Webprogrammierung mit JavaScript



Eine kurze Einführung/eine kurze Übersicht über JavaScript für erfahrene Programmierer.

Dozent: Prof. Dr. Michael Eichberg

Kontakt: michael.eichberg@dhbw.de, Raum 149B

Version: 2.0

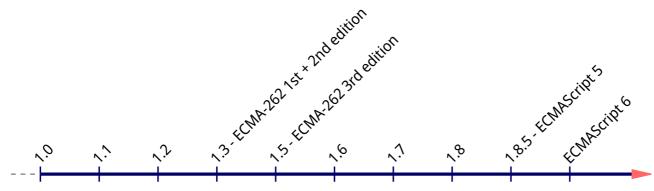
.....

Folien: https://delors.github.io/web-javascript/folien.de.rst.html

https://delors.github.io/web-javascript/folien.de.rst.html.pdf

Fehler melden: https://github.com/Delors/delors.github.io/issues

Historie



03/1996 08/1996 06/1997 10/1998 11/2000 11/2005 10/2006 06/2008 07/2010 06/2015

Seit 2016 gibt es jährliche Updates (ECMAScript 2016, 2017, 2018, 2019, 2020, 2021, 2022, ...)

1. Grundlegende Sprachkonstrukte

Grundlagen

- Objektorientiert
 - Protoypische Vererbung
 - Objekte erben von anderen Objekten
 - Objekte als allgemeine Container
 - (Im Grunde eine Vereinheitlichung von Objekten und Hashtabellen.)
 - seit ES6 werden auch Klassen unterstützt; diese sind aber nur syntaktischer Zucker
- Skriptsprache
 - Loose Typing/Dynamische Typisierung
 - Load and go-delivery (Lieferung als Text/Quellcode)
 - Garbage Collected
 - Single-Threaded
- Funktionen sind Objekte erster Klasse
- (im Wesentlichen) ein (globaler) Namespace
- Syntaktisch eine Sprache der "C"-Familie (viele Ähnlichkeiten zu Java)
- Standardisiert durch die ECMA (ECMAScript)
- Verwendet ganz insbesondere in Browsern, aber auch Serverseitig (z. B. Node.js) oder in Desktop-Anwendungen (z. B. Electron)

Reservierte Schlüsselworte

Schlüsselworte:

- function, async, await, return, yield
- break, continue, case, default, do, else, for, if, instanceof, of, typeof, switch, while
- throw, try, finally, catch
- class, delete, extends, in, new, static, super, this
- const, let, var
- export, import

_C Bemerkung —

Nicht genutzte Schlüsselworte:

enum, implements, interface, package, private, protected, public, void, with (no longer)

Bezeichner (*Identifier*)

(Sehr vergleichbar mit Java.)

- Buchstaben (Unicode), Ziffern, Unterstriche, Dollarzeichen
- Ein Identifier darf nicht mit einer Ziffer beginnen
- Nameskonventionen:
 - Klassen beginnen mit einem Großbuchstaben (*UpperCamelCase*)
 - Variablen und Funktionen beginnen mit einem Kleinbuchstaben (*lowerCamelCase*)
 - Konstanten sind komplett in Großbuchstaben

Global Verfügbare Objekte

Standard

```
console
```

Number, Boolean, Date, BigInt, Math, ...

Von Browsern zur Verfügung gestellte Objekte (Ein Auszug)

```
■ window ■ document (bzw. window. document) ■ alert ■ navigator ■ location
```

Von Node.js zur Verfügung gestellte Objekte (Ein Auszug)

```
■ module ■ exports ■ require ■ process ■ crypto
```

Deklaration von Variablen (const und let)

```
// Der "Scope" ist auf den umgebenden Block begrenzt.
    // Eine Änderung des Wertes ist möglich.
    let y = "yyy";
 3
 4
    // Der "Scope" ist auf den umgebenden Block begrenzt.
 5
    // Eine Änderung des Wertes ist nicht möglich.
 6
    const z = "zzz";
 7
 8
    log("y, z:", y, z);
 9
10
    function doIt() {
11
    const y = "---";
12
13
    log("y, z:", y, z);
     return "";
14
15
16
17 | ilog('"doIt done"', doIt());
18 log("y, z:", y, z);
```

Um diesen und den Code auf den folgenden Folien ggf. mit Hilfe von Node.js auszuführen, muss am Anfang der Datei:

```
import { ilog, log, done } from "./log.mjs";
```

und am Ende der Datei:

```
done();
```

hinzugefügt werden.

Den entsprechenden Code der Module (log.mjs und später Queue.mjs) finden Sie auf:

https://github.com/Delors/delors.github.io/tree/main/web-javascript/code

Datentypen und Operatoren

```
console.log("Undefined ----");
let u = undefined;
console.log("u", u);

console.log("Number ----");
let i = 1; // double-precision 64-bit binary IEEE 754 value
let f = 1.0; // double-precision 64-bit binary IEEE 754 value
```

```
8 let l = 10_000;
   let binary = 0b1010;
9
10 console.log("0b1010", binary);
11 let octal = 0012;
12 | console.log("0012", octal );
    let hex = 0xA;
13
14 | console.log("0xA", hex);
15 console.log(
     Number.MIN_VALUE,
16
     Number.MIN_SAFE_INTEGER,
17
18
     Number.MAX_SAFE_INTEGER,
     Number.MAX_VALUE,
19
20
    );
   let x = NaN;
21
2.2
    let y = Infinity;
    let z = -Infinity;
23
2.4
   // Standard Operatoren: +, - , *, /, %, ++, --, **
25
   // Bitwise Operatoren: &, |, ^, ~, <<, >>, >>>
26
                         (operieren immer auf dem Ganzzahlwert der Bits)
27
   console.log("i =", i, "; i++ ", i++); // 1 oder 2?
28
   console.log("i =", i, "; ++i ", ++i); // 2 oder 3?
29
   console.log("2 ** 4 \equiv 0 ", 2 ** 4);
30
    console.log("7 % 3 \equiv ", 7 % 3);
31
32
   console.log("1 / 0 == ", 1 / 0);
33
34
    console.log("BigInt -----");
35
    let ib = 1n;
36
    console.log(100n \implies PigInt(100));
37
    console.log(Number.MAX_SAFE_INTEGER + 2102); // 9007199254743092
38
    console.log(RigInt(Number.MAX_SAFE_INTEGER) + 2102n); // 9007199254743093n
39
40
41
    console.log("Boolean ------");
42
43
    let b = true; // oder false
    console.log("Boolean(undefined)", Boolean(undefined)); // true oder false?
44
    console.log(null = true ? "true" : "false"); // true oder false?
45
46
47
    console.log("(Quasi-)Logische Operatoren -----");
48
49
    console.log('1 && "1": ', 1 & "1");
   console.log('null && "1": ', null && "1");
50
   console.log("null && true: ", null && true);
51
    console.log("true && null: ", true & null);
52
    console.log("null && false: ", null && false);
53
   console.log("{} && true: ", {} && true);
54
55
   // Neben den Standardoperatoren: ``&\`, ``||``, ``!`` gibt es auch noch ``??``
56
   // Der ``??``-Operator gibt den rechten Operanden zurück, wenn der linke Operand
57
   // ``null`` oder ``undefined`` ist. Andernfalls gibt er den linken Operanden
58
    // zurück.
59
   // ``??`` ist der *nullish coalescing operator (??) (vergleichbar zu ||)*
60
   console.log('1 ?? "1": ', 1 ?? "1");
61
62 console.log('null ?? "1": ', null ?? "1");
    console.log("null ?? true: ", null ?? true);
6.3
64 console.log("true ?? null: ", true ?? null);
65 console.log("null ?? false: ", null ?? false);
    console.log("{} ?? true: ", {} ?? true);
66
67
```

```
console.log('undefined ?? "1": ', undefined ?? "1");
    console.log('undefined ?? "1": ', undefined ?? "1");
 69
    console.log("undefined ?? true: ", undefined ?? true);
 70
    console.log("true ?? undefined: ", true ?? undefined);
 71
    console.log("undefined ?? false: ", undefined ?? false);
 72
     console.log("undefined ?? undefined: ", undefined ?? undefined);
 73
 74
 75
    console.log("Strings -----"):
 76
     let s = "42";
 77
    console.log("Die Antwort ist " + _s + "."); // String concatenation
 78
    console.log(`Die Antwort ist ${_s}.`); // Template literals (Template strings)
 79
 80
    // multiline Strings
    console.log(
 81
        Die Antwort mag ${ s} sein.
 82
        aber was ist die Frage?`);
 83
 84
    console.log(String(42)); // "42"
 85
 86
 87
     console.log("Objekte -----");
 88
     let emptyObject = null;
 89
     let anonymousObj = {
 90
 91
      i: 1,
 92
      u: { j: 2, v: { k: 3 } },
 93
      toString: function () {
       return "anonymousObj";
 94
 95
      "?" : "question mark"
 96
 97
    // Zugriff auf die Eigenschaften eines Objekts
 98
    anonymousObj.j = 2; // mittels Bezeichner ("j") (eng. Identifier)
 99
100 | anonymousObj["j"] = 4; // mittels String ("j")
    anonymousObj["k"] = 3;
101
102
    console.log("anonymousObj:
                                              ", anonymousObj);
    103
     delete anonymousObj["?"]; // delete dient dem Löschen von Eigenschaften
104
    delete anonymousObj.toString; // delete dient dem Löschen von Eigenschaften
105
    console.log("anonymousObj.toString() [original]", anonymousObj.toString());
106
    // Der Chain-Operator kann verwendet werden, um auf Eigenschaften (Properties)
107
    // von Objekten zuzugreifen, ohne dass eine Fehlermeldung ausgegeben wird,
108
    // wenn eine (höher-liegende) Eigenschaft nicht definiert ist.
109
    // Besonders nützlich beim Verarbeiten von komplexen JSON-Daten.
110
111
    console.log("anonymousObj.u?.v.k", anonymousObj.u?.v.k);
     console.log("anonymousObj.u.v?.k", anonymousObj.u.v?.k);
112
    console.log("anonymous0bj.u.v?.z", anonymous0bj.u.v?.z);
113
    console.log("anonymousObj.u.q?.k", anonymousObj.u.q?.k);
114
     console.log("anonymous0bj.p?.v.k", anonymous0bj.p?.v.k);
115
116
     // Nützliche Zuweisungen, um den Fall undefined und null gemeinsam zu behandeln:
117
    118
119
120
121
     console.log("Date -----");
122
     let date = new Date("8.6.2024"); // ACHTUNG: Locale-Settings
123
124
     console.log(date);
125
126
127 console.log("Funktionen sind auch Objekte -----");
```

```
128 let func = function () {
129
      return "Hello World";
     };
130
131 | console.log(func, func());
132
133
134 console.log("Arrays -----"):
135 let temp = undefined;
136
     let a = [1];
     console.log("let a = [1]; a, a.length", a, a.length);
137
138 \$a.push(2); // append
139 console.log("$a.push(2); $a", $a);
     temp = \$a.unshift(0); // "prepend" \rightarrow return new length
140
     console.log("temp = $a.unshift(0); temp, $a", temp, $a);
141
     temp = $a.shift(); // remove first element → return removed element
142
     console.log("temp = $a.shift(); temp, $a", temp, $a);
143
     // Um zu prüfen ob eine Datenstruktur ein Array ist:
144
145 | console.log("Array.isArray($a)", Array.isArray($a));
console.log("Array.isArray({})", Array.isArray({}));
     console.log("Array.isArray(1)", Array.isArray(1));
147
148
149
150 | console.log("Symbols -----");
     let sym1 = Symbol("1"); // a unique and immutable primitive value
151
152 let sym2 = Symbol("1");
153 let obj1Values = { sym1: "value1", sym2: "value2" };
154
     console.log(obj1Values);
     console.log(`sym1 in ${JSON.stringify(obj1Values)): `, sym1 in obj1Values);
155
156 let obj2Values = { [sym1]: "value1", [sym2]: "value2" };
157 | console.log(obj2Values);
158
     console.log(`sym1 in ${JSON.stringify(obj2Values)}: `, sym1 in obj2Values);
159
    console.log(obj1Values, " vs. ", obj2Values);
160
161 | console.log( { sym1 : "this", sym1 : "that" }); // ??? { sym1: "that" }
162 | console.log("sym1 = sym2", sym1 = sym2);
```

Funktionsdefinitionen

```
// Die Funktionsdeklaration der Funktion "hello" ist "hochgezogen" (🛂 hoisted)
   // und kann hier verwendet werden.
   hello("Michael");
 3
 4
 5
    function hello(person = "World" /* argument with default value */) {
     log(`fun: Hello ${person}!`):
 6
 7
 8
    hello():
 9
    waitOnInput();
10
11
12 const helloExpr = function () { // Anonymer Funktionsausdruck
    log("expr: Hello World!");
13
14
15
16 // Arrow Functions
17 | const times3 = (x) \Rightarrow x * 3;
    log("times3(5)", times3(5)); // 15
18
19
20 | const helloArrow = () ⇒ log("arrow: Hello World!");
21 | const helloBigArrow = () ⇒ {
const s = "Hello World!";
```

```
log("arrow: " + s);
24
      return s;
2.5
    helloExpr();
26
    helloArrow();
27
2.8
    var helloXXX = function helloYYY() { // benannter Funktionsausdruck
29
     // "helloYYY" ist _nur_ innerhalb der Funktion sichtbar und verwendbar
30
31
     // "arguments" ist ein Arrays-vergleichbares Objekt
     // und enthält alle Argumente der Funktion
32
33
     log(`Hello: `, ...arguments); // "..." ist der "Spread Operator"
34
    helloXXX("Michael", "John", "Jane");
35
36
    waitOnInput():
37
38
    function sum(...args) {
39
    // rest parameter
40
41
      log("typeof args: " + typeof args + "; isArray: "+ Array.isArray(args));
      log("args: " + args);
42
43
      log("args:", ...args); // die Arraywerte werden als einzelne Args. übergeben
44
      return args.reduce((a, b) \Rightarrow a + b, 0); // function nesting
45
    log(sum(1, 2, 3, 4, 5)); // 15
46
    log(sum());
47
48
    /* Generator Functions */
49
50
    function* fib() {
51
    // generator
52
      let a = 0,
53
        b = 1;
54
      while (true) {
55
        yield a;
56
        [a, b] = [b, a + b];
57
58
59
    const fibGen = fib();
    log(fibGen.next().value); // 0
60
   log(fibGen.next().value); // 1
61
62 log(fibGen.next().value); // 1
    log(fibGen.next().value); // 2
63
    /* Will cause an infinite loop:
64
    for (const i of fib()) console.log(i);
      // 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 ... */
```

Übung - JavaScript und Node.js erste Schritte

Voraussetzung: Installieren Sie Node.js (http://nodejs.org/).

1.1. Hello World in Node.js

Starten Sie die Konsole/Terminal und schreiben Sie ein einfaches JavaScript Programm, das "Hello World" ausgibt.

