Javascript

Grundlagen

Funktionale Programmierung

“Funktionale Programmierung ist ein Programmierparadigma, innerhalb dessen **Funktionen** nicht nur definiert und angewendet werden können, sondern auch **wie Daten miteinander verknüpft, als Parameter verwendet und als Funktionsergebnisse auftreten können**.”

aus Wikipedia

Historie

Javascript (ursprünglich LiveScript) wurde von Netscape erstellt um dynamisch HTML und CSS-Stylings zu erstellen bzw. zu manipulieren.

Heute kommt Javascript im Browser aber immer stärker auch im Backend sowie Embedded-Bereich zur Anwendung.

Historie - Überblick

ECMAScript Editions

Year	Name	Description
1997	ECMAScript 1	First Edition.
1998	ECMAScript 2	Editorial changes only.
1999	ECMAScript 3	Added Regular Expressions. Added try/catch.
	ECMAScript 4	Was never released.
2009	ECMAScript 5	Added "strict mode". Added JSON support.
2011	ECMAScript 5.1	Editorial changes.
2015	ECMAScript 6	Added classes and modules.
2016	ECMAScript 7	Added exponential operator (**). Added Array.prototype.includes.

Quelle: w3schools.com

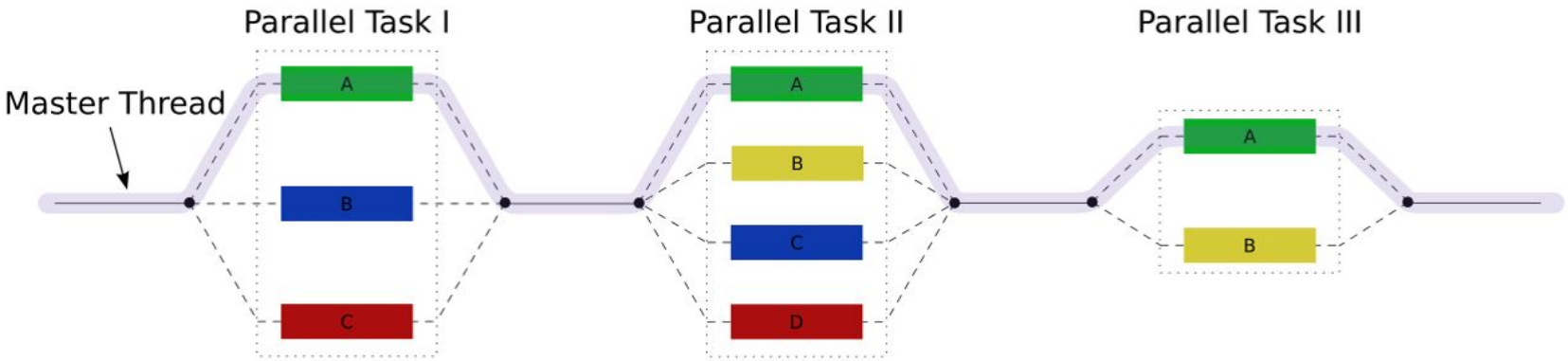
Verwendete Paradigmen & Auswirkung

Javascript vereinigt unterschiedliche Programmierparadigmen.

- Funktionale Programmierung
- Prozedurale Programmierung
- Objektorientierte Programmierung

In Javascript ist es möglich sowohl Klassen mit deren Methoden und Attributen zu vererben (vertikal) als auch einzelne Methoden unter den Objekten zu teilen (horizontal)

