Grammar description

```
program        -> vfc_defs stmts | stmts | vfc_defs | eps
vfc_defs       -> vfc_def vfc_defs'
vfc_defs'      -> vfc_def vfc_defs' | eps
vfc_def        -> var_def | func_def | class_def
class_def      -> class ID '(' ID ')' : NEWLINE INDENT class_body DEDENT
class_body     -> pass NEWLINE | vf_defs
vf_defs        -> vf_def vf_defs'
vf_defs'       -> vf_def vf_defs' | eps
vf_def         -> var_def | func_def

func_def       -> def ID '(' typed_args ')' ->type : NEWLINE INDENT func_body DEDENT |
                  def ID '(' typed_args ')' : NEWLINE INDENT func_body DEDENT |
                  def ID '(' ')' -> type : NEWLINE INDENT func_body DEDENT |
                  def ID '(' ')' : NEWLINE INDENT func_body DEDENT
typed_args     -> typed_var | typed_var , typed_args
func_body      -> all_decls stmts | stmts
all_decls      -> all_decl all_decls'
all_decls'     -> all_decl all_decls' | eps
all_decl       -> global_decl | nonloc_decl | var_def | func_def

typed_var      -> ID : type
type           -> ID | STRING | [ type ]
global_decl    -> global ID NEWLINE
nonloc_decl    -> nonlocal ID NEWLINE
var_def        -> typed_var = literal NEWLINE

stmt           -> si_stmt NEWLINE | if_stmt | wh_stmt | for_stmt
if_stmt        -> if expr : if_block | if expr : if_block else : block
if_block       -> block elif_stmts'
elif_stmts     -> elif_stmt elif_stmts'
elif_stmts'    -> elif_stmts elif_stmts' | eps
elif_stmt      -> elif expr : block
wh_stmt        -> while expr : block
for_stmt       -> for ID in expr : block

si_stmt        -> pass | expr | return expr | return | asgn_stmts
asgn_stmts     -> asgn_stmt asgn_stmts'
asgn_stmts'    -> asgn_stmt asgn_stmts' | eps
asgn_stmt      -> target = expr asgn_stmt'
asgn_stmt'     -> = target asgn_stmt' | eps
target         -> cexpr target' | ID target'
target'        -> . ID target' | [ expr ] target' | eps

block          -> NEWLINE INDENT stmts DEDENT
stmts          -> stmt stmts'
stmts'         -> stmt stmts' | eps

literal        -> None | True | False | INTEGER | STRING

expr           -> or_expr if expr else expr | or_expr #ask about right associativity
or_expr        -> and_expr or_expr'
or_expr'       -> or and_expr or_expr' | eps
and_expr       -> not_expr and_expr'
and_expr'      -> and not_expr | eps
not_expr       -> not cexpr | cexpr

cexpr          -> aexpr rel_op aexpr | aexpr
rel_op         -> == | != | < | > | <= | >= | is
aexpr          -> mexpr aexpr'
aexpr'         -> add_op mexpr aexpr' | eps
add_op         -> + | -
mexpr          -> uexpr mexpr'
mexpr'         -> mul_op uexpr mexpr' | eps
```

```
mul_op          -> * | // | %
uexpr           -> - uexpr | mi_expr #ask about left recursion
mi_expr         -> fexpr mi_expr'
mi_expr'        -> . i_or_f mi_expr' | [ expr ] mi_expr' | eps
i_or_f          -> ID | ID '(' args ')' | ID `(` `)`
fexpr           -> ID '(' args ')' | pexpr
pexpr           -> '(' expr ')' | lexpr
lexpr           -> [ args ] | literal | ID
args            -> expr | expr, args
```