JS Recap

1. .addEventListener

A [method](https://recap.kea-alt-del.dk/" \l "method) on [Nodes](https://recap.kea-alt-del.dk/" \l "node) allowing us to hook up [events](https://recap.kea-alt-del.dk/" \l "events). When the event occurs, it (JavaScript) fires of a "callback" (a function).

The function that receives the event receives information about the event as an argument (often called e because we're lazy)

Basic syntax

someNode.addEventListener("event", callback);

function callback(e) {

//do stuff

}

Often used with [anonymous functions](https://recap.kea-alt-del.dk/" \l "anonymousfunctions).

someStuff.addEventListener("click", function (e) {

//do stuff

});

And since ES6, with the "fat arrow" syntax:

someElement.addEventListener("click", (e) => {

//do stuff

1. document

The document is an object living inside the window object. So technically, the document can be accessed with window.document, but we can also simply write document.

The document is the HTML that's loaded into the browser, including <head>, <body> etc.

w3schools maintain a list of properties on the document object

1. DOM

Document Object model is a term describing the relationship between Nodes in the document. The main thing to understand is that the DOM is a hierachy, consiting of parents, siblings and children.

The following snippet will be used to describe the various relationships.

<ul>

<li><a href=""></a></li>

<li><a href=""></a></li>

<li><a href=""></a></li>

<li><a href=""></a></li>

</ul>

***parent***

Each node can have one, and only one parent. That is, the element that the node is nested inside. The <html> node is the only node that does not have a parent.

I the above example, each <li> is a parent to an <a>node, and the <ul> is the parent to all the <li>'s

***sibling***

Each node can have a number of siblings. Siblings a re nodes nested on the same level, that have the same parent. In the example above, the <li>'s are siblings, the <a>s are not.

***children***

You probably get this by now, above, the <ul> & <li>'s have children. <ul>has four, each <li> have one

1. Events

See also [.removeEventListener](https://recap.kea-alt-del.dk/" \l "removeeventlistener) and [.addEventListener](https://recap.kea-alt-del.dk/" \l "addeventlistener). Events (another key concept in JS) are things that happen in our applications. The most common ones are listed below, but a lot of plugins add more, and [the full list](https://developer.mozilla.org/en-US/docs/Web/Events) is huge. We say that an event "fires" when someone or something triggers it.

The [callback](https://recap.kea-alt-del.dk/" \l "callback) function that handles the event, receives an argument that contains information about the event. See the examples below.

Common Events

| **Event** | **Action** |
| --- | --- |
| animationend | Triggers when any CSS animation ends on the target |
| click | When the user clicks the event target |
| ended | Triggers when the target audio/video ends |
| keyup | Triggers everytime a key (keyboard) is released |
| load | Triggers when the page, all scripts and all stylesheets have loaded |
| mouseout | Triggers when the mouse leaves the target |
| mouseover | Triggers when the mouse enters the target element |
| transitionend | Triggers when any CSS transition ends on the target |

[MDN has a list that is a bit bigger :-)](https://developer.mozilla.org/en-US/docs/Web/Events)

Examples

See [.addEventListener](https://recap.kea-alt-del.dk/" \l "addeventlistener) for more examples.

load

window.addEventListener("load", allReady);

function allReady() {

// everything is loaded, we're good to go

}

keyup

window.addEventListener("keyup", fingerLifted);

function fingerLifted(e) {

// The callback receives an argument (here, I chose to call it e)

console.log(e); //Look at the console, lot's of useful stuff in here

/\* for instance, e.code will tell us which key was pressed, with a simple if statement,

we can start all sorts of crazy stuff \*/

if (e.code === "KeyS") {

//s key released

//do whatever

}

}

1. Functions

Functions are a huge topic, so let's dive in. First off,

1. functions are named pieces of code that can be reused.
2. Secondly, they provide us with a new scope. This means, that variables declared inside the function is **only available inside that function**
3. They help us manage and structure our scripts as they grow