1.2. Hello World auf der JavaScript Console

Starten Sie einen Browser und aktivieren Sie die JavaScript Console in den Entwicklerwerkzeugen. Schreiben Sie ein einfaches JavaScript Programm, das "Hello World" ausgibt.

Übung - die JavaScript Konsole

1.3. Prototyping mit der JavaScript Konsole

Schreiben Sie ein kurzes JavaScript Programm, das programmatisch zum Ende des Dokuments scrollt. Hinweise:

- das von document. body referenziert HTML Element enthält den gesamten Inhalt des Dokuments
- die aktuellen Abmaße des Dokuments können Sie mit der Funktion window. getComputedStyle(<html Element>). height ermitteln; geben Sie den Wert auf der Konsole aus bevor Sie das Dokument scrollen; was fällt Ihnen auf?
- um zu scrollen, können Sie window.scrollTo(x,y) verwenden
- um den Integer Wert eines Wertes in Pixeln zu bestimmen, können Sie parse Int verwenden

Vergleich von Werten und implizite Typumwandlung

```
// Gleichheit
                               // mit Typumwandlung (auch bei \langle, \rangle, \leq, \geq)
                       == // ohne Typumwandlung
 3
    // strikt gleich
    // strike Ungleichheit ≠ // ohne Typumwandlung
 4
 5
    log('1 = "1": ', 1 = "1");
 6
 7
    log('1 == "1": ', 1 == "1");
    \log("1.0 = 1: ", 1 = 1.0);
 8
    log("1.0 = 1: ", 1 = 1.0);
 9
    log("1 == 1n: ", 1 == 1n);
10
    log("1 = 1n: ", 1 = 1n);
11
    log('1 < "1"', 1 < "1");
12
    log('0 < "1"', 0 < "1");
13
    log('0 \leq "0"', 0 \leq "0");
14
    log('"abc" ≤ "d"', "abc" ≤ "d");
15
16
    log("asdf" \equiv "as" + "df"], "asdf" \equiv "as" + "df"]; // unlike Java!
17
18
    log("NaN == NaN: ", NaN == NaN);
19
    log("NaN = NaN: ", NaN = NaN);
20
    log("null === NaN: ", null === NaN);
21
    log("null = NaN: ", null = NaN);
22
    log("null ≡ null: ", null ≡ null);
    log("null = null: ", null = null);
24
    log("undefined == undefined: ", undefined == undefined);
25
    log("undefined = undefined: ", undefined = undefined);
26
    log("null == undefined: ", null == undefined);
27
    log("null = undefined: ", (null = undefined) + "!");
28
29
30
31
    const a1 = [1, 2, 3];
    const a2 = [1, 2, 3];
    log("const a1 = [1, 2, 3]; a1 = [1, 2, 3]: ", a1 = [1, 2, 3]);
33
    log("const a1 = [1, 2, 3]; a1 = a1: ", a1 = a1);
34
    log("const a1 = [1, 2, 3]; a1 == a1: ", a1 == a1);
35
36
    log("const a1 = [1, 2, 3]; const a2 = [1, 2, 3]; a1 == a2: ", a1 == a2);
    log("const a1 = [1, 2, 3]; const a2 = [1, 2, 3]; a1 = a2: ", a1 = a2);
37
38
    loa(
39
     "flatEquals(a1,a2):",
     a1.length = a2.length \& a1.every((v, i) \Rightarrow v = a2[i])
40
    );
41
42
43
    let firstJohn = { person: "John" };
44
45 | show('let firstJohn = { person: "John" };');
```

```
46 let secondJohn = { person: "John" };
47
    show('let secondJohn = { person: "John" };');
    let basedOnFirstJohn = Object.create(firstJohn);
48
   | show("let basedOnFirstJohn = Object.create(firstJohn)");
49
    log("firstJohn = firstJohn: ", firstJohn = firstJohn);
50
    \log(\text{"firstJohn} \implies \text{secondJohn});
51
    log("firstJohn = secondJohn: ", firstJohn = secondJohn);
52
53
    log("firstJohn == basedOnFirstJohn: ", firstJohn == basedOnFirstJohn);
    log("firstJohn = basedOnFirstJohn: ", firstJohn = basedOnFirstJohn);
54
55
56
57
58
      const obj = {
        name: "John",
59
        age: 30,
60
        city: "Berlin",
61
62
63
      log("\nTyptests und Feststellung des Typs:");
64
      log("typeof obj", typeof obj);
      log("obj instanceof Object", obj instanceof Object);
65
66
      log("obj instanceof Array", obj instanceof Array);
67
68
      const obj = { a: "lkj" };
69
70
      const obj2 = Object.create(obj);
      log("obj2 instanceof obj.constructor", obj2 instanceof obj.constructor);
71
72
73
74
    log("\n?-Operator/if condition and Truthy and Falsy Values:");
75
    log('""', "" ? "is truthy" : "is falsy");
    log("f()", (() \Rightarrow {}) ? "is truthy" : "is falsy");
76
77
   log("Array ", Array ? "is truthy" : "is falsy");
78 log("obj ", {} ? "is truthy" : "is falsy");
   log("undefined ", undefined ? "is truthy" : "is falsy");
79
80 log("null ", null ? "is truthy" : "is falsy");
81 log("0", 0 ? "is truthy" : "is falsy");
82 log("1", 1 ? "is truthy" : "is falsy");
```

NaN (Not a Number) repräsentiert das Ergebnis einer Operation die keinen sinnvollen Wert hat. Ein Vergleich mit NaN ist *immer* false. Um zu überprüfen, ob ein Wert NaN ist muss is NaN (<Value>) verwendet werden.

Bedingungen und Schleifen

```
1 const arr = [1, 3, 4, 7, 11, 18, 29];
 2
    log("if-else_if-else:");
 3
   if (arr.length = 7) {
 4
     ilog("arr.length = 7");
   } else if (arr.length ≤ 7) {
 6
     ilog("arr.length < 7");</pre>
 7
 8
    } else {
 9
      ilog("arr.length > 7");
10
11
12
   log("\nswitch (integer value):");
13
   switch (arr.length) {
14
      case 7:
15
        ilog("arr.length = 7");
        break;
16
```

```
17
      case 6:
        ilog("arr.length = 6");
18
19
        break;
20
      default:
        ilog("arr.length \neq 6 and \neq 7");
21
22
23
    log("\nswitch (string value):");
24
    switch ("foo") {
25
      case "bar":
26
        ilog("it's bar");
27
28
        break;
29
      case "foo":
        ilog("it's foo");
30
        break:
31
      default:
32
        ilog("not foo, not bar");
33
34
35
    log("\nswitch (integer - no type conversion):");
36
37
    switch (
38
      1 // Vergleich auf strikte Gleichheit (■)
    ) {
39
40
      case "1":
41
        ilog("string(1)");
        break:
42
43
      case 1:
        ilog("number(1)");
44
45
        break;
46
47
    ilog("\nfor-continue:");
48
    for (let i = 0; i < arr.length; i \leftrightarrow) {
49
      const v = arr[i];
50
51
      if (v \% 2 = 0) continue;
52
      log(v);
53
54
    ilog("\n(for)-break with label:");
55
    outer: for (let i = 0; i < arr.length; i \leftrightarrow) {
56
      for (let j = 0; j < i; j \leftrightarrow i) {
57
58
        if (j = 3) break outer;
         log(arr[i], arr[j]);
59
      }
60
61
62
    ilog("\nin (properties of Arrays; i.e. the indexes):");
63
    for (const key in arr) {
64
6.5
      log(key, arr[key]);
66
67
    ilog("\nof (values of Arrays):");
68
    for (const value of arr) {
69
70
      log(value);
71
72
    ilog("\nArray and Objects - instanceof:");
73
74
    log("arr instanceof Object", arr instanceof Object);
75
    log("arr instanceof Array", arr instanceof Array);
76
```

```
const obj = {
 77
       name: "John",
 78
 79
       age: 30,
       city: "Berlin",
 80
     };
 81
 82
      ilog("\nin (properties of Objects):");
 83
     for (const key in obj) {
 84
       log(key, obj[key]);
 85
 86
 87
 88
     /* TypeError: obj is not iterable
 89
     for (const value of obj) {
         log(value);
 90
 91
     */
 92
 93
 94
 95
       ilog("\nIteration über Iterables (here: Map):");
 96
       const m = new Man();
       m.set("name", "Elisabeth");
 97
       m.set("alter", 50);
 98
       log("Properties of m: ");
 99
       for (const key in m) {
100
101
         log(key, m[key]);
       }
102
        log("Values of m: ");
103
       for (const [key, value] of m) {
104
105
          log(key, value);
106
107
108
109
        ilog("\nWhile Loop: ");
110
111
       let c = 0;
112
       while (c < arr.length) {</pre>
113
          const v = arr[c];
          if (v > 10) break;
114
         log(v);
115
116
          C++;
117
118
119
120
       ilog("\nDo-While Loop: ");
121
122
       let c = 0;
       do {
123
         log(arr[c]);
124
125
         C++;
126
       } while (c < arr.length);</pre>
127
```

Die Tatsache, dass insbesondere null als auch undefined falsy sind, wird of in Bedingungen ausgenutzt (z. B., if (!x)...).

Fehlerbehandlung

```
console.log("try-catch-finally - Grundlagen -----");
try {
```

```
let i = 1 / 0; // Berechnungen erzeugen nie eine Exception
 4
 5
      console.log("i", i);
    } catch {
 6
 7
      console.error("console.log failed");
 8
    } finally {
      console.log("computation finished");
 9
10
11
    console.log("Programmierfehler behandeln ------
12
13
     const obj = {};
14
     obj = \{ a: 1 \};
15
   } catch ({ name, message }) {
16
     console.error(message);
17
18
   } finally {
19
     console.log("object access finished");
20
21
   console.log("Handling of a specific error ------");
22
23
    try {
     throw new RangeError("out of range");
24
25 } catch (error) {
     if (error instanceof RangeError) {
26
27
        const { name, message } = error;
        console.error("a RangeError:", name, message);
2.8
      } else {
29
30
        throw error;
31
32
   } finally {
     console.log("error handling finished");
33
34
```

In JavaScript können während der Laufzeit Fehler auftreten, die (z. B.) in Java während des kompilierens erkannt werden.

Übung - Bedingungen und Schleifen

1.4. removeNthElement

Implementieren Sie eine Funktion, die ein Array übergeben bekommt und ein neues Array zurückgibt in dem jedes n-te Element nicht vorkommt.

Beispiel: removeNthElement([1,2,3,4,5,6,7], 2) \Rightarrow [1,3,5,7]

- Schreiben Sie Ihren Code in eine JavaScript Datei und führen Sie diese mit Hilfe von Node.js aus.
- Testen Sie Ihre Funktion mit verschiedenen Eingaben und lassen Sie sich das Ergebnis ausgeben (z. B. console.log(removeNthElement([1,2,3,4,5,6,7],2)))!

Übung - Fehlerbehandlung

1.5. removeNthElement mit Fehlerbehandlung

- Erweitern Sie die Implementierung von removeNthElement so, dass die Funktion einen Fehler wirft, wenn das übergebene Array kein Array ist oder wenn der zweite Parameter kein positiver Integer ist.
- Testen Sie alle Fehlerzustände und fangen Sie die entsprechenden Fehler ab (catch) und geben Sie die Nachrichten aus.

Übung - Funktionen

1.6. Einfacher RPN Calculator

Implementieren Sie einen einfachen RPN (Reverse Polish Notation) Calculator, der eine Liste von Zahlen und Operatoren (+, -, *, /) als Array entgegennimmt und das Ergebnis berechnet.

Nutzen Sie keine if oder switch Anweisung, um die Operatoren zu unterscheiden. Nutzen Sie stattdessen ein Objekt. Sollte der Operator unbekannt sein, dann geben Sie eine entsprechende Fehlermeldung aus.

```
Beispiel: eval([2,3,"+",4,"*"]) \Rightarrow 20
```

Variables (var)

(Neuer Code sollte var nicht mehr verwenden!)

```
let y = "yyy"; // wie zuvor
    const z = "zzz";
 2
 3
    // Der Gültigkeitsbereich von var ist die umgebende Funktion oder der
 4
    // globale Gültigkeitsbereich.
 5
    // Die Definition ist hochgezogen (eng. "hoisted") (initialisiert mit undefined);
 6
 7
    var x = "xxx";
 8
    function sumIfDefined(a, b) {
 9
     // Der folgende Code ist NICHT empfehlenswert!
10
11
      // Er dient der Visualisierung des Verhaltens von var.
      if (parseInt(a)) {
12
13
        var result = parseInt(a);
14
      } else {
        result = 0:
15
16
17
      const bVal = parseFloat(b);
18
      if (bVal) {
19
        result == bVal;
20
      }
21
      return result:
22
23
    ilog("sumIfDefined()", sumIfDefined()); // 0
24
    ilog("sumIfDefined(1)", sumIfDefined(1)); // 1
25
    ilog("sumIfDefined(1, 2)", sumIfDefined(1, 2)); // 3
26
    ilog('sumIfDefined(1, "2")', sumIfDefined(1, "2")); // 3
27
    ilog("undefined + 2", undefined + 2);
28
    ilog('sumIfDefined(undefined, "2")', sumIfDefined(undefined, "2")); // 2
29
30
31
    function global x() {
32
      ilog("global_x():", x, y, z);
33
34
    function local_var_x() {
35
      ilog("local_var_x(): erste Zeile (x)", x);
36
37
      var x = 1; // the declaration of var is hoisted, but not the initialization
38
      let y = 2;
39
40
      const z = 3;
41
42
      ilog("local_var_x(): letzte Zeile (x, y, z)", x, y, z); // 1 2 3
43
44
    global_x();
45
46
    local_var_x();
47
48 | ilog("nach global_x() und local_var_x() - x, y, z:", x, y, z);
```

```
49
50
    // Hier, ist nur die Variablendeklaration (helloExpr) "hoisted", aber nicht
51
52 // die Definition. Daher kann die Funktion nicht vorher im Code aufgerufen
53 // werden!
54
   try {
    helloExpr();
55
   } catch ({error, message}) {
56
57
     log("calling helloExpr() failed:", error, "; message: ", message);
58
59
    var helloExpr = function () {
60
    log("expr: Hello World!");
61
62 // ab jetzt funktioniert es
63 helloExpr():
```

Destrukturierung (Destructuring)

```
log("Array Destructuring:");
 2
 3
    let [val1, val2] = [1, 2, 3, 4];
    ilog("[val1, val2] = [1, 2, 3, 4]:", "val1:", val1, ", val2:", val2); // 1
 4
 5
    log("Object Destructuring:");
 6
 7
    let { a, b } = { a: "aaa", b: "bbb" };
 8
    ilog('let { a, b } = { a: "aaa", b: "bbb" }: ', "a:", a, ", b:", b); // 1
9
10
    let { a: x, b: y } = { a: "aaa", b: "bbb" };
11
    ilog('let { a: x, b: y } = { a: "aaa", b: "bbb" }: ', "x:", x, ", y:", y); // 1
12
13
    let { a: u, b: v, ...w } = { a: "+", b: "-", c: "*", d: "/" };
14
15 | ilog(
      'let { a: u, b: v, ...w } = { a: "+", b: "-", c: "*", d: "/" }:',
16
17
18
      u,
      ", V:",
19
20
      ٧,
      ", W:"
21
      JSON. stringify(w), // just for better readability/comprehension
22
23
24
25 | let { k1, k2 } = { a: "a", b: "b" };
26 | iloq('let { k1, k2 } = { a: "a", b: "b" }:', "k1:", k1, ", k2:", k2);
27 // "undefined undefined", weder k1 noch k2 sind definiert
```

JSON (JavaScript Object Notation)

```
1
    const someJSON = `{
        "name": "John",
 2
        "age": 30,
 3
        "cars": {
 4
            "American": ["Ford"],
 5
            "German": ["BMW", "Mercedes", "Audi"],
 6
 7
            "Italian": ["Fiat", "Alfa Romeo", "Ferrari"]
 8
 9
10
11
12 // JSON.parse(...) JSON String ⇒ JavaScript Object
```

```
const someObject = JSON.parse(someJSON);

someObject.age = 31;
someObject.cars.German.push("Porsche");
someObject.cars.Italian.pop();
console.log(someObject);

// JSON.stringify(...) JavaScript Object ⇒ JSON String
console.log(JSON.stringify(someObject, null, 2));
```

JSON requires that keys must be strings and strings must be enclosed in double quotes.

