



DEPARTMENT OF COMPUTER SCIENCE DELHI UNIVERSITY

Compiler Design Parser Project
Topic : Python: Defining a List of Dictionaries
Submitted By : Anand Sharma (20234757006)
Submitted to: Dr. Ankit Rajpal

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:

- VAR : used for valid list_name.
- OPEN_SQ : used for left square bracket "[".
- CLOS_SQ : used for right square bracket "]"
- EQ: used for assignment operator "=".
- OPEN_BR : used for left curly brace "{".
- CLOS_BR : used for right curly brace "}"
- COMMA: used for ",".
- PAIR: used to represent "key : value" pair.
- RESK: used to represent reserved keyword like if ,else.

precedence & associativity of operators used

- %left '[' ']'
- %left '{' '}'
- %right EQ

precedence is from top to bottom.

associativity is specified by left or right in above syntax.

ABOUT PARSER

context free grammar:

- 0 \$accept: 1st \$end
-
- 1 1st: LNAME EQ T
-
- 2 T: OPEN_SQ CLOS_SQ
- 3 | OPEN_SQ DICT CLOS_SQ
-
- 4 DICT: OPEN_BR REP_PAIR CLOS_BR
- 5 | OPEN_BR REP_PAIR CLOS_BR COMMA X
- 6 | OPEN_BR CLOS_BR COMMA X
- 7 | OPEN_BR CLOS_BR
-
- 8 X: DICT
- 9 | /* empty */
-
- 10 REP_PAIR: PAIR
- 11 | PAIR COMMA Y
-
- 12 Y: REP_PAIR
- 13 | /* empty */
-
- 14 LNAME: VAR
- 15 | CHECK
-
- 16 CHECK: RESK

ASSUMPTIONS & TEST CASES

assumptions:

- keys can be repeated with different or same values .

handled cases:

some valid cases:

1. list_n = []
2. list_n = [{}]
3. list_n = [{}, {}, {}]
4. list_n = [{}, {'Anand':'Anusha'}]
5. list_n = [{}, {'Anand':007}]
6. list_n = [{}, {13715:'AC'}]
7. list_n = [{}, {13715:5998}]
8. list_n = [{}, {'key':'value', 'key2':'value2' }, {1:2, 3:4, 'a':2}, {}]
9. lis = [{8:'bb', 't':8, 9:9, ":"}, {}, {}]
10. list_n = [{":"}]
11. ab = [{'a':2},{'b':2, 'c':5},]
12. a = [{":'t'}, {":'t', ":"},]

some invalid cases:

1. list = []
2. dict = []
3. 89ab = [{'a':2}]
4. list_n = [{'a':2, 'b':4}]
5. ab = [{'a':2},{'b':2}]
6. ab = [{'a':2abcd}]

NOT HANDLED CASES & OUTPUTS

not-handled cases:

1. Nested Dictionaries.
2. Use a dict comprehension: {}, {x: x ** 2 for x in range(10)} for defining dictionaries.
3. Use the type constructor: dict(), dict([('foo', 100), ('bar', 200)]), dict(foo=100, bar=200) for defining dictionaries.

