Token Patterns

Literals

NUM := [0-9]+|[0-9]+\.[0-9]+

Variables

VAR := (?i:a)|(?i:b)|(?i:c)|(?i:x)|(?i:y)|(?i:z)|(?i:sum)|(?i:count)

Operators

AS_PLUS := "+=" # Increment the left variable by the right operand
AS_MINUS := "-=" # Decrement the left variable by the right operand
AS_MULT := "*=" # Multiply the left variable by the right operand
AS_DIV := "/=" # Divide the left variable by the right operand
AS_POW := "^=" # Raise the left variable to the right operand
AS_MOD := "%=" # Modulate the left variable by the right operand
EX_PLUS := "+" # Add both operands
EX_MINUS := "-" # Subtract the right operand from the left operand
EX_MULT := "*" # Multiply both operands
EX_DIV := "/" # Divide the left operand by the right operand
EX_POW := "^" # Raise the left operand to the right operand
EX_MOD := "%" # Modulate the left operand by the right operand
EX_EQ := "=="|"=" # Return 1 if both operands are equal, else 0
EX_LT := "<" # Return 1 if the left operand is smaller than the right operand, else 0
EX_GT := ">" # Return 1 if the left operand is greater than the right operand, else 0
EX_LE := "<=" # Return 1 if the left operand is smaller than or equal to the right operand, else 0
EX_GE := ">=" # Return 1 if the left operand is greater than or equal to the right operand, else 0
EX_NE := "!=" # Return 0 if both operands are equal, else 1
EX_CLAIMATION := "!" # Return 1 if the right operand is 0, else return 0 (unary)

Keywords

WHILE := (?i:while) # While the first argument evaluates to non-zero, evaluate the second argument
ENDWHILE := (?i:end\ while)|(?i:endwhile)
IF := (?i:if) # If the first argument evaluates to non-zero, evaluate the second argument
ELSE := (?i:else) # If the previous corresponding if statement evaluated to zero, evaluate the first argument.
ENDIF := (?i:end\ if)|(?i:endif)
SAY := (?i:say)|(?i:print) # Print the evaluation of the first argument to stdout
OUTPUT := (?i:output) # Print the control code '#OUTPUT(CHANNEL, VAL)' to stdout, where OUTPUT is the first argument and VAL is the evaluation of the second argument.

Other

RPAREN := ")" # Used to explicitly specify order of operations
LPAREN := "(" # Used to explicitly specify order of operations.