# GO - OPERATORS



An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. Go language is rich in built-in operators and provides the following types of operators –

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Miscellaneous Operators

This tutorial explains arithmetic, relational, logical, bitwise, assignment, and other operators one by one.

## Arithmetic Operators

Following table shows all the arithmetic operators supported by Go language. Assume variable **A** holds 10 and variable **B** holds 20 then –

Show Examples

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands | A + B gives 30 |
| - | Subtracts second operand from the first | A - B gives -10 |
| * | Multiplies both operands | A * B gives 200 |
| / | Divides the numerator by the denominator. | B / A gives 2 |
| % | Modulus operator; gives the remainder after an integer division. | B % A gives 0 |
| ++ | Increment operator. It increases the integer value by one. | A++ gives 11 |
| -- | Decrement operator. It decreases the integer value by one. | A-- gives 9 |

## Relational Operators

The following table lists all the relational operators supported by Go language. Assume variable **A** holds 10 and variable **B** holds 20, then –

Show Examples

| Operator | Description | Example |
|---|---|---|
| == | It checks if the values of two operands are equal or not; if yes, the condition becomes true. | $A == B$ is not true. |
| != | It checks if the values of two operands are equal or not; if the values are not equal, then the condition becomes true. | $A! = B$ is true. |
| > | It checks if the value of left operand is greater than the value of right operand; if yes, the condition becomes true. | $A > B$ is not true. |
| < | It checks if the value of left operand is less than the value of the right operand; if yes, the condition becomes true. | $A < B$ is true. |
| >= | It checks if the value of the left operand is greater than or equal to the value of the right operand; if yes, the condition becomes true. | $A >= B$ is not true. |
| <= | It checks if the value of left operand is less than or equal to the value of right operand; if yes, the condition becomes true. | $A <= B$ is true. |

## Logical Operators

The following table lists all the logical operators supported by Go language. Assume variable **A** holds 1 and variable **B** holds 0, then –

Show Examples

| Operator | Description | Example |
|---|---|---|
| && | Called Logical AND operator. If both the operands are non-zero, then condition becomes true. | A && B is false. |
| \|\| | Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true. | $A\|\|B$ is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !A && B is true. |

The following table shows all the logical operators supported by Go language. Assume variable **A** holds true and variable **B** holds false, then –

| Operator | Description | Example |
|---|---|---|
| && | Called Logical AND operator. If both the operands are false, then the condition becomes false. | A && B is false. |
| \|\| | Called Logical OR Operator. If any of the two operands is true, then the condition becomes true. | $A \|\| B$ is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. | !A && B is true. |

## Bitwise Operators

Bitwise operators work on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows –

| p | q | p & q | p \| q | p ^ q |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

Assume A = 60; and B = 13. In binary format, they will be as follows –

A = 0011 1100

B = 0000 1101

----------

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A  = 1100 0011

The Bitwise operators supported by C language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then –

Show Examples

| Operator | Description | Example |
|---|---|---|
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | A & B will give 12, which is 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | $A \mid B$ will give 61, which is 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | $A^B$ will give 49, which is 0011 0001 |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 will give 240 which is 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 will give 15 which is 0000 1111 |

## Assignment Operators

The following table lists all the assignment operators supported by Go language –

Show Examples

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operator, Assigns values from right side operands to left side operand | C = A + B will assign value of A + B into C |
| += | Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand | C -= A is equivalent to C = C - A |
| *= | Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator, It divides left operand | C /= A is equivalent to C = C / A |

| | | |
|---|---|---|
| | with the right operand and assign the result to left operand | |
| %= | Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator | C &= 2 is same as C = C & 2 |
| ^= | bitwise exclusive OR and assignment operator | C ^= 2 is same as C = C ^ 2 |
| \|= | bitwise inclusive OR and assignment operator | C \|= 2 is same as C = C \| 2 |

## Miscellaneous Operators

There are a few other important operators supported by Go Language including **sizeof** and **?:.**

Show Examples

| Operator | Description | Example |
|---|---|---|
| & | Returns the address of a variable. | &a; provides actual address of the variable. |
| * | Pointer to a variable. | *a; provides pointer to a variable. |

## Operators Precedence in Go

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Show Examples

| Category | Operator | Associativity |
|---|---|---|
| Postfix | [] -> . ++ - - | Left to right |
| | | |

| | | |
|---|---|---|
| Unary | + - ! ~ ++ - - *type** & sizeof | Right to left |
| Multiplicative | * / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | \| | Left to right |
| Logical AND | && | Left to right |
| Logical OR | \|\| | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %=>>= <<= &= ^= \|= | Right to left |
| Comma | , | Left to right |