JavaScript Logical Operators

Summary: in this tutorial, you will learn how to use the JavaScript logical operators including the logical NOT operator(`!`), the logical AND operator (`&&`) and the logical OR operator (`||`).

The logical operators are important in JavaScript because they allow you to compare variables and do something based on the result of that comparison.

For example, if the result of the comparison is `true` , you can run a block of code; if it's `false` , you can execute another code block.

JavaScript provides three logical operators:

- ! (Logical NOT)
- || (Logical OR)
- && (Logical AND)

1) The Logical NOT operator (!)

JavaScript uses an exclamation point `!` to represent the logical NOT operator. The `!` operator can be applied to a single value of any type, not just a Boolean value.

When you apply the `!` operator to a boolean value, the `!` returns `true` if the value is `false` and vice versa. For example:

```
let eligible = false,
    required = true;

console.log(!eligible);
console.log(!required);
```

Output:

```
true
false
```

In this example, the `eligible` is `true` so `!eligible` returns `false` . And since the `required` is `true` , the `!required` returns `false` .

When you apply the `!` operator to a non-Boolean value. The `!` operator converts the value to a boolean value and then negates it.

The following example shows how to use the `!` operator:

```
!a
```

The logical `!` operator works based on the following rules:

- If `a` is `undefined`, the result is `true`.
- If `a` is `null`, the result is `true`.
- If `a` is a number other than `0`, the result is `false`.
- If `a` is `NaN`, the result is `true`.
- If `a` is an object, the result is false.
- If `a` is an empty string, the result is `true`. If `a` is a non-empty string, the result is `false`

The following demonstrates the results of the logical `!` operator when applying to a non-boolean value:

```
console.log(!undefined); // true
console.log(!null); // true
console.log(!20); //false
console.log(!0); //true
console.log(!NaN); //true
console.log(!{}); // false
console.log(!''); //true
console.log(!'OK'); //false
console.log(!false); //true
console.log(!true); //false
```

Double-negation (!!)

Sometimes, you may see the double negation (`!!`) in the code. The `!!` uses the logical NOT operator (`!`) twice to convert a value to its real boolean value.

The result is the same as using the Boolean() function. For example:

```
let counter = 10;
console.log(!!counter); // true
```

The first `!` operator negates the Boolean value of the `counter` variable. If the `counter` is `true`, then the `!` operator makes it false and vice versa.

The second `!` operator negates the result of the first `!` operator and returns the real boolean value of the `counter` variable.

2) The Logical AND operator (&&)

JavaScript uses the double ampersand (`&&`) to represent the logical AND operator. The following expression uses the `&&` operator:

```
let result = a && b;
```

If `a` can be converted to `true`, the `&&` operator returns the `b`; otherwise, it returns the `a`. This rule is applied to all boolean values.

The following truth table illustrates the result of the `&&` operator when it is applied to two Boolean values:

a	**b**	**a && b**
true	true	true
true	false	false
false	true	false
false	false	false

The result of the `&&` operator is true only if both values are `true`, otherwise, it is `false`. For example:

```
let eligible = false,
    required = true;

console.log(eligible && required); // false
```

In this example, the `eligible` is `false`, therefore, the value of the expression `eligible && required` is `false`.

See the following example:

```
let eligible = true,
    required = true;

console.log(eligible && required); // true
```

In this example, both `eligible` and `required` are `true`, therefore, the value of the expression `eligible && required` is `true`.

Short-circuit evaluation

The `&&` operator is short-circuited. It means that the `&&` operator evaluates the second value only if the first one doesn't suffice to determine the value of an expression. For example:

```
let b = true;
let result = b && (1 / 0);
console.log(result);
```

Output:

```
Infinity
```

In this example, `b` is `true` therefore the `&&` operator could not determine the result without further evaluating the second expression (`1/0`).

The result is `Infinity` which is the result of the expression (`1/0`). However:

```
let b = false;
let result = b && (1 / 0);
console.log(result);
```

Output:

```
false
```

In this case, `b` is `false` , the `&&` operator doesn't need to evaluate the second expression because it can determine the final result as `false` based value of the first value.

The chain of && operators

The following expression uses multiple `&&` operators:

```
let result = value1 && value2 && value3;
```

The `&&` operator carries the following:

- Evaluates values from left to right.
- For each value, convert it to a boolean. If the result is `false` , stops and returns the original value.
- If all values are truthy values, return the last value.

In other words, The `&&` operator returns the first falsey value or the last truthy value.

> If a value can be converted to `true` , it is called a truthy value. If a value can be converted to `false` , it is a called a falsey value.

3) The Logical OR operator (||)

JavaScript uses the double-pipe `||` to represent the logical OR operator. You can apply the `||` operator to two values of any type:

```
let result = a || b;
```

If `a` can be converted to `true`, returns `a`; else, returns `b`. This rule is also applied to boolean values.

The following truth table illustrates the result of the `||` operator based on the value of the operands:

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

The `||` operator returns `false` if both values are evaluated to `false`. In case either value is `true`, the `||` operator returns `true`. For example:

```
let eligible = true,
    required = false;

console.log(eligible || required); // true
```

See another example:

```
let eligible = false,
    required = false;

console.log(eligible || required); // false
```

In this example, the expression `eligible || required` returns `false` because both values are `false`.

The || operator is also short-circuited

Similar to the `&&` operator, the `||` operator is short-circuited. It means that if the first value evaluates to `true`, the `&&` operator doesn't evaluate the second one.

The chain of || operators

The following example shows how to use multiple || operators in an expression:

```
let result = value1 || value2 || value3;
```

The `||` operator does the following:

- Evaluates values from left to right.
- For each value, convert it to a boolean value. If the result of the conversion is `true`, stops and returns the value.
- If all values have been evaluated to `false`, returns the last value.

In other words, the chain of the `||` operators returns the first truthy value or the last one if no truthy value is found.

Logical operator precedence

When you mix logical operators in an expression, the JavaScript engine evaluates the operators based on a specified order. This order is called the *operator precedence*.

In other words, the operator precedence is the order of evaluating logical operators in an expression.

The precedence of the logical operator is in the following order from the highest to the lowest:

1. Logical NOT (!)
2. Logical AND (&&)
3. Logical OR (||)

Summary

- The NOT operator (`!`) negates a boolean value. The (`!!`) converts a value into its real boolean value.
- The AND operator (`&&`) is applied to two Boolean values and returns true if both are true.
- The OR operator (`||`) is applied to two Boolean values and returns `true` if one of the operands is `true` .
- Both `&&` and `||` operator are short-circuited. They can also be applied to non-Boolean values.
- The logical operator precedence from the highest to the lowest is `!` , `&&` and `||` .
- The AND operator returns the first falsey value or the last truthy value.
- The || operator returns the first falsey value.