

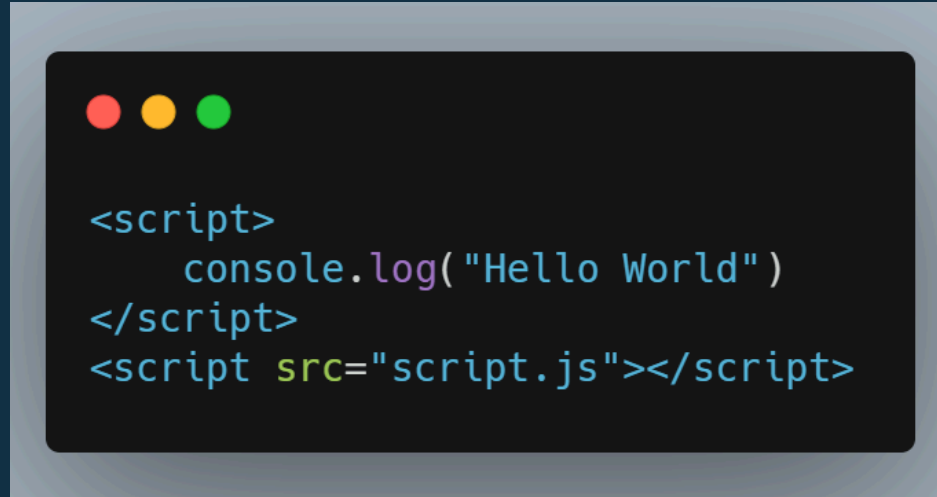
The logo consists of a solid yellow square centered on a dark blue background. Inside the yellow square, the letters 'JS' are written in a large, bold, dark gray sans-serif font.

JS

JAVASCRIPT

# HTML UND JAVASCRIPT

JavaScript kann über das Script-Element in HTML Dateien eingebunden werden.



```
<script>  
    console.log("Hello World")  
</script>  
<script src="script.js"></script>
```

Die Einbindung einer externen Datei erfolgt im `<head>` oder vor dem `</body>`-Element.

# LADEN DER DATEIEN - *BODY*

*Das Setzen des **script-Tags** am Ende des Body hat zur Folge, dass zuerst die HTML parse und dann die js ausgeführt wird. Sollte grundlegend benutzt werden, um auch ältere Browser zu unterstützen.*

# LADEN DER DATEIEN - *DEFER*

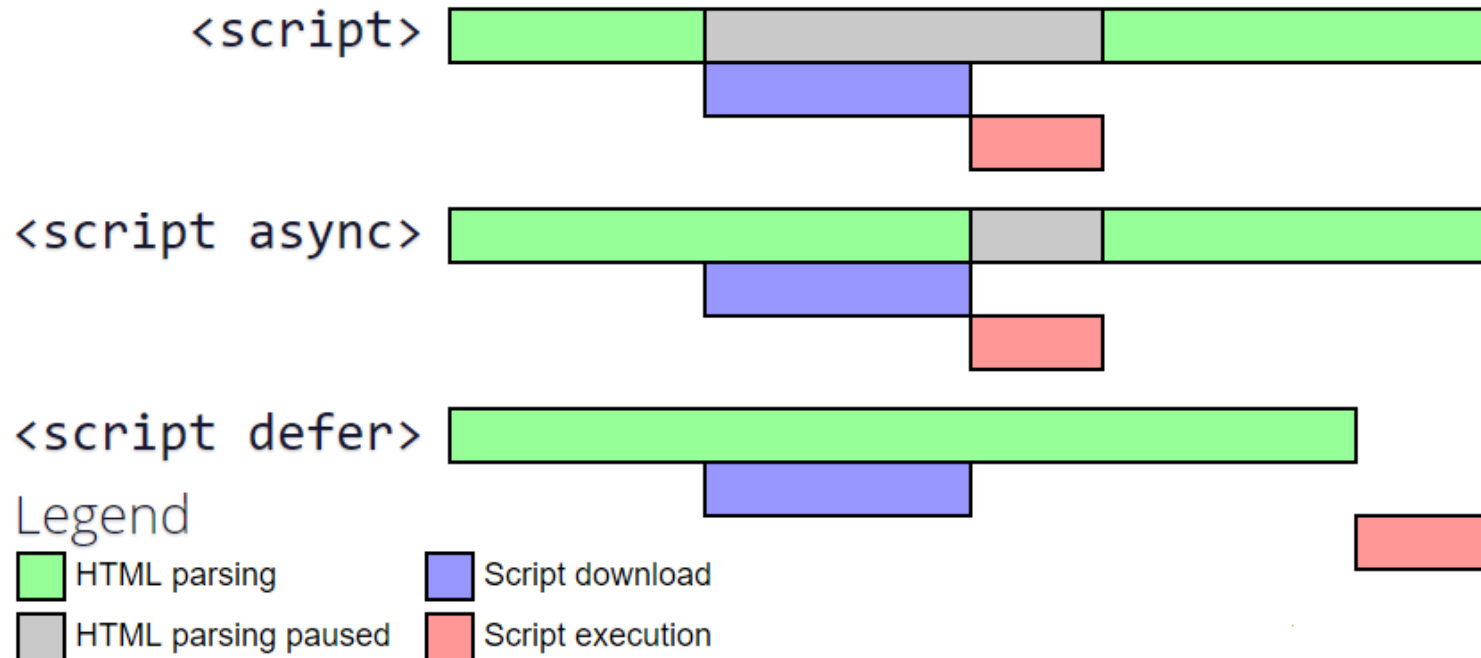
*Das Setzen des Attributs **defer** im **script-tag** hat zur Folge, dass das **html** Dokument zuerst parse und dann die **js** in der Reihenfolge ausgeführt wird, wie sie im Dokument erscheint.*

# LADEN DER DATEIEN - ASYNC

*Das Setzen des Attributs **async** im **script-tag** hat zur Folge, dass die js während des parse des **html** Dokuments runterlädt. Wenn diese runtergeladen ist, wird das parse des **html** Dokuments gestoppt und die js direkt ausgeführt. Scripte mit hoher Priorität und unabhängig von restlichen Dateien wie Google Analytics können mit **async** eingebunden werden.*

# LADEN DER DATEIEN

## async vs defer



# QUELLEN

Doc-1 Doc-2

# EXKURS: GITHUB PAGES/JEKYLL

Github Pages bietet einen statisches deployment eines Github Repos. Das Project kann dann online abgerufen werden.



