DEPARTMENT OF COMPUTER SCIENCE DELHI UNIVERSITY

Compiler Design Parser Project

Topic : Python: Defining a List of Dictionaries

Submitted By : Praveen kumar

Submitted to: Dr. Ankit Rajpal

SYNTAX

syntax for list:

- list_name = [element1, element2,....] element can be
- character , string, integer , list , dictionary, tuples (each with nesting), objects, etc. .

syntax for simple list of dictionaries: list_name = [dict1,

- dict2,....] dict can be {} empty dictionary with nesting also.
- syntax for simple dictionary having key:value pairs. eg
- {key1:value1, key2:value2,..}

where key's must be unique and it should be in quotes (if it's character or string), it can be integer. same implies with values except it's not necessary to be unique.

ABOUT PARSER

tokens used in grammar:

```
{identifier}      { return IDENTIFIER; }
{number}          { return LITERAL; }
"="               { return EQUAL; }
";"               { return SEMICOLON; }
","               { return COMMA; }
"{"               { return LBRACES; }
"}"               { return RBRACES; }
"["               { return LSQBRACKET; }
"]"               { return RSQBRACKET; }
":"               { return COLON; }
"\n"              { return EOL; }
{string}          { return LITERAL; }
{ws}              { /* ignore whitespace */ }
.                 { /* ignore unknown characters */ }
```

ABOUT PARSER

context free grammar:

```
stmt :  listdict EOL { printf("Valid statement\n"); return 0; }
    |  listdict SEMICOLON EOL { printf("Valid statement\n"); return 0; }
    ;

listdict  : IDENTIFIER EQUAL LSQBRACKET dict RSQBRACKET
          ;

dict : single_dict | single_dict COMMA dict
     ;

single_dict:  LBRACES values RBRACES
    ;

values      : value_pair
            | value_pair COMMA values
            | /*empty*/;

list : LSQBRACKET mul_val RSQBRACKET

mul_val : LITERAL
        | LITERAL COMMA mul_val
        ;

value_pair : LITERAL COLON LITERAL
           | LITERAL COLON list
           ;
```

ASSUMPTIONS

assumptions:

- keys cannot be repeated with different or same values .

LEX FILE

```
/* Definition Section*/

%{
// file required by GCC yfile_name.tab.h that  contains the definitions that scanner needs.

#include "y.tab.h"
%}

/* Rules Section*/

identifier  [a-zA-Z_][a-zA-Z0-9_-]*
number      [0-9]+
string      (\"([^"\\]|\\.)*\")
ws          [ \t]+
/* returning specified tokens for matched strings based on rules defined on left side */

%%
{identifier}        { return IDENTIFIER; }
{number}            { return LITERAL; }
"="                 { return EQUAL; }
";"                 { return SEMICOLON; }
","                 { return COMMA; }
"{"                 { return LBRACES; }
"}"                 { return RBRACES; }
"["                 { return LSQBRACKET; }
"]"                 { return RSQBRACKET; }
":"                 { return COLON; }
"\n"                { return EOL; }
{string}            { return LITERAL; }
{ws}                { /* ignore whitespace */ }
.                   { /* ignore unknown characters */ }

%%

/* Auxiliary functions*/
//yywrap is called by lex when input is exhausted
//Return 1 if  done

int yywrap(){
   return 1;
}
```

YACC FILE

```
/* Definition section */


%{

#include <stdio.h>  // used for standard input output

#include <stdlib.h> //header file provides declarations for several functions that are used for memory allocation

#include <string.h> // header file provides declarations for several functions

extern int yylex(); // including this would eliminate the warnings about yylex

void yyerror(char *); //discussed below

%}


//defining tokens


%token IDENTIFIER EQUAL COMMA SEMICOLON LBRACES RBRACES LSQBRACKET RSQBRACKET COLON LITERAL EOL


//grammar productions and the actions for each production


%%

stmt :  listdict EOL { printf("Valid statement\n"); return 0; }

    |  listdict SEMICOLON EOL { printf("Valid statement\n"); return 0; }

    ;


listdict  : IDENTIFIER EQUAL LSQBRACKET dict RSQBRACKET
```

```
        ;


dict : single_dict | single_dict COMMA dict

    ;


single_dict:  LBRACES values RBRACES

   ;


values     : value_pair

           | value_pair COMMA values

           | /*empty*/;


list : LSQBRACKET mul_val RSQBRACKET


mul_val : LITERAL

       | LITERAL COMMA mul_val

       ;


value_pair : LITERAL COLON LITERAL

          | LITERAL COLON list

         ;


%%
```

```
//yyerror() function is called when all productions in the grammar in second section doesn't match to the input statement.


void yyerror(char *msg){
   printf("%s\n", msg);
}


int main(){


//call yyparse() to initiate the parsing process.
   yyparse();
   return 0;
}
```

TEST CASES:-

VALID TEST CASES:-

FOR EMPTY LIST :-

```
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp\Desktop\Compiler project>flex dictionaries.l

C:\Users\hp\Desktop\Compiler project>bison -dy dictionaries.y

C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
my_list = [{}]
Valid statement
```