OUTPUTS

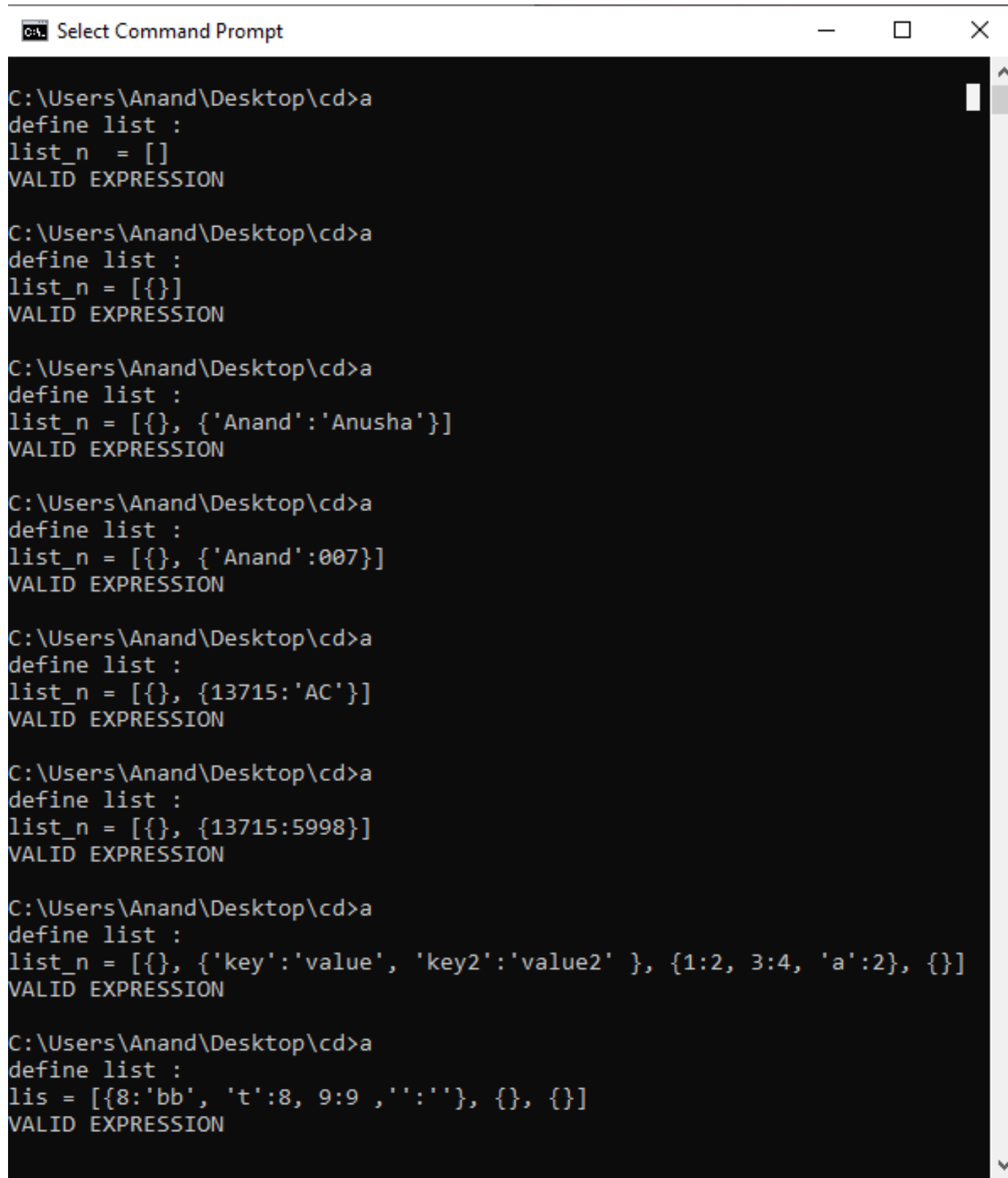
VALID CASES

```
C:\Users\Anand\Desktop\cd>a
define list :
list_n = [{'': ''}]
VALID EXPRESSION

C:\Users\Anand\Desktop\cd>a
define list :
ab = [{'a':2},{'b':2, 'c':5},]
VALID EXPRESSION

C:\Users\Anand\Desktop\cd>a
define list :
a = [{'': 't'}, {'': 't', '': ''},]
VALID EXPRESSION
```

