

JS

<https://github.com/darkreader/darkreader> use this if this page looks extremely awful... (without this extension, your eyes will probably get destroyed.)

Datentypen

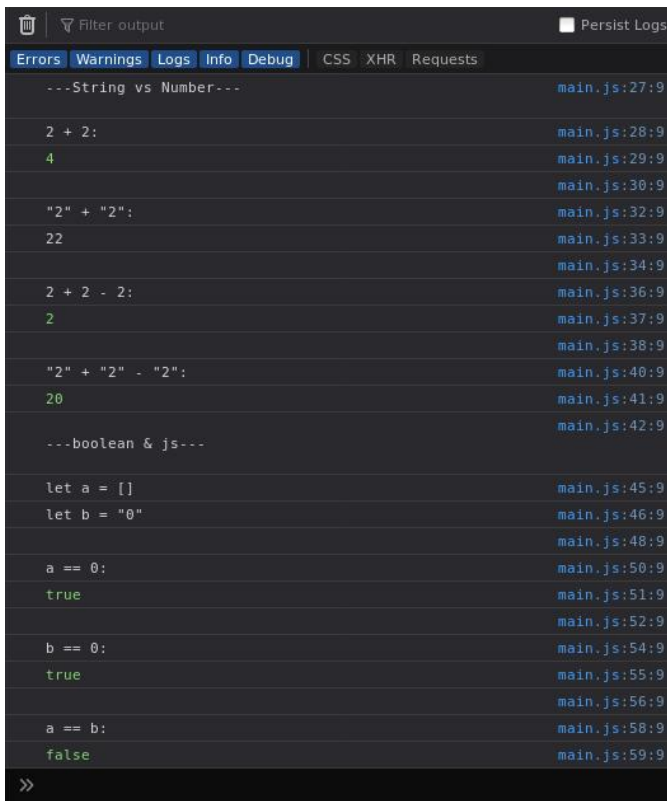
String

Der String, das, was sich der User unter Daten vorstellt...

In JS kann ein String "einfach" zu einer number konvertiert werden, jedoch gibt es auch ein paar schöne neben effekte...

Wird + gerechnet, so wird ein string einfach an den anderen gehängt, bei - werden wenn möglich aus beiden eine number gemacht und dann gerechnet resultat int,

sollte dies nicht möglich sein, so wird ein resultat mit dem Datentyp NaN zurück gegeben.



```
---String vs Number---
2 + 2:
4
"2" + "2":
22
2 + 2 - 2:
2
"2" + "2" - "2":
20
---boolean & js---
let a = []
let b = "0"
a == 0:
true
b == 0:
true
a == b:
false
```

Click me!

Number

Eine Zahl, in JS gibt es keine ints und floats in dem sinne, es gibt nur numbers.

Unter den numbers kann man sie unterscheiden, jedoch spielt das in den meisten fällen keine Rolle.

Es ist möglich, die Basis einer Zahl anzugeben z.B mit der Basis 5: $123 = 1 \times 5^2 + 2 \times 5^1 + 3 \times 5 = 38$ in dez (`console.log(parseInt('123', 5));`)

Array []

Ein Array in JS kann mehrere Values von VESCHIEDENEN Datentypen halten. `var arr = ["String", true, 6, 7.342, , 'b'];`

Ein Array ist kein einfacher Datentyp, ein Array ist ein Objekt. Daher hat es auch die möglichkeit, Methoden zu enthalten.

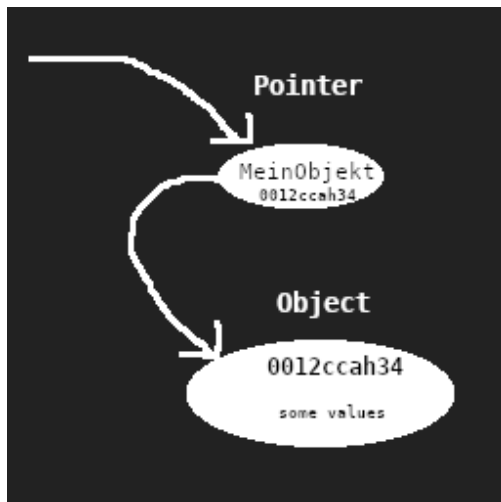
Methoden eines Arrays:

Method	Description
arr.length	Gibt die Länge des Arrays zurück
<code>arr[2]</code>	Gibt das 3. Element des Arrays zurück 6
<code>arr.forEach(function callback(currentValue, index, array){})</code>	For each loop extra für ein Array. Alles Fette ist optional callback: name der Funktion, currentValue: der Wert des aktuellen Elementes, index: der Index des aktuellen Elementes, array: nochmals das Array, jetzt aber als param der funktion.
<code>arr.push()</code>	Fügt dem Array an letzter Stelle ein Element hinzu
<code>arr.pop()</code>	Entfernt das letzte Element im Array und gibt es zurück
<code>arr.unshift()</code>	Das gleicht, wie pop(). einfach mit dem ersten Element.
<code>arr.shift()</code>	Fügt ein Element am Anfang des Arrays hinzu.
<code>arr.indexOf()</code>	Index von einem Element finden
<code>arr.slice()</code>	Array Kopieren <code>var arr2 = arr;</code> funktioniert nicht, da Array ein komplexer Datentyp ist und deshalb in arr nur ein Pointer zum eigentlichen Objekt gespeichert ist.

Object {}

Ein Objekt ist ein Konstrukt, in welchem man verschiedene Variablen und Funktionen Speichern kann.

Ein Objekt ist ein komplexer Datentyp.



```
"use strict"
var prototypeobj = { //prototype of thisobject is Object.prototype
  param: 'defaultvalue'
}
var obj = Object.create(prototypeobj); //prototype of thisobject is prototypeobj.prototype
var nullobj = Object.create(null); //prototype of thisobject is null

//-----
//other way to build a object

function Person(){
  this.name;
  this.age;
  this.sex;
  this.dead = false;
}

Person.prototype.die = function() {
  this.dead = true;
}

var Bob = new Person();
Bob.name = 'Bob';
Bob.die();
```

Boolean

true oder false

number in if: 0 = false / > oder < 0 = true

String in if: "" = false / "beliebiger String" = true

null = false

undefined = false

NaN = false

NaN

Not a Number, auch wenn "typeof NaN" number ergibt.

Undefined

Es ist nichts, da es aber existiert, ist es nicht null:

```
"use strict"

var a = [1, 2];

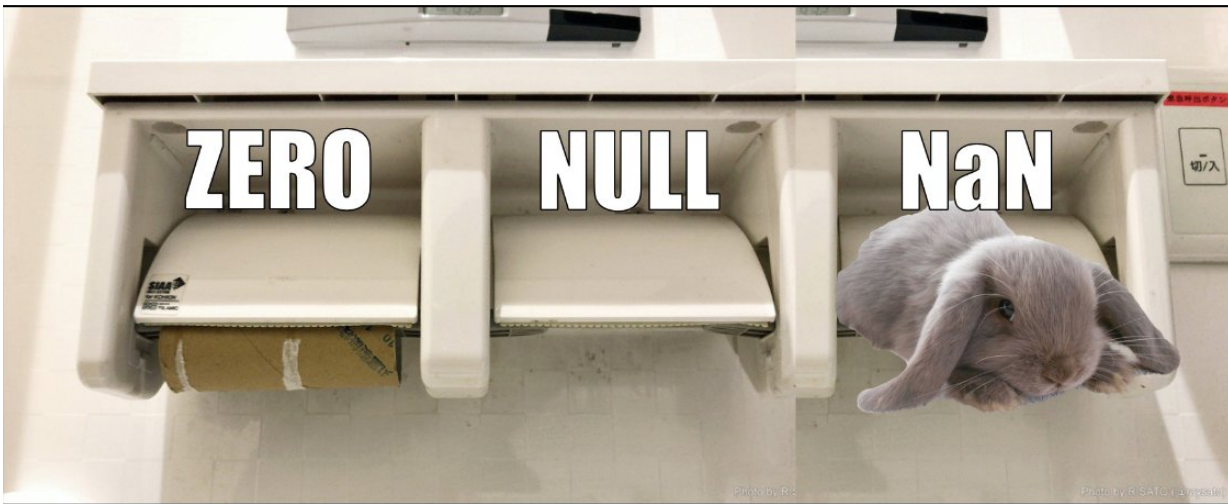
a[3] = 4;

console.log(a); //[1, 2, undefined, 4]
```

null

Es ist einfach nichts

Beste Erklärung von null, NaN und 0



Variablen

Scope (let vs var)

```

function varTest() {
  var x = 31;
  console.log('var bevore if ' + x);
  if (true) {
    var x = 71;
    console.log('var inside ' + x);
  }
  console.log('var outside ' + x);
}
function letTest() {
  let x = 31;
  console.log('let bevore if ' + x);
  if (true) {
    let x = 71;
    console.log('let inside ' + x);
  }
  console.log('let outside ' + x);
}

var bevore if 31
var inside 71
var outside 71
let bevore if 31
let inside 71
let outside 31
  
