

Webprogrammierung mit JavaScript

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

Dozent: Prof. Dr. Michael Eichberg

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

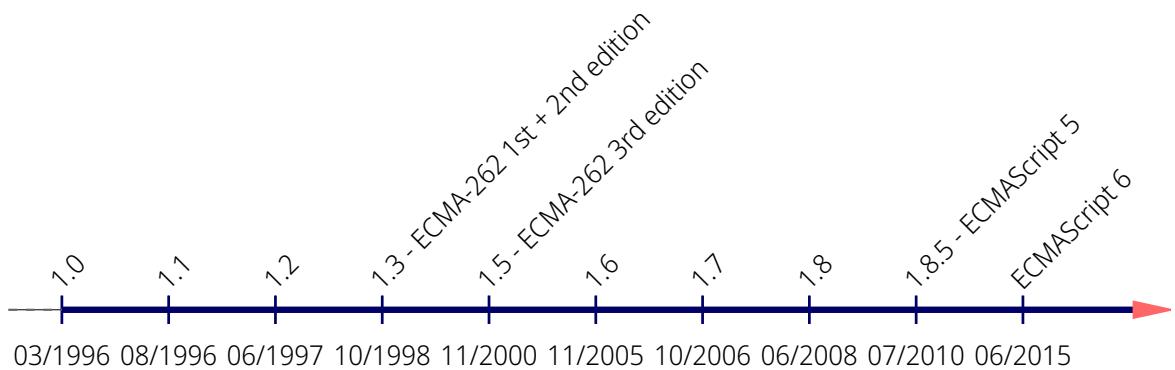
Version: 2.1.2

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



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

1. Grundlegende Sprachkonstrukte

Grundlagen

- Objektorientiert
 - Prototypische 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 Schlüsselwörter

Genutzte Schlüsselwörter

- `function, async, await, return, yield, void`
- `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`

◆ Bemerkung

Nicht genutzte Schlüsselwörter: `enum, implements, interface, package, private, protected, public`

Nicht mehr verwendet: `with`

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`)

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

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

Variables (var)

Achtung!

Neuer Code sollte var nicht mehr verwenden!

```

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

```

```
47
48 iLog("nach global_x() und local_var_x() - x, y, z:", x, y, z);
49
50
51 // Hier, ist nur die Variablen-deklaration (helloExpr) "hoisted", aber nicht
52 // die Definition. Daher kann die Funktion nicht vorher im Code aufgerufen
53 // werden!
54 try {
55   helloExpr();
56 } catch ({error, message}) {
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();
```

Datentypen und Operatoren

```
1 log("\nUndefined -----");
2 let u = undefined;
3 log("u", u);
4
5 log("\nNumber -----");
6 let i = 1; // double-precision 64-bit binary IEEE 754 value
7 let f = 1.0; // double-precision 64-bit binary IEEE 754 value
8 let l = 10_000;
9 let binary = 0b1010;
10 log("0b1010", binary);
11 let octal = 0o12;
12 log("0o12", octal);
13 let hex = 0xa;
14 log("0xA", hex);
15 log(
16     Number.MIN_VALUE,
17     Number.MIN_SAFE_INTEGER,
18     Number.MAX_SAFE_INTEGER,
19     Number.MAX_VALUE,
20 );
21 let x = NaN;
22 let y = Infinity;
23 let z = -Infinity;
24
25 // Standard Operatoren: +, - , *, /, %, ++, --, **
26 // Bitwise Operatoren: &, |, ^, ~, <<, >>, >>>
27 // werden immer auf dem Ganzzahlwert der Bits angewendet:
28 // Beispiel: 1234.5678e4 << 1 = 24691356
29 log("i =", i, "; i++ ", i++); // 1 oder 2?
30 log("i =", i, "; ++i ", ++i); // 2 oder 3?
31 log("2 ** 4 === 0 ", 2 ** 4);
32 log("7 % 3 === ", 7 % 3);
33 log("1 / 0 === ", 1 / 0);
34
35 log("\nBigInt -----");
36 let ib = 1n;
37 log(100n === BigInt(100));
38 log(Number.MAX_SAFE_INTEGER + 2102); // 9007199254743092
39 log(BigInt(Number.MAX_SAFE_INTEGER) + 2102n); // 9007199254743093n
40
41 log("\nBoolean -----");
42 let b = true; // oder false
43 log("Boolean(undefined)", Boolean(undefined)); // true oder false?
44 log(null == true ? "true" : "false"); // true oder false?
45
46 log("\n(Quasi-)Logische Operatoren -----");
47 log('1 && "1": ', 1 && "1");
48 log('null && "1": ', null && "1");
49 log("null && true: ", null && true);
50 log("true && null: ", true && null);
51 log("null && false: ", null && false);
```

```

52 log("{} && true: ", {} && true);
53
54 // Neben den Standardoperatoren: ``&&``, ``||`` , ``!`` gibt es auch noch ``??``
55 // Der ``??``-Operator gibt den rechten Operanden zurück, wenn der linke Operand
56 // ``null`` oder ``undefined`` ist. Andernfalls gibt er den linken Operanden
57 // zurück.
58 // ``??`` ist der *nullish coalescing operator (??)*
59 // Falls der linke Wert null oder undefined ist, dann ist er vergleichbar zu ||
60 log('1 ?? "1": ', 1 ?? "1");
61 log('null ?? "1": ', null ?? "1");
62 log("null ?? true: ", null ?? true);
63 log("true ?? null: ", true ?? null);
64 log("null ?? false: ", null ?? false);
65 log("{} ?? true: ", {} ?? true);
66
67 log('undefined ?? "1": ', undefined ?? "1");
68 log("undefined ?? true: ", undefined ?? true);
69 log("true ?? undefined: ", true ?? undefined);
70 log("undefined ?? false: ", undefined ?? false);
71 log("undefined ?? undefined: ", undefined ?? undefined);
72
73 log("\nStrings -----");
74 let _s = "42";
75 log("Die Antwort ist " + _s + "."); // String concatenation
76 log(`Die Antwort ist ${_s}.`); // Template literals (Template strings)
77 // multiline Strings
78 log(`
79     Die Antwort mag ${_s} sein,
80     aber was ist die Frage?`);
81
82 log(String(42)); // "42"
83
84 // ACHTUNG Objekte und Errors am Besten direkt an log übergeben,
85 // damit die Objekteigenschaften ausgegeben werden.
86 log("State: " + { a: "abc" }, { a: "abc" });
87
88 log("\nObjekte -----");
89 let emptyObject = null;
90 let anonymousObj = {
91     i: 1,
92     u: { j: 2, v: { k: 3 } },
93     toString: function () {
94         return "anonymousObj";
95     },
96     "?": "question mark",
97 };
98 // Zugriff auf die Eigenschaften eines Objekts
99 anonymousObj.j = 2; // mittels Bezeichner ("j") (eng. Identifier)
100 anonymousObj["j"] = 4; // mittels String ("j")
101 anonymousObj["k"] = 3;
102 log("anonymousObj:           ", anonymousObj);
103 log("anonymousObj.toString(): ", anonymousObj.toString());
104