## ⚙ General

## Access

Collaborators

Moderation options ▾

## Code and automation

Branches

Tags

Actions ▾

Webhooks

Environments

Codespaces

Pages

## GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at <https://sqz0111.github.io/PokemonMemoryTest/>

Last deployed by SQZ0111 yesterday

Visit site

...

## Build and deployment

## Source

Deploy from a branch ▾

## Branch

Your GitHub Pages site is currently being built from the `main` branch. [Learn more.](#)

main ▾

/ (root) ▾

Save

# EXKURS GITHUB-PAGES/JEKYLL

*mit dem base -Tag kann das default Target für alle Navigationen festgelegt werden.*

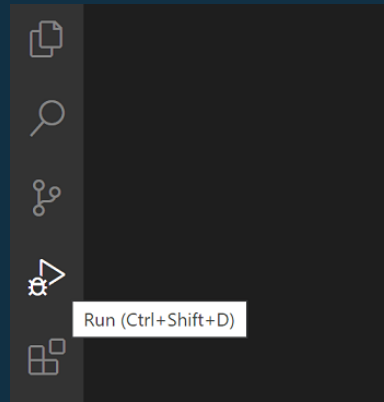
*Unter Custom domain kann eines eigen  
Domaine angegeben werden.*

*Der Workflow kann mit Github Actions  
verändert werden.*

[Github-Pages](#)

# KONSOLE ÜBER VSCODE

Konsolen Ausgaben lassen sich auch über die Debugging Funktion in VSCode anzeigen.



<https://code.visualstudio.com/docs/nodejs/browser-debugging>

# VARIABLEN

JavaScript kennt verschiedene Arten von Variablen.

```
var x = 5;  
  
let y = 10;  
  
const name = "Ada";
```

# VARIABLEN

## Scopes

	var	let	const
Global Scope	✓	✓	✓
Function Scope	✓	✓	✓
Block Scope	✗	✓	✓
Can be reassigned?	✓	✓	✗
Can be redeclared?	✓	✗	✗

# VARIABLEN SCOPE

```
let firstname = "Ada"; // Global Scope

function sayName(name) { // Function Scope
  console.log(name);
}

if (firstname === "Ada") {
  // Block Scope {}
  console.log("Hello %s", firstname)
}
```

# VARIABLEN SCOPE

```
var x = "outside";  
function foo() {  
  var x = "inside";  
  console.log(x);  
}  
foo();  
console.log(x);
```

*Was wird ausgegeben?*

# VARIABLEN SCOPE

```
let i = 0;  
if (true) {  
  let i = 1;  
}  
console.log(i);
```

*Was wird ausgegeben?*



# DATENTYPEN

Boolean

Zwei Werte: *true* und *false*

Null

Einen Wert: *null*

Undefined

Eine Variable der noch kein Wert zugewiesen wurde.

Number

Zahlen; Ganzzahlig und Gleitkommazahl

String

Zeichenketten; stehen in Anführungszeichen

# ARRAYS UND OBJEKTE

Zur *Speicherung* mehrere Daten in einer Variable kennt JavaScript Arrays und Objekte.

```
// Array
var color = ["blue", "red", "yellow"] // Zugriff über Index (0...)

// Object
var person = {lastname: "Lovelace", firstname: "Ada",
  name: "value"}
```

Auch Arrays sind Objekte.

# OBJECT REFERENCE

Variablen für ein Objekt speichern nicht den Inhalt sondern lediglich eine Referenz (Zeiger) auf das Objekt.  
Dadurch lässt sich ein Objekt durch eine Zuweisung **nicht** kopieren.

```
const array1 = [1, 2];  
const array2 = array1;  
// Änderungen an einem der Arrays wirken sich auf beide Arrays aus  
array2.pop(); // Löschen des letzten Elements  
console.log(array1);  
// Ausgabe: [1]
```

# KOPIEREN EINES OBJECTS / ARRAYS

zwei Methoden

```
// Array Methode  
const clone = original.slice();
```

```
// Object  
const clone = Object.assign({}, original);
```

# AUFGABE 03 - 1

Schreibe ein Programm, dass den Benutzer nach einem Alter in Jahren fragt. Anschließend soll das Programm das Alter jeweils in Monaten, Tagen und Stunden ausgeben.

```
prompt("Bitte gebe ein Alter in Jahren ein!");
```



# BEDINGUNGEN

Über Bedingungen lassen sich Fallentscheidungen treffen. In JavaScript werden hierfür die Schlüsselwörter **if** und **else**, sowie eine Kombination verwendet.

```
if (alter >= 18) {  
  console.log("Du darfst Auto fahren!");  
} else if (alter >= 17) {  
  console.log("Du darfst mit Begleitung fahren.");  
} else {  
  console.log("Du darfst noch kein Auto fahren");  
}
```

# VERGLEICHSOPERATOREN

Operator	Beschreibung
==	Equal – true, wenn Operanden gleich.
===	Strict Equal - true, wenn Operanden und Typ gleich.
!=	Not equal – true, wenn Operanden ungleich sind
!==	Strict not equal – true, wenn Operanden und Typ ungleich.
>	Greater than – true, wenn der linke Operand größer dem rechten Operanden ist.
>=	Greater than or equal – true, wenn der linke Operand größer als, oder gleich dem rechten Operanden ist.
<	Less than – true, wenn der linke Operand kleiner dem rechten Operanden ist.
<=	Less than or equal – true, wenn der linke Operand kleiner als, oder gleich dem rechten Operanden ist.

# BEDINGUNGEN

Bedingung mit dem logischen && Operator

```
// Ergebnis vor dem && ist wahr:  
true && console.log("Hier ist eine Ausgabe")  
// Ergebnis vor dem && ist falsch:  
false && console.log("Hier kommt keine Ausgabe")
```

Der &&-Operator gibt den ersten falschen (False) Wert aus.  
Abbildung von *else* nicht möglich!



