Inhalt

[Variablendeklaration 1](#_Toc510342)

[Datentypen 1](#_Toc510343)

[Arithmetische Operatoren 1](#_Toc510344)

[Arrays: 1](#_Toc510345)

[If-Statement 2](#_Toc510346)

[Switch Statement 2](#_Toc510347)

[Schleifen 2](#_Toc510348)

[Arithmetische Operatoren 2](#_Toc510349)

[increment , decrement 3](#_Toc510350)

[interpolieren von Daten in einen String 3](#_Toc510351)

[if statement 2 3](#_Toc510352)

[? operator 4](#_Toc510353)

[switch case 4](#_Toc510354)

[funktion deklaration, Default Wert 4](#_Toc510355)

[aufrufen von Funktionen 4](#_Toc510356)

[andere Möglichkeit funktionen aufzurufen 4](#_Toc510357)

[andere Möglichkeit Funktionen zu erstellen mit Arrow 5](#_Toc510358)

[mehrere Parameter in der Pfeil-Sprache 5](#_Toc510359)

[Arrays 2 5](#_Toc510360)

[Schleifen 2 6](#_Toc510361)

[.findIndex() 7](#_Toc510362)

[Objekte erstellen mit Attributen 7](#_Toc510363)

[Zugriff auf Attribute 7](#_Toc510364)

[Werte für Objekte eingeben 7](#_Toc510365)

[aufrufen von Objektmethoden 8](#_Toc510366)

[Nested Objects 8](#_Toc510367)

[ansprechen von Nested Objects 8](#_Toc510368)

[getter Methode in objekten 9](#_Toc510369)

[aufrufen von getter Methoden 9](#_Toc510370)

[Setter Methoden in Objekten 9](#_Toc510371)

[Setter Methode setzten 9](#_Toc510372)

[Viele Objekte gleichzeitig erstellen 10](#_Toc510373)

[Klasse erstellen mit Konstruktor 10](#_Toc510374)

[Objekt erstellen 11](#_Toc510375)

[Methoden aufrufen 11](#_Toc510376)

[Vererbung + Babbel ES6 in ES5 konfigurieren 11](#_Toc510377)

[Module 11](#_Toc510378)

[importieren + exportieren von Modulen 12](#_Toc510379)

[Promise Objekt erstellen 12](#_Toc510380)

[Promise machen 13](#_Toc510381)

[Promise mit .then(onFulFilled, OnRejected) funktion 13](#_Toc510382)

[Verkettung von Resolves 14](#_Toc510383)

[Promise.all() um alle Promises zu starten 14](#_Toc510384)

[asynchrone Funktionen erstellen 14](#_Toc510385)

[XHR GET Request mit XMLHttpRequest 15](#_Toc510386)

# Variablendeklaration

var Wert = "Hallo";

console.log(Wert); // Gibt Wert auf der Console aus

// Einzeiliger Kommentar

/\* mehrzeiliger Kommentar \*/

# Datentypen

var myNumber = 2; //(Alle Zahlen)

var myString = "Hallo"; //(‘Zeichen‘, “Zeichen“)

var myBoolean = true;

let myNumber1 = Null;

/\*Undefined

Symbol

Object (Kollektion von verwandten Daten) \*/

# Arithmetische Operatoren

myNumber = myNumber+2-2\*3/3; // +,-,\*,/,%

myString = myString + "Hallo";

// /=,\*=,-=,+=,%=

// <,>,=<,=>,&&,||,!

//Einige Methoden:

console.log("hallo".toUpperCase());

console.log("hallo".startsWith("H"));

// Definition Datentypen:

var myString ="Hello,World!";

var myArray = [];

var myObject = {};

# Arrays:

// Definition:

var myArray = [1,2,3];

var myArray = newArray(1,2,3);

/\*(Addressierung von Arrays)\*/ console.log(myArray[1]);

//Man kann Werte mit verschiedenen Datentypen in Array schreiben

// Mit Array.push(1); kann man Werte in Array einfügen

//Mit Array.pop(); löscht Werte aus Array

//myArray.unshift(); fügt Wert an Anfang von Array

// mit .splice(Anfang, Ende) kann man Werte von Arrays in neues Array Schreiben

# If-Statement

/\*confirm(“ “);

