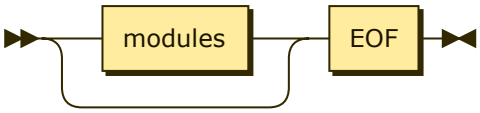
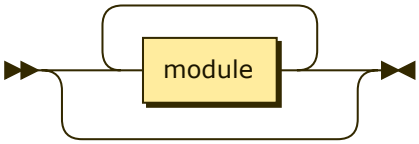
start:




```
start    ::= modules? EOF
```

no references

modules:

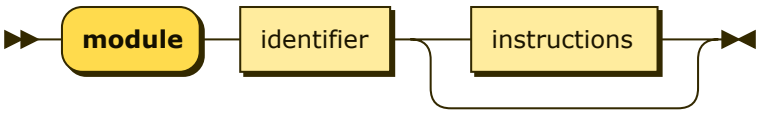



```
modules  ::= module*
```

referenced by:

- start

module:

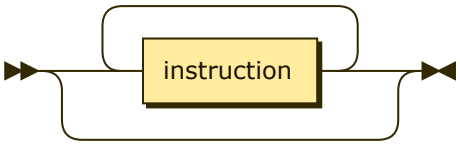



```
module   ::= 'module' identifier instructions?
```

referenced by:

- modules

instructions:

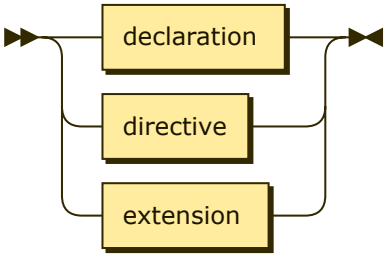



```
instructions
         ::= instruction*
```

referenced by:

- module

instruction:

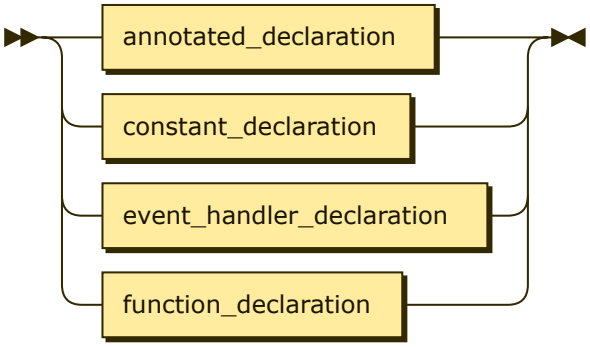



```
instruction
          ::= declaration
            | directive
            | extension
```

referenced by:

- instructions

declaration:

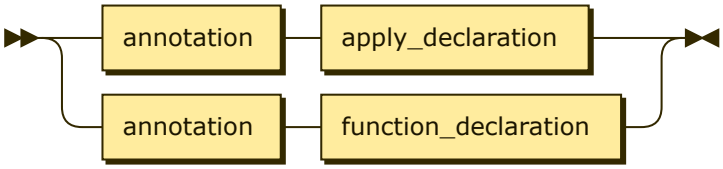



```
declaration
          ::= annotated_declaration
            | constant_declaration
            | event_handler_declaration
            | function_declaration
```

referenced by:

- instruction

annotated_declaration:




```
annotated_declaration
          ::= annotation apply_declaration
            | annotation function_declaration
```

referenced by:

- declaration

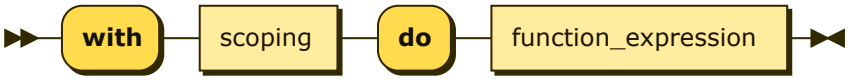
annotation:


```
annotation
         ::= '@' function_call_expression
```

referenced by:

- annotated_declaration

apply_declaration:

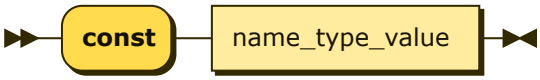



```
apply_declaration
         ::= 'with' scoping 'do' function_expression
```

referenced by:

- annotated_declaration

constant_declaration:




```
constant_declaration
         ::= 'const' name_type_value
```

referenced by:

- declaration

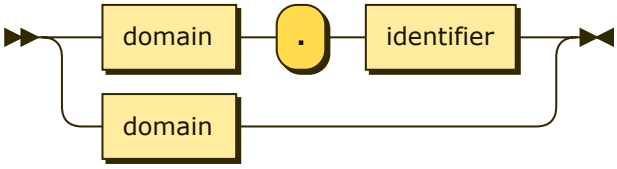
event_handler_declaration:


```
event_handler_declaration
         ::= event_timing scoping function_expression 'do' function_expression
```

referenced by:

- declaration

scoping:




```
scoping  ::= domain '.' identifier
           | domain
```

referenced by:

- apply_declaration
- event_handler_declaration

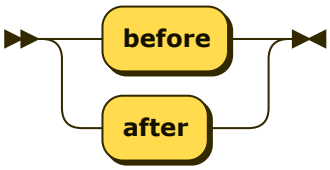
domain:


```
domain   ::= identifier
```

referenced by:

- extension
- scoping

event_timing:

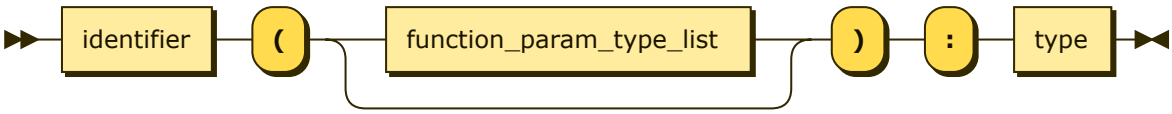



```
event_timing
         ::= 'before'
           | 'after'
```

referenced by:

- event_handler_declaration

function_prototype:

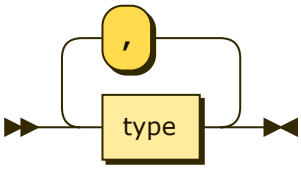



```
function_prototype
         ::= identifier '(' function_param_type_list? ')' ':' type
```

referenced by:

- import_directive

function_param_type_list:

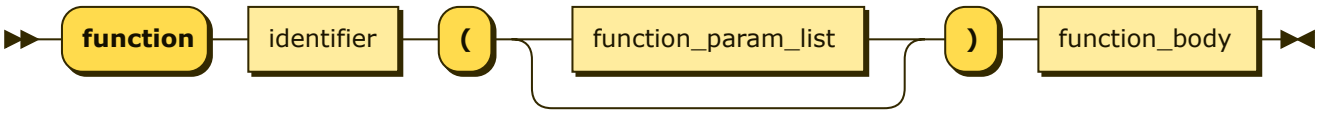



```
function_param_type_list
         ::= type ( ',' type )*
```

referenced by:

- function_prototype

function_declaration:

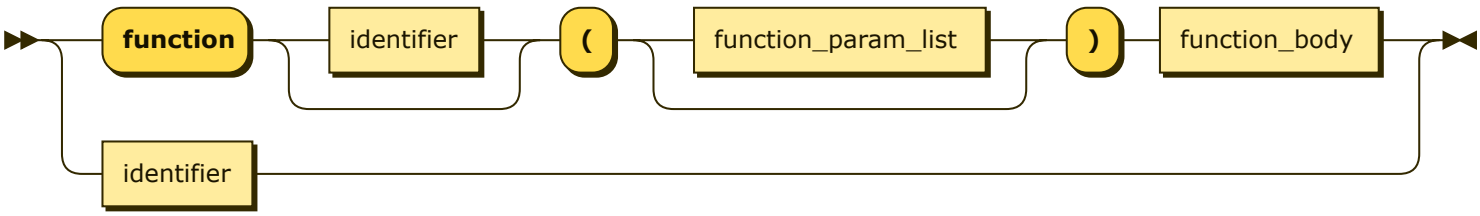



```
function_declaration
         ::= 'function' identifier '(' function_param_list? ')' function_body
```

referenced by:

- annotated_declaration
- declaration

function_expression:

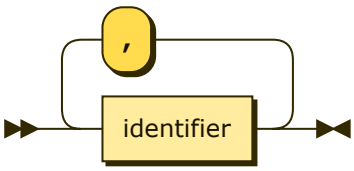



```
function_expression
         ::= 'function' identifier? '(' function_param_list? ')' function_body
           | identifier
```

referenced by:

- apply_declaration
- event_handler_declaration

function_param_list:




```
function_param_list
```

```
          ::= identifier ( ',' identifier )*
```

referenced by:

- function_declaration
- function_expression

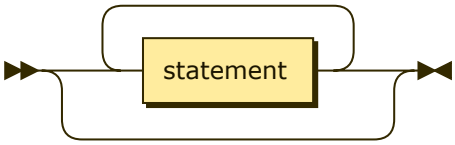
function_body:


```
function_body
          ::= block_statement
```

referenced by:

- function_declaration
- function_expression

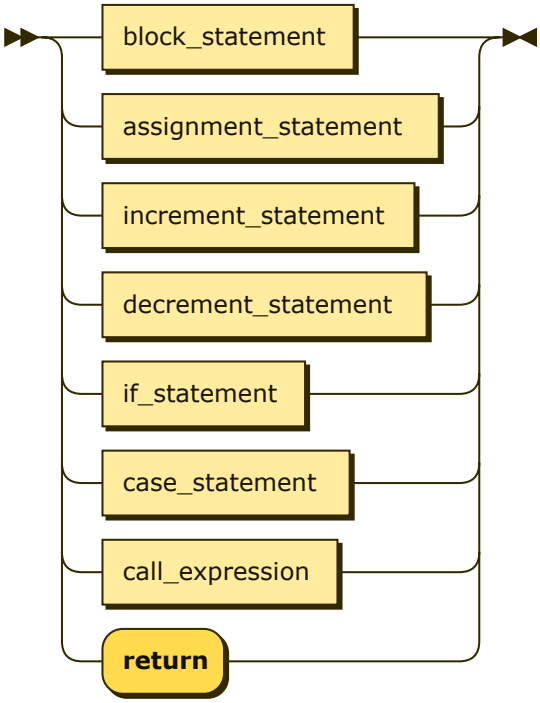
statements:




```
statements
          ::= statement*
```

no references

statement:

