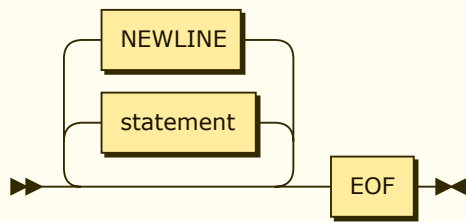


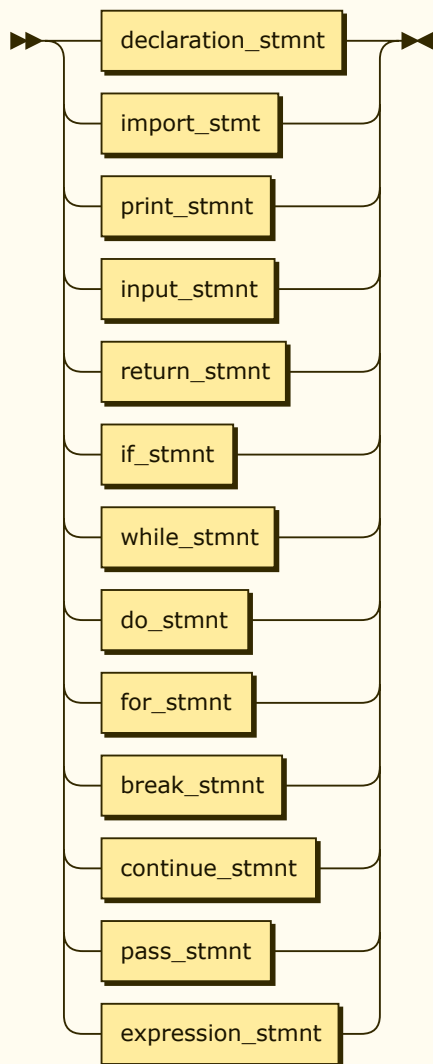
program:



```
program ::= ( statement | NEWLINE )* EOF
```

no references

statement:

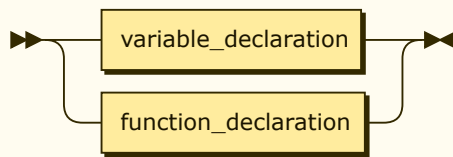


```
statement
  ::= declaration_stmtnt
  | import_stmt
  | print_stmtnt
  | input_stmtnt
  | return_stmtnt
  | if_stmtnt
  | while_stmtnt
  | do_stmtnt
  | for_stmtnt
  | break_stmtnt
  | continue_stmtnt
  | pass_stmtnt
  | expression_stmtnt
```

referenced by:

- [block](#)
- [program](#)

declaration_stmt:

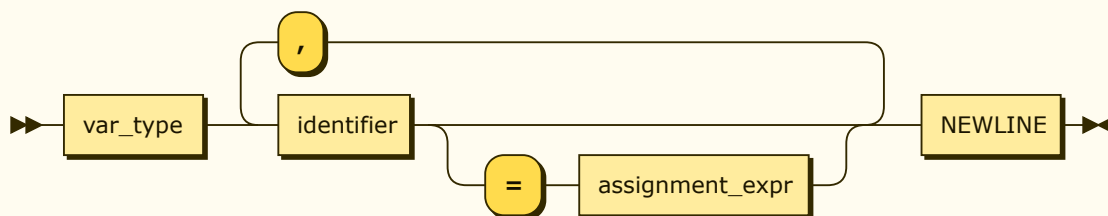


```
declaration_stmt
  ::= variable_declaration
     | function_declaration
```

referenced by:

- [statement](#)

variable_declaration:

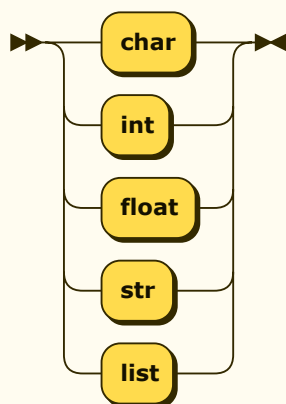


```
variable_declaration
  ::= var_type identifier ( '=' assignment_expr )? ( ',' identifier ( '=' assignment_expr )? )* NEWLINE
```

referenced by:

- [declaration_stmt](#)

var_type:

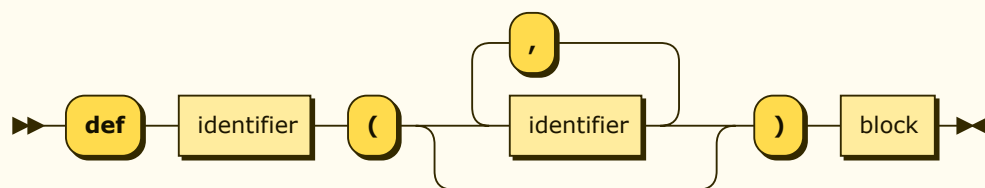


```
var_type ::= 'char'
          | 'int'
          | 'float'
          | 'str'
          | 'list'
```

referenced by:

- variable declaration

function_declaration:

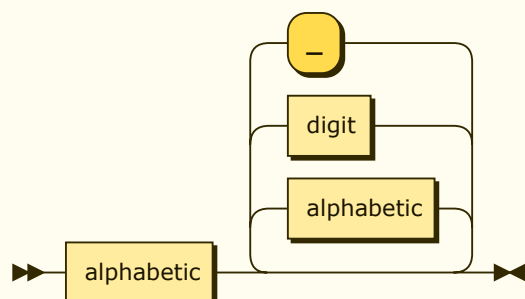


```
function_declaration
    ::= 'def' identifier '(' ( identifier ( ',' identifier )* )? ')' block
```

referenced by:

- declaration_stmtnt

identifier:

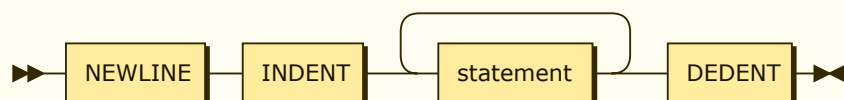


```
identifier
    ::= alphabetic ( alphabetic | digit | '_' )*
```

referenced by:

- function_call
- function_declaration
- input_stmtnt
- variable_declaration

block:

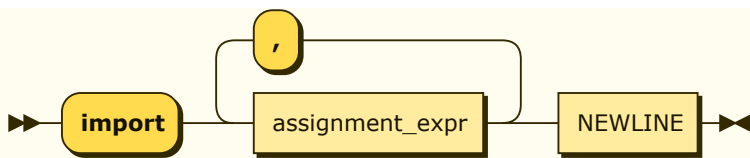


```
block    ::= NEWLINE INDENT statement+ DEDENT
```

referenced by:

- do_stmtnt
- function_declaration
- if_stmtnt
- while_stmtnt

import_stmt:

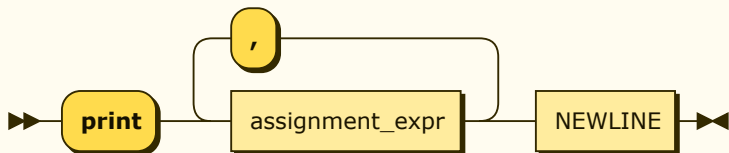


```
import_stmt
    ::= 'import' assignment_expr ( ',' assignment_expr )* NEWLINE
```

referenced by:

- [statement](#)

print_stmt:

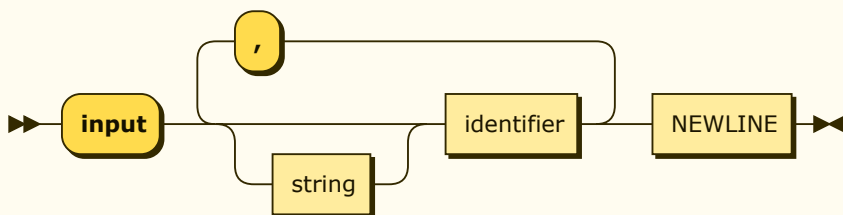


```
print_stmt
    ::= 'print' assignment_expr ( ',' assignment_expr )* NEWLINE
```

referenced by:

- [statement](#)

input_stmt:

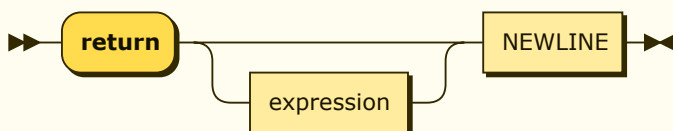


```
input_stmt
    ::= 'input' string? identifier ( ',' string? identifier )* NEWLINE
```

referenced by:

- [statement](#)

return_stmt:

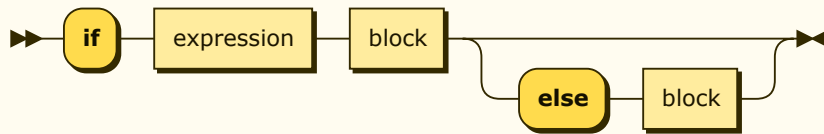


```
return_stmt
    ::= 'return' expression? NEWLINE
```

referenced by:

- [statement](#)

if_stmt:



```
if_stmt ::= 'if' expression block ( 'else' block )?
```

referenced by:

- [statement](#)

while_stmt:



```
while_stmt  
    ::= 'while' expression block
```

referenced by:

- [statement](#)

do_stmt:

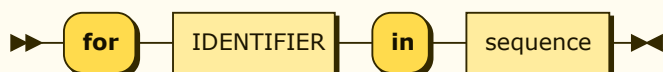


```
do_stmt ::= 'do' block 'while' expression NEWLINE
```

referenced by:

- [statement](#)

for_stmt:



```
for_stmt  
    ::= 'for' IDENTIFIER 'in' sequence
```

referenced by:

- [statement](#)

break_stmt:



```
break_stmt  
    ::= 'break' NEWLINE
```

referenced by:

- [statement](#)

continue_stmt:

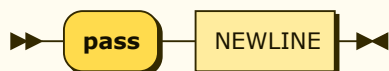


```
continue_stmt  
    ::= 'continue' NEWLINE
```

referenced by:

- [statement](#)

pass_stmt:



```
pass_stmt  
    ::= 'pass' NEWLINE
```

referenced by:

- [statement](#)

expression_stmt:

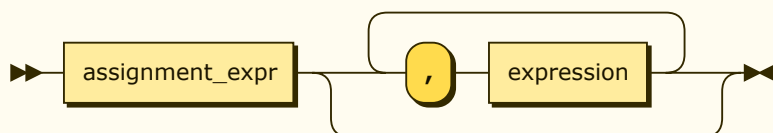


```
expression_stmt  
    ::= expression NEWLINE
```

referenced by:

- [statement](#)

expression:

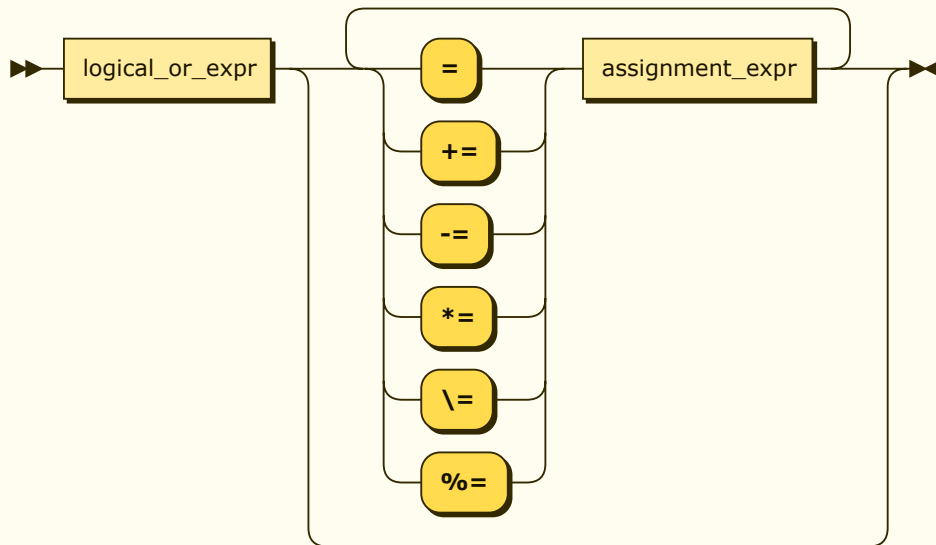


```
expression  
    ::= assignment_expr ( ',' expression )*
```

referenced by:

- [do_stmt](#)
- [expression](#)
- [expression_stmt](#)
- [if_stmt](#)
- [primary_expr](#)
- [return_stmt](#)
- [while_stmt](#)

assignment_expr:

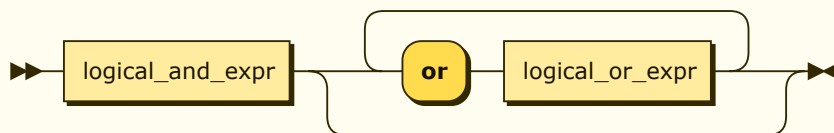


```
assignment_expr
    ::= logical_or_expr ( ( '=' | '+=' | '-=' | '*=' | '\\=' | '%=' ) assignment_expr )*
```

referenced by:

- [assignment_expr](#)
- [expression](#)
- [function_call](#)
- [import_stmt](#)
- [list_const](#)
- [print_stmt](#)
- [variable_declaration](#)

logical_or_expr:

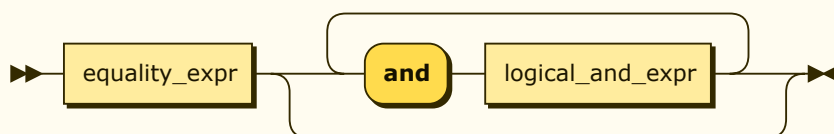


```
logical_or_expr
    ::= logical_and_expr ( 'or' logical_or_expr )*
```

referenced by:

- [assignment_expr](#)
- [index](#)
- [list_append](#)
- [list_insert](#)
- [logical_or_expr](#)
- [slice](#)

logical_and_expr:

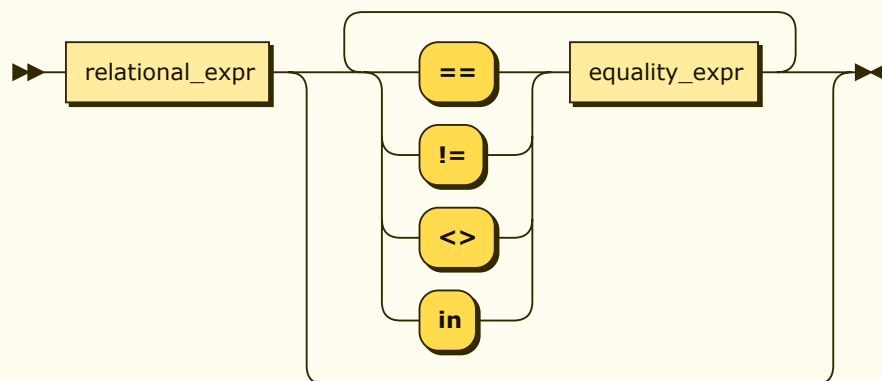


```
logical_and_expr
    ::= equality_expr ( 'and' logical_and_expr )*
```

referenced by:

- logical and expr
- logical or expr

equality_expr:

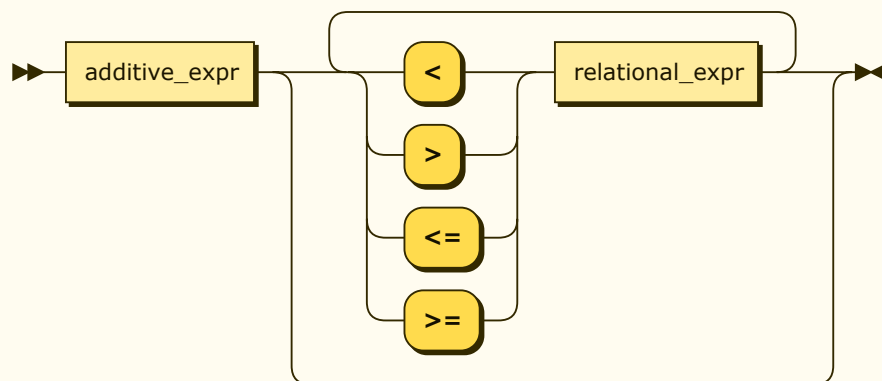


```
equality_expr
    ::= relational_expr ( ( '=' | '!=' | '<>' | 'in' ) equality_expr )*
```