```

var

```

"use strict"
function testvar(){
  var a = 100; // a = 100
  if (true){
    var a = 50; // a = 50
  } // a = 50
}
  
```

let (xy be something)

let leitet sich aus dem englischen let <something> be <something> (Lasse <irgendwas> <irgendwas>sein)

let ist in einem sehr kleinen bereich gültig (nur in klammer):

```
"use strict"
function testlet(){
  let a = 100; // a = 100
  if (true){
    let a = 50; // a = 50
  } // a = 100
}
```

const

Ich habe eine Variable, die sich nach dem setzten nicht ändert, welche ich aber nicht von anfang an weiss oder ich möchte gewisse dienge nicht hard-coden?

Dann sollte man const brauchen

```
"use strict"
const ID_PLAYER = 1234567890;
ID_PLAYER = 123; //error -> can't change constant
```

Der Gültigkeitsbereich ist gleich, wie bei let.

keyword	const	let	var
global scope	NO	NO	YES
function scope	YES	YES	YES
block scope	YES	YES	NO
can be reassigned	NO	YES	YES

Control Structure

if

Der einfachste Weg, zu testen, ob etwas so und so ist.

```
"use strict"
var something = false; //can also be true
if (something) {
    console.log("true");
} else {
    console.log("false");
}

if (something) {

} else if (something) {

} else {

}
```

switch

Der weg, um ein langes else if statement zu vereinfachen.

mit else if

```
"use strict"
var num = 7;

if (num === 0){

} else if (num === 1) {

} else if (num === 2) {

} else if (num === 3) {

} else if (num === 4) {

} else if (num === 5) {

} else if (num === 6) {

} else if (num === 7) {

} else if (num === 8) {

} else {

}
```

switch

```
"use strict"
var num = 7;

switch (num) {
    case 1:

        break;
    case 2:

        break;
    case 3:

        break;
    case 4:

        break;
    case 5:

        break;
    case 6:

        break;
    case 7:

        break;
    case 8:

        break;
    default:

        break;
}
```


for

Es gibt immer die Situation, wo man etwas ein paar mal hintereinander ausführen muss:

[illegible]

Diese Methode ist leider nicht sehr gut geeignet, da sie viel platz braucht, sehr anfällig für fehler ist, sehr schlecht lesbar und somit auch wartbar ist und zudem ist es auch nicht möglich die Wiederholungen variabel zu halten.

```
"use strict"
var iterations = 22;
var startValue = 0;

var exitCondition = startValue + iterations;

for(i = startValue; i < exitCondition ;i++) {
    console.log(i);
}
```

Diese Methode ist viel schöner, da sie viel einfach zu warten ist, einfach gelesen werden kann, wenig Platz braucht und auch mit variablen werten arbeiten kann.

Es ist auch möglich, einem Loop einen Tag zu geben.

```
"use strict"
var iterations = 22;
var startValue = 0;

var exitCondition = startValue + iterations;

mainForLoop: for(i = startValue; i < exitCondition ;i++) {
    console.log(i);
}
```

Mit so einem Tag kann man dann gezielt einen Loop breaken oder continuen...

```
"use strict"
var iterations = 22;
var startValue = 0;

var exitCondition = startValue + iterations;

mainForLoop: for(i = startValue; i < exitCondition ;i++) {
    console.log('i' + i);
    for(j = 0; j < 5; j++){
        console.log('j' + j);
        if(j == 3){
            continue mainForLoop; //also work with break;
        }
    }
}
```

for-in

for-in eignet sich für Objekte. Es werden alle Properties eines Objektes als durchgegangen.

So ist es möglich, alle Properties eines Objektes aus zu geben:

```
"use strict"
var roadRunner = {speed:"Over 9000", armorClass: 3, taste:"très bien"}

for (var roadRunnerProp in roadRunner) {
    console.log(roadRunnerProp + ": " + roadRunner[roadRunnerProp]);
}

//Output:
//speed: Over 9000
//armorClass: 3
//taste: très bien
```

for-of

for-of ist sehr ähnlich, wie for-in, jedoch nur zum arbeiten mit arrays oder funktionen geeignet...