Reguläre Ausdrücke

- Eingebaute Unterstützung basierend auf entsprechenden Literalen (Strings in "/") und einer API
- inspiriert von der Perl Syntax
- Methoden auf regulären Objekten: test (e.g., Regixp. test (String)).
- Methoden auf Strings, die reguläre Ausdrücke verarbeiten: search, match, replace, split, ...

```
1
 2
      const p = /.*[1-9]+H/; // a regexp
 3
      console.log(p.test("ad13H"));
      console.log(p.test("ad13"));
 4
 5
      console.log(p.test("13H"));
 6
 7
 8
      const p = /[1-9] + H/g;
      const s = "1H, 2H, 3P, 4C";
 9
      console.log(s.match(p));
10
      console.log(s.replace(p, "XX"));
11
12 }
```

Klassen und Vererbung

```
1
    class Figure {
 2
      calcArea() {
        throw new Error("calcArea is not implemented");
 3
 4
 5
    class Rectangle extends Figure {
 6
 7
      height;
 8
      width;
 9
      constructor(height, width) {
10
        super();
11
        this.height = height;
12
13
        this.width = width;
      }
14
15
      calcArea() {
16
        return this.height * this.width;
17
18
19
20
      get area() {
21
        return this.calcArea();
22
23
      set area(value) {
24
        throw new Error("Area is read-only");
25
26
```

```
27 }
28
29
   const r = new Rectangle(10, 20);
30 console.log("r instanceof Figure", r instanceof Figure); // true
31 | console.log(r.width);
   console.log(r.height);
33 console.log(r.area); // 200
34
35 try {
    r.area = 300; // Error: Area is read-only
36
37
   | catch (e) {
    console.error(e.message);
38
```

Grundlagen von ECMAScript Modulen

Queue.mjs exportiert die Klasse Queue

```
1 /* Modul für den Datentyp Warteschlange (Queue). */
    export class Queue {
 3
      #last = null; // private field
 4
      #first = null;
      constructor() {} // "default constructor"
 5
      enqueue(elem) {
 6
 7
        if (this.#first == null) {
          const c = { e: elem, next: null };
 8
          this.#first = c;
 9
10
          this.#last = c;
        } else {
11
          const c = { e: elem, next: null };
12
13
          this.#last.next = c;
          this.#last = c;
14
        }
15
      }
16
17
      dequeue() {
        if (this.#first 	≡ null) {
18
19
         return null;
        } else {
20
21
          const c = this.#first;
22
          this.#first = c.next;
          return c.e;
23
24
25
      }
      head() {
26
        if (this.#first ≡ null) {
27
         throw new Error("Queue is empty");
28
29
        } else {
          return this.#first.e;
30
31
      }
32
33
      last() {
34
        if (this.#first ≡ null) {
         throw new Error("Queue is empty");
35
        } else {
36
37
          return this.#last.e;
38
39
40
      isEmpty() {
41
        return this.#first == null;
42
43 }
```

log.mjs verwendet (import) die Klasse Queue und exportiert Funktionen zum Loggen

```
import { Queue } from "./Queue.mjs"; // import des Moduls "Queue.mjs"
 2
 3
    const messages = new Queue();
 4
    export function log(...message) {
 5
     if (messages.isEmpty()) {
 6
        messages.enqueue(message);
 7
      } else {
 8
 9
        message.unshift("\n");
        messages.last().push(...message);
10
11
12
```

ECMAScript Module verwenden immer den strict mode.

Import Statements erlauben das selektierte importieren als auch das Umbenennen von importierten Elementen (z. B., import { Queue as Q } from "./Queue.mjs";).

Alles ist ein Objekt

- this ist ein "zusätzlicher" Parameter, dessen Wert von der aufrufenden Form abhängt
- this ermöglicht den Methoden den Zugriff auf ihr Objekt
- this wird zum Zeitpunkt des Aufrufs gebunden (außer bei Arrow-Funktionen)

```
//"use strict";
 1
 2
 3
    function counter () {
        // console.log(this \equiv globalThis); // true
 4
 5
        if(this.count) // this is the global object if we don't use strict mode
 6
            this.count ++;
 7
        else {
 8
            this.count = 1;
 9
10
        return this.count;
11
12
13
    const counterExpr = function () {
14
        if(this.count)
15
            this.count ++;
16
17
        else {
18
            this.count = 1;
19
20
21
        return this.count;
22
23
24
    const counterArrow = () \Rightarrow \{
25
        console.log(this);
        console.log(this = globalThis);
26
        this.count = this.count ? this.count + 1 : 1;
27
        return this.count;
28
29
30
    console.log("\nCounter");
31
    console.log(counter()); // 1
32
33
    console.log(counter()); // 2
    console.log(`Counter (${qlobalThis.count})`);
34
35
```

```
console.log("\nCounterExpression");
37
    console.log(counterExpr()); // 3
    console.log(counterExpr()); // 4
38
39
    console.log("\nCounter");
40
41
    const obj = {};
    console.log(counter.apply(obj)); // 1 - we set a new "this" object!
42
    console.log(counterExpr.apply(obj)); // 2
43
44
45
    console.log(`\nCounterArrow (${this.count})`);
46
    console.log(counterArrow.apply(obj)); // 1
    console.log(counterArrow.apply(undefined)); // 2
47
48
    console.log(counterArrow.apply()); // 3
    console.log(counterArrow.apply(obj)); // 4
49
50
    console.log(counterArrow.apply({})); // 5
51
    console.log("\nCounter (global)");
52
53 | console.log(counter());
54 | console.log(counterExpr());
```

Partial Function Application

```
function add(x, y) {
 2
      return x + y;
 3
 4
    // Partial function application:
 5
    const add2 = add.bind(null, 2); // "null" is the value of "this"
 6
    console.log(add2(3));
 7
 8
9
    function addToValue(b) {
10
11
     return this.x + b;
12
   console.log(addToValue.call(\{x : 0\}, -101));
```

Prototype basierte Vererbung

Verwendung von <u>Object.create</u> zur Initialisierung der *Prototype Chain*:

Verwendung der Eigenschaften von Prototypen:

```
const p = { s : "p" };
2
  const c = Object.create(p);
3
  const gc = Object.create(c);
4 | qc.t = "q";
    gc
                                С
                                                          p
         proto
                                    proto
                                                              proto
                                                                                 →Object.prototype
                   "a"
                                                                        "p"
           t.
                                                                S
```

```
5  gc.s = "gc"
6  console.log(gc.s); // gc
7  delete gc.s;
8  console.log(gc.s); // p
```

Pseudoclassical Inheritance

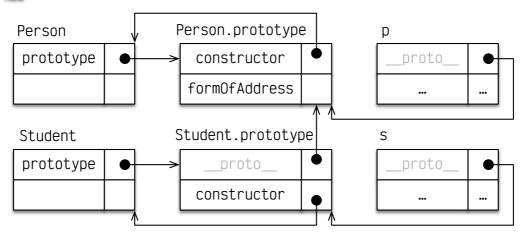
```
// constructor for Person objects:
    function Person(name, title){ this.name = name; this.title = title; }
 2
 3
    Person.prototype.formOfAddress = function (){
       const foa = "Dear ";
 4
       if(this.title){ foa += this.title+" "; }
 5
       return foa + this.name;
 6
 7
    function Student(name, title, id, email) {
 8
       Person.call(this, name, title); // super constructor call
 9
10
       this.id = id;
       this.email = email;
11
12
    Student.prototype = Object.create(Person.prototype);
13
    Student.prototype.constructor = Student;
14
15
   const aStudent = new Student("Emily Xi", "Mrs.", 12441, 'emily@xi.de');
16
```

Objektabhängigkeiten

```
function Person(name, title){ ... }
Person.prototype.formOfAddress = function (){ ... }

function Student(name, title, id, email) { ... }
Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;

const p = new Person(...);
const s = new Student(...);
```



Die Eigenschaft prototype einer Funktion (F) verweist auf das Objekt, dass als Prototype (__proto__) verwendet wird, wenn die Funktion als Konstruktor verwendet wird. D. h. im Falle einer Instantiierung von F (d. h. const newF = new F()) wird das Objekt, das durch F.prototype referenziert wird, als Prototype (newF.__proto__) des neu erstellten Objekts (newF) verwendet.

```
// Prototypen
console.log("{}.__proto__: ",{}.__proto__);
console.log("Array.prototype: ",Array.prototype);
console.log("Array.prototype.__proto__: ",Array.prototype.__proto__);
console.log("Object.prototype: ",Object.prototype);
console.log("Object.__proto__: ",Object.__proto__);
```

```
let 0 = { created: "long ago" };
var p = Object.create(o);
console.log("Object.getPrototypeOf(o): " + Object.getPrototypeOf(o));
console.log("o.isPrototypeOf(p):" + o.isPrototypeOf(p));
console.log("Object.prototype.isPrototypeOf(p): " + Object.prototype.isPrototypeOf(p));
```

Praktische Verwendung von Prototypen basierter Vererbung

```
let a = [1, 10, 100, 1000];
                           try { console.log(a.fold()); } catch (error) {
                                      console.log("error: ", error.message);
        3
        4
        5
                          // - ATTENTION! -----
        6
                          // ADDING FUNCTIONS TO Array.prototpye IS NOT RECOMMENDED! IF ECMAScript
                         // EVENTUALLY ADDS THIS METHOD (I.E. fold) TO THE PROTOTYPE OF ARRAY OBJECTS,
                          // IT MAY CAUSE HAVOC.
       9
                          Array.prototype.fold = function (f) {
 1.0
                                       if (this.length == 0) {
 11
                                                   throw new Error("array is empty");
 12
                                       } else if (this.length == 1) {
 13
14
                                                  return this[0];
 15
                                      } else {
                                                  let result = this[0];
 16
                                                  for (let i = 1; i < this.length; i \leftrightarrow this.len
 17
                                                             result = f(result, this[i]);
 18
                                                  }
 19
 20
                                                  return result;
 21
                         };
 22
 23
24 | console.log(a.fold((u, v) \Rightarrow u + v));
```

DOM Manipulation

```
1
    <html lang="en">
 2
        <head>
            <meta charset="utf-8" />
 3
 4
             <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 5
            <title>DOM Manipulation with JavaScript
             <script>
 6
 7
                 function makeScriptsEditable() {
                     const scripts = document.getElementsByTagName("script");
 8
 9
                     for (const scriptElement of scripts) {
                          scriptElement.contentEditable = false;
10
                         const style = scriptElement.style;
11
                         style.display = "block";
12
                         style.whiteSpace = "preserve";
13
                         style.padding = "1em";
14
15
                         style.backgroundColor = "yellow";
16
17
                 }
             ⟨/script⟩
18
        </head>
19
        <body>
20
             <h1>DOM Manipulation with JavaScript</h1>
21
22
            \langle p id = "demo" \rangleThis is a paragraph. \langle p \rangle
             <button
23
24
                 type="button"
                 onclick="
25
```

```
26
                         document.getElementById('demo').style.color = 'red';
27
                         makeScriptsEditable();
                         document.guerySelector('button').style.display = 'none';"
2.8
29
            >
30
                Magic!
31
             ⟨/button⟩
32
             <script>
33
                const demoElement = document.getElementById("demo");
34
35
                const style = demoElement.style;
                 demoElement.addEventListener(
36
37
                     "mouseover".
                     () ⇒ (style.color = "green"),
38
                 );
39
                 demoElement.addEventListener(
40
                     "mouseout".
41
                     () \Rightarrow (style.color = "unset"),
42
43
                );
44
             ⟨/script⟩
45
            Position der Mouse: <span id="position"></span>
46
47
             <script>
                 window.addEventListener("mousemove", () ⇒ {
48
49
                     document.getElementById("position").innerHTML =
                          (${event.clientX}, ${event.clientY})`;
50
                });
51
             </script>
52
53
        </body>
54 </html>
```

Minimaler Server mit Express JS

```
// "express" and "cors" are CommonJS modules, which requires us to use the
    // "default import" syntax.
 3
    import express from "express";
 4
    // Cross-Origin Resource Sharing (CORS); This is required to allow the browser
 5
    // using a different domain to load the HTML to make requests to this server.
 6
    // I. e., we can use the HTML file from the "web-javascript" project to make
 7
    // requests to this server.
    import cors from "cors";
 9
10
    const APP PORT = 5080;
11
    const app = express();
12
13
    app.get("/users", cors(), function (req, res) {
14
      res.set("Content-Type", "application/json");
15
      res.end(`{
16
            "user1" : {
17
               "name" : "dingo",
18
               "password" : "1234",
19
               "profession" : "chef",
20
               "id": 1
21
22
            },
23
            "user2" : {
24
25
               "name" : "ringo",
               "password" : "asdf",
26
               "profession" : "boss",
27
28
               "id": 3
29
```

```
31    });
32
33    app.listen(APP_PORT, function () {
      console.log(`Users App @ http://127.0.0.1:${APP_PORT}`);
35    });
```

Express ist ein minimalistisches Web-Framework für Node.js, das die Entwicklung von Webanwendungen vereinfacht. Die Installation kann über einen Packagemanager erfolgen.