JavaScript - Ausführung u. Interpreter



Vorbereitung // IDE

// Visual Code

https://code.visualstudio.com/

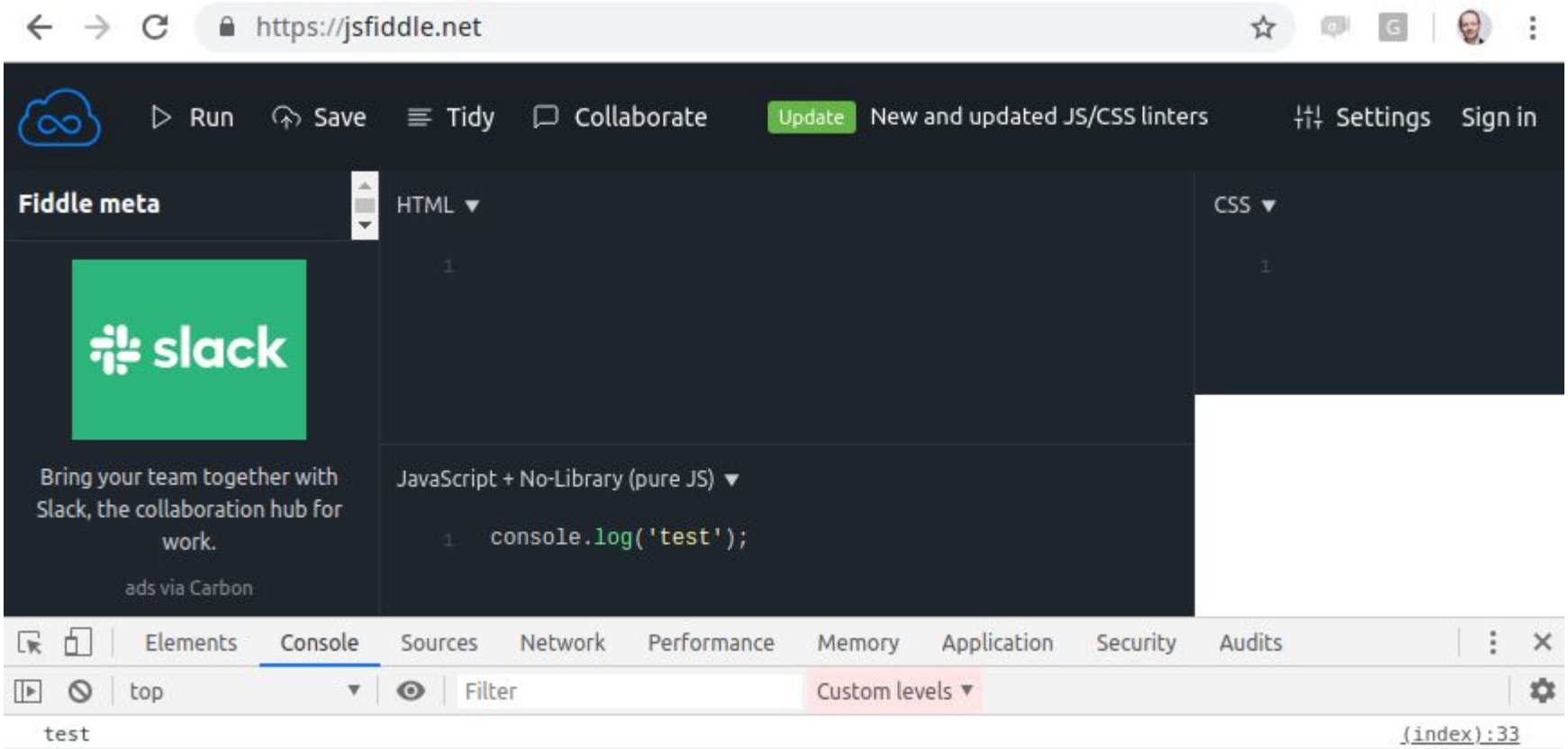
// NodeJS

https://nodejs.org/en/download/package-manager/

// JSFiddle

https://jsfiddle.net/

Vorbereitung // Debug-Ausgabe mit console.log



Der Aufbau von Skripten // Per Tag

//Im HTML Umfeld

// 1. Entweder innerhalb des <script>-Tags

```
<script>

  console.log('Hallo Welt!');

</script>
```

// 2. Oder als Script-Referenz

```
<script src="my.js"></script>
```

Datentypen // Variablen

// Javascript ist eine schwach bzw. dynamisch typisierte Programmiersprache

// Alle Variablen beginnen werden mit dem Schlüsselbegriff var eingeleitet

```
var test = 'hallo';

var test = 123;

var test = false;
```

// String-Variable können mit ' oder " eingeleitet werden

```
var test = 'hallo' + " rafael";
```

Datentypen // Überblick

Primitive Datentypen sind:

- boolean
- null
- undefined
- number
- string

Objekte sind komplexe bzw. zusammengesetzte Datentypen (Array ist auch ein Objekt):

- object
- function

Datentypen // Date-Objekt

```
// Aktuelles Datum erstellen
var current_date = new Date();
console.log(current_date);

// Spezifisches Datum erstellen (1. Januar 2012)
var second_date = new Date(2012, 0, 1);
console.log(second_date);

// Abstand zwischen zwei Datumangaben errechnen
var milliseconds = current_date.getTime() - second_date.getTime();
console.log(milliseconds/(1000*60*60*24*365));
```

