Swift - Operators

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

* Arithmetic Operators
* Comparison Operators
* Logical Operators
* Bitwise Operators
* Assignment Operators
* Range Operators
* Misc Operators

This tutorial will explain the arithmetic, relational, logical, bitwise, assignment and other operators one by one.

## **Arithmetic Operators**

The following table shows all the arithmetic operators supported by Swift 4 language. Assume variable **A** holds 10 and variable **B** holds 20, then −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Adds two operands | A + B will give 30 |
| − | Subtracts second operand from the first | A − B will give -10 |
| \* | Multiplies both operands | A \* B will give 200 |
| / | Divides numerator by denominator | B / A will give 2 |
| % | Modulus Operator and remainder of after an integer/float division | B % A will give 0 |

## **Comparison Operators**

The following table shows all the relational operators supported by Swift 4 language. Assume variable **A** holds 10 and variable **B** holds 20, then −

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

## **Logical Operators**

The following table shows all the logical operators supported by Swift 4 language. Assume variable **A** holds 1 and variable **B** holds 0, then −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false. |
| || | Called Logical OR Operator. If any of the two operands is non-zero, 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 the 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

Bitwise operators supported by Swift 4 language are listed in the following table. Assume variable **A** holds 60 and variable **B** holds 13, then 7−

|  |  |  |
| --- | --- | --- |
| **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 | 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 Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) will give -61, which is 1100 0011 in 2's complement form. |
| << | 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**

SSwift 4 supports the following assignment operators −

|  |  |  |
| --- | --- | --- |
| **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 assigns 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 assigns 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 assigns the result to left operand | C \*= A is equivalent to C = C \* A |
| /= | Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator, It takes modulus using two operands and assigns 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 |

## **Range Operators**

Swift 4 includes two range operators, which are shortcuts for expressing a range of values. The following table explains these two operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| Closed Range | (a...b) defines a range that runs from a to b, and includes the values a and b. | 1...5 gives 1, 2, 3, 4 and 5 |
| Half-Open Range | (a..< b) defines a range that runs from a to b, but does not include b. | 1..< 5 gives 1, 2, 3, and 4 |
| One- sided Range | a… , defines a range that runs from a to end of elements  …a , defines a range starting from start to a | 1… gives 1 , 2,3… end of elements  …2 gives beginning… to 1,2 |

## **Misc Operators**

Swift 4 supports a few other important operators including **range** and ? : which are explained in the following table.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| Unary Minus | The sign of a numeric value can be toggled using a prefixed - | -3 or -4 |
| Unary Plus | Returns the value it operates on, without any change. | +6 gives 6 |
| Ternary Conditional | Condition ? X : Y | If Condition is true ? Then value X : Otherwise value Y |

## **Operators Precedence**

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.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| Primary Expression Operators | () [] . expr++ expr-- | left-to-right |
| Unary Operators | \* & + - ! ~ ++expr --expr  \* / %  + -  >> <<  < > <= >=  == != | right-to-left |
| Binary Operators | &  ^  |  &&  || | left-to-right |
| Ternary Operator | ?: | right-to-left |
| Assignment Operators | = += -= \*= /= %= >>= <<= &=^= |= | right-to-left |
| Comma | , | left-to-right |

## **Classification of Operators on Terminology**

There are three types of operators: unary, binary and ternary.

**Unary operator:** Unary operators are used on a single target (like -a). Unary prefix operators appear immediately before their target (such as !b). Unary postfix operators appear immediately after their target (such as c!).

**Binary Operator:** Binary operators are used when we have to operate on two targets. (like 2 + 3) and are infix because they appear in between their two targets.

**Ternary Operator:** Ternary operators operate on three targets. Like C language, Swift also has only one ternary operator called the ternary conditional operator (a ? b : c).