```

```

105 // delete dient dem Löschen von Eigenschaften:
106 delete anonymousObj["?"];
107 delete anonymousObj.toString();
108 log("anonymousObj.toString() [original]", anonymousObj.toString());
109
110 // Der Chain-Operator "?.:" kann verwendet werden, um auf Eigenschaften
111 // (Properties) von Objekten zuzugreifen, ohne dass eine Fehlermeldung
112 // ausgegeben wird, wenn eine (höher-liegende) Eigenschaft nicht definiert ist.
113 // Besonders nützlich beim Verarbeiten von komplexen JSON-Daten.
114 log("anonymousObj.u?.v.k", anonymousObj.u?.v.k);
115 log("anonymousObj.u.v?.k", anonymousObj.u.v?.k);
116 log("anonymousObj.u.v?.z", anonymousObj.u.v?.z);
117 log("anonymousObj.u.q?.k", anonymousObj.u.q?.k);
118 log("anonymousObj.p?.v.k", anonymousObj.p?.v.k);
119
120 // Nützliche Zuweisungen, um den Fall undefined und null gemeinsam zu behandeln:
121 anonymousObj.name ||= "Max Mustermann";
122
123 log("\nDate -----");
124 let date = new Date("8.6.2024"); // ACHTUNG: Locale-Settings
125 log(date);
126
127 log("\nFunktionen sind auch Objekte -----");
128 let func = function () {
129     return "Hello World";
130 };
131 log(func, func());
132
133 log("\nArrays -----");
134 let temp = undefined;
135 let $a = [1];
136 log("let $a = [1]; $a, $a.length", $a, $a.length);
137 $a.push(2); // append
138 log("$a.push(2); $a", $a);
139 temp = $a.unshift(0); // "prepend" → return new length
140 log("temp = $a.unshift(0); temp, $a", temp, $a);
141 temp = $a.shift(); // remove first element → return removed element
142 log("temp = $a.shift(); temp, $a", temp, $a);
143 // Um zu prüfen ob eine Datenstruktur ein Array ist:
144 log("Array.isArray($a)", Array.isArray($a));
145 log("Array.isArray({})", Array.isArray({}));
146 log("Array.isArray(1)", Array.isArray(1));
147
148 log("\nSymbols -----");
149 let sym1 = Symbol("1"); // a unique and immutable primitive value
150 let sym2 = Symbol("1");
151 let obj1Values = { sym1: "value1", sym2: "value2" };
152 log(obj1Values);
153 log(`sym1 in ${JSON.stringify(obj1Values)}: `, sym1 in obj1Values);
154 let obj2Values = { [sym1]: "value1", [sym2]: "value2" };
155 log(obj2Values);
156 log(`sym1 in ${JSON.stringify(obj2Values)}: `, sym1 in obj2Values);
157 log(obj1Values, " vs. ", obj2Values);

```

```
158 | log({ sym1: "this", sym1: "that" }); // ??? { sym1: "that" }
159 | log("sym1 = sym2", sym1 == sym2);
```

Funktionsdefinitionen

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

```
52
53 log("\nGenerator Functions -----");
54 /* Generator Functions */
55 function* fib() {
56     // generator
57     let a = 0,
58         b = 1;
59     while (true) {
60         yield a;
61         [a, b] = [b, a + b];
62     }
63 }
64 const fibGen = fib();
65 log(fibGen.next().value); // 0
66 log(fibGen.next().value); // 1
67 log(fibGen.next().value); // 1
68 log(fibGen.next().value); // 2
69 /* Will cause an infinite loop:
70    for (const i of fib()) console.log(i);
71    // 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 Snippet (in der Konsole des Browsers), das programmatisch zum Ende des Dokuments scrollt.

※ Hinweis

- `document.body` referenziert das HTML Body Element.
- 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.

(Sei der String: "100px", dann liefert `parseInt`, den Wert 100).

Vergleich von Werten und implizite Typumwandlung