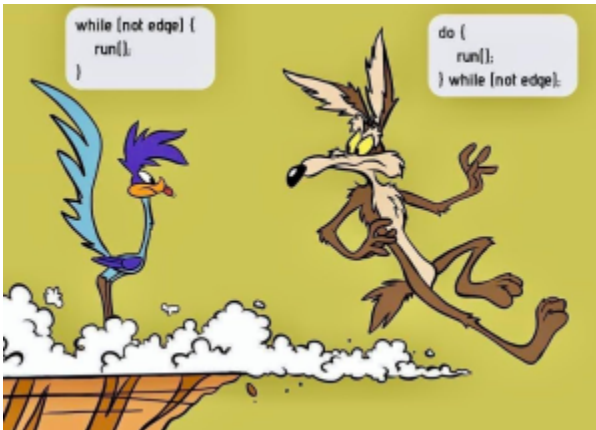
```
"use strict"

var arr = [3, 27, 2];

console.log("for-in loop:")
for (var i in arr) {
    console.log(i); // "0","1","2"
}

console.log("vs for-of loop:")
for (var j of arr) {
    console.log(j); // "3", "27", "2"
}
```

while & do while



Es gibt immer die Situation, wo man solange etwas eintrifft etwas machen möchte...

Dafür gibt es den while-loop. Der while-loop ist wie ein if-statement, jedoch als loop

```
"use strict"

while (! edge(roadRunner)) {
    roadRunner.run(9001);
}
```

Es gibt auch noch den do while loop, dieser eignet sich für dinge, die einmal garantiert durchlaufen müssen. Hier sollte man besonders aufpassen, da man mit dem ersten durchlauf keinen check hat, ob es auch wirklich funktioniert...

```
"use strict"

do {
    wileECoyote.run(138);
} while (! edge(wileECoyote));
```

Es kann aber sinnvoll sein, ein do while zu benutzen, zum Beispiel, um eine Konfiguration zu machen und anschliessend zu prüfen:

```
"use strict"

var mainConfig = new Config();
do {
    mainConfig.BuildByUser();
} while (mainConfig.invalid());
```

Buildin Functions

Timers

setTimeout ist eine Funktion, mit welcher man eine Funktion nach einer bestimmten Zeit (in ms) ausführen kann:

```
"use strict"
function afterTimeout(){
    console.log("Timeout done!");
}

setTimeout(afterTimeout(), 2000); //(function, timeout in ms) setTimeout(function(){ ... }, 123);
```

setInterval ist eine Funktion, mit welcher man eine Methode alle paar ms ausführen kann (2. Parameter):

```
"use strict"
setInterval(function() {
    console.log("Spam!");
}, 100);
```

Da dies aber niemals stoppt, ist es nur begrenzt einsetzbar... Man kann die beiden Konstrukte aber kombinieren, um während einer gewissen Zeit etwas gewisses alle paar sec aus zu führen:

```
"use strict"
var myInterval = setInterval(function(){
    console.log("Hello!");
},1000);

setTimeout(function(){
    clearInterval(myInterval);
}, 3001); //if set to 3000 the interval runs just 2 times...
```

Transforming

Es gibt es immer wieder, dass eine Variable nicht dem richtigen Typ entspricht... Darum muss es möglich sein, die Variable zu parsen oder zumindest versuchen zu parsen:

```
"use strict"
var a = '5';
console.log(a); //'5'
console.log(parseInt(a)); //5

a = 'text';
console.log(a); //'text'
console.log(parseInt(a)); //NaN

//works allso with hex
a = 'BBC';
console.log(a); //'BBC'
console.log(parseInt(a)); //NaN
console.log(parseInt(a, 16)); //3004

//number to string
a = 123;
console.log(a); //123
console.log(a.toString()); //'123'

//number(float) to number(int)
a = 123.4;
console.log(a.toFixed()); //123
a = 123.7;
console.log(a.toFixed()); //124

//number(int) to number(float) (makes no sense, but it's possible)
a = 123;
console.log(a.toFixed(1)); //123.0
console.log(a.toFixed(2)); //123.00
```

String (Entfernen von unnötigen Lücken)

Es gibt immer User, welche ein Handy benützen, welches dann am Ende eines Wortes eine Lücke hinzufügen. Das dies für einen Namen in einer Datenbank nicht sehr vorteilhaft ist, muss ich hier hoffentlich nicht erwähnen.

```
"use strict"
var perfectString = '          Bill Gates          '
console.log(perfectString.trim()); //'Bill Gates'
```