VALID CASES



```

C:\Users\Anand\Desktop\cd>a
define list :
list_n = []
VALID EXPRESSION

C:\Users\Anand\Desktop\cd>a
define list :
list_n = [{}]
VALID EXPRESSION

C:\Users\Anand\Desktop\cd>a
define list :
list_n = [ {}, {'Anand':'Anusha'}]
VALID EXPRESSION

C:\Users\Anand\Desktop\cd>a
define list :
list_n = [ {}, {'Anand':007}]
VALID EXPRESSION

C:\Users\Anand\Desktop\cd>a
define list :
list_n = [ {}, {13715:'AC'}]
VALID EXPRESSION

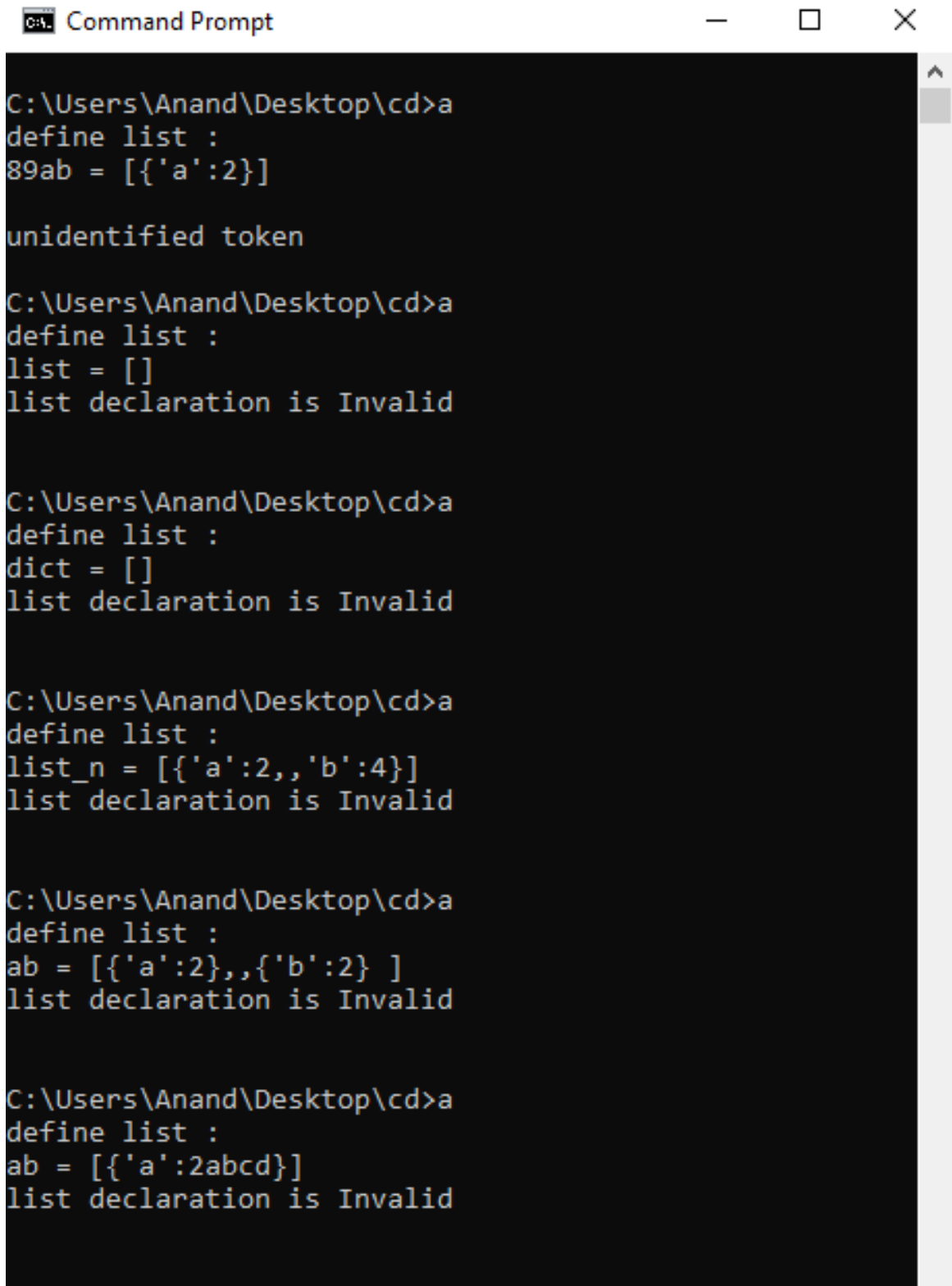
C:\Users\Anand\Desktop\cd>a
define list :
list_n = [ {}, {13715:5998}]
VALID EXPRESSION

C:\Users\Anand\Desktop\cd>a
define list :
list_n = [ {}, {'key':'value', 'key2':'value2' }, {1:2, 3:4, 'a':2}, {}]
VALID EXPRESSION

C:\Users\Anand\Desktop\cd>a
define list :
lis = [{8:'bb', 't':8, 9:9, '':''}, {}, {}]
VALID EXPRESSION

