Learn web development

Basic math in JavaScript — numbers and operators

see all contributors

Previous Overview: First steps Next

At this point in the course we discuss math in JavaScript — how we can use operators and other features to successfully manipulate numbers to do our bidding.

Prerequisites:	Basic computer literacy, a basic understanding of HTML and CSS, an understanding of what JavaScript is.
Objective:	To gain familiarity with the basics of Math in JavaScript.

Everybody loves math

Ok, maybe not. Some of us like math, some of us have hated math ever since we had to learn multiplication tables and long division in school, and some of us sit somewhere in between the two. But none of us can deny that math is a fundamental part of life that we can't get very far without. This is especially true when we are learning to program JavaScript (or any other language for that matter) — so much of what we do relies on processing numerical data, calculating new values, etc. that you won't be surprised to learn that JavaScript has a full-featured set of math functions available.

This article discusses only the basic parts that you need to know now.

Types of numbers

In programming, even the humble decimal number system that we all know so well is more complicated than you might think. We use different terms to describe different types of decimal numbers, for example:

- **Integers** are whole numbers, e.g. 10, 400, or -5.
- **Floating point numbers** (floats) have decimal points and decimal places, for example 12.5, and 56.7786543.
- **Doubles** are a specific type of floating point number that have greater precision than standard floating point numbers (meaning that they are accurate to a greater number of decimal places).

We even have different types of number system! Decimal is base 10 (meaning it uses 0–9 in each column), but we also have things like:

- **Binary** — The lowest level language of computers; 0s and 1s.
- **Octal** — Base 8, uses 0–7 in each column.
- **Hexadecimal** — Base 16, uses 0–9 and then a–f in each column. You may have encountered these numbers before when setting colours in CSS.

Before you start to get worried about your brain melting, stop right there! For a start, we are just going to stick to decimal numbers throughout this course; you'll rarely come across a need to start thinking about other types, if ever.

The second bit of good news is that unlike some other programming languages, JavaScript only has one data type for numbers, you guessed it, `Number`. This means that whatever type of numbers you are dealing with in JavaScript, you handle them in exactly the same way.

It's all numbers to me

Let's have a quick play with some numbers to reacquaint ourselves with the basic syntax we need. Enter the commands listed below into your developer tools JavaScript console, or use the simple built in console seen below if you'd prefer.

>

Open in new window

1. First of all, let's declare a couple of variables and initialize them with an integer and a float, respectively, then type the variable names back in to check that everything is in order:

   ```
   var myInt = 5;
   var myFloat = 6.667;
   myInt;
   myFloat;
   ```

2. Number values are typed in without quote marks — try declaring and initializing a couple more variables containing numbers. Before you move on.
3. Now let's check that both our original variables are of the same datatype. There is an operator called `typeof` in JavaScript that does this. Enter the below two lines as shown:

   ```
   typeof myInt;
   typeof myFloat;
   ```

   You should get `"number"` returned in both cases — this makes things a lot easier for us than if different numbers had different data types, and we had to deal with them in different ways. Phew!

Arithmetic operators

Arithmetic operators are the basic operators that we use to do sums:

Operator	Name	Purpose	Example

Operator	Name	Purpose	Example
+	Addition	Adds two numbers together.	`6 + 9`
-	Subtraction	Subtracts the right number from the left.	`20 - 15`
*	Multiplication	Multiplies two numbers together.	`3 * 7`
/	Division	Divides the left number by the right.	`10 / 5`
%	Remainder (sometimes called modulo)	Returns the remainder left over after you've shared the left number out into a number of integer portions equal to the right number.	`8 % 3` (returns 2, as three goes into 8 twice, leaving 2 left over.)

Note: You'll sometimes see numbers involved in sums referred to as operands.

We probably don't need to teach you how to do basic math, but we would like to test your understanding of the syntax involved. Try entering the examples below into your developer tools JavaScript console, or use the simple built in console seen earlier if you'd prefer, to familiarize yourself with the syntax.

1. First try entering some simple examples of your own, such as

```
10 + 7
9 * 8
60 % 3
```

2. You can also try declaring and initializing some numbers inside variables, and try using those in the sums — the variables will behave exactly like the values they hold for the purposes of the sum. For example:

```
var num1 = 10;
var num2 = 50;
9 * num1;
num2 / num1;
```

3. Last for this section, try entering some more complex expressions, such as:

```
5 + 10 * 3;
num2 % 9 * num1;
num2 + num1 / 8 + 2;
```

Some of this last set of sums might not give you quite the result you were expecting; the below section might well give the answer as to why.

