**EBNF Ruby**

**prog :** expression\_list;

**expression\_list :** (expression terminator) {expression terminator};

**expression :** function\_definition | require\_block | if\_statement | unless\_statement | rvalue | return\_statement | while\_statement | for\_statement ;

**require\_block :** REQUIRE literal\_t;

**function\_definition :** function\_definition\_header function\_definition\_body END;

**function\_definition\_body :** expression\_list;

**function\_definition\_header :** DEF function\_name [function\_definition\_params] crlf ;

**function\_name :** id\_function | id ;

**function\_definition\_params:** [LEFT\_RBRACKET] function\_definition\_params\_list [RIGHT\_RBRACKET]

**function\_definition\_params\_list :** (id) {COMMA id};

**return\_statement :** RETURN rvalue;

**function\_call :** (function\_name) (LEFT\_RBRACKET [function\_call\_param\_list] RIGHT\_RBRACKET | function\_call\_param\_list);

**function\_call\_param\_list :** function\_call\_params;

**function\_call\_params :** (rvalue) {COMMA rvalue};

**if\_elsif\_statement :** ELSIF rvalue crlf expression\_list [ELSE crlf expression\_list] {ELSIF rvalue crlf expression\_list [ELSE crlf expression\_list]};

**if\_statement :** IF rvalue (crlf | THEN) expression\_list (END | ELSE | if\_elsif\_statement)

[[ crlf] expression\_list] END;

**unless\_statement :** UNLESS rvalue crlf expression\_list END;

**while\_statement :** WHILE rvalue crlf while\_expression\_list END;

**while\_expression\_list :** (expression | RETRY | BREAK) terminator {(expression | RETRY | BREAK) terminator};

**for\_statement :** FOR [LEFT\_RBRACKET] expression SEMICOLON expression SEMICOLON expression [RIGHT\_RBRACKET] crlf for\_expression\_list END;

**for\_expression\_list :** (expression | RETRY | BREAK) terminator {(expression | RETRY | BREAK) terminator};

**assignment :** lvalue ( PLUS\_ASSIGN | MINUS\_ASSIGN | MUL\_ASSIGN | DIV\_ASSIGN | MOD\_ASSIGN | EXP\_ASSIGN | ASSIGN ) rvalue;

**array\_assignment :** (lvalue) (array\_definition ASSIGN rvalue | ASSIGN array\_definition);

**array\_definition :** LEFT\_SBRACKET [array\_definition\_elements] RIGHT\_SBRACKET;

**array\_definition\_elements :** (rvalue) {COMMA rvalue};

**array\_selector :** (id | id\_global | function\_call) LEFT\_SBRACKET rvalue RIGHT\_SBRACKET;

**int\_result :** (int\_t) ( MUL | DIV | MOD | PLUS | MINUS ) int\_t {( MUL | DIV | MOD | PLUS | MINUS ) int\_t};

**float\_result :** (float\_t | int\_result) ( MUL | DIV | MOD | PLUS | MINUS) (float\_t | int\_result)

({( MUL | DIV | MOD | PLUS | MINUS ) (float\_t | int\_result)};

**string\_result :** [int\_result MUL] literal\_t {MUL int\_result};

**lvalue :** id | id\_global ;

**rvalue :** [LEFT\_RBRACKET] [NOT | BIT\_NOT]

((lvalue | array\_assignment | int\_result | float\_result | string\_result | assignment | function\_call | literal\_t | bool\_t | float\_t | int\_t | nil\_t )

(EXP | MUL | DIV | MOD | PLUS | MINUS | BIT\_SHL | BIT\_SHR | BIT\_AND | BIT\_OR | BIT\_XOR | LESS | GREATER | LESS\_EQUAL | GREATER\_EQUAL | EQUAL | NOT\_EQUAL | OR | AND)

(lvalue | array\_assignment | int\_result | float\_result | string\_result | assignment | function\_call | literal\_t | bool\_t | float\_t | int\_t | nil\_t))

