEMENTS]
                              + [STATEMENT]
                                + [ASSIGNMENT]
                                  + [SELF]
                                    + name
                                  + =
                                  + name
                            + [STATEMENT]
                              + [ASSIGNMENT]
                                + [SELF]
                                  + age
                                + =
                                + age
                          + [STATEMENT]
                            + [ASSIGNMENT]
```

```
                                + [SELF]
                                  + score
                                + =
                                + score
                    + [STATEMENT]
                      + [FUNCTION]
                        + add_score
                        + [SELF]
                        + [VARIABLES]
                          + score
                        + [STATEMENTS]
                          + [STATEMENT]
                            + [OPERATION]
                              + [SELF]
                                + score
                              + =
                              + [EXPRESSION]
                                + [EXPRESSION]
                                  + [TERM]
                                    + [FACTOR]
                                      + [SELF]
                                        + score
                                + +
                                + [TERM]
                                  + [FACTOR]
                                    + score
                  + [STATEMENT]
                    + [FUNCTION]
                      + print_info
                      + [SELF]
                      + [STATEMENTS]
                        + [STATEMENT]
                          + [PRINT]
                            + [VARIABLES]
                              + [VARIABLES]
                                + [SELF]
                                  + name
                              + [SELF]
                                + age
          + [STATEMENT]
            + [ASSIGNMENT]
              + a
              + =
              + Student
              + [EXPRESSIONS]
                + [EXPRESSIONS]
                  + [EXPRESSIONS]
                    + [EXPRESSION]
                      + [TERM]
                        + [FACTOR]
                          + 'xiaoming'
                  + [EXPRESSION]
                    + [TERM]
                      + [FACTOR]
                        + 12
                + [EXPRESSION]
                  + [TERM]
                    + [FACTOR]
```

```
                      + 20
        + [STATEMENT]
          + [RUN_FUNCTION]
            + a
            + add_score
            + [EXPRESSIONS]
              + [EXPRESSION]
                + [TERM]
                  + [FACTOR]
                    + 60
      + [STATEMENT]
        + [RUN_FUNCTION]
          + a
          + print_info
          + [EXPRESSIONS]
            + [NONE]
```

翻译结果



```
{'score': 60.0, 'age': 12.0, 'name': 'xiaoming', 'self.name': 'xiaoming', 'self.age': 12.0, 'self.score': 80.0}
xiaoming 12.0
{'score': 60.0, 'age': 12.0, 'name': 'xiaoming', 'self.name': 'xiaoming', 'self.age': 12.0, 'self.score': 80.0}
{'a': ('Student', <translation.Tran object at 0x00000165F77F88B0>)}
```