Relational and Boolean Operators

In this lesson, we'll cover relational and Boolean operators.
Let's begin!

WE'LL COVER THE FOLLOWING

Relational operators

Relational operators

Besides the equality operators, JavaScript defines four relational operators, similar to other programming languages.

These are less than (`<`), greater than (`>`), less than or equal to (`<=`), and greater than or equal to (`>=`). Each returns a Boolean value representing the result of the comparison.

Operand rules

These operands work according to these rules:

> 1. If both operands are numbers, a numeric comparison is carried out.
> 2. If both operands are strings, the character codes of each corresponding character in the string are compared.
> 3. If one operand is a number, the other operand is converted to a number, and a numeric comparison is performed.
> 4. If an operand is an object, `valueOf()` is called, and its result is used to perform the comparison according to the previous rules. If `valueOf()` is not available, `toString()` is invoked, and the returned value is used according to the previous rules.
> 5. If an operand is a Boolean, it is converted to a number and the comparison is performed.

Examples

Here are several examples that demonstrate these rules:

index.js

```
console.log(3 < 4);        // true
console.log(3 > 4);        // false
console.log(3 <= "4");     // true
console.log(13 <= "4");    // false
console.log("13" <= 4);    // false
console.log("13" <= "4"); // true
console.log("x" < 3);      // false
console.log("x" >= 3);    // false
```

The last two expressions both result false, because `"x"` is converted to NaN, and as you know, NaN is not equal to anything, including itself.

Boolean operators

JavaScript supports three Boolean operators: logical NOT (`!`), logical AND (`&&`), and logical OR (`||`).

While in other programming languages Boolean operators work on only Boolean values, JavaScript allows using them on values of any types.

The logical `NOT` operator is a unary operator, and it always returns a Boolean value, regardless of the data type it is used on. First, it converts the operand to a Boolean value and then negates it.

Rules for the logical NOT (`!`) operator

Logical NOT works according to these rules:

> 1. If the operand is undefined or NaN, true is returned.
> 2. An object results in false.
> 3. A null value results in true.
> 4. An empty string results true, while a nonempty string returns false.
> 5. If the operand is the number 0, true is returned.
> 6. If the operand is any number other than 0 (including Infinity), false is returned.

Examples

Let’s see a few examples:

index.js

```
console.log(!true);      // false
console.log(!"hello");   // false
console.log(!null);      // true
console.log(!undefined); // true
console.log(!"");        // true
console.log(!23.2);      // false
console.log(!NaN);       // true
```

Provided its operands are Boolean values, logical AND returns true if and only if both of its operands are true. This operator is a short-circuited operation, meaning that if the first operand determines the result, the second operand is never evaluated.

In the case of logical AND, if the first operand results false, it immediately returns false, because no matter what the value of the second operand is, the result can’t be true.

You are already used to the fact that the **loosely-typed** nature of JavaScript often allows using any type of values.

Accordingly, logical AND can be used with any type of operand, not just Boolean values.

Rules for the logical AND (`&&`) operator

When either operand is not a primitive Boolean, logical AND does not always return a Boolean value; instead, it applies these rules:

1. If either operand is NaN, undefined, or null, then NaN, undefined, or null is returned, respectively.
2. If the first operand is an object, then the second operand is always returned.
3. If the second operand is an object, then the object is returned only if the first operand evaluates to true.

4. If both operands are objects, then the second operand is returned.

Examples

Here are a few examples that demonstrate these rules:

> 📜**NOTE**: Line 6 in the code below throws an error ❌, as expected

index.js

```
var obj1 = new Object(1);
var obj2 = new Object(2);
console.log(true && false); // false
console.log(false && obj1); // false
console.log(obj1 && obj2);  // Number {}
console.log(true && undef);  // Never executes
```

Provided its operands are Boolean values, logical OR returns true if at least one of its operands is true. Just like logical AND, it is short-circuited and if the first operand is true, it immediately returns true, leaving the second operand unevaluated.

Rules for logical OR (`||`) operator

When either operand is not a primitive Boolean, just like logical AND, logical OR does not always return a Boolean value; instead, it applies these rules:

1. If either operand is NaN, undefined, or null, then NaN, undefined, or null is returned, respectively.
2. If the first operand is an object, then the first operand is returned.
3. If the first operand evaluates to false, then the second operand is returned.
4. If both operands are objects, then the first operand is returned.

Examples

This short code snippet demonstrates the **logical OR** rules:

> 📃 **Note:** Uncommenting line 7 will give an error ❌

index.js

```
var obj1 = new Object(1);
var obj2 = new Object("x");
console.log(true || false);  // true
console.log(false || obj2);  // String
console.log(obj1 || obj2);   // Number
console.log(null || NaN);    // NaN
//console.log(false || undef); // Never executes
```

In the *next lesson*, we'll visit the bitwise operators!

See you there!