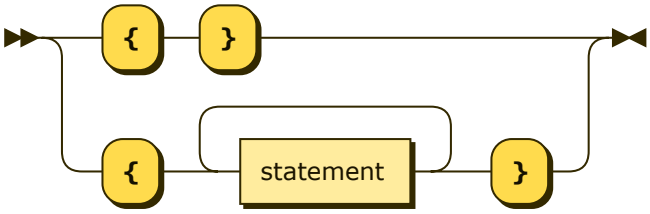


```
statement
          ::= block_statement
            | assignment_statement
            | increment_statement
            | decrement_statement
            | if_statement
            | case_statement
            | call_expression
            | 'return'
```

referenced by:

- block_statement
- case_clause
- if_statement
- statements

block_statement:

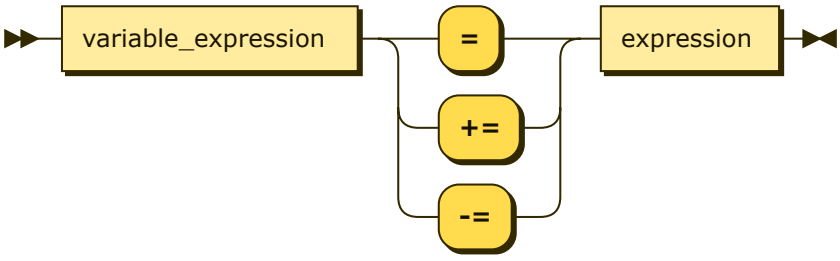



```
block_statement
          ::= '{' '}'
            | '{' statement+ '}'
```

referenced by:

- case_clause
- function_body
- statement

assignment_statement:




```
assignment_statement
          ::= variable_expression ( '=' | '+=' | '-=' ) expression
```

referenced by:

- statement

increment_statement:

```
increment_statement
        ::= variable_expression '++'
```

referenced by:

- statement

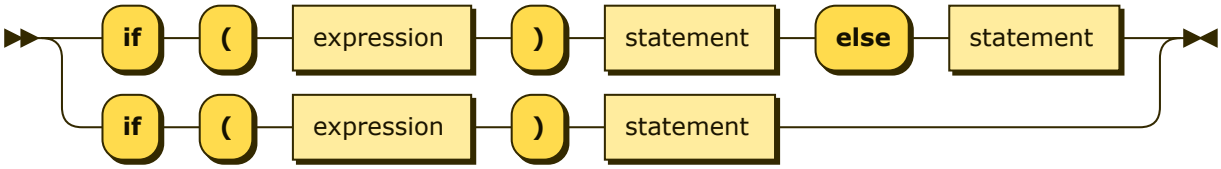
decrement_statement:


```
decrement_statement
        ::= variable_expression '--'
```

referenced by:

- statement

if_statement:

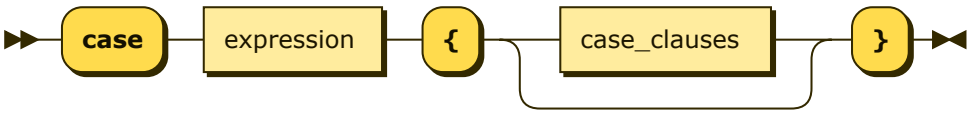



```
if_statement
        ::= 'if' '(' expression ')' statement 'else' statement
          | 'if' '(' expression ')' statement
```

referenced by:

- statement

case_statement:

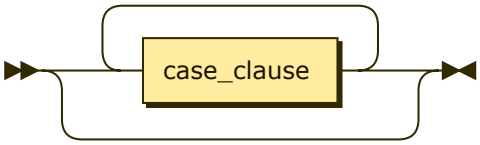



```
case_statement
        ::= 'case' expression '{' case_clauses? '}'
```

referenced by:

- statement

case_clauses:

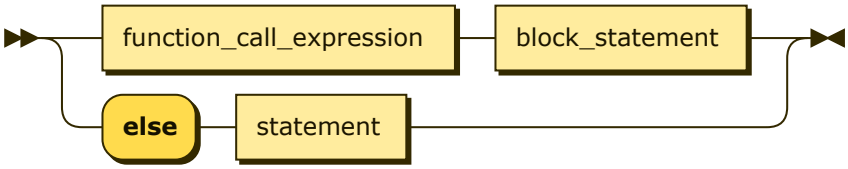


```
case_clauses
        ::= case_clause*
```

referenced by:

