

JS Variables and Numbers

Learning Objectives

- knowing the difference between `var`, `let` and `const`
 - understanding the different data types
 - using basic math operations
-

Variable Declarations

Variables are a `reference` or `alias` for data stored in memory. You can access this data by using this variable. You can use three different keywords to declare a variable:

- `const` - declares a constant, the value can't be changed. Default way to declare variables.
- `let` - declares a variable, the value can be changed. Only used when reassigning a new value is necessary.
- `var` - outdated, not used anymore.

Normally the keyword `const` is used to declare a variable.

```
const aNewVariable = 1234;
```

The keyword `let` is only used when you need to reassign a value, for example when you want to increase a counter.

```
let counter = 0;  
counter = counter + 1; // reassigning the value of counter
```

The `=` sign in programming doesn't quite work like the mathematical equality that you (maybe) remember from school. It means: "the value of the item on the right of the equal sign is saved in the item on the left of it". What the item on the right actually represents is calculated first and saved afterwards.

Primitive Data Types

JavaScript is a dynamically typed language, which means, that you don't have to specify what kind of value you want to store, JavaScript detects this automatically.

There are 7 primitive data types:

type	represents
<code>string</code>	a sequence of characters: "abcd"
<code>number</code>	a number: 1234

type	represents
boolean	a binary statement, can be <code>true</code> or <code>false</code>
null	represents "nothing", is typically set by developers
undefined	represents the state of "not existing". Anything not specified or not found in JavaScript defaults to the value <code>undefined</code>
BigInt	uncommon, used for integers larger than 9007199254740991
Symbol	uncommon, used for creating unique elements

Variable Naming

Expressive variable names are very important for the **readability of the code**. The Code becomes easier to understand and needs less comments. There are some key guidelines you should follow when naming a variable:

- use camel case: `socialFeedEntry` instead of `socialfeedentry`
- write out all words: `error` instead of `e`, `followerButton` instead of `flBtn`
- be very specific, longer names are better than shorter: `updatedFollowerCounter` instead of `counter`.

Math & Operators

As a programmer you sometimes have to use mathematical operations to calculate certain widths or positions of elements. Operators calculate values based on one or two expressions.

operator	effect
+	adds two numbers together.
-	subtracts two numbers
*	multiplies two numbers
/	divides two numbers
**	potentiates two numbers: <code>2 ** 4 → 16</code>
%	The remainder or modulus. Gives you what remains after a whole number division: <code>8 % 3 → 2</code> .

The remainder is a very useful operator, but might be difficult to understand at first. A real life example would be time on a clock. After noon, you don't reach 13am but you start over at 1pm. 3 hours after midnight you don't have 15pm (or 27h in the 24h format), but 3am. It is whatever hour we have mod 12:

```
5 % 12; // → 5
12 % 12; // → 0
13 % 12; // → 1
15 % 12; // → 3
27 % 12; // → 3
```

You can also use this operator to determine if a number is even or odd:

```
6 % 2; // → 0
```


This is 0 for all **even** numbers, because after dividing an even number by 2 nothing remains.


```
5 % 2; // → 1
```

This is 1 for all **odd** numbers, because after this division you have always 1 left over.

Operator Precedence

In maths, some operators have a higher precedence than others. This means that they are performed before operators with a lower precedence. For example, multiplication comes before addition.

 You can read more about [Operator precedence in the mdn](#).

 If you are uncertain, use parentheses around calculations to denote precedence manually. Prettier will remove any unnecessary parentheses from your expression automatically.

Assignment Operators

You already know the default assignment operator `=`. This operator just assigns the value on the right to the element on the left. There are more assignment operators for very common actions like increasing a variable by a fixed value.

operator	effect
<code>+=</code>	Increases the value of the variable on the left about the value on the right: <code>count += 6</code> → count is increased by 6
<code>-=</code>	Decreases the value of the variable on the left about the value on the right
<code>*=</code>	Multiplies the variable on the left with the value on the right
<code>/=</code>	Divides the variable on the left with the value on the right
<code>++</code>	Increments the value of a variable by one: <code>count++</code> → count is increased by one
<code>--</code>	Decrements the value of a variable by one: <code>count--</code> → count is decreased by one

Type Coersion

When you use an operator with a variable with an unfitting type, JavaScript will automatically convert (coerse) this variable into a fitting type. For example:

```
4 / "2"; // → 4 / 2
```

There is no `/` operator for strings, so JavaScript converts the string into a number if possible. This is also true for boolean operators which we will cover in a later session.

! There is another `+` operator in JavaScript, that links two strings together: `"a" + "b" → "ab"`. When 'adding' a number and a string, the number is converted to a string: `"a" + 6 → "a6"`. Make sure that both variables are numbers if you want to add them.

 Read more about [Type coercion in the mdn](#).

Number Systems

When working with computers, it is sometimes useful to work with a different number system than the standard 10 digit system, since a computer only understands **binary** numbers composed of only 0 and 1. You don't have to learn these systems by heart, but it is good if you heard about them.

- **decimal system**: the standard numbers, has 10 symbols "0" to "9".
- **binary system**: only has 2 symbols "0" and "1". If you want to write a bigger number than 1, you add another digit: 2 → "10" in binary.
- **hexadecimal system**: has 16 symbols "0" to "9" and "a" to "f". If you want to write a number bigger than 15 you add another digit: 12 → "c" in hexadecimal.

Resources

- [Operator Precedence in the mdn](#)
- [Type coercion in the mdn](#)