# Wichtige JavaScript Konzepte aus dem Udemy Kurs

## Variablen und Funktionen

* Variablen sinnvollerweise mit Schlüsselwort „let“ deklarieren. Das Schlüsselwort kann z.B. in einer Schleife genutzt werden um einer Variablen temporär einen neuen Wert zuzuweisen

let a = 5;

console.log(a); // Ausgabe: 5

for(i=0;i<5;i++) {

let a = 6;

console.log(a); // Ausgabe: 6

}

console.log(a); // Ausgabe: 5

* Unterschied zwischen „let“ und „var“. Let erstellt Variable auf BLOCKebene, var erstellt Variable auf FUNKTIONSebene

//Gibt 1, 2, 3 aus, da für jeden Schleifendurchlauf eine neue Variable i erzeugt wird (der alte Wert wird übernommen)

for(let i=0;i < 3; i++) {

setTimeout(() => {

console.log(i);

},1000);

}

// Gibt 3, 3, 3 aus, da j auf Funktionsebene erstellt wird, d.h. es existiert bereits vor Beginn der Schleife.

// Die Logausgabe im Timeout nutzt also die "außerhalb" der Schleife deklarierte Variable j die nach dem Schleifendurchlauf

// den Wert 3 besitzt

for(var j=0;j < 3; j++) {

setTimeout(() => {

console.log(j);

},1000);

}

* „use strict“ um auf Fehler oder schlechten Programmierstil hingewiesen zu werden z.B. wenn eine Variable nicht deklariert wurde (ohne LET)

"use strict";

a=5;

console.log(a); // Wirft Fehlermeldung, dass a nicht deklariert wurde

* *Anonyme Funktionen durch einfache Zuweisung*

let myFunc = function() {

console.log(5);

}

myFunc(); // Ausgabe: 5

* *Funktion in Funktion übergeben*

let run = function(callback) {

console.log("run() wurde ausgeführt")

console.log(callback());

}

let f = function() {

return "hallo welt";

}

run(f);

// Oder

run(function() { return "hallo welt"; });

* *Event Loop: Diese kann nicht einfach unterbrochen werden durch ein Sleep, daher müssen Funktionen verkettet werden z.B. Timer mit einer zweiten Funktion verketten, damit diese nach x Sekunden ausgeführt wird*

let f2 = function() {

console.log("hallo welt");

}

setTimeout(f2, 5000); // wird nach 5 Sek ausgeführt

console.log("fertig"); // wird SOFORT ausgegeben, da setTimeout asynchron ausgeführt wird

*(!) Sollte dennoch eine lange blockierende Funktion stattfinden, so wird die Ausführung von setTimeout(…) so lange blockiert, bis die ressourcenhungrige Funktion beendet ist.*

* *Asynchrone Funktionen*

let f2 = function() {

console.log("hallo welt");

}

setTimeout(f2, 5000); // wird nach 5 Sek ausgeführt

setInterval(f2, 2000); // wird ALLE 2 Sek ausgeführt

// identisch mit d.h. die Funktionen werden im window Kontext ausgeführt

window.setTimeout(f2, 5000);

window.setInterval(f2, 2000);

// setInterval und setTimeout geben einen Rückgabewert mit einer eindeutigen ID zurück ...

let interval = window.setInterval(f2, 2000);

let timer = window.setTimeout(f2, 5000);

// ... mit der das Interval bzw. der Timeout beendet/abgebrochen werden kann

clearInterval(interval);

clearTimeout(timer);

* *Eigenheit bei Vergleichen mit NaN (Not a number)*

let a = NaN;

//Ergibt false

if(a==NaN)

//Ergibt false

if(a==a);

//Ergibt true

if(isNaN(a))

## Zugriff auf den HTML DOM und Eventlistener

* *Event Listener auf Button im DOM*

<html>

<head>

<script type="text/javascript" src="index.js"></script>

</head>

<body>

<button id="click-me">Klick mich</button>

</body>

</html>

// Warten bis Fenster geladen ist (sonst ist der Button noch nicht da)

window.addEventListener("load", function() {

// Bei Klick auf Button wird Funktion ausgeführt

document.getElementById("click-me").

