Lexical rules

1. Comments in Python begin with "#" and continue to the end of the line.
2. Lines containing only blanks, tabs, and comments are ignored entirely (they have no effect on indentation level). We call these blank lines hereafter.
3. Blanks and tabs may separate lexical tokens. They are required only to separate two tokens only when they could be interpreted differently if concatenated together (e.g. "if" followed by "x" could be interpreted as "ifx" without intervening space). Blanks and tabs may not appear within individual lexical tokens (described in this section) other than string literals.
4. Blanks and tabs immediately at the beginnings of lines are significant. The amount of indentation of a line is the equivalent number of blanks at the beginning of that line. A tab following the equivalent of N blanks is equivalent to 8-(N mod 8) blanks, so that each tab brings the equivalent number of blanks up to the next multiple of 8.
5. If one line is indented more than the preceding non-blank line, it is taken to be preceded by an INDENT token. It is an error for the first non-blank line in a file to be indented.
6. If one line is indented less than the preceding non-blank line, it is taken to be preceded by enough DEDENT tokens to match all unmatched INDENT tokens introduced by preceding more-indented lines. The end of a file is preceded by enough DEDENT tokens to match all unmatched INDENT tokens.

Lexical tokens

```
NEWLINE: the new-line character
ENDMARKER: the end of file token
NUMBER: is defined by regular expression [0-9]+
ID: is defined by regular expression [_a-zA-Z][_a-zA-Z0-9]*
STRING: a squence of characters enclosed by either single or double quotes
```

Grammar notations

1. `{ rule }*` means any number of applications of *rule*, including no application (zero application) of *rule*.
2. `{ rule }+` means one or more applications of *rule*.

3. `[ rule ]` means *rule* is optional.

4. `{( or-separated-entities ) rule }` means if one of the *or-separated-entities* is present, then *rule* applies.

5. Strings with single quotes around them are terminals.

Grammar rules

```
file_input: {NEWLINE | stmt}* ENDMARKER

stmt: simple_stmt | compound_stmt
simple_stmt: (print_stmt | assign_stmt | array_ops | call_stmt | return_stmt) NEWLINE
print_stmt: 'print' [ testlist ]
assign_stmt: ID '=' ( test | array_init )
compound_stmt: if_stmt | for_stmt | func_def
call_stmt: call
return_stmt: 'return' [test]

testlist: test {',' test}*

suite: NEWLINE INDENT stmt+ DEDENT
func_suite: NEWLINE INDENT (stmt | return_stmt)+ DEDENT
if_stmt: 'if' test ':' suite {'elif' test ':' suite}* ['else' ':' suite]
for_stmt: 'for' ID 'in' 'range' '(' testlist ')' ':' suite

func_def: 'def' ID '(' [parameter_list] ')' ':' func_suite
call: ID '(' { testlist } ')'
parameter_list: ID {, ID}*

subscription: ID '[' test ']'
array_init: '[' [ testlist ] ']
array_ops: ID '.' ( 'append' | 'pop' ) '(' test ')'

array_len: 'len' '(' ID ')'
```

```
test: or_test
or_test: and_test {'or' and_test}*
and_test: not_test {'and' not_test}*
not_test: 'not' not_test | comparison
comparison: arith_expr {comp_op arith_expr}*

comp_op: '<'|'>'|'=='|'>='|'<='|'<>'|'!='

arith_expr: term {('+'|'-') term}*
term: factor {('*'|'/'|'%'|'//') factor}*
factor: {'-'} factor | atom | call | subscription | array_len
atom: ID | NUMBER | STRING | '(' test ')'
```