Datentypen // function I

```
// Wie werden Funktionen deklariert

function name(vorname, nachname) {
   return vorname + " " + nachname;
};

var zusammengesetzter_name = name("Rafael", "Sobek");

console.log(zusammengesetzter_name);
```

Datentypen // function II // Unterschiede

```
// Deklarative Funktion

function addiere1(a,b) {
  return a + b;
}

//Anonyme Funktion ohne Namen

var addiere2 = function(a,b) {
  return a + b;
}

console.log(addiere1(1,2), addiere2(1,2));
```

Datentypen // function III // Callback

```
// Erwartet eine Function

function anfrage(antwort) {
  return "Hallo wer bist Du -> " + antwort();
}

// Wird verwendet
function antwort() {
  return "Ich bin es. Rafael. Wie gehts?";
}

console.log(anfrage(antwort));
```

Datentypen // function IV

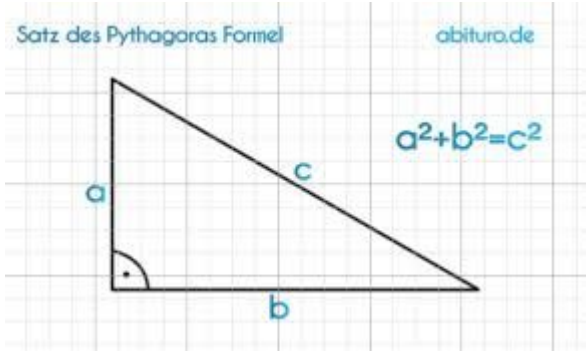
// Javascript ist eine funktionale Programmiersprache (eigentlich auch prozedural und objektorientiert)
// Das heißt Funktionen können als Variablen deklariert werden.

```
var f1 = function(otherFunction) {
   return otherFunction() + 1;
};
var f2 = function(a, b) {
   return function() {
       return a + b;
   }
};

console.log(f1(f2(1,2)));
```

Aufgabe I - Einfache Funktion erstellen

Erstellen sie eine Funktion welche mit Hilfe des Satzes von Pythagoras die Hypothenuse errechnet (Wurzelfunktion Math.sqrt) und ausgibt.



$$c = \sqrt{a^2 + b^2}$$

aus Abituro.de

Datentypen // function V // Rekursion

```
// Eine Funktion, welche sich selber aufruft
var add = function(number, max_iterations, idx) {

   var result = number + idx;
   if (idx < max_iterations) {
      return add(result, max_iterations, ++idx);
   } else {
      return result;
   }
};

console.log(add(0, 3, 0)); // 0 + 1
```

Datentypen // object

```
var person_object = {
   name: "Rafael Sobek",
   adresse: {
       stadt: "Karlsruhe”
   }
};

console.log(person_object);

person_object.adresse.plz = 76135;

console.log(person_object);
```

Datentypen // object II

```
//Objekte können Funktionen enthalten

person_object.eine_funktion = function() {
   console.log(this);
};

person_object.eine_funktion();
```

Aufgabe II - Funktion Alter berechnen

1. Erweitern Sie das person_object um ein Geburtsdatum.
2. Erweitern Sie das person_object um eine Funktion, welche das Alter der Person errechnet.

Datentypen // array I

```
// Wie initialisiere und befülle ich ein Array

var array_1 = [1,2,3,4,5,6];

console.log(array_1);

var array_2 = [1, "Rafael", {rechnung: {id: 1}}];

console.log(array_2[0], array_2[1], array_2[2]);
```

Datentypen // array II // Funktionen

```
// Filtern

var zahlen_fuer_filter = [1,2,3,-1,-2,6];

var filtered = zahlen_fuer_filter.filter(function(item) {
   if (item > 2) {
       return true;
   } else {
       return false;
   }
});
console.log(filtered);
```

Datentypen // array III // Funktionen

```
// Sortieren

var zahlen_fuer_sortieren = [-1,1,3,-2,6,2];

var sorted = zahlen_fuer_sortieren.sort(function(a,b) {
   return a - b;

});

console.log(sorted);
```

Datentypen // array IV

```
// Wie durchlaufe ich ein Array

var mein_array = ["Michael", "Melanie", "Stefan"];

mein_array.forEach(function(name) {
   console.log(name);
});
```

Aufgabe III - Elemente addieren

1. Erstellen Sie ein Array mit mehreren Zahlen
2. Bilden Sie die Summe aller Zahlenelemente im Array

Aufgabe IV - Häufigkeit berechnen u. Eindeutigkeit

1. Erstellen Sie eine Funktion, welche die Häufigkeit aller Zahlen innerhalb eines Arrays errechnet bspw. für [1, 1, 1, 2, 2, 3] => {“1”: 3, “2”: 2, “3”: 1}

2. Erstellen Sie eine Funktion, welche aus einem Eingabearray ein Ausgabearray erstellt welches keine Duplikate mehr enthält bspw => [1,2,3].

