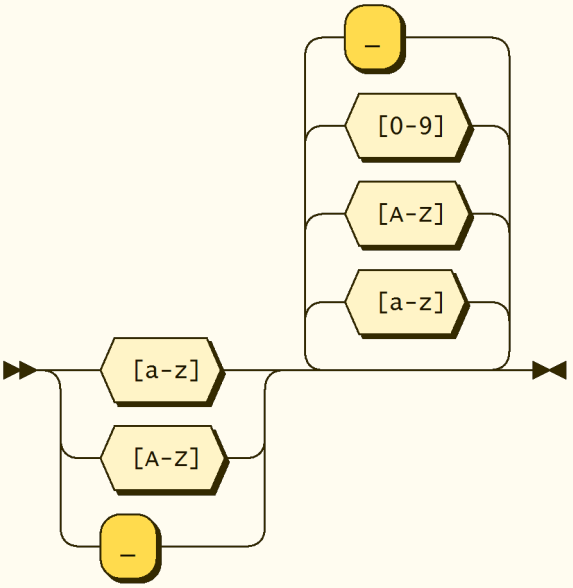
identifier:

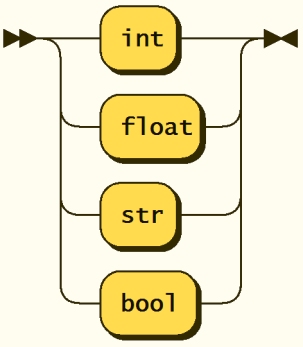



```
identifier
        ::= [a-zA-Z_] [a-zA-Z0-9_]*
```

referenced by:

- assignment
- forLoop
- functionDef
- function_call
- parameter
- variableDef

type:

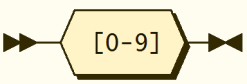



```
type     ::= 'int'
           | 'float'
           | 'str'
           | 'bool'
```

referenced by:

- functionDef
- variableDef

digit:

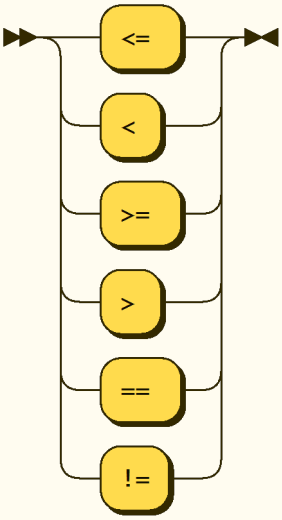



```
digit    ::= [0-9]
```

referenced by:

- decimalConstant
- integerConstant

logicalUnaryOperator:

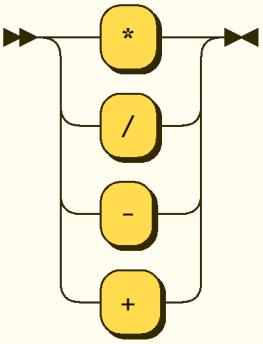



```
logicalUnaryOperator
         ::= '<='
           | '<'
           | '>='
           | '>'
           | '=='
           | '!='
```

referenced by:

- logicalExpression
- recursiveLogicalExpression

arithmeticUnaryOperator:

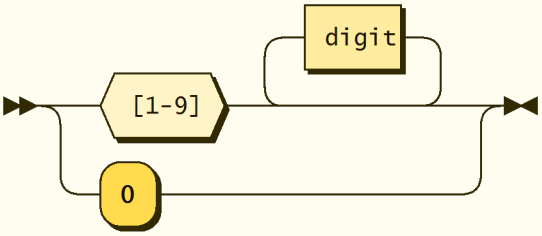



```
arithmeticUnaryOperator
         ::= '*'
           | '/'
           | '-'
           | '+'
```

referenced by:

- arithmeticExpression
- recursiveArithmeticExpression

integerConstant:



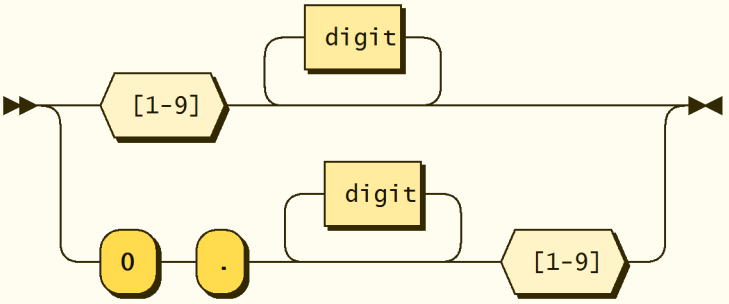

```
integerConstant
```

```
          ::= [1-9] digit*
            | '0'
```

referenced by:

- constantValue
- forLoop

decimalConstant:

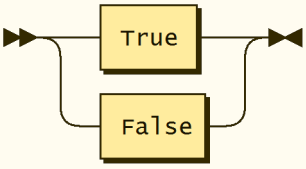



```
decimalConstant
          ::= [1-9] digit*
            | '0' '.' digit* [1-9]
```

referenced by:

- constantValue

logicalConstant:

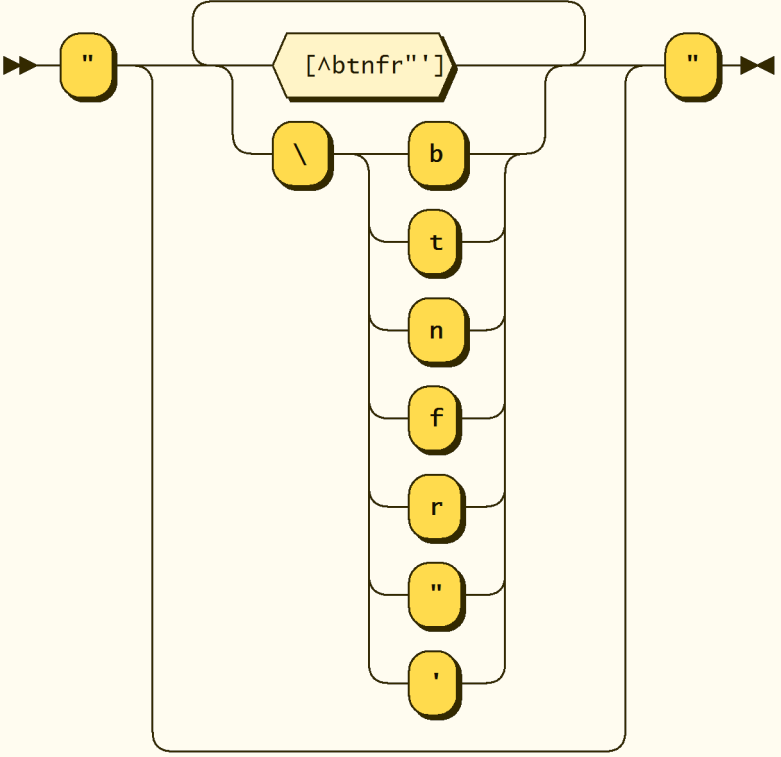



```
logicalConstant
          ::= True
            | False
```

referenced by:

- constantValue

string:

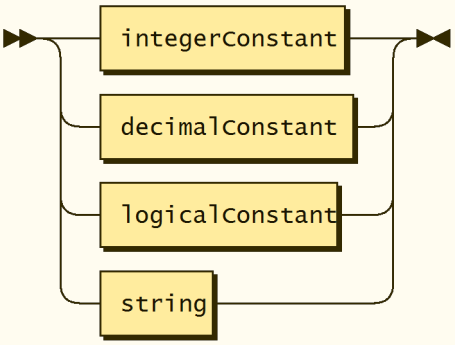



```
string   ::= '"' ( [^btnfr"'] | '\' [btnfr"'] )* '"'
```

referenced by:

- constantValue

constantValue:

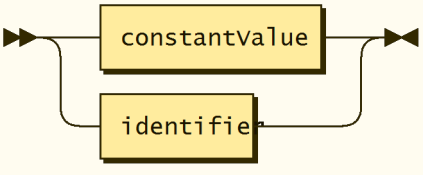



```
constantValue
         ::= integerConstant
           | decimalConstant
           | logicalConstant
           | string
```

referenced by:

- assignment
- parameter

parameter:

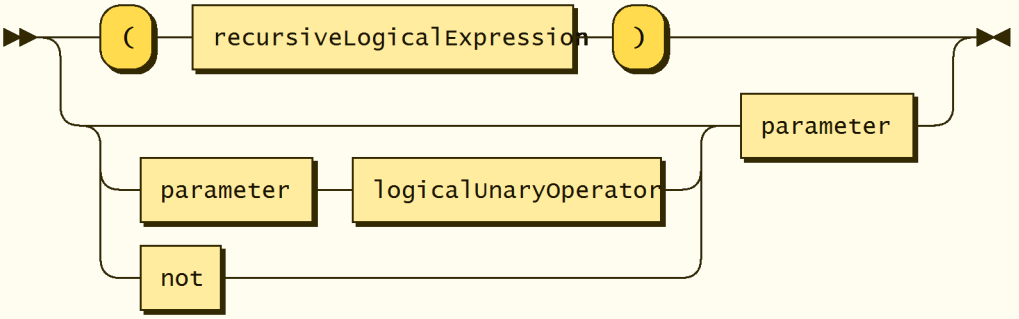



```
parameter
         ::= constantValue
           | identifier
```

referenced by:

- arithmeticExpression
- function_call
- logicalExpression
- variableDef

logicalExpression:

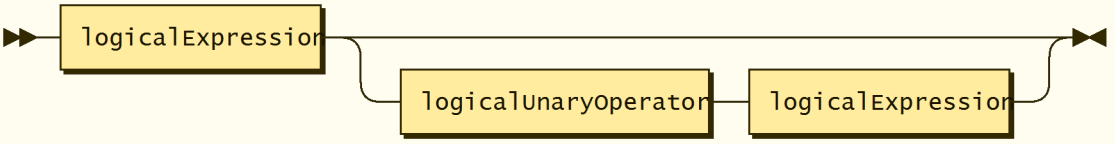



```
logicalExpression
         ::= '(' recursiveLogicalExpression ')'
           | ( parameter logicalUnaryOperator | not )? parameter
```

referenced by:

- assignment
- ifStatement
- recursiveLogicalExpression
- whileLoop

recursiveLogicalExpression:

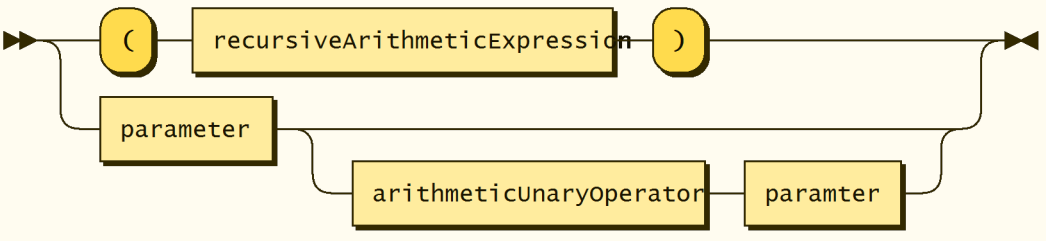



```
recursiveLogicalExpression
         ::= logicalExpression ( logicalUnaryOperator logicalExpression )?
```

referenced by:

- logicalExpression

arithmeticExpression:

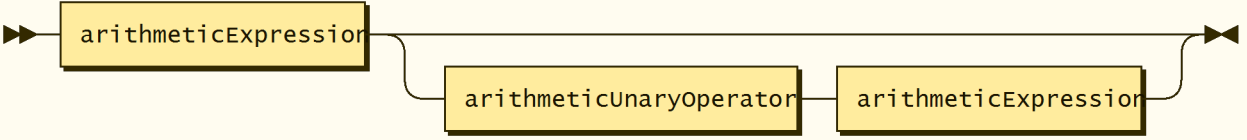



```
arithmeticExpression
         ::= '(' recursiveArithmeticExpression ')'
           | parameter ( arithmeticUnaryOperator paramter )?
```

referenced by:

- assignment
- recursiveArithmeticExpression

recursiveArithmeticExpression:

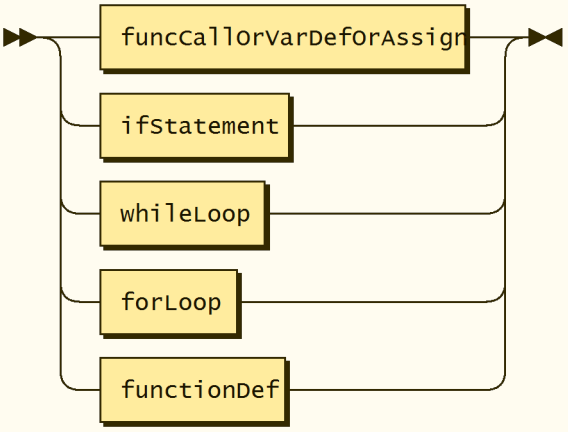



```
recursiveArithmeticExpression
        ::= arithmeticExpression ( arithmeticUnaryOperator arithmeticExpression )?
```

referenced by:

- arithmeticExpression

statement:

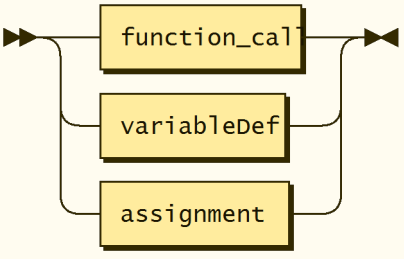



```
statement
         ::= funcCallOrVarDefOrAssign
           | ifStatement
           | whileLoop
           | forLoop
           | functionDef
```

referenced by:

- forLoop
- functionDef
- ifStatement
- program
- whileLoop

funcCallOrVarDefOrAssign:

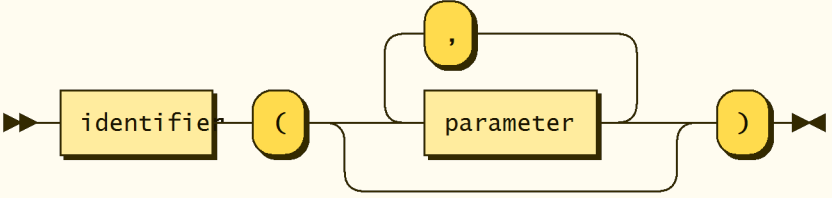



```
funcCallOrVarDefOrAssign
         ::= function_call
           | variableDef
           | assignment
```

referenced by:

- statement

function_call:

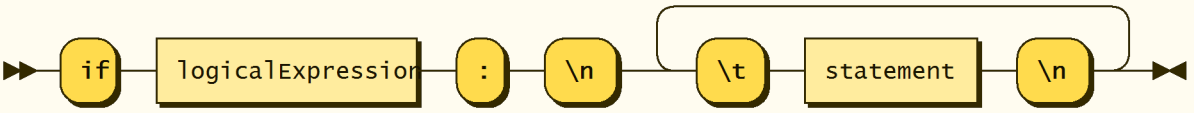



```
function_call
        ::= identifier '(' ( parameter ( ',' parameter )* )? ')'
```

referenced by:

- assignment
- funcCallOrVarDefOrAssign

ifStatement:

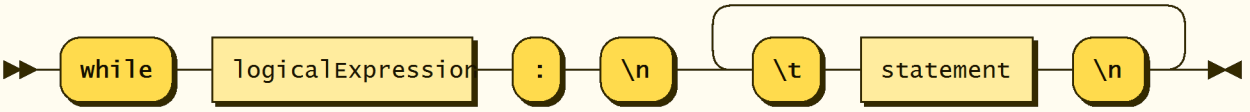



```
ifStatement
        ::= 'if' logicalExpression ':' '\n' ( '\t' statement '\n' )+
```

referenced by:

- statement

whileLoop:

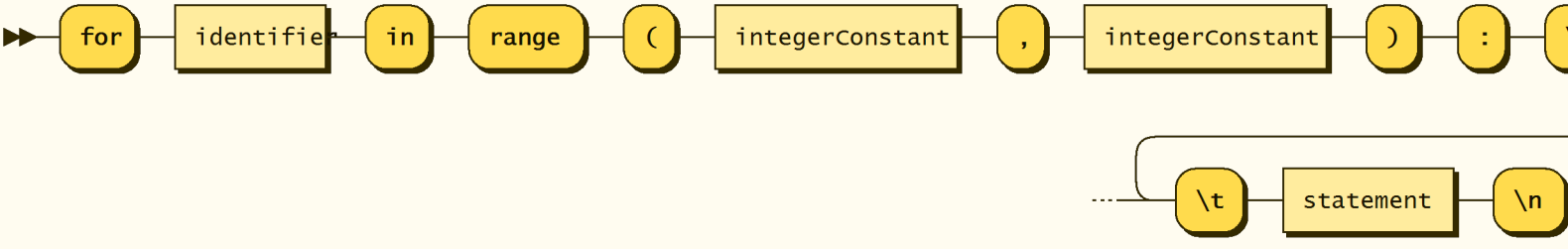



```
whileLoop
        ::= 'while' logicalExpression ':' '\n' ( '\t' statement '\n' )+
```

referenced by:

- statement

forLoop:

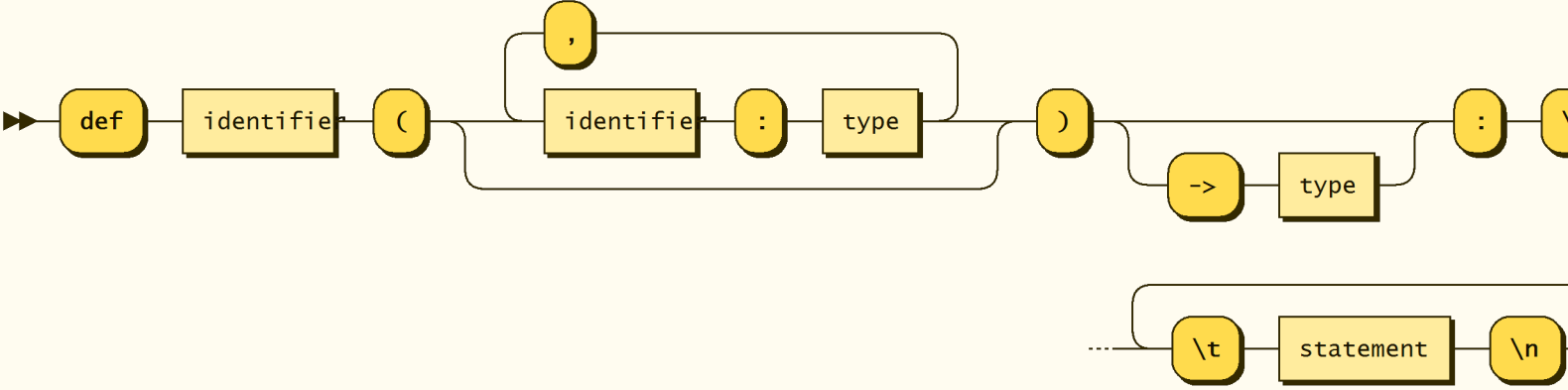



```
forLoop  ::= 'for' identifier 'in' 'range' '(' integerConstant ',' integerConstant ')' ':' '\n' ( '\t' state
```

referenced by:

- statement

functionDef:

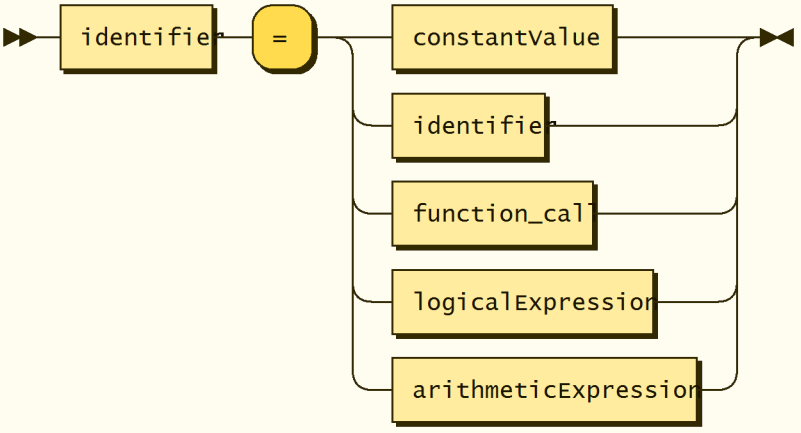



```
functionDef
        ::= 'def' identifier '(' ( identifier ':' type ( ',' identifier ':' type )* )? ')' ( '->' type )? '
'\n' )+
```

referenced by:

- statement

assignment:

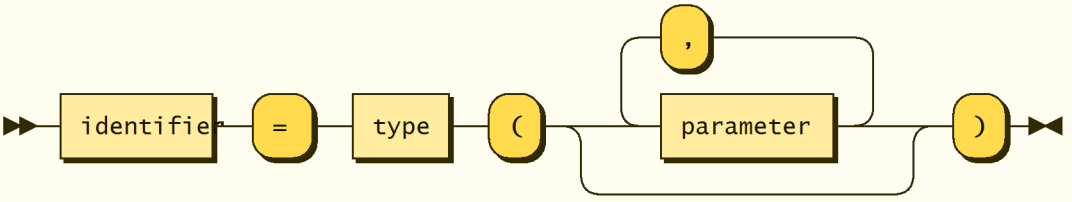



```
assignment
        ::= identifier '=' ( constantValue | identifier | function_call | logicalExpression | arithmeticExp
```

referenced by:

- funcCallOrVarDefOrAssign

variableDef:

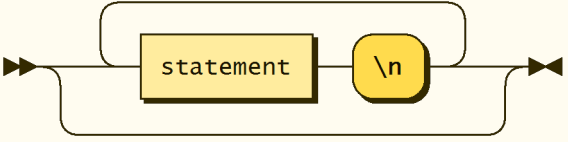



```
variableDef
        ::= identifier '=' type '(' ( parameter ( ',' parameter )* )? ')'
```

referenced by:

- funcCallOrVarDefOrAssign

program:




```
program  ::= ( statement '\n' )*
```

no references