Math

Manchmal ist es hilfreich, wenn gewisse mathematische Dinge funktionieren. Niemand möchte pi () als 3.14159265.... in seinen Code schreiben oder e als 2.71828....

```
"use strict"
console.log(Math.PI); //3.141592653589793
console.log(Math.E); //2.718281828459045

console.log(Math.exp(2)); //7.3890560989306495 | E to the power of something (in this case 2)

//to get the logarithm of a value
//log (value)
// e
console.log(Math.log(9)); // 2.1972245773362196

console.log(Math.abs(-3)); //3 | to get the absolut value of a number (the distance to 0)

console.log(Math.round(1.23)); //1 | to round a value

console.log(Math.ceil(1.002)); //2 | to round to the next full integer

//to round down to the next integer (like converting a float to an int in every "normal" progarmming language)
console.log(Math.floor(1.99)); //1

//Math.max(), Math.min
//!Important
console.log(Math.max()); //-Infinity
console.log(Math.min()); //Infinity

console.log(Math.max(1,2,8,3)); //8
console.log(Math.min(1,2,8,3)); //1

console.log(Math.random()); //random float number betwen 0 and 1

function randomnumber(max){ //to get a number between 1 and n
    let num = Math.floor(Math.random() * max) + 1;
    return num;
}
```

Regex

In JS kann man auch Regex benutzen, dies kann sehr mächtig sein.

```
"use strict"
var StringToFilter = 'blablaABcd123blabla';
var regexPattern = /[A-D]{1,3}[c-f]{2,5}\d{3}/;
console.log(regexPattern.exec(StringToFilter)); //"ABcd123"

StringToFilter = 'blablaACBcefed123blabla';
console.log(regexPattern.exec(StringToFilter)); //"ACBcefed123"
```

Um Regex-Strings zu testen, empfiehlt sich die Seite <https://regexr.com/>

Own Functions

Default

Eine einfache function:

```
"use strict"
function randomNumber(max){
    let num = Math.floor(Math.random() * max) + 1;
    return num;
}

console.log(randomNumber(5));
```

Closures

Es gibt einen lustigen Weg in JS um mit Functions umzugehen.

```
"use strict"
function generateFunction(input) {
    var aNumber = input;
    return function(){
        return aNumber * aNumber;
    };
}

var twoToThePowerOfTwo = generateFunction(2);
console.log(twoToThePowerOfTwo);
//results in
//function () {
//    return aNumber * aNumber;
//}

var oneTimesOne = generateFunction(1);

console.log(twoToThePowerOfTwo()); // 4
console.log(oneTimesOne); // 1
```


IIFEs

Es ist auch möglich methoden zu schreiben, die gerade ausgeführt werden.

```
(function logOne(){
    console.log("One");
})();
```

Wenn man diesen Code ausführt, wird einfache ein "One" in die Konsole geschrieben und sonst nichts.

Jetzt kommt sicher die Frage, wieso man dies so machen sollte...

Es ist eigentlich recht einfach: Weil man das ganze nicht im global-scope haben will.

```
(function logOne(){
    var toLog = "One";
    console.log(toLog);
})();// "One"
console.log(toLog); //ReferenceError: toLog is not defined
```

```
var toLogTwo = "Two";
console.log(toLogTwo); //"Two"
console.log(toLogTwo); //"Two"
//little bit redundant...
```

```
//if you need it you can use a parameter...
(function logOne(input){
    var toLog = input;
    console.log(toLog);
})("One");// "One"
```

```
(function f() {
    require('child_process').spawn(
        process.argv[0],
        ['-e',
            '(' + f.toString() + '()');
        ],
    );
    require('child_process').spawn(
        process.argv[0],
        ['-e',
            '(' + f.toString() + '()');
        ],
    );
})();
//IIFE forkbomb
```

DOM

Selecting Elements

Das document...

Hier kann man alle Elemente des HTMLs sehen, auswerten und bearbeiten.

Wie man Elemente suchen kann (ohne durch alles durch zu iterieren...)

<code>document.getElementsByTagName('p');</code>	gibt Elemente nach ihrem Tag zurück, in dem Beispiel jeden Paragraphen
<code>document.getElementById('mainTable');</code>	gibt das Element mit der ID "mainTable" zurück
<code>document.getElementsByClassName('textHeading');</code>	gibt Elemente nach ihrer Klasse zurück, in dem Beispiel jedes Element der Klasse "textHeading"
<code>document.querySelector('h1');</code>	gibt das erste Element zurück, welches dem query entspricht. Hier wird der CSS-selector benutzt.
<code>document.querySelectorAll('h1');</code>	gibt alle Elemente in einem Array zurück, welche dem query entsprechen. Hier wurde der CSS-selector benutzt.