Datentypen // array III

// Arrays sind auch Objekte
// Eine Unterscheidung kann nicht mit Hilfe der "typeof"-Funktion erfolgen
// Hierfür gibt es eine Hilfsmethode in der Klasse Arrays

```
var a = [1,2,3];
var b = {a: 1, b: 2};

console.log(typeof(a));
console.log(typeof(b));

console.log("Ist a ein Array -> " + Array.isArray(a));
```

Aufgabe V - Durchschnittsalter berechnen

1. Erstellen Sie ein Array mit mehreren Personen-Objekten mitsamt Geburtsdatum und Funktion zur Berechnung des Alters.
2. Durchlaufen Sie alle Personen-Objekte und berechnen Sie das Durschnittsalter der enthaltenen Personen.

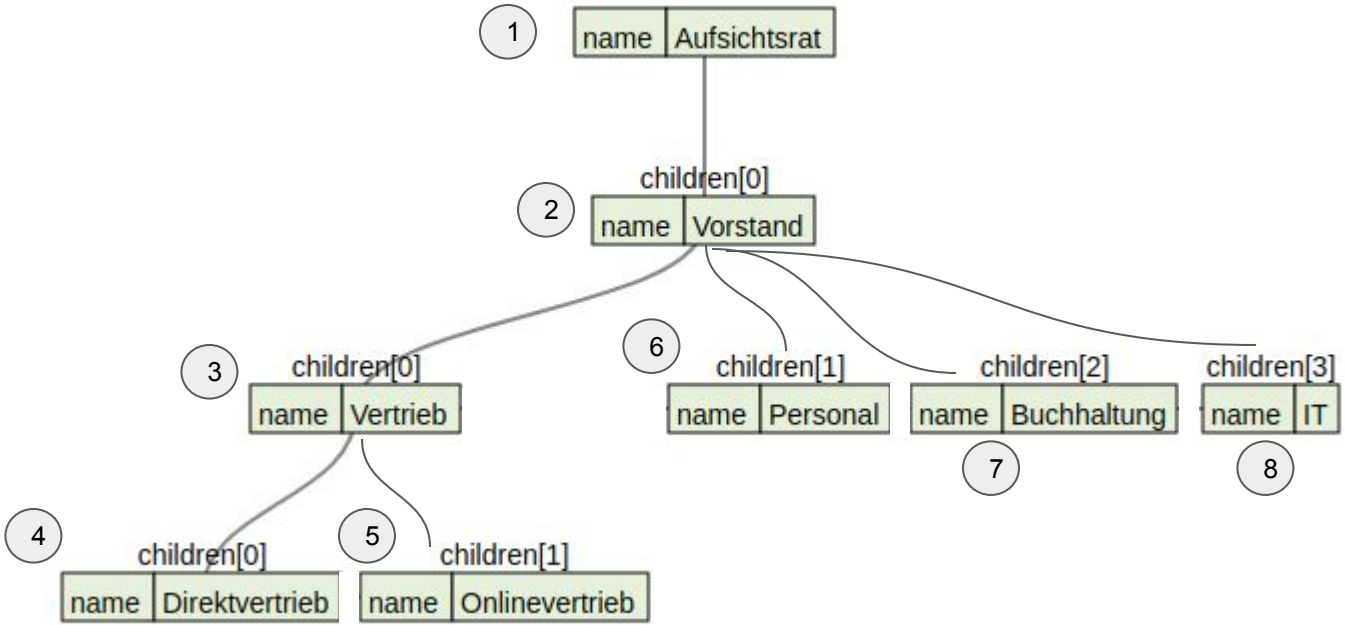
Aufgabe VI - Einen Baum traversieren

1. Erstellt eine JSON-Hierarchie

```
var orga = {
    name: 'Aufsichtsrat',
    children: [
        {
            name: 'Vorstand',
            children: [
                {
                    name: 'Vertrieb',
                    children: [
                        { name: 'Direktvertrieb' },
                        { name: 'Onlinevertrieb' }
                    ]
                },
                { name: 'Personal' },
                { name: 'Buchhaltung' },
                { name: 'IT' }
            ]
        }
    ]
}
```

Aufgabe VI - Einen Baum traversieren

2. Implementieren sie eine rekursive Top-Down-Traversierung. Geben Sie dabei die Namen per console.log aus.



Aufgabe VI - Einen Baum traversieren

3. Erweitern Sie die Rekursion um eine weitere Methode, welche übergeben wird und die pro Rekursionsschritt ausgeführt werden kann.