Operator precedence

Let's look at the last example from above, assuming that `num2` holds the value 50 and `num1` holds the value 10 (as originally stated above):

```
num2 + num1 / 8 + 2;
```

As a human being, you may read this as *"50 plus 10 equals 60"*, then *"8 plus 2 equals 10"*, and finally *"60 divided by 10 equals 6"*.

But the browser does *"10 divided by 8 equals 1.25"*, then *"50 plus 1.25 plus 2 equals 53.25"*.

This is because of **operator precedence** — some operators will be applied before others when calculating the result of a sum (referred to as an expression, in programming). Operator precedence in JavaScript is the same as is taught in math classes in school — Multiply and divide are always done first, then add and subtract (the sum is always evaluated from left to right).

If you want to override operator precedence, you can put parentheses round the parts that you want to be explicitly dealt with first. So to get a result of 6, we could do this:

```
(num2 + num1) / (8 + 2);
```

Try it and see.

Note: A full list of all JavaScript operators and their precedence can be found in Expressions and operators.

Increment and decrement operators

Sometimes you'll want to repeatedly add or subtract one to/from a numeric variable value. This can be conveniently be done using the increment (++) and decrement(--) operators. We used ++ in our "Guess the number" game back in our first splash into JavaScript article, when we added 1 to our `guessCount` variable to keep track of how many guesses the user has left after each turn.

```
guessCount++;
```

Note: They are most commonly used in loops, which you'll learn about later on in the course. For example, say you wanted to loop through a list of prices, and add sales tax to each one. You'd use a loop to go through each value in turn and do the necessary calculation for adding the sales tax in each case. The incrementor is used to move to the next value when needed. We've actually provided a simple example showing how this is done — check it out live, and look at the source code to see if you can spot the incrementors! We'll look at loops in detail later on in the course.

Let's try playing with these in your console. For a start, note that you can't apply these directly to a number, which might seem strange, but we are assigning a variable a new updated value, not operating on the value itself. The following will return an error:

```
3++;
```

So, you can only increment an existing variable. Try this:

```
var num1 = 4;
num1++;
```

Ok, strangeness number 2! When you do this, you'll see a value of 4 returned — this is because the browser returns the current value, *then* increments the variable. You can see that it's been incremented if you return the variable value again:

```
num1;
```

The same is true of -- : try the following

```
var num2 = 6;
num2--;
num2;
```

Note: You can make the browser do it the other way round — increment/decrement the variable *then* return the value — by putting the operator at the start of the variable instead of the end. Try the above examples again, but this time use `++num1` and `--num2`.

Assignment operators

Assignment operators are operators that assign a value to a variable. We have already used the most basic one, =, loads of times — it simply assigns the variable on the left the value stated on the right:

```
var x = 3; // x contains the value 3
var y = 4; // y contains the value 4
x = y; // x now contains the same value y contains, 4
```

But there are some more complex types, which provide useful shortcuts to keep your code neater and more efficient. The most common are listed below:

Operator	Name	Purpose	Example	Shortcut for
`+=`	Addition assignment	Adds the value on the right to the variable value on the left, then returns the new variable value	`x = 3;` `x += 4;`	`x = 3;` `x = x + 4;`
`-=`	Subtraction assignment	Subtracts the value on the right from the variable value on the left, and returns the new variable value	`x = 6;` `x -= 3;`	`x = 6;` `x = x - 3;`
`*=`	Multiplication assignment	Multiples the variable value on the left by the value on the right, and returns the new variable value	`x = 2;` `x *= 3;`	`x = 2;` `x = x * 3;`
`/=`	Division assignment	Divides the variable value on the left by the value on the right, and returns the new variable value	`x = 10;` `x /= 5;`	`x = 10;` `x = x / 5;`

Try typing some of the above examples into your console, to get an idea of how they work. See if you can guess what the value is before you type the second line in, in each case.

Note that you can quite happily use other variables on the right hand side of each expression, for example:

```
var x = 3; // x contains the value 3
var y = 4; // y contains the value 4
x *= y; // x now contains the value 12
```

Note: There are lots of other assignment operators available, but these are the basic ones you should learn now.

Active learning: sizing a canvas box

In this exercise we are going to get you to fill in some numbers and operators to manipulate the size of a box. The box is drawn using a browser API called the Canvas API. There is no need to worry about how this works — just concentrate on the math for now. The width and height of the box (in pixels) are defined by the variables x and y, which are initially both given a value of 50.