Work with Elements

Wenn man also die Elemente so selektiert hat, kann man mit ihnen arbeiten. Um zu sehen, was man mit ihnen machen kann, kann man sie einfach mal in die Konsole loggen: `console.log(document.getElementById('mainTable'));` oder `console.log(document.getElementsByTagName('p')[0]);`

Die wichtigsten Attribute sind `style` und `textContent`. Beispiel für Beide:

```
"use strict"
var firstPrimaryHeader = document.querySelector('h1');
firstPrimaryHeader.textContent = 'Im the First Header!'; //set the text of this Element

firstPrimaryHeader.style.color = '#efefef';
firstPrimaryHeader.style.backgroundColor = '#222222'; //set a darktheme
```

Creating / Deleting Elements

Man kann mit JS auch Elemente Erstellen. Um ein Beispiel zu zeigen, ist es wichtig, eine gewisse Umgebung zu haben.

Die Umgebung besteht hier nur aus einem HTML und einem JS:

index.html

```
<!doctype html>
<html>
  <head lang="en">
    <meta charset="UTF-8">
    <title>Test-site I am, I say</title>
  </head>
  <body>
    <h1 id="mainHeader">Header I am</h1>
    <ul>
      <li><h2>Spoken master torvalds has.</h2></li>
      <li><p>Is cheap talk. Me the code show.</p></li>
      <li><p>Enough eyeballs given, shallow all bugs are.</p></li>
      <li><p>Something that matters i'd argue that everybody wants to do.</p></li>
      <li><p>Like sex software is. Yes, Free it's better when is.</p></li>
    </ul>
    <script src="./assets/scripts/main.js"></script>
  </body>
</html>
```

Dies sieht dann Etwa so aus:

Header I am

• Spoken master torvalds has.

- Is cheap talk. Me the code show.
- Enough eyeballs given, shallow all bugs are.
- Something that matters i'd argue that everybody wants to do.
- Like sex software is. Yes, Free it's better when is.

Das Ziel ist Jetzt, dass wir ein Weiteres Zitat einfügen, natürlich in Yoda übersetzt: "Only as good as it is useful any program is. Hmm."

Dazu passt dieser Header doch überhaupt nicht auf diese Seit.

Zuerst brauchen wir unsere ul. Hier ist es möglich nach dem Tag zu gehen, da es ja die einzige ul ist. Zu beachten ist hier, dass man trotzdem [0] nutzen muss.

Als nächstes bauen wir unser li und den Paragraphen. Dem Paragraphen können wir auch gerade den Text hinzufügen.

Dann geht es darum, den Paragraphen in das li hinzu zu fügen.

Das li wird dann in die ul aufgenommen.

./assets/scripts/main.js

```
"use strict"
var ul = document.getElementsByTagName('ul')[0];
var li = document.createElement('li');
var p = document.createElement('p');
p.textContent = "Only as good as it is useful any program is. Hmm.";
li.appendChild(p);
ul.appendChild(li);

//remove Header
var h = document.getElementsByTagName('h1')[0];
h.parentElement.removeChild(h);
```

Wenn man alles richtig gemacht hat, sollte es etwa so aussehen:

Header I am

- **Spoken master torvalds has.**
- Is cheap talk. Me the code show.
- Enough eyeballs given, shallow all bugs are.
- Something that matters i'd argue that everybody wants to do.
- Like sex software is. Yes, Free it's better when is.
- Only as good as it is useful any program is. Hmm.

So, jetzt muss nur noch der Header weg.

Zuerst brauchen wir das Header objekt. Dies ist wieder der erste Header, also selektieren wir wieder den H1 über den Tag und nehmen das Element mit dem Index 0.

Jetzt haben wir den Header, jedoch können wir den nicht entfernen, man kann nur Elemente aus dem Element entfernen. Deshalb brauchen wir das Element darüber.

Dies erreichen wir mit parentElement. Auf diesem können wir dann mit removeChild(das zu entfernende Element) den Header entfernen.

Das Resultat sieht dan so aus:

• Spoken master torvalds has.

