Schulung

JavaScript

Michael Brüggemann

Allianz

JavaScript

- Script-Sprache für Webseiten (seit 1995)
- Auch als ECMAScript bekannt, mittlerweile jährlichen Versionen
- Just in time (JIT) Kompilierung
- Ausführung in einer Sandbox je Browser-Tab bzw. iFrame

Einbindung

- Inline in einer HTML Datei

```
<script>
  console.log("Hello World!");
</script>
```

- Aus einer extra Datei

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

- Per Kommandozeile

```
> node logic.js
```

Sprach Basics - Variablen

- Sichtbarkeit abhängig von der Art der Definition
- Variable kann den Datentyp ändern (nicht empfohlen!)

```
// visible in the scope of the function
function() {
  // global
  window.varGlobal = 'hello world';

  // local
  let varLocal = 'hello again';
  varLocal = 4;
  varGlobal2 = 'still global';
  const local = 3;
}
```

Sprach Basics - Datentypen

```
undefined  // Undefined: typeof y === "undefined"
x = true;  // Boolean: typeof x === "boolean"
x = 1;     // Number: typeof x === "number"
x = 'a';   // String: typeof x === "string"
x = {};    // Object: typeof x === "object"

x = function() {}; // Function: typeof x === "function"
```

Sprach Basics - Funktionen

```
// Traditional Function
function (a){
  return a + 100;
}

// Arrow Function Break Down
// 1. Remove the word "function" and place arrow between the argu
ment and opening body bracket
(a) => {
  return a + 100;
}

// 2. Remove the body brackets and word "return" --
 the return is implied.
(a) => a + 100;

// 3. Remove the argument parentheses
a => a + 100;
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Zugriff auf den DOM

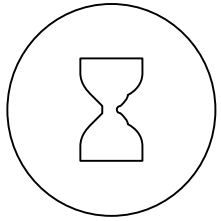
1. Element z.B. per querySelector / querySelectorAll selektieren
   - erwartet einen CSS Selektor
2. Attribute / Methoden des Elements nutzen

```
const myHeading = document.querySelector("h1");
myHeading.textContent = "Hello world!";

myButton.addEventListener(
  "click",
  (event) => console.log("clicked")
);
```

Aufgabe 1

Erstellt ein Eingabefeld und einen Button.
Bei klick auf den Button soll die Eingabe auf der Konsole ausgegeben werden.

Legt dazu in src/01-helloWorld/vorlage/ eine Datei index.html und logic.js an.



15 Minuten

Tools - Parcel

- https://parceljs.org/ ist ein „web application bundler“
- Fast einzelne Dateien in einer zusammen (.js, .css)
- „cache buster“ Dateinamen (z.B. logic.587b78d9.js)
- JavaScript Kompilierung für altere Versionen / Browser

  alternative in package.json: `"browserslist": ["last 1 Chrome version"]`
- Automatischer reload bei Änderungen

=> Vereinfachter Entwicklungsprozess und Auslieferung

```
> npm install -d parcel-bundler
> npm run parcel ./src/02-Taschenrechner/vorlage/index.html
```

Aufgabe 2

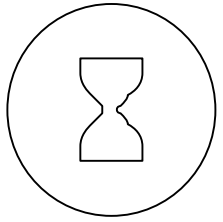
Erstelle einen „mini“ Taschenrechner (plus / minus reicht)

- 2 Eingabefelder
- 2 Button für plus / minus
- Ergebnisausgabe

- Installiere parcel

- Nutze die Vorlage in: Aufgabe_2/vorlage

- Starte dev-Server per: `npm run parcel ./Aufgabe_2/lösung/index.html`



15 Minuten

Klassen

- Mittlerweile gibt es in JavaScript auch Klassen
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(`${this.name} makes a noise.`);
  }
}
```

ES6 Module

- Aufteilung von Logik in einzelne Dateien
- Eigener Namespace je Datei
- export definierter Objekte
- Import unter anderem Namen möglich

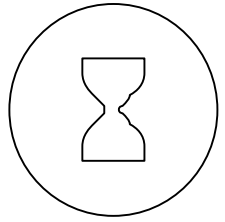
```
export class CoolLogic {
  ... several methods doing stuff ...
}

// This helper function isn't exported.
function resizeCanvas() {
  ...
}
```

```
Import {CoolLogic} from "CoolLogic.js";
```

Aufgabe 3

Baue den Taschenrechner auf eine Klasse Calculator und export/import um.

- *Kopiere die Lösung von Aufgabe 2 als Vorlage*



15 Minuten

Promises

- „Versprechen auf ein Ergebnis in der Zukunft“
- Für Asynchrone Aktionen (z.B. Netzwerkzugriffe)
- promise.then(), promise.catch(), promise.finally()
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

```
myPromise
  .then(handleResolvedA)
  .then(handleResolvedB)
  .catch(handleRejectedAny);
```

async / await

- async markiert eine Funktion als asynchron (= return Wert ist ein Promise)
- await wartet auf einen Promise (blockiert also den Rest!)
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await

```
const myPromise = new Promise(resolve => {
  setTimeout(() => {
    resolve('juhu');
  }, 2000);
});
const result = await myPromise();
```

Fetch - Netzwerkaufrufe

- Für alle REST Operationen (GET, POST, PUT, DELETE)
- Optionen (z.B. Methode, Header) als zweiten Parameter
- response.json() ist auch asynchron!
- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

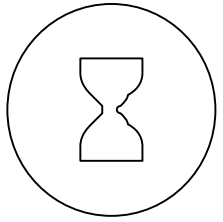
```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```

Aufgabe 4

Baue eine Minianwendung um für eine Stadt das Wetter abzufragen.

- Eingabefeld für die Stadt
- Ausgabe diverser Wetterdaten

- *API-Beschreibung: http://openweathermap.org/current*
- *API Id/Key: f3a44fe9aaf8d153e7e1a56870852478*
- *Hinweis: der API key ist für 60 Aufrufe pro Sekunde. Wenn also alle gleichzeitig Wetterdaten abfragen könnte ein Fehler kommen*



30 Minuten

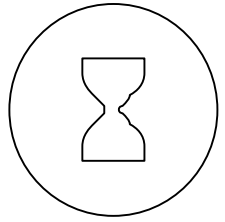
WebComponent

- Standard um eigene html-Tags zu erstellen
- Rendering in gekapseltem shadow DOM
- Wird als JavaScript paketiert
- Kleine Lib mit Hilfsfunktionen: https://lit.dev/
- Tutorial: https://lit.dev/tutorial/

Aufgabe 5

Baue die Wetterabfrage auf eine Web-Komponente um.

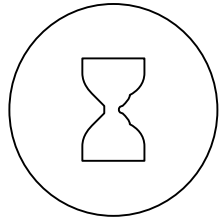
- *Nimm deine eigene Wetterabfrage oder die Lösung von Aufgabe 4 als Vorlage.*



30 Minuten

Aufgabe 6

Baue einen Chat mit Express und socket.io:
https://socket.io/get-started/chat



30 Minuten