```
1 // Gleichheit          = // mit Typumwandlung (auch bei <, >, ≤, ≥)
2
3 // strikt gleich      === // ohne Typumwandlung
4 // strikte Ungleichheit !== // ohne Typumwandlung
5
6 log('1 == "1": ', 1 == "1");
7 log('1 === "1": ', 1 === "1");
8 log("1.0 == 1: ", 1 == 1.0);
9 log("1.0 === 1: ", 1 === 1.0);
10 log("1 === 1n: ", 1 === 1n); // 1n ist ein bigint mit den Wert 1
11 log("1 == 1n: ", 1 == 1n);
12 log('1 < "1"', 1 < "1");
13 log('0 < "1"', 0 < "1");
14 log('0 ≤ "0"', 0 ≤ "0");
15 log('"abc" ≤ "d"', "abc" ≤ "d");
16
17 log('"asdf" === "as" + "df"', "asdf" === "as" + "df")); // unlike Java!
18
19 log("NaN === NaN: ", NaN === NaN);
20 log("NaN == NaN: ", NaN == NaN);
21 log("null === NaN: ", null === NaN);
22 log("null == NaN: ", null == NaN);
23 log("null === null: ", null === null);
24 log("null == null: ", null == null);
25 log("undefined === undefined: ", undefined === undefined);
26 log("undefined == undefined: ", undefined == undefined);
27 log("null === undefined: ", null === undefined);
28 log("null == undefined: ", (null == undefined) + "!");
29
30 const a1 = [1, 2, 3];
31 const a2 = [1, 2, 3];
32 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 log("const a1 = [1, 2, 3]; const a2 = [1, 2, 3]; a1 === a2: ", a1 === a2);
36 log("const a1 = [1, 2, 3]; const a2 = [1, 2, 3]; a1 == a2: ", a1 == a2);
37 log(
38   "flatEquals(a1,a2):",
39   a1.length == a2.length && a1.every((v, i) => v === a2[i]),
40 );
41
42 let firstJohn = { person: "John" };
43 show('let firstJohn = { person: "John" };');
44 let secondJohn = { person: "John" };
45 show('let secondJohn = { person: "John" };');
46 let basedOnFirstJohn = Object.create(firstJohn);
47 show("let basedOnFirstJohn = Object.create(firstJohn)");
48 log("firstJohn == firstJohn: ", firstJohn == firstJohn);
49 log("firstJohn === secondJohn: ", firstJohn === secondJohn);
50 log("firstJohn == secondJohn: ", firstJohn == secondJohn);
51 log("firstJohn === basedOnFirstJohn: ", firstJohn === basedOnFirstJohn);
```

```

52 log("firstJohn == basedOnFirstJohn: ", firstJohn == basedOnFirstJohn);
53
54 {
55   const obj = {
56     name: "John",
57     age: 30,
58     city: "Berlin",
59   };
60   show("Typtests und Feststellung des Typs:");
61   log("typeof obj", typeof obj);
62   log("obj instanceof Object", obj instanceof Object);
63   log("obj instanceof Array", obj instanceof Array);
64
65   // ! Das Array Objekt ist Kontextabhängig:
66   //
67   //     iframe.contentWindow.Array === window.Array;
68   //
69   // D.h. ein instanceof Check liefert ggf. unerwartete Resultate
70 }
71 {
72   const obj = { a: "lkj" };
73   const obj2 = Object.create(obj);
74   log("obj2 instanceof obj.constructor", obj2 instanceof obj.constructor);
75 }
76
77 log("\n?-Operator/if condition and Truthy and Falsy Values:");
78 log('""', "" ? "is truthy" : "is falsy");
79 log("f()", () => {}) ? "is truthy" : "is falsy";
80 log("Array ", Array ? "is truthy" : "is falsy");
81 log("obj ", {} ? "is truthy" : "is falsy");
82 log("undefined ", undefined ? "is truthy" : "is falsy");
83 log("null ", null ? "is truthy" : "is falsy");
84 log("0", 0 ? "is truthy" : "is falsy");
85 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 `isNaN(<Value>)` verwendet werden.

Bedingungen und Schleifen

```
1 const arr = [1, 3, 4, 7, 11, 18, 29];
2
3 log("if-else_if-else:");
4 if (arr.length == 7) {
5     ilog("arr.length == 7");
6 } else if (arr.length < 7) {
7     ilog("arr.length < 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");
16         break;
17     case 6:
18         ilog("arr.length = 6");
19         break;
20     default:
21         ilog("arr.length ≠ 6 and ≠ 7");
22}
23
24 log("\nswitch (string value):");
25 switch ("foo") {
26     case "bar":
27         ilog("it's bar");
28         break;
29     case "foo":
30         ilog("it's foo");
31         break;
32     default:
33         ilog("not foo, not bar");
34}
35
36 log("\nswitch (integer - no type conversion):");
37 switch (
38     1 // Vergleich auf strikte Gleichheit (≡)
39 ) {
40     case "1":
41         ilog("string(1)");
42         break;
43     case 1:
44         ilog("number(1)");
45         break;
46}
47
48 ilog("\nfor-continue:");
49 for (let i = 0; i < arr.length; i++) {
50     const v = arr[i];
51     if (v % 2 == 0) continue;
```

```

52     log(v);
53 }
54
55 ilog("\n(for)-break with label:");
56 outer: for (let i = 0; i < arr.length; i++) {
57     for (let j = 0; j < i; j++) {
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):");
64 for (const key in arr) {
65     log(key, arr[key]);
66 }
67
68 ilog("\nof (values of Arrays):");
69 for (const value of arr) {
70     log(value);
71 }
72
73 ilog("\nArray and Objects - instanceof:");
74 log("arr instanceof Object", arr instanceof Object);
75 log("arr instanceof Array", arr instanceof Array);
76
77 const obj = {
78     name: "John",
79     age: 30,
80     city: "Berlin",
81 };
82
83 ilog("\nin (properties of Objects):");
84 for (const key in obj) {
85     log(key, obj[key]);
86 }
87
88 /* TypeError: obj is not iterable
89
90     for (const value of obj) {
91         log(value);
92     }
93 */
94
95 {
96     ilog("\nIteration über Iterables (here: Map):");
97     const m = new Map();
98     m.set("name", "Elisabeth");
99     m.set("alter", 50);
100    log("Properties of m: ");
101    for (const key in m) {
102        log(key, m[key]);
103    }
104    log("Key-Values of m: ");
105    for (const [key, value] of m) {