The rectangle is 50px wide and 50px high.

```
var x = 50; var y = 50;

// Edit the two lines below here ONLY
x = 50;
y = 50;

ctx.fillStyle = 'green';
ctx.fillRect(10, 10, x, y);
```

Reset

Open in new window

In the editable code box above, there are two lines marked clearly with a comment that we'd like you to update to make the box grow/shrink to certain sizes, using certain operators and/or values in each case. We'd like you to answer the following questions:

- Change the line that calculates x so the box is still 50px wide, and the 50 is calculated using the numbers 43 and 7, and an arithmetic operator.
- Change the line that calculates y so the box is 75px high, and the 75 is calculated using the numbers 25 and 3, and an arithmetic operator.
- Change the line that calculates x so the box is 250px wide, and the 250 is calculated using two numbers and the remainder (modulo) operator.
- Change the line that calculates y so the box is 150px high, and the 150 is calculated using three numbers, and the subtraction and divison operators.

- Change the line that calculates x so the box is 200px wide, and the 200 is calculated using the number 4 and an assignment operator.
- Change the line that calculates y so the box is 200px high, and the 200 is calculated using the numbers 50 and 3, the multiplication operator, and the addition assignment operator.

Don't worry if you totally mess the code up. You can always press the *Reset* button to get things working again. After you've answered all the above questions correctly, you can feel free to play with the code some more, or set your friends/classmates some challenges.

Comparison operators

Sometimes we will want to run true/false tests, then act accordingly depending on the result of that test — to do this we use **comparison operators**.

Operator	Name	Purpose	Example
`===`	Strict equality	Tests whether the left and right values are identical to one another	`5 === 2 + 4`
`!==`	Strict-non-equality	Tests whether the left and right values **not** identical to one another	`5 !== 2 + 3`
`<`	Less than	Tests whether the left value is smaller than the right one.	`10 < 6`
`>`	Greater than	Tests whether the left value is greater than the right one.	`10 > 20`
`<=`	Less than or equal to	Tests whether the left value is smaller than or equal to the right one.	`3 <= 2`
`>=`	Greater than or equal to	Tests whether the left value is greater than or equal to the right one.	`5 >= 4`

Note: You may see some people using `==` and `!=` in their code for equality and non-equality — these are valid operators in JavaScript, but they differ from `===`/`!==` — the former test whether the values are the same, but the datatype can be different, whereas the latter strict versions test if the value and the dataype are the same. The strict versions tend to result in less errors going undetected, so we are recommending that you use those.

If you try entering some of these values in a console, you'll see that they all return `true`/`false` values — those booleans we mentioned in the last article. These are very useful as they allow us to make decisions in our code — these are used every time we want to make a choice of some kind, for example:

- Displaying the correct text label on a button depending on whether a feature is turned on or off.

- Displaying a game over message if a game is over, or a victory message if the game has been won.
- Displaying the correct seasonal greeting depending what holiday season it is.
- Zooming a map in or out depending on what zoom level is selected.

We'll look at how to code such logic when we look at conditional statements in a future article. For now, let's look at a quick example:

```html
<button>Start machine</button>
<p>The machine is stopped.</p>
```

```javascript
var btn = document.querySelector('button');
var txt = document.querySelector('p');

btn.addEventListener('click', updateBtn);

function updateBtn() {
  if (btn.textContent === 'Start machine') {
    btn.textContent = 'Stop machine';
    txt.textContent = 'The machine has started!';
  } else {
    btn.textContent = 'Start machine';
    txt.textContent = 'The machine is stopped.';
  }
}
```

Start machine

The machine is stopped.

Open in new window

You can see the equality operator being used just inside the `updateBtn()` function. In this case we are not testing if two mathematical expressions have the same value — we are testing whether the text content of a button contains a certain string — but it is still the same principle at work. If the button is currently saying "Start machine" when it is pressed, we change its label to "Stop machine", and update the label as appropriate. If the button is currently saying "Stop machine" when it is pressed, we swap the display back again.

Note: Such a control that swaps between two states is generally referred to as a **toggle**. It toggles between one state and another — light on, light off, etc.

Summary

In this article we have covered the fundamental information you need to know about numbers in JavaScript, for now. You'll see numbers used again and again, all the way through your JavaScript learning, so it's a good idea to get this out of the way now. If you are one of those people that doesn't enjoy math, you can take comfort in the fact that this chapter was pretty short.

In the next article, we'll explore text, and how JavaScript allows us to manipulate it.

Note: If you do enjoy math and want to read more about how it is implemented in JavaScript, you can find a lot more detail in MDN's main JavaScript section. Great places to start are our Numbers and dates and Expressions and operators articles.

Previous   Overview: First steps   Next