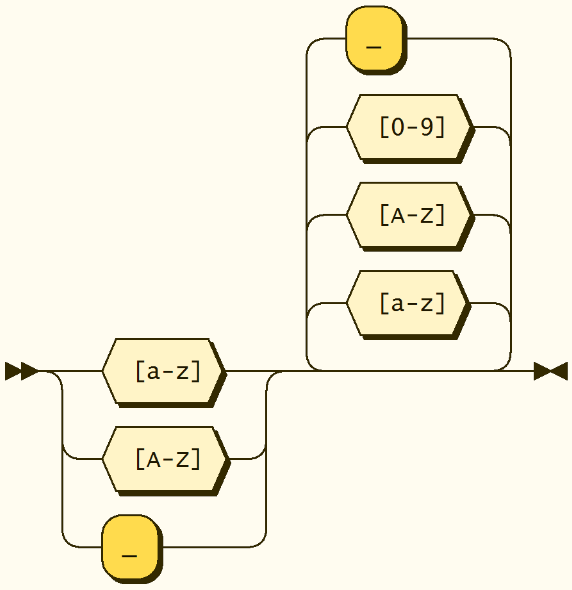
identifier:

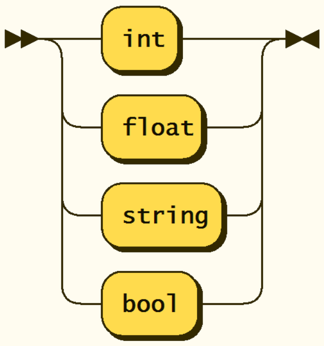


```
identifier
         ::= [a-zA-Z_] [a-zA-Z0-9_]*
```

referenced by:

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

type:

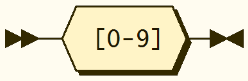


```
type     ::= 'int'
           | 'float'
           | 'string'
           | 'bool'
```

referenced by:

- functionDef
- variableDef

digit:

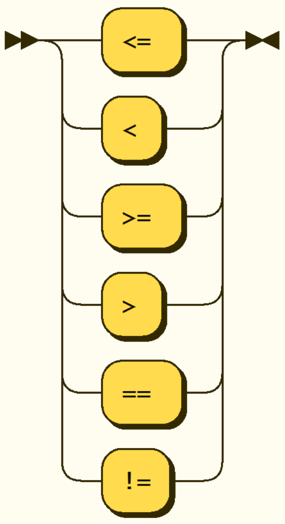


```
digit    ::= [0-9]
```

referenced by:

- decimalConstant
- integerConstant

logicalUnaryOperator:

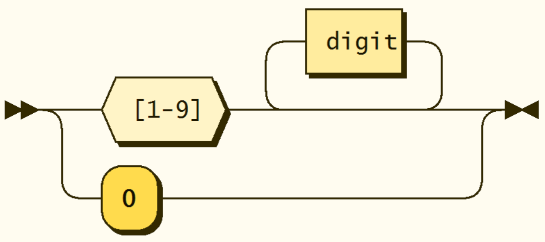


```
logicalUnaryOperator
         ::= '<='
           | '<'
           | '>='
           | '>'
           | '=='
           | '!='
```

referenced by:

- logicalFormula
- recursiveLogicalExpression

integerConstant:

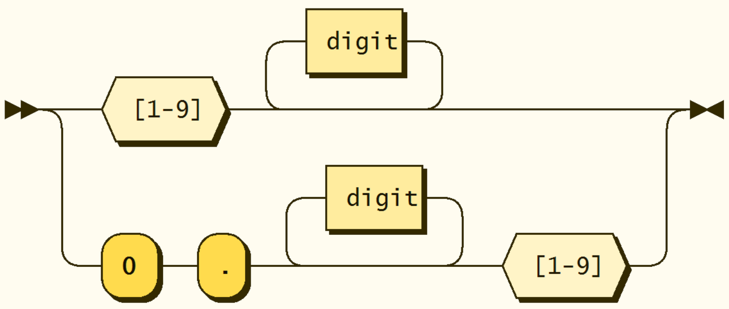


```
integerConstant
         ::= [1-9] digit*
           | '0'
```

referenced by:

- forLoop
- value

decimalConstant:

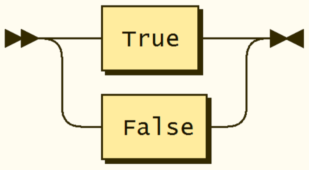


```
decimalConstant
         ::= [1-9] digit*
           | '0' '.' digit* [1-9]
```

referenced by:

- value

logicalConstant:

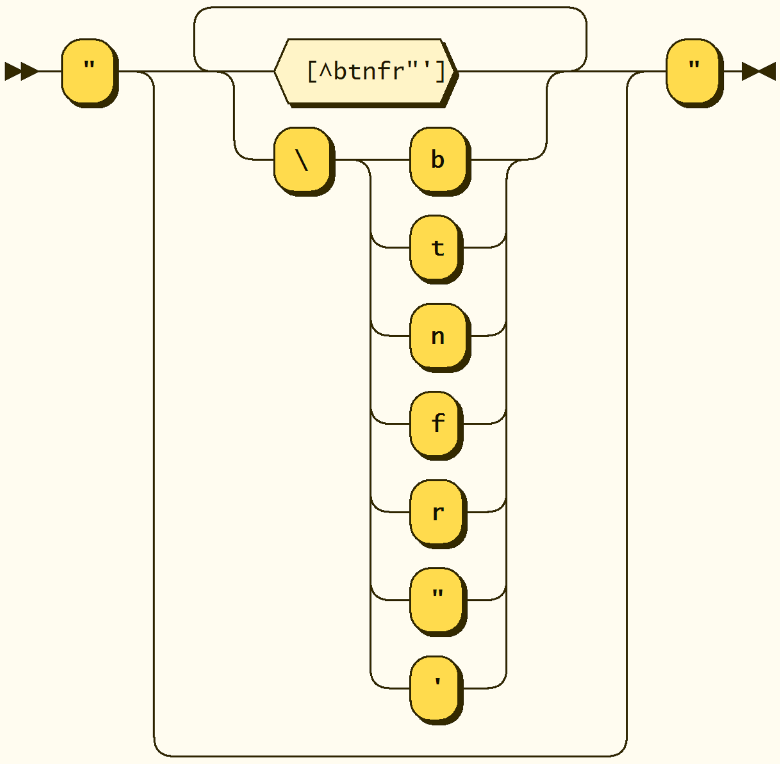


```
logicalConstant
         ::= True
           | False
```

referenced by:

- value

string:

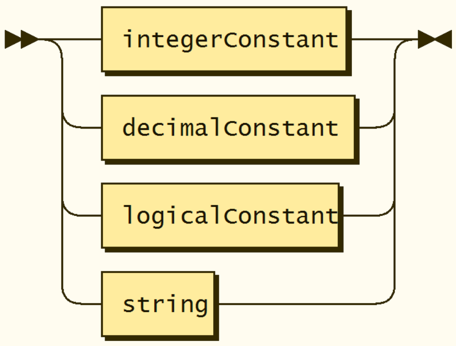


```
string   ::= '"' ( [^btnfr"'] | '\' [btnfr"'] )* '"'
```

referenced by:

- value

value:

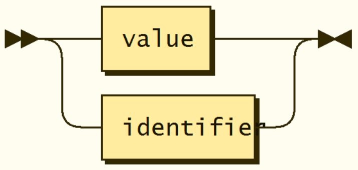


```
value    ::= integerConstant
           | decimalConstant
           | logicalConstant
           | string
```

referenced by:

- assignment
- parameter

parameter:

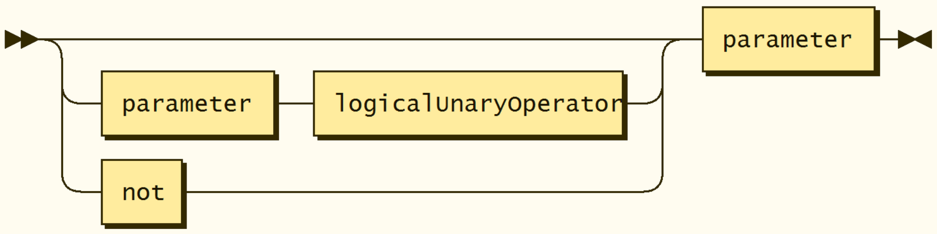


```
parameter
         ::= value
           | identifier
```

referenced by:

- arithmeticExpression
- function_call
- logicalFormula
- variableDef

logicalFormula:

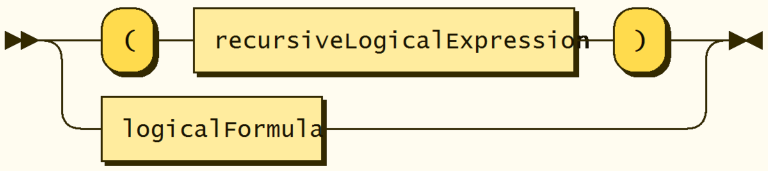


```
logicalFormula
         ::= ( parameter logicalUnaryOperator | not )? parameter
```

referenced by:

- logicalExpression

logicalExpression:

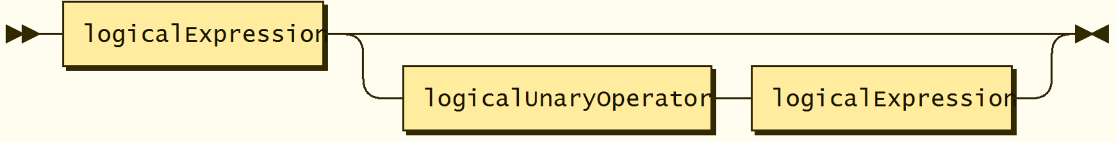


```
logicalExpression
         ::= '(' recursiveLogicalExpression ')'
           | logicalFormula
```

referenced by:

- assignment
- ifStatement
- recursiveLogicalExpression
- whileLoop

recursiveLogicalExpression:

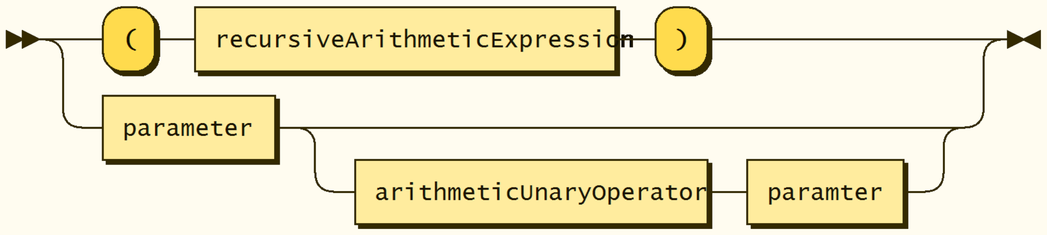


```
recursiveLogicalExpression
         ::= logicalExpression ( logicalUnaryOperator logicalExpression )?
```

referenced by:

- logicalExpression

arithmeticExpression:

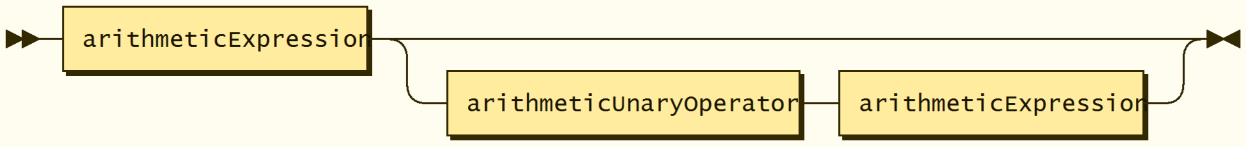


```
arithmeticExpression
         ::= '(' recursiveArithmeticExpression ')'
           | parameter ( arithmeticUnaryOperator paramter )?
```

referenced by:

- assignment
- recursiveArithmeticExpression

recursiveArithmeticExpression:

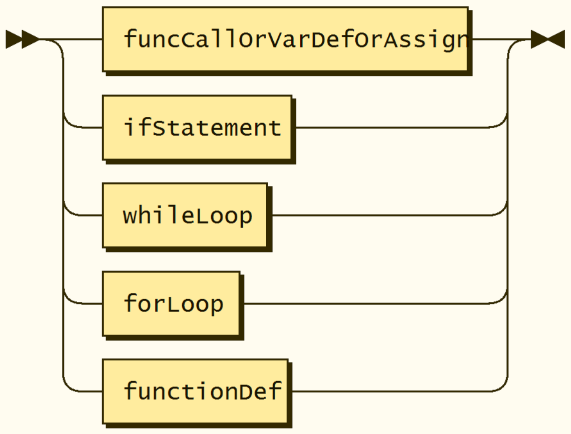


```
recursiveArithmeticExpression
         ::= arithmeticExpression ( arithmeticUnaryOperator arithmeticExpression )?
```

referenced by:

- arithmeticExpression

statement:

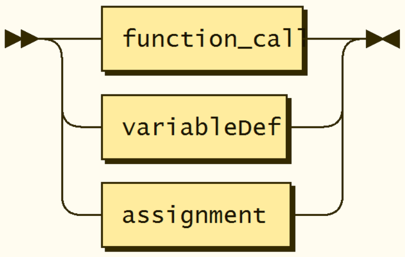


```
statement
         ::= funcCallOrVarDefOrAssign
           | ifStatement
           | whileLoop
           | forLoop
           | functionDef
```

referenced by:

- forLoop
- functionDef
- ifStatement
- program
- whileLoop

funcCallOrVarDefOrAssign:

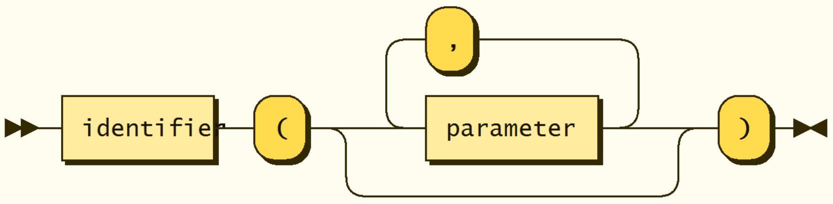


```
funcCallOrVarDefOrAssign
         ::= function_call
           | variableDef
           | assignment
```

referenced by:

- statement

function_call:

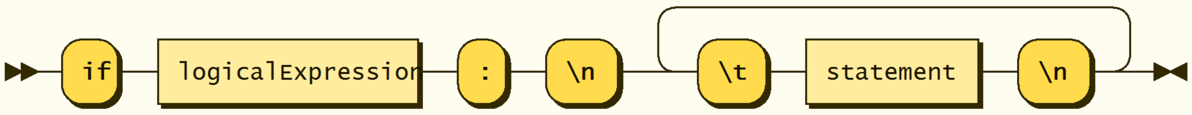


```
function_call
         ::= identifier '(' ( parameter ( ',' parameter )* )? ')'
```

referenced by:

- assignment
- funcCallOrVarDefOrAssign

ifStatement:

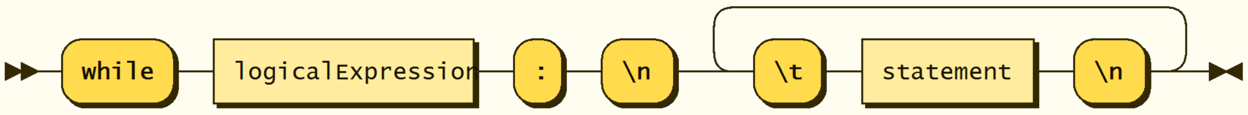


```
ifStatement
         ::= 'if' logicalExpression ':' '\n' ( '\t' statement '\n' )+
```

referenced by:

- statement

whileLoop:

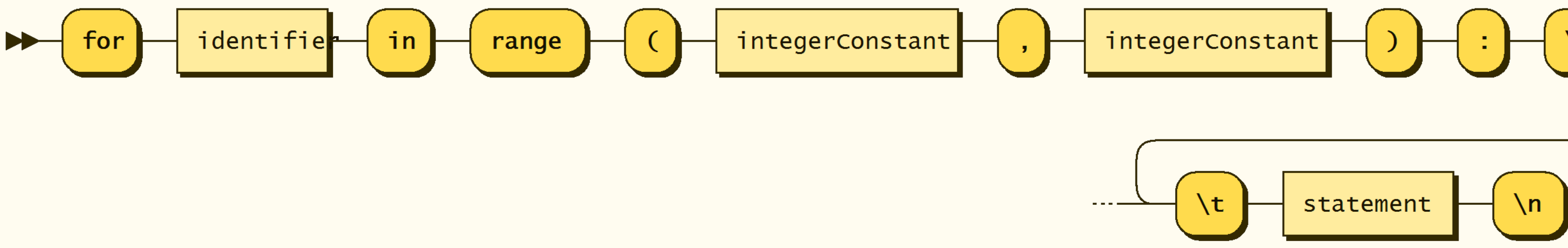


```
whileLoop
         ::= 'while' logicalExpression ':' '\n' ( '\t' statement '\n' )+
```

referenced by:

- statement

forLoop:

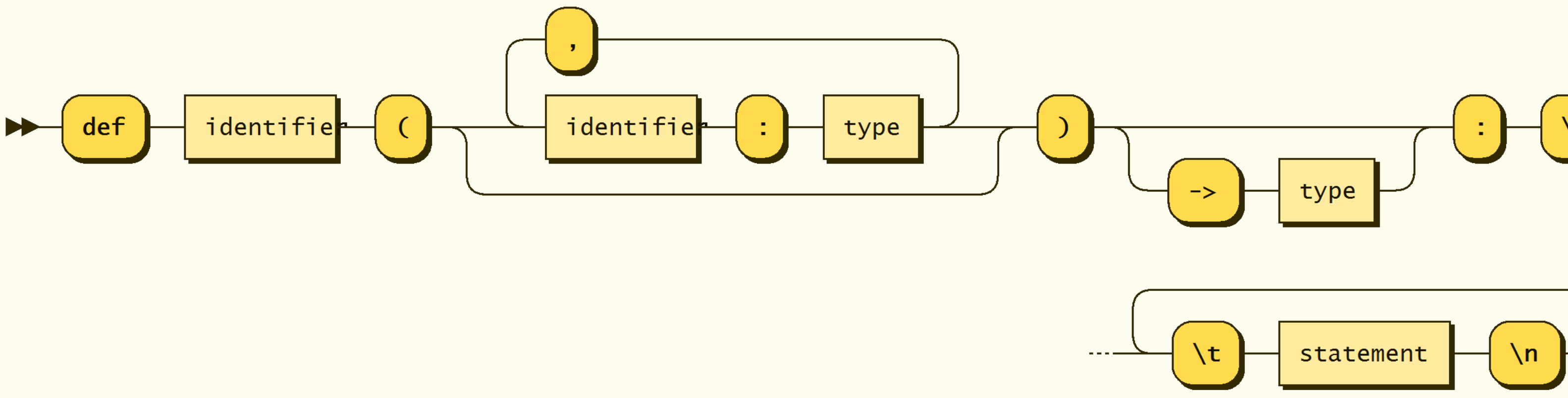


```
forLoop  ::= 'for' identifier 'in' 'range' '(' integerConstant ',' integerConstant ')' ':' '\n' ( '\t' state
```

referenced by:

- statement

functionDef:

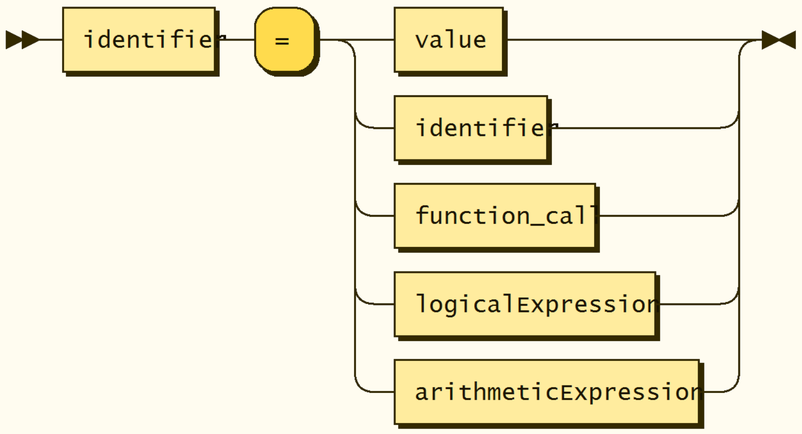


```
functionDef
         ::= 'def' identifier '(' ( identifier ':' type ( ',' identifier ':' type )* )? ')' ( '->' type )? '
'\n' )+
```

referenced by:

- statement

assignment:

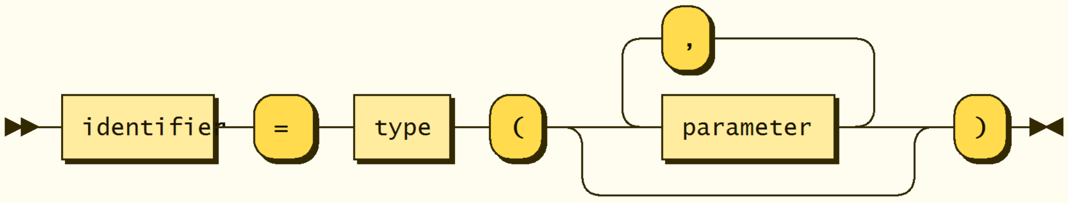


```
assignment
         ::= identifier '=' ( value | identifier | function_call | logicalExpression | arithmeticExpression
```

referenced by:

- funcCallOrVarDefOrAssign

variableDef:

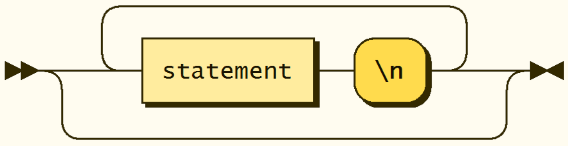


```
variableDef
         ::= identifier '=' type '(' ( parameter ( ',' parameter )* )? ')'
```

referenced by:

- funcCallOrVarDefOrAssign

program:



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

no references