[RIGHT\_RBRACKET]

{[LEFT\_RBRACKET] [NOT | BIT\_NOT]

(lvalue | array\_assignment | int\_result | float\_result | string\_result | assignment | function\_call | literal\_t | bool\_t | float\_t | int\_t | nil\_t )

(EXP | MUL | DIV | MOD |PLUS | MINUS | BIT\_SHL | BIT\_SHR | BIT\_AND | BIT\_OR | BIT\_XOR | LESS | GREATER | LESS\_EQUAL | GREATER\_EQUAL | EQUAL | NOT\_EQUAL | OR | AND)

(lvalue | array\_assignment | int\_result | float\_result | string\_result | assignment | function\_call | literal\_t | bool\_t | float\_t | int\_t | nil\_t)) [RIGHT\_RBRACKET]};

**literal\_t :** LITERAL;

**float\_t :** FLOAT;

**int\_t :** INT;

**bool\_t :** TRUE | FALSE ;

**nil\_t :** NIL;

**id :** ID;

**id\_global :** ID\_GLOBAL;

**id\_function :** ID\_FUNCTION;

**terminator :** (SEMICOLON | crlf) {SEMICOLON | crlf};

**crlf :** CRLF ;

**ESCAPED\_QUOTE :** '\\"';

**LITERAL :** '"' ( ESCAPED\_QUOTE | ~('\n'|'\r') )\*? '"' | '\'' ( ESCAPED\_QUOTE | ~('\n'|'\r') )\*? '\'';

**COMMA :** ',';

**SEMICOLON :** ';';

**CRLF :** '\n';

**REQUIRE :** 'require';

**END :** 'end';

**DEF :** 'def';

**RETURN :** 'return';

**IF:** 'if';

**THEN :** 'then';

**ELSE :** 'else';

**ELSIF :** 'elsif';

**UNLESS :** 'unless';

**WHILE :** 'while';

**RETRY :** 'retry';

**BREAK :** 'break';

**FOR :** 'for';

**TRUE :** 'true';

**FALSE :** 'false';

**PLUS :** '+';

**MINUS :** '-';

**MUL :** '\*';

**DIV :** '/';

**MOD :** '%';

**EXP :** '\*\*';

**EQUAL :** '==';

**NOT\_EQUAL :** '!=';

**GREATER :** '>';

**LESS :** '<';

**LESS\_EQUAL :** '<=';

**GREATER\_EQUAL :** '>=';

**ASSIGN :** '=';

**PLUS\_ASSIGN :** '+=';

**MINUS\_ASSIGN :** '-=';

**MUL\_ASSIGN :** '\*=';

**DIV\_ASSIGN :** '/=';

**MOD\_ASSIGN :** '%=';

**EXP\_ASSIGN :** '\*\*=';

**BIT\_AND :** '&';

**BIT\_OR :** '|';

**BIT\_XOR :** '^';

**BIT\_NOT :** '~';

**BIT\_SHL :** '<<';

**BIT\_SHR :** '>>';

**AND :** 'and' | '&&';

**OR :** 'or' | '||';

**NOT :** 'not' | '!';

**LEFT\_RBRACKET :** '(';

**RIGHT\_RBRACKET :** ')';

**LEFT\_SBRACKET :** '[';

**RIGHT\_SBRACKET :** ']';

**NIL :** 'nil';

**SL\_COMMENT :** ('#' ~('\r' | '\n')\* '\n') -> skip;

**ML\_COMMENT :** ('=begin' .\*? '=end\n') -> skip;

**WS :** (' '|'\t')+ -> skip;

**INT :** [0-9]+;

**FLOAT :** [0-9]+'.'[0-9]+;

**ID :** [a-zA-Z\_][a-zA-Z0-9\_]\*;

**ID\_GLOBAL :** DOLLAR ID;

**ID\_FUNCTION :** ID [!?];

**DOLLAR** : ‘$’;