Operators

➔ **Operators** are mathematical symbols which are used to perform operations on a value or data.

➔ The data on which operators work are called operands.

Consider the following expression −

7 + 5 = 12

Here, the values 7, 5, and 12 are **operands**, while + and = are **operators**.

Types -

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Bitwise operators
5. Assignment operators
6. Ternary/conditional operator
7. String operator
8. Type Operator

Arithmetic Operators

Operator	Description	Example
+ (Addition)	returns the sum of the operands	a + b is 15
- (Subtraction)	returns the difference of the values	a - b is 5
* (Multiplication)	returns the product of the values	a * b is 50
/ (Division)	performs division operation and returns the quotient	a / b is 2
% (Modulus)	performs division operation and returns the remainder	a % b is 0
++ (Increment)	Increments the value of the variable by one	a++ is 11
-- (Decrement)	Decrements the value of the variable by one	a-- is 9

Relational Operators

Relational Operators test or define the kind of relationship between two entities. Relational operators return a Boolean value, i.e., true/ false.

- Assume the value of A is 10 and B is 20.

Operator	Description	Example
>	Greater than	(A > B) is False
<	Lesser than	(A < B) is True
>=	Greater than or equal to	(A >= B) is False
<=	Lesser than or equal to	(A <= B) is True
==	Equality	(A == B) is false
!=	Not equal	(A != B) is True

Logical Operators

Logical Operators are used to combine two or more conditions. Logical operators return a Boolean value.

- Assume the value of variable A is 10 and B is 20.

Operator	**Description**	**Example**
&& (And)	The operator returns true only if all the expressions specified return true	(A > 10 && B > 10) is False
|| (OR)	The operator returns true if at least one of the expressions specified return true	(A > 10 || B >10) is True
! (NOT)	The operator returns the inverse of the expression's result. For E.g.: !(>5) returns false	!(A >10) is True

& (Bitwise AND)	It performs a Boolean AND operation on each bit of its integer arguments.	(A & B) is 2

| (BitWise OR)	It performs a Boolean OR operation on each bit of its integer arguments.	(A | B) is 3
^ (Bitwise XOR)	It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.	(A ^ B) is 1
~ (Bitwise Not)	It is a unary operator and operates by reversing all the bits in the operand.	(~B) is -4
<< (Left Shift)	It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.	(A << 1) is 4
>> (Right Shift)	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.	(A >> 1) is 1
>>> (Right shift with Zero)	This operator is just like the >> operator, except that the bits shifted in on the left are always zero.	(A >>> 1) is 1

Assignment Operators

Operator	Description	Example
= (Simple Assignment)	Assigns values from the right side operand to the left side operand	C = A + B will assign the value of A + B into C
+= (Add and Assignment)	It adds the right operand to the left operand and assigns the result to the left operand.	C += A is equivalent to C = C + A
-= (Subtract and Assignment)	It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*= (Multiply and Assignment)	It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/= (Divide and Assignment)	It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A

Note − Same logic applies to Bitwise operators, so they will become <<=, >>=, >>=, &=, |= and ^=.

Conditional Operator (?)

This operator is used to represent a conditional expression. The conditional operator is also sometimes referred to as the ternary operator.

The syntax is as given below −

Test ? expr1 : expr2

- Test − refers to the conditional expression
- expr1 − value returned if the condition is true
- expr2 − value returned if the condition is false

Eg.

```
var num:number = -2
var result = num > 0 ?"positive":"non-positive"
console.log(result)
```

String Operators: Concatenation operator (+)

The + operator when applied to strings appends the second string to the first.

Eg.

```
var msg:string = "hello"+"world"
console.log(msg)
```

Type Operators

There are a collection of operators available which can assist you when working with objects in TypeScript. Operators such as typeof, instanceof, in, and delete are the examples of Type operators. The detailed explanation of these operators is given below.

Operator_Name	Description	Example
in	It is used to check for the existence of a property on an object.	let Bike = {make: 'Honda', model: 'CLIQ', year: 2018}; console.log('make' in Bike); // **Output:** true

delete	It is used to delete the properties from the objects.	let Bike = { Company1: 'Honda',
		Company2: 'Hero',
		Company3: 'Royal Enfield'
		};
		delete Bike.Company1;
		console.log(Bike); //
		Output:
		{ Company2: 'Hero', Company3: 'Royal Enfield' }

typeof	It returns the data type of the operand.	let message = "Welcome to " + "JavaTpoint"; console.log(typeof message); // **Output:** String

instanceof	It is used to check if the object is of a specified type or not.	let arr = [1, 2, 3];
		console.log(arr instanceof Array); // true
		console.log(arr instanceof String); // false