### Naming functions

Functions follow the same naming rules as [variables](https://recap.kea-alt-del.dk/" \l "variablesletconstvar) and you can't (you can but shouldn't) call a functions the same thing you've called a variables, so the following is generally really bad:

// BAD STUFF HERE!!!

var word = "here's text";

function word() {

console.log("Do something");

}

### Function definitions

Defining a function is easy (although once you really get to learn the language, you'll see several other ways to do it). First off we have the function keyword, and then the name, followed by parenthesis () and finally the "body" of the function, wrapped in curly brackets {}

function someMadeUpStuff() {

//everything in here will get executed when the function is called

}

### Calling functions

Calling a function is also easy :-)

Once the function is defined, we add () after the function name to "call it"

//defining it

function sayHello() {

console.log("Hello");

}

//calling it

sayHello();

//and again

sayHello();

### Function parameters

Functions can receive arguments (often called parameters, the difference is not important now), which makes them really flexible. These parameters become local variables in the function, let's see

function sayHello(name) {

//now this function has a local variable called `name`

console.log("Hi " + name);

}

//calling it, and supplying a name, is as easy as

sayHello("Jonas");

So, when calling the function (as in sayHello("Jonas") ), the string "Jonas" is passed to the function, and in the function, it becomes the variable name.

The function does not know what name contains, and it doesn't care, in our example it just console.log's it.

To round off, we're gonna make a function that receives three parameters:

function addItUp(a, b, c) {

let sum = a + b + c;

console.log("The sum is " + sum);

}

addItUp(2, 8, 5); // a = 2, b = 8, c = 5

### Functions as event listeners

Please see [.addEventListener](https://recap.kea-alt-del.dk/" \l "addeventlistener) for usage. When using function as eventlisteners, they automatically receive 1 argument, information about the event. This argument is an object containing information about the event (such as when, where, what). Do a console.log on it to see what it contains.

### A few best practices

A (non-exhaustive) list of things to keep in mind.

1. Functions should do one thing, and one thing only
2. Name your functions wisely (you will get lost otherwise)
3. Keep it DRY (Don't Repeat Yourself)
4. if, else, else if

Understanding [Booleans](https://recap.kea-alt-del.dk/" \l "boolean) before proceeding is a really, really good idea :-)

We can make our code execute only if a specific condition is met using if, else if and else statements.

The if (and else if) statements takes an expression, evaluate it, and if the result is truethe code in the following brackets is executed.

let hobby = "JS"; //Let's imagine it comes from somewhere else

if (hobby === "Horses") {

console.log("Good for you");

} else if (hobby === "Beer") {

console.log("Great");

} else {

console.log("Fantastic, " + hobby + " is a nice hobby");

}

1. Inline Scripts

See also [Linking our JS files](https://recap.kea-alt-del.dk/" \l "linkingourjsfiles).

Besides having our JS in separate files, we can embed it directly in our HTML documents like this:

...

<body>

<div class="stuff"></div>

...

<script>

//here we go

</script>

</body>

...

This limits code-reuse quite a lot, but it's excellent for quick tests

1. Linking our JS files

See also Inline Scripts

The proper way of linking our script is like this:

<script src="folder/file.js"></script>

And the best place to do it is right before closing the <body>-tag. That ensures that all the DOM elements are ready before we start manipulating them

1. variables, let, const & var

In JS we have three ways of creating variables, and each have their own advantages & disadvantages. const and let are new additions to the language, and not fully supported in all browsers, but for now, go ahead and use them.

var

Fully supported, the old kid on the block. Has "function scope", meaning that it's available everywhere in the function it's declared.

var myName = "Jonas";

function sayHi() {

//myName is available

if (2 > 1) {

var x = "Hi";

}

//x is available

}

let

One of the new kids on the block. Has a "block scope", meaning that it's available in the block it was declared in

let myName = "Jonas";

function sayHi() {

//myName is available here

if (2 > 1) {

let x = "Hi";

// x is ONLY available in this if statement

}

}

const

Another new kid. Has the same scope as let variables, but cannot be redeclared, and the value cannot change.

const message = "Hi";

//ERROR:

message = "goodbye";

const students = [];

//FINE:

students.push("Jonas");

1. The console

An object in the browser, allowing us to write JS and see the output of our scripts in the browser. Not useful without it's [methods](https://recap.kea-alt-del.dk/" \l "method). Have it open at all times!