# BEDINGUNGEN

## Bedingung mit Conditional Operator

```
// Condition ? if true : if false
```

```
true ? console.log("Ausgabe") : console.log("Keine Ausgabe")
```

# AUFGABE 03 - 1

Wenn die Eingabe des Alters negativ ist soll eine Fehlermeldung ausgegeben werden.



Löse die Aufgabe mit *if* sowie dem Conditional Operator.

# SCHLEIFEN

JavaScript kennt verschiedene Schleifentypen.

- for Schleife
- while Schleife
- do...while Schleife

# FOR-SCHLEIFE

```
for (var i = 0; i < 9; i++) {  
  console.log(i);  
  // more statements  
}  
// Inkrement Operator (++), addiert eine eins und  
// gibt den Wert zurück.
```

Die For-Schleife läuft so lange wie die Bedingung *true* ist.

# ITERABLE OBJECTS

Mit *for* können iterable Objekte durchlaufen werden.

```
let arr = [3, 5, 7]; // Array

for (let i in arr) {
  console.log(i); // logs "0", "1", "2"
}

for (let i of arr) {
  console.log(i); // logs "3", "5", "7"
}
```

*Iterator i und Iterable arr*

# WHILE SCHLEIFE

```
var n = 0;  
  
while (n < 3) {  
    n++;  
}
```

Die While-Schleife läuft so lange wie die Bedingung war (*true*) ist.

# DO...WHILE SCHLEIFE

```
var i = 0;  
do {  
  i += 1;  
  console.log(i);  
} while (i < 5);
```

Die do...while-Schleife läuft so lange wie die Bedingung *true* ist, jedoch mindestens einmal (Fußgesteuerte Schleife).

# AUFGABE 03 - 2

Schreibe ein Programm, dass aus einem Zahlen-Array die größte Zahl ermittelt und ausgibt.  
Nutze hierfür die for...of Schleife.

```
const zahlen = [12, 34, 29, 120, 55];  
...  
// Ausgabe: "Die größte Zahl ist 120"
```



Aufgabenstellung



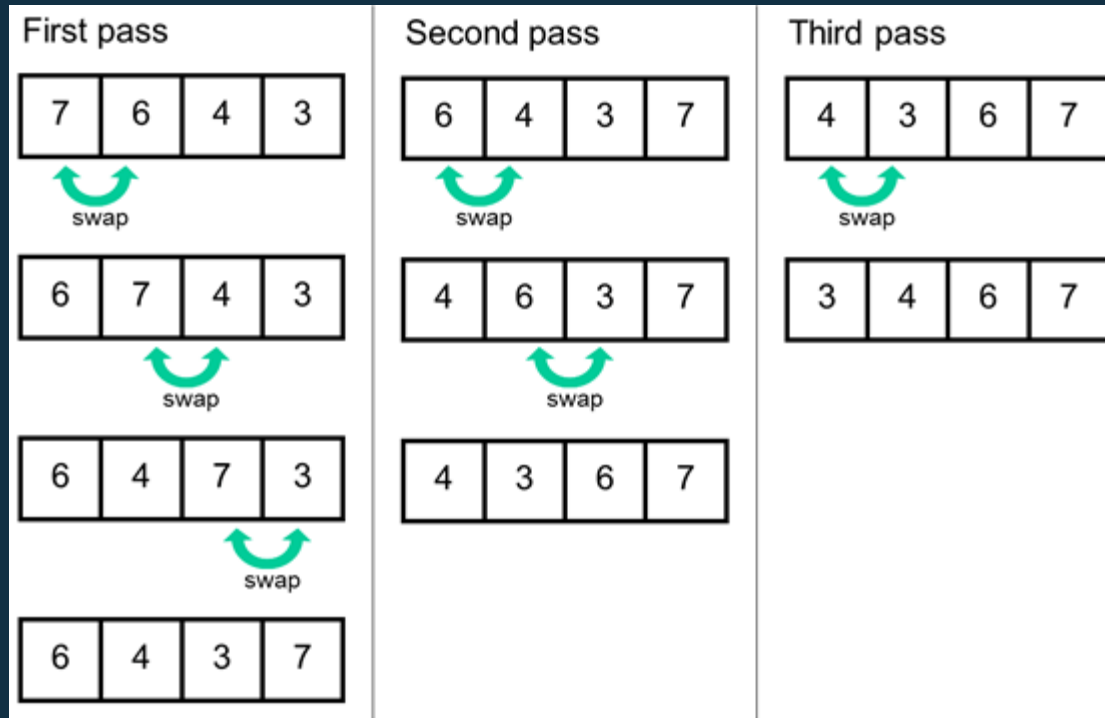
# SORTIER ALGORITHMEN

Mit Hilfe der Grundstrukturen einer Programmiersprache lassen sich verschiedene Sortier-Algorithmen abbilden.

- Bubble Sort
- Selection Sort
- Insertion Sort
- ...

<https://de.wikipedia.org/wiki/Sortierverfahren>

# BUBBLE SORT



*Zahlen werden durchlaufen und mit dem rechten Nachbarn verglichen. Bei Regelverletzung werden die Zahlen getauscht.*

# BUBBLE SORT

## Bubble Sort mit JavaScript

```
let arr = [12 , 5, 99, 34];

for (let n = arr.length; n>1; n--) {
  for(let i = 0; i < n-1; i++) {
    if(arr[i] > arr[i+1]) {
      let arr_old = arr[i];
      arr[i] = arr[i+1];
      arr[i+1] = arr_old;
    }
  }
}
```



# FUNKTIONEN

Eine Funktion beschreibt eine Reihe von Anweisungen, um eine Aufgabe auszuführen.

```
function square(number) {  
  return number * number;  
}  
  
const quadrat = square(2);
```

Häufig haben Funktionen Übergabeparameter () sowie einen oder mehrere Rückgabewerte (*return*).

# ANONYMOUS FUNCTIONS

In JavaScript lassen sich Funktionen ohne Namen erzeugen.

```
let show = function () {  
  console.log('Anonymous function');  
};  
show();
```

Zum späteren Aufruf können anonyme Funktionen in eine Variable gespeichert werden. In einigen Fällen ist eine Speicherung mit Namen jedoch nicht erforderlich.

# AUFGABE 03 - 2

Implementiere die Rückgabe des höchsten Wertes in eine Funktion.