 === überprüft ob Datentyp und Wert identisch sind

\*/

# Switch Statement

Switch(variable)

{

case “Privat“:

case “Sergeant“:

Anweisungen;

break;

default:

break;

# Schleifen

 for(i=0;i<3;i = i+1){ Anweisung;};

\*/

//variablendeklaration

var favouriteFood = 'pizza';

var favouriteFood1; // wert undefined

let favouriteFood2 = 'hallo';

console.log(favouriteFood.toUpperCase()); // => macht Schrift groß

favouriteFood.trim(); //=> schneidet Leerzeichen am Anfang und Ende ab

Math.random(); // => gibt zufällige zahl aus

Math.floor(4); //=> rundet auf nächste ganze zahl runter

Math.ceil(4); //=> gibt den kleinsten integer größer oder gleich der d

// Dezimalzahl aus

Math.isInteger(3); //=> gibt dir aus ob es ein Integer ist

// Konstante

const myName = 'Kevin';

# Arithmetische Operatoren

var x =20;

x -=5;

x +=5;

x/=5;

x\*= 5;

# increment , decrement

var a =10;

a++; // a= 11

a--; // a= 10

// man kann Strings addieren

# interpolieren von Daten in einen String

const myPet = 'armadillo';

console.log(`i own a pet ${myPet}.`);

// typeof gibt den Typ der Variable aus

const unknown1 = 'foo';

console.log(typeof unkown1);

# if statement 2

if (true) {

console.log('This message will print!');

}

// if- else

if (false) {

console.log('The code in this block will not run.');

} else {

console.log('But the code in this block will!');

}

// vergleichoperatoren <,>,<=,>=, ===,!==

//logische Operaten &&,||,!

//truthy ,falsy

// falsy Werte sind : 0,"" ,'',null, undefined, Nan

var myVariable = 'I Exist!';

if (myVariable) {

console.log(myVariable)

} else {

console.log('The variable does not exist.')

}

// short-circuit evaluation

var defaultName = username || 'Stranger';

# ? operator

var isNightTime = true;

isNightTime ?

console.log('Turn on the lights!') :

console.log('Turn off the lights!');

// man kann auch else if statement benutzen

# switch case

var groceryItem = 'tomato';

switch (groceryItem) {

case 'tomato':

console.log('Tomatoes are $0.49');

break;

case 'lime':

console.log('Limes are $1.49');

break;

case 'papaya':

console.log('Papayas are $1.29');

break;

default:

console.log('Invalid item');

break;

}

# funktion deklaration, Default Wert

function greetWorld(name = 'hallo') {

return console.log('Hello, World!');

}

# aufrufen von Funktionen

greetWorld();

# andere Möglichkeit funktionen aufzurufen

const helloWorld = function(){console.log('hallo');};

helloWorld;

# andere Möglichkeit Funktionen zu erstellen mit Arrow

const width =1;

const height =1;

const rectangleArea = (width, height) => {

let area = width \* height;

return area

}

# mehrere Parameter in der Pfeil-Sprache

const functionName = () => {};

const functionName = paramOne => {};

const functionName = (paramOne,paramTwo) => {};

# Arrays 2

// Arrays in javaScript können verschiedene Datentypen enthalten

// Array Deklaration mit Array Literal

var neuesArray = ['element example',10,true];

// Index fängt bei 0 an

// Zugriff auf Array

console.log(neuesArray[0]);

// Wert updaten

neuesArray[0] = 'update';

// länge vom array erhalten

console.log(neuesArray.length);

// mit .push() kann man werte in einem Array hinter das Array hinzufügen

neuesArray.push('item3','item4');

// mit .pop() wird der letzte eintrag eines Array gelöscht

neuesArray.pop();

// mit .shift() wird der erste Eintrag in einem Array gelöscht

neuesArray.shift();

// mit .unshift() kann am Anfang eines Arrays ein Wert hinzugefügt werden

neuesArray.unshift();

// mit .slice() kann man beliebige Werte in einem Array zusammenfassen

neuesArray.slice(10,true);

// nested Arrays

const nestedArr = [[1], [2, 3]];