### console.log(msg)

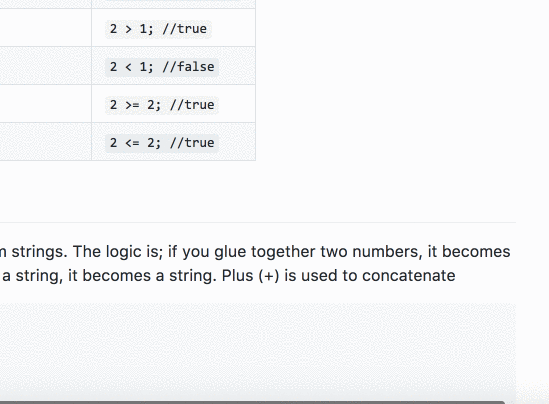
Outputs the content of a variable, expression etc, in the console.

let x = document.querySelectorAll("p");

console.log(x);

console.log(2 > 1);

### Disabling the cache

When developing (and this goes for CSS too), caching can be quite irritating. So if you disable the cache (while devtools are open), you'll save yourself from lots of pain 

1. Datatypes

Used to describe the various types of data we can store in variables. E.g. [Numbers](https://recap.kea-alt-del.dk/" \l "number), [Strings](https://recap.kea-alt-del.dk/" \l "strings), [Arrays](https://recap.kea-alt-del.dk/" \l "arrays), [Booleans](https://recap.kea-alt-del.dk/" \l "boolean)

1. Concatenation

The process of "gluing" variables together to form strings. The logic is; if you glue together two numbers, it becomes a new number. If you glue together anything with a string, it becomes a string. Plus (+) is used to concatenate

let a = 2;

let b = 3;

let c = "Hi";

let d = "4";

let e = a + b; //e is 5, addition

let f = d + d; //f is "44", concatenation

let g = a + b + c; //g is "5Hi", both

let h = a + "" + b; //h is "23"

1. Strings

Yet another [datatype](https://recap.kea-alt-del.dk/" \l "datatypes). A string is any combination of characters wrapped in ' or ", meaning that "42" is still a string. JavaScript does a thing called coercion for us, so that often we don't have to think too much about it, but there are several pitfalls. Example:

let a = "42"; //a string

let b = 8; //a number

let c = ""; //a string

let d = "10";

let e = 2;

a + b; // gives the string "428", since one is a string, it does concatenation instead

b + e + d; //gives "1010", it first does math (8+2) and the concatenation

1. Arrays

A [datatype](https://recap.kea-alt-del.dk/" \l "datatypes). A list of values, stored in one variable. Really important when we don't necessarily know the number of values needed.

//BAD!!

let student1 = "Jonas";

let student2 = "Alan";

//...

//GOOD

let students = ["Jonas", "Alan", "..."];

Accessing the elements is done using bracket notation and the "index" of the item (starting at 0)

let hobbies = ["Beer", "Barbecue", "JS"];

let firstHobby = hobbies[0];

let lastHobby = hobbies[2];

To make this really clever, we need a [loop](https://recap.kea-alt-del.dk/" \l "foreach)

### .length

A property on [arrays](https://recap.kea-alt-del.dk/" \l "arrays) & [strings](https://recap.kea-alt-del.dk/" \l "strings) telling us how many items are in it

// with an array

let beers = ["IPA", "APA", "stout"];

let x = beers.length; // x is 3

// or with a string

let name = "Jonas";

let numChars = name.length; // numChars is 5

1. callback

A name for a function that is passed on, to be called at a later state. Look at [clearInterval](https://recap.kea-alt-del.dk/" \l "clearinterval) where the sayHi function is the callback.