Installieren Sie (z. B.) pnpm (https://pnpm.io/) und nutzen Sie danach pnpm, um die benötigten Module zu installieren:

```
$ pnpm init
$ pnpm install express
```

}`);

Danach starten Sie Ihren Server mit:

```
node --watch UsersServer.mjs
```

Interaktion mit Server mit Hilfe von Fetch

```
1
    <html lang="en">
 2
        <head>
 3
            <meta charset="utf-8" />
            <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 4
            <title>Eventhandling</title>
 5
        </head>
 6
        <body>
 7
            <script>
 8
 9
                /* Using Promises:
                function getUsers() {
10
                    fetch('http://127.0.0.1:4080/users')
11
                         .then(response ⇒ response.json())
12
13
                         .then(users \Rightarrow {
                             const usersElement = document.getElementById('users');
14
                             usersElement.innerText = JSON.stringify(users);
15
                        });
16
                        }
17
                */
18
19
                /* Using async/await: */
20
                async function getUsers() {
21
                     let response = await fetch("http://127.0.0.1:5080/users");
22
                    let users = await response.json();
23
                    const usersElement = document.getElementById("users");
24
25
                    usersElement.innerText = JSON.stringify(users);
2.6
            ⟨/script⟩
27
28
            <div id="users"></div>
29
            <button onclick="getUsers()">Get Users
30
31
        </body>
32 \\/html>
```

Beispiel - Rumpf einer einfachen Webanwendung ("Quizzy")

Im Folgenden verwenden wir zur Client-/Server-Kommunikation insbesondere Websockets.

Server

```
1 | const express = require('express');
```

```
const app = express();
 3
    const expressWs = require('express-ws')(app);
 4
 5
    let clients = 0;
 6
 7
    let playerWSs = [];
 8
    let adminWS = null;
 9
    let answersCount = 0;
10
    let correctAnswersCount = 0:
11
12
    app.use(express.static('.')); // required to serve static files
13
14
15
    function sendCurrentPlayers() {
16
        if (adminWS ‱ playerWSs.length > 0) {
17
            allPlayers = playerWSs
18
                 .filter(player \Rightarrow player.name)
19
                 .map(player ⇒ { return { "id": player.id, "name": player.name } })
2.0
             console.log("Sending current players: " + JSON.stringify(allPlayers));
21
            adminWS.send(JSOM.stringify({ "type": "players", "players": allPlayers }));
2.2
        }
2.3
24
25
26
    function sendNextQuestion() {
27
        answersCount = 0;
28
        correctAnswersCount = 0;
29
        const question = "What is the capital of France?";
        const answers = ["Paris", "London", "Berlin", "Madrid"];
30
        const correct = "Paris";
31
32
33
        const nextQuestion = JSON.stringify({
            "type": "question",
34
             "question": question,
35
36
             "answers": ["Paris", "London", "Berlin", "Madrid"]
37
        })
38
        playerWSs.forEach(player ⇒ player.ws.send(nextQuestion));
        adminWS.send(JSON.stringify({
39
            "type": "question",
40
            "question": question,
41
            "answers": answers,
42
43
            "correct": correct
        }));
44
4.5
46
    function sendResults() {
47
        const results = playerWSs.map(player \Rightarrow {
48
             return { "id": player.id, "name": player.name, "wins": player.wins }
49
50
        });
        const sortedResults = results.sort((a, b) \Rightarrow b.wins - a.wins);
51
        const resultsMsg = JSON.stringify({
52
             "type": "results",
53
            "results": sortedResults
54
55
        playerWSs.forEach(player ⇒ player.ws.send(resultsMsg));
56
57
        adminWS.send(resultsMsq):
58
59
60
61
62
   function handleAnswer(clientId, answer) {
```

```
63
          const correct = answer.answer == "Paris";
          const player = playerWSs.find(player \Rightarrow player.id \rightleftharpoons clientId);
 64
 65
          if (correct) {
              if (correctAnswersCount ≡ 0) {
 66
 67
                  player.wins₩;
 68
 69
              correctAnswersCount++;
          }
 70
 71
          answersCount++;
          if (answersCount == playerWSs.length) {
 72
              // sendNextQuestion();
 73
 74
              sendResults();
 75
          } else {
              adminWS.send(JSON.stringify({
 76
                  "type": "answers",
 77
                  "count": answersCount,
 78
                  "correctAnswersCount": correctAnswersCount
 79
              }));
 80
          }
 81
 82
 83
 84
      app.ws('/player', function (ws, request) {
 85
          const clientId = clients++;
 86
 87
          const playerData = { "ws": ws, "id": clientId, "wins": 0 }
          playerWSs.push(playerData);
 88
 89
          ws.onmessage = function (event) {
              message = JSON.parse(event.data);
 90
              switch (message.type) {
 91
                  case "registration":
 92
 93
                      const name = message.name;
                      console.log("Registration: " + clientId + "/" + name);
 94
 95
                      playerData.name = name;
                      sendCurrentPlayers();
 96
 97
                      break;
 98
                  case "answer":
 99
                      const answer = message;
100
                      handleAnswer(clientId, answer);
101
102
                      break:
103
104
                  default:
105
                      console.log("Unknown message: " + message);
106
                      break;
107
          };
108
109
          ws.onclose = function () {
              console.log("Player disconnected: " + clientId);
110
111
              playerWSs = playerWSs.filter(player ⇒ player.id ≠ clientId);
              sendCurrentPlayers();
112
          };
113
          ws.onerror = function () {
114
              console.log("Player error: " + clientId);
115
116
              playerWSs = playerWSs.filter(player ⇒ player.id ≠ clientId);
              sendCurrentPlayers();
117
118
          };
     });
119
120
      app.ws('/admin', function (ws, req) {
121
122
          adminWS = ws;
```

```
123
          sendCurrentPlayers(); // when admin registers her/himself, send current players
124
         ws.onmessage = function (event) {
             message = JSON.parse(event.data);
125
126
             switch (message.type) {
                  case "start":
127
                      console.log("Start game");
128
                      sendNextQuestion();
129
130
                      break;
131
                  default:
132
                      console.log("Unknown message: " + message);
                      break;
133
             }
134
         };
135
136
         ws.onclose = (event) \Rightarrow {
137
138
             console.log("Admin disconnected");
139
              adminWS = null;
             sendCurrentPlayers();
140
141
         };
142
         ws.onerror = (event) \Rightarrow {
143
144
              console.log("Admin error: " + event);
145
              sendCurrentPlayers();
         };
146
147
148 });
149
150
151
     var server = app.listen(8800, function () {
         console.log("Quizzy running at http://127.0.0.1:8800/");
152
153 })
```

Client - Players

```
<!DOCTYPE html>
 2
    <html lang="en">
 3
 4
    <head>
        <script>
 5
             const ws = new WebSocket("ws://localhost:8800/player");
 6
 7
             ws.onmessage = (event) \Rightarrow {
                 const data = JSON.parse(event.data);
 8
 9
                 switch (data.type) {
10
                     case "question":
                          console.log("Question: " + data.question);
11
12
                          showQuestion(data);
                          break;
13
                     case "results":
14
                          const main = document.getElementById("main")
15
16
                          main.innerText = "Results: " + event.data;
17
                          break:
18
                     default:
19
                         console.log("Unknown message: " + data);
20
                          break;
                 }
21
             };
22
23
             ws.onclose = (event) \Rightarrow \{
24
                 console.log("Connection closed: " + event);
25
26
             ws.onerror = (event) \Rightarrow {
                 console.error("Error: " + event);
27
```

```
28
29
            function showQuestion(data) {
30
                const main = document.getElementById("main")
31
                main.innerHTML = `<h1>Question</h1>${data.question}`;
32
33
                function createAnswerButton(answer) {
34
                    const button = document.createElement("button");
35
                    button.innerText = answer;
36
                    button.onclick = submitAnswer(answer);
37
                    return button;
38
                }
39
40
                for (answer of data.answers) {
41
                    main.appendChild(createAnswerButton(answer));
42
                }
43
            }
44
45
46
            function submitAnswer(answer) {
                return () \Rightarrow {
47
                    ws.send(JSON.stringify({
48
49
                        "type": "answer",
                        "answer": answer
50
51
                    }));
52
                    doWait();
                }
53
            }
54
55
            function submitUsername() {
56
                const name = document.getElementById("username").value;
57
                ws.send(JSON.stringify({
58
                    "type": "registration",
59
                    "name": name
60
                }));
61
62
63
                doWait();
            }
64
65
            function doWait() {
66
                const main = document.getElementById("main");
67
                main.innerHTML = "Waiting for other players...";
68
69
        ⟨/script⟩
70
71
72
    <body>
73
74
        <main id="main">
75
            <form>
                <input type="text" id="username" placeholder="Username">
76
                <button type="button" onclick="submitUsername();">Submit
77
78
            </form>
        ⟨/main⟩
79
    </body>
80
81
82 </html>
```

Client - Admin

```
1 <!DOCTYPE html>
2 <html lang="en">
3
```

```
4
    <head>
 5
        <script>
            const ws = new WebSocket("ws://localhost:8800/admin");
 6
 7
            ws.onmessage = (event) \Rightarrow {
 8
                const data = JSON.parse(event.data);
 9
                console.log("Received: " + event.data);
10
                 switch (data.type) {
11
                    case "players":
12
                         const players = document.getElementById("players")
13
                         players.innerText =
14
                             "["+data.players.length + " players] " +
15
16
                             data.players
                                 .map(player ⇒ player.id + ": " + player.name)
17
                                 .join(", ");
18
19
                         break;
                     case "question":
20
                         showQuestion(data);
21
22
                         break:
                     case "results":
23
                        const main = document.getElementById("main")
24
25
                         main.innerText = "Result: " + event.data;
26
                         break;
                    default:
27
                         console.log("unknown: " + event.data);
28
29
                         break:
                }
30
            };
31
32
33
            ws.onclose = (event) \Rightarrow {
                console.log("Connection closed: " + event);
34
                const main = document.getElementById("main")
35
                main.innerText = "Connection closed - you need to restart.";
36
            };
37
38
            ws.onerror = (event) \Rightarrow {
39
                 console.log("Connection error: " + event);
            };
40
41
            function startGame() {
42
                ws.send(JSON.stringify({"type": "start"}));
43
44
45
            function showQuestion(data) {
46
                 document.getElementById("main").innerText = `
47
                    question: ${data.question}; correct answer: ${data.correct}
48
49
50
        ⟨/script⟩
51
    </head>
52
53
54
    <body>
55
        <main id="main">
56
            <h1>Players</h1>
57
            <button type="button" onclick="startGame();">Start Game/button>
58
        </main>
59
60
    </body>
61
62 (\/html>
```

Die Implementierung dient nur dazu die grundlegenden Konzepte zu verdeutlichen. Es fehlen **viele** Aspekte wie z. B., Sicherheit.

Authentifizierung mit JWT (und Express)

Im Folgenden wird primär die Verwendung eines JWTs zur Authentifizierung von Benutzern demonstriert.

Die initiale Authentifizierung, die im folgenden Beispiel über ein per get-Request übermittelten Benutzernamen und Passwort erfolgt, ist **nicht sicher**. In einer realen Anwendung sollte für die initiale Authentifizierung ein sicherer Mechanismus verwendet werden. Eine Möglichkeit wäre z. B. die Verwendung von DIGEST Authentication (nicht empfohlen bzw. nur für einfachste Fälle). Sinnvoll wäre Basic Authentication *in Verbindung mit HTTPS* oder zum Beispiel der Einsatz von OAuth.

Warnung

Basic Authentication ohne HTTPS ist nicht sicher!

D.h. *Basic Authentication* ist genauso unsicher wie die hier gezeigte Lösung für die initiale Authentifizierung.

Server

```
1 import express from "express";
    import fs from "fs";
    import path from "node:path";
 3
 4 import { fileURLToPath } from "url";
 5 import jwt from "jsonwebtoken";
    import crypto from "crypto";
 6
    import bodyParser from "body-parser";
 7
 9
    const app = express();
10
    const SERVER SECRET = crypto.randomBytes(64).toString("hex");
11
12 const users = JSON.parse(
      fs.readFileSync(
13
14
        path.resolve(path.dirname(fileURLToPath(import.meta.url)), "users.json"),
15
        "utf8",
16
     ),
    );
17
    console.log("Users: " + JSON.stringify(users));
18
19
    app.use(express.static("."));
20
21
    app.use(express.json());
    app.use(bodyParser.text());
22
23
    const verifyToken = (reg, res, next) \Rightarrow {
24
      console.log("Headers: " + JSON.stringify(req.headers));
25
26
      const token = req.headers["authorization"].split(" ")[1];
27
28
      if (!token) {
        return res.status(401).json({ error: "Unauthorized" });
29
30
31
      jwt.verify(token, SERVER\_SECRET, (err, decoded) \Rightarrow {
32
33
        console.log("Decoded: " + JSOM.stringify(decoded));
34
        if (err) {
          return res.status(401).json({ error: "Unauthorized" });
35
36
        req.userIndex = decoded.userIndex;
37
38
        next();
      });
39
40 }:
```

```
41
42
    app.get("/admin/login", function (req, res) {
43
      const name = reg.guery.name;
      const password = req.query.password; // in a real app use hashed passwords!
44
45
      if (!name | !password) {
46
47
        res.status(400).send("Missing name or password");
48
        return;
      }
49
50
51
      let userIndex = -1;
52
      for (let i = 0; i < users.length; i++) {</pre>
53
        if (users[i].name == name & users[i].password == password) {
54
          userIndex = i;
          break:
55
56
57
      if (userIndex = −1) {
58
59
        res.status(401).send("Credentials invalid.");
60
        return;
61
62
      console.log(
        "Authenticated: " + users[userIndex].name + " " + users[userIndex].password,
63
64
65
      // Here, we can use the userIndex to identify the user;
66
      // but his only works as long as the user list is fixed.
67
      // In a real app use, e.g., a users's email.
68
69
      const token = jwt.sign({ userIndex: userIndex }, SERVER_SECRET, {
70
        expiresIn: "2h",
71
      });
      res.status(200).json({ token });
72
73
    });
74
75
    app.post("/admin/question", verifyToken, function (req, res) {
76
      const userIndex = req.userIndex;
77
      const question = req.body;
      console.log("Received question: " + question + " from user: " + users[userIndex].name);
78
79
     res.status(200).send("Question stored. Preliminary answer: 42.");
80
    });
81
82
    // Attention: a port like 6666 will not work on (most?) browsers
83
84 const port = 8080;
    var server = app.listen(port, function () {
85
     console.log(`Running at http://127.0.0.1:${port}/`);
86
87 |});
Client (JavaScript)
 1
    Initializes the login interface.
 2
 3
    */
 4
    document
      .getElementsByTagName("main")[0]
 5
       .replaceChildren(document.getElementById("log-in").content.cloneNode(true));
 б
 7
    document.getElementById("login-dialog").showModal();
    document.getElementById("login-button").addEventListener("click", login);
 8
 9
    let jwt = null; // JSON Web Token for authentication
10
11
```

```
12 async function login() {
13
      const name = document.getElementById("administrator").value;
      const password = document.getElementById("password").value;
14
      const urlEncodedName = encodeURIComponent(name);
15
16
      const urlEncodedPassword = encodeURIComponent(password);
17
      const response = await fetch(
18
        "http://" +
          location.host +
19
          "/admin/login?name=" +
20
21
          urlEncodedName +
22
          "%password=" +
          urlEncodedPassword,
23
24
      );
      if (response.status ≠ 200) {
25
        console.error("Login failed: " + response.status);
26
27
        return;
28
29
      const responseJSON = await response.json();
30
      jwt = responseJSON.token;
      console.log("Received JWT: " + jwt);
31
32
33
      document.getElementById("login-dialog").close();
34
35
      document
        .getElementsByTagName("main")[0]
36
        .replaceChildren(document.getElementById("logged-in").content.cloneNode(true));
37
      document.getElementById("enter-question-dialog").showModal();
38
39
      document.getElementById("send-question").addEventListener("click", sendQuestion);
40
41
    async function sendQuestion() {
42
      const question = document.getElementById("question").value;
43
44
      const response = await fetch("http://" + location.host + "/admin/question", {
45
46
        method: "POST",
47
        headers: {
          "Content-Type": "text/plain",
48
          Authorization: `Bearer ${jwt}`,
49
50
        },
        body: question,
51
52
53
      const text = await response.text();
      showAnswer(text);
54
55
56
57
    function showAnswer(text) {
      document.getElementById("answer-dialog").showModal(false);
58
      document.getElementById("answer-paragraph").textContent = text;
59
60 }
```

