// single line comment in Javascript
/* multi lines comment in javascript*/

Use to be called livewire, has nothing to do with Java, the name is a marketing decision.
In the body of the HTML (inline): <script type="text/javascript"> alert('Hi there');   </script>
Or even better an **external file**, like for CCS so this can be used with all my html pages.

You can add the file in the header part:
<script type="text/javascript" src="javascript.js"></script>
But it's better to put the external file declaration at the end of the html code as having it at the beginning slows down the loading of page hence a worse user experience. But sometime you'll need the javascript file at the start for some initial display features. In that case, keep the file as small as possible to reduce the loading time of the page. You can have more than on javascript file.

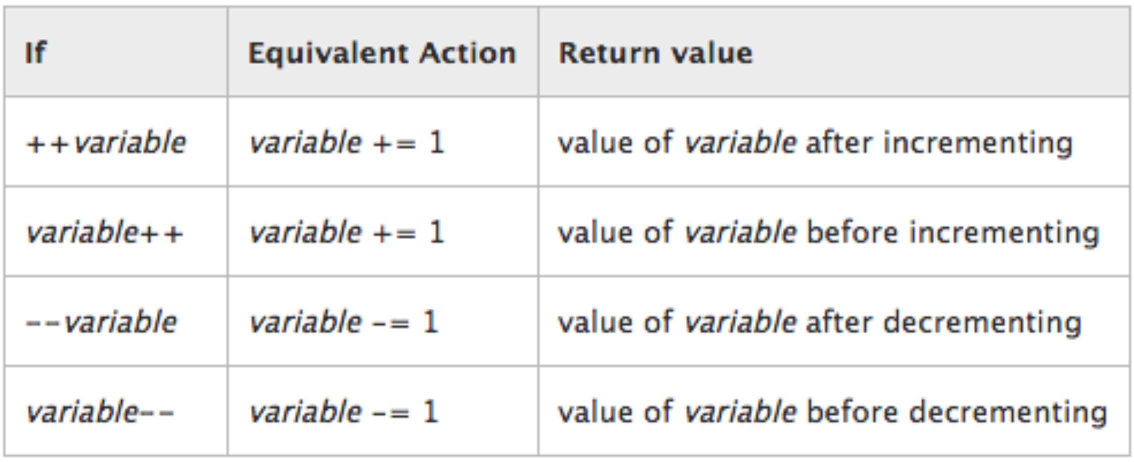
Var username = 5; Username = 'Jones'; Alert(username);

Most browsers won't need a **semi colon** in the code if you did a carriage return at the end of the line, but most people still put the semi colon as they **minify** (compact) their file (try to make their file as small as possible and put everything in one line) .

Various objects: date, string, math and more. They have properties and methods. To access them, just add a dot and the name of the property/method to the object name.

Javascript is case-sensitive.

= to assign
+ - * / ++ (increment) – (decrement) % modulo, give the remainder of the division
result **+=** expression is the same as result = result + expression , but result is evaluated only once.



If	Equivalent Action	Return value
++*variable*	*variable* += 1	value of *variable* after incrementing
variable++	*variable* += 1	value of *variable* before incrementing
--*variable*	*variable* -= 1	value of *variable* after decrementing
variable--	*variable* -= 1	value of *variable* before decrementing

List of comparison operators:
> Greater than
< Less than
<= Less than or equal to
>= Greater than or equal to
=== strict equality (values and types) **==** equality (values)
!== Not equal to

Booleans: && || !
Falsy values: null undefined NaN, "" , 0
Truthy values: everything else

Ternary operator:
If you need to assign a value to a variable, and that value changes depending on another, then use a ternary statement to do the assignment.
Using an if statement, it looks like this:

```
var isAllowed;
if(age > 18) {
   isAllowed = "yes";
} else {
   isAllowed = "no";
}
```

This is a very simple example, but it can be shortened to this:

`var isAllowed = (age > 18) ? "yes": "no";`

The benefit of a ternary is that it is less code, and you can do a declaration and condition-dependant assignment on one line. The first statement after the question mark (the "yes") is what will be assigned if the preceding condition evaluates to true, the second statement (the "no") is what will be assigned if the condition evaluates to false.

