06-08-2021

Palle Westermann

TTec

Client Side Programmering

JavaScript

# Indholdsfortegnelse

[Indholdsfortegnelse 1](#_Toc79390009)

[Getting Started 2](#_Toc79390010)

[What is JavaScript? 2](#_Toc79390011)

[What can you do with it? 2](#_Toc79390012)

[Where does JavaScript run? 2](#_Toc79390013)

[JavaScript vs. ECMAScript? 2](#_Toc79390014)

[Let’s see JS in Action 3](#_Toc79390015)

[Setting Up the Development Environment 6](#_Toc79390016)

[JavaScript in browsers 11](#_Toc79390017)

[Separation of Concerns 13](#_Toc79390018)

[JavaScript in Node 15](#_Toc79390019)

[Basics 16](#_Toc79390020)

[Variables 16](#_Toc79390021)

[There are rules for naming variables 17](#_Toc79390022)

[Different ways of declaring variables 17](#_Toc79390023)

[Constants 18](#_Toc79390024)

[Primitive Types 19](#_Toc79390025)

[JavaScript is a Dynamic Language (using Dynamic Typing) 20](#_Toc79390026)

[Objects 23](#_Toc79390027)

[Arrays 27](#_Toc79390028)

[Functions 30](#_Toc79390029)

[Types of functions 32](#_Toc79390030)

[Operators 33](#_Toc79390031)

[JavaScript Operators 33](#_Toc79390032)

[Control Flow 34](#_Toc79390033)

[Objects 35](#_Toc79390034)

[Arrays 36](#_Toc79390035)

[Functions 37](#_Toc79390036)

# Getting Started

## What is JavaScript?

It is one of the most popular programming languages in the world today. With it you can become a Front-end developer, a Back-end developer and a Full-stack developer (that is both Front-end and Back-end developer).

## What can you do with it?

Not so many years ago it was only used on browsers to build interactive web pages and it was not recognized as a real programming language. Those days have changed. Today you can build:

* Web/Mobile apps
* Real-time Networking Apps
* Command-line tools
* Games

## Where does JavaScript run?

Originally JS was used only in browsers. So, every browser has a so-called JavaScript Engine that can execute JS code.

The JS Engine in Firefox is called SpiderMonkey, and in Chrome it is called V8.

In 2009 Node.js was invented by an engineer named Ryan Dahl. Fun fact, as of this writing Node.js is 12 years old, JavaScript is 26 years old and the Internet is 32 years old.

Ryan Dahl took the open-source Google JS Engine, V8, and put it inside a C++ program. That program is called Node. Hence, Node includes Googles JS Engine, V8.

With Node.js we can run JS code outside a browser. So, we can pass our JS program to Node for execution.

This means we can also build the Back-end for our Web Applications.

Browsers and Node.js provides a so-called Run-time Environment for our JS Code.

## JavaScript vs. ECMAScript?

ECMAScript is just a specification (kind of just a class/interface. Risky comparison :)...).

JavaScript is a programming language that confirms to this specification. (kind of instantiating the class, implementing the interface... :)

The organization, ECMA, is responsible for defining standards. They defined version 1 in 1997.

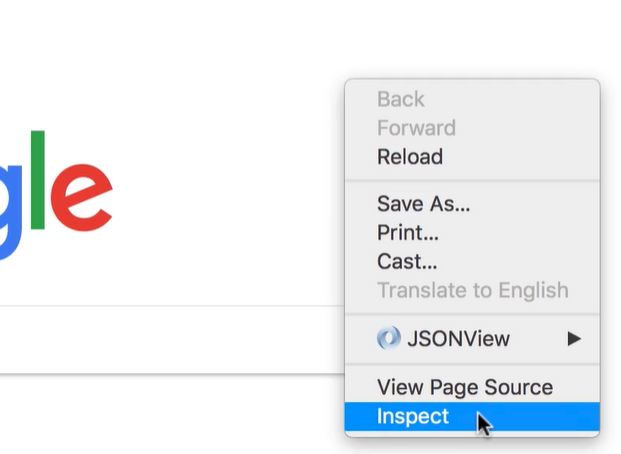
In 2015 they came up with ES2015, also called ES6.

## Let’s see JS in Action

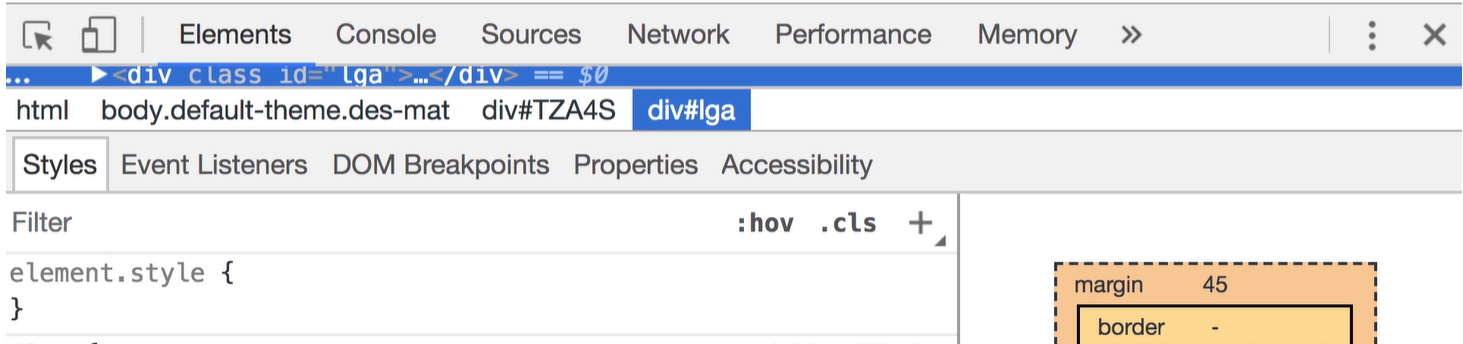
Every browser has a JS Engine and we can easily write JS code here without any additionally tools. Of course, for a real app, we will use other tools. This is just for demo purposes.

So, open up Chrome.

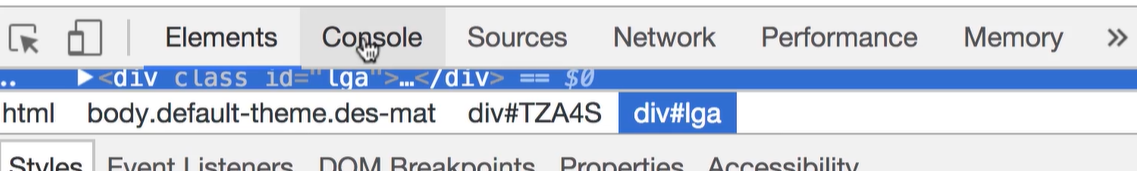
right-click on an empty area and go to “inspect”/undersøg:



This will open Chrome Developer Tools (the placement of your Developer Tools may be to the side, fx):

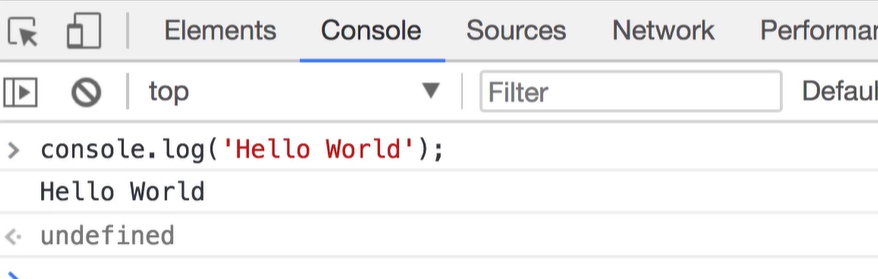


Select the Console tap:

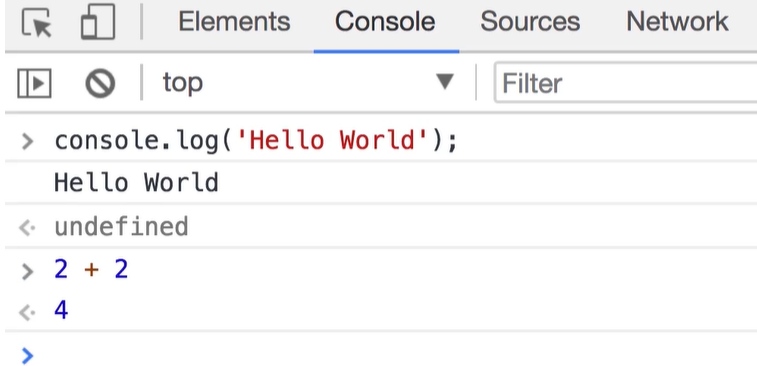


This gets you a place where you can write any JS code:

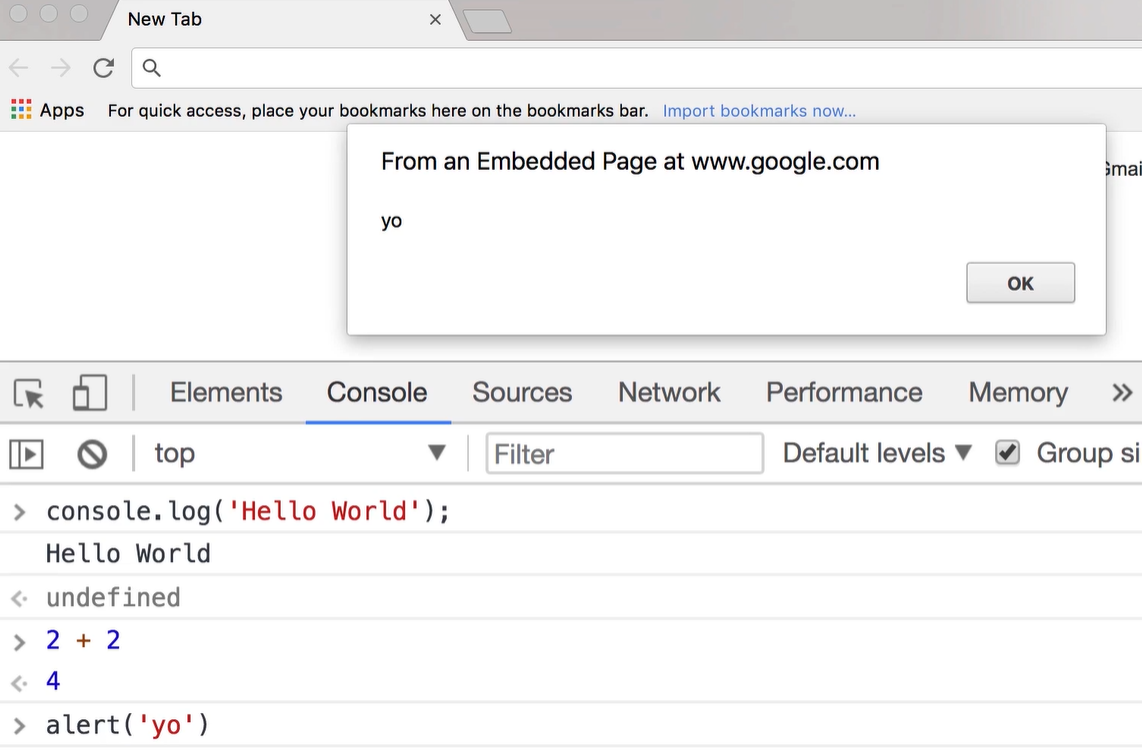
Type the *console.log* command and press return. You get the answer just below the command:



Try writing a mathematical expression:



Or you can call the *alert* function:



## Setting Up the Development Environment

You need a Code Editor, and you can choose between many of them;

Visual Studio Code, Sublime Text, Atom etc. I will be using Visual Studio Code, you do not have to!

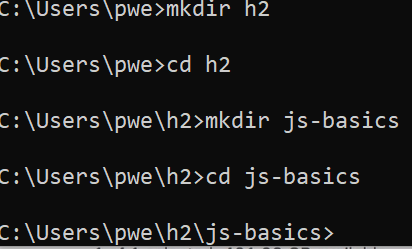
But, you can download Visual Studio Code from <https://code.visualstudio.com>

VS Code is a powerful cross-platform editor.

However, you do need to install Node.js: go to <https://nodejs.org> and install the latest stable version, the “LTS”.

We need node.js to install 3rd party libraries. We do not necessarily need it to execute code in this tutorial. It is the NPM (Node Package Manager) that we are after from the Node.js installation.

Now, create a new folder, you need it for the code that you will write in this course. I went into the command prompt and created a folder for H2, within that folder I created a new folder, js-basics, like so:

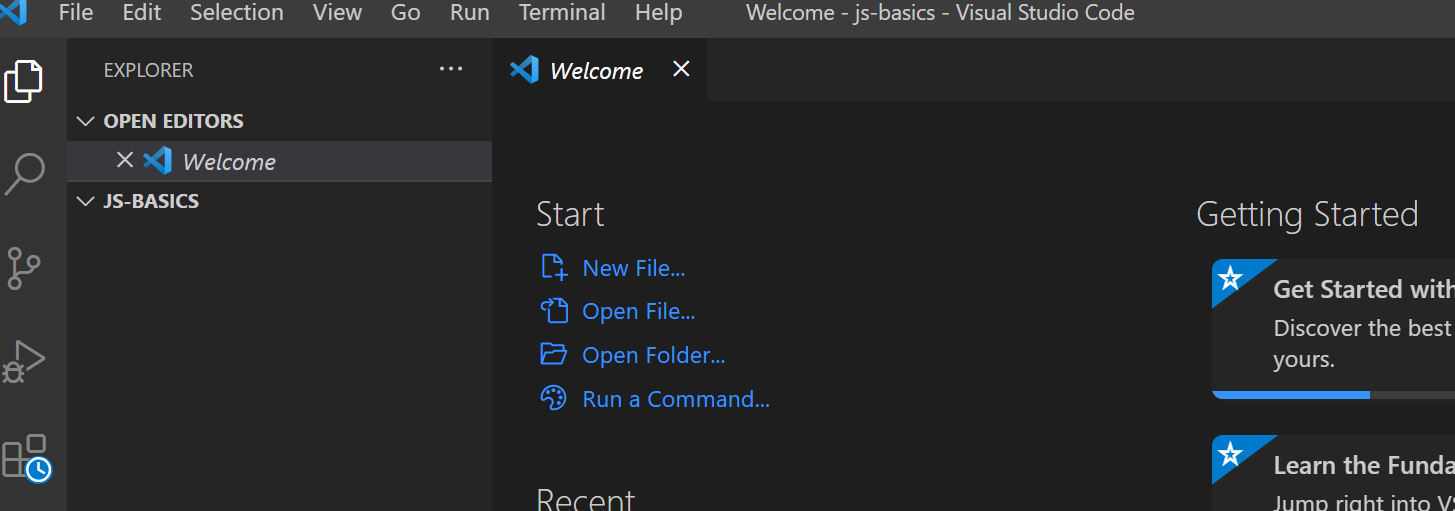


You can create your folder where ever you want!

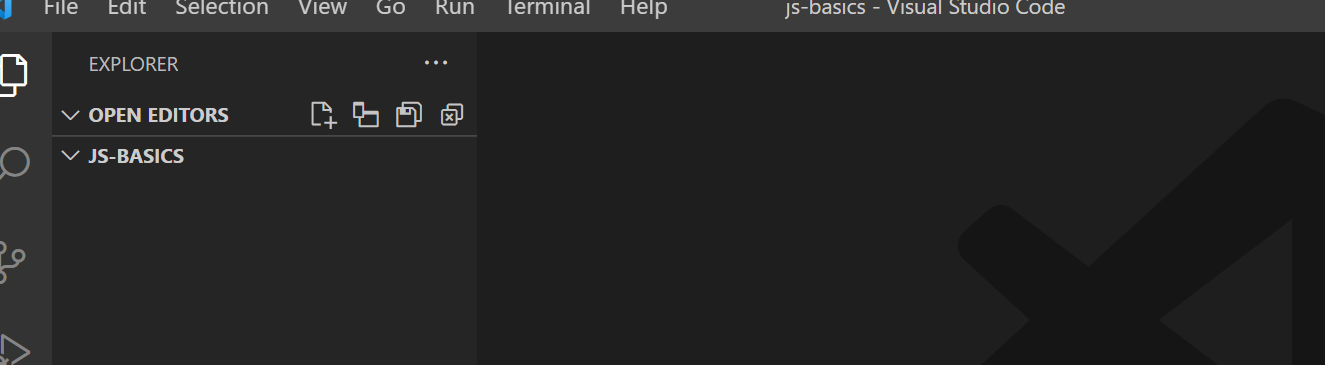
Now, you can open Visual Studio Code from within the Command Prompt by typing **code .**

Like so:

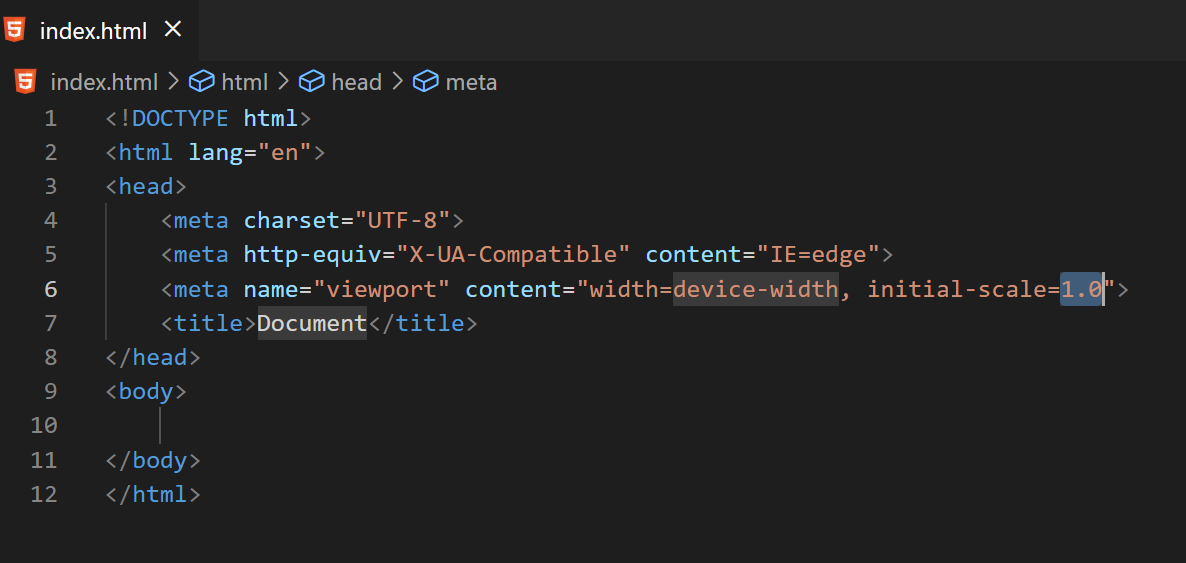




Click away the *Welcome* tag.