- case_statement

case_clause:




```
case_clause
        ::= function_call_expression block_statement
          | 'else' statement
```

referenced by:

- case_clauses

expression:


```
expression
        ::= logical_expression
```

referenced by:

- argument_list
- assignment_statement
- case_statement
- comparison
- if_statement
- list_literal
- name_type_exp

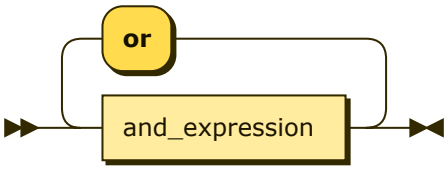
logical_expression:


```
logical_expression
        ::= or_expression
```

referenced by:

- expression
- primary_expression

or_expression:

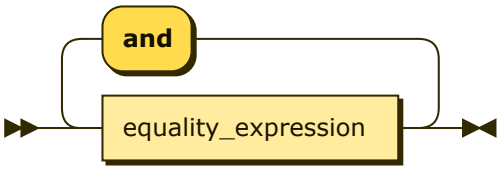



```
or_expression
        ::= and_expression ( 'or' and_expression )*
```

referenced by:

- logical_expression

and_expression:

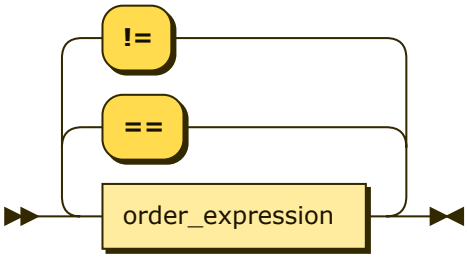



```
and_expression
        ::= equality_expression ( 'and' equality_expression )*
```

referenced by:

- or_expression

equality_expression:

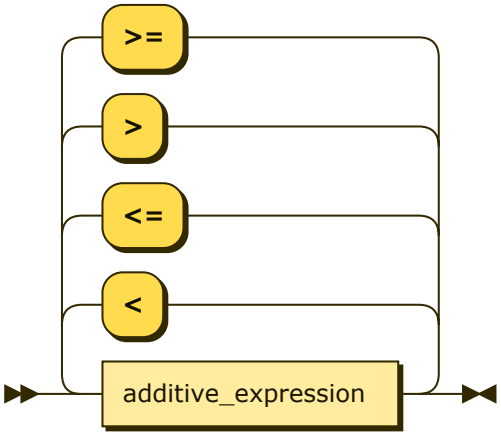



```
equality_expression
        ::= order_expression ( ( '==' | '!=' ) order_expression )*
```

referenced by:

- and_expression

order_expression:

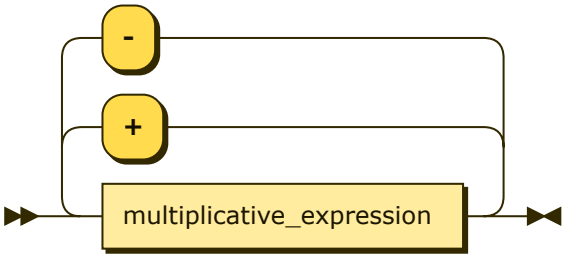



```
order_expression
         ::= additive_expression ( ( '<' | '<=' | '>' | '>=' ) additive_expression )*
```

referenced by:

- equality_expression

additive_expression:

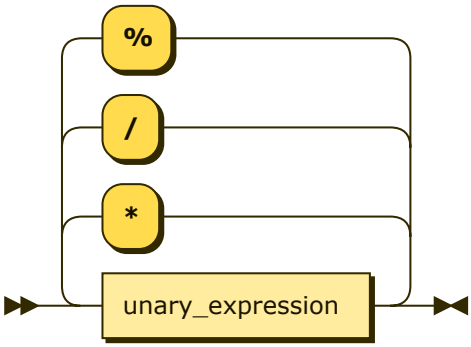



```
additive_expression
         ::= multiplicative_expression ( ( '+' | '-' ) multiplicative_expression )*
```

referenced by:

- order_expression

multiplicative_expression:

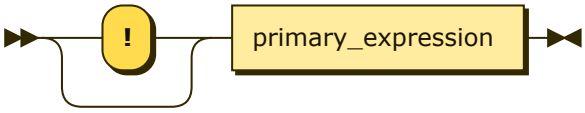



```
multiplicative_expression
         ::= unary_expression ( ( '*' | '/' | '%' ) unary_expression )*
```

referenced by:

- additive_expression

unary_expression:

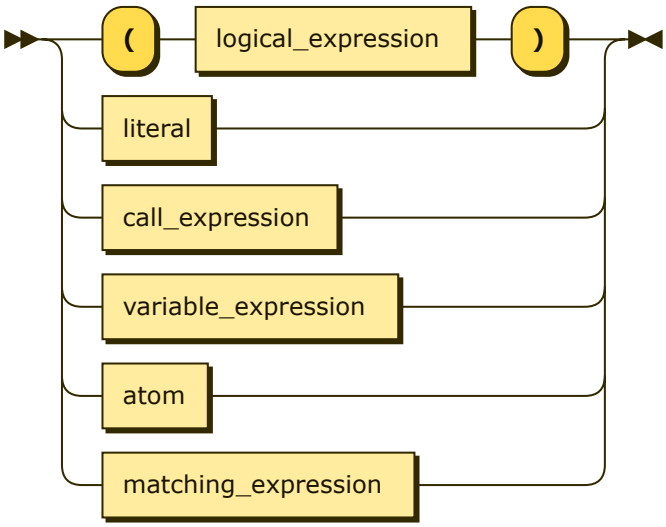



```
unary_expression
          ::= '!'? primary_expression
```

referenced by:

- multiplicative_expression

primary_expression:

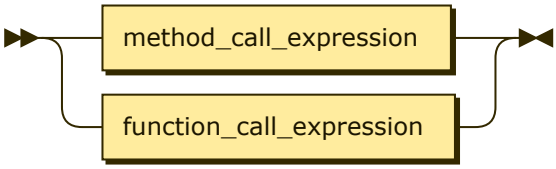



```
primary_expression
          ::= '(' logical_expression ')'
            | literal
            | call_expression
            | variable_expression
            | atom
            | matching_expression
```

referenced by:

- unary_expression

call_expression:

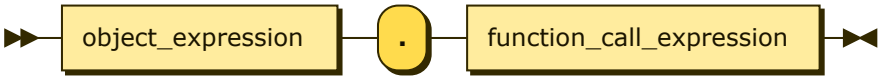



```
call_expression
          ::= method_call_expression
            | function_call_expression
```

referenced by:

- primary_expression
- statement

method_call_expression:

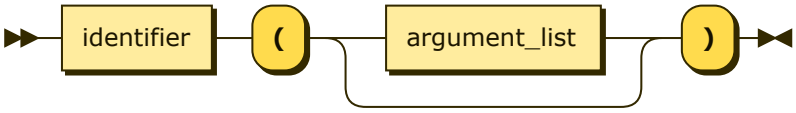



```
method_call_expression
        ::= object_expression '.' function_call_expression
```

referenced by:

- call_expression

function_call_expression:

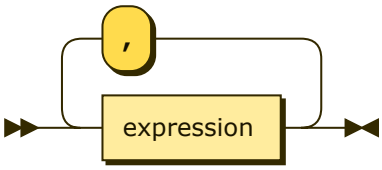



```
function_call_expression
        ::= identifier '(' argument_list? ')'
```

referenced by:

- annotation
- call_expression
- case_clause
- method_call_expression

argument_list:

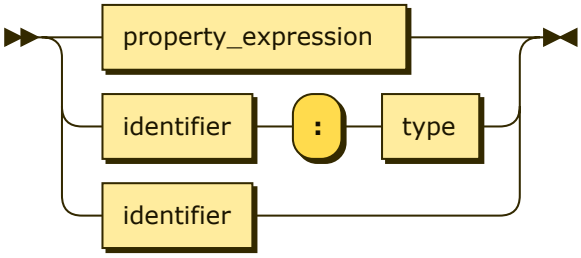



```
argument_list
        ::= expression ( ',' expression )*
```

referenced by:

- function_call_expression

variable_expression:

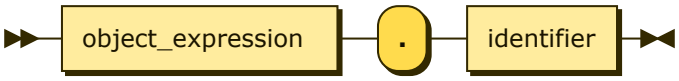



```
variable_expression
         ::= property_expression
           | identifier ':' type
           | identifier
```

referenced by:

- assignment_statement
- decrement_statement
- increment_statement
- primary_expression

property_expression:

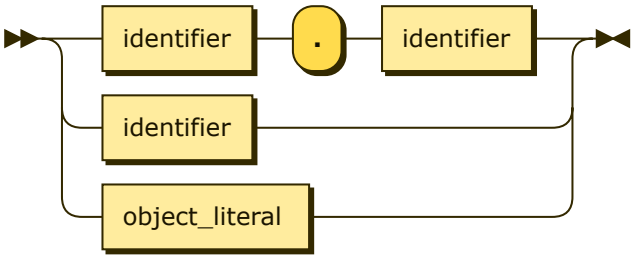



```
property_expression
         ::= object_expression '.' identifier
```

referenced by:

- variable_expression

object_expression:




```
object_expression
         ::= identifier '.' identifier
           | identifier
           | object_literal
```

referenced by:

- method_call_expression
- property_expression

directive:

```
directive
        ::= import_directive
```

referenced by:

- instruction

import_directive:


```
import_directive
        ::= 'from' identifier 'import' function_prototype
```

referenced by:

- directive

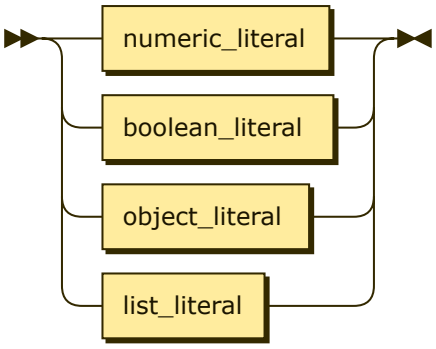
extension:


```
extension
        ::= 'extend' domain 'with' object_literal
```

referenced by:

- instruction

literal:

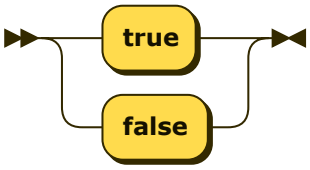



```
literal  ::= numeric_literal
           | boolean_literal
           | object_literal
           | list_literal
```

referenced by:

- name_type_value
- primary_expression

boolean_literal:

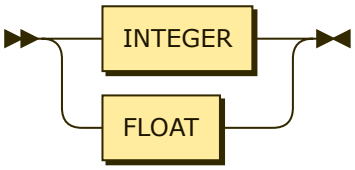



```
boolean_literal
         ::= 'true'
           | 'false'
```

referenced by:

- literal

numeric_literal:

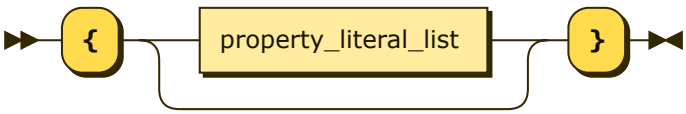



```
numeric_literal
         ::= INTEGER
           | FLOAT
```

referenced by:

- literal

object_literal:

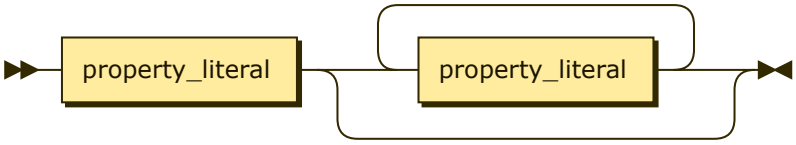



```
object_literal
         ::= '{' property_literal_list? '}'
```

referenced by:

- extension
- literal
- object_expression

property_literal_list:

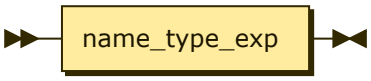



```
property_literal_list
         ::= property_literal property_literal*
```

referenced by:

- object_literal

property_literal:

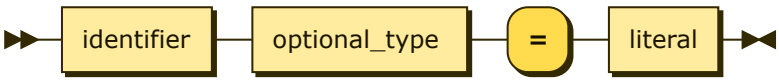



```
property_literal
         ::= name_type_exp
```

referenced by:

- property_literal_list

name_type_value:




```
name_type_value
         ::= identifier optional_type '=' literal
```

referenced by:

- constant_declaration

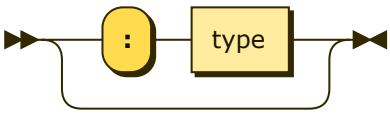
name_type_exp:


```
name_type_exp
         ::= identifier optional_type '=' expression
```

referenced by:

- property_literal

optional_type:



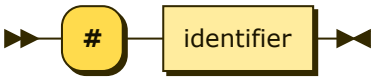

```
optional_type
         ::= ':' type
           |
```

referenced by:

- name_type_exp

- name_type_value

atom:

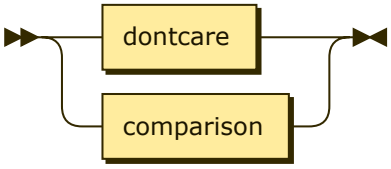



```
atom     ::= '#' identifier
```

referenced by:

- primary_expression

matching_expression:

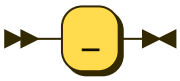



```
matching_expression
         ::= dontcare
           | comparison
```

referenced by:

- primary_expression

dontcare:




```
dontcare ::= '_'
```

referenced by:

- matching_expression

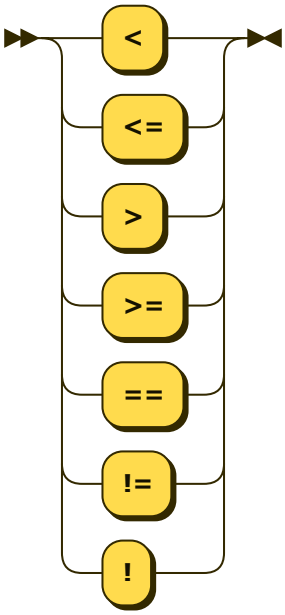
comparison:


```
comparison
         ::= comparator expression
```

referenced by:

- matching_expression

comparator:

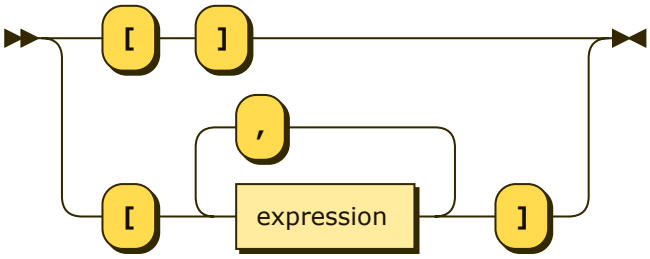



```
comparator
          ::= '<'
            | '<='
            | '>'
            | '>='
            | '=='
            | '!='
            | '!'
```

referenced by:

- comparison

list_literal:

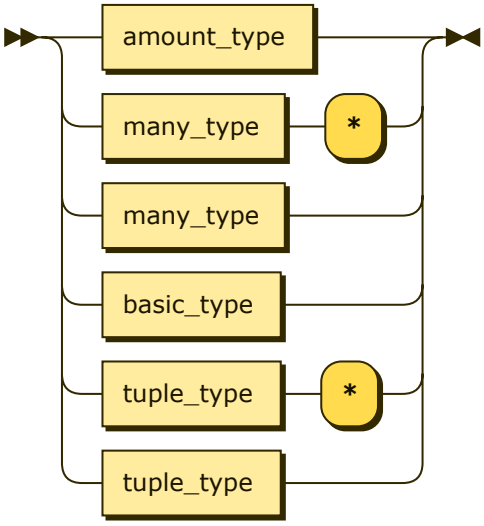



```
list_literal
          ::= '[' ']'
            | '[' expression ( ',' expression )* ']'
```

referenced by:

- literal

type:




```
type      ::= amount_type
            | many_type '*'
            | many_type
            | basic_type
            | tuple_type '*'
            | tuple_type
```

referenced by:

- function_param_type_list
- function_prototype
- optional_type
- tuple_type
- variable_expression

many_type:


```
many_type
          ::= basic_type '*'
```

referenced by:

- type

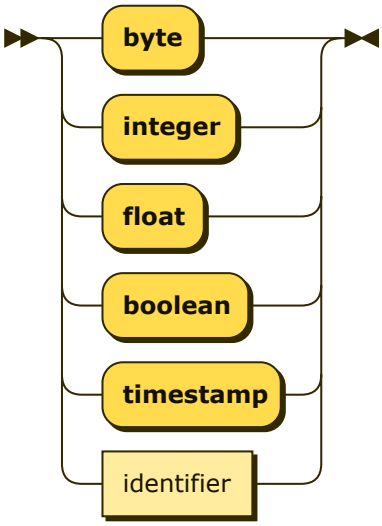
basic_type:


```
basic_type
          ::= type_identifier
```

referenced by:

- amount_type
- many_type
- type

type_identifier:

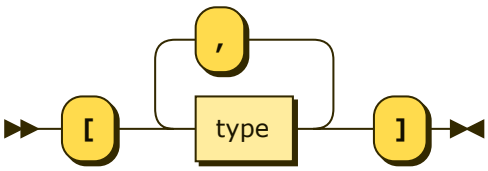



```
type_identifier
          ::= 'byte'
            | 'integer'
            | 'float'
            | 'boolean'
            | 'timestamp'
            | identifier
```

referenced by:

- basic_type

tuple_type:




```
tuple_type
          ::= '[' type ( ',' type )* ']'
```

referenced by:

- type

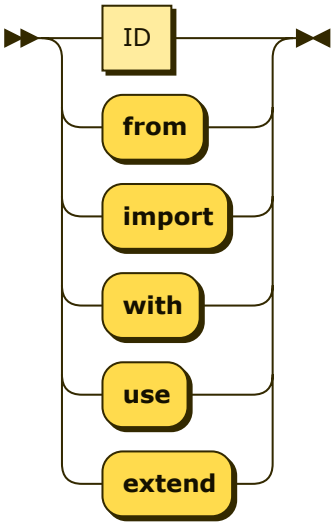
amount_type:


```
amount_type
          ::= basic_type '[' INTEGER ']'
```

referenced by:

- type

identifier:

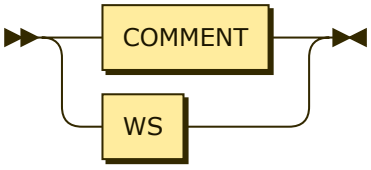



```
identifier
          ::= ID
            | 'from'
            | 'import'
            | 'with'
            | 'use'
            | 'extend'
```

referenced by:

- atom
- domain
- function_call_expression
- function_declaration
- function_expression
- function_param_list
- function_prototype
- import_directive
- module
- name_type_exp
- name_type_value
- object_expression
- property_expression
- scoping
- type_identifier
- variable_expression