addEventListener("click",

function() {

console.log("Button clicked!");

},

false);

});

* *Wichtige DOM Elemente und Methoden zum Finden von Objekten*

// Enthält Informationen über das zugrundeliegende HTML Dokument (welche Elemente, Adresse, etc)

document

// Enthält Informationen über den Client (z.B. Browserart und Version, aber auch Funktionen wie z.B. setTimeout oder setInterval, …)

window

<html>

<head>

<script type="text/javascript" src="index.js"></script>

</head>

<body>

<h3 id="heading">Test</h3>

<button class=“ id="click-me">Klick mich</button>

</body>

</html>

"use strict";

window.addEventListener("load", function() {

// Holt Überschrift mit der Id "heading" ab

let heading = this.document.getElementById("heading");

heading.innerText = "Miau";

// Holt alle Element ab, die der Klasse "my-button" angehören

let button = this.document.getElementsByClassName("my-button");

this.console.log(button.innerText);

// Holt alle H3-Überschriften ab

let h3s = this.document.getElementsByTagName("h3");

this.console.log(h3s);

});

* *Styling verändern*

<html>

<head>

<script type="text/javascript" src="index.js"></script>

<style>

.standard {

background-color: aquamarine;

}

.special {

background-color: chocolate;

}

</style>

</head>

<body>

<h3 class="standard">My heading</h3>

<button id="that-button" >Klick mich</button>

</body>

</html>

"use strict";

window.addEventListener("load", function() {

let button = this.document.getElementById("that-button");

let heading = this.document.getElementById("that-heading");

button.addEventListener("click", function() {

heading.innerText = "Hey!";

//Direktes Styling

heading.style.border = "3px solid";

//Entferne Klasse "standard" und füge "special" hinzu

heading.classList.remove("standard");

heading.classList.add("special");

});

});

* *CSS Selektoren*

<html>

<head>

<meta charset="UTF-8" />

<style>

.box\_\_content\_\_class {

background: #FF0154;

}

</style>

    </head>

<body>

<div id="box" moep="wahnsinn">

<div id="box\_\_content" class="special box\_\_content\_\_class">

Und das hier, ist der Inhalt

</div>

</div>

<script type="text/javascript" src="index.js"></script>

</body>

</html>

"use strict";

window.addEventListener("load", function() {

//Suche ein Element mit der gewünschten Attribut-Wert Kombination

let myElement = document.querySelector("[moep='wahnsinn']");

// …oder alle laden

let myElements = document.querySelectorAll("[moep='wahnsinn']");

}, false);

* *Elemente in den DOM einfügen*

window.addEventListener("load", function() {

let newButton = document.createElement("button");

newButton.innerText ="Click me!";

myElement.appendChild(newButton);

}, false);

## Arrays

* *Arrays anlegen, befüllen, leeren und auslesen*

// Lege Array mit Personen an

let personen = ["paula", "erik", "julia", "max"];

// Da nicht streng typisiert --> gemischte Arrays sind möglich

let gemischt = ["toller string", 215];

// Füge an erster Stelle ein neues Element hinzu

personen.unshift("noah");

// Füge an letzter Stelle ein neues Element hinzu

personen.push("marius");

//Entferne erstes Element und speichere es in Variable

let erstesElement = personen.unshift();

// Entferne letztes Element

personen.pop();

// Ermittele Index von Person „erik“; -1, wenn Person nicht gefunden

let erikIndex = personen.indexOf("erik");

// Gib letztes Element des Arrays aus

console.log(personen[personen.length-1]);

* *Arrays iterieren*

// Lege Array mit Personen an

let personen = ["paula", "erik", "julia", "max"];

//Iteriere durch Personenliste und gib jede aus

for(let element of personen) {

console.log(element);

}

//Iteriere durch Personenliste und gib jede aus (Alternative)

personen.forEach(element => {

console.log(element);