Alle Quellen:

admin.js admin.html admin.css server.mjs start_server.sh Users.json

Referenzen

■ HTML DOM API

Deklaration von Variablen (const und let)

```
// Der "Scope" ist auf den umgebenden Block begrenzt.
    // Eine Änderung des Wertes ist möglich.
 3
    let y = "yyy";
 4
    // Der "Scope" ist auf den umgebenden Block begrenzt.
 5
    // Eine Änderung des Wertes ist nicht möglich.
 6
    const z = "zzz";
 7
 8
 9
    log("y, z:", y, z);
10
11
    function doIt() {
      const y = "---";
12
13
      log("y, z:", y, z);
      return "";
14
15
16
   ilog('"doIt done"', doIt());
17
18 log("y, z:", y, z);
```

Um diesen und den Code auf den folgenden Folien ggf. mit Hilfe von Node.js auszuführen, muss am Anfang der Datei:

```
import { ilog, log, done } from "./log.mjs";
```

und am Ende der Datei:

```
done();
```

hinzugefügt werden.

Den entsprechenden Code der Module (log.mjs und später Queue.mjs) finden Sie auf:

https://github.com/Delors/delors.github.io/tree/main/web-javascript/code

Datentypen und Operatoren

```
console.log("Undefined -----");
   let u = undefined;
   console.log("u", u);
 3
 4
   console.log("Number -----");
 5
   let i = 1; // double-precision 64-bit binary IEEE 754 value
   let f = 1.0; // double-precision 64-bit binary IEEE 754 value
 7
   let l = 10_{000};
   let binary = 0b1010;
9
   console.log("0b1010", binary);
10
11
   let octal = 0012;
   console.log("0012", octal );
12
   let hex = 0xA;
13
14 | console.log("0xA", hex);
15
   console.log(
     Number.MIN_VALUE,
16
    Number.MIN_SAFE_INTEGER,
17
     Number.MAX_SAFE_INTEGER,
18
     Number.MAX_VALUE,
19
20
   );
21
   let x = NaN;
22
   let y = Infinity;
   let z = -Infinity;
2.3
24
   // Standard Operatoren: +, - , *, /, %, ++, --, **
25
   // Bitwise Operatoren: &, |, ^, ~, <<, >>, >>>
26
                     (operieren immer auf dem Ganzzahlwert der Bits)
27
  console.log("i =", i, "; i++ ", i++); // 1 oder 2?
28
   console.log("i =", i, "; ++i ", ++i); // 2 oder 3?
29
30
   console.log("2 ** 4 == 0 ", 2 ** 4);
   console.log("7 % 3 == ", 7 % 3);
31
32
   console.log("1 / 0 \equiv ", 1 / 0);
33
34
35 | console.log("BigInt -----");
   let ib = 1n;
36
37
   console.log(100n \implies PigInt(100));
   console.log(Number.MAX_SAFE_INTEGER + 2102); // 9007199254743092
38
   console.log(RigInt(Number.MAX_SAFE_INTEGER) + 2102n); // 9007199254743093n
39
40
41
   console.log("Boolean ------");
42
   let b = true; // oder false
43
    console.log("Boolean(undefined)", Boolean(undefined)); // true oder false?
44
    console.log(null = true ? "true" : "false"); // true oder false?
45
46
47
   console.log("(Quasi-)Logische Operatoren -----");
48
49
   console.log('1 && "1": ', 1 && "1");
  console.log('null && "1": ', null && "1");
50
   console.log("null && true: ", null && true);
51
   console.log("true && null: ", true && null);
52
53
   console.log("null && false: ", null 🍇 false);
   console.log("{} && true: ", {} && true);
54
55
  // Neben den Standardoperatoren: ``&\``, ``||``, ``!`` gibt es auch noch ``??``
56
57 // Der ``??``-Operator gibt den rechten Operanden zurück, wenn der linke Operand
```

```
// ``null`` oder ``undefined`` ist. Andernfalls gibt er den linken Operanden
 58
    // zurück.
 59
     // ``??`` ist der *nullish coalescing operator (??) (vergleichbar zu ||)*
 60
 61 console.log('1 ?? "1": ', 1 ?? "1");
 62 | console.log('null ?? "1": ', null ?? "1");
     console.log("null ?? true: ", null ?? true);
 63
     console.log("true ?? null: ", true ?? null);
 64
     console.log("null ?? false: ", null ?? false);
 65
     console.log("{} ?? true: ", {} ?? true);
 66
 67
     console.log('undefined ?? "1": ', undefined ?? "1");
 68
     console.log('undefined ?? "1": ', undefined ?? "1");
 69
     console.log("undefined ?? true: ", undefined ?? true);
 70
     console.log("true ?? undefined: ", true ?? undefined);
 71
     console.log("undefined ?? false: ", undefined ?? false);
 72
     console.log("undefined ?? undefined: ", undefined ?? undefined);
 73
 74
 75
 76
    console.log("Strings -----");
     let _s = "42";
 77
     console.log("Die Antwort ist " + _s + "."); // String concatenation
 78
 79
     console.log(`Die Antwort ist ${_s}.`); // Template literals (Template strings)
    // multiline Strings
 80
     console.log(
 81
        Die Antwort mag ${_s} sein,
 82
        aber was ist die Frage?`);
 83
 84
     console.log(String(42)); // "42"
 85
 86
 87
     console.log("Objekte -----");
 88
     let emptyObject = null;
 89
     | let anonymousObj = {
 90
 91
      i: 1,
      u: { j: 2, v: { k: 3 } },
 92
      toString: function () {
 93
       return "anonymousObj";
 94
 95
       "?" : "question mark"
 96
 97
     // Zugriff auf die Eigenschaften eines Objekts
 98
     anonymousObj.j = 2; // mittels Bezeichner ("j") (eng. Identifier)
 99
     anonymousObj["j"] = 4; // mittels String ("j")
100
    anonymousObj["k"] = 3;
101
     console.log("anonymous0bj:
                                                 ", anonymousObj);
102
     103
     delete anonymousObj["?"]; // delete dient dem Löschen von Eigenschaften
104
     delete anonymousObj.toString; // delete dient dem Löschen von Eigenschaften
105
106
     console.log("anonymousObj.toString() [original]", anonymousObj.toString());
    // Der Chain-Operator kann verwendet werden, um auf Eigenschaften (Properties)
107
108
    // von Objekten zuzugreifen, ohne dass eine Fehlermeldung ausgegeben wird,
     // wenn eine (höher-liegende) Eigenschaft nicht definiert ist.
109
110
     // Besonders nützlich beim Verarbeiten von komplexen JSON-Daten.
    console.log("anonymousObj.u?.v.k", anonymousObj.u?.v.k);
111
     console.log("anonymousObj.u.v?.k", anonymousObj.u.v?.k);
112
     console.log("anonymousObj.u.v?.z", anonymousObj.u.v?.z);
113
114
     console.log("anonymousObj.u.q?.k", anonymousObj.u.q?.k);
115
     console.log("anonymousObj.p?.v.k", anonymousObj.p?.v.k);
116
117
     // Nützliche Zuweisungen, um den Fall undefined und null gemeinsam zu behandeln:
```

```
119
120
121
122 console.log("Date -----");
     let date = new Date("8.6.2024"); // ACHTUNG: Locale-Settings
123
    console.log(date);
124
125
126
     console.log("Funktionen sind auch Objekte -----");
127
128 let func = function () {
     return "Hello World";
129
130
131
    console.log(func, func());
132
133
    console.log("Arrays -----");
134
135 let temp = undefined;
136 let $a = [1];
    console.log("let a = [1]; a, a.length", a, a.length");
137
138
    a.push(2); // append
    console.log("$a.push(2); $a", $a);
139
140
    temp = $a.unshift(0); // "prepend" → return new length
141
     console.log("temp = $a.unshift(0); temp, $a", temp, $a);
    temp = $a.shift(); // remove first element → return removed element
142
console.log("temp = $a.shift(); temp, $a", temp, $a);
    // Um zu prüfen ob eine Datenstruktur ein Array ist:
144
145
    console.log("Array.isArray($a)", Array.isArray($a));
    console.log("Array.isArray({})", Array.isArray({}));
146
147
    console.log("Array.isArray(1)", Array.isArray(1));
148
149
150 console.log("Symbols -----");
    let sym1 = Symbol("1"); // a unique and immutable primitive value
151
152
    let sym2 = Symbol("1");
    let obj1Values = { sym1: "value1", sym2: "value2" };
153
    console.log(obj1Values);
154
     console.log(`sym1 in ${JSON.stringify(obj1Values)}: `, sym1 in obj1Values);
155
    let obj2Values = { [sym1]: "value1", [sym2]: "value2" };
156
157 console.log(obj2Values);
    console.log(`sym1 in ${JSON.stringify(obj2Values)}: `, sym1 in obj2Values);
158
    console.log(obj1Values, " vs. ", obj2Values);
159
160
161 | console.log( { sym1 : "this", sym1 : "that" }); // ??? { sym1: "that" }
162 | console.log("sym1 = sym2", sym1 = sym2);
```

Funktionsdefinitionen

```
1 // Die Funktionsdeklaration der Funktion "hello" ist "hochgezogen" ( hoisted)
    // und kann hier verwendet werden.
    hello("Michael");
 3
 4
    function hello(person = "World" /* argument with default value */) {
 5
     log(`fun: Hello ${person}!`);
 6
 7
 8
    hello();
 9
    waitOnInput();
10
11
    const helloExpr = function () { // Anonymer Funktionsausdruck
12
    log("expr: Hello World!");
13
    };
14
15
    // Arrow Functions
16
17
    const times3 = (x) \Rightarrow x * 3;
    log("times3(5)", times3(5)); // 15
18
19
20 | const helloArrow = () ⇒ log("arrow: Hello World!");
21 | const helloBigArrow = () ⇒ {
     const s = "Hello World!";
22
23
     log("arrow: " + s);
24
     return s;
25
    };
    helloExpr();
26
27
    helloArrow();
28
    var helloXXX = function helloYYY() { // benannter Funktionsausdruck
29
30
     // "helloYYY" ist _nur_ innerhalb der Funktion sichtbar und verwendbar
     // "arguments" ist ein Arrays-vergleichbares Objekt
31
32
     // und enthält alle Argumente der Funktion
     log(`Hello: `, ...arguments); // "..." ist der "Spread Operator"
33
34
35
    helloXXX("Michael", "John", "Jane");
36
37
    waitOnInput();
38
    function sum(...args) {
39
     // rest parameter
40
      log("typeof args: " + typeof args + "; isArray: "+ Array.isArray(args));
41
      log("args: " + args);
42
      log("args:", ...args); // die Arraywerte werden als einzelne Args. übergeben
43
      return args.reduce((a, b) \Rightarrow a + b, 0); // function nesting
44
45
    log(sum(1, 2, 3, 4, 5)); // 15
46
    log(sum());
47
48
49
    /* Generator Functions */
50 function* fib() {
     // generator
51
52
     let a = 0,
53
       b = 1;
      while (true) {
54
       yield a;
55
56
        [a, b] = [b, a + b];
      }
57
```

```
58  }
59  const fibGen = fib();
60  log(fibGen.next().value); // 0
61  log(fibGen.next().value); // 1
62  log(fibGen.next().value); // 1
63  log(fibGen.next().value); // 2
64  /* Will cause an infinite loop:
65  for (const i of fib()) console.log(i);
66  // 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 ... */
```

Übung - JavaScript und Node.js erste Schritte

Voraussetzung: Installieren Sie Node.js (http://nodejs.org/).

1.7. Hello World in Node.js

Starten Sie die Konsole/Terminal und schreiben Sie ein einfaches JavaScript Programm, das "Hello World" ausgibt.

1.8. Hello World auf der JavaScript Console

Starten Sie einen Browser und aktivieren Sie die JavaScript Console in den Entwicklerwerkzeugen. Schreiben Sie ein einfaches JavaScript Programm, das "Hello World" ausgibt.