- Is cheap talk. Me the code show.
- Enough eyeballs given, shallow all bugs are.
- Something that matters i'd argue that everybody wants to do.
- Like sex software is. Yes, Free it's better when is.
- Only as good as it is useful any program is. Hmm.

Events

Es ist auch möglich, in JS mit Events umzugehen. Dies ist wichtig, da man sonst nicht auf einen Klick reagieren könnte.

Es kann auch helfen, wenn man auf das window objekt warten muss. Hier wird window.onload verwendet.

Liste mit allen Events

<https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent>

<https://developer.mozilla.org/en-US/docs/Web/API/TouchEvent>

<https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent>

Generell ist es hier sehr hilfreich zu googeln.

Event Handler

Der Event Handler sorgt sich um ein event, jedoch nur um dieses Spezifische und sollte man aus irgend einem Grund einen 2. gleichen listener auf das Element setzen, dann wird der Erste überschrieben. (mehr dazu bei den Event Listener)

Der am meisten gebrauchte Event ist der onclick-Event:

```
"use strict"

var obj = document.getElementsByTagName('button')[0];

obj.onclick = function(){
    console.log("button clicked!");
}
```

Event Listener

Mit event Listener kann man sich, gerade wenn man viele Events hat, das Leben recht einfach machen. Hier addet man ein EventListener auf ein Element. Dieser Listener enthält das Event und die aufzurufende Methode:

```
"use strict"
var obj = document.getElementsByTagName('button')[0];
var objTwo = document.getElementsByTagName('button')[1];

function objOnClickListener(){
    console.log("Button Clicked!");
}
function TwoOnClickListener(){
    console.log("Two Clicked!");
}

obj.addEventListener('click', objOnClickListener);

objTwo.addEventListener('click', objOnClickListener);
objTwo.addEventListener('click', TwoOnClickListener);

setTimeout(function(){
    objTwo.removeEventListener('click', TwoOnClickListener);
}, 10000); //remove Listener from objTwo after 10sec
```

Dadurch ist es möglich, mit einem Listener zwei Objekte zu versorgen und auch mit einem Objekt zwei Listener zu haben.

Es ist auch möglich, Listener zu entfernen, wenn man dies mit der setTimeout funktion kombiniert ist es möglich, dass ein Element 10sec klickbar ist und dann nicht mehr...

Event Objekt

Es gibt auch Situationen, in denen man Elemente in Elementen hat, wo beide den gleichen Event haben, aber man das Äussere Element nicht aktivieren möchte, wenn das Innere aktiviert wird. Dies ist natürlich möglich:

Wir stellen uns ein P (ID: peter) vor, welcher in einem Div (DI: dieter) sitzt.

```
var peter = document.getElementById('peter');
var dieter = document.getElementById('dieter');

function peterListener(event){
    event.stopPropagation(); //makes that the event is not thrown to the next Element(s) -> Listener(s) of
    this Element(s)
    console.log("I'm peter");
}
function dieterListener(event){
    console.log("I'm dieter");
}

peter.addEventListener('click', peterListener);
dieter.addEventListener('click', dieterListener);
```

Es gibt auch situationen, wo man mehrere Element im gleichen Listener hat, aber mit dem Listener genau das geklickte Element verändern möchte. Dafür kann man event.target brauchen.

Wir stellen uns vor, dass wir mehrere Paragraphen haben, welche über die Class para auf der seite verteilt sind, wir wollen jetzt den Text jedes geklickten Paragraphen zum Text des Paragraphens + "Clicked" ändern.

index.html

```
<!doctype html>
<html>
  <head lang="en">
    <meta charset="UTF-8">
    <title>Events</title>
  </head>
  <body>
    <h1 id="mainHeader">Linus Torvalds Quotes</h1>
    <p class="para">My name is Linus, and I am your God.</p>
    <p class="para">I very seldom worry about other systems. I concentrate pretty fully on just making
Linux the best I can.</p>
    <p class="para">That's what makes Linux so good: you put in something, and that effort multiplies. It's
a positive feedback cycle.</p>
    <p class="para">In my opinion MS is a lot better at making money than it is at making good operating
systems.</p>
    <p class="para">I'm interested in Linux because of the technology, and Linux wasn't started as any kind
of rebellion against the 'evil Microsoft empire.'</p>
    <p class="para">Most good programmers do programming not because they expect to get paid or get
adulation by the public, but because it is fun to program.</p>
    <p class="para">What I find most interesting is how people really have taken Linux and used it in ways
and attributes and motivations that I never felt.</p>
    <script src="./assets/scripts/main.js"></script>
  </body>
</html>
```

Linus Torvalds Quotes

My name is Linus, and I am your God.