=> Struktur:

```
function(currentNode) {
    //my code
}
```

Bsp: function(currentNode) {console.log(currentNode.name)};

Aufgabe VI - Einen Baum traversieren

4. Erweitern Sie die Rekursion so, dass sie jeweils auch die Ebene/Level ermitteln bzw. auslesen können

=> Struktur der Rekursionsmethode kann dann wir folgt erweitert werden:

```
function(currentNode, level) {
    //my code
}
```

Bsp:

```
function(currentNode, level) {
   console.log(currentNode.name + ‘ on level ’ + level)
};
```

Aufgabe VI - Einen Baum traversieren

5. Erstellen Sie eine Rekursions-Methode, welche jedem Namen ein “LV 1871 - ” voranstellt .

Datentypen // Programmatische Typüberprüfung

// Die Typüberprüfung erfolgt mit Hilfe der "typeof"-Funktion:

```
var a = 1;

if (typeof(b) === "undefined") {
    out("b existiert nicht");
}

out("a ist vom Datentyp -> " + typeof(a));
```

Fehlerbehandlung // try-catch-Block

```
// Mit Hilfe des "try-catch"-Blocks können Fehler gefangen
werden

try {
   console.log(existiere_nicht);
} catch(e) {
   console.log('Variable ist unbekannt');
}
```

Datentypen // globale Variablen

// globale Variablen sind Variable ausserhalb aller Blöcke
// oder Variable die in Blöcken ohne "var"-Schlüsselbegriff deklariert werden

```
var a = 1; // ich bin eine globale Variable
b = 2; // ich bin eine globale Variable

var f1 = function() {
  var c = 3;
  d = 4;
  out('ich sehe folgende vars: ', a, b, c, d);
};
f1();
out('ich sehe folgende vars: ', a, b, typeof(c), d);
```

Kontrollstrukturen // if

```
var a = 1;

if (a > 0) {
    console.log("a ist größer als 0");
} else if (a === 0) {
    console.log("a ist gleich 0");
} else {
    console.log("a ist gleich 0");
}
```

Kontrollstrukturen // switch - Block

```
var a = 'INITIAL';

switch (a) {
  case 'INITIAL':
     out('Initialzustand');
     break;
  case 'ENDING':
     out('Beendigung');
     break;
  default:
     out('Unbekannter Zustand');
     break;
}
```

Schleife // for und while-Schleife

```
// Die For-Schleife
for (var i = 0; i < 10; i++) {
    out('idx ist ' + i);
}

// Die While-Schleife
var i = 0;
while (i < 10) {
   i = i + 1;
   out(i);
}
```

Schleife // Lose vs. strikte Gleichheit

```
// Lose Gleichheit mit ==

// Beispiel mit String u. Zahl

out(0 == "0");


// Strikte Gleichheit mit ===

out(0 === "0");
```

Functionen // Callback und Asynchrone Functions I

```
// Führe Funktion aus wenn Zeit abgelaufen
var seconds = 2; // 2s
setTimeout(function() {
    out('Angekommen nach ' + seconds + ' secs.');
}, 2 * 1000);

// Führe alle 2 Sekunden eine Funktion aus
setInterval(function() {
    out('Komme wieder nach ' + seconds + ' secs.');
}, 2 * 1000);
```

Sonnstiges // Arguments Object

```
// Auf Argumente programatisch zugreifen

function argumentsFunc(a, b, c) {
  out(arguments[0]);
  out(arguments[1]);
  out(arguments[2]);
}

argumentsFunc(1, 2, 3);
```

DOM Funktionen // getElementById

```
// Im HTML Editor

<div id="meine_id">test</div>

// Elemente finden per ID

var elem = document.getElementById("meine_id");

// Inhalt ausgeben

console.log(elem.textContent);
```

DOM Funktionen // getElementsByTagName

```
// Im HTML Editor

<mein_tag>tag1</mein_tag>
<mein_tag>tag2</mein_tag>

// Elemente finden per ID

var elems = document.getElementsByTagName("mein_tag");
elems = Array.from(elems);

// Inhalt ausgeben
elems.forEach(function(elem) {
    console.log(elem.textContent);
});
```