referenced by:

- equality_expr
- logical and expr

relational_expr:

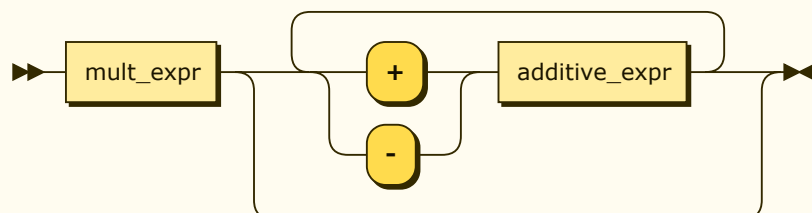


```
relational_expr
    ::= additive_expr ( ( '<' | '>' | '<=' | '>=' ) relational_expr )*
```

referenced by:

- equality_expr
- relational_expr

additive_expr:

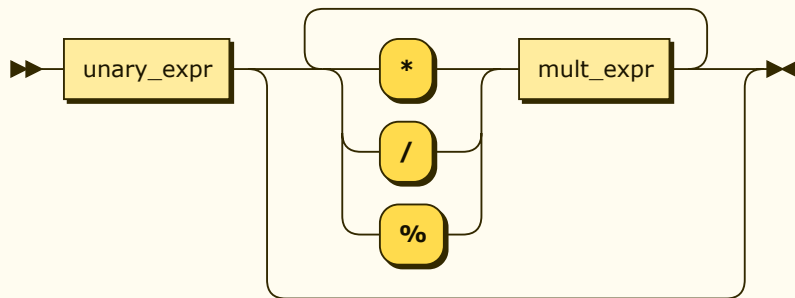


```
additive_expr
    ::= mult_expr ( ( '+' | '-' ) additive_expr )*
```


referenced by:

- additive_expr
- relational_expr

mult_expr:

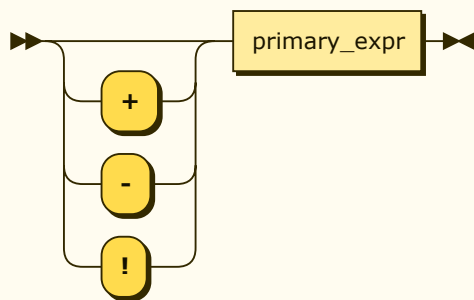


```
mult_expr
  ::= unary_expr ( ( '*' | '/' | '%' ) mult_expr )*
```

referenced by:

- additive_expr
- mult_expr

unary_expr:

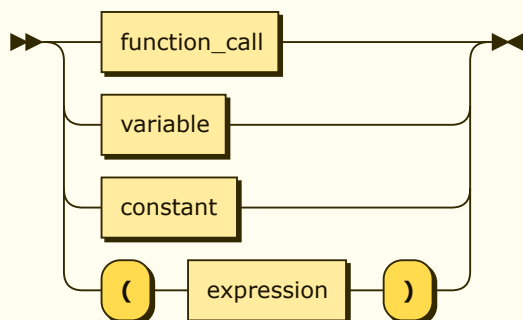


```
unary_expr
  ::= ( '+' | '-' | '!' )? primary_expr
```

referenced by:

- mult_expr

primary_expr:



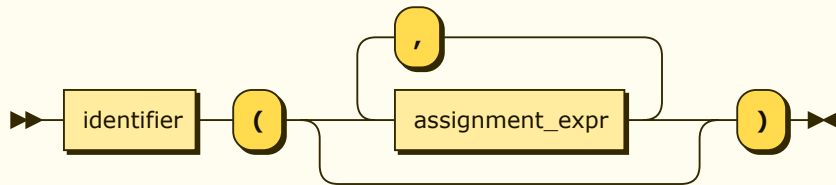
```
primary_expr
  ::= function_call
  | variable
```

```
| constant  
| '(' expression ')'
```

referenced by:

- unary_expr

function_call:

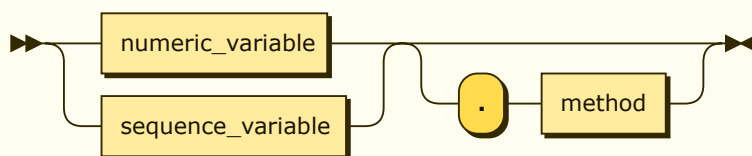


```
function_call  
  ::= identifier '(' ( assignment_expr ( ',' assignment_expr )* )? ')'
```

referenced by:

- primary_expr

variable:

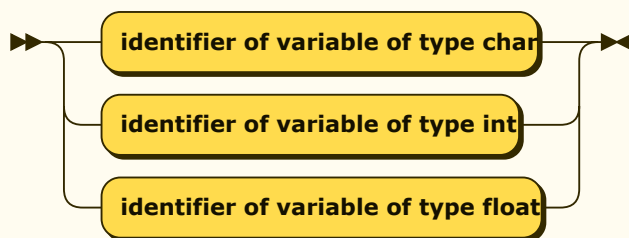


```
variable ::= ( numeric_variable | sequence_variable ) ( '.' method )?
```

referenced by:

- primary_expr

numeric_variable:

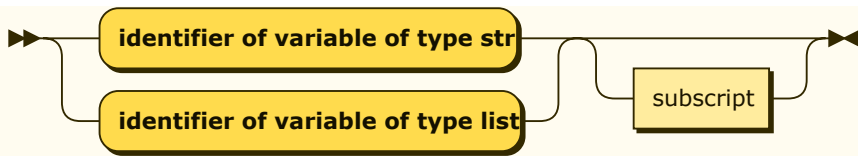


```
numeric_variable  
  ::= 'identifier of variable of type char'  
  | 'identifier of variable of type int'  
  | 'identifier of variable of type float'
```

referenced by:

- variable

sequence_variable:

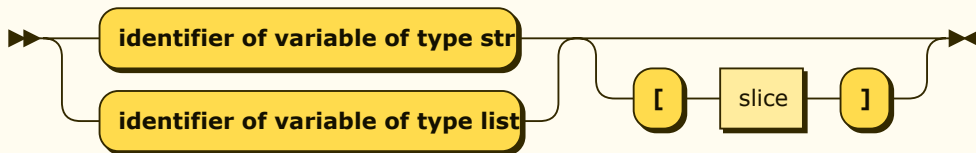


```
sequence_variable
    ::= ( 'identifier of variable of type str' | 'identifier of variable of type list' ) subscript?
```

referenced by:

- [variable](#)

sequence:

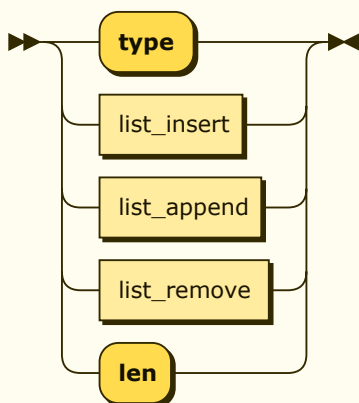


```
sequence ::= ( 'identifier of variable of type str' | 'identifier of variable of type list' ) ( '[' slice ']' )?
```

referenced by:

- [for_stmt](#)

method:



```
method ::= 'type'
        | list_insert
        | list_append
        | list_remove
        | 'len'
```

referenced by:

- [variable](#)

list_insert:

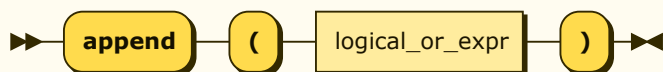


```
list_insert
    ::= 'insert' '(' index ',' logical_or_expr ')'
```

referenced by:

- [method](#)

list_append:

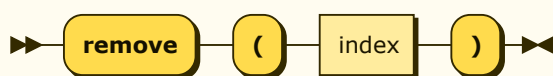


```
list_append
    ::= 'append' '(' logical_or_expr ')'
```

referenced by:

- [method](#)

list_remove:

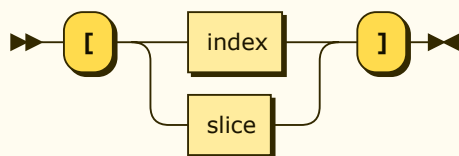


```
list_remove
    ::= 'remove' '(' index ')'
```

referenced by:

- [method](#)

subscript:

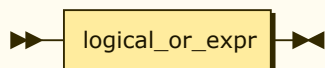


```
subscript
    ::= '[' ( index | slice ) ']'
```

referenced by:

- [sequence_variable](#)

index:

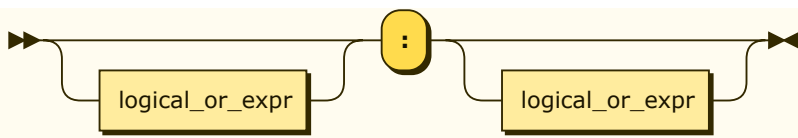


```
index    ::= logical_or_expr
```

referenced by:

- [list_insert](#)
- [list_remove](#)
- [subscript](#)

slice:

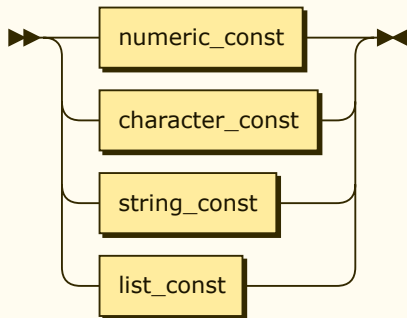


```
slice ::= logical_or_expr? ':' logical_or_expr?
```

referenced by:

- [sequence](#)
- [subscript](#)

constant:

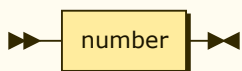


```
constant ::= numeric_const
           | character_const
           | string_const
           | list_const
```

referenced by:

- [primary_expr](#)

numeric_const:

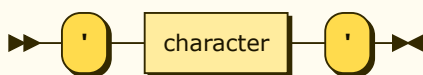


```
numeric_const ::= number
```

referenced by:

- [constant](#)

character_const:

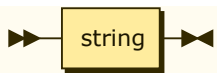


```
character_const ::= "'" character "'"
```

referenced by:

- [constant](#)

string_const:

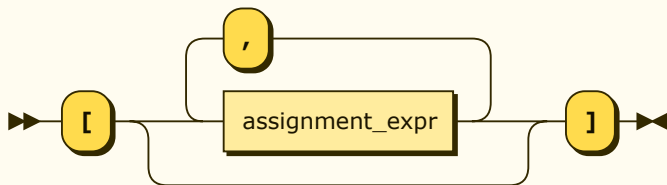


```
string_const
    ::= string
```

referenced by:

- [constant](#)

list_const:

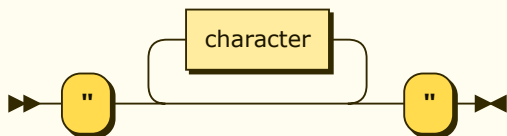


```
list_const
    ::= '[' ( assignment_expr ( ',' assignment_expr )* )? ']'
```

referenced by:

- [constant](#)

string:

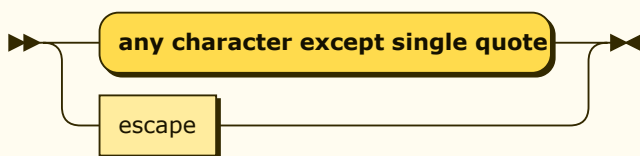


```
string    ::= '"' character* '"'
```

referenced by:

- [input_stmt](#)
- [string_const](#)

character:

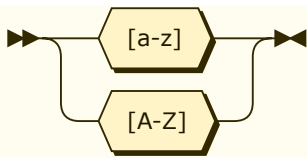


```
character
    ::= 'any character except single quote'
    | escape
```

referenced by:

- [character_const](#)
- [string](#)

alphabetic:

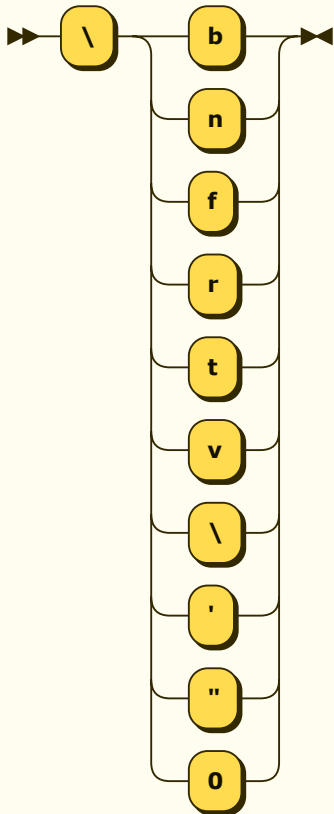


```
alphabetic  
  ::= [a-zA-Z]
```

referenced by:

- identifier

escape:

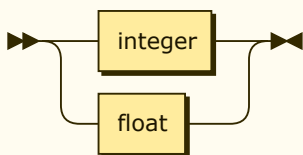


```
escape  ::= '\\' [bnfrtv\\'\"0]
```

referenced by:

- character

number:

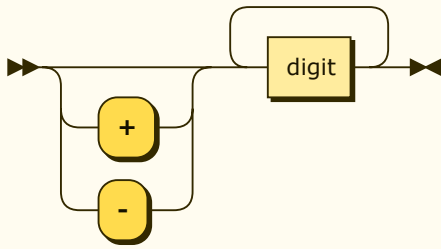


```
number  ::= integer  
          | float
```

referenced by:

- numeric const

integer:

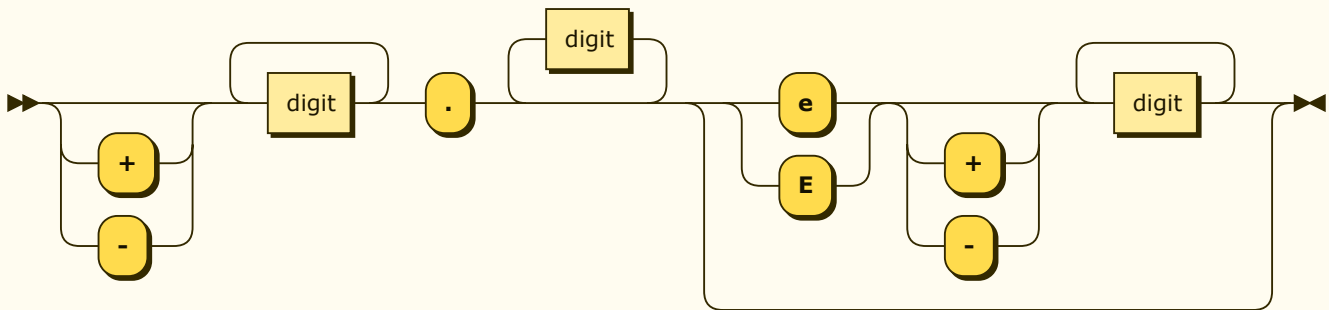


```
integer ::= ( '+' | '-' )? digit+
```

referenced by:

- [number](#)

float:

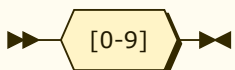


```
float ::= ( '+' | '-' )? digit+ '.' digit* ( ( 'e' | 'E' ) ( '+' | '-' )? digit+ )?
```

referenced by:

- [number](#)

digit:



```
digit ::= [0-9]
```

referenced by:

- [float](#)
- [identifier](#)
- [integer](#)