digit = "0" | "1" | "2" | "3" | "4 | "5" | "6" | "7" | "8" | "9";

comma = ",";

dot = ".";

semicolon = ";";

asign\_operator = "=";

backslash = "\";

new\_line = backsklash, "n";

comment = "#", [text];

include\_statement = "from", library\_name, "import", object\_name, {coma, object\_name}, semicolon;

library\_name = identifier;

object\_name = identifier;

break\_statement = "break", semicolon ;

letter = "a" | ... | "z" | "A" | ... | "Z";

quote = "'";

whitespace = " ";

identifier = letter, {letter | digit};

variable\_name = identifier;

constant = digit, { digit };

text = quote, { letter | number | whitespace | character }, quote;

type = "bool" | "int" | "float" | "string" | "list";

bool\_values = "false" | "true";

int\_value = constant

float\_value = constant, [ ".", constant ];

string\_value = text;

variable\_value = bool\_value | int\_value | float\_value| string\_value;

array= "[", [element], {comma , element} , "]";

array\_name = variable\_name;

sum\_operator = "+" | "-";

multiply\_operator = "\*" | "/";

relational\_operator = "<" | ">" | "<=" | ">=" | "==" | "!=";

and\_operator = "and";

or\_operator = "or";

negation\_operator = "!";

arth\_expression = term, [sum\_operator, term];

term = factor, [ multiply\_operator, factor ];

factor = constant | variable | function\_call | "(", arth\_expression, ")";

parameters = [ variable\_name, {comma, variable\_name} ];

arguments = [ expression, {comma, expression} ];

function\_name = identifier;

function\_definition = "def", function\_name, "(", parameters , ")" , statements;

function\_call = function\_name, '(', arguments, ')', semicolon;

| variable\_name, dot, function\_name, '(', arguments, ')', {dot, function\_name, '(', arguments, ')'} , semicolon;

assign\_expression = expression;

variable\_assignment = variable\_name, [asign\_operator, assign\_expression], semicolon;

expression = realtion\_condition, { and\_operator, relation\_condition | or\_operator, relation\_condition};

relation\_condition = logic\_statement\_part, [relational\_operator, logic\_statement\_part];

logic\_statement\_part = [negation\_operator], arth\_expression;

if = "if", "(", expression, ")", statements, ["else", statements];

while = "while", "(", expression, ")", statements;

return = "return", "(", expression, ")";

statement = variable\_assignment

| function\_call

| if

| while

| break\_statement

| return;

statements = "{", {statement}, "}";

program = { function\_definition };