Title: How to perform arithmatic operations in javascript?

Answer:

Arithmetic operators are symbols that indicate a mathematical operation and return a value. In the equation `3 + 7 = 10`, the `+` is syntax that stands for addition.

JavaScript has many familiar operators from basic math, as well as a few additional operators specific to programming.

Here is a reference table of JavaScript arithmetic operators.

Operator	Syntax	Example	Definition
Addition	`+`	`x + y`	Sum of `x` and `y`
Subtraction	`-`	`x - y`	Difference of `x` and `y`
Multiplication	`*`	`x * y`	Product of `x` and `y`
Division	`/`	`x / y`	Quotient of `x` and `y`
Modulo	`%`	`x % y`	Remainder of `x / y`
Exponentiation	`**`	`x ** y`	`x` to the `y` power
Increment	`++`	`x++`	`x` plus one
Decrement	`--`	`x--`	`x` minus one

We will go into more detail on each of these operators throughout this article.

Addition and Subtraction

Addition and **subtraction** operators are available in JavaScript, and can be used to find the sum and difference of numerical values. JavaScript has a built-in calculator, and mathematical operations can be done directly in the console.

We can do some simple addition with numbers, for example adding `10` and `20`, using the plus sign (`+`).

```
10 + 20;

Output
30
```

In addition to doing math with plain numbers, we can also assign numbers to variables and perform the same calculations. In this case, we will assign the numerical values to `x` and `y` and place the sum in `z`.

```
// Assign values to x and y
let x = 10;
let y = 20;

// Add x and y and assign the sum to z
let z = x + y;

console.log(z);

Output
30
```

Similarly, we use the minus sign (-) to subtract numbers or variables representing numbers.

```
// Assign values to x and y
let x = 10;
let y = 20;

// Subtract x from y and assign the difference to z
let z = y - x;

console.log(z);

Output
10
```

We can also add and subtract with negative numbers and floats (decimals).

```
// Assign values to x and y
let x = -5.2;
let y = 2.5;

// Subtract y from x and assign the difference to z
let z = x - y;

console.log(z);

Output
-7.7
```

One interesting thing to note and be aware of in JavaScript is the result of adding a number and a string. We know that `1 + 1` should equal `2`, but this equation will have unexpected results.

```
let x = 1 + "1";

console.log(x);
typeof x;

Output
11 'string'
```

Instead of adding the two numbers, JavaScript will convert the entire statement into a string and concatenate them together. It's important to be careful with JavaScript's dynamically-typed nature, as it can have undesired outcomes.

A common reason to use addition or subtraction in JavaScript would be to scroll to an id minus the height in pixels of a fixed navigation bar.

```
function scrollToId() {
    const navHeight = 60;
    window.scrollTo(0, window.pageYOffset - navHeight);
}

window.addEventListener('hashchange', scrollToId);
```

In the above example, clicking on an id will scroll to 60 pixels above the id.

Addition and subtraction are two of the most common mathematical equations you will use in JavaScript.

Multiplication and Division

Multiplication and division operators are also available in JavaScript, and are used to find the product and quotient of numerical values.

An asterisk (*) is used to represent the multiplication operator.

```
// Assign values to x and y
let x = 20;
let y = 5;

// Multiply x by y to get the product
let z = x * y;

console.log(z);

Output
100
```

Multiplication might be used to calculate the price of an item after applying sales tax.

```
const price = 26.5;    // Price of item before tax
const taxRate = 0.082; // 8.2% tax rate

// Calculate total after tax to two decimal places
let totalPrice = price + (price * taxRate);
totalPrice.toFixed(2);

console.log("Total:", totalPrice);

Output
Total: 28.67
```

A slash (`/`) is used to represent the division operator.

```
// Assign values to x and y
let x = 20;
let y = 5;

// Divide y into x to get the quotient
let z = x / y;

console.log(z);

Output
4
```

Division is particularly useful when calculating time, such as finding the number of hours in a quantity of minutes, or when calculating the percent of correct answers completed in a test.

Modulo

One arithmetic operator that is slightly less familiar is the modulo (sometimes known as modulus) operator, which calculates the remainder of a quotient after division. Modulo is represented by a percentage sign (`%`).

As an example, we know that `3` goes into `9` exactly three times, and there is no remainder.

```
9 % 3;

Output
0
```

We can use the modulo operator to determine whether a number is even or odd, as seen with this function:

```
// Initialize function to test if a number is even
const isEven = x => {
    // If the remainder after dividing by two is 0, return true
    if (x % 2 === 0) {
        return true;
    }
    // If the number is odd, return false
    return false;
}

// Test the number
isEven(12);

Output
true
```

In the above example, `12` divides evenly into `2`, therefore it is an even number.

Often in programming, modulo is used in conjunction with conditional statements for flow control.

Exponentiation

Exponentiation is one of the newer operators in JavaScript, and it allows us to calculate the power of a number by its exponent. The syntax for exponentiation is two asterisks in a row (`**`).

10 to the fifth power, or 10^5, is written like this:

```
10 ** 5;

Output
100000
```

`10 ** 5` represents the same as `10` multiplied by `10` five times:

```
10 * 10 * 10 * 10 * 10;
```

Another way of writing this is with the `Math.pow()` method.

```
Math.pow(10, 5);

Output
100000
```

Using the exponentiation operator is a concise way of finding the power of a given number, but as usual, it is important to keep consistent with the style of your code base when choosing between a method and an operator.

Increment and Decrement

Increment and decrement operators increase or reduce the numerical value of a variable by one. They are represented by two plus signs (`++`) or two minus signs (`--`), and are often used with loops (https://www.digitalocean.com/community/tutorials/how-to-construct-for-loops-in-javascript#for-loop).

Note that increment and decrement operators can only be used on variables; attempting to use them on a raw number will result in an error.

```
7++

Output
Uncaught ReferenceError: Invalid left-hand side expression in postfix operat
ion
```

Increment and decrement operators can be classified as a prefix or postfix operation, depending on whether or not the operator is placed before or after the variable.

First, we can text the prefix incrementation, with `++x` .

```
// Set a variable
let x = 7;

// Use the prefix increment operation
let prefix = ++x;

console.log(prefix);

Output
8
```

The value of `x` was increased by one. To see the difference, we will test the postfix incrementation, with `y++` .

```
// Set a variable
let y = 7;

// Use the prefix increment operation
let postfix = y++;

console.log(postfix);

Output
7
```

The value of `y` was not increased in the postfix operation. This is because the value will not be incremented until after the expression has been evaluated. Running the operation twice will then increment the value.

```
let y = 7;

y++;
y++;

console.log(y);

Output
8
```

The increment or decrement operator will be seen most often in a loop. In this `for` loop example, we will run the operation ten times, starting with `0`, and increasing the value by `1` with each iteration.

```
// Run a loop ten times
for (let i = 0; i < 10; i++) {
  console.log(i);
}

Output
0
1
2
3
4
5
6
7
8
9
```

The code above shows an iteration through a loop that is achieved through using the increment operator.

We can think of `x++` as shorthand for `x = x + 1`, and `x--` as shorthand for `x = x - 1`.

Tags: javascript, numbers