```

```
106     log(key, value);
107 }
108 {
111   ilog("\nWhile Loop: ");
112   let c = 0;
113   while (c < arr.length) {
114     const v = arr[c];
115     if (v > 10) break;
116     log(v);
117     c++;
118   }
119 }
120 {
122   ilog("\nDo-While Loop: ");
123   let c = 0;
124   do {
125     log(arr[c]);
126     c++;
127   } while (c < arr.length);
128 }
```

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

Fehlerbehandlung

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

※ Hinweis

Einem Array können Sie einen neuen Wert mittels `push` hinzufügen: `const a = [].push(1);`

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.

Nutzen Sie node.js als Test-/Ausführungsumgebung.

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

Destrukturierung (destructuring)

```
1 log("Array Destructuring:");
2
3 let [val1, val2] = [1, 2, 3, 4];
4 ilog("[val1, val2] = [1, 2, 3, 4]:", "val1:", val1, ", val2:", val2); // 1
5
6 log("Object Destructuring:");
7
8 let { a, b } = { a: "aaa", b: "bbb" };
9 ilog('let { a, b } = { a: "aaa", b: "bbb" }: ', "a:", a, ", b:", b); // 1
10
11 {
12     let { a: x, b: y } = { a: "aaa", b: "bbb" };
13     ilog('let { a: x, b: y } = { a: "aaa", b: "bbb" }: ', "x:", x, ", y:", y); // 1
14 }
15 {
16     let { a: x, c: y } = { a: "aaa", b: "bbb" };
17     ilog('let { a: x, c: y } = { a: "aaa", b: "bbb" }: ', "x:", x, ", y:", y); // 1
18 }
19
20 let { a: u, b: v, ...w } = { a: "+", b: "-", c: "*", d: "/" };
21 ilog(
22     'let { a: u, b: v, ...w } = { a: "+", b: "-", c: "*", d: "/" }:',
23     "u:",
24     u,
25     ", v:",
26     v,
27     ", w:",
28     JSON.stringify(w), // just for better readability/comprehension
29 );
30
31 let { k1, k2 } = { a: "a", b: "b" };
32 ilog('let { k1, k2 } = { a: "a", b: "b" }:', "k1:", k1, ", k2:", k2);
33 // "undefined undefined", weder k1 noch k2 sind definiert
```

JSON (JavaScript Object Notation)

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

JSON verlangt, dass Schlüssel Strings sein müssen und Strings in doppelten Anführungszeichen eingeschlossen werden müssen.

Reguläre Ausdrücke

- Eingebaute Unterstützung basierend auf entsprechenden Literalen (Strings in "/") und einer API
- inspiriert von der Perl Syntax
- Methoden auf regulären RegExps: `test` (e.g., `<RegExp>.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"));
4    console.log(p.test("ad13"));
5    console.log(p.test("13H"));
6  }
7  {
8    const p = /[1-9]+H/g;
9    const s = "1H, 2H, 3P, 4C";
10   console.log(s.match(p));
11   console.log(s.replace(p, "XX"));
12 }
```

Klassen und Vererbung

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

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, da erfolgt die Bindung zum Zeitpunkt der Definition und es wird das `this` aus dem umgebenden Context geerbt.)

```
1 //"use strict";  
2  
3 function counter() {  
4     // console.log(this === globalThis); // true  
5     if (this.count)  
6         // this is the global object if we don't use strict mode  
7         this.count++;  
8     else {  
9         this.count = 1;  
10    }  
11  
12    return this.count;  
13}  
14  
15 const counterExpr = function () {  
16     if (this.count) this.count++;  
17     else {  
18         this.count = 1;  
19     }  
20  
21    return this.count;  
22};  
23  
24 const counterArrow = () => {  
25     console.log(this);  
26     console.log(this === globalThis); // depends on the execution context:  
27                                         // Browser (non-strict): true  
28                                         // Node.js: false  
29     this.count = this.count ? this.count + 1 : 1;  
30     return this.count;  
31};  
32  
33 console.log("\nCounter");  
34 console.log(counter()); // 1  
35 console.log(counter()); // 2  
36 console.log(`globalThis.count (${globalThis.count})`);  
37  
38 console.log("\nCounterExpression");  
39 console.log(counterExpr()); // 3  
40 console.log(counterExpr()); // 4  
41  
42 console.log("\nCounter");  
43 const obj = {};  
44 console.log(counter.apply(obj)); // 1 - we pass in obj as "this"!
```

```
45 console.log(counterExpr.apply(obj)); // 2
46
47 console.log(`\nCounterArrow (${this.count})`);
48 console.log(counterArrow.apply(obj)); // 1
49 console.log(counterArrow.apply(undefined)); // 2
50 console.log(counterArrow.apply()); // 3
51 console.log(counterArrow.apply(obj)); // 4
52 console.log(counterArrow.apply({})); // 5
53
54 console.log("\nCounter (global)");
55 console.log(counter());
56 console.log(counterExpr());
```

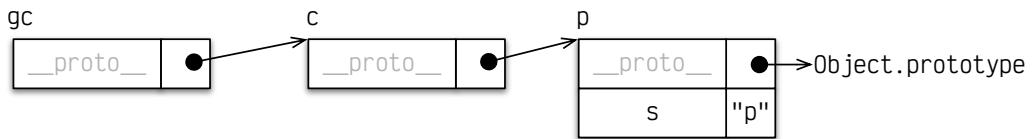
Partial Function Application

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

Prototype basierte Vererbung

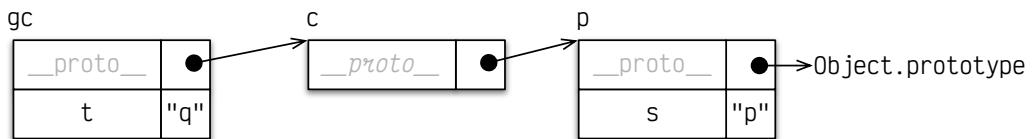
Verwendung von `Object.create` zur Initialisierung der *Prototype Chain*:

```
1 const p = { s : "p" };
2 const c = Object.create(p);
3 const gc = Object.create(c);
```



Verwendung der Eigenschaften von Prototypen:

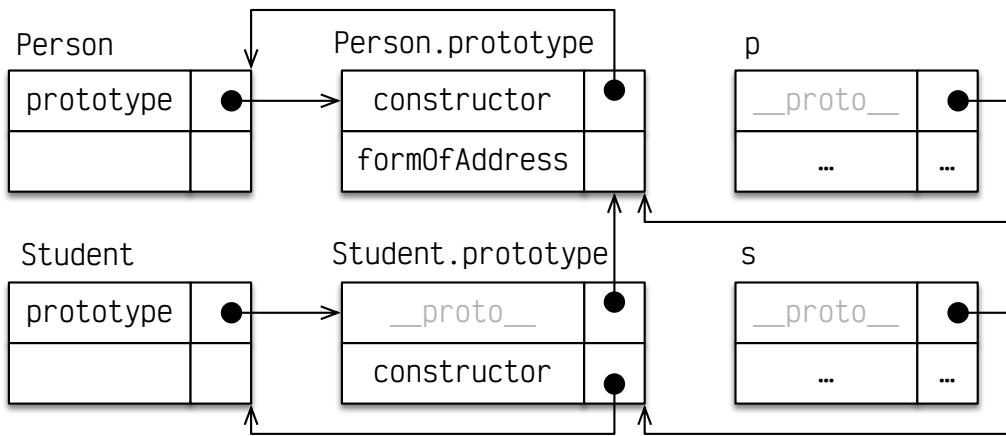
```
1 const p = { s : "p" };
2 const c = Object.create(p);
3 const gc = Object.create(c);
4 gc.t = "q";
```



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

Pseudoclassical Inheritance

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



Objektabhängigkeiten

```

1 function Person(name, title){ ... }
2 Person.prototype.formOfAddress = function (){ ... }
3
4 function Student(name, title, id, email) { ... }
5 Student.prototype = Object.create(Person.prototype);
6 Student.prototype.constructor = Student;
7
8 const p = new Person(...);
9 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
2 console.log("{}.__proto__: ",{}.__proto__);
3 console.log("Array.prototype: ",Array.prototype);
4 console.log("Array.prototype.__proto__: ",Array.prototype.__proto__);
5 console.log("Object.prototype: ",Object.prototype);
6 console.log("Object.__proto__: ",Object.__proto__);
7
8 let o = { created: "long ago" };
9 var p = Object.create(o);
10 console.log("Object.getPrototypeOf(o): " + Object.getPrototypeOf(o));
11 console.log("o.isPrototypeOf(p):" + o.isPrototypeOf(p));
12 console.log("Object.prototype.isPrototypeOf(p): " + Object.prototype.isPrototypeOf(p));

```

Praktische Verwendung von Prototypen basierter Vererbung

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

Grundlagen von ECMAScript Modulen

Queue.mjs exportiert die Klasse Queue

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

log.mjs verwendet die Klasse Queue ([import](#)) und exportiert Funktionen ([export](#)) zum Loggen

```
1 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 |   } else {
9 |     message.unshift("\n");
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";`).

DOM Manipulation

```
1 <html lang="en">
2   <head>
3     <meta charset="utf-8" />
4     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
5     <title>DOM Manipulation with JavaScript</title>
6     <script>
7       function makeScriptsEditable() {
8         const scripts = document.getElementsByTagName("script");
9         for (const scriptElement of scripts) {
10           scriptElement.contentEditable = true;
11           const style = scriptElement.style;
12           style.display = "block";
13           style.fontFamily = "menlo, monospaced";
14           style.whiteSpace = "preserve";
15           style.padding = "0.25em";
16           style.backgroundColor = "lightyellow";
17         }
18       }
19     </script>
20   </head>
21   <body>
22     <h1>DOM Manipulation with JavaScript</h1>
23     <p id="demo">This is a paragraph.</p>
24     <button
25       type="button"
26       onclick="
27         document.getElementById('demo').style.color = 'red';
28         makeScriptsEditable();
29         document.querySelector('button').style.display = 'none' ;
30       >
31       Magic!
32     </button>
33
34     <script>
35       const e = document.getElementById("demo");
36       const style = e.style;
37       e.addEventListener("mouseover", () => (style.color = "green"));
38       e.addEventListener("mouseout", () => (style.color = "unset"));
39     </script>
40
41     <p>Position der Mouse: <span id="position"></span></p>
42     <script>
43       window.addEventListener("mousemove", () => {
44         document.getElementById("position").innerHTML =
45           `(${event.clientX}, ${event.clientY})`;
46       });
47     </script>
48   </body>
49 </html>
```