Übung - die JavaScript Konsole

1.9. Prototyping mit der JavaScript Konsole

Schreiben Sie ein kurzes JavaScript Programm, das programmatisch zum Ende des Dokuments scrollt. Hinweise:

- das von document. body referenziert HTML Element enthält den gesamten Inhalt des Dokuments
- die aktuellen Abmaße des Dokuments können Sie mit der Funktion window.getComputedStyle(<html Element>).height ermitteln; geben Sie den Wert auf der Konsole aus bevor Sie das Dokument scrollen; was fällt Ihnen auf?
- um zu scrollen, können Sie window.scrollTo(x,y) verwenden
- um den Integer Wert eines Wertes in Pixeln zu bestimmen, können Sie parseint verwenden

Vergleich von Werten und implizite Typumwandlung

```
// Gleichheit
                     = // mit Typumwandlung (auch bei \langle, \rangle, \leq, \geq)
    // strikt gleich 	≡ // ohne Typumwandlung
 3
    // strike Ungleichheit ≠ // ohne Typumwandlung
 4
 5
 6 \log('1 = "1": ', 1 = "1");
    \log('1 \equiv "1": ', 1 \equiv "1");
 7
   \log("1.0 = 1: ", 1 = 1.0);
 8
 9 \log("1.0 = 1: ", 1 = 1.0);
10 \log("1 = 1n: ", 1 = 1n);
    log("1 = 1n: ", 1 = 1n);
11
    log('1 < "1"', 1 < "1");
12
13 log('0 < "1"', 0 < "1");
14 \log('0 \leq ''0''', 0 \leq ''0'');
    log('"abc" ≤ "d"', "abc" ≤ "d");
15
16
    log('"asdf" == "as" + "df"', "asdf" == "as" + "df"); // unlike Java!
17
18
19
    log("NaN === NaN: ", NaN === NaN);
20
    \log(\text{"NaN} = \text{NaN}: \text{", NaN} = \text{NaN});
21 \log(\text{"null} \implies \text{NaN}: \text{", null} \implies \text{NaN});
    log("null = NaN: ", null = NaN);
22
23
    log("null == null: ", null == null);
    log("null = null: ", null = null);
24
    log("undefined == undefined: ", undefined == undefined);
25
    log("undefined = undefined: ", undefined = undefined);
26
    log("null == undefined: ", null == undefined);
27
    log("null = undefined: ", (null = undefined)+ "!");
28
29
30
    const a1 = [1, 2, 3];
31
32
    const a2 = [1, 2, 3];
    log("const a1 = [1, 2, 3]; a1 = [1, 2, 3]: ", a1 = [1, 2, 3]);
33
    \log(\text{"const a1} = [1, 2, 3]; \text{ a1} = \text{a1}: \text{", a1} = \text{a1});
34
35 \log("const a1 = [1, 2, 3]; a1 \implies a1: ", a1 \implies a1);
    log("const a1 = [1, 2, 3]; const a2 = [1, 2, 3]; a1 == a2: ", a1 == a2);
36
37
    log("const a1 = [1, 2, 3]; const a2 = [1, 2, 3]; a1 = a2: ", a1 = a2);
    log(
38
     "flatEquals(a1,a2):".
39
     a1.length = a2.length \& a1.every((v, i) \Rightarrow v = a2[i])
40
    );
41
42
43
    let firstJohn = { person: "John" };
44
    show('let firstJohn = { person: "John" };');
45
    let secondJohn = { person: "John" };
46
    show('let secondJohn = { person: "John" };');
47
    let basedOnFirstJohn = Object.create(firstJohn);
48
49
    show("let basedOnFirstJohn = Object.create(firstJohn)");
    log("firstJohn = firstJohn: ", firstJohn = firstJohn);
50
    \log(\text{"firstJohn} \implies \text{secondJohn});
51
    log("firstJohn = secondJohn: ", firstJohn = secondJohn);
52
    log("firstJohn == basedOnFirstJohn: ", firstJohn == basedOnFirstJohn);
53
    log("firstJohn = basedOnFirstJohn: ", firstJohn = basedOnFirstJohn);
54
55
56
57 {
```

```
58
      const obj = {
59
         name: "John",
60
         age: 30,
61
         city: "Berlin",
62
      };
       log("\nTyptests und Feststellung des Typs:");
63
64
       log("typeof obj", typeof obj);
65
       log("obj instanceof Object", obj instanceof Object);
       log("obj instanceof Array", obj instanceof Array);
66
67
68
      const obj = { a: "lkj" };
69
70
      const obj2 = Object.create(obj);
       log("obj2 instanceof obj.constructor", obj2 instanceof obj.constructor);
71
72
73
    log("\n?-Operator/if condition and Truthy and Falsy Values:");
74
    log('""', "" ? "is truthy" : "is falsy");
75
76 \log(\text{"f}()\text{"}, (() \Rightarrow \{\})? "is truthy" : "is falsy");
    log("Array ", Array ? "is truthy" : "is falsy");
log("obj ", {} ? "is truthy" : "is falsy");
77
78
79 log("undefined ", undefined ? "is truthy" : "is falsy");
80 log("null ", null ? "is truthy" : "is falsy");
81 log("0", 0 ? "is truthy" : "is falsy");
82 log("1", 1 ? "is truthy" : "is falsy");
```

NaN (Not a Number) repräsentiert das Ergebnis einer Operation die keinen sinnvollen Wert hat. Ein Vergleich mit NaN ist *immer* false. Um zu überprüfen, ob ein Wert NaN ist muss is well (<\alpha\left) verwendet werden.

Bedingungen und Schleifen

```
const arr = [1, 3, 4, 7, 11, 18, 29];
 2
    log("if-else_if-else:");
   if (arr.length = 7) {
 4
     ilog("arr.length = 7");
 5
    } else if (arr.length < 7) {
 7
      ilog("arr.length < 7");</pre>
    } else {
 8
9
      ilog("arr.length > 7");
10
11
    log("\nswitch (integer value):");
12
    switch (arr.length) {
13
14
      case 7:
        ilog("arr.length = 7");
15
        break:
16
17
      case 6:
        ilog("arr.length = 6");
18
19
        break;
20
      default:
        ilog("arr.length \neq 6 and \neq 7");
21
22
23
    log("\nswitch (string value):");
24
    switch ("foo") {
25
      case "bar":
26
        ilog("it's bar");
27
28
        break:
      case "foo":
29
30
        ilog("it's foo");
31
        break;
32
      default:
33
        ilog("not foo, not bar");
34
35
    log("\nswitch (integer - no type conversion):");
36
37
    switch (
      1 // Vergleich auf strikte Gleichheit (■)
38
    ) {
39
      case "1":
40
        ilog("string(1)");
41
42
        break;
43
      case 1:
        ilog("number(1)");
44
45
        break;
46
47
    ilog("\nfor-continue:");
48
49
    for (let i = 0; i < arr.length; i \leftrightarrow b) {
      const v = arr[i];
50
      if (v \% 2 = 0) continue;
51
52
      log(v);
53
54
55
    ilog("\n(for)-break with label:");
   | outer: for (let i = 0; i < arr.length; i \leftrightarrow) {
      for (let j = 0; j < i; j \leftrightarrow b) {
```

```
58
         if (j = 3) break outer;
 59
          log(arr[i], arr[j]);
       }
 60
     }
 61
 62
 63
      ilog("\nin (properties of Arrays; i.e. the indexes):");
     for (const key in arr) {
 64
       log(key, arr[key]);
 65
 66
 67
     ilog("\nof (values of Arrays):");
 68
     for (const value of arr) {
 69
 70
       log(value);
 71
 72
 73
      ilog("\nArray and Objects - instanceof:");
      log("arr instanceof Object", arr instanceof Object);
 74
      log("arr instanceof Array", arr instanceof Array);
 75
 76
 77
      const obj = {
       name: "John",
 78
 79
       age: 30,
       city: "Berlin",
 80
 81
 82
     ilog("\nin (properties of Objects):");
 83
     for (const key in obj) {
 84
       log(key, obj[key]);
 85
 86
 87
      /* TypeError: obj is not iterable
 88
      for (const value of obj) {
 89
         log(value);
 90
 91
 92
      */
 93
 94
        ilog("\nIteration über Iterables (here: Map):");
 95
       const m = new Man();
 96
 97
       m.set("name", "Elisabeth");
       m.set("alter", 50);
 98
 99
        log("Properties of m: ");
100
       for (const key in m) {
          log(key, m[key]);
101
102
103
       log("Values of m: ");
104
       for (const [key, value] of m) {
          log(key, value);
105
106
107
108
109
       ilog("\nWhile Loop: ");
110
111
       let c = 0;
       while (c < arr.length) {</pre>
112
         const v = arr[c];
113
         if (v > 10) break;
114
         log(v);
115
116
         C++;
117
118 }
```

```
119
120 {
121    ilog("\nDo-While Loop: ");
122    let c = 0;
123    do {
      log(arr[c]);
125    c++;
126    } while (c < arr.length);
127 }</pre>
```

Die Tatsache, dass insbesondere null als auch undefined falsy sind, wird of in Bedingungen ausgenutzt (z. B., if (!x)...).

Fehlerbehandlung

```
console.log("try-catch-finally - Grundlagen -----");
 2
 3
   try {
    let i = 1 / 0; // Berechnungen erzeugen nie eine Exception
 4
    console.log("i", i);
 5
 6 } catch {
    console.error("console.log failed");
 7
   } finally {
 8
9
    console.log("computation finished");
10
11
   console.log("Programmierfehler behandeln -----");
12
13 try {
   const obj = {};
14
   obj = { a: 1 };
15
16 | } catch ({ name, message }) {
   console.error(message);
17
18 | finally {
    console.log("object access finished");
19
20
21
console.log("Handling of a specific error -----");
23
   try {
    throw new RangeError("out of range");
24
25 } catch (error) {
26
   if (error instanceof RangeError) {
     const { name, message } = error;
27
     console.error("a RangeError:", name, message);
28
    } else {
29
30
     throw error;
    }
31
32 } finally {
    console.log("error handling finished");
33
34 }
```

In JavaScript können während der Laufzeit Fehler auftreten, die (z. B.) in Java während des kompilierens erkannt werden.

.....

Übung - Bedingungen und Schleifen

1.10. removeNthElement

Implementieren Sie eine Funktion, die ein Array übergeben bekommt und ein neues Array zurückgibt in dem jedes n-te Element nicht vorkommt.

Beispiel: removeNthElement([1,2,3,4,5,6,7], 2) \Rightarrow [1,3,5,7]

- Schreiben Sie Ihren Code in eine JavaScript Datei und führen Sie diese mit Hilfe von Node.js aus.
- Testen Sie Ihre Funktion mit verschiedenen Eingaben und lassen Sie sich das Ergebnis ausgeben (z. B. console.log(removeNthElement([1,2,3,4,5,6,7],2)))!

Übung - Fehlerbehandlung

1.11. removeNthElement mit Fehlerbehandlung

- Erweitern Sie die Implementierung von removeNthElement so, dass die Funktion einen Fehler wirft, wenn das übergebene Array kein Array ist oder wenn der zweite Parameter kein positiver Integer ist.
- Testen Sie alle Fehlerzustände und fangen Sie die entsprechenden Fehler ab (catch) und geben Sie die Nachrichten aus.

Übung - Funktionen

1.12. Einfacher RPN Calculator

Implementieren Sie einen einfachen RPN (Reverse Polish Notation) Calculator, der eine Liste von Zahlen und Operatoren (+, -, *, /) als Array entgegennimmt und das Ergebnis berechnet.

Nutzen Sie keine if oder switch Anweisung, um die Operatoren zu unterscheiden. Nutzen Sie stattdessen ein Objekt. Sollte der Operator unbekannt sein, dann geben Sie eine entsprechende Fehlermeldung aus.

Beispiel: eval([2,3,"+",4,"*"]) \Rightarrow 20

Variables (var)

(Neuer Code sollte var nicht mehr verwenden!)

```
let y = "yyy"; // wie zuvor
    const z = "zzz";
 2
 3
    // Der Gültigkeitsbereich von var ist die umgebende Funktion oder der
 4
    // globale Gültigkeitsbereich.
 5
    // Die Definition ist hochgezogen (eng. "hoisted") (initialisiert mit undefined);
 6
    var x = "xxxx";
 7
    function sumIfDefined(a, b) {
9
10
     // ! Der folgende Code ist NICHT empfehlenswert!
     // Er dient der Visualisierung des Verhaltens von var.
11
12
     if (parseInt(a)) {
        var result = parseInt(a);
13
14
      } else {
15
        result = 0;
      }
16
17
      const bVal = parseFloat(b);
      if (bVal) {
18
19
        result == bVal;
      }
20
21
      return result;
    }
22
23
    ilog("sumIfDefined()", sumIfDefined()); // 0
    ilog("sumIfDefined(1)", sumIfDefined(1)); // 1
25
    ilog("sumIfDefined(1, 2)", sumIfDefined(1, 2)); // 3
26
    ilog('sumIfDefined(1, "2")', sumIfDefined(1, "2")); // 3
27
    ilog("undefined + 2", undefined + 2);
28
    ilog('sumIfDefined(undefined, "2")', sumIfDefined(undefined, "2")); // 2
29
30
    function global_x() {
31
      ilog("global_x():", x, y, z);
32
    }
33
34
    function local var x() {
35
36
      ilog("local_var_x(): erste Zeile (x)", x);
37
      var x = 1; // the declaration of var is hoisted, but not the initialization
38
39
      let y = 2;
      const z = 3:
40
41
      ilog("local_var_x(): letzte Zeile (x, y, z)", x, y, z); // 1 2 3
42
    }
43
44
45
    global_x();
46
    local_var_x();
47
    ilog("nach global_x() und local_var_x() - x, y, z:", x, y, z);
48
49
50
51 // Hier, ist nur die Variablendeklaration (helloExpr) "hoisted", aber nicht
    // die Definition. Daher kann die Funktion nicht vorher im Code aufgerufen
52
53 // werden!
54
   try {
     helloExpr();
55
56 | catch ({error, message}) {
```

```
log("calling helloExpr() failed:", error, "; message: ", message);

var helloExpr = function () {
   log("expr: Hello World!");
};

// ab jetzt funktioniert es
helloExpr();
```

Destrukturierung (■ *Destructuring***)**

```
log("Array Destructuring:");
1
 2
    let [val1, val2] = [1, 2, 3, 4];
 3
    ilog("[val1, val2] = [1, 2, 3, 4]:", "val1:", val1, ", val2:", val2); // 1
 4
 5
    log("Object Destructuring:");
 6
 7
    let { a, b } = { a: "aaa", b: "bbb" };
 8
9
    ilog('let { a, b } = { a: "aaa", b: "bbb" }: ', "a:", a, ", b:", b); // 1
10
    let { a: x, b: y } = { a: "aaa", b: "bbb" };
11
    ilog('let { a: x, b: y } = { a: "aaa", b: "bbb" }: ', "x:", x, ", y:", y); // 1
12
13
    let { a: u, b: v, ...w } = { a: "+", b: "-", c: "*", d: "/" };
14
15
      'let { a: u, b: v, ...w } = { a: "+", b: "-", c: "*", d: "/" }:',
16
      "u:",
17
18
      u,
      ", V:",
19
20
      ٧,
21
     ", W:",
      JSON. stringify(w), // just for better readability/comprehension
22
23
24
25
    let { k1, k2 } = { a: "a", b: "b" };
    ilog('let { k1, k2 } = { a: "a", b: "b" }:', "k1:", k1, ", k2:", k2);
27 // "undefined undefined", weder k1 noch k2 sind definiert
```

JSON (JavaScript Object Notation)

```
const someJSON = `{
        "name": "John",
 2
        "age": 30,
 3
        "cars": {
 4
            "American": ["Ford"],
 5
            "German": ["BMW", "Mercedes", "Audi"],
 6
            "Italian": ["Fiat", "Alfa Romeo", "Ferrari"]
 7
 8
 9
10
11
    // JSON.parse(...) JSON String \Rightarrow JavaScript Object
12
    const someObject = JSON.parse(someJSON);
13
14
    someObject.age = 31;
15
    someObject.cars.German.push("Porsche");
16
17 | someObject.cars.Italian.pop();
18 console.log(someObject);
19
20 // JSON.stringify(...) JavaScript Object ⇒ JSON String
console.log(JSON.stringify(someObject, null, 2));
```

JSON requires that keys must be strings and strings must be enclosed in double quotes.