Bei Funktionsaufruf soll ein Array übergeben werden und der höchste Wert zurückgegeben werden.



Aufgabenstellung

# INTERVAL UND TIMEOUT

Über *setInterval* und *setTimeout* lassen sich Funktionsaufrufe zeitlich steuern.

```
// Alle drei Sekunden wird Hello ausgegeben.  
setInterval(function(){ console.log("Hello"); }, 3000);  
// Nach drei Sekunden wird Hello ausgegeben.  
setTimeout(function(){ console.log("Hello"); }, 3000);
```

Die Angabe der Zeit erfolgt in Millisekunden.

# ARRAY METHODEN

Arrays haben viele vordefinierte Methoden um die Daten zu verarbeiten.

```
// Entfernen des letzten Elements und Rückgabe  
const letztesElement = zahlen.pop();  
// Sortieren  
zahlen.sort();
```



## Überblick der Methoden

*Ähnliche Methoden gibt es auch für Strings (siehe Übungen).*



# ARRAY METHODEN






.map(=>) => 

.filter() => 






.every() => false

.some() => true

.fill(1, ) => 

.findIndex(el => el===) => 3

.find() => 

.reduce((acc, cur)=>acc+cur)=> 

# AUFGABE 03 - 2

Erweitere die Zahlen-Array Aufgabe um die Funktion `sort()`.  
Gebe die Zahlen in umgekehrter Reihenfolge (Groß nach Klein) aus.



Aufgabenstellung

# ARRAY METHODE MAP()

Wendet eine Funktion auf jedes Array Element an und gibt ein neues Array zurück.

```
const numbers = [4, 9, 16, 25];  
const newArr = numbers.map(verdoppeln)  
  
function verdoppeln(zahl) {  
    return zahl * 2;  
}
```

# ARRAY METHODE FILTER()

Prüft jedes Element gegen eine Funktion deren Rückgabe *true* oder *false* ist.

```
const ages = [32, 33, 16, 40];  
ages.filter(checkAdult) // Returns [32, 33, 40]  
  
function checkAdult(age) {  
  return age >= 18;  
}
```

# ARRAY METHODE CONCAT()

Zusammenführen von zwei Arrays (merge).

```
const hege = ["Cecilie", "Lone"];  
const stale = ["Emil", "Tobias", "Linus"];  
const children = hege.concat(stale);  
// Ausgabe: ['Cecilie', 'Lone', 'Emil', 'Tobias', 'Linus']
```

# AUFGABE 03 - 2

Schreibe ein Programm, dass die Zahlen in einem Array filtert, so dass nur gerade Zahlen ausgegeben werden.



Aufgabenstellung

# TRY...CATCH

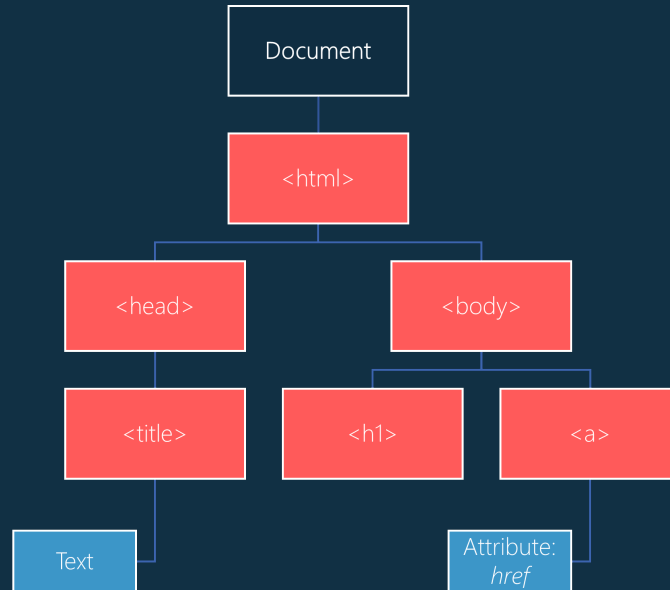
Block von Anweisungen bei dem im Fehlerfall eine gewünschte Reaktion ausgeführt wird.

```
try {  
    // Block of code to try  
}  
catch(err) {  
    // Block of code to handle errors  
}
```

Der Fehlerfall kann sowohl unvorhergesehen, als auch gewollt sein.

# HTML DOM

## Document Object Model





# HTML DOM

*Das Object Model beschreibt alle Objekte einer HTML-Datei mit Eigenschaften, Methoden und Events.*

*Zusätzlich kann JavaScript über eine API auf das Object Model zugreifen und Informationen auslesen sowie verändern.*

# JAVASCRIPT UND HTML DOM

Um über JavaScript auf HTML-Elemente zuzugreifen, müssen diese aus dem HTML DOM geladen werden.

```
const element = document.getElementById("element");  
element.innerText = "Das ist ein Text";  
  
document.getElementById("element").innerText = "Das ist ein Text";
```

# ELEMENTE LADEN

```
// Das erste gefundene Element wird zurückgeben.  
const a = document.getElementById("main");  
const b = document.getElementsByClassName("intro");  
  
// querySelectorAll() liefert ein Objekt mit Elementen zurück  
// Query entspricht dem CSS Selektor  
const c = document.querySelectorAll(".intro");  
const d = document.querySelector(".intro");
```

# QUERYSELECTORALL

querySelectorAll() gibt ein Objekt mit gefundenen Elementen zurück. Mit Hilfe der Methode forEach() lassen sich diese Elemente durchlaufen (Iteration) und nacheinander verarbeiten.

```
const elemente = document.querySelectorAll(".button");
elemente.forEach( function (element) {
    // Hier wird ein einzelnes Element verarbeitet
    console.log(element.innerText);
})
```

*Alternativ kann auch eine "normale" for-Schleife zur Iteration genutzt werden.*

# ATTRIBUTE UND METHODEN

HTML DOM Elemente (Objekte) haben verschiedenste Attribute und Methoden über die sich die Webseite manipulieren lässt.

```
document.getElementById(id).innerHTML = "Inner HTML";  
// Zugriff auf CSS Styles  
document.getElementById(id).style.color = "blue";  
// Zugriff auf Klassen eines Elements  
document.getElementById(id).classList;  
// Inhalt eines Input-Elements auslesen  
document.querySelector("input").value;
```