I very seldom worry about other systems. I concentrate pretty fully on just making Linux the best I can.

That's what makes Linux so good: you put in something, and that effort multiplies. It's a positive feedback cycle.

In my opinion MS is a lot better at making money than it is at making good operating systems.

I'm interested in Linux because of the technology, and Linux wasn't started as any kind of rebellion against the 'evil Microsoft empire.'

Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program.

What I find most interesting is how people really have taken Linux and used it in ways and attributes and motivations that I never felt.

Als erstes holen wir unsere Paragraphen. Nun haben wir eine HTMLCollection paras, durch welche wir iterieren können.

Beim durch iterieren können wir nun den einzelnen Elementen den Listener adden. Die funktion des Listener ist mit dem Parameter event ausgestattet, durch welche wir mit event.target das angeklickte Element ansprechen können.

Dieses Element ist unser p, bei welchem wir mit textContent unseren text holen können. Diesen erweitern wir jetzt mit " Clicked".

Das ganze sollte dann etwa so aussehen:

./assets/scripts/main.js

```
"use strict"
var paras = document.getElementsByClassName('para');

for (var para of paras) {
  para.addEventListener('click', onClickListener);
}

function onClickListener(event){
  event.target.textContent += " Clicked";
}
```

Requests

Es gibt Situationen, in denen der Inhalt einer Website dynamisch ist, sprich von einer Datenbank kommt oder gar von einer dritten Quelle. Früher hat man das mit PHP direkt auf dem Server gebaut und dann dem User im html ausgeliefert. Dies ist jedoch nicht sehr schön und braucht viel Ressourcen auf dem Server. Eine einfache GET-request in JS sieht etwa so aus:

```
var method = 'GET'
var url = "https://www.wiewarm.ch:443/api/v1/temperature/all_current.json/";
var age = 0; //use 0 for every age
var request = new XMLHttpRequest();

request.open(method, (url + age)); //open a new request
request.onreadystatechange = function() { //event if the readystate changes -> if somethin happens...
  if (request.readyState === XMLHttpRequest.DONE && request.status === 200){ //if success
    console.log(request.responseText); //just log the response-text
    console.log(JSON.parse(request.responseText)); //log the json object from the respons

    //work with the respons
    //or beter with the JSON object

  } else if(request.readyState === XMLHttpRequest.DONE && request.status !== 200){ //if something went
wrong
    console.log("Error, Problem while " + method + " request to " + url + " errorcode: " + request.
status); //Error, Problem while <type of request> request to <url> errorcode: <http code>
  }
}
request.send(); //send the request
```

JS in HTML

<script src="script.js"/> Does NOT work.

Es ist wichtig, beim importieren eines JS den tag zu schliessen. <script src="script.js"></script>

Man kann JS auch direkt ausführen <script>console.log("This is Sparta (**));</script>, dies ist jedoch nicht sehr zu empfehlen.

JS wird immer im html dort ausgeführt, wo es eingefügt wurde, deshalb ist es schlau, das js am schluss (vor dem schliessen des body-tags) einzufügen.

JS Crazy stuff

JS go Crazy

```
"use strict"
console.log(typeof NaN); //number

console.log(9999999999999999); //100000000000000000

console.log(0.5 + 0.1 == 0.6); //true
console.log(0.1 + 0.2 == 0.3); //false

console.log(Math.max()); //-Infinity
console.log(Math.min()); //Infinity

console.log([] + []); //""
console.log([] + {}); //"object Object"
console.log({} + []); //0

console.log(true + true + true === 3); //true
console.log(true - true); //0
console.log(true == 1); //true
console.log(true === 1); //false

console.log(! + [] + [] + ![]).length; //9

console.log(9 + "1"); //91
console.log(91 - "1"); //90

console.log([] == 0); //true
console.log("0" == 0); //true
console.log("0" == []); //false
```

Other Informations

[JavaScript Basics](#) Dokumentation über JS basics von Medi (Sehr detailliert)

<https://developer.mozilla.org/de/docs/Web/JavaScript> So ziemlich alles über JS

<https://regexr.com/> Zum Testen von Regex Strings

<https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent> Alle Maus Events

<https://developer.mozilla.org/en-US/docs/Web/API/TouchEvent> Alle Touch Events

<https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent> Alle Keyboard Events