Reguläre Ausdrücke

- Eingebaute Unterstützung basierend auf entsprechenden Literalen (Strings in "/") und einer API
- inspiriert von der Perl Syntax
- Methoden auf regulären Objekten: test (e.g., Regixn. test (String)).
- Methoden auf Strings, die reguläre Ausdrücke verarbeiten: search, match, replace, split, ...

```
1 {
      const p = /.*[1-9]+H/; // a regexp
 2
      console.log(p.test("ad13H"));
 3
      console.log(p.test("ad13"));
 4
      console.log(p.test("13H"));
 5
 6
 7
      const p = /[1-9] + H/g;
 8
      const s = "1H, 2H, 3P, 4C";
 9
      console.log(s.match(p));
10
11
      console.log(s.replace(p, "XX"));
12 }
```

Klassen und Vererbung

```
1 | class Figure {
 2
      calcArea() {
        throw new Error("calcArea is not implemented");
 3
 4
 5
    class Rectangle extends Figure {
 6
 7
      height;
 8
      width;
 9
      constructor(height, width) {
10
11
        super();
        this.height = height;
12
13
        this.width = width;
      }
14
15
16
      calcArea() {
        return this.height * this.width;
17
      }
18
19
      get area() {
20
21
      return this.calcArea();
22
23
      set area(value) {
24
        throw new Error("Area is read-only");
25
26
    }
27
28
    const r = new Rectangle(10, 20);
29
30
    console.log("r instanceof Figure", r instanceof Figure); // true
    console.log(r.width);
31
32 console.log(r.height);
    console.log(r.area); // 200
33
34
35 try {
     r.area = 300; // Error: Area is read-only
36
    } catch (e) {
37
      console.error(e.message);
38
39 }
```

Grundlagen von ECMAScript Modulen

Queue.mjs exportiert die Klasse Queue

```
/* Modul für den Datentyp Warteschlange (Queue). */
    export class Queue {
 2
 3
      #last = null; // private field
      #first = null;
 4
 5
      constructor() {} // "default constructor"
      enqueue(elem) {
 6
 7
        if (this.#first ≡ null) {
 8
          const c = { e: elem, next: null };
 9
          this.#first = c;
10
          this.#last = c;
        } else {
11
          const c = { e: elem, next: null };
12
          this.#last.next = c;
13
          this.#last = c;
14
        }
15
      }
16
17
      dequeue() {
        if (this.#first == null) {
18
         return null;
19
        } else {
20
          const c = this.#first;
21
22
          this.#first = c.next;
23
          return c.e;
        }
24
      }
25
      head() {
26
27
        if (this.#first ≡ null) {
          throw new Error("Queue is empty");
28
29
        } else {
30
          return this.#first.e;
        }
31
      }
32
      last() {
33
34
        if (this.#first == null) {
          throw new Error("Queue is empty");
35
36
        } else {
37
          return this.#last.e;
        }
38
39
      isEmpty() {
40
41
        return this.#first == null;
42
43 }
```

log.mjs verwendet (import) die Klasse Queue und exportiert Funktionen zum Loggen

```
import { Queue } from "./Queue.mjs"; // import des Moduls "Queue.mjs"
 2
 3
    const messages = new Queue();
 4
 5
    export function log(...message) {
 6
      if (messages.isEmpty()) {
 7
        messages.enqueue(message);
 8
        message.unshift("\n");
 9
10
        messages.last().push(...message);
11
```

12 |}

.....

ECMAScript Module verwenden immer den strict mode.

Import Statements erlauben das selektierte importieren als auch das Umbenennen von importierten Elementen (z. B., import { Queue as Q } from "./Queue.mjs";).

Alles ist ein Objekt

- this ist ein "zusätzlicher" Parameter, dessen Wert von der aufrufenden Form abhängt
- this ermöglicht den Methoden den Zugriff auf ihr Objekt
- this wird zum Zeitpunkt des Aufrufs gebunden (außer bei Arrow-Funktionen)

```
//"use strict";
 1
 2
    function counter () {
 3
        // console.log(this \equiv globalThis); // true
 4
 5
        if(this.count) // this is the global object if we don't use strict mode
 6
            this.count ++;
 7
        else {
            this.count = 1;
 8
 9
10
        return this.count;
11
12
13
14
    const counterExpr = function () {
15
        if(this.count)
16
            this.count ++;
        else {
17
            this.count = 1;
18
19
20
        return this.count;
21
    }
22
23
    const counterArrow = () \Rightarrow \{
24
        console.log(this);
25
26
        console.log(this = globalThis);
        this.count = this.count ? this.count + 1 : 1;
27
        return this.count;
28
29
30
31
    console.log("\nCounter");
32 console.log(counter()); // 1
    console.log(counter()); // 2
33
    console.log(`Counter (${globalThis.count})`);
34
35
36 console.log("\nCounterExpression");
    console.log(counterExpr()); // 3
37
    console.log(counterExpr()); // 4
38
39
    console.log("\nCounter");
40
41
    const obj = \{\};
    console.log(counter.apply(obj)); // 1 - we set a new "this" object!
42
    console.log(counterExpr.apply(obj)); // 2
43
44
    console.log(`\nCounterArrow (${this.count})`);
45
    console.log(counterArrow.apply(obj)); // 1
46
    console.log(counterArrow.apply(undefined)); // 2
47
    console.log(counterArrow.apply()); // 3
48
    console.log(counterArrow.apply(obj)); // 4
49
50 | console.log(counterArrow.apply({})); // 5
51
    console.log("\nCounter (global)");
52
53 | console.log(counter());
```

Partial Function Application

```
1 | function add(x, y) {
 2
      return x + y;
 3
 4
 5 // Partial function application:
   const add2 = add.bind(null, 2); // "null" is the value of "this"
    console.log(add2(3));
 7
 8
 9
    function addToValue(b) {
10
     return this.x + b;
11
12
console.log(addToValue.call({x : 0}, -101));
```

Prototype basierte Vererbung

Verwendung von Object. create zur Initialisierung der Prototype Chain:

Verwendung der Eigenschaften von Prototypen:

```
const p = { s : "p" };
2
  const c = Object.create(p);
3
  const gc = Object.create(c);
4 gc.t = "q";
    qc
                                С
                                                         р
         proto
                                                             proto
                                    proto
                                                                                 →Object.prototype
                   "q"
                                                                        "p"
           t
                                                                S
  gc.s = "gc"
   console.log(gc.s); // gc
7
   delete gc.s;
8 | console.log(gc.s); // p
```

Pseudoclassical Inheritance

```
// constructor for Person objects:
    function Person(name, title){ this.name = name; this.title = title; }
 2
    Person.prototype.formOfAddress = function (){
 3
       const foa = "Dear ":
 4
       if(this.title){ foa += this.title+" "; }
 5
       return foa + this.name;
 6
 7
 8
    function Student(name, title, id, email) {
       Person.call(this, name, title); // super constructor call
 9
       this.id = id;
10
       this.email = email;
11
12
    Student.prototype = Object.create(Person.prototype);
13
    Student.prototype.constructor = Student;
14
15
    const aStudent = new Student("Emily Xi", "Mrs.", 12441, 'emily@xi.de');
```

Objektabhängigkeiten

```
function Person(name, title){ ... }

Person.prototype.formOfAddress = function (){ ... }

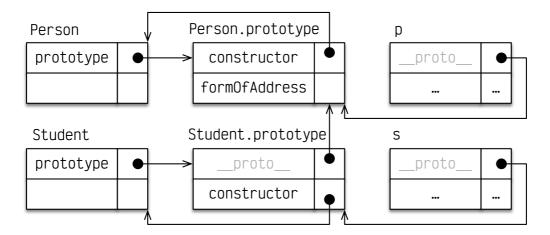
function Student(name, title, id, email) { ... }

Student.prototype = Object.create(Person.prototype);

Student.prototype.constructor = Student;

const p = new Person(...);

const s = new Student(...);
```



Die Eigenschaft prototype einer Funktion (F) verweist auf das Objekt, dass als Prototype (__proto__) verwendet wird, wenn die Funktion als Konstruktor verwendet wird. D. h. im Falle einer Instantiierung von F (d. h. const newF = new F()) wird das Objekt, das durch F.prototype referenziert wird, als Prototype (newF. proto) des neu erstellten Objekts (newF) verwendet.

```
1 // Prototypen
    console.log("{}.__proto__: ",{}.__proto__);
 2
    console.log("Array.prototype: ",Array.prototype);
 3
    console.log("Array.prototype.__proto__: ",Array.prototype.__proto__);
    console.log("Object.prototype: ",Object.prototype);
 5
    console.log("Object.__proto__: ",Object.__proto__);
 6
 7
    let o = { created: "long ago" };
 8
    var p = Object.create(o);
 9
    console.log("Object.getPrototypeOf(o): " + Object.getPrototypeOf(o));
10
    console.log("o.isPrototypeOf(p):" + o.isPrototypeOf(p));
11
12 | console.log("Object.prototype.isPrototypeOf(p): " + Object.prototype.isPrototypeOf(p));
```

Praktische Verwendung von Prototypen basierter Vererbung

```
let a = [1, 10, 100, 1000];
    try { console.log(a.fold()); } catch (error) {
     console.log("error: ", error.message);
 3
 4
 5
    // - ATTENTION! -----
 7
    // ADDING FUNCTIONS TO Array.prototpye IS NOT RECOMMENDED! IF ECMAScript
    // EVENTUALLY ADDS THIS METHOD (I.E. fold) TO THE PROTOTYPE OF ARRAY OBJECTS,
 9
   // IT MAY CAUSE HAVOC.
    Array.prototype.fold = function (f) {
10
11
     if (this.length 	≡ 0) {
        throw new Error("array is empty");
12
      } else if (this.length == 1) {
13
      return this[0];
14
     } else {
15
       let result = this[0];
16
        for (let i = 1; i < this.length; <math>i \leftrightarrow t) {
17
         result = f(result, this[i]);
18
19
        return result;
20
21
     }
    };
22
23
24 | console.log(a.fold((u, v) \Rightarrow u + v));
```

DOM Manipulation

```
<html lang="en">
 2
        <head>
 3
            <meta charset="utf-8" />
            <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 4
 5
            <title>DOM Manipulation with JavaScript</title>
 6
            <script>
                function makeScriptsEditable() {
 7
 8
                     const scripts = document.getElementsByTagName("script");
 9
                     for (const scriptElement of scripts) {
10
                         scriptElement.contentEditable = false;
11
                         const style = scriptElement.style;
12
                         style.display = "block";
13
                         style.whiteSpace = "preserve";
                         style.padding = "1em";
14
                         style.backgroundColor = "yellow";
15
                    }
16
                 }
17
18
            </script>
19
        </head>
20
        <body>
            <h1>DOM Manipulation with JavaScript</h1>
21

id="demo">This is a paragraph.
22
23
            <button
                 type="button"
24
25
                 onclick="
                         document.getElementById('demo').style.color = 'red';
26
27
                        makeScriptsEditable();
                         document.guerySelector('button').style.display = 'none';"
28
29
30
                Magic!
            ⟨/button⟩
31
32
33
            <script>
                const demoElement = document.getElementById("demo");
34
35
                const style = demoElement.style;
                 demoElement.addEventListener(
36
37
                     "mouseover",
                    () ⇒ (style.color = "green"),
38
                ):
39
                demoElement.addEventListener(
40
                     "mouseout",
41
                    () ⇒ (style.color = "unset"),
42
                );
43
            ⟨/script⟩
44
45
            Position der Mouse: <span id="position"></span>
46
            <script>
47
                 window.addEventListener("mousemove", () \Rightarrow {
48
49
                     document.getElementById("position").innerHTML =
                         `(${event.clientX}, ${event.clientY})`;
50
                }):
51
            ⟨/script⟩
52
        </body>
53
    </html>
54
```

Minimaler Server mit Express JS

```
1 // "express" and "cors" are CommonJS modules, which requires us to use the
 2
    // "default import" syntax.
 3
    import express from "express";
 4
    // Cross-Origin Resource Sharing (CORS); This is required to allow the browser
 5
    // using a different domain to load the HTML to make requests to this server.
 6
    // I. e., we can use the HTML file from the "web-javascript" project to make
 7
 8
    // requests to this server.
    import cors from "cors";
 9
    const APP PORT = 5080;
10
11
12
    const app = express();
13
14 | app.get("/users", cors(), function (req, res) {
     res.set("Content-Type", "application/json");
15
      res.end(`{
16
           "user1" : {
17
              "name" : "dingo",
18
               "password" : "1234"
19
               "profession" : "chef",
20
               "id": 1
21
            },
22
23
            "user2" : {
24
25
               "name" : "ringo",
               "password" : "asdf",
26
               "profession" : "boss",
27
               "id": 3
28
29
30
         }`);
    });
31
32
    app.listen(APP PORT, function () {
33
     console.log(`Users App @ http://127.0.0.1:${APP_PORT}`);
34
35 |});
```

Express ist ein minimalistisches Web-Framework für Node.js, das die Entwicklung von Webanwendungen vereinfacht. Die Installation kann über einen Packagemanager erfolgen.

Installieren Sie (z. B.) pnpm (https://pnpm.io/) und nutzen Sie danach pnpm, um die benötigten Module zu installieren:

```
$ pnpm init
$ pnpm install express
```

Danach starten Sie Ihren Server mit:

```
node --watch UsersServer.mjs
```

Interaktion mit Server mit Hilfe von Fetch

```
<html lang="en">
        <head>
 2
            <meta charset="utf-8" />
 3
            <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 4
 5
            <title>Eventhandling</title>
        </head>
 6
 7
        <body>
            <script>
 8
 9
                /* Using Promises:
                function getUsers() {
10
                     fetch('http://127.0.0.1:4080/users')
11
                         .then(response \Rightarrow response.json())
12
                         .then(users \Rightarrow {
13
                             const usersElement = document.getElementById('users');
14
                             usersElement.innerText = JSON.stringify(users);
15
16
                        });
                        }
17
                */
18
19
                /* Using async/await: */
20
21
                async function getUsers() {
                     let response = await fetch("http://127.0.0.1:5080/users");
22
23
                     let users = await response.json();
                     const usersElement = document.getElementById("users");
24
                     usersElement.innerText = JSON.stringify(users);
25
26
            ⟨/script⟩
27
28
            <div id="users"></div>
29
30
            <button onclick="getUsers()">Get Users
        </body>
31
32 \langle html>
```

Beispiel - Rumpf einer einfachen Webanwendung ("Quizzy")

Im Folgenden verwenden wir zur Client-/Server-Kommunikation insbesondere Websockets.

