ES6 operators

The operator can be defined as a symbol that tells the system to implement a particular operation. In [JavaScript](https://www.javatpoint.com/javascript-tutorial), there is a rich set of operators, and by using specific operators, you can perform any particular task.

The operators are used in the expressions for evaluating different operands.

An expression is a kind of statement that returns a value. The expression includes:

* **Operators:** It is responsible for performing some operations on operands
* **Operands:** It represents the data.

**For example:** Suppose an expression like**x\*y.** In this expression,**x**and **y**are the **operands,** and the asterisk **(\*)** symbol is the multiplication operator.

Types of Operators

Operators in JavaScript can be classified as:

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Assignment Operators
* Bitwise Operators
* Type Operators
* Miscellaneous Operators

Let us try to elaborate on these operators in detail.

Arithmetic Operators

Arithmetic operators are the basic mathematical operators that are available in JavaScript [ES6](https://www.javatpoint.com/es6). These operators are responsible for performing all mathematical operations such as addition, subtraction, etc. in JavaScript.

|  |  |
| --- | --- |
| **Operators** | **Functions** |
| **+ (Addition)** | It returns the sum of the value of operands |
| **- (Subtraction)** | It returns the difference between the value of operands |
| **\* (Multiplication)** | It returns the product of operands values. |
| **/ (Division)** | It is used to perform division, and it returns quotient. |
| **% (Modulus)** | It also performs division and returns the remainder. |
| **++ (Increment)** | It increments the value of a variable by one. |
| **- (Decrement)** | It decrements the value of a variable by one. |

**For example**

In this example, we are using all arithmetic operators that are listed above:

1. var x = 30;
2. var y = 20 ;
4. console.log("Addition: " + (x + y) );
5. console.log("Subtraction: " + (x - y) );
6. console.log("Multiplication: " + (x \* y) );
7. console.log("The Division will give you the quotient: " + (x / y) );
8. console.log("Modulus will give you the Remainder: " + (x % y) );
9. // pre-increment
10. console.log("Value of x after pre-increment: "  + (++x) );
11. // post-increment
12. console.log("Value of x after post-increment: " + (x++) );
13. // pre-decrement
14. console.log("Value of y after pre-decrement: "  + (--y) );
15. // post-decrement
16. console.log("Value of y after post-decrement: " + (y--) );

**Output:**

When you execute the above code in the terminal, you will get the following output:

Addition : 50

Subtraction: 10

Multiplication: 600

The division will give you the quotient: 1.5

Modulus will give you the Remainder: 10

Value of x after pre-increment: 31

Value of x after post-increment: 31

Value of y after pre-decrement: 19

Value of y after post-decrement: 19

Relational Operators

Relational operators are used for comparing the two values and return either true or false based on the expression. These operators are sometimes called Comparison Operators.

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **> (Greater than)** | It returns true if the left operand is greater than right operand else it returns false. |
| **< (Less than)** | It returns true if the left operand is smaller than right operand else it returns false. |
| **>= (Greater than or equal to)** | It returns true if the left operand is greater than or equal to right operand else it returns false. |
| **<= (Less than or equal to)** | It returns true if the left operand is smaller than or equal to right operand else it returns false. |
| **== (Equality)** | It returns true if the value of both operands is the same else it returns false. |
| **!= (Not Equal to)** | It returns true if the value of operands is not the same else it returns false. |

**For example:**

In this example, we are using all relational operators that are listed above

1. var x = 20;
2. var y = 15;
4. console.log("Value of x: " + x);
5. console.log("Value of y: " + y);
6. var result = x > y;
7. console.log("x is greater than y: " + result);
8. result = x < y;
9. console.log("x is smaller than y: " + result);
10. result = x >= y;
11. console.log("x is greater than or equal to  y: " + result);
12. result = x <= y;
13. console.log("x is smaller than or equal to y: " + result);
14. result = x == y;
15. console.log("x is equal to y: " + result);
16. result = x != y;
17. console.log("x not equal to  y: " + result);

**Output:**

When you execute this code in the terminal, you will get the following output:

Value of x: 20

Value of y: 15

x is greater than y: true

x is smaller than y: false

x is greater than or equal to y: true

x is smaller than or equal to y: false

x is equal to y: false

x not equal to y: true

Logical Operators

Logical operators are generally used for combining two or more relational statements. They also return Boolean values.