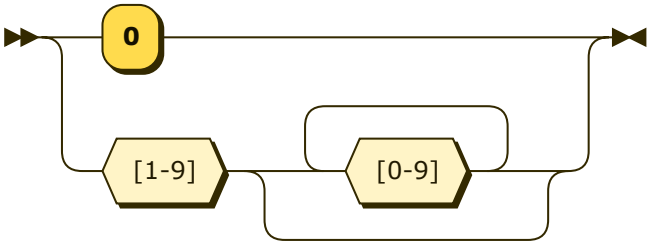
_:




```
_         ::= COMMENT
            | WS
           /* ws: definition */
```

no references

INTEGER:

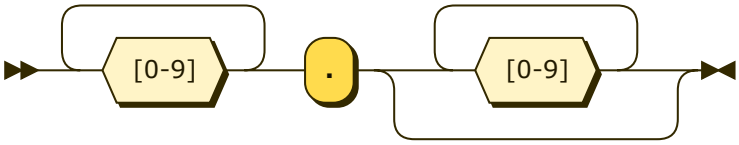



```
INTEGER  ::= '0'
           | [1-9] [0-9]*
```

referenced by:

- amount_type
- numeric_literal

FLOAT:

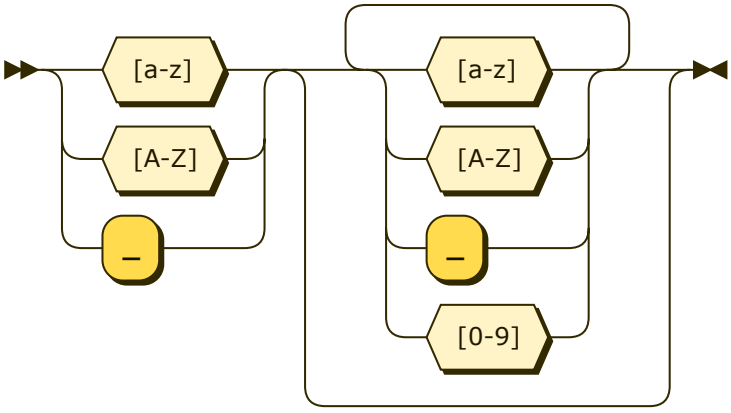



```
FLOAT    ::= [0-9]+ '.' [0-9]*
```

referenced by:

- numeric_literal

ID:

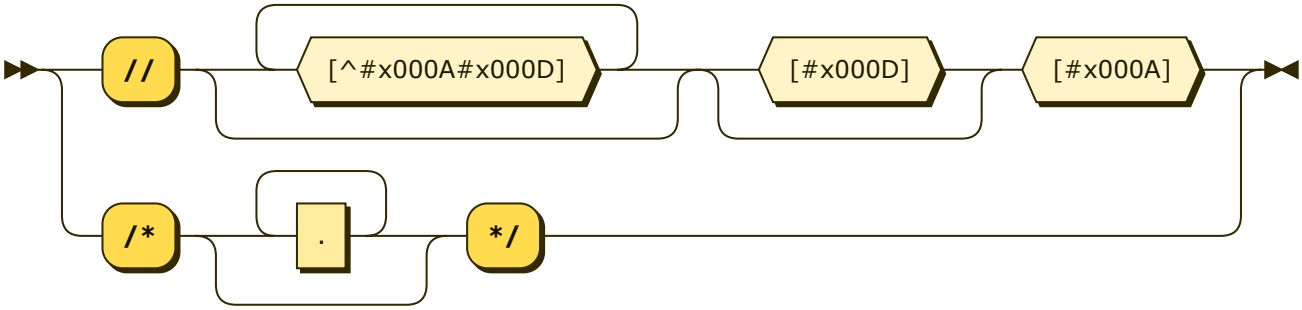



```
ID       ::= ( [a-z] | [A-Z] | '_' ) ( [a-z] | [A-Z] | '_' | [0-9] )*
```

referenced by:

- identifier

COMMENT:

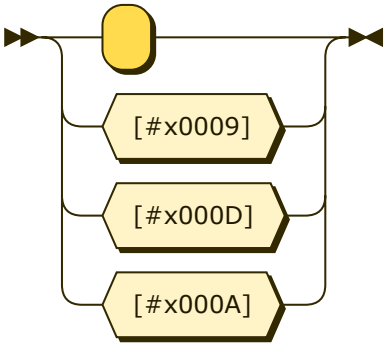



```
COMMENT? ::= '//' [^#x000A#x000D]* #x000D? #x000A
           | '/*' .* '*/'
```

referenced by:

- _

WS:




```
WS       ::= ' '
           | #x0009
           | #x000D
           | #x000A
```

referenced by:

- _

EOF:


```
EOF      ::= $
```

referenced by:

- start