DOM Funktionen // getElementsByClassName

```
// Im HTML Editor

<div class="meine_klasse">test1</div>
<div>test2</div>
<div class="meine_klasse">test3</div>

// Elemente finden per ID

var elems = document.getElementsByClassName("meine_klasse");

// Inhalt ausgeben

console.log(elems.length);
```

DOM Funktionen // document.querySelectorAll I

```
ACHTUNG: querySelector gibt nur ein Element zurück
// Elemente in HTML selektieren

1. Erstellen Sie folgendes HTML
<section>
   <div class="meine_klasse">
      test1
      <span>test</span>
   </div>
   <button>test2</button>
   <span>huhu</span>
   <div id="meine_id">test3</div>
   <input type="text"></input>
</section>
```

DOM Funktionen // document.querySelectorAll II

```
// Elemente in HTML selektieren
2. Erstellen Sie folgende Selektoren

2.1 Elemente finden per ID

var elems = document.querySelectorAll("#meine_id");
console.log(elems, elems.length);

2.2 Elemente finden per Tag

var elems = document.querySelectorAll("span");
console.log(elems, elems.length);

2.3 Elemente finden per Klasse

var elems = document.querySelectorAll(".meine_klasse");
console.log(elems, elems.length);
```

DOM Funktionen // document.querySelectorAll III

```
// Elemente in HTML mit Hilfe von Pfaden selektieren bzw. eingrenzen
4. Erstellen Sie folgende Selektoren

4.1 Elemente finden mit Hilfe von Leveling Pfaden

var elems = document.querySelectorAll("section .meine_klasse span");
console.log(elems, elems.length);

4.2 Children Elemente referenzieren

var elems = document.querySelectorAll("section > *");
console.log(elems, elems.length);
```

DOM Funktionen // Attribute setzen

```
// Elemente in HTML mit Hilfe von Pfaden selektieren bzw. eingrenzen
5. Attribute setzen

5.1 Elemente finden

var elems = document.querySelectorAll("section span");

console.log(elems, elems.length);

5.2 Schriftfarbe auf rot setzen

for (var i = 0; i < elems.length; ++i) {
  elems[i].setAttribute("style", "color:red;");
}
```

DOM Funktionen // Eventhandler (z.B. Mouse Click)

```
// Elemente in HTML mit Hilfe von Pfaden selektieren bzw. eingrenzen
6. Attribute setzen

6.1 Elemente finden

var elems = document.querySelectorAll("section span");

console.log(elems, elems.length);

6.2 Schriftfarbe auf rot setzen

for (var i = 0; i < elems.length; ++i) {
  elems[i].addEventListener("click", function() {
     alert("es funktioniert");
  });
}
```

DOM Funktionen // innerHTML

```
// Elemente in HTML mit Hilfe von Pfaden selektieren bzw. eingrenzen
7. HTML ersetzen

7.1 Elemente finden

var elems = document.querySelector("section span");

console.log(elems, elems.length);

7.2 Bold-Text erstellen

for (var i = 0; i < elems.length; ++i) {
 elems[i].innerHTML = "<b>" + elems[i].textContent + "</b>";
}
```

DOM Funktionen // Formulare

```
// Elemente in HTML mit Hilfe von Pfaden selektieren bzw. eingrenzen
8. Elemente löschen

8.1 Elemente finden

var elems = document.querySelector("section span");

console.log(elems, elems.length);

8.2 Löschen

for (var i = 0; i < elems.length; ++i) {
 var element = elems[i];
 element.parentNode.removeChild(element);
}
```

DOM Funktionen // Formulare

```
// Elemente in HTML mit Hilfe von Pfaden selektieren bzw. eingrenzen
9. Eingabe tracken

8.1 Elemente finden

var elems = document.querySelector("section input");

console.log(elems, elems.length);

8.2 Löschen

for (var i = 0; i < elems.length; ++i) {
 var element = elems[i];
 element.addEventListener("keydown", function(event) {
     console.log("Ausgabe: ", event.target.value);
 });
}
```

Aufgaben

Bitte löst die nachfolgenden

1. Alle Links auf der lv1871 Seite sollen auf sueddeutsche.de verlinken.
2. Löscht bitte den Werbebanner auf der Seite und ersetzt diesen durch ein neues Bild
3. Beim Klick auf das Bild soll ein Alert Dialog mit 'Willkommen' angezeigt werden.