Callbacks are a core concept in JavaScript.

1. clearInterval

Clears a timer set with [setInterval()](https://recap.kea-alt-del.dk/" \l "setinterval)

let count = 0;

let intervalId = setInterval(sayHi, 5000);

function sayHi() {

count++;

console.log("Hi, 5 seconds have passed");

if (count === 5) {

clearInterval(intervalId);

}

}

1. .classList

A [property](https://recap.kea-alt-del.dk/" \l "property) on [nodes](https://recap.kea-alt-del.dk/" \l "node). Not useful on it's own, but with the following three [methods](https://recap.kea-alt-del.dk/" \l "method) it's extremely powerful.

### .add

Add a CSS-class to a [node](https://recap.kea-alt-del.dk/" \l "node). If the node already has the class, nothing happens.

Consider the following HTML

<div>

<h1 class="main-heading">Hi mom</h1>

</div>

And the JS

let header = document.querySelector("h1");

//let's add the highlight class to the <h1>

// possibly triggering awesome animations/transitions in the process

header.classList.add("highlight");

//and here, nothing happens, it already has the class

header.classList.add("main-heading");

### .remove

Remove a CSS-class from a [node](https://recap.kea-alt-del.dk/" \l "node). If the node does not have the class, nothing happens.

//chaining

document.querySelector("hi").classList.remove("main-heading");

// same as

let x = document.querySelector("h1");

x.classList.remove("main-heading");

### .toggle

Toggle a CSS-class on a [node](https://recap.kea-alt-del.dk/" \l "node). If the node already has the class, it's removed, otherwise it's added.

let x = document.querySelector("h1");

x.classList.toggle("active"); //now it's there

x.classList.toggle("active"); //now it's gone

1. method

Often used interchangeably with the term [function](https://recap.kea-alt-del.dk/" \l "functions), but there's a slight difference. A method is a function that lives on/in an object, and it's invoked using the dot operator. So .querySelector() is a method, that works on an element. .random() and it's brothers, are methods that work on the Math object.

1. Selector

In this case, a selector is a CSS selector used in JS. Methods such as [.querySelector](https://recap.kea-alt-del.dk/#querySelector) and [.querySelectorAll](https://recap.kea-alt-del.dk/#queryselectorall) use selectors to grab elements in the [DOM](https://recap.kea-alt-del.dk/#dom).

With the right selectors, we can grab almost anything, and getting good at the various CSS selectors will make our lives so much easier.

You can find [a great reference on w3schools](https://www.w3schools.com/cssref/css_selectors.asp) and an [awesome interactive tutorial at CSS Diner](http://flukeout.github.io/)

1. .querySelector

.querySelector is a method we can use to grab elements in the DOM, including inline SVG elements!. The method can be used on the [document](https://recap.kea-alt-del.dk/" \l "document) or on an already selected [node](https://recap.kea-alt-del.dk/" \l "node).

We use it like this: element.querySelector(cssSelector) The method returns the [node](https://recap.kea-alt-del.dk/" \l "node)

//Select the first <header> with the clss main in the DOM

let myHeader = document.querySelector("header.main");

// with an element selected, we can find that element's children

// find the first h1 inside myHeader

let h1 = myHeader.querySelector("h1");

1. .querySelectorAll

Works just like [.querySelector](https://recap.kea-alt-del.dk/" \l "queryselector), except it returns a [NodeList](https://recap.kea-alt-del.dk/" \l "nodelist).

// find all paragraphs in the document

let ps = document.querySelectorAll("p");

// or find all links in the header

let links = document.querySelectorAll("header a");

1. .removeEventListener

Used to remove an existing eventlistener (previously set with [addEventListener](https://recap.kea-alt-del.dk/" \l "addeventlistener)). The call to removeEventListener must have the exact same call back as the call to [.addEventListener](https://recap.kea-alt-del.dk/" \l "addeventlistener). This means that events registered with anonymous functions are really difficult to remove.