KEY VALUE PAIR:-

```
C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
my_list = [{"name":"praveen","rollno":34}]
Valid statement

C:\Users\hp\Desktop\Compiler project>
```

MULTIPLE DICTIONARIES IN LIST:-

```
C:\Users\hp\Desktop\Compiler project>flex dictionaries.l

C:\Users\hp\Desktop\Compiler project>bison -dy dictionaries.y

C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
my_list = [{},{"name":"praveen"},{"rollno":34}]
Valid statement
```

STRING & INTEGER:-

```
C:\Users\hp\Desktop\Compiler project>flex dictionaries.l

C:\Users\hp\Desktop\Compiler project>bison -dy dictionaries.y

C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
my_list = [{8750087615:"phone number"}]
Valid statement
```

FOR EMPTY KEY WITH VALUE:-

```
C:\Users\hp\Desktop\Compiler project>flex dictionaries.l

C:\Users\hp\Desktop\Compiler project>bison -dy dictionaries.y

C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
my_list = [{"":"praveen"}]
Valid statement
```

FOR EMPTY KEY WITH EMPTY VALUE:-

```
C:\Users\hp\Desktop\Compiler project>flex dictionaries.l

C:\Users\hp\Desktop\Compiler project>bison -dy dictionaries.y

C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
my_list = [{"":""}]
Valid statement
```

INVALID TEST CASES:-

```
C:\Users\hp\Desktop\Compiler project>flex dictionaries.l

C:\Users\hp\Desktop\Compiler project>bison -dy dictionaries.y

C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
my_list = []
syntax error
```

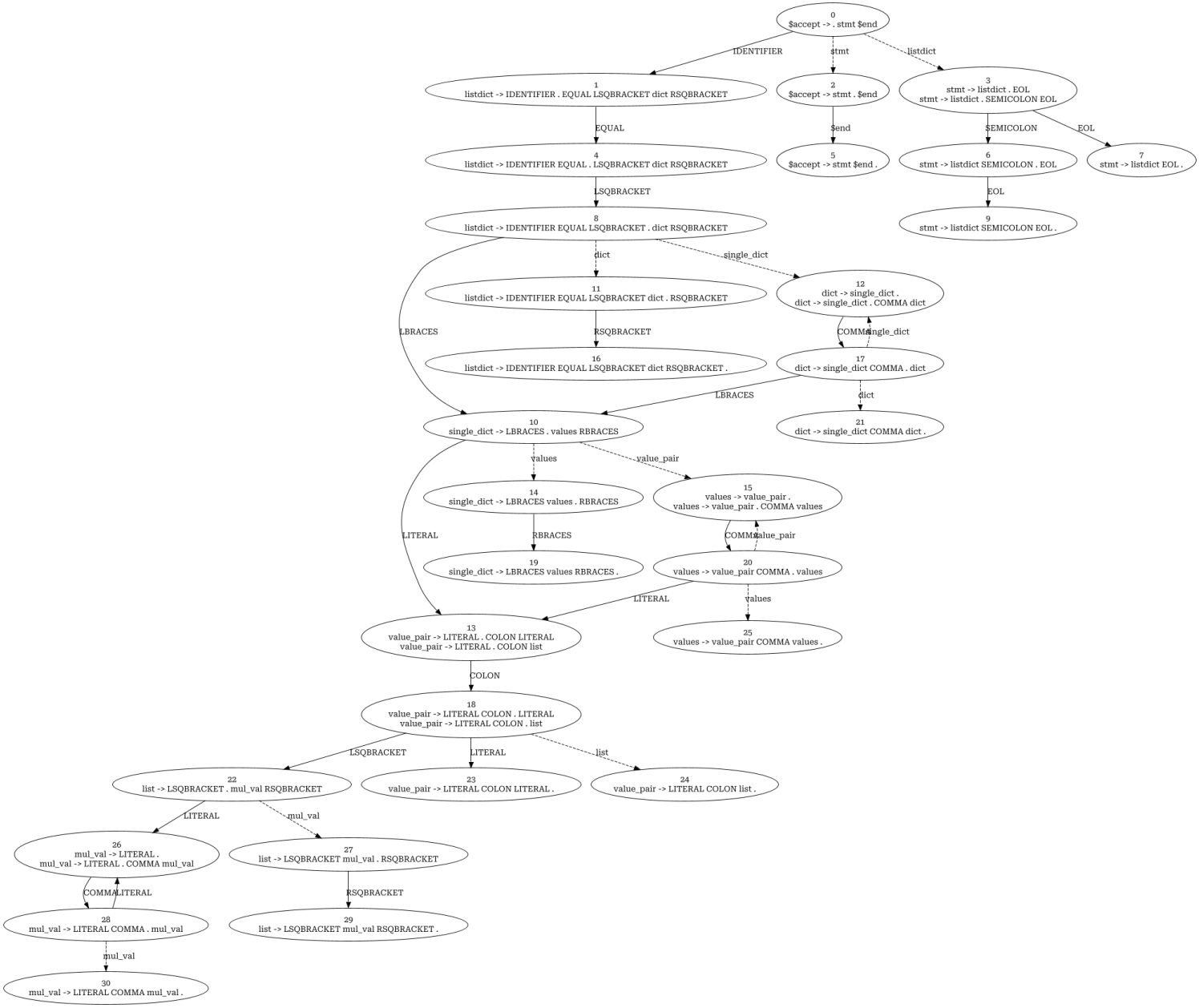
```
C:\Users\hp\Desktop\Compiler project>flex dictionaries.l

C:\Users\hp\Desktop\Compiler project>bison -dy dictionaries.y

C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
34pc = [{"P":34}]
syntax error
```

```
C:\Users\hp\Desktop\Compiler project>flex dictionaries.l

C:\Users\hp\Desktop\Compiler project>bison -dy dictionaries.y

C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
ab = [{'p':34},{'C':22}]
syntax error
```

```
C:\Users\hp\Desktop\Compiler project>flex dictionaries.l

C:\Users\hp\Desktop\Compiler project>bison -dy dictionaries.y

C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
my_list = [{"name":"praveen"},,{"rollno":34}]
syntax error
```

```
C:\Users\hp\Desktop\Compiler project>flex dictionaries.l

C:\Users\hp\Desktop\Compiler project>bison -dy dictionaries.y

C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
my_list = ["name":"praveen"@34}]
syntax error
```

```
C:\Users\hp\Desktop\Compiler project>flex dictionaries.l

C:\Users\hp\Desktop\Compiler project>bison -dy dictionaries.y

C:\Users\hp\Desktop\Compiler project>gcc lex.yy.c y.tab.c

C:\Users\hp\Desktop\Compiler project>a.exe
my_list = [{"name":"praveen",,"rollno":34}]
syntax error
```