# Schleifen 2

// for-Schleife

for (var counter = 0; counter < 4; counter++) {

console.log(counter);

}

// durch Arrays loopen mit for

const animals = ['Grizzly Bear', 'Sloth', 'Sea Lion'];

for (var i = 0; i < animals.length; i++){

console.log(animals[i]);

}

// while-Schleife

var counterTwo = 1;

while (counterTwo < 4) {

console.log(counterTwo);

counterTwo++;

}

// do-while Schleife

var countString = '';

var i = 0;

do {

countString = countString + i;

i++;

} while (i < 5);

// mit break springt man aus schleifen

// .forEach() methode geht durch einen Array und untersucht jedes element

const groceries = ['brown sugar','salt','cranberries','walnuts'];

groceries.forEach(groceryItem => console.log(groceryItem));

// .map() nimmt ein argument von der "callback" funktion und erstellt neuen Array und gibt ihn zurück

const numbers = [1, 2, 3, 4, 5];

const bigNumbers = numbers.map(number => {

return number \* 10;

});

.filter()

// .filter gibt einen Array zurück mit den Elementen die man gefiltert hat

const words = ['chair', 'music', 'pillow', 'brick', 'pen', 'door'];

const shortWords = words.filter(word => {

return word.length < 6;

});

# .findIndex()

// mit findIndex() kann man den ersten Index zurückgeben, der in einem Array einen Wert entspricht

const jumbledNums = [123, 25, 78, 5, 9];

const lessThanTen = jumbledNums.findIndex(num => {

return num < 10;

});

# Objekte erstellen mit Attributen

var spaceship = {

'Fuel Type' : 'diesel',

color: 'silver'

};

# Zugriff auf Attribute

spaceship.color;

spaceship["Fuel Type"]; // gibt diesel zurück

//Zugriffe auf Attribute bei Funktionen

let returnAnyProp = (objectName, propName) => objectName[propName];

returnAnyProp(spaceship, 'homePlanet');

# Werte für Objekte eingeben

spaceship['Fuel Type'] = 'vegetable oil';

// Eingeschaft / Attribut löschen

delete spaceship["Fuel Type"];

//methoden in Objekte

const alienShip = {

invade: function () {

console.log('Hello! We have come to dominate your planet. Instead of Earth, it shall be called New Xaculon.')

}

};

// andere Schreibweise

const alienShip1 = {

invade () {

console.log('Hello! We have come to dominate your planet. Instead of Earth, it shall be called New Xaculon.')

}

};

# aufrufen von Objektmethoden

alienShip.invade();

# Nested Objects

const spaceship = {

telescope: {

yearBuilt: 2018,

model: '91031-XLT',

focalLength: 2032

},

crew: {

captain: {

name: 'Sandra',

degree: 'Computer Engineering',

encourageTeam() { console.log('We got this!') }

}

},

engine: {

model: 'Nimbus2000'

},

nanoelectronics: {

computer: {

terabytes: 100,

monitors: 'HD'

},

'back-up': {

battery: 'Lithium',

terabytes: 50

}

}

};

# ansprechen von Nested Objects

spaceship.nanoelectronics['backup'].battery;

// es gibt auch Pass by reference

// mit let ... in kann man durch objekte gehen und werte rausbekommen

for (let crewMember in spaceship.crew) {

console.log(`${crewMember}: ${spaceship.crew[crewMember].name}`)

};

// this referiert auf das derzeitige Objekt

// wenn man vor einer Eigenschaft \_ verwendet, soll diese nicht geändert werden

// z.B. \_amount :1000

# getter Methode in objekten

const person = {

\_firstName: 'Kevin',

get fullName(){

if(this.\_firstName){

return `${this.\_firstName}`;

}

}

}

# aufrufen von getter Methoden

person.fullName;

# Setter Methoden in Objekten

const person1 ={

\_age: 37,

set age(newAge){

if (typeof newAge === 'number'){

this.\_age = newAge;

}

}

}

# Setter Methode setzen

person1.\_age = 40;

// um viele Objekte zu erstellen kann man eine "Factory Funktion" benutzen

const monsterFactory = (name, age, energySource, catchPhrase) => {

return {

name: name,

age: age,

energySource: energySource,

scare() {

console.log(catchPhrase);

