

JavaScript Variable Tutorial for Beginners

DECEMBER 10, 2019 BY ROBIN WIERUCH - [EDIT THIS POST](#)

 Follow on Twitter

18k

 Follow on Facebook



f
t
in

In every programming language, you will find variables. This also holds true for JavaScript. Essentially variables are used to carry information. Without them, it wouldn't be really possible to code any applications. At some point in time, you always have to keep or transfer information in variables. In this JavaScript tutorial, I want to dive with you into JavaScript Variables by example.

JAVASCRIPT VARIABLE

For instance, let's say we have the following JavaScript variable:

```
var name = 'Robin Wieruch';
```

In JavaScript, you can reference it in your code. For instance, you can output something in JavaScript the following way:

```
console.log('Robin Wieruch');
```

So instead of using the information explicitly, you can put the information into a variable, and use this variable instead the implicit way:

```
var name = 'Robin Wieruch';  
  
console.log(name);  
// "Robin Wieruch"
```

Not only can you carry around the information in this variable and use it somewhere in your JavaScript code, you can use it more than once as well:



```
var name = 'Robin Wieruch';  
  
console.log(name);  
// "Robin Wieruch"  
console.log(name);  
// "Robin Wieruch"
```

Now imagine you are using a variable multiple times throughout your JavaScript application at different places. If you wouldn't have a variable and use the information explicitly, you would have to change it at many places:

```
console.log('Dennis Wieruch');  
console.log('Dennis Wieruch');
```

Instead, by having a variable for this information in place, you can change it once and affect all the places where it is used:


```
var name = 'Dennis Wieruch';  
  
console.log(name);  
// "Dennis Wieruch"  
console.log(name);  
// "Dennis Wieruch"
```

```
var name = 'Robin Wieruch';  
  
console.log(name);  
// "Robin Wieruch"  
  
name = 'Dennis Wieruch';  
  
console.log(name);  
// "Dennis Wieruch"
```

In this section, you have used a **string primitive** from a set of available **JavaScript data types**. In the following, you will learn more about this and other data types in JavaScript.

Exercises:

- Head over to [CodeSandbox](#), remove the JavaScript placeholder content, and play around with

 JavaScript Variables



JAVASCRIPT VARIABLES: STRINGS

A string primitive consists of one or multiple characters. If a string primitive is **defined** as a variable, it has to be set in quotes. Otherwise, JavaScript would think it's just another variable.

```
var firstName = Robin;  
// doesn't work, because Robin is an undefined variable  
  
var lastName = 'Wieruch';  
// does work, because Wieruch is set in quotes
```

You can concatenate strings to a new string variable:

```
var firstName = 'Robin';  
  
console.log(`${firstName} Wieruch`);  
// "Robin Wieruch"
```

You can also define the other string as its own variable:

```
var name = `${firstName} ${lastName}`;  
  
console.log(name);  
// "Robin Wieruch"
```

What you have done here is called **string interpolation**. By putting your new string into back ticks instead of single quotes, any JavaScript variable can be referenced with `${}` in between to create a new JavaScript string. The back ticks notation is called **template literals** in JavaScript.

```
var firstName = 'Robin';  
var lastName = 'Wieruch';  
  
var sentence = `Hello, my name is ${firstName} ${lastName}.`;   
  
console.log(sentence);  
// "Hello, my name is Robin Wieruch."
```



In earlier JavaScript versions, template literals as a feature weren't available and you would have used **string concatenation** instead of string interpolation with the `+` operator:



```
var firstName = 'Robin';  
var lastName = 'Wieruch';  
  
var sentence = 'Hello, my name is ' + firstName + ' ' + lastName + ' '   
  
console.log(sentence);  
// "Hello, my name is Robin Wieruch."
```

JavaScript strings are only one of six **JavaScript primitives** which are a subset of **JavaScript data types**.

Exercises:

- Head over to [CodeSandbox](#), remove the JavaScript placeholder content, and play around with JavaScript Strings
- Read more about [JavaScript Template Literals](#)

DEFINITION AND DECLARATION

```
var name = 'Robin Wieruch';
```

It only takes one step to **declare** and **define** a variable. But there is a difference between both. A **variable declaration** already takes place if no value is **assigned** to the variable.

```
var name;  
  
console.log(name);  
// undefined
```

In another step, the **variable definition** can take place. Because it has been declared before, there is no other declaration needed, but just an **assignment**:

```
var name;  
  
console.log(name);  
// undefined  
  
name = 'Robin Wieruch';  
  
console.log(name);  
// "Robin Wieruch"
```

Both steps, the **JavaScript variable declaration** and **JavaScript variable definition** can take place in one line of code by declaring and defining the variable right away.

```
var name = 'Robin Wieruch';  
  
console.log(name);  
// "Robin Wieruch"
```

A JavaScript variable can be **re-assigned** too, by just overwriting the defined value without another declaration:

```
var name = 'Robin Wieruch';  
  
console.log(name);  
// "Robin Wieruch"  
  
name = 'Dennis Wieruch';
```

It's also called **mutation of the variable** or **mutating the variable** – which are just more technical terms than **changing a variable**. Later this knowledge is useful, because there is a difference between **mutable and immutable data structures**. After all, mutating a variable just means you are re-assigning the variable's value.

```
var name = 'Dennis Wieruch';

console.log(name);
// "Dennis Wieruch"

var myBrother = name;

console.log(myBrother);
// "Dennis Wieruch"
console.log(name);
// "Dennis Wieruch"
```



The last code snippet shows that it's also possible to declare/define a new variable based on another declared variable.



