**JavaScript Tutorial for Beginners**

***Learn by doing with this interactive tutorial.***

[Tutorials](https://edabit.com/tutorials)

JavaScript



This tutorial is designed for **beginners**. It will take you from knowing nothing about JavaScript to solving bite-sized [JavaScript exercises](https://edabit.com/challenges/javascript) when you're done.

You don't need to know anything about JavaScript to start. You'll be able to tinker with code and get **immediate feedback** right here in your browser.

Grab a cup of coffee, sit back, and relax!

**Table of Contents**

[Strings](https://edabit.com/tutorial/javascript#strings)[Variables](https://edabit.com/tutorial/javascript#variables)[Numbers](https://edabit.com/tutorial/javascript#numbers)[Operators](https://edabit.com/tutorial/javascript#operators)[Functions](https://edabit.com/tutorial/javascript#functions)[Arrays](https://edabit.com/tutorial/javascript" \l "arrays)

[Indexes](https://edabit.com/tutorial/javascript#indexes)[Mutability](https://edabit.com/tutorial/javascript#mutability)[.length](https://edabit.com/tutorial/javascript#length)

[Objects](https://edabit.com/tutorial/javascript" \l "objects)

[Key-Value Pairs](https://edabit.com/tutorial/javascript#key-value-pairs)[Dot Notation](https://edabit.com/tutorial/javascript#dot-notation)

[Loops](https://edabit.com/tutorial/javascript" \l "loops)

[.map()](https://edabit.com/tutorial/javascript#map)[.filter()](https://edabit.com/tutorial/javascript#filter)

A collection of characters (letters, numbers, symbols) is known as a **string**. Strings must begin and end with **quotation marks**. Either single ' or double " will work, so long as you use the same at the beginning and end.

"edabit.com"

'edabit.com'

The quotes are there to indicate the enclosed text is a **value**, not code.

The word "variable" means "can change"; they're used to store values that can change.

A variable is like a **box** with a **sticky note** stuck to it. Referencing the name on the sticky note will allow you to access whatever is inside the box (variable). Like the x and y variables used in mathematics, they're a simple name to represent the data we want to refer to.

Let's use the keyword var to **declare** (i.e. create) a variable named animal:

var animal = "Fox"

var is shorthand for variable and = means *store the value on the right-hand side in the variable on the left-hand side*. You can name a variable anything you want, but it **can't contain spaces**.

After the equals = sign, enter the string "edabit.com". Remember that **strings** must be wrapped in quotes " ".

var website = "edabit.com"

In modern JavaScript, var is rarely used to declare variables. Instead, we use const (shorthand for **constant**) and let. For this beginner tutorial, we'll be using const from now on.

Let's create a variable from scratch. Declare the variable food and assign it the value "pizza". Use const instead of var.

// enter code below this line

const food = "pizza"

Like strings, numbers are **values** but they're **not wrapped in quotes**. They can be written with or without decimals and can be positive or negative.

const temperature = -7.5

If a number does not contain a decimal, it's referred to as an **integer**.

Declare a **variable** named day and assign it the value 19.

// enter code below this line

const day = 19

Operators are the symbols between values that allow different operations like addition +, subtraction -, multiplication \*, division /, etc. JavaScript has dozens of operators, but we'll just focus on **arithmetic operators**, since they're the ones you’ll use most as a beginner.

Divide 100 by 2.

const division = 100 / 2

Of course, there are other **operators** like modulus %, exponentiation \*\*, increment ++, decrement --, and many more.

Use the + operator to **concatenate** (combine) two **strings** together.

* Declare the variable name.
* Concatenate firstName with lastName (notice the N is capitalized).

Your code should look exactly like what you've done in the above example, only you'll be adding (using the + operator) two **strings** together.

const firstName = "Luke "

const lastName = "Skywalker"

​

// enter code below this line

const name = firstName + lastName

Functions are **blocks of code** that can be named and reused. They are given data, do something specific with the data, and return a result.

This is what a basic function looks like:

function addTwoNumbers(num1, num2) {

return num1 + num2

}

Can you guess what it does? Let's look at each part:

* addTwoNumbers is the name of the function.
* num1 and num2 are **parameters** (i.e. variables containing input data).
* return is the **keyword** that exits the function and returns a value (output).

**Using the** return **keyword is very important. When you encounter a function, you must** return **your answer, otherwise the code won't work.**

The below function takes two **parameters**, each containing a number.

* **Multiply** the first parameter num1 by the second parameter num2.
* Remember to use the return keyword, as shown in the above example.

function multiplyTwoNumbers(num1, num2) {

return num1 \* num2

}

Although functions usually take **parameters**, they don't necessarily have to.

function hello() {

return "Hello World!"

}

When the above function hello() is called, it will output "Hello World!".