[https://www.w3schools.com/jsref/dom\\_obj\\_all.asp](https://www.w3schools.com/jsref/dom_obj_all.asp)

# CSS EIGENSCHAFTEN UND JAVASCRIPT

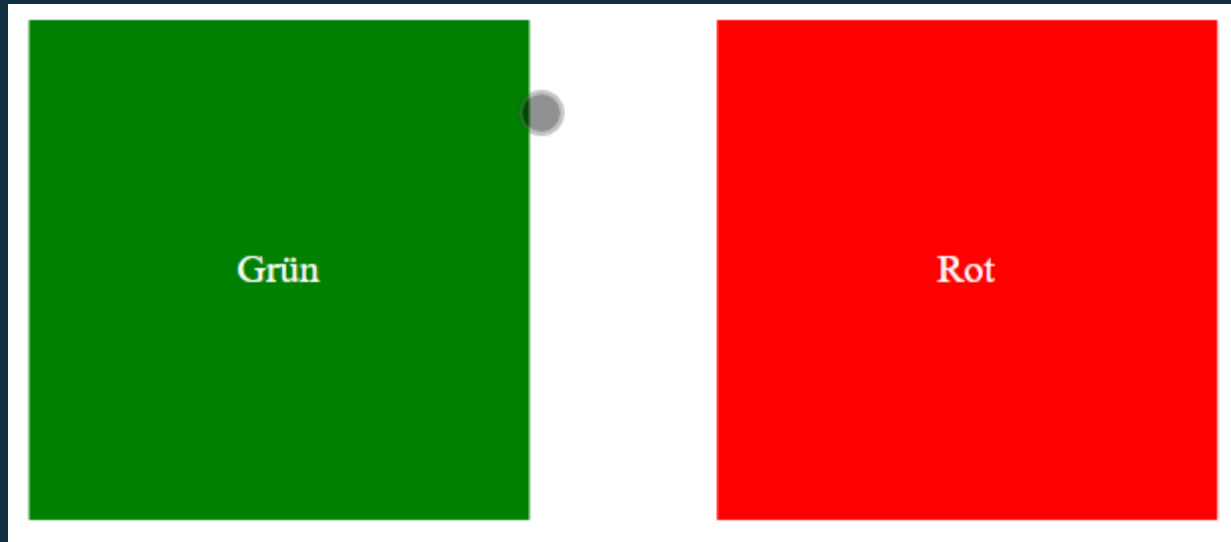
CSS Eigenschaften mit einem Bindestrich werden in JavaScript ohne Bindestrich geschrieben. Der folgende Buchstabe wird groß geschrieben.

```
// CSS background-color  
document.getElementById("first").style.backgroundColor = "red";
```

# AUFGABE 03 - 3

Verändere die Webseite mit Hilfe von JavaScript.

## Aufgabenstellung



# DESTRUCTURING

## *Extrahieren von Daten aus Arrays und Objekten*

```
// Extrahieren eines Arrays in zwei Variablen
let formen = ["🔴", "🟡"];
let [kreis, rechteck] = formen;
// kreis === "🔴"; rechteck === "🟡";

// Tauschen von Variablen
let a = 10; let b = 20;
[a, b] = [b, a]
// a === 20; b === 10;

// Überspringen von Elementen
var [, , third] = ["foo", "bar", "baz"];
// third === "baz";
```



# DESTRUCTURING

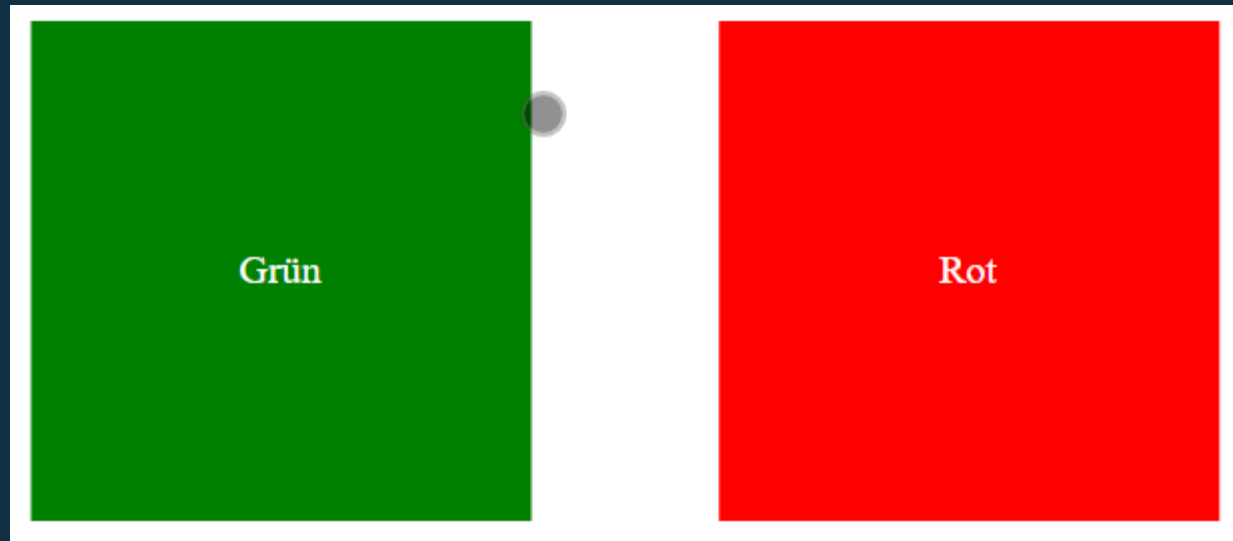
Werte eines Objekts können über den Namen extrahiert werden.

```
var { foo, bar } = { foo: "lorem", bar: "ipsum" };  
console.log(foo);  
// "lorem"  
console.log(bar);  
// "ipsum"
```

# AUFGABE 03 - 3

Wende das Destructuring beim Tausch der Texte an.

Aufgabenstellung



# REST- / SPREAD-OPERATOR

Einsammlung der restlichen bzw. extrahieren von Werten.