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **&& (Logical AND)** | This operator returns true if all relational statements that are combined with && are true, else it returns false. |
| **|| (Logical OR)** | This operator returns true if at least one of the relational statements that are combined with || are true, else it returns false. |
| **! (Logical NOT)** | It returns the inverse of the statement's result. |

**For example:**

In this example, we are using all logical operators that are listed above.

1. var x = 30;
2. var y = 80;
4. console.log("Value of x = " + x );
5. console.log("Value of y = " + y );
7. var result = ((x < 40) && (y <= 90));
8. console.log("(x < 40) && (y <= 90): ", result);
10. var result = ((x == 50) || (y > 80));
11. console.log("(x == 50) || (y > 80): ", result);
13. var result = !((x > 20) && (y >= 80));
14. console.log("!((x > 20) && (y >= 80)): ", result);

**Output:**

Value of x = 30

Value of y = 80

(x < 40) && (y <= 90): true

(x == 50) || (y > 80): false

!((x > 20) && (y >= 80)): false

Assignment Operators

Assignment operators are used for assigning a value to the variable. The operand on the left side of the assignment operator is a variable, and the operand on the right side of the assignment operator is a value.

The right-side value must be of the same data-type of the left-side variable; otherwise, the compiler will raise an error.

|  |  |
| --- | --- |
| **Operators** | **Functions** |
| **= (Simple Assignment)** | It simply assigns the value of the right operand to the left operand |
| **+= (Add and Assignment)** | This operator adds the value of the right operand to the value of the left operand and assigns the result to the left operand. |
| **-= (Subtract and Assignment)** | This operator subtracts the value of the right operand from the value of the left operand and assigns the result to the left operand. |
| **\*= (Multiply and Assignment)** | This operator multiplies the value of the right operand to the value of the left operand and assigns the result to the left operand. |
| **/= (Divide and Assignment)** | This operator divides the value of the right operand to the value of the left operand and assigns the result to the left operand. |

**For example:**

In this example, we are using all logical operators that are listed above.

1. var x = 20;
2. var y = 40;
4. x = y;
5. console.log("After assignment the value of x is:  " + x);
7. x += y;
8. console.log("x+=y: " + x);
10. x -= y;
11. console.log("x-=y: " + x);
13. x \*= y;
14. console.log("x\*=y: " + x);
16. x /= y;
17. console.log("x/=y: " + x);
19. x %= y;
20. console.log("x%=y: " + x);

**Output:**

After assignment the value of x is: 40

x+=y: 80

x-=y: 40

x\*=y: 1600

x/=y: 40

x%=y: 0

Bitwise Operators

Bitwise operators are used for performing the bitwise operations on binary numerals or bit patterns that involve the manipulation of individual bits. Bitwise operators perform the operation on the binary representation of arguments

Generally, bitwise operators are less used and relevant for the applications and hyper-performance programs.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| **Bitwise AND (&)** | It compares every bit of the first operand to the corresponding bit of the second operand. If both of the bits are 1, then the result bit will set to 1, else it will set to 0. |
| **Bitwise OR (|)** | It compares every bit of the first operand to the corresponding bit of the second operand. If both of the bits are 0, then the result bit will set to 0, else it will set to 1. |
| **Bitwise XOR (^)** | It takes two operands and does XOR on each bit of both operands. It returns 1 if both of the two bits are different and returns 0 in any other case. |
| **Bitwise NOT (~)** | It flips the bits of its operand, i.e., 0 becomes 1 and 1 becomes 0. |
| **Left shift (<<)** | It shifts the value of the left operand to the left by the number of bits specified by the right operand. |
| **Sign-propagating Right shift (>>)** | It shifts the value of the left operand to the right by the number of bits specified by the right operand. This is sign-propagating because the bits that we are adding from the left depends upon the sign of the number (0 represents positive, and 1 represents negative). |
| **Zero-fill right shift** | It accepts two operands. The first operand specifies the number, and the second operator determines the number of bits to shift. Every bit gets shifted towards the right, and the overflowing bits will be discarded. Because the **0-bit**is added from the left, that's why it is a**zero-fill right shift.** |

**For example:**

In this example, we are using all logical operators that are listed above.