Exercises:



Head over to [CodeSandbox](#), remove the JavaScript placeholder content, and play around with JavaScript Definitions and Declarations

- Check what happens if you declare a variable twice

JAVASCRIPT DATA TYPES AND DATA STRUCTURES

In contrast to many other programming languages, JavaScript is a **loosely typed** language – which only means that variables are not assigned to a particular **data type**. As you have learned before, a variable can be re-assigned, which means it can also change the data type.

```
// string
var age = '30';

// number
age = 30;

// boolean
age = true;
```

number primitive and re-assigned again to a **boolean primitive**. A JavaScript variable can be assigned to any data type. Seven of eight data types in JavaScript are primitives:

- String
- Number
- Boolean
- Undefined
- Null
- BigInt
- Symbol

The eighth data type is an **JavaScript object**. Before exploring the JavaScript object, let's go through the most important JavaScript data types step by step with the simplest explanations for them:

- **String:** Consists of one or multiple characters defined in single/double quotes or backticks.
- **Number:** Consists of one or multiple numbers defined without quotes. Also commonly called **integer**.
- **Boolean:** Can be either `true` or `false` if it is defined. It's used to make decisions in an application.
- **Undefined:** If a variable is declared but not defined, it is undefined.
- **Null:** Can only be null. It is used if a variable has intentionally no value.

in

```
var anything = '30'; // a string
anything = 30; // a number (also called integer)
anything = true; // a boolean
anything = false; // a boolean
anything = undefined; // not defined
anything = null; // no value
```

Very rarely you will use `BigInt` or `Symbol` in JavaScript, that's why I keep them out for this introduction to JavaScript variables for the sake of keeping you in flow for this learning experience. You can read more about those in the exercises below.

Then there are JavaScript objects. For the sake of keeping it beginner friendly again, I will introduce the object as a more complex JavaScript data structure that allows us to hold more/other information than just a string or number. The most commonly used objects in JavaScripts are:

- Object
- Array
- Function

information. The entries in the list can have any data type:

```
var names = ['Robin Wieruch', 'Dennis Wieruch'];  
var anything = ['Robin Wieruch', 30, true];
```

However, usually all entries in an array have the same data type; which doesn't mean that it's not possible to have an array with different data types (e.g. integers, booleans, strings). In contrast to objects, arrays have a specific order:

```
var names = ['Robin Wieruch', 'Dennis Wieruch'];  
console.log(names);  
// ["Robin Wieruch", "Dennis Wieruch"]
```

f You can access each entry in an array by its position (index). The index starts by 0 though, which is commonly seen in other programming languages as well:



```
var names = ['Robin Wieruch', 'Dennis Wieruch'];
```



```
var name = names[1];  
console.log(name);  
// "Dennis Wieruch"
```

As mentioned before, objects have unordered information defined within the object as key/value pairs, whereas any data type can be present:

```
var person = {  
  firstName: 'Robin',  
  lastName: 'Wieruch',  
  age: 30,  
  isMarried: true,  
};
```

Since an object has no order, you can access the values by its keys:

```
var person = {  
  firstName: 'Robin',  
  lastName: 'Wieruch',  
};
```



```
var name = `${person.firstName} ${person.lastName}`;  
  
console.log(name);  
// "Robin Wieruch"
```

Since an array (also called list) can hold any types as entries (also called items), it can hold a list of objects too:

```
var me = {  
  firstName: 'Robin',  
  lastName: 'Wieruch',  
};  
  
var myBrother = {  
  firstName: 'Dennis',  
  lastName: 'Wieruch',  
};  
  
var persons = [me, myBrother];  
  
console.log(persons[0].firstName);  
// "Robin"
```

Since objects can have any types as values, it can hold arrays too:

```
var me = {  
  firstName: 'Robin',  
  lastName: 'Wieruch',  
  parents: ['Heike Wieruch', 'Thomas Wieruch'],  
};  
  
console.log(me.parents);  
// ["Heike Wieruch", "Thomas Wieruch"]
```

This can go on and on with objects within objects, arrays within arrays – so called multi-dimensional arrays –, arrays within objects, and objects within arrays. All permutations of data types are possible in this more complex data structures.