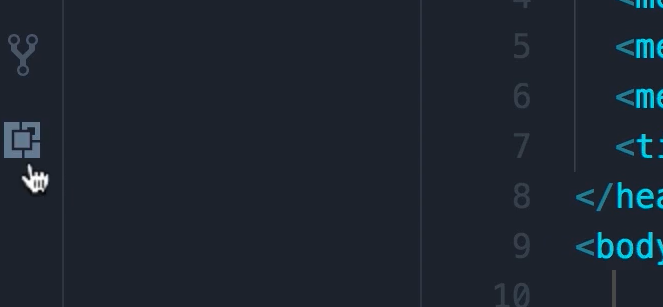


Add a new file, **index.html**. In that file type the exclamation sign “**!**” and press the Tab key -> to get the HTML automatically inserted. Like so:

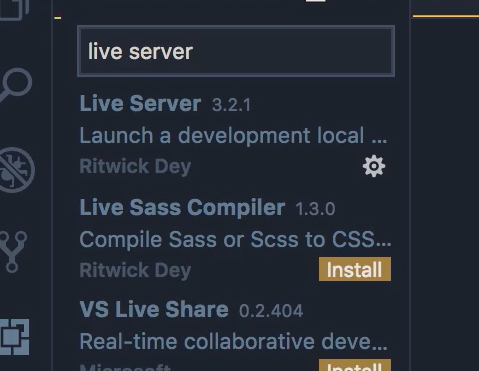


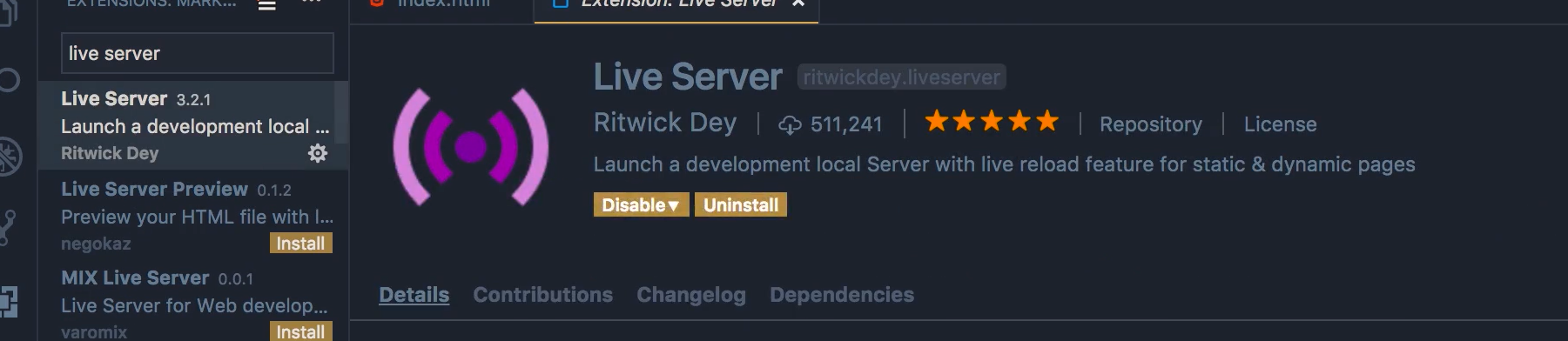
We are going to use this index.html page to host our JS code. So, if you haven’t done so, do save the changes.

Now, open the Extensions Tap:



In the top box that appears, search for **Live Server**:

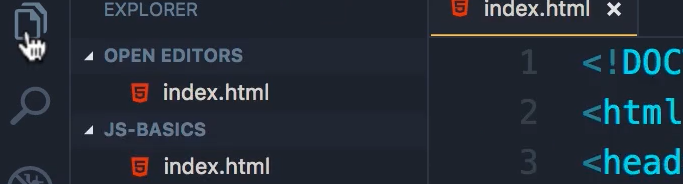




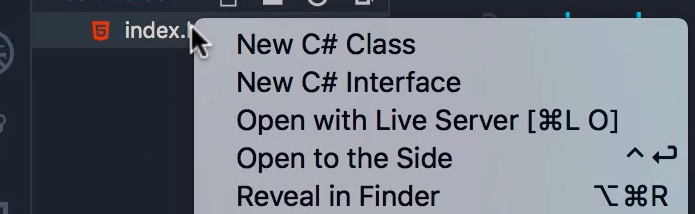
**Live Server** is a very light weight server that we will use to serve our web application. Go ahead and install it!

Next, you have to restart VS Code.

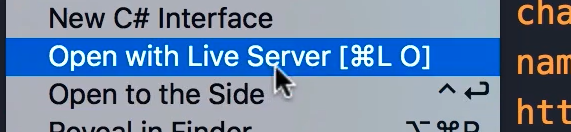
When re-started – go to the Explorer Tap:



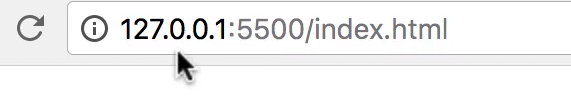
Right-click the index.html file:



and select:



This will open Chrome (or your default browser) and point it to this address:

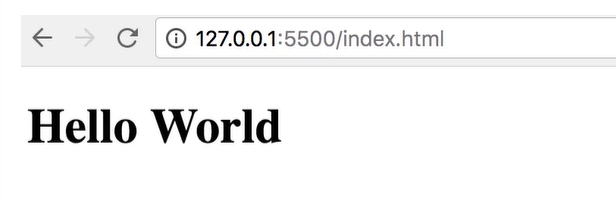


That is where our web application is served from.

Currently we have an empty page. To test go back to VS Code and add a little html code, like so:



Save the changes.



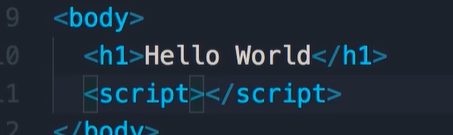
Tadaaaaaaa :)

## JavaScript in browsers

In order to write JS code we need a *Script Element* inside our HTML code (index.html) file.

We can place that *script tag* in either the HEAD or the BODY section of our HTML file.

*Best practice* is to place it at the end of the BODY section:



Below the *h1* section you can type “scr” and press the Tab key to get the *script tag* auto completed.

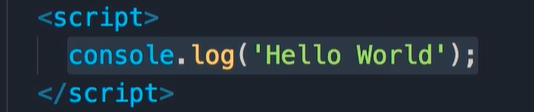
Why is best practice to put the script tag at the end of the body tag?

1. The browser *parses* the HTML file from “top to bottom”. So, if you put the script tag in the Head section, you might have a lot of JS files there, the browser might get busy running all that code and hence slowing down the rendering of the page. -> BAD USER EXPERIENCE...
2. Almost always our code in JS will refer to the HTML tags, so having JS code here will ensure that all HTML elements will be rendered by the browser, hence making our JS code able to refer to those elements.

Exceptions will be 3rd party code that sometimes has to be placed in the *head section*.

Let’s code a little.

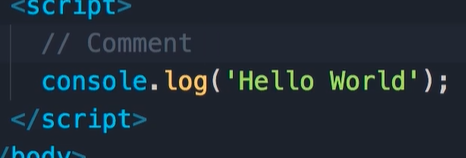
Within the *Script Section* insert this code:



In this *Script section* we have a so-called *statement*. A *statement* is a piece of code to be executed. In this particular case we want to write a message on the console.log (inside the Developer Tools of our browser).