}

}

};

# Viele Objekte gleichzeitig erstellen

const ghost = monsterFactory('Ghouly',251,'ectoplasm','Boo!');

// zur vereinfachung von oben

const monsterFactory = (name, age) => {

return {

name,

age

}

};

// destructed Assignment um die Zuweisung zu erleichtern

/\*const { day } = vampire.preferences;

console.log(day);\*/

//Object.key(objekt) speichert die Attributnamen in einem Array

//Object.entries() gibt mehrere Arrays zurück mit dem key und dem Wert von den Eigenschaften

// Object.assign({eigenschaft:wert},objekt) gibt alle werte des objekts in einen array und fügt weitere hinzu

# Klasse erstellen mit Konstruktor

class Dog {

constructor(name) {

this.\_name = name;

this.behavior = 0;

//Getter Methode, Methoden

get\_name()

{

return this.\_name

}

get\_behavior()

{

return this.\_behavior;

}

incrementBehavior()

{

this.\_behavior++;

}

}

# Objekt erstellen

const

halley = new Dog('Halley');

halley

.

incrementBehvaior();

# Methoden aufrufen

let

nikko = new Dog('Nikko');

incrementBehavior();

}

# Vererbung + Babbel ES6 in ES5 konfigurieren

// Vererbung von Klassen man erbt mit extends Oberklasse

// statische Methoden können mit static deklariert werden

// babbel JS-version ES6 in ES5

// npm init erstellt package.json file

// npm install -> installiert Node Pakete

// babel-cli -D node package für Babbel

// babbel-preset-env -D ES6 in ES5 syntax mappen

// npm run build run script and transpile it in ES5

// .babelrc file erstellen und {"presets" : ["env"]} hinzufügen

// "build" : "babel src -d lib" in package.json hinzufügen

// npm run build ausführen

# Module

// Module für reusable code // für andere das modul zur Verfügung stellen

let Menu = {};

Menu.specialty = "Roasted Beet Burger with Mint Sauce";

export default Menu;

// importieren von Modulen

let Airplane = {};

Airplane.myAirplane = "StarJet";

export default Airplane;

function displayAirplane() {

console.log(Airplane.myAirplane);

}

displayAirplane();

# importieren + exportieren von Modulen

// importieren von modulen, die in einem Objekt gespeichert sind

const Airplane = {};

export default {

myAirplane: "CloudJet",

displayAirplane: function() {

return this.myAirplane;

}

};

// import in ES6 import Menu from './menu';

// exportieren von modulteilen durch variablen

//export { specialty, isVegetarian };

// importieren von Modulteilen durch variablen

//import { specialty, isVegetarian } from './menu';

// verändern von Variablen beim exportieren

// export { specialty as chefsSpecial, isVegetarian as isVeg, isLowSodium };

// importieren und Variablen umschreiben

// import \* as Carte from './menu';

// man kann imports und exports kombinieren

# Promise Objekt erstellen

const executorFunction = (resolve, reject) => { };

const myFirstPromise = new Promise(executorFunction);

# Promise machen

const inventory = {

sunglasses: 0,

pants: 1088,

bags: 1344

};

const myExecutor = (resolve,reject) => {

if(inventory.sunglasses > 0){

resolve('Sunglasses order processed.');

} else{

reject('That item is sold out.');

}

}

const orderSunglasses= () =>{

return new Promise(myExecutor);

}

const orderPromise = orderSunglasses();

console.log(orderPromise)

// Funktion setTimeout(function,millisek); function soll ausgeführt werden nach einer gewissen Zeit

# Promise mit .then(onFulFilled, OnRejected) funktion

let prom = new Promise((resolve, reject) => {

let num = Math.random();

if (num < .5 ){

resolve('Yay!');

} else {

reject('Ohhh noooo!');

}

});

const handleSuccess = (resolvedValue) => {

console.log(resolvedValue);

};

const handleFailure = (rejectionReason) => {

console.log(rejectionReason);

};

prom.then(handleSuccess, handleFailure);

// mit .catch() können failures gehandelt werden

prom

.then((resolvedValue) => {

console.log(resolvedValue);