Checking to see if something is truthy can simply be done like this:

`var duration = x || 4;`

This is checking if the x variable has a truthy value, if it does, assign it to duration, else assign 4 to duration. If you need to assign 0 to duration, then this approach would not work as 0 is a falsy value and therefore it would fall through to the 4 being assigned to duration.

You can also use this syntax to do an inline-if statement:

`var duration = hasDuration && x;`

This checks to see if hasDuration is a truthy value, if it is, then it falls through to the next part and assigns the value of x to duration

if (condition) { code will execute} else {do this}
if (/* Some condition */) { Do something }
else if (/* Some other condition */) { Do something else}
else { Otherwise, Do a third thing}

switch allows you to preset a number of options (called cases), then check an expression to see if it matches any of them. If there's a match, the program will perform the action for the matching case; if there's no match, it can execute a default option.

```
switch (some expression) {
case ‘value1’ : do something ;  break;
case ‘value 2’ : do something else; break;
etc…
default :  do whatever
}
```

for (var i = val1; i < valuemax ; i++) {instruction;}
Be very careful with your syntax—if you write a loop that can't properly end, it's called **an infinite loop**. It will crash your browser!   i-- or i+=x (i+=3 to increment by 3)

The while loop is ideal when you want to use a loop, but you don't know how many times you'll have to execute that loop:
While (condition) { // do something }
Sometimes you want to make sure your loop runs at least one time no matter what. When this is the case, you want a modified while loop called **a do/while loop**.
Do {something} while (condition);

Functions

var functionName = **function** (parameter1, par2,par3) { ;   ; ;};
At the end of each line of code (within the { }) and after the entire function (after the { }),put a semi-colon. If return, put it at the end of the function body.

A function can have no name when it’s inline javascript:
Function bob(x, y, z) {code of the function}
Then further in the code bob(5, 0, 6);

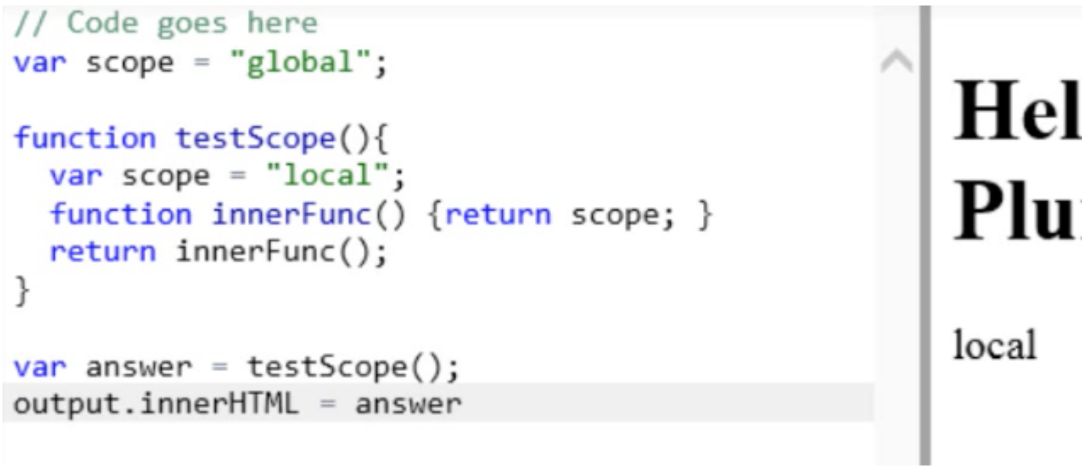
Trick to delimit a quote sign “ ’ ” or for a double quote sign ‘ “ ‘

Using the var keyword inside a function, declares a new **local variable that** only exists within that function.

hoisting teaches that variable and function declarations are physically moved to the top of your coding, but this is not what happens at all. What does happen is that **variable and function declarations are put into memory during the compile phase**, but stays exactly where you typed it in your coding.
One of the advantages of JavaScript putting function declarations into the memory before it executes any code segment is that it allows you to use a function before you declare it in your code.

