Grammatik

```
<stmt list>::= <stmt><stmt list>
            | <stmt>
<stmt>::= <return stmt>
         | <io_stmt>
         | <sel stmt>
         | <iter_stmt>
         | <assign_stmt>
            <function stmt>
            <expr>
<function stmt::= <function def> | <function call>
<function def>::= FUNCTION <func name> '('<parameter list>*')'
                <stmt list> /FUNCTION
<function call>::= <func name> '('<argument list>*')'
<argument list>::= <stmt>
                  | <stmt>, <argument_list>
<parameter_list>::= '<' <type> '>' <var_dec>
                 | '<' <type> '>' <var dec>, <parameter list>
<comments>::= # comments /#
<sel stmt>::= IF '(' <expr> ')' <stmt list> ELSE <stmt list> /IF
     <elseif>::= IF '(' <expr> ')' <stmt list> ELSE <elseif>
     | IF '(' <expr> ') ' <stmt list> ELSE <stmt list>
         IF '(' <expr> ')' <stmt list>
<iter stmt>::= WHILE '(' <expr> ')'<stmt list>/WHILE
        FOR '('<iter var> IN <num> TO <num> INCBY <num> ')' <stmt list> /FOR
        EACH '('<iter var> IN <var dec> ')' <stmt list> /EACH
<iter var>::= <' <type> '>' <var dec>
<data stmt>::= EMPTY <var dec>
             | REMOVE <var dec> '[' <NUM> ']'
              | REMOVE <var_dec> '{' <STR> '}'
             | ADD <var dec> <list>
              | ADD <var dec> : <hash>
<list>::= '[' <type list> ']' | '[' ']'
<hash>::='{' < type hash> '}' | {' '}'
<type list>::= <type> | <type list> ',' <type>
<type_hash>::= <str> '>' <type> | <type_hash> ','<str> '>' <type>
                 DECL '<' <type> '>' <var dec>: /DECL
<assign stmt>::=
                  DECL '<' <type> '>' <var_dec>: <expr> /DECL
HASH <var_dec> : <hash> /HASH
                   | LIST '<' <type>'>' <var dec>: <list> /LIST
<io stmt>::= <print stmt>
          | <read stmt>
```

```
<print stmt>::= PRINT <stmt list> /PRINT
<read stmt>::= READ <var dec> /READ | READ '<' <type> '>' <var dec> /READ
<return stmt>::= DONE <stmt list> /DONE
<expr>::= <expr> <operator a> <term>
         | <term>
<term>::= <term> <operator b> <log>
         | <log>
<log>::= <log> <log operator> <comp>
        | <log operator not> <comp>
        | <comp>
<comp>::= <comp> <comp operator> <factor>
       | <factor>
<factor>::= <function call>
           | (<expr>)
            | <type>
            | <var dec>
            | <data>
            | <list>
            | <hash>
| <var dec> '{' <var dec> '}'
                                       // call hash
<comp operator>::= == | =/= | > | >= | < | <=</pre>
<log operator>::= && | '||' | AND | OR
<log operator not>::= ! | NOT
<operator a>::= + | -
<operator b>::= * | / |%
<type>::= <num>
          | <str>
          | <bool>
          | <all>
<num>::= Float
   | '-' Float
<str>::= sequence /"[\w\s!\?]*"/
<var dec>::= sequence /[A-Z][a z0-9 ]*/
<func name>::= sequence /[A-Z]+/
<bool>::= TRUE | FALSE
```