});

* *Mehrdimensionale Arrays*

// Lege mehrdimensionales Array mit zwei Teilarrays und je zwei Personen an

let personen = [

["paula", "erik"],

["julia", "max"]

];

// Gibt "erik" aus (zweites Element in erstem Teilarray)

console.log(personen[0][1]);

// Gibt alle Elemente aus

for(let teilarray of personen) {

for(let person of teilarray) {

console.log(person);

}

}

// Gibt auch alle Elemente aus

personen.forEach(teilarray => {

teilarray.forEach(person => {

console.log(person);

});

});

* *Schleifen*

// Lege mehrdimensionales Array mit zwei Teilarrays und je zwei Personen an

let personen = [

"paula", "erik",

"julia", "max"

];

for(let i=0;i < personen.length; i++) {

console.log(personen[i]);

}

let i2 = 0;

while(i2<personen.length) {

console.log(personen[i2]);

i2++;

}

let i3 = 0;

do {

console.log(personen[i3]);

i3++;

} while(i3<personen.length);

## Objekte und Objektorierentierung

* *Einfache JavaScript Objekte anlegen*

// Neues Objekt erstellen

let person = { firstname: "Max", lastname: "Mustermann", age: 24 };

// Auf Objekt zugreifen

console.log(person.firstname);

console.log(person["firstname"]);

// Objekt erweitern ...

person.height = 178;

// ... oder Werte überschreiben

person.firstname = "Julian";

// Attribute löschen

delete person.height;

* *Klassen*

"use strict";

class Person {

constructor(firstname, lastname, age) {

this.firstname = firstname;

this.lastname = lastname;

this.age = age;

}

printPerson() {

console.log(this.firstname + " " + this.lastname + " " + "(" + this.age + ")");

}

}

class Student extends Person {

constructor(firstname, lastname, age, subject) {

// Super() MUSS aufgerufen werden, auch wenn kein Konstruktor in der Basisklasse explizit erstellt wurde

super(firstname,lastname,age);

this.subject = subject;

}

printPerson() {

console.log("Student " + this.firstname + " " + this.lastname + " " + "(" + this.age + ")" + " is enrolled in " + this.subject);

}

}

let p = new Person("Max", "Mustermann", 24);

p.printPerson();

let s = new Student("Max", "Mustermann", 24, "EIS");

s.printPerson();

* *(!) Kontextuelles Problem mit „this“*

*Beispiel 1:*

"use strict";

class Person {

constructor(firstname, lastname, age) {

this.firstname = firstname;

this.lastname = lastname;

this.age = age;

}

printPerson() {

console.log(this.firstname + " " + this.lastname + " " + "(" + this.age + ")");

}

}

let p = new Person("Max", "Mustermann", 24);

// Funktioniert

p.printPerson();

// Funktioniert NICHT, da der Kontext verloren gegangen ist. „This” ist hier undefined

let print = p.printPerson;

print();

*Beispiel 2:*

"use strict";

class Person {

constructor(firstname, lastname, age) {

this.firstname = firstname;

this.lastname = lastname;

this.age = age;

}

enroll() {

setTimeout(function() {

// "this" verweist hier auf das Window Objekt, nicht auf das "Person" Objekt, da es im Kontext von window.setTimeout(..) ausgeführt wird

console.log(this);

}, 1000);

}

}

*Lösung zu Beispiel 2: Der Fat-Arrow Operator =>*

"use strict";

class Person {

constructor(firstname, lastname, age) {

this.firstname = firstname;

this.lastname = lastname;

this.age = age;

}

enroll() {

setTimeout(()=> {

// "this" zeigt nun auf das "Person" Objekt

console.log(this);

}, 1000);

}

}

* *KlassenvorES6 (Prototype Inheritance)*

"use strict";

// Klasse wird in Funktion umgewandelt => Stellt Konstruktor dar

function BankAccount(balance) {

this.\_initialBalance = balance;

this.\_withdraws = [];