...

```
let rest;  
[a, b, ...rest] = [10, 20, 30, 40, 50];  
// rest === [30, 40, 50]
```

# SPREAD OPERATOR

Zusammenführen von Arrays (*anstelle von concat()*)

```
let einkaufsliste = ["Salat", "Gurke", "Wasser"];  
let neueListe = ["Zwiebeln", "Äpfel", "Brot"];  
  
console.log([...einkaufsliste, ...neueListe]);
```

# JAVASCRIPT UND HTML EVENTS

JavaScript kann HTML Events überwachen und bei einem Ereignis Funktionen aufrufen. *addEventListener(event, function)*

```
document.getElementById("myBtn").addEventListener("click", displayDat
```

Neben *Click* gibt es viele weitere Events:

[https://www.w3schools.com/jsref/dom\\_obj\\_event.asp](https://www.w3schools.com/jsref/dom_obj_event.asp)

# AUFGABE 03 - 4

## EINGABEN ANZEIGEN

Schreibe den Code, um über JavaScript die Eingabe des Input-Elements auf der Webseite darzustellen.

Aufgabenstellung

# INNERHTML

Nach Laden des Parent-Elements kann neuer HTML-Code über die innerHTML Eigenschaft integriert werden.

```
document.querySelector(".parent").innerHTML =  
    "<div class='child'>Text</div>";
```

# NEUE ELEMENTE ERZEUGEN

Über JavaScript lassen sich neue HTML Elemente erzeugen.

```
var neuesElement = document.createElement("div");  
neuesElement.innerText = "Hier wird ein Text hinterlegt"  
// Hinzufügen von Klassen über classList  
neuesElement.classList.add("Klasse1", "Klasse2")
```



# NEUE ELEMENTE EINFÜGEN

Erzeugte Element müssen anschließend noch an eine Stelle definiert Stelle des HTML DOM gehangen werden.

```
// Anhängen eines neuen Child-Elements an das Ende des Parent-Element  
document.querySelector(".parent").appendChild(neuesElement);  
  
// Einfügen vor dem geladenen Element  
document.querySelector(".parent").firstChild.insertBefore(neuesElement,
```

# AUFGABE 03 - 4

## EINGABEN POSTEN

Schreibe den JavaScript Code, um über HTML DOM die Eingabe des Input-Elements auf der Webseite darzustellen.

Aufgabenstellung

# AUFGABE 03 - ZUSATZ

## TRAVELBLOG ERGÄNZEN

Ergänze den Travelblog um einen FAQ Bereich und einen Darkmode Toggle Button.

Aufgabenstellung

# ECMAScript

ECMAScript ist der standardisierte Sprachkern von JavaScript.

- **Bedeutende Version:** 6th Edition – ECMAScript 2015  
*Klassen, Modules, for...of, Arrow Functions, let und const, promises, ...*
- **Aktuelle Version:** ECMAScript 2021 / ES12

# ARROW FUNCTIONS

Mit ES6 wurde eine neue kürzere Schreibweise für anonyme Funktionen eingeführt.

```
// Klassische Syntax
function (param1, param2) {
  // statements
}

// Arrow Function
(param1, param2) => {
  // statements
}
```

*Sofern nur ein Übergabeparameter vorhanden ist, können die runden Klammern weggelassen werden.*

# AUFGABE (5)

Verändere die vorliegende Funktion in eine anonyme Arrow-Function, welche über *square()* aufgerufen werden kann.

```
function square(number) {  
  return number * number;  
}  
  
console.log(square(2));
```

# BUNDELING

*Bundeling ist der Prozess, bei dem mehrere JavaScript-Dateien zu einer einzigen Datei zusammengefasst werden. Dieser Prozess wird von einem sogenannten Bundler durchgeführt. Ein Bundler ist ein Tool, das mehrere separate Module in ein einziges großes Modul (den sogenannten Bundle) zusammenfasst. Dieses Bundle kann dann in einer HTML-Datei eingebunden werden.*


# BUNDELING MIT TYPE="MODULE"

*Mit dem Attribut `type="module"` kann eine JavaScript-Datei als Modul definiert werden.*

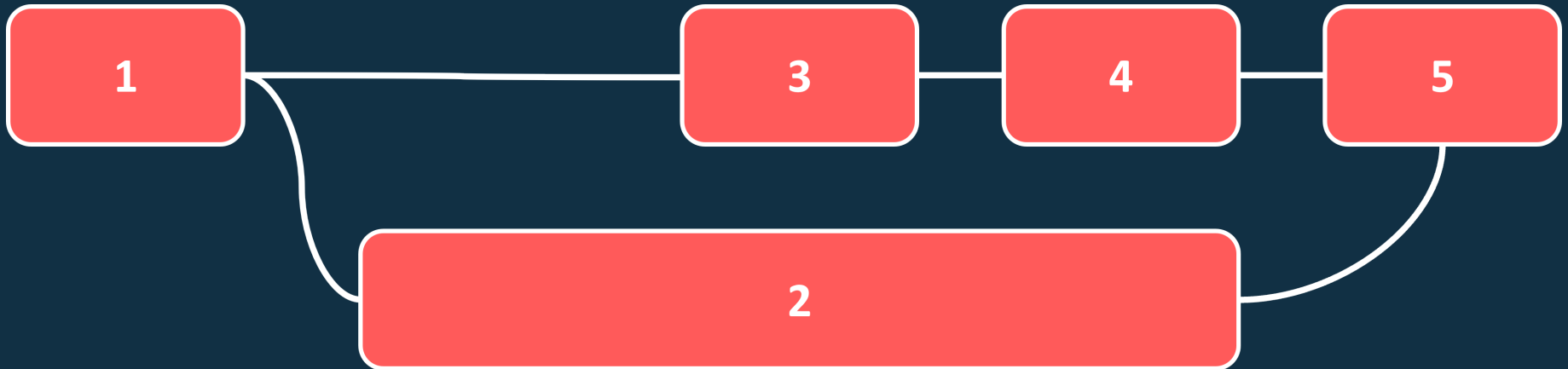
*Die Datei muss über einen Webserver aufgerufen werden.*



# ZUSATZ

Im Ordner JavaScript befinden sich zusätzliche  
Übungsaufgaben. 