//first we register an eventlistener to an element

let terms = document.querySelector(".acceptTerms");

terms.addEventListener("click", termsClicked);

function termsClicked(e) {

alert("You have accepted our terms");

terms.removeEventListener("click", termsClicked);

}

For more, see [MDN](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/removeEventListener)

1. setInterval

Execute a callback every X millisecond

// setInterval(callback, interval)

function sayHi() {

console.log("Hi");

}

// call the function sayHi ten times per second

setInterval(sayHi, 100);

See [clearInterval](https://recap.kea-alt-del.dk/" \l "clearinterval) as well

1. setTimeout

Works just like [setInterval](https://recap.kea-alt-del.dk/" \l "setinterval) except, it only fires once.

// Execute the callback after 2 seconds

setTimeout(() => {

console.log("2 seconds passed");

}, 2000);

1. Number

A [Datatype](https://recap.kea-alt-del.dk/" \l "datatypes) for.... numbers. Represents both integers (3) and floating point numbers (3.14). If you have a string, you can convert it to a number with the Number "function":

let x = 7; // a number

let a = "42"; // a string

let asANumber = Number(a); // convert "a" to a number

1. State variables

State variables aren't really a JavaScript thing, it's a common concept in all programming languages. We declare a variable that has an initial state (value), and then we use that variable to decide what our program should do.

let counter = 0; //this is our state variable

document.querySelector("p").addEventListener("click", (e) => {

counter++;

if (counter > 10) {

console.log("You clicked the counter more than 10 times");

}

});

1. Node

A node is an element in the [DOM](https://recap.kea-alt-del.dk/#dom). It's what you get when you use [.querySelector](https://recap.kea-alt-del.dk/#queryselector). These nodes contain a lot more than meets the eye. Try selecting a node and do a console.dir on it:

let b = document.querySelector("body");

console.dir(b);

1. NodeList

A NodeList is a list of nodes, usually gotten via [.querySelectorAll](https://recap.kea-alt-del.dk/#queryselectorall). While not technically an [array](https://recap.kea-alt-del.dk/#arrays), it shares a lot of the same capabilities, it has a .length and we can loop through it.

1. Boolean

Booleans are a special [datatype](https://recap.kea-alt-del.dk/#datatypes) that can only contain the values true or false (no quotes). They are often used as "gates" / [state variables](https://recap.kea-alt-del.dk/#statevariables)

let myDiv = document.querySelector("div");

let hasBeenClicked = false;

myDiv.addEventListener("click", divClicked);

function divClicked(e) {

if (!hasBeenClicked) {

//false, the first time

console.log("Nice");

hasBeenClicked = true;

} else {

console.log("Stop clicking me");

}

}

1. Math

Math is an object containing lots of useful methods concerning math. Stuff like rounding, PI, randomness etc are located here, let's take a look at a few of them.

### .random()

Math.random() will return a random number between 0 (inclusive) and 1 (not inclusive). Using a little bit of math, we can generate a random number in any range:

let x = Math.random() \* 11; //x is now a number between 0 and 10.99999999.....

### .floor()

Math.floor() takes number as argument, rounds it down and returns it, making it an ideal partner for [Math.random()](https://recap.kea-alt-del.dk/#math)

let y = Math.floor(1.9); // y is now 1

//generating a random number between 0 & 10 (both inclusive)

let rand = Math.random(); // 0 - 0.99999....

rand = rand \* 11; // 0 - 10.99999999....

rand = Math.floor(rand);

//or in one line

1. .style

The .style property lives on [Nodes](https://recap.kea-alt-del.dk/#node), and allows us to **set** CSS properties. It does NOT allow us to read existing rules that were not set using .style.

CSS properties that have a dash (-) are "camelCased". So background-color becomes backgroundColor.

// select the second li in the ul with id="main"

let li = document.querySelector("ul#main li:nth-child(2)");

//set/overqrite it's color & background-color

li.style.color = "red";

li.style.backgroundColor = "blue";

The .style property is the most important type of CSS, meaning that it will always take precedence over CSS written in a stylesheet, inline etc. Don't overuse it.