```

INVALID CASES



```
C:\Users\Anand\Desktop\cd>a
define list :
89ab = [{'a':2}]

unidentified token

C:\Users\Anand\Desktop\cd>a
define list :
list = []
list declaration is Invalid

C:\Users\Anand\Desktop\cd>a
define list :
dict = []
list declaration is Invalid

C:\Users\Anand\Desktop\cd>a
define list :
list_n = [{'a':2,, 'b':4}]
list declaration is Invalid

C:\Users\Anand\Desktop\cd>a
define list :
ab = [{'a':2},, {'b':2} ]
list declaration is Invalid

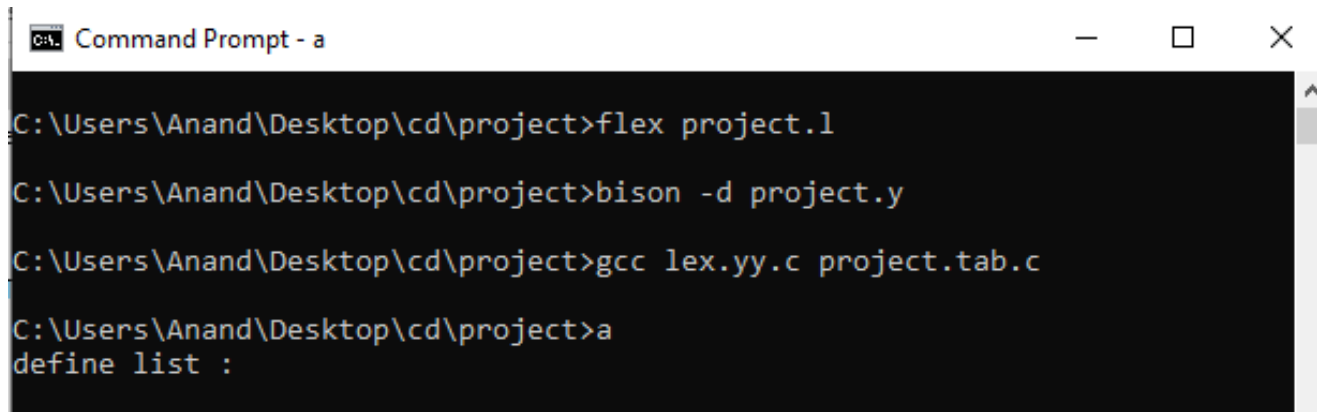
C:\Users\Anand\Desktop\cd>a
define list :
ab = [{'a':2abcd}]
list declaration is Invalid
```