1. var x = 70; /\* 70 = 0100 0110 \*/
2. var y = 80; /\* 80 = 0101 0000 \*/
3. var res = 0;
5. console.log("Value of 70 in binary 0100 0110" );
6. console.log("Value of 80 in binary 0101 0000" );

9. res = x & y;       /\* 64 = 0100 0000 \*/
10. console.log("Value of x & y = %d\n", res );
12. res = x | y;       /\* 86 = 0101 0110 \*/
13. console.log("Value of x | y = %d\n", res );
15. res = x ^ y;       /\* 22 = 0001 0110 \*/
16. console.log("Value of x ^ y = %d\n", res );
18. res = ~x;          /\*-71 = -0100 0111 \*/
19. console.log("Value of ~ x = %d\n", res );
21. res = x << 2;     /\* 280 = 1000 11000 \*/
22. console.log("Value of x << 2 = %d\n", res );
23. res = x >> 2;     /\* 17 = 0001 0001 \*/
24. console.log("Value of x >> 2 = %d\n", res );

**Output:**

Value of 70 in binary 0100 0110

Value of 80 in binary 0101 0000

Value of x & y = 64

Value of x | y = 86

Value of x ^ y = 22

Value of ~ x = -71

Value of x << 2 = 280

Value of x >> 2 = 17

Note: The logic of assignment operators is also applied to Bitwise operators, so they become <<=, >>=, &=, |=, ^=.

Miscellaneous Operators

These are the operators that perform different operations in different circumstances.

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **+ (Concatenation Operator)** | It applies to strings and appends the second string to first. |
| **- (Negation Operator)** | It changes the sign of the value. |
| **? (Conditional Operator)** | It is used for representing the conditional expression. It is also called a **ternary operator.** |

Let us try to understand the miscellaneous operators in detail:

The Negation Operator (-)

It is used to change the sign of the value.

**For example:**

1. var num1 = 80;
2. var num2 = -num1;
3. console.log("Value of num1 = " +num1); // It will give 80
4. console.log("Value of num2 = " +num2); // It will give -80

**Output:**

Value of num1 = 80

Value of num2 = -80

The Concatenation Operator (+)

It applies on strings and appends the second string to first. You can understand it by using the following example:

**Example:**

1. var str1 = 'Hello' + 'World';
2. var str2 = 'Welcome ' + 'Back';
3. console.log(str1);
4. console.log(str2);

**Output:**

HelloWorld

Welcome Back

The concatenation operator does not add the space between the strings. It concatenates multiple strings in a single statement. If you want to show the space between your strings, then you have to define it manually. In the above example, the string **"HelloWorld"**does not contain any space, but the second string **"Welcome Back"**has space because we have manually added it.

The Conditional Operator (?)

This operator represents the conditional expression. It is also called the **'ternary operator.'**

**Syntax:**

1. variablename = (condition) ? value1 : value2

Where,

**condition:** It refers to the conditional expression.

**value1:** If the condition is true, then this value will be returned.

**value2:** If the condition is false, then this value will be returned.

**Example:**

1. var num1 = 30;
2. var num2 = 20;
3. var res = num1 > num2 ? "Yes 30 is greater than 20" : "No It's not";
4. console.log(res);

**Output:**

Yes 30 is greater than 20

Type Operators

It is a unary operator that returns the data type of the operand.

**Syntax:**

1. typeof(operand)

You can see the data types and values in the following table that are returned by the **typeof** [operator in JavaScript](https://www.javatpoint.com/javascript-operators):

|  |  |
| --- | --- |
| **Type** | **String Returned by typeof** |
| String | "string" |
| Boolean | "boolean" |
| Number | "number" |
| Object | "object" |

**Example:**

1. var a = 20;
2. var b = **true**;
3. var c = 'Hello';
4. var d = 'true';
5. var e;
6. console.log("Variable a is " +typeof(a));
7. console.log("Variable b is " +typeof(b));
8. console.log("Variable c is a " +typeof(c));
9. console.log("Variable d is a " +typeof(d));
10. console.log("Variable e is " +typeof(e));

**Output:**

Variable a is number

Variable b is boolean

Variable c is a string

Variable d is a string

Variable e is undefined