})

.catch((rejectionReason) => {

console.log(rejectionReason);

});

# Verkettung von Resolves

firstPromiseFunction()

.then((firstResolveVal) => {

return secondPromiseFunction(firstResolveVal);

})

.then((secondResolveVal) => {

console.log(secondResolveVal);

});

# Promise.all() um alle Promises zu starten

let myPromises = Promise.all([returnsPromOne(), returnsPromTwo(), returnsPromThree()]);

myPromises

.then((arrayOfValues) => {

console.log(arrayOfValues);

})

.catch((rejectionReason) => {

console.log(rejectionReason);

});

# asynchrone Funktionen erstellen

async function myFunc() {

// Function body here

};

myFunc();

// oder

const myFunc = async () => {

// Function body here

};

myFunc();

// await operator wartet, bis ein Promise ausgeführt wurde

async function asyncFuncExample(){

let resolvedValue = await myPromise();

console.log(resolvedValue);

}

asyncFuncExample();

// asynchrone funktionen mit try-Catch error handling

async function usingTryCatch() {

try {

let resolveValue = await asyncFunction('thing that will fail');

let secondValue = await secondAsyncFunction(resolveValue);

} catch (err) {

// Catches any errors in the try block

console.log(err);

}

}

# XHR GET Request mit XMLHttpRequest

const xhr = new XMLHttpRequest();

const url = "https://api-to-call.com/endpoint";

xhr.responseType = 'json'; // soll in json datei zurückgegeben wird

xhr.onreadystatechange = () => {

if(xhr.readyState === XMLHttpRequest.DONE){ //prüft ob request beendet ist

return xhr.response;

}

};

xhr.open('GET',url); // new Request

xhr.send(); // absenden vom Request

// XMLHttpRequest Beispiel

// Information to reach API

const url ="https://api.datamuse.com/words?";

const queryParams = 'rel\_rhy=';

// Selecting page elements

const inputField = document.querySelector('#input');

const submit = document.querySelector('#submit');

const responseField = document.querySelector('#responseField');

// AJAX function

const getSuggestions = () => {

const wordQuery = inputField.value;

const endpoint =`${url}${queryParams}${wordQuery}`;

const xhr = new XMLHttpRequest();

xhr.responseType = 'json';

xhr.onreadystatechange = () => {

if (xhr.readyState === XMLHttpRequest.DONE) {

renderResponse(xhr.response);

}

};

xhr.open('GET',endpoint);

xhr.send();

}

// Clear previous results and display results to webpage

const displaySuggestions = (event) => {

event.preventDefault();

while(responseField.firstChild){

responseField.removeChild(responseField.firstChild);

};

getSuggestions();

}

submit.addEventListener('click', displaySuggestions);

// Formats response to look presentable on webpage

const renderResponse = (res) => {

// Handles if res is falsey

if(!res){

console.log(res.status);

}

// In case res comes back as a blank array

if(!res.length){

responseField.innerHTML = "<p>Try again!</p><p>There were no suggestions found!</p>";

return;

}

// Creates an empty array to contain the HTML strings

let wordList = [];

// Loops through the response and caps off at 10

for(let i = 0; i < Math.min(res.length, 10); i++){

// creating a list of words

wordList.push(`<li>${res[i].word}</li>`);

}

// Joins the array of HTML strings into one string

wordList = wordList.join("");

// Manipulates responseField to render the modified response

responseField.innerHTML = `<p>You might be interested in:</p><ol>${wordList}</ol>`;

return

}

// Renders response before it is modified

const renderRawResponse = (res) => {

// Takes the first 10 words from res

let trimmedResponse = res.slice(0, 10);

// Manipulates responseField to render the unformatted response

responseField.innerHTML = `<text>${JSON.stringify(trimmedResponse)}</text>`;

}

// get und Post request siehe Bilder

// um query parameter zu benutzen muss man ? + parameter schreiben

//fetch request

fetch("https://api-to-call.com/endpoint")

.then((response) => {

if(response.ok){

return response.json();

}throw new Error('Request failed!');

},(networkError) =>{

console.log(networkError.message);

})

.then(jsonResponse => {

return jsonResponse;

})