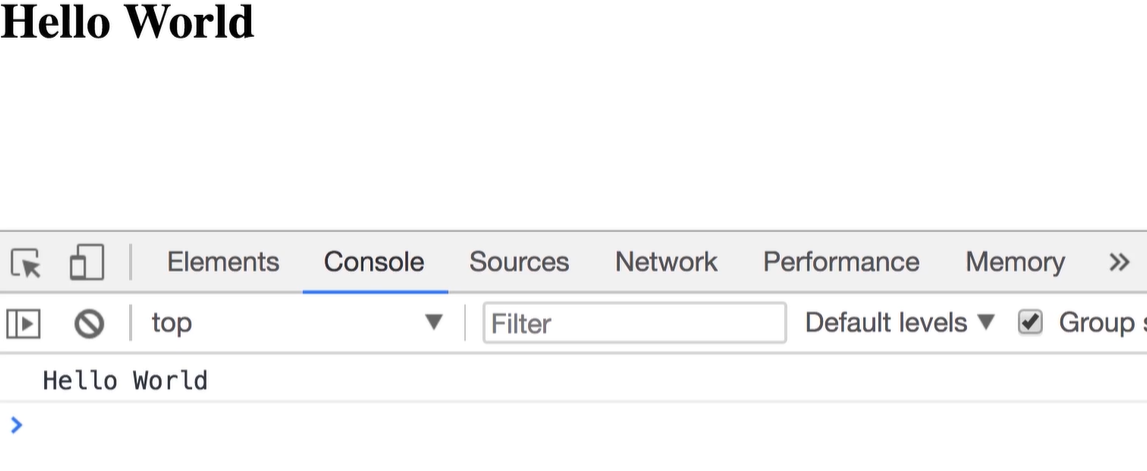
Though not necessary – it is *best practice* to end statements with a semi colon “**;**”.

To include comments use “//”:



Comments are ignored by the JS Engine (= not executed).

Save the changes -> Open Live Server -> and go to the Console tab of the Developer Tools. Here the result of running the *console.log()* statement is shown:



## Separation of Concerns

*Best practice* with JS in HTML files is to separate JS files out into their own files – and then link to them from within the HTML page.

JS is all about *behavior*!

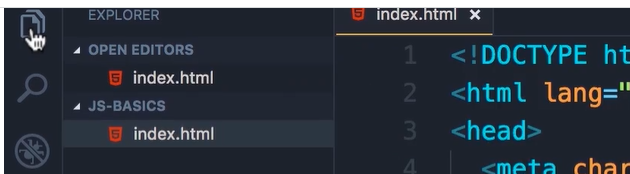
HTML is all about *content!*

CSS is all about *styling*! (and some behavior).

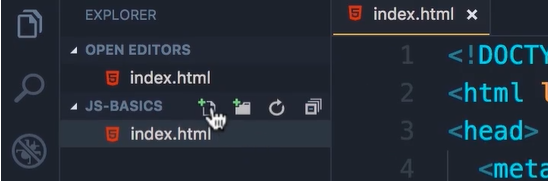
Having all these different types of code in their own files ensures *separation of concerns*!

This reduces bugs – and enables reuse.

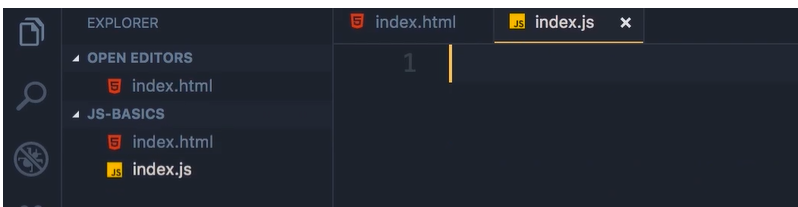
So, open the Explorer window:



Add a New File:

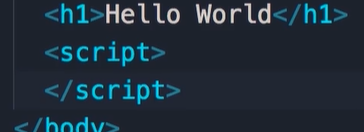


Name it **index.js**:



Now, open **index.html**

Remove (Crtl+x) the content of the *script tag* (we want to move it to the new index.js file):

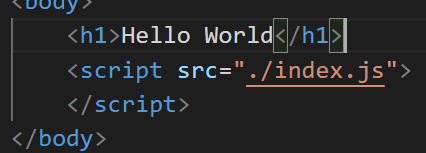


Open **index.js** – and insert the code:



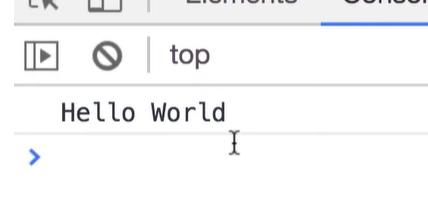
Save all changes.

Open **index.html** and add a reference to our **index.js** file:



The “**./**” part of the *src* address is a reference to current folder – and not a parent or sub folder.

Save the changes – make sure Live Server is up and running – and check that all still works:



## JavaScript in Node

In the last section we executed the code in the **index.js** from within a browser.

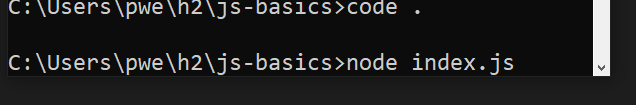
Let’s run the same code in Node.js.

Open a Command Prompt:



Your location and command prompt might look different.

Run code: **node index.js** and press Enter.





Congrats on using Node.js :)

But, it ends here – for this tutorial :)

# Basics

## Variables

In programming we use a variable to store data temporarily in the memory of a computer.

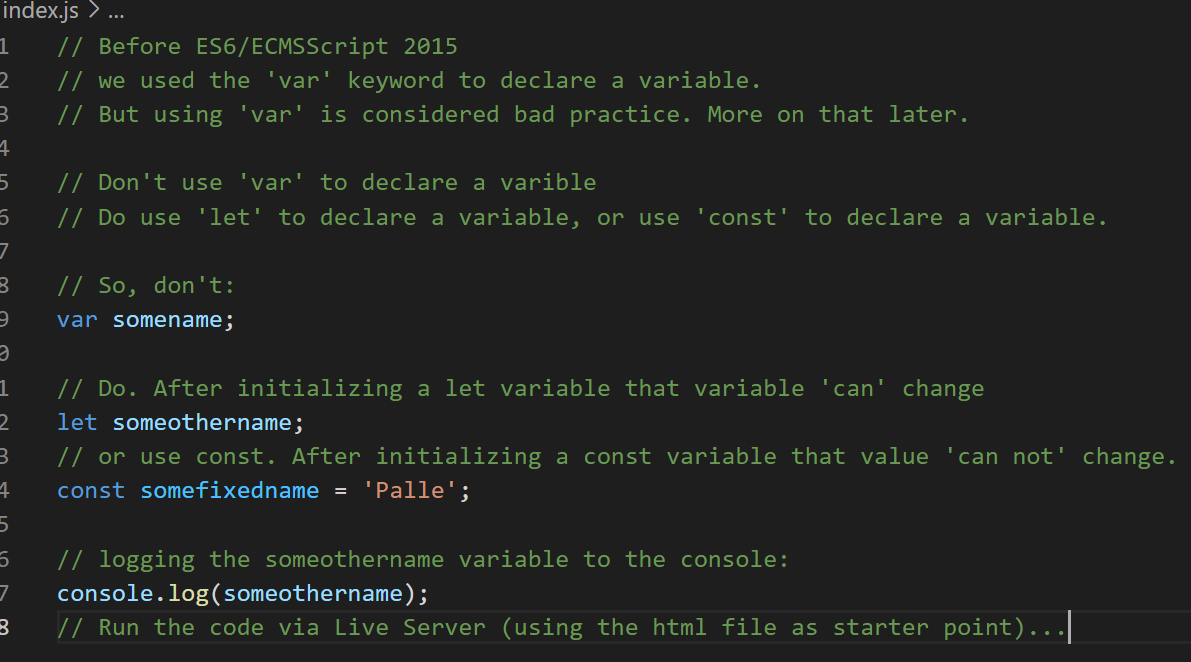
So, we store our data somewhere in memory. That somewhere is a location in memory that we give a name -> *the variable name*.

With *the variable name* we can read or write the data at any point during program execution.

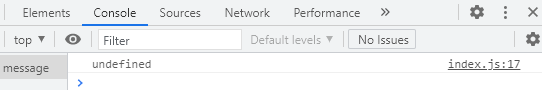
Think of a variable as a drawer (skuffe) with 1 compartment (rum). You name that drawer. So, you can tell someone; “hey, go to my drawer named **underwear** – and fetch my data/one pair of underwear... bad example, up front it looks like I only have one pair – scchhh, don’t tell :)

Now, we continue in code.

Open VS Code and the **index.js** file. Erase/delete the old content. You now should have a clean slate :). Read the information in the code below – and insert the 4 lines of actual code.