0
$accept -> . stmt $end
IDENTIFIER
stmt
listdict
1
listdict -> IDENTIFIER . EQUAL LSQBRACKET dict RSQBRACKET
2
$accept -> stmt . $end
3
stmt -> listdict . EOL
stmt -> listdict . SEMICOLON EOL
EQUAL
$end
SEMICOLON
EOL
4
listdict -> IDENTIFIER EQUAL . LSQBRACKET dict RSQBRACKET
5
$accept -> stmt $end .
6
stmt -> listdict SEMICOLON . EOL
7
stmt -> listdict EOL .
LSQBRACKET
EOL
8
listdict -> IDENTIFIER EQUAL LSQBRACKET . dict RSQBRACKET
9
stmt -> listdict SEMICOLON EOL .
dict
single_dict
11
listdict -> IDENTIFIER EQUAL LSQBRACKET dict . RSQBRACKET
12
dict -> single_dict .
dict -> single_dict . COMMA dict
LBRACES
RSQBRACKET
COMMA
single_dict
16
listdict -> IDENTIFIER EQUAL LSQBRACKET dict RSQBRACKET .
17
dict -> single_dict COMMA . dict
LBRACES
dict
10
single_dict -> LBRACES . values RBRACES
21
dict -> single_dict COMMA dict .
values
value_pair
14
single_dict -> LBRACES values . RBRACES
15
values -> value_pair .
values -> value_pair . COMMA values
LITERAL
RBRACES
COMMA
value_pair
19
single_dict -> LBRACES values RBRACES .
20
values -> value_pair COMMA . values
LITERAL
values
13
value_pair -> LITERAL . COLON LITERAL
value_pair -> LITERAL . COLON list
25
values -> value_pair COMMA values .
COLON
18
value_pair -> LITERAL COLON . LITERAL
value_pair -> LITERAL COLON . list
LSQBRACKET
LITERAL
list
22
list -> LSQBRACKET . mul_val RSQBRACKET
23
value_pair -> LITERAL COLON LITERAL .
24
value_pair -> LITERAL COLON list .
LITERAL
mul_val
26
mul_val -> LITERAL .
mul_val -> LITERAL . COMMA mul_val
27
list -> LSQBRACKET mul_val . RSQBRACKET
COMMA
LITERAL
RSQBRACKET
28
mul_val -> LITERAL COMMA . mul_val
29
list -> LSQBRACKET mul_val RSQBRACKET .
mul_val
30
mul_val -> LITERAL COMMA mul_val .

CONCLUSION:-

In conclusion, the parser project aimed to create a program that can parse a list of dictionaries in Python and extract relevant information such as the keys, values, and their types. The project required a deep understanding of the Python programming language, regular expressions, and parsing techniques.

The project involved implementing a parser that can identify the list of dictionaries and parse them to extract relevant information. This was achieved through a combination of regular expressions, string manipulation techniques, and data type detection algorithms.

The project was successful in achieving its goal of creating a parser that can effectively extract information from a list of dictionaries in Python. This can be a useful tool for developers working with large datasets or JSON files, and needing to quickly extract relevant data.

However, there is always room for improvement, and further enhancements can be made to the parser's functionality and performance. For example, the parser could be expanded to handle more complex dictionaries with nested structures, or to extract additional information such as the length of the list of dictionaries.

In conclusion, the parser project to parse a list of dictionaries in Python was a valuable learning experience and has the potential to be further developed to meet the needs of various Python developers working with data-intensive applications.