## Grammar description

G = { T, N, S, P } , where
T = terminal symbols
N = non-terminal symbols
S = start symbol
P = derivation rules

T = {a..z, A..Z, , , . , ' ' , " ", {, }, (, ), : , ; , 0..9, + , - , * , / , < , > , = , <=, >=, !=, &&, ||, #, return, print, func, var, if, else, while, true, false }

N = {<program>, <function>, <id>, <expression>, <operation>, <expression>, <operation>, <declaration>, <function-call>, <display>, <flow-control>, <arithmetic-operation>, <arithmetic-operator>, <logic-operation>, <logic-operator>, <letters>, <digits>, <string>, <char>, <characters>, <number>, <boolean> }

P = {

<program> → <function>

<function> → func <id> (<id>*) { <expression>+ [ return <id> | <expression>|<operation>]}

<id> → <letters> [<letters>*| <digits>*| _ ]

<expression> → <declaration> |
<function-call> |
<flow-control>

<operation> → <arithmetic-operation> |
<logic-operation>

<declaration> → var <id> [= <id> | <operation> | <function-call> | <string>| <number>] |
const <id> [= <id> | <operation> | <function-call> | <string>| <number>]

<function-call> → <display> ([<id>* | <numbers>* | <char>* | <string>*]) |
<id> ([<id>* | <numbers>* | <char>* | <string>*])

<display> → print

<flow-control> → if <logic-operation> {<expression>+} [else {<expression>+}] |
while <operation>:
for ( var <id> = <number> ; <id> <logic-operator> <id> ;
<id> <arithmetic- operator> ) {<expression>}

<arithmetic-operation> → <number> <arithm-operator> <number> |
<id> <arithm-operator> <id>

<arithmetic-operator> → + | - | / | * | %

<logic-operation> → <number> <logic-operator> <number> |
<id> <logic-operator> <id>

<logic-operator> → = | > | < | >= | <= | != | && | ||

<number> → <digits>$^{+}$ [. <digits>$^{+}$]

<letters> → a..z, A...z

<digits> → 0..9

<string> → " <characters>* "

<char> → ‘ <characters>* ‘

<characters>→
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?[\\]^_`{|}~ \t\n

<boolean> → true | false

}