Running this will result in:

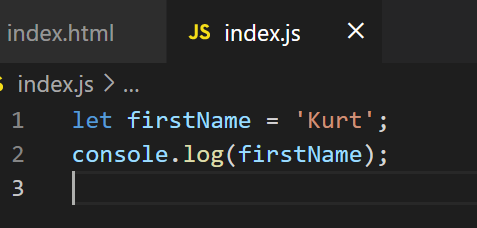


The value in the console will show **‘undefined’.**

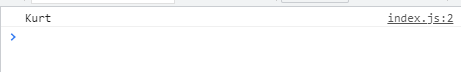
By default, declaring a variable in JS and not initializing it, will result in a value that is ‘**undefined’**.

Now, erase/delete the content again.

Insert this code which will initialize a variable named **firstName**. Initializing a variable is optional:



This time the console.log will show ‘Kurt’ as the value:



Note, I’ve been using single quotes to declare strings – but in JS it is also ok to use double quotes for strings.

## There are rules for naming variables

* Cannot be reserved keywords (if, else, let etc.)
* Should be meaningful
* Cannot start with a number (like 1name... it is also meaningless)
* Cannot contain a space or hyphen (-) (use **camel Notation/camel case, i.e. firstName for two-word variable names**).
* variable names are *case-sensitive*: **firstName** is not the same as **FirstName**.

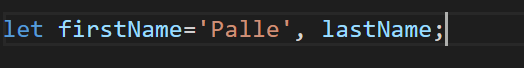
## Different ways of declaring variables

Not initialized:



* both are ‘**undefined’.**

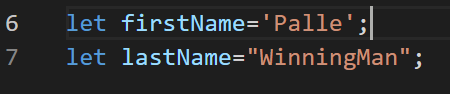
One Initialized:



Both initialized:



**But,** *best practice* is to declare both on their own line (with or without initial values):



Do remember to test these variables by running the **console.log(**variable name**)** command!

## Constants

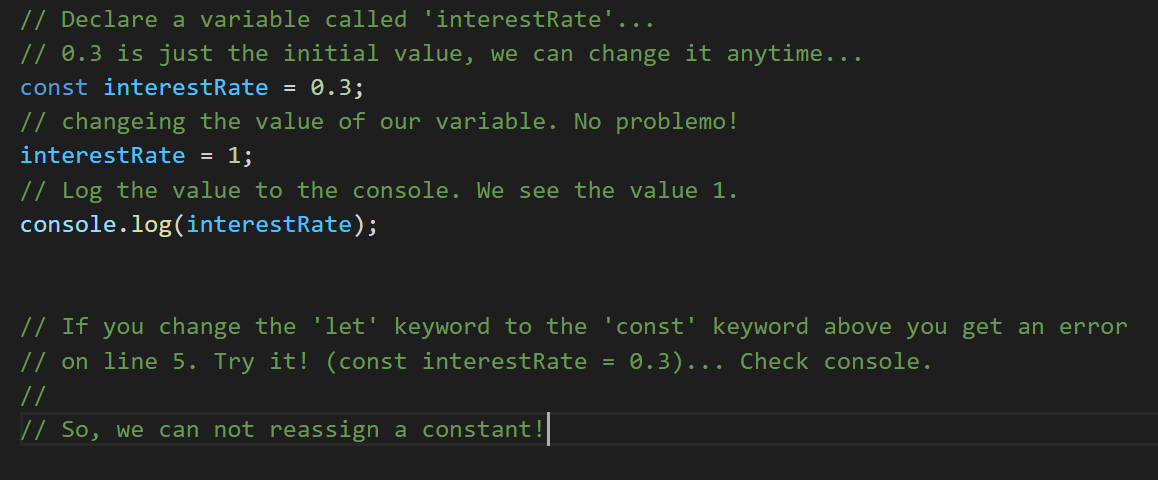
Use it when you don’t want the initial value to accidentally change during run time.

We can’t reassign a constant. (See code below).

Again, clear/delete all content of the **index.js** file.

We continue in VS Code. Remember to code along:

An error in below code, **const** should initially be **let**.



*Best practice:* If you do NOT need to reassign a variable use **const**. If you do need to reassign new values to a variable use **let**.

## Primitive Types

**What kind of types can we assign to a variable?**

We have 2 types in JS:

* **Primitives** – also called **Value Types**
* **Reference Types**

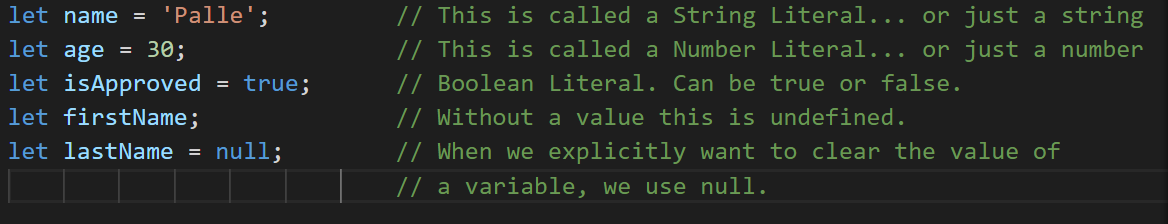
In this section we will address the Primitive types (Value Type).

Of the primitive types we have:

* String
* Number
* Boolean
* undefined
* null

Erase/delete content of **index.js** file.

Go to VS Code and code along:



More on the **null** value:

If we present the user with a list of colors and the user does not select anything – then we might have a variable named **selectedColor**. We want to set this to null to indicate that nothing was changed/selected. Later the user might select fx. yellow. Then the value of **selectedColor** should be set to this value of yellow.

A new type has arrived named ‘symbol’. We do not discuss that value here.

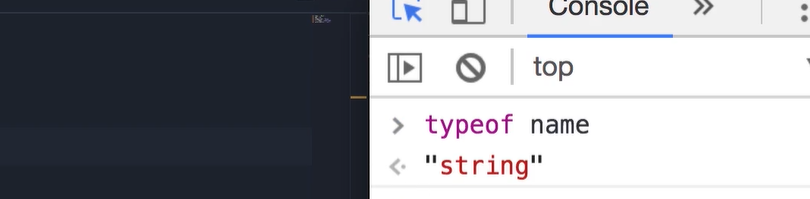
## JavaScript is a Dynamic Language (using Dynamic Typing)

In the programming world we have 2 types of languages:

* **STATIC** (Statically-typed) fx. C#: *string name = “Kim”; // Type cannot be changed*
* **DYNAMIC** (Dynamically-typed) fx. JS: let name = ‘Kim’; // Type can change at run time

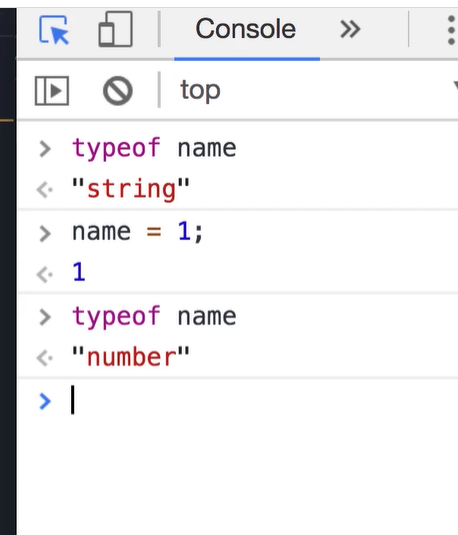
In VS Code we have the *name* variable set to ‘Palle’ -> a string type.

We can check this by going to the console and issue this command:



Running **typeof name** gives us the current type of the variable.

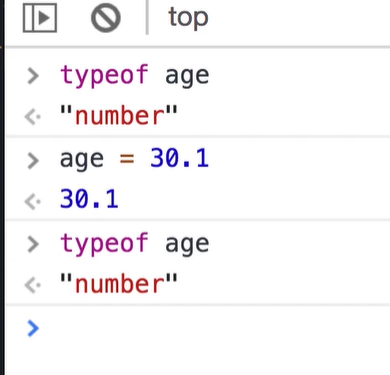
Stay in the console and type all this to see the type changed:



So, the type of variables changes at ‘run time’ based on the values that we assign to them.

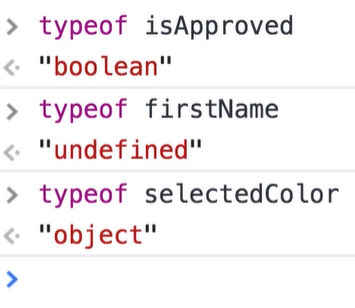
You can now clear the console by clicking within the console window – and press: **crtl + l (l as in all)**.

Test these commands within the console:



Notice that changing the value of age from an Integer to a Floating Point number does not change the type -> it is still just a ‘Number’. JS does not have 2 or more kinds of numbers -> just Number.

More tests in the console:



Notice: typeof firstName is “undefined”. Special thing here: “undefined” is both a value and a type.

Also notice: setting a variable to null turns the type into an Object type.

## Objects

We have now seen the Primitive Types/Value Types. It’s time to look at the Reference Types!

**Reference Types:**

* **Object**
* **Array**
* **Function**

In this section we will take a look at the **Object Type**.