Server

```
const express = require('express');
    const app = express();
 3
    const expressWs = require('express-ws')(app);
 5
    let clients = 0;
 7
    let playerWSs = [];
 8
    let adminWS = null;
9
10
    let answersCount = 0;
11
    let correctAnswersCount = 0;
12
    app.use(express.static('.')); // required to serve static files
13
14
15
16
    function sendCurrentPlayers() {
        if (adminWS ‱ playerWSs.length > 0) {
17
            allPlayers = playerWSs
18
19
                 .filter(player ⇒ player.name)
                 .map(player ⇒ { return { "id": player.id, "name": player.name } })
20
            console.log("Sending current players: " + JSON.stringify(allPlayers));
21
             adminWS.send(JSOM.stringify({ "type": "players", "players": allPlayers }));
22
23
24
25
    function sendNextQuestion() {
2.6
27
        answersCount = 0;
28
        correctAnswersCount = 0;
29
        const question = "What is the capital of France?";
        const answers = ["Paris", "London", "Berlin", "Madrid"];
30
        const correct = "Paris";
31
32
        const nextQuestion = JSOM.stringify({
33
            "type": "question",
34
            "question": question,
35
             "answers": ["Paris", "London", "Berlin", "Madrid"]
36
        })
37
        playerWSs.forEach(player \Rightarrow player.ws.send(nextQuestion));
38
        adminWS.send(JSON.stringify({
39
            "type": "question",
40
            "question": question,
41
            "answers": answers,
42
             "correct": correct
43
        }));
44
45
46
    function sendResults() {
47
        const results = playerWSs.map(player \Rightarrow {
48
            return { "id": player.id, "name": player.name, "wins": player.wins }
49
50
51
        const sortedResults = results.sort((a, b) \Rightarrow b.wins - a.wins);
52
        const resultsMsg = JSON.stringify({
            "type": "results",
53
            "results": sortedResults
54
```

```
55
         playerWSs.forEach(player ⇒ player.ws.send(resultsMsg));
 56
          adminWS.send(resultsMsg);
 57
 58
 59
 60
 61
      function handleAnswer(clientId, answer) {
 62
          const correct = answer.answer == "Paris";
 63
          const player = playerWSs.find(player ⇒ player.id == clientId);
 64
 65
          if (correct) {
             if (correctAnswersCount == 0) {
 66
 67
                  player.wins₩;
 68
              correctAnswersCount +++;
 69
         }
 70
 71
          answersCount++;
         if (answersCount == playerWSs.length) {
 72
 73
             // sendNextQuestion();
 74
              sendResults();
         } else {
 75
 76
             adminWS.send(JSON.stringify({
 77
                  "type": "answers",
                  "count": answersCount,
 78
 79
                  "correctAnswersCount": correctAnswersCount
             }));
 80
         }
 81
 82
 83
 84
      app.ws('/player', function (ws, request) {
 85
          const clientId = clients++;
 86
          const playerData = { "ws": ws, "id": clientId, "wins": 0 }
 87
          playerWSs.push(playerData);
 88
 89
         ws.onmessage = function (event) {
 90
             message = JSON.parse(event.data);
 91
              switch (message.type) {
                  case "registration":
 92
 93
                      const name = message.name;
                      console.log("Registration: " + clientId + "/" + name);
 94
 95
                      playerData.name = name;
 96
                      sendCurrentPlayers();
 97
                      break;
 98
 99
                  case "answer":
100
                      const answer = message;
101
                      handleAnswer(clientId, answer);
102
                      break;
103
104
                      console.log("Unknown message: " + message);
105
                      break;
106
107
              }
108
         };
         ws.onclose = function () {
109
              console.log("Player disconnected: " + clientId);
110
              playerWSs = playerWSs.filter(player ⇒ player.id ≠ clientId);
111
              sendCurrentPlayers();
112
113
          };
         ws.onerror = function () {
114
```

```
115
              console.log("Player error: " + clientId);
116
              playerWSs = playerWSs.filter(player ⇒ player.id ≠ clientId);
117
              sendCurrentPlayers();
         };
118
119
     });
120
     app.ws('/admin', function (ws, req) {
121
         adminWS = ws;
122
         sendCurrentPlayers(); // when admin registers her/himself, send current players
123
         ws.onmessage = function (event) {
124
125
             message = JSON.parse(event.data);
126
             switch (message.type) {
127
                  case "start":
                      console.log("Start game");
128
                      sendNextOuestion();
129
130
                      break;
131
                  default:
                      console.log("Unknown message: " + message);
132
133
                      break;
             }
134
135
         };
136
137
         ws.onclose = (event) \Rightarrow {
138
              console.log("Admin disconnected");
139
              adminWS = null;
140
              sendCurrentPlayers();
         };
141
142
         ws.onerror = (event) \Rightarrow {
143
             console.log("Admin error: " + event);
144
145
              sendCurrentPlayers();
         };
146
147
148 });
149
150
     var server = app.listen(8800, function () {
151
         console.log("Quizzy running at http://127.0.0.1:8800/");
152
153 })
```

Client - Players

```
<!DOCTYPE html>
 2
    <html lang="en">
 3
 4
    <head>
 5
        <script>
             const ws = new WebSocket("ws://localhost:8800/player");
 6
 7
             ws.onmessage = (event) \Rightarrow {
 8
                 const data = JSOM.parse(event.data);
 9
                 switch (data.type) {
                     case "question":
10
                         console.log("Question: " + data.question);
11
12
                         showQuestion(data);
13
                         break;
14
                     case "results":
15
                         const main = document.getElementById("main")
                         main.innerText = "Results: " + event.data;
16
17
                         break;
18
                     default:
19
                         console.log("Unknown message: " + data);
```

```
20
                         break;
21
2.2
             };
             ws.onclose = (event) \Rightarrow {
23
                 console.log("Connection closed: " + event);
24
25
             ws.onerror = (event) \Rightarrow {
26
                 console.error("Error: " + event);
27
28
29
             function showQuestion(data) {
30
31
                 const main = document.getElementById("main")
32
                 main.innerHTML = `<h1>Question</h1>${data.question}`;
33
                 function createAnswerButton(answer) {
34
                     const button = document.createElement("button");
35
                     button.innerText = answer;
36
                     button.onclick = submitAnswer(answer);
37
38
                     return button;
                 }
39
40
41
                 for (answer of data.answers) {
                     main.appendChild(createAnswerButton(answer));
42
43
             }
44
45
             function submitAnswer(answer) {
46
                 return () \Rightarrow {
47
48
                     ws.send(JSON.stringify({
                         "type": "answer",
49
                         "answer": answer
50
51
                     }));
                     doWait();
52
                 }
53
54
55
             function submitUsername() {
56
                 const name = document.getElementById("username").value;
57
                 ws.send(JSON.stringify({
58
                     "type": "registration",
59
                     "name": name
60
61
                 }));
62
                 doWait();
63
64
65
             function doWait() {
66
                 const main = document.getElementById("main");
67
                 main.innerHTML = "Waiting for other players...";
68
             }
69
         </script>
70
71
72
    <body>
73
74
        <main id="main">
             <form>
75
                 <input type="text" id="username" placeholder="Username">
76
77
                 <button type="button" onclick="submitUsername();">Submit</button>
78
             ⟨form>
         ⟨/main⟩
79
80 </body>
```

```
81
82 \/html>
```

Client - Admin

```
2
    <html lang="en">
 3
    <head>
 4
 5
         <script>
             const ws = new WebSocket("ws://localhost:8800/admin");
 6
 7
 8
             ws.onmessage = (event) \Rightarrow {
                 const data = JSON.parse(event.data);
 9
                 console.log("Received: " + event.data);
10
                 switch (data.type) {
11
                     case "players":
12
                          const players = document.getElementById("players")
13
14
                          players.innerText =
15
                              "["+data.players.length + " players] " +
                              data.players
16
17
                                  .map(player \Rightarrow player.id + ": " + player.name)
                                  .join(", ");
18
19
                          break:
20
                     case "question":
21
                         showQuestion(data);
22
                          break:
                     case "results":
23
                         const main = document.getElementById("main")
24
                          main.innerText = "Result: " + event.data;
25
26
                          break:
27
                     default:
                          console.log("unknown: " + event.data);
28
29
                          break;
                 }
30
             };
31
32
             ws.onclose = (event) \Rightarrow {
33
                 console.log("Connection closed: " + event);
34
35
                 const main = document.getElementById("main")
                 main.innerText = "Connection closed - you need to restart.";
36
37
             };
             ws.onerror = (event) \Rightarrow {
38
                 console.log("Connection error: " + event);
39
             }:
40
41
             function startGame() {
42
                 ws.send(JSON.stringify({"type": "start"}));
43
44
45
             function showQuestion(data) {
46
                 document.getElementById("main").innerText = `
47
                     question: ${data.question}; correct answer: ${data.correct}
48
49
             }
50
         </script>
51
    </head>
52
53
54
    <body>
55
         <main id="main">
56
             <h1>Players</h1>
```

Die Implementierung dient nur dazu die grundlegenden Konzepte zu verdeutlichen. Es fehlen **viele** Aspekte wie z. B., Sicherheit.

Authentifizierung mit JWT (und Express)

Im Folgenden wird primär die Verwendung eines JWTs zur Authentifizierung von Benutzern demonstriert.

Die initiale Authentifizierung, die im folgenden Beispiel über ein per get-Request übermittelten Benutzernamen und Passwort erfolgt, ist **nicht sicher**. In einer realen Anwendung sollte für die initiale Authentifizierung ein sicherer Mechanismus verwendet werden. Eine Möglichkeit wäre z. B. die Verwendung von DIGEST Authentication (nicht empfohlen bzw. nur für einfachste Fälle). Sinnvoll wäre Basic Authentication *in Verbindung mit HTTPS* oder zum Beispiel der Einsatz von OAuth.

Warnung

Basic Authentication ohne HTTPS ist nicht sicher!

D.h. *Basic Authentication* ist genauso unsicher wie die hier gezeigte Lösung für die initiale Authentifizierung.

Server

```
import express from "express";
    import fs from "fs";
   import path from "node:path";
 3
    import { fileURLToPath } from "url";
 5 import jwt from "jsonwebtoken";
   import crypto from "crypto";
 7
    import bodyParser from "body-parser";
 8
 9
    const app = express();
10
    const SERVER_SECRET = crypto.randomBytes(64).toString("hex");
11
    const users = JSON.parse(
12
13
     fs.readFileSync(
14
        path.resolve(path.dirname(fileURLToPath(import.meta.url)), "users.json"),
15
        "utf8",
      ),
16
    );
17
    console.log("Users: " + JSON.stringify(users));
18
19
    app.use(express.static("."));
20
    app.use(express.json());
21
    app.use(bodyParser.text());
22
23
    const verifyToken = (req, res, next) \Rightarrow {
24
      console.log("Headers: " + JSON.stringify(req.headers));
25
26
      const token = req.headers["authorization"].split(" ")[1];
27
      if (!token) {
28
        return res.status(401).json({ error: "Unauthorized" });
29
30
31
      jwt.verify(token, SERVER\_SECRET, (err, decoded) \Rightarrow {
32
        console.log("Decoded: " + JSON.stringify(decoded));
33
34
          return res.status(401).json({ error: "Unauthorized" });
35
36
        req.userIndex = decoded.userIndex;
37
38
        next();
      });
39
    };
40
41
42 app.get("/admin/login", function (req, res) {
```

```
const password = req.query.password; // in a real app use hashed passwords!
44
45
       if (!name | !password) {
46
        res.status(400).send("Missing name or password");
47
        return;
48
49
50
51
      let userIndex = -1;
52
      for (let i = 0; i < users.length; <math>i \leftrightarrow b) {
53
        if (users[i].name == name & users[i].password == password) {
54
          userIndex = i;
55
          break;
        }
56
57
      if (userIndex = -1) {
58
         res.status(401).send("Credentials invalid.");
59
        return;
60
61
      }
62
      console.log(
        "Authenticated: " + users[userIndex].name + " " + users[userIndex].password,
63
64
65
      // Here, we can use the userIndex to identify the user;
66
      // but his only works as long as the user list is fixed.
67
68
      // In a real app use, e.g., a users's email.
      const token = jwt.sign({ userIndex: userIndex }, SERVER_SECRET, {
69
70
        expiresIn: "2h",
71
72
      res.status(200).json({ token });
    });
73
74
75 app.post("/admin/question", verifyToken, function (reg, res) {
76
      const userIndex = req.userIndex;
77
      const question = req.body;
      console.log("Received question: " + question + " from user: " + users[userIndex].name);
78
79
80
      res.status(200).send("Ouestion stored. Preliminary answer: 42.");
   });
81
82
    // Attention: a port like 6666 will not work on (most?) browsers
83
84
    const port = 8080;
    var server = app.listen(port, function () {
     console.log(`Running at http://127.0.0.1:${port}/`);
86
87 });
Client (JavaScript)
 1
    Initializes the login interface.
    */
 3
 4
    document
     .getElementsByTagName("main")[0]
 5
      .replaceChildren(document.getElementById("log-in").content.cloneNode(true));
 6
     document.getElementById("login-dialog").showModal();
 7
    document.getElementById("login-button").addEventListener("click", login);
 8
 9
    let jwt = null; // JSON Web Token for authentication
10
11
    async function login() {
12
13
     const name = document.getElementById("administrator").value;
```

43

const name = req.query.name;

```
14
       const password = document.getElementById("password").value;
15
       const urlEncodedName = encodeURIComponent(name);
       const urlEncodedPassword = encodeURIComponent(password);
16
       const response = await fetch(
17
18
         "http://" +
19
           location.host 💺
           "/admin/login?name=" +
20
          urlEncodedName +
21
           "%password=" +
22
          urlEncodedPassword.
23
24
       );
       if (response.status ≠ 200) {
25
         console.error("Login failed: " + response.status);
26
27
         return;
28
       const responseJSON = await response.json();
29
       jwt = responseJSON.token;
30
31
       console.log("Received JWT: " + jwt);
32
       document.getElementById("login-dialog").close();
33
34
35
       document
36
         .getElementsByTagName("main")[0]
         .replaceChildren(document.getElementById("logged-in").content.cloneNode(true));
37
       document.getElementById("enter-question-dialog").showModal();
38
       document.getElementById("send-question").addEventListener("click", sendQuestion);
39
40
41
     async function sendQuestion() {
42
       const question = document.getElementById("question").value;
43
44
       const response = await fetch("http://" + location.host + "/admin/question", {
45
         method: "POST",
46
         headers: {
47
48
           "Content-Type": "text/plain",
          Authorization: `Bearer ${jwt}`,
49
         },
50
51
         body: question,
52
       });
53
       const text = await response.text();
54
       showAnswer(text);
55
56
57
    function showAnswer(text) {
       document.getElementById("answer-dialog").showModal(false);
58
       document.getElementById("answer-paragraph").textContent = text;
59
60 }
Alle Quellen:
admin.js
admin.html
admin.css
server.mjs
```

start_server.sh

Users.json

Referenzen

■ HTML DOM API