# ASYNCHRONE PROGRAMMIERUNG



# SYNCHRONE PROGRAMMIERUNG

*Mit Ausführung jeder Codezeile liegt das  
Ergebnis unmittelbar vor.  
Synchrone Kommunikation*

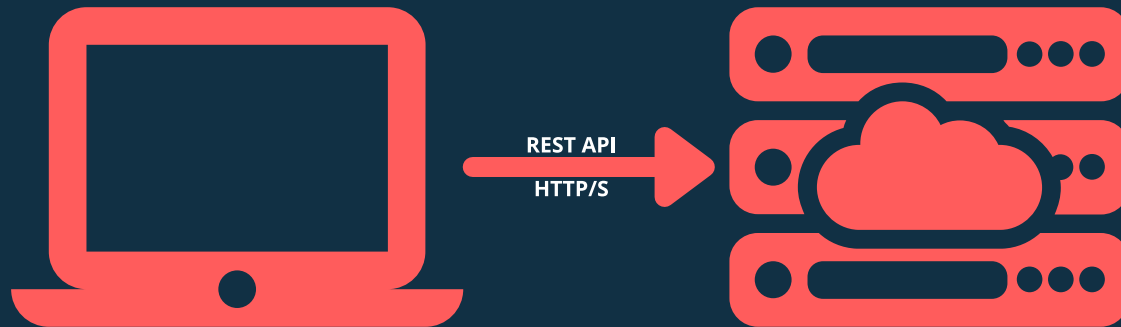
```
console.log("line one");  
console.log("line two");  
console.log("line three");
```

# ASYNCHRONE PROGRAMMIERUNG

*Ergebnisse einer Codezeile liegen nicht  
unmittelbar mit Ausführung vor.  
Asynchrone Kommunikation*

```
console.log("line one");
setTimeout(() => {
  console.log("Ich erscheine erst nach 2 Sekunden.");
}, 2000)
console.log("line two");
console.log("line three");
```

# ASYNCHRONE PROGRAMMIERUNG EINER FULL STACK WEB APP



Sowohl der **API Aufruf** des Frontends, als auch die **Interaktion mit Datenbanken** im Backend sind asynchrone Funktionen.

# ASYNCHRONE PROGRAMMIERUNG

Wichtig: Wir müssen kaum selbst Promise Funktionen schreiben, sondern wir müssen wissen wie mit diesen Objekten umzugehen ist und wie diese verarbeitet werden können (Promise-based Apis).

# ASYNCHRONE PROGRAMMIERUNG

## PROBLEMSTELLUNG

*Ergebnis einer asynchronen Funktion verarbeiten*

```
// Funktion zur Demonstration einer asynchronen Aufgabe
// i.d.R. haben wir keinen Einfluss auf die Funktion
// -> console.log() innerhalb der Funktion ist nicht möglich
function loginUser(username, password) {
  setTimeout ( () => {
    return {userEmail : "kontakt@codingschule.de"}
  }, 1500)
}

const user = loginUser("Codingschule", "123456");
console.log(user);
```

# CALLBACK FUNCTION

Eine Callback Funktion ist eine Funktion, die als Übergabeparameter an eine andere Funktion übergeben wurde und dort aufgerufen wird.

```
function berechne(zahl1, zahl2, rueckruffunktion) {  
    return rueckruffunktion(zahl1, zahl2); }
```

```
function berechneSumme(zahl1, zahl2) {  
    return zahl1 + zahl2; }
```

```
function berechneProdukt(zahl1, zahl2) {  
    return zahl1 * zahl2; }
```

```
// Gibt 20, die Summe von 5 and 15, aus  
alert(berechne(5, 15, berechneSumme));  
// Gibt 75, das Produkt von 5 und 15, aus  
alert(berechne(5, 15, berechneProdukt));
```



# CALLBACK BEISPIELE

Callback Funktionen spielen bei vielen, im JavaScript Sprachkern, genutzten Funktionen eine wichtige Rolle.

```
// Übergabe einer Funktion die nach 1 Sekunde aufgerufen wird.  
setTimeout(meineFunktion, 1000);  
  
// Übergabe einer Funktion beim Click-Event aufgerufen wird.  
document.getElementById("myBtn").addEventListener("click", displayDat
```

*auch asynchrone Funktionen*

# CALLBACKS UND ASYNCHRONE PROGRAMMIERUNG

```
function loginUser(username, password, callback) {  
  setTimeout ( () => {  
    callback ({userEmail : "kontakt@codingschule.de"})  
  }, 1500)  
};  
  
function ausgabe(user) {  
  console.log(user)  
}  
  
const user = loginUser("Codingschule", "123456", ausgabe);
```

# CALLBACKS UND ASYNCHRONE PROGRAMMIERUNG

```
function loginUser(username, password, callback) {  
  setTimeout ( () => {  
    callback ({userEmail : "kontakt@codingschule.de"})  
  }, 1500)  
};  
  
// mit anonymer Arrow Function  
const user = loginUser("Codingschule", "123456", (user) => {  
  console.log(user);  
});
```

# VERSCHACHTELUNG VON CALLBACKS

```
function loginUser(username, password, callback) {
  setTimeout (() => {
    callback ({userEmail : "kontakt@codingschule.de"});
  }, 1500)
}
function getUserDetails(userEmail, callback) {
  setTimeout (() => {
    callback ({userPostcode : "40476"});
  }, 1500)
}
const user = loginUser("Codingschule", "123456", (user) => {
  console.log(user);
  getUserDetails(user.userEmail, (userProfile) => {
    console.log(userProfile);
  });
});
```

# ES6 PROMISES

Promises sind Objekte, die das Ergebnis einer asynchronen Aktion abfangen und es zurück geben.

```
const result = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve({user: "marc"});
    //reject("Fehler");
  }, 2000)
});

result
  .then(user => {console.log(user)})
  .catch(err => console.log(err));
```

Mit *.then()* und *.catch()* können die Ergebnisse verarbeitet werden.