Arrays are **lists** of items. They look like this:

const fruit = ["apple", "banana", "orange", "mango", "tomato"]

Each fruit in the above array is called an **element**. Although every element in the above example is a string, array elements can be any value:

const stuff = [true, 1976, null, "hey"]

Each **element** in an array has an **index** that starts at 0. Using the first fruit example, "apple" is at index 0, "banana" is at index 1, "orange" is at 2, and so on until the end of the array.

To access a specific **element** within an array, we do this:

fruit[2]

That's the **index** for **"orange"**.

Return the element "Donatello".

* Remember the **first element** in an array is 0.
* Always return your answer!

const turtles = ["Raphael", "Michelangelo", "Leonardo", "Donatello"]

​

function turtlePower(turtles) {

return turtles [3]

}

Elements in arrays are **mutable**, which means they can be **changed**. For example, let's say we've got an array of numbers:

const numbers = [14, 56, 78]

To **change** the value of 14 (at index 0), we do this:

numbers[0] = 35

Given an array of numbers, set the value of the data stored at index 1 to 88.

const numbers = [1, 4, 6, 8, 0]

​

// enter code below this line

return numbers[1] = 88

In most cases, you won't know the exact **length** of an array. In other words, you won't know how many **elements** it contains. That's where the .length **property** comes in.

Let's say we have an array of movie names:

const movies = ["The Matrix", "Se7en", "The Wizard of Oz", "Marry Poppins"]

To get the exact length of the movies array, we would use the .length property.

movies.length

That would give you 4, as there are four **elements** in the movies array.

Given an array arr of unknown length, return its length.

function getLength(arr) {

return arr.length

}

Arrays are good for lists, but for other tasks they can be hard to work with. Consider an array of names:

const firstNames = ["Tyrion", "Jon", "Jorah", "Arya", "Joffrey"]

const lastNames = ["Lannister", "Snow", "Mormont", "Stark", "Baratheon"]

What if you have a last name in mind (let's say it's Snow) and want to look up his first name. With arrays, it takes a lot of work because "Jon" is in one array, and "Snow" (his last name) is in a totally different array.

This can get very messy because if we add a new person to the firstName array, we have to also update the lastName array. If we want to keep track of more than a person's first and last names, things get complicated.

Objects are an easier way to store and maintain information, like this:

const person1 = {

firstName: "Joffery",

lastName: "Baratheon",

email: "joff@widowswail.com"

}

​

const person2 = {

firstName: "Jon",

lastName: "Snow",

email: "brooding@thewall.com"

}

​

const person3 = {

firstName: "Tyrion",

lastName: "Lannister",

email: "tyrion@pourmeanother.com"

}

Now we have a **variable** for each person that can be used to get their **values** in a more maintainable and readable way.

Objects are like keys on a keyring that open a specific door and behind each door is a room that can store many things. If each key is labeled, you can quickly open doors and access the stuff inside.

const person2 = {

firstName: "Jon",

lastName: "Snow",

email: "brooding@thewall.com"

}

The things on the left of the : are called **keys** and the things on the right are **values**. We refer to this as **key-value pairs**.

Input the following key-value pairs for person4.

* Give firstName the value "Daenerys".
* Give lastName the value "Targaryen".
* Give email the value "dragonlady@gmail.com".

const person4 = {

firstName: "Daenerys",

lastName: "Targaryen",

email: "dragonlady@gmail.com"

}

If you want to access the lastName of person3, you would return person3.lastName. This is called **dot notation** because it uses a . to specify the **key** you want to access.

Use **dot notation** to access the email of person.

function getEmail(person) {

return person.email

}

Loops offer a quick and easy way to do something **repeatedly**. This section will cover the most **common** ways of doing loops in modern JavaScript.

The .map() method applies a function to **every element** in an array. A new array is then returned. In other words, .map() takes an array, **does something** to every element in the array and returns a new array.

Let's say we have an array of numbers:

const numbers = [1, 4, 9, 16]

To **multiply** every element in the numbers array by 2, we do this:

numbers.map(x => x \* 2)

Which would return:

[2, 8, 18, 32]

**Subtract** 2 from every element in the numbers array.

const numbers = [1, 4, 9, 16]

​

function subtractTwo(numbers) {

return numbers.map (x => x - 2)

}

The .filter() method returns a new array containing all elements that **pass a test** from a function you provide. In other words, .filter() takes an array, **tests** every element and returns a new array containing **only** elements that pass a test you provide.

Let's say we have an array of strings:

const words = ["thaw", "achievement", "gain", "outlet", "difference"]

To return **only** elements in the words array **less than 5** characters in length, we do this:

words.filter(word => word.length < 5)

Which returns the following output:

["thaw", "gain"]

Return only numbers in the years array **greater than** 1950.

const years = [1763, 1972, 1925, 1916, 1984, 1124, 1950, 2020]

​

function getYears(years) {

return years.filter(year => year > 1950)

}