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
5 // Cross-Origin Resource Sharing (CORS); This is required to allow the browser
6 // using a different domain to load the HTML to make requests to this server.
7 // I. e., we can use the HTML file from the "web-javascript" project to make
8 // requests to this server.
9 import cors from "cors";
10 const APP_PORT = 5080;
11
12 const app = express();
13
14 app.get("/users", cors(), function (req, res) {
15   res.set("Content-Type", "application/json");
16   res.end(`{
17     "user1" : {
18       "name" : "dingo",
19       "password" : "1234",
20       "profession" : "chef",
21       "id": 1
22     },
23
24     "user2" : {
25       "name" : "ringo",
26       "password" : "asdf",
27       "profession" : "boss",
28       "id": 3
29     }
30   `);
31 });
32
33 app.listen(APP_PORT, function () {
34   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" />
4     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
5     <title>Eventhandling</title>
6   </head>
7   <body>
8     <script>
9       /* Using Promises:
10        function getUsers() {
11          fetch('http://127.0.0.1:4080/users')
12            .then(response => response.json())
13            .then(users => {
14              const usersElement = document.getElementById('users');
15              usersElement.innerText = JSON.stringify(users);
16            });
17        }
18      */
19
20      /* Using async/await: */
21      async function getUsers() {
22        let response = await fetch("http://127.0.0.1:5080/users");
23        let users = await response.json();
24        const usersElement = document.getElementById("users");
25        usersElement.innerText = JSON.stringify(users);
26      }
27    </script>
28
29    <div id="users"></div>
30    <button onclick="getUsers()">Get Users</button>
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');
2 const app = express();
3
4 const expressWs = require('express-ws')(app);
5
6 let clients = 0;
7 let playerWSs = [];
8
9 let adminWS = null;
10 let answersCount = 0;
11 let correctAnswersCount = 0;
12
13 app.use(express.static('.')); // required to serve static files
14
15
16 function sendCurrentPlayers() {
17     if (adminWS && playerWSs.length > 0) {
18         allPlayers = playerWSs
19             .filter(player => player.name)
20             .map(player => { return { "id": player.id, "name": player.name } })
21         console.log("Sending current players: " + JSON.stringify(allPlayers));
22         const m = JSON.stringify({ "type": "players", "players": allPlayers });
23         adminWS.send(m);
24     }
25 }
26
27 function sendNextQuestion() {
28     answersCount = 0;
29     correctAnswersCount = 0;
30     const question = "What is the capital of France?";
31     const answers = ["Paris", "London", "Berlin", "Madrid"];
32     const correct = "Paris";
33
34     const nextQuestion = JSON.stringify({
35         "type": "question",
36         "question": question,
37         "answers": ["Paris", "London", "Berlin", "Madrid"]
38     })
39     playerWSs.forEach(player => player.ws.send(nextQuestion));
40     adminWS.send(JSON.stringify({
41         "type": "question",
42         "question": question,
43         "answers": answers,
44         "correct": correct
45     }));
46 }
47
48 function sendResults() {
```

```

49 const results = playerWSs.map(player => {
50   return { "id": player.id, "name": player.name, "wins": player.wins }
51 });
52 const sortedResults = results.sort((a, b) => b.wins - a.wins);
53 const resultsMsg = JSON.stringify({
54   "type": "results",
55   "results": sortedResults
56 });
57 playerWSs.forEach(player => player.ws.send(resultsMsg));
58 adminWS.send(resultsMsg);
59
60 }
61
62
63 function handleAnswer(clientId, answer) {
64   const correct = answer.answer === "Paris";
65   const player = playerWSs.find(player => player.id === clientId);
66   if (correct) {
67     if (correctAnswersCount === 0) {
68       player.wins++;
69     }
70     correctAnswersCount++;
71   }
72   answersCount++;
73   if (answersCount === playerWSs.length) {
74     // sendNextQuestion();
75     sendResults();
76   } else {
77     adminWS.send(JSON.stringify({
78       "type": "answers",
79       "count": answersCount,
80       "correctAnswersCount": correctAnswersCount
81     }));
82   }
83 }
84
85
86 app.ws('/player', function (ws, request) {
87   const clientId = clients++;
88   const playerData = { "ws": ws, "id": clientId, "wins": 0 }
89   playerWSs.push(playerData);
90   ws.onmessage = function (event) {
91     message = JSON.parse(event.data);
92     switch (message.type) {
93       case "registration":
94         const name = message.name;
95         console.log("Registration: " + clientId + "/" + name);
96         playerData.name = name;
97         sendCurrentPlayers();
98         break;
99
100      case "answer":
101        const answer = message;
102        handleAnswer(clientId, answer);

```