# ES6 PROMISES

```
function loginUser(username, password) {  
  return new Promise((resolve, reject) => {  
    setTimeout (() => {resolve ({userEmail : "kontakt@codingschule.de"  
      1500)  
    })}  
  })  
}  
  
function getUserDetails(userEmail) {  
  return new Promise((resolve, reject) => {  
    setTimeout (() => {resolve ({userPostcode : "40476"});}, 1500)  
  })  
}  
  
loginUser("Codingschule", "123456")  
  .then ( user => getUserDetails(user.userEmail))  
  .then ( profile => console.log(profile))
```

*then() wird ausgeführt, wenn das Ergebnis des Promise vorliegt.*

# ASYNC & AWAIT

Seit ES8 bietet JavaScript eine weitere Methode um asynchrone Operationen (Promises) zu verarbeiten. *await* erzeugt eine Funktion die auf ein Promises wartet.

```
const result = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    resolve({user: "marc"});  
    //reject("Fehler");  
  }, 2000)  
});  
  
async function showResult () { console.log(await result);}  
showResult()
```

*await* kann nur in *async* Funktionen verwendet werden.

# ASYNCHRONE FUNKTIONEN

Asynchrone Funktionen laufen außerhalb des üblichen Kontrollflusses und geben als Ergebnis ein Promise Objekt zurück.

*Schlüsselwort: `async`*

```
async function getUser() {  
  const user = await loginUser("Codingschule", "123456")  
  console.log(user);  
}
```



# AUFGABE (5)

Passe den vorliegenden Code auf *await* an.

Aufgabenstellung

# AJAX

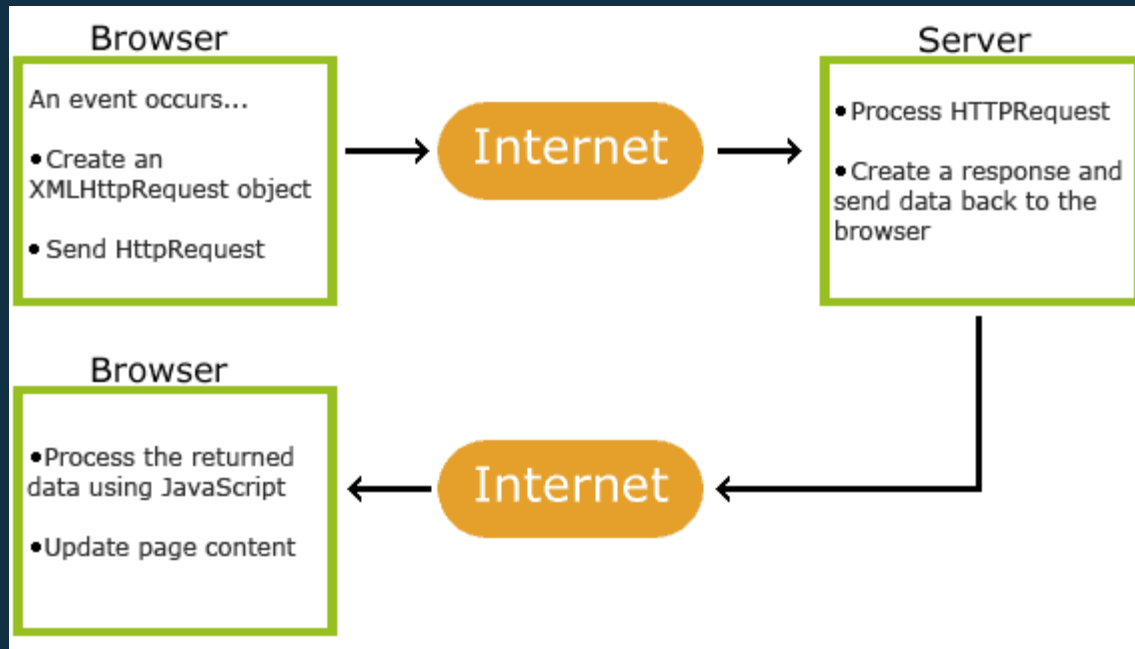
Asynchronous JavaScript And XML

Konzept zur asynchronen Datenübertragung zwischen  
Browser und Server.

*Nachladen* von Inhalten einer Webseite über JavaScript.

*XML = Extensible Markup Language*

# AJAX



# AJAX

## AJAX über XMLHttpRequest

```
function loadDoc() {  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            document.getElementById("demo").innerHTML = this.responseText;  
        }  
    };  
    xhttp.open("GET", "ajax_info.txt", true);  
    xhttp.send();  
}
```

# JAVASCRIPT FETCH API

Abfragen von APIs mit Promises

`fetch()` ruft eine REST-API über HTTP auf und erzeugt automatisch ein **Promise**-Objekt.

```
// then()
fetch("https://127.0.0.1/api/user")
.then(result => console.log(result));

// Await
const result = await fetch("https://127.0.0.1/api/user");
```

# FETCH API PARAMETER

Neben der aufzurufenen URL können viele weitere Informationen im HTTP-Request definiert werden.

```
fetch("https://127.0.0.1/api/user", {  
  method: 'POST',  
  headers: {  
    'Content-Type': 'application/json',  
  },  
  body: JSON.stringify(update),  
});
```

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

# FETCH API UND JSON

Aus einer Fetch API Rückmeldung kann der JSON-Body geladen werden (parsen).

```
const data = await result.json();
```

Da `json()` auch ein Promise zurückliefert, muss der Aufruf ebenfalls mit `await` erfolgen.

# AUFGABE (6)

Über die Open Weather Map API lassen sich weltweite Wetterdaten abfragen und auf einer eigenen Webseite darstellen.

## Aufgabenstellung





# ES6 MODULE

Seit ES6 lassen sich in JavaScript Module nutzen, um Programmcode in anderen Dateien nutzbar zu machen. JavaScript nutzt hierfür die beiden Schlüsselwörter **import** und **export**.



# ES6 MODUL - DEFAULT EXPORT

```
let Car = {};
```

```
export default Car;
```

```
import irgendeinName from './app.js'
```

```
// Der Name des Imports gilt nur in dieser Datei
```

# ES6 MODUL - NAME EXPORT

```
let Car = {};
```

```
export Car;
```

```
import {Car} from './app.js'
```

# ES6 MODUL - LIST EXPORT

```
...  
export {Car, farbe};
```

```
import {Car, farbe as color} from './app.js'  
// Bezeichner von Export und Import können auch verändert werden.
```

# BABEL

Transcompiler um ECMAScript 2015+-Code in eine  
abwärtskompatible Version umzuwandeln.

<https://babeljs.io/>