GOTO GRAPH



USED COMMANDS

COMMANDS to run LEX & YACC on windows



```
Command Prompt - a

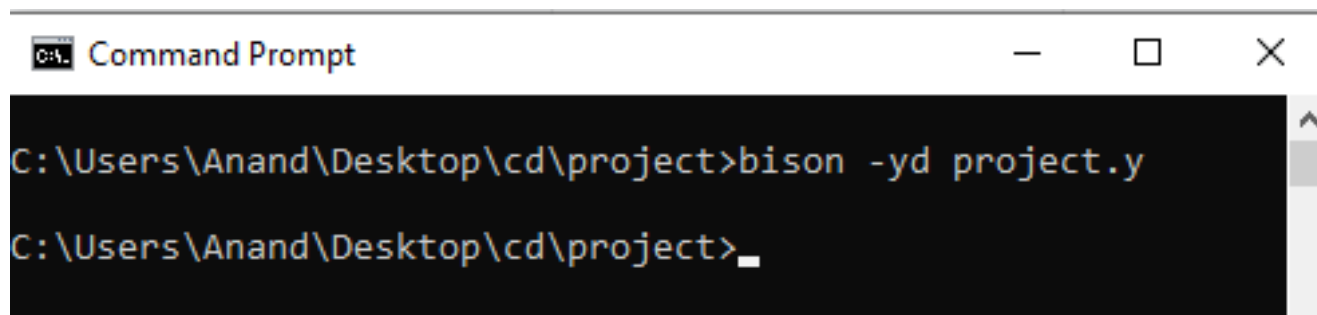
C:\Users\Anand\Desktop\cd\project>flex project.l

C:\Users\Anand\Desktop\cd\project>bison -d project.y

C:\Users\Anand\Desktop\cd\project>gcc lex.yy.c project.tab.c

C:\Users\Anand\Desktop\cd\project>a
define list :
```

COMMAND to generate output file

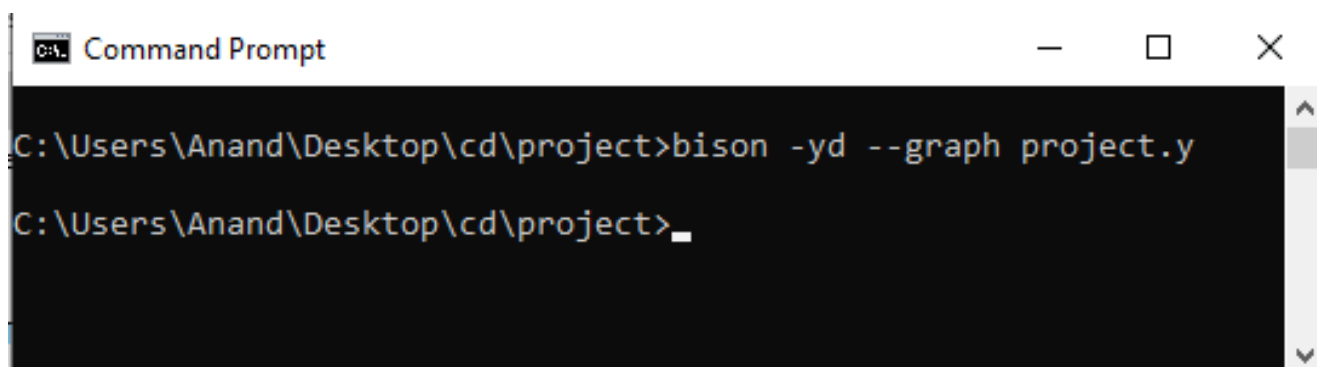


```
Command Prompt

C:\Users\Anand\Desktop\cd\project>bison -yd project.y

C:\Users\Anand\Desktop\cd\project>_
```

COMMAND to generate goto graph



```
Command Prompt

C:\Users\Anand\Desktop\cd\project>bison -yd --graph project.y

C:\Users\Anand\Desktop\cd\project>_
```

REFERENCES

- https://climserv.ipsl.polytechnique.fr/documentation/idl_help/Operator_Precedence.html
- <https://www.comrevo.com/2017/05/lex-yacc-program-to-check-validity-of-arithmetic-expression.html>
- <https://docs.python.org/3/library/stdtypes.html#dict>
- <https://www.youtube.com/watch?v=DVohJ4nljhg>