```
103         break;
104
105     default:
106         console.log("Unknown message: " + message);
107         break;
108     }
109 }
110 ws.onclose = function () {
111     console.log("Player disconnected: " + clientId);
112     playerWSs = playerWSs.filter(player => player.id !== clientId);
113     sendCurrentPlayers();
114 };
115 ws.onerror = function () {
116     console.log("Player error: " + clientId);
117     playerWSs = playerWSs.filter(player => player.id !== clientId);
118     sendCurrentPlayers();
119 };
120 });
121
122 app.ws('/admin', function (ws, req) {
123     adminWS = ws;
124     sendCurrentPlayers(); // on admin registration send current players
125     ws.onmessage = function (event) {
126         message = JSON.parse(event.data);
127         switch (message.type) {
128             case "start":
129                 console.log("Start game");
130                 sendNextQuestion();
131                 break;
132             default:
133                 console.log("Unknown message: " + message);
134                 break;
135         }
136     };
137
138     ws.onclose = (event) => {
139         console.log("Admin disconnected");
140         adminWS = null;
141         sendCurrentPlayers();
142     };
143
144     ws.onerror = (event) => {
145         console.log("Admin error: " + event);
146         sendCurrentPlayers();
147     };
148
149 });
150
151
152 const PORT = process.env.QUIZZY_PORT || 8800;
153
154 var server = app.listen(PORT, function () {
155     console.log(`Quizzy running at http://127.0.0.1:${PORT}/`);
156 })
```

Client - Players

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <script>
6          let ws = undefined
7          try {
8              const protocol = location.protocol === "https:" ? "wss:" : "ws:";
9              ws = new WebSocket(` ${protocol}//${location.host}/player`);
10         } catch (e) {
11             alert("WebSocket connection failed: " + e);
12             const m = "WebSocket connection failed. Please check the server.";
13             document.getElementById("main").innerText = m;
14         }
15         ws.onmessage = (event) => {
16             const data = JSON.parse(event.data);
17             switch (data.type) {
18                 case "question":
19                     console.log("Question: " + data.question);
20                     showQuestion(data);
21                     break;
22                 case "results":
23                     const main = document.getElementById("main")
24                     main.innerText = "Results: " + event.data;
25                     break;
26                 default:
27                     console.log("Unknown message: " + data);
28                     break;
29             }
30         };
31         ws.onclose = (event) => {
32             console.log("Connection closed: " + event);
33         }
34         ws.onerror = (event) => {
35             console.error("Error: " + event);
36         }
37
38         function showQuestion(data) {
39             const main = document.getElementById("main")
40             main.innerHTML = `<h1>Question</h1><p>${data.question}</p>`;
41
42             function createAnswerButton(answer) {
43                 const button = document.createElement("button");
44                 button.innerText = answer;
45                 button.onclick = submitAnswer(answer);
46                 return button;
47             }
48
49             for (answer of data.answers) {
50                 main.appendChild(createAnswerButton(answer));
51             }
52         }
53     </script>
54 </head>
55 <body>
56     <div id="main"></div>
57 </body>
58 </html>
```

```

53
54     function submitAnswer(answer) {
55         return () => {
56             ws.send(JSON.stringify({
57                 "type": "answer",
58                 "answer": answer
59             }));
60             doWait();
61         }
62     }
63
64     function submitUsername() {
65         const name = document.getElementById("username").value;
66         ws.send(JSON.stringify({
67             "type": "registration",
68             "name": name
69         }));
70
71         doWait();
72     }
73
74     function doWait() {
75         const main = document.getElementById("main");
76         main.innerHTML = "Waiting for other players...";
77     }
78 </script>
79
80 <body>
81
82     <main id="main">
83         <form>
84             <input type="text" id="username" placeholder="Username">
85             <button type="button" onclick="submitUsername();">Submit</button>
86         </form>
87     </main>
88 </body>
89
90 </html>

```

Client - Admin

```

1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <script>
6          const protocol = location.protocol === "https:" ? "wss:" : "ws:";
7          const ws = new WebSocket(` ${protocol}//${location.host}/admin`);
8
9          ws.onmessage = (event) => {
10              const data = JSON.parse(event.data);
11              console.log("Received: " + event.data);
12              switch (data.type) {
13                  case "players":

```

```

14  const players = document.getElementById("players")
15  players.innerText =
16      "[" + data.players.length + " players] " +
17      data.players
18      .map(player => player.id + ": " + player.name)
19      .join(", ");
20      break;
21  case "question":
22      showQuestion(data);
23      break;
24  case "results":
25      const main = document.getElementById("main")
26      main.innerText = "Result: " + event.data;
27      break;
28  default:
29      console.log("unknown: " + event.data);
30      break;
31  }
32};

34 ws.onclose = (event) => {
35     console.log("Connection closed: " + event);
36     const main = document.getElementById("main")
37     main.innerText = "Connection closed - you need to restart.";
38 };
39 ws.onerror = (event) => {
40     console.log("Connection error: " + event);
41 };

43 function startGame() {
44     ws.send(JSON.stringify({ "type": "start" }));
45 }

47 function showQuestion(data) {
48     document.getElementById("main").innerText = `
49         question: ${data.question}; correct answer: ${data.correct}
50     `;
51 }
52 </script>
53 </head>
54
55 <body>
56     <main id="main">
57         <h1>Players</h1>
58         <p id="players"></p>
59         <button type="button" onclick="startGame();">Start Game</button>
60     </main>
61 </body>
62
63 </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";
2 import fs from "fs";
3 import path from "node:path";
4 import { fileURLToPath } from "url";
5 import jwt from "jsonwebtoken";
6 import crypto from "crypto";
7 import bodyParser from "body-parser";
8
9 const app = express();
10
11 const SERVER_SECRET = crypto.randomBytes(64).toString("hex");
12 const users = JSON.parse(
13   fs.readFileSync(
14     path.resolve(path.dirname(fileURLToPath(import.meta.url)), "users.json"),
15     "utf8",
16   ),
17 );
18 console.log("Users: " + JSON.stringify(users));
19
20 app.use(express.static("."));
21 app.use(express.json());
22 app.use(bodyParser.text());
23
24 const verifyToken = (req, res, next) => {
25   console.log("Headers: " + JSON.stringify(req.headers));
26
27   const token = req.headers["authorization"].split(" ")[1];
28   if (!token) {
29     return res.status(401).json({ error: "Unauthorized" });
30   }
31
32   jwt.verify(token, SERVER_SECRET, (err, decoded) => {
```

```

33  console.log("Decoded: " + JSON.stringify(decoded));
34  if (err) {
35    return res.status(401).json({ error: "Unauthorized" });
36  }
37  req.userIndex = decoded.userIndex;
38  next();
39 });
40 };
41
42 app.get("/admin/login", function (req, res) {
43   const name = req.query.name;
44   const password = req.query.password; // in a real app use hashed passwords!
45
46   if (!name || !password) {
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++) {
53     if (users[i].name === name && users[i].password === password) {
54       userIndex = i;
55       break;
56     }
57   }
58   if (userIndex === -1) {
59     res.status(401).send("Credentials invalid.");
60     return;
61   }
62   console.log(
63     "Authenticated: " + users[userIndex].name + " " + users[userIndex].password,
64   );
65
66 // Here, we can use the userIndex to identify the user;
67 // but this only works as long as the user list is fixed.
68 // In a real app use, e.g., a user's email.
69 const token = jwt.sign({ userIndex: userIndex }, SERVER_SECRET, {
70   expiresIn: "2h",
71 });
72 res.status(200).json({ token });
73 });
74
75 app.post("/admin/question", verifyToken, function (req, res) {
76   const userIndex = req.userIndex;
77   const question = req.body;
78   console.log("Received question: " + question + " from user: " + users[userIndex].name);
79
80   res.status(200).send("Question stored. Preliminary answer: 42.");
81 });
82
83 // Attention: a port like 6666 will not work on (most?) browsers
84 const port = 8080;
85 var server = app.listen(port, function () {
86   console.log(`Running at http://127.0.0.1:${port}/`);