};

//Einzelne Methoden werden über prototyp Eigenschaft gepflegt

BankAccount.prototype.withdraw = function(amount) {

if (this.getBalance() - amount < 0) {

console.log("FEHLER!");

} else {

this.\_withdraws.push(amount);

}

};

BankAccount.prototype.getBalance = function() {

let balance = this.\_initialBalance;

for(let withdraw of this.\_withdraws) {

balance = balance - withdraw;

}

return balance;

};

// ist identisch mit:

class BankAccount {

constructor(balance) {

this.\_initialBalance = balance;

this.\_withdraws = [];

}

withdraw(amount) {

if (this.getBalance() - amount < 0) {

console.log("FEHLER!");

} else {

this.\_withdraws.push(amount);

}

}

getBalance() {

let balance = this.\_initialBalance;

for(let withdraw of this.\_withdraws) {

balance = balance - withdraw;

}

return balance;

}

}

// Verwendung der Klasse wie bei der klassischen Objektorientierung

// Wichtig: new-Operator

let b = new BankAccount(1100);

b.withdraw(200);

b.withdraw(900);

console.log(b.getBalance());

## jQuery

* *Zugriff auf Oberflächenelement*

<html>

<head>

<meta charset="UTF-8" />

</head>

<body>

<p class="absatz">Hallo Welt</p>

<p class="absatz super">Hallo Welt</p>

<p class="absatz super">Hallo Welt</p>

<script type="text/javascript" src="./lib/jquery/jquery-3.1.1.js"></script>

<script type="text/javascript" src="index.js"></script>

</body>

</html>

jQuery(document).ready(function($) {

// Zugriff auf ALLE "p" Elemente. Es wird KEINE Schleife benötigt

$("p.absatz")

.css("color", "green")

.css("background-color", "red");

// Zugriff auf ALLE "p" Elemente mit der Klasse "absatz"

$("p.absatz")

.css("color", "green")

.css("background-color", "red");

});

* *Eventhandler*

// Alle "p" Elemente mit den Klassen "absatz" und "super" erhalten einen Click-Handler.

// "This" verweist auf das Element, das angeklickt wurde

$("p.absatz.super").click(function(event) {

console.log(this);

$(this).css("background-color", "blue");

});

// Manche Handler erwarten auch mehrere Handler (Hover = Maus betritt und verlässt Element)

$("p.absatz.super").hover(

function() {

$(this).css("background-color", "yellow");

}, function() {

$(this).css("background-color", "red");

}

);

Standardverhalten verhindern z.B. Navigieren über Link

<html>

<head>

<meta charset="UTF-8" />

<style>

.myButtonStyle {

padding: 5px;

background-color: beige;

}

</style>

</head>

<body>

<a href="http://www.google.de" class="myButtonStyle">Hallo Welt</a>

<script type="text/javascript" src="./lib/jquery/jquery-3.1.1.js"></script>

<script type="text/javascript" src="index.js"></script>

</body>

</html>

$(".myButtonStyle").click(function(event) {

//Verhindert das normale Verhalten (Link wird bei Klick nicht verfolgt)

event.preventDefault();

// Gibt die angesteuerte Adresse aus

console.log($(this).attr("href"));

$(this).css("background-color", "green");

});

* *Schleifen*

<html>

<head>

<meta charset="UTF-8" />

</head>

<body>

<p>Hallo Welt</p>

<p>Hallo Welt</p>

<p>Hallo Welt</p>

<p>Hallo Welt</p>

<script type="text/javascript" src="./lib/jquery/jquery-3.1.1.js"></script>

<script type="text/javascript" src="index.js"></script>

</body>

</html>

// Ebenfalls Zugriff auf ALLE "p" Elemente. Enthält eine Referenz auf

//das aktuelle Element des Schleifendurchgangs und eine Zählervariable

$("p").each(function(i, element) {

if(i%2==0) {

$(element).css("background-color", "green");

} else {

$(element).css("background-color", "red");

}

});

## NodeJS