In many languages, braces create scope, but not in javasscript! The only thing that creates scope in javascript is function.

Closure: a function scopes its variables at the time it is declared, not at the time it is run.



```
// Code goes here
var scope = "global";

function testScope(){
  var scope = "local";
  function innerFunc() {return scope; }
  return innerFunc();
}

var answer = testScope();
output.innerHTML = answer
```

```
Hel
Plu

local
```

Objects and collections
Objects are collections of properties. You can add additional properties on the fly. Non existent properties return undefined. Objects can have functions (methods).

Arrays act as collections

Array, can be mix data: var namOfArray = [data1, data2, …];
Arrays have 0-based indexing, so we start counting the positions from 0. arrayName[i-1] to access the ith element.
For (var i = 0; i < arrayName.length; i++) {action nameArray[i]} to search the whole array.
Sometimes you want arrays that aren't as nice and even as your 3 x 3 two-dimensional array: you may have three elements in the first row, one element in the second row, and two elements in the third row. JavaScript allows those, and they're called **jagged arrays.**

Strings:
use either single or double quotes
escape sequences begin with \ for example: \n for new line
use + for concatenation
many methods availabe: charat(index), indexOf(string), split cut a string into an array, slice copies a piece of the string.
regexp: myString(search / Whatever you look for/)

AJAX = asynchronous Javascript and xml

The idea is to have a webpage sending small piece of data to the server without reloading the webpage. The user can do other things while some other things happens in the background

JSON = Javascript Object Notation https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

JSON is a standard format **for representing structured data as JavaScript objects**, which is commonly used for representing and **transmitting data on web sites** (i.e. sending some data from the server to the client, so it can be displayed on a web page).

Even though it is based on JavaScript syntax, it can be used independently from JavaScript, and many programming environments feature the ability to read (parse) and generate JSON.

JSON can exist as an object, or a string — the former is used when you want to read data out of the JSON, and the latter is used when you want to send the JSON across the network. This is not a big issue — JavaScript provides a global JSON object that has methods available for converting between the two.

- JSON is purely a data format — it contains only properties, no methods.
- **JSON requires double quotes to be used to be valid.** It is safest to write it with double quotes, not single quotes.
- Even a single misplaced comma or colon can cause a JSON file to go wrong, and not work. You should be careful to validate any data you are attempting to use (although computer-generated JSON is less likely to include errors, as long as the generator program is working correctly). You can validate JSON using an application like JSONLint.
- JSON can actually take the form of any data type that is valid for inclusion inside a standard JSON object, not just arrays or objects. So for example, a single string or number would be a valid JSON object. Not that this would be particularly useful...

To load our JSON into our page, we are going to use an API called **XMLHttpRequest (often called XHR)**. This is a very useful JavaScript object that allows us to make network requests to retrieve resources from a server via JavaScript (e.g. images, text, JSON, even HTML snippets), meaning that we can update small sections of content without having to reload the entire page. This has led to more responsive web pages

API: a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.
In general terms, it's a **set of clearly defined methods of communication between various software components**. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.

Client Side web APIs: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs

Server-side website programming: https://developer.mozilla.org/en-US/docs/Learn/Server-side

Javascript server side : https://developer.mozilla.org/en-US/docs/Learn/Server-side

Getting started with server-side programming is usually easier than with client-side development, because dynamic websites tend to perform a lot of very similar operations (retrieving data from a database and displaying it in a page, validating user-entered data and saving it in a database, checking user permissions and logging users in, etc.), and are constructed using web frameworks that make these and other common web server operations easy.