```

87 |});

Client (JavaScript)

```
1  /*
2   * Initializes the login interface.
3   */
4  document
5    .getElementsByTagName("main")[0]
6    .replaceChildren(document.getElementById("log-in").content.cloneNode(true));
7  document.getElementById("login-dialog").showModal();
8  document.getElementById("login-button").addEventListener("click", login);
9
10 let jwt = null; // JSON Web Token for authentication
11
12 async function login() {
13   const name = document.getElementById("administrator").value;
14   const password = document.getElementById("password").value;
15   const urlEncodedName = encodeURIComponent(name);
16   const urlEncodedPassword = encodeURIComponent(password);
17   const response = await fetch(
18     "http://" +
19       location.host +
20       "/admin/login?name=" +
21       urlEncodedName +
22       "&password=" +
23       urlEncodedPassword,
24   );
25   if (response.status !== 200) {
26     console.error("Login failed: " + response.status);
27     return;
28   }
29   const responseJSON = await response.json();
30   jwt = responseJSON.token;
31   console.log("Received JWT: " + jwt);
32
33   document.getElementById("login-dialog").close();
34
35   document
36    .getElementsByTagName("main")[0]
37    .replaceChildren(document.getElementById("logged-in").content.cloneNode(true));
38  document.getElementById("enter-question-dialog").showModal();
39  document.getElementById("send-question").addEventListener("click", sendQuestion);
40}
41
42 async function sendQuestion() {
43   const question = document.getElementById("question").value;
44
45   const response = await fetch("http://" + location.host + "/admin/question", {
46     method: "POST",
47     headers: {
48       "Content-Type": "text/plain",
49       Authorization: `Bearer ${jwt}`,
50     },
51   });
52 }
```

```
51     body: question,  
52   });
53   const text = await response.text();
54   showAnswer(text);
55 }
56
57 function showAnswer(text) {
58   document.getElementById("answer-dialog").showModal(false);
59   document.getElementById("answer-paragraph").textContent = text;
60 }
```

Alle Quellen:

[admin.js](#)

[admin.html](#)

[admin.css](#)

[server.mjs](#)

[start_server.sh](#)

[Users.json](#)

Referenzen

- [HTML DOM API](#)

2. Web Komponenten

Work in progress!

Quizzy

Bei der Quizzy Komponenten handelt es sich um eine (ganz) einfache Client-Server basierte Komponente für Quizzes. Die Komponente besteht aus der Definition der Web Komponente und einem Server. Die Komponente und der Server kommunizieren über Web Sockets. Der Server führt in Hinblick auf Cross-origin requests keine besonderen Prüfungen durch!

⚠ Warnung

Es handelt sich nur um einen minimalen Prototyp, der lediglich der Demonstration von Webkomponenten und der Kommunikation selbiger mit Servern dient. Es existiert keinerlei Sicherheit!

Quizzy - Server Code (server.js)

```
1 const express = require('express');
2 const app = express();
3
4 const expressWs = require('express-ws')(app);
5
6 let clients = 0;
7 let playerWSs = [];
8
9 let adminWS = null;
10 let answersCount = 0;
11 let correctAnswersCount = 0;
12
13 app.use(express.static('.')); // required to serve static files
14
15 function sendCurrentPlayers() {
16     if (adminWS && playerWSs.length > 0) {
17         allPlayers = playerWSs
18             .filter(player => player.name)
19             .map(player => { return { "id": player.id, "name": player.name } })
20         console.log("Sending current players: " + JSON.stringify(allPlayers));
21         adminWS.send(JSON.stringify({ "type": "players", "players": allPlayers }));
22     }
23 }
24
25 function sendNextQuestion() {
26     answersCount = 0;
27     correctAnswersCount = 0;
28     const question = "What is the capital of France?";
29     const answers = ["Paris", "London", "Berlin", "Madrid"];
30     const correct = "Paris";
31
32     const nextQuestion = JSON.stringify({
33         "type": "question",
34         "question": question,
35         "answers": ["Paris", "London", "Berlin", "Madrid"]
36     })
37     playerWSs.forEach(player => 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 => {
48         return { "id": player.id, "name": player.name, "wins": player.wins }
49     });
50     const sortedResults = results.sort((a, b) => 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   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   if (correct) {
65     if (correctAnswersCount === 0) {
66       player.wins++;
67     }
68     correctAnswersCount++;
69   }
70   answersCount++;
71   if (answersCount === playerWSs.length) {
72     // sendNextQuestion();
73     sendResults();
74   } else {
75     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   const playerData = { "ws": ws, "id": clientId, "wins": 0 }
87   playerWSs.push