Last but not, there are functions. Same as objects and arrays, I will not go into too much detail here. Rather I want to give you an introduction to elaborate more on these JavaScript data structures later. Functions are used as mini programs to run in your JavaScript application.

```
function getName(person) {
```

Fundamentally a function has the function statement (1), a name (e.g. getName) (2), input parameters (e.g. person) (3), some business related internal statements (4), and a return statement (5) – for giving something back from this mini program. The returned value can be stored in a variable again when calling the function (6).

```
// (1)(2)(3)
function getName(person) {
  // (4)
  var name = `${person.firstName} ${person.lastName}`;

  // (5)
  return name;
}

var me = {
  firstName: 'Robin',
  lastName: 'Wieruch',
};

// (6)
var nameOfPerson = getName(me);

console.log(nameOfPerson);
// "Robin Wieruch"
```

Within a function, the mini program (4) can be as long as you need to have it to fulfil a business related task for your program. If there is only one computed variable, like in our case for (4), we can use an **immediate return** as well, without assigning a new variable in between.

```
function getName(person) {
  return `${person.firstName} ${person.lastName}`;
}
```

Both, input parameters (3) and return statement (5) are optional:

```
function addTwoPlusThree() {
  console.log(2 + 3);
}

addTwoPlusThree();
// 5
```

However, it's a good practice to have input and output for a function:

```
function sum(valueOne, valueTwo) {  
  return valueOne + valueTwo;  
}  
  
var result = sum(2, 3);  
console.log(result);  
// 5  
  
result = sum(result, 5);  
console.log(result);  
// 10
```

It keeps a function versatile and more input/output focused which makes your code more robust against any bugs. Anyway, you are here to learn about JavaScript variables and data

ftypes/structures in JavaScript. In JavaScript, **functions are first-class citizens** – which only means that you can make use of them in a more powerful way than in many other programming languages. For instance, one property of being a first-class citizen function is the ability to assign them to a variable:

in

```
var whatsYourName = function(name) {  
  return `My name is ${name}.`;  
}  
  
var result = whatsYourName('Robin Wieruch');  
console.log(result);  
// "My name is Robin Wieruch."
```

Since the function has no name, it's called an **anonymous function**.

```
function whatsYourName(name) {  
  return `My name is ${name}.`;  
}  
  
var result = whatsYourName('Robin Wieruch');  
console.log(result);  
// "My name is Robin Wieruch."
```

An anonymous function isn't much different in comparison to the previous code snippet, but later we will encounter how this is a powerful way of using functions more dynamically in JavaScript than in many other programming languages.

null, more complex data structures are represented by objects, arrays and functions.

Exercises:

- Read more about [JavaScript Data Types and Data Structures](#)
 - Read more about String, Number, and Boolean
 - Clarify for yourself the difference between Null and Undefined
 - Find out how to concat two strings without the previously learned string interpolation, but with the built-in string concat () method.
- Head over to [CodeSandbox](#), remove the JavaScript placeholder content, and play around with various JavaScript Data Structures



Show Comments



KEEP READING ABOUT [JAVASCRIPT](#) >

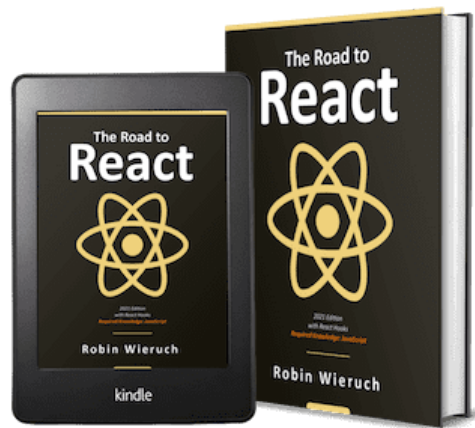


CONST VS LET, AND VAR IN JAVASCRIPT

There are three different ways to declare a variable in JavaScript: const, let and var. Historically, var has been the only way of declaring a JavaScript variable : An addition to JavaScript – to be...

JAVASCRIPT NAMING CONVENTIONS

A JavaScript naming conventions introduction by example – which gives you the common sense when it comes to naming variables, functions, classes or components in JavaScript. No one is enforcing these...



THE ROAD TO REACT

Learn React by building real world applications. No setup configuration. No tooling. Plain React in 200+ pages of learning material. Learn React like

50.000+ readers.

GET THE BOOK >

Get it on Amazon.



TAKE PART

NEVER MISS AN ARTICLE ABOUT WEB DEVELOPMENT AND JAVASCRIPT.

- ✓ Join 50.000+ Developers
- ✓ Learn Web Development

✓ Personal Development as a Software Engineer

SUBSCRIBE

[View our Privacy Policy.](#)



PORTFOLIO

[Online Courses](#)

[Open Source](#)

[Tutorials](#)

ABOUT

[About me](#)

[What I use](#)

[How to work with me](#)

[How to support me](#)

© Robin Wieruch



[Contact Me](#) [Privacy & Terms](#)