*What is an object (in JavaScript)?*

Objects in JS are not much different than objects in real life: cars, buildings and persons.

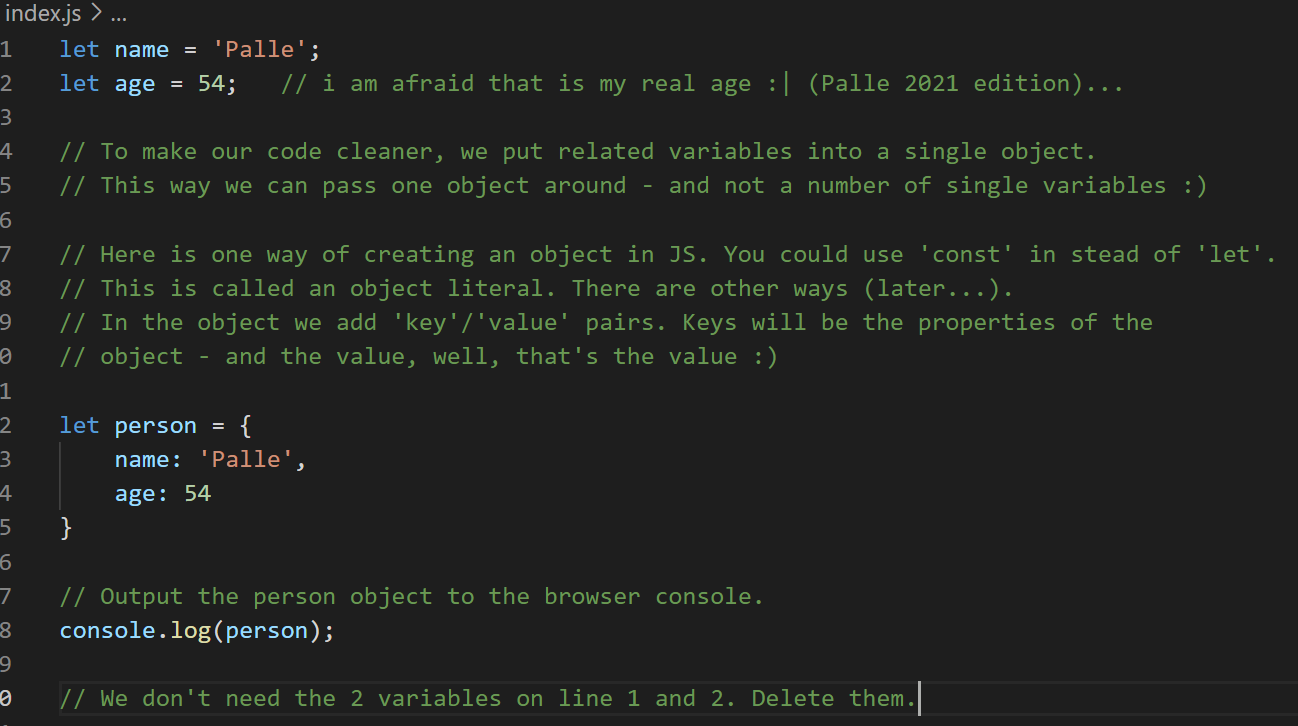
A Person has:

* name
* age
* address etc.

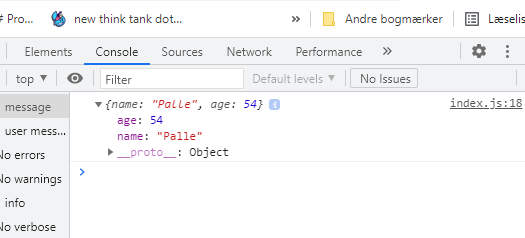
These are (some of) the *properties*of a Person.

When we deal with variables that are related, fx. name, age and address, we can put these related properties into an object.

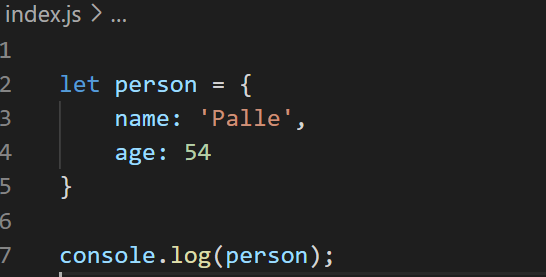
Open VS Code – and empty the **index.js** file. Enter this code:



Result of the console.log command:

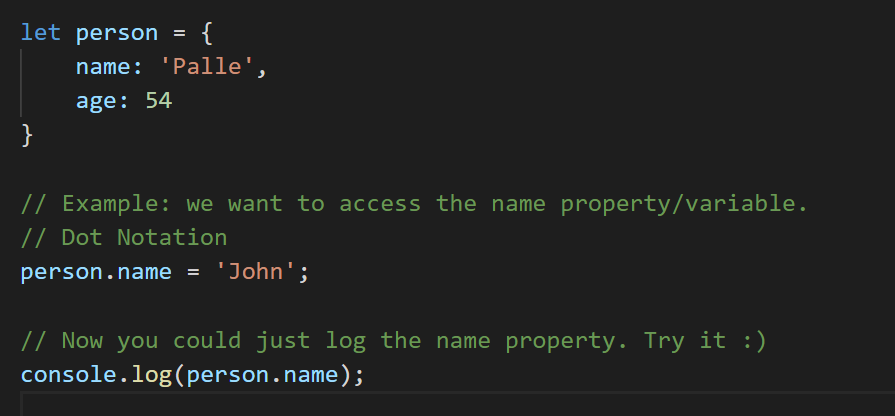


Back in VS Code. Make your code look like this:

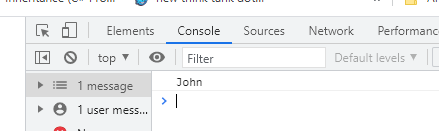


**2 *Ways to work with the properties of this object*:** **Dot Notation** and **Bracket Notation**!

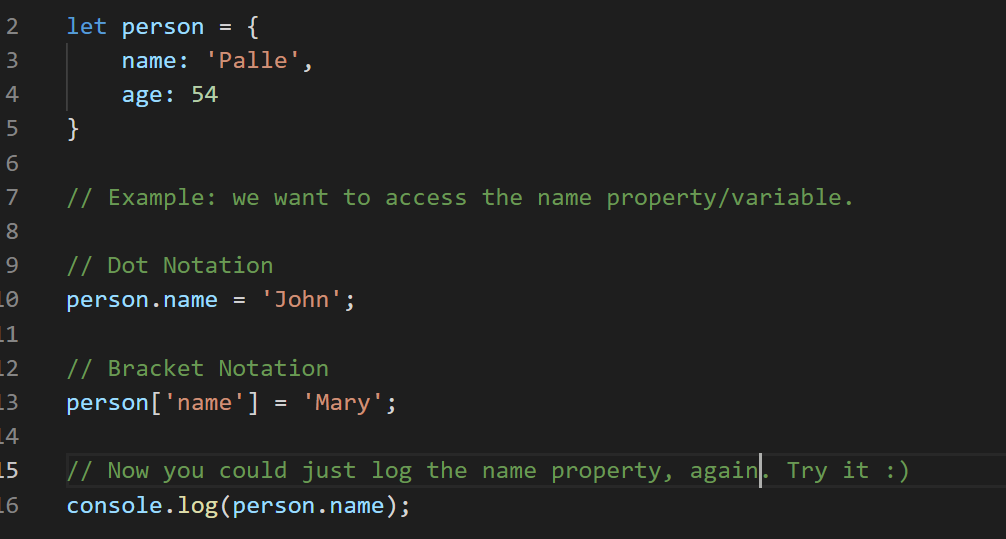
Go back to VS Code:



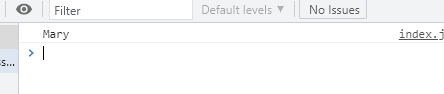
After you log the new value:



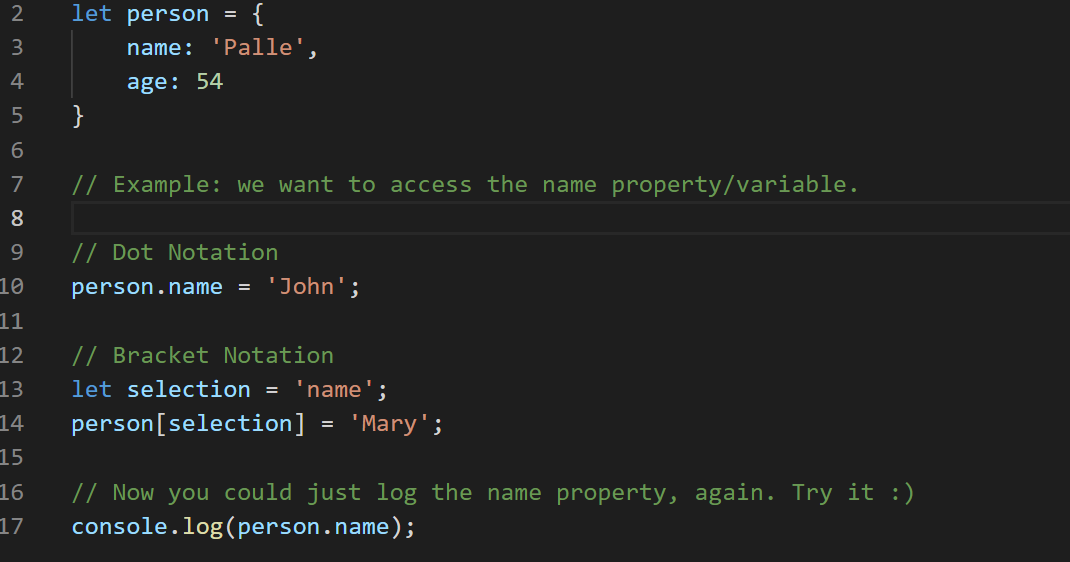
Back to VS Code:



After you log the new value:



*Best practice:* default to using the Dot Notation. It’s simpler. But Bracket Notation has its place: sometimes you do not know the name of a property until ‘run time’. Go to VS Code:

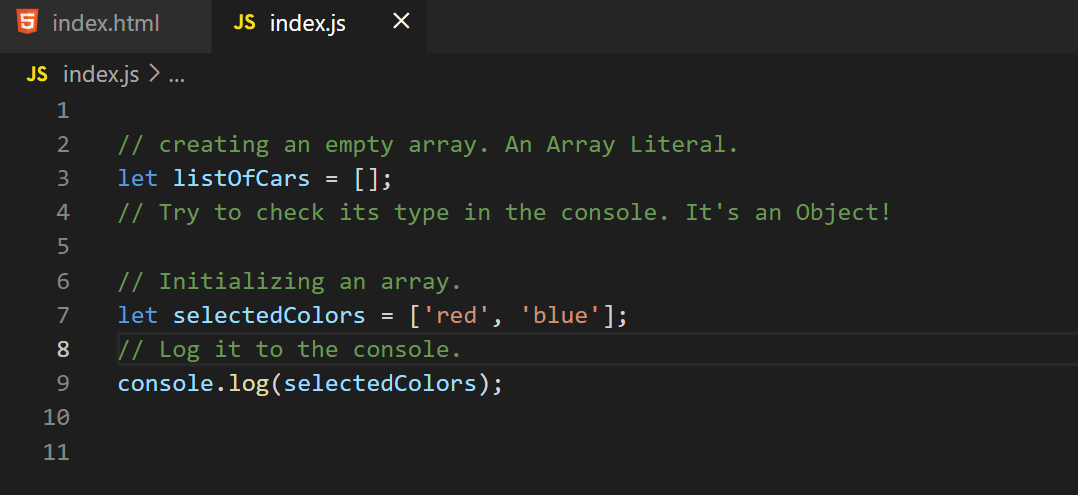


Notice the changes in lines 13 and 14.

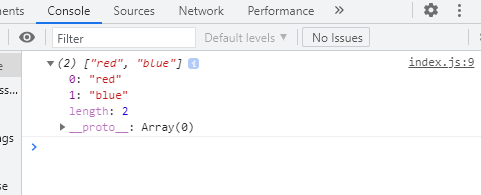
## Arrays

Good to have if you’re dealing with a list of objects/values.

Clear your **index.js** file – and start typing:

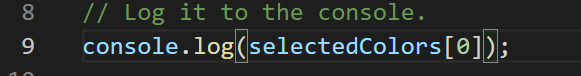


Check the console:



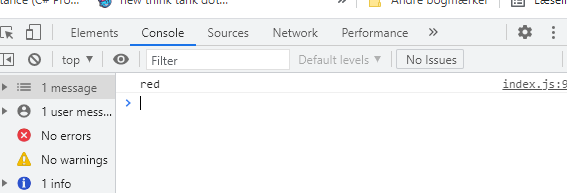
Notice, each element has an index. Indexes in an array starts with the value 0 and increments with 1, so, element 0 is the first element in an array, index 1 is the second element in the array.

Accessing elements in an array. Open VS Code:



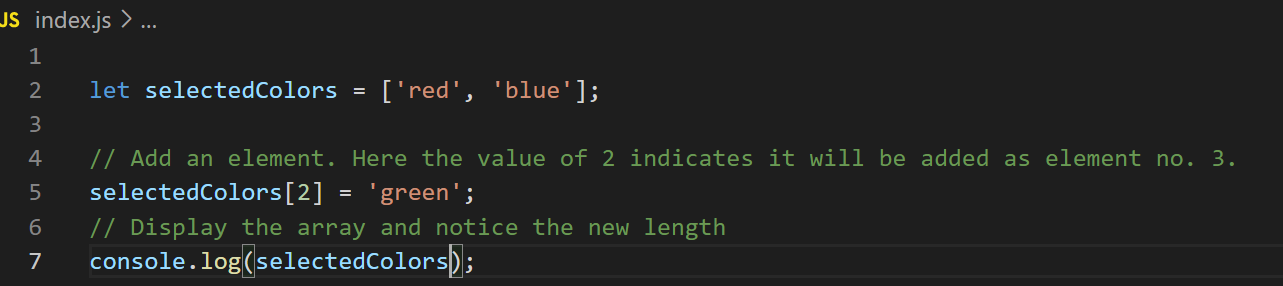
Notice the change in line 9.

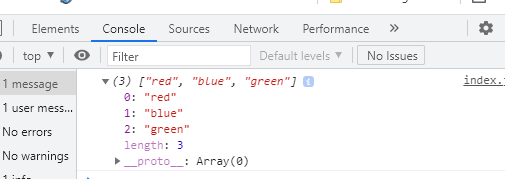
Check the console:



***Array content, length and types can change dynamically***.

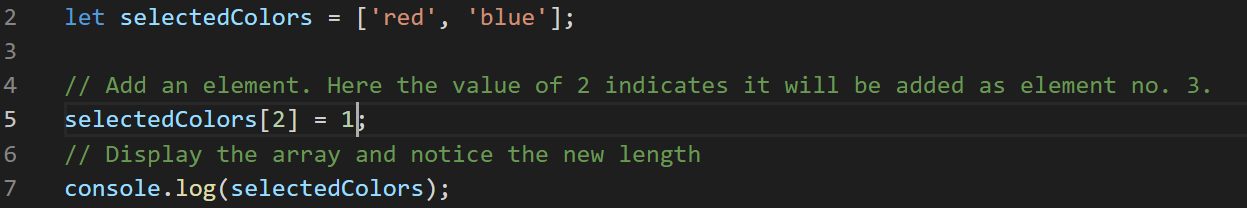
Back to VS Code and type:

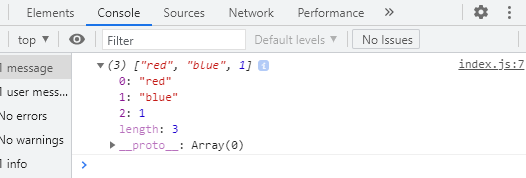




We can change any of the values to a new type. Back in VS Code:

Notice the change to line no. 5:





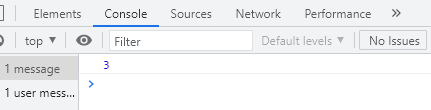
So, the types and length of an Array is Dynamic in JS.

As mentioned earlier, technically an Array is an Object.

You can always get the length of an array. VS Code:



Check the value in the console:



## Functions

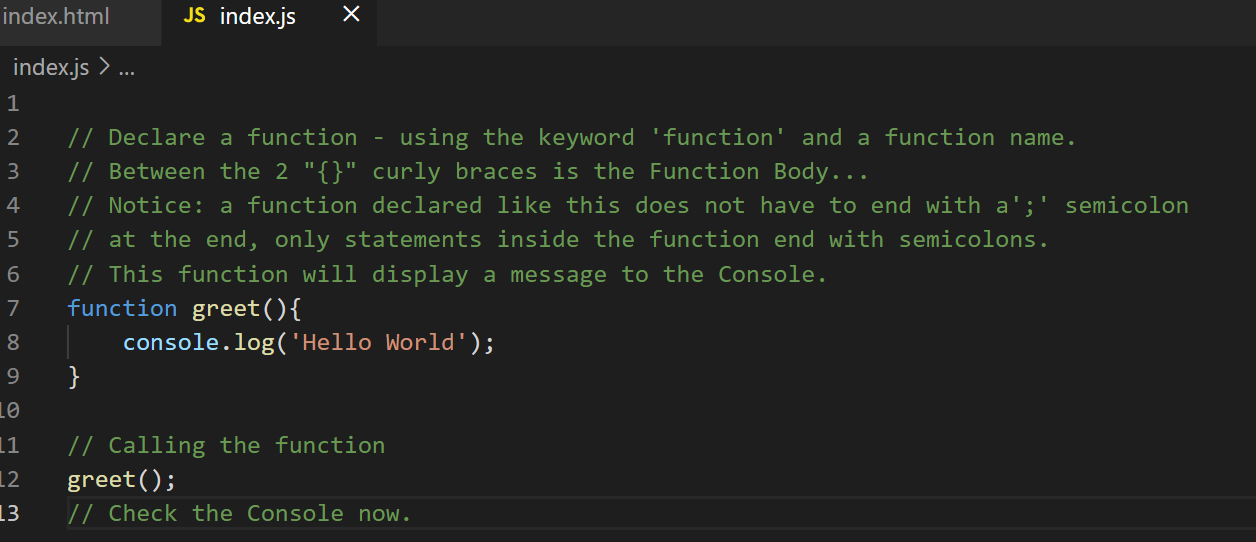
So far, we have worked with Arrays and Objects, 2 of the 3 Reference types.

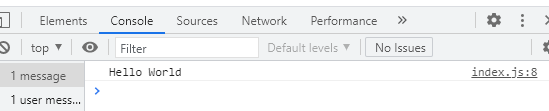
It’s time to look at the 3rd – Functions.

Functions are one of the fundamental building blocks of JS.

Basically, a Function is a set of statements that perform a task... or calculate a value.

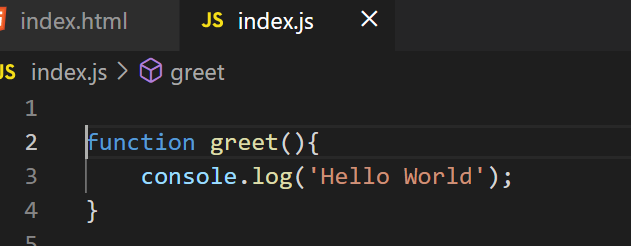
Open VS Code and follow along. First, though, empty the JS file:



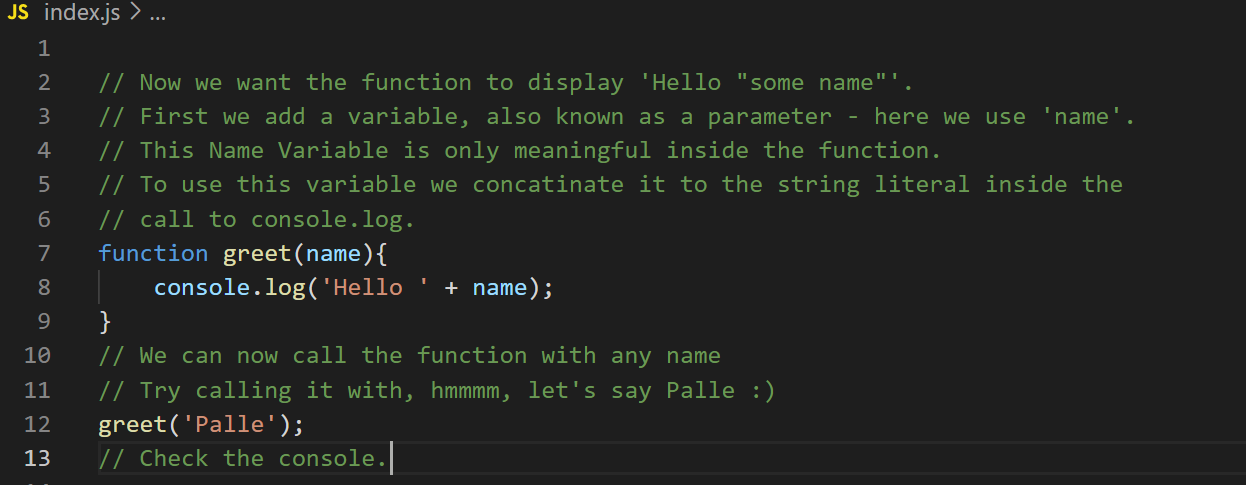


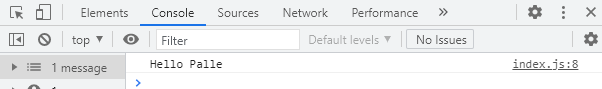
Our functions can have inputs to make them more dynamic.

Make your code look like this. Keep only the function declaration:



And code along:

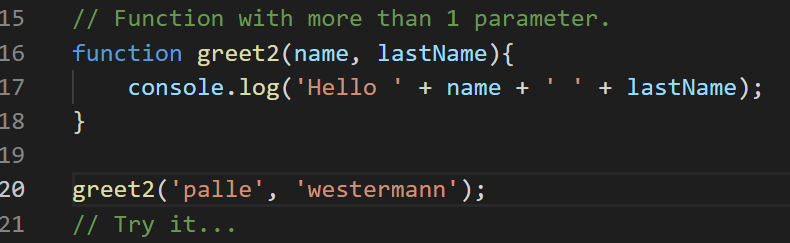




A *parameter* is the Variable inside the function declaration – between the parenthesis.

An *argument* is the Actual Value we use in the different calls to the function (that goes into the parameter).

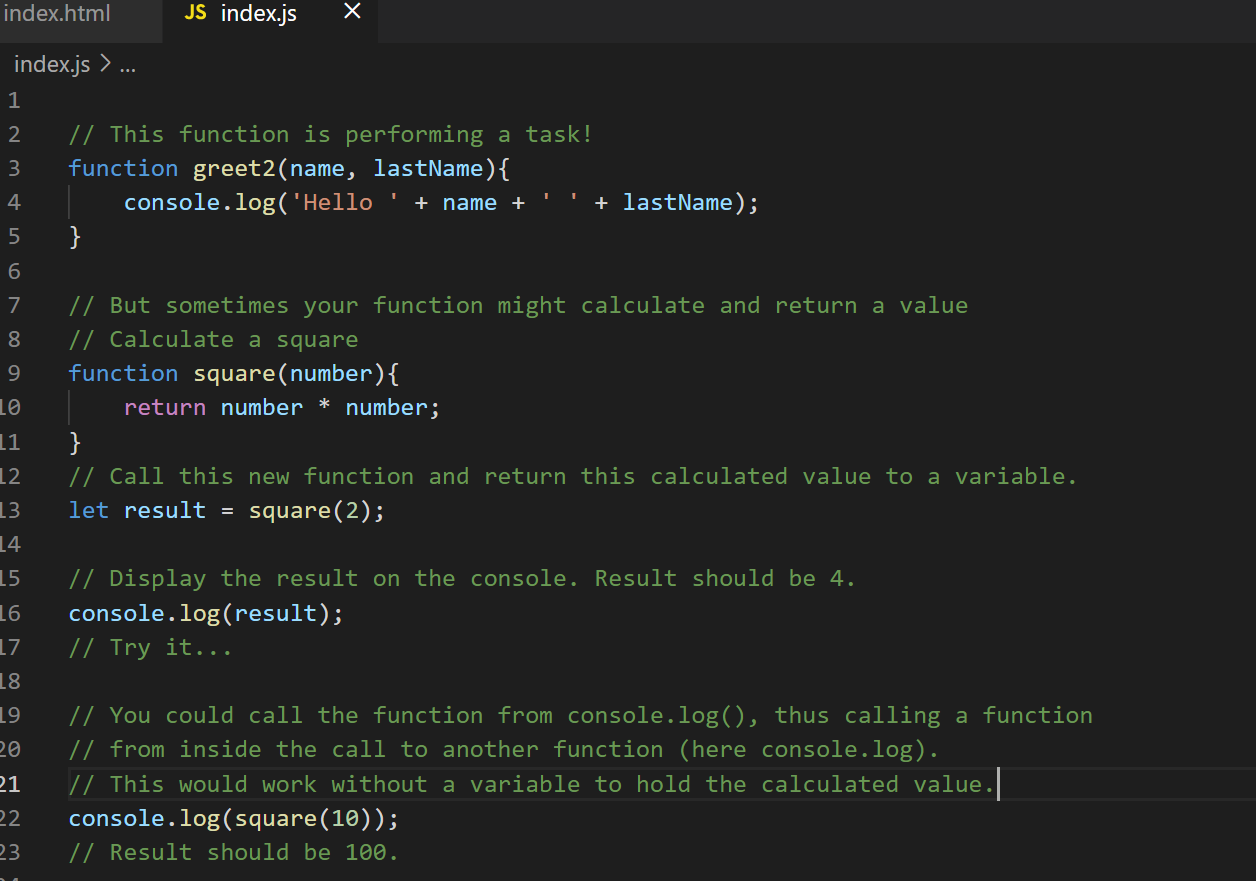
A function can have multiple parameters. Code along:



If you forget to call a Function fulfilling all of its parameters, the missing argument will be displayed as ‘undefined’ in the console.

## Types of functions

Code along. For starters keep only function greet2:



# Operators

## JavaScript Operators

x

# Control Flow

x

# Objects

x

# Arrays

x

# Functions

x