Variable

Data type

Operator

Condition

Image

Dom finder

Local Storage

Export

1. First, go to your test site and create a new folder named 'scripts' (without the quotes). Then, within the new scripts folder you just created, create a new file called main.js. Save it in your scripts folder.
2. Next, in your index.html file enter the following element on a new line just before the closing </body> tag:

<script src="scripts/main.js"></script>

1. This is basically doing the same job as the <link> element for CSS — it applies the JavaScript to the page, so it can have an effect on the HTML (along with the CSS, and anything else on the page).
2. Now add the following code to the main.js file:
3. var myHeading = document.querySelector('h1');

myHeading.textContent = 'Hello world!';

var myVariable;

After declaring a variable, you can give it a value:

myVariable = 'Bob';

You can do both these operations on the same line if you wish:

var myVariable = 'Bob';

You can retrieve the value by just calling the variable by name:

myVariable;

After giving a variable a value, you can later choose to change it:

var myVariable = 'Bob';

myVariable = 'Steve';

Note that variables may hold values that have different data types:

| **Variable** | **Explanation** | **Example** |
| --- | --- | --- |
| **String** | A sequence of text known as a string. To signify that the value is a string, you must enclose it in quote marks. | var myVariable = 'Bob'; |
| **Number** | A number. Numbers don't have quotes around them. | var myVariable = 10; |
| **Boolean** | A True/False value. The words true and false are special keywords in JS, and don't need quotes. | var myVariable = true; |
| **Array** | A structure that allows you to store multiple values in one single reference. | var myVariable = [1,'Bob','Steve',10]; Refer to each member of the array like this: myVariable[0], myVariable[1], etc. |
| **Object** | Basically, anything. Everything in JavaScript is an object, and can be stored in a variable. Keep this in mind as you learn. | var myVariable = document.querySelector('h1'); All of the above examples too. |

Comments

You can put comments into JavaScript code, just as you can in CSS:

/\*

Everything in between is a comment.

\*/

If your comment contains no line breaks, it's often easier to put it behind two slashes like this:

// This is a comment

Operators

An operator is a mathematical symbol which produces a result based on two values (or variables). In the following table you can see some of the simplest operators, along with some examples to try out in the JavaScript console.

| **Operator** | **Explanation** | **Symbol(s)** | **Example** |
| --- | --- | --- | --- |
| **Addition** | Used to add two numbers together or glue two strings together. | + | 6 + 9; "Hello " + "world!"; |
| **Subtraction, Multiplication, Division** | These do what you'd expect them to do in basic math. | -, \*, / | 9 - 3; 8 \* 2; // multiply in JS is an asterisk 9 / 3; |
| **Assignment** | You've seen this already: it assigns a value to a variable. | = | var myVariable = 'Bob'; |
| **Equality** | Does a test to see if two values are equal to one another and returns a true/false (Boolean) result. | === | var myVariable = 3; myVariable === 4; |
| **Not, Does-not-equal** | Returns the logically opposite value of what it precedes; it turns a trueinto a false, etc. When it is used alongside the Equality operator, the negation operator tests whether two values are *not* equal. | !, !== | The basic expression is true, but the comparison returns false because we've negated it:  var myVariable = 3; !(myVariable === 3);  Here we are testing "is myVariable NOT equal to 3". This returns false because myVariable IS equal to 3.  var myVariable = 3; myVariable !== 3; |

### Conditionals

Conditionals are code structures which allow you to test if an expression returns true or not, running alternative code revealed by its result. A very common form of conditionals is the if ... else statement. For example:

var iceCream = 'chocolate';

if (iceCream === 'chocolate') {

alert('Yay, I love chocolate ice cream!');

} else {

alert('Awwww, but chocolate is my favorite...');

}

let x = 1;

switch(x) {

case 0:

let foo;

break;

case 1:

let foo; // SyntaxError for redeclaration.

break;

}

The expression inside the if ( ... ) is the test — this uses the identity operator (as described above) to compare the variable iceCream with the string chocolate to see if the two are equal. If this comparison returns true, the first block of code is run. If the comparison is not true, the first block is skipped and the second code block, after the else statement, is run instead.

### Functions

Functions are a way of packaging functionality that you wish to reuse. When you need the procedure you can call a function, with the function name, instead of rewriting the entire code each time. You have already seen some uses of functions above, for example:

1. var myVariable = document.querySelector('h1');
2. alert('hello!');
3. function multiply(num1,num2) {
4. var result = num1 \* num2;
5. return result;
6. }
7. Try running the above function in the console, then test with several arguments. For example:
8. multiply(4,7);
9. multiply(20,20);
10. multiply(0.5,3);

### Events

Real interactivity on a website needs events. These are code structures which listen for things happening in browser, running code in response. The most obvious example is the click event, which is fired by the browser when you click on something with your mouse. To demonstrate this, enter the following into your console, then click on the current webpage:

document.querySelector('html').onclick = function() {

alert('Ouch! Stop poking me!');

}

There are many ways to attach an event to an element. Here we select the <html>element, setting its onclick handler property equal to an anonymous (i.e. nameless) function, which contains the code we want the click event to run.

Note that

document.querySelector('html').onclick = function() {};

is equivalent to

var myHTML = document.querySelector('html');

myHTML.onclick = function() {};

Adding an image changer

In this section, we'll add an additional image to our site using some more DOM API features, using some JavaScript to switch between the two when the image is clicked.

1. First of all, find another image you'd like to feature on your site. Be sure it is the same size as the first image, or as close as possible.
2. Save this image in your images folder.
3. Rename this image 'firefox2.png' (without quotes).
4. Go to your main.js file, and enter the following JavaScript. (If your "hello world" JavaScript is still there, delete it.)
5. var myImage = document.querySelector('img');
6. myImage.onclick = function() {
7. var mySrc = myImage.getAttribute('src');
8. if(mySrc === 'images/firefox-icon.png') {
9. myImage.setAttribute ('src','images/firefox2.png');
10. } else {
11. myImage.setAttribute ('src','images/firefox-icon.png');
12. }

}

1. Save all files and load index.html in the browser. Now when you click the image, it should change to the other one!

You store a reference to your <img> element in the myImage variable. Next, you make this variable's onclick event handler property equal to a function with no name (an "anonymous" function). Now, every time this element is clicked:

1. You retrieve the value of the image's src attribute.
2. You use a conditional to check whether the src value is equal to the path to the original image:
   1. If it is, you change the src value to the path to the 2nd image, forcing the other image to be loaded inside the <img> element.
   2. If it isn't (meaning it must already have changed), the src value swaps back to the original image path, to the original state.

Adding a personalized welcome message

Next we will add another bit of code, changing the page's title to a personalized welcome message when the user first navigates to the site. This welcome message will persist, should the user leave the site and later return — we will save it using the Web Storage API. We will also include an option to change the user and, therefore, the welcome message anytime it is required.

1. In index.html, add the following line just before the <script> element:

<button>Change user</button>

1. In main.js, place the following code at the bottom of the file, exactly as written — this takes references to the new button and the heading, storing them inside variables:
2. var myButton = document.querySelector('button');

var myHeading = document.querySelector('h1');

1. Now add the following function to set the personalized greeting — this won't do anything yet, but we'll fix this in a moment:
2. function setUserName() {
3. var myName = prompt('Please enter your name.');
4. localStorage.setItem('name', myName);
5. myHeading.textContent = 'Mozilla is cool, ' + myName;

}

This function contains a prompt() function, which brings up a dialog box, a bit like alert(). This prompt(), however, asks the user to enter some data, storing it in a variable after the user presses **OK***.* In this case, we are asking the user to enter their name. Next, we call on an API called localStorage, which allows us to store data in the browser and later retrieve it. We use localStorage's setItem() function to create and store a data item called 'name', setting its value to the myName variable which contains the data the user entered. Finally, we set the textContent of the heading to a string, plus the user's newly stored name.

1. Next, add this if ... else block — we could call this the initialization code, as it structures the app when it first loads:
2. if(!localStorage.getItem('name')) {
3. setUserName();
4. } else {
5. var storedName = localStorage.getItem('name');
6. myHeading.textContent = 'Mozilla is cool, ' + storedName;

}

This block first uses the negation operator (logical NOT, represented by the !) to check whether the name data exists. If not, the setUserName() function is run to create it. If it exists (that is, the user set it during a previous visit), we retrieve the stored name using getItem() and set the textContent of the heading to a string, plus the user's name, as we did inside setUserName().

1. Finally, put the below onclick event handler on the button. When clicked, the setUserName() function is run. This allows the user to set a new name, when they wish, by pressing the button:
2. myButton.onclick = function() {
3. setUserName();

}

Export

The **export** statement is used when creating JavaScript modules to export functions, objects, or primitive values from the module so they can be used by other programs with the import statement.

## Syntax

export { name1, name2, …, nameN };

export { variable1 as name1, variable2 as name2, …, nameN };

export let name1, name2, …, nameN; // also var

export let name1 = …, name2 = …, …, nameN; // also var, const

export function FunctionName(){...}

export class ClassName {...}

export default expression;

export default function (…) { … } // also class, function\*

export default function name1(…) { … } // also class, function\*

export { name1 as default, … };

export \* from …;

export { name1, name2, …, nameN } from …;

export { import1 as name1, import2 as name2, …, nameN } from …;

export { default } from …;

**nameN**

Identifier to be exported (so that it can be imported via import in another script).

## Description

There are two different types of export, **named** and **default**. You can have multiple named exports per module but only one default export. Each type corresponds to one of the above syntax:

* Named exports:
* // exports a function declared earlier
* export { myFunction };
* // exports a constant

export const foo = Math.sqrt(2);

* Default exports (function):

export default function() {}

* Default exports (class):

export default class {}

Named exports are useful to export several values. During the import, it is mandatory to use the same name of the corresponding object.

But a default export can be imported with any name for example:

export default k = 12; // in file test.js

import m from './test' // note that we got the freedom to use import m instead of import k, because k was default export

console.log(m); // will log 12

The following syntax does not export a default export from the imported module:

export \* from …;

If you need to export the default, write the following instead:

import mod from 'mod';

export default mod;

## Examples

### Using named exports

In the module, we could use the following code:

// module "my-module.js"

function cube(x) {

return x \* x \* x;

}

const foo = Math.PI + Math.SQRT2;

var graph = {

options:{

color:'white',

thickness:'2px'

},

draw: function(){

console.log('From graph draw function');

}

}

export { cube, foo, graph };

This way, in another script, we could have:

import { cube, foo, graph } from 'my-module';

graph.options = {

color:'blue',

thickness:'3px'

};

graph.draw();

console.log(cube(3)); // 27

console.log(foo); // 4.555806215962888

### Using the default export

If we want to export a single value or to have a fallback value for our module, we could use a default export:

// module "my-module.js"

export default function cube(x) {

return x \* x \* x;

}

Then, in another script, it will be straightforward to import the default export:

import cube from 'my-module';

console.log(cube(3)); // 27