**Operators in depth**

In this reading, you will learn about additional operators, operator precedence and operator associativity. I'll also provide you with some examples of logical operators.

**1. Additional operators**

* Logical AND operator: **&&**
* Logical OR operator: **||**
* Logical NOT operator: **!**
* The modulus operator: **%**
* The equality operator: **==**
* The strict equality operator: **===**
* The inequality operator: **!=**
* The strict inequality operator: **!==**
* The addition assignment operator: **+=**
* The concatenation assignment operator: **+=** (it's the same as the previous one - more on that later)

**The logical AND operator in JavaScript: &&**

The **logical AND operator** is, for example, used to confirm if multiple comparisons will return true.

In JavaScript, this operator consists of two ampersand symbols together: **&&**.

Let's say you're tasked with coming up with some code that will check if the **currentTime** variable is between **9 a.m.** and **5 p.m**. The code needs to console.log **true** if **currentTime > 9** **and** if **currentTime < 17**.

Here's a solution:

var currentTime = 10;

console.log(currentTime > 9 && currentTime < 17);

How does this code work?

First, on line one, I set the **currentTime** variable, and assign the value of **10** to it.

Next, on line two I console log two comparisons:

**currentTime > 9**

**currentTime < 17**

I also use the **&&** logical operator to join the two comparisons.

Effectively, my code is interpretted as the following:

console.log(10 > 9 && 10 < 17);

The comparison of **10 > 9** will return **true**.

Also, the comparison of **10 < 17** will return **true**.

This means I can further re-write the line two of my solution as follows:

console.log(true && true);

In essence, this is how my code works.

Now, the question is, what will be the result of **console.log(true && true)**?

To understand the answer, you need to know the behavior of the **&&** logical operator.

The **&&** logical operator returns a single value: the boolean **true** or **false**, based on the following rules:

* It returns **true** if both the values on its right and on its left are evaluated to **true**
* It returns **false** in all the other instances

In other words:

**console.log(true && true)** will output: **true**

**console.log(true && false)** will output: **false**

**console.log(false && true)** will output: **false**

**console.log(false && false)** will output: **false**

**The logical OR operator in JavaScript: ||**

The logical OR operator in JavaScript consists of two pipe symbols together: **||**.

It is used when you want to check if at least one of the given comparisons evaluates to **true**.

Consider the following task: You need to write a program in JavaScript which will return **true** if the value of the **currentTime** variable is not between **9** and **17**. Put differently, your code needs to console.log **true** if the value of the variable **currentTime** is either less than **9** or greater than **17**.

Here's a solution:

var currentTime = 7;

console.log(currentTime < 9 || currentTime > 17);

In line one of the code I assign the number **7** to the variable **currentTime**.

On line two, I console log the result of checking if either **currentTime < 9** or **currentTime > 17** will evaluate to **true**.

It's the same as this:

var currentTime = 7;

console.log(true || false);

Here are the rules of how the **||** operator evaluates given values:

**console.log(true || true)** will output: **true**

**console.log(true || false)** will output: **true**

**console.log(false || true)** will output: **true**

**console.log(false || false)** will output: **false**

The logical OR operator will always return **true**, except when both sides evaluate to **false**. In other words, for the logical OR operator to return **false**, the results of both comparisons **must** return false.

Going back to the example of checking if either **currentTime < 9** or **currentTime > 17**, this makes sense: the only time you will get **false** is when the value stored in the **currentTime** variable is greater than **9** and less then **17**.

**The logical NOT operator: !**

In JavaScript, the logical NOT operator's symbol is the exclamation mark: **!**.

You can think of the **!** operator as a switch, which flips the evaluated boolean value from **true** to **false** and from **false** to **true**.

For example if I assign the boolean value of **true** to the **petHungry** variable:

**var petHungry = true;**

...then I can console log the fact that the pet is no longer hungry by using the **!** operator to flip the boolean value stored inside of the **petHungry** variable, like so:

**console.log('Feeding the pet'); console.log("Pet is hungry: ", !petHungry); console.log(petHungry);**

This is the output of the above code:

Pet is hungry: true

Feeding the pet

Pet is hungry: false

true

The reason for the changed output in the console is because you have flipped the value stored inside the **petHungry** variable, from **true** to **false**.

Notice, however, that the code on line five of the example above still outputs **true** - that's due to the fact that I didn't reassign the value of the **petHungry** variable.

Here's how I could permanently change the value stored in the **petHungry** variable from **true** to **false**:

var petHungry = true;

petHungry = !petHungry;

In this example, I first assign the value of **true** to the new variable of **petHungry**. Then, on line two, I assign the opposite value, the **!true** - read: not true - to the existing **petHungry** variable.

**The modulus operator: %**

The modulus operator is another mathematical operator in JavaScript. It returns the remainder of division.

To demonstrate how it works, imagine that a small restaurant that has 4 chairs per table, and a total of 5 tables, suddenly receives 22 guests.

How many guests will not be able to sit down in the restaurant?

You can use the modulus operator to solve this.

console.log(22 % 5); // 2

The output is **2**, meaning, when I divide **22** and **5**, I get a **4**, and the remainder is **2**, meaning, there are **2** people who couldn't get a place in this restaurant.

**The equality operator, ==**

The equality operator checks if two values are equal.

For example, this comparison returns **true**: **5 == 5**. Indeed, it is true that 5 is equal to 5.

Here's an example of the equality operator returning **false**: **5 == 6**. Indeed, it is true that 5 is not equal to 6.

Additionally, even if one of the compared values is of the number type, and the other is of the string type, the returned value is still **true**: **5 == "5"**.

This means that the equality operator compares only the values, but not the types.

**The strict equality operator, ===**

The strict equality operator compares for both the values and the data types.

With the strict equality operator, comparing **5 === 5** still returns **true**. The values on each side of the strict equality operator have the same value and the same type. However, comparing **5 == "5"** now returns **false**, because the values are equal, but the data type is different.

**The inequality operator, !=**

The inequality operator checks if two values are not the same, but it does not check against the difference in types.

For example, **5 != "5"** returns false, because it's false to claim that the number 5 is not equal to number 5, even though this other number is of the string data type.

**The strict inequality operator !==**

For the strict inequality operator to return **false**, the compared values have to have the same value and the same data type.

For example, **5 !== 5** returns **false** because it is false to say that the number 5 is not of the same value and data type and another number 5.

However, comparing the number 5 and the string 5, using the strict inequality operator, returns **true**.

console.log(5 !== "5")

**2. Using the + operators on strings and numbers**

**Combining two strings using the + operator**

The **+** operator, when used with number data type, adds those values together.

However, the **+** operator is also used to join string data type together.

For example:

"inter" + "net" // "internet"

"note" + "book" // "notebook"

If the **+** operator is used to join strings, then it is referred to as the *concatenation* operator, and you'll say that it's used to *concatenate* strings.

When used with numbers, the **+** operator is the **addition operator**, and when used with strings, the **+** operator is the **concatenation operator**.

**Combining strings and numbers using the + operator**

But what happens when one combines a string and a number using the **+** operator?

Here's an example:

365 + " days" // "365 days"

12 + " months" // "12 months"

Here, JavaScript tries to help by converting the numbers to strings, and then **concatenating the number and the string together**, ending up with **a string value**.

The process of this "under-the-hood" conversion of values in JavaScript is referred to as "coercion". JavaScript *coerces* a number value to a string value - so that it can run the **+** operator on disparate data types.

The process of coercion can sometimes be a bit unexpected.

Consider the following example:

1 + "2"

What will be the result of **1 + "2"**?

Note that the value of **1** is of the number data type, and the value of **"2"** is of the string data type, and so JavaScript will coerce the number **1** to a string of **"1"**, and then concatenate it with the string of **"2"**, so the result is a string of **"12"**.

**The addition assignment operator, +=**

The addition assignment operator is used when one wants to accumulate the values stored in a variable.

Here's an example: You are counting the number of overtime hours worked in a week.

You don't have to specify the type of work, you just want to count total hours.

You might code a program to track it, like this:

var mon = 1;

var tue = 2;

var wed = 1;

var thu = 2;

var fri = 3;

console.log(mon + tue + wed + thu + fri); // 9

You can simplify the above code by using the addition assignment operator, as follows:

var overtime = 1;

overtime += 2;

overtime += 1;

overtime += 2;

overtime += 3;

console.log(overtime); // 9

Using the addition assignment operator reduces the lines of your code.

**The concatenation assignment operator, +=**

This operator's syntax is exactly the same as the addition assignment operator. The difference is in the data type used:

var longString = "";

longString += "Once";

longString += " upon";

longString += " a";

longString += " time";

longString += "...";

console.log(longString); // "Once upon a time..."

**Operator precedence and associativity**

Operator precedence is a set of rules that determines which operator should be evaluated first.

Consider the following example:

1 \* 2 + 3

The result of the above code is 5, because the multiplication operator has precedence over the addition operator.

Operator associativity determines how the precedence works when the code uses operators with the same precedence.

There are two kinds:

* left-to-right associativity
* right-to-left associativity

For example, the assignment operator is right-to-left associative, while the greater than operator is left-to-right associative:

var num = 10; // the value on the right is assigned to the variable name on the left

5 > 4 > 3; // the 5 > 4 is evaluated first (to `true`), then true > 3 is evaluated to `false`, because the `true` value is coerced to `1`

Go to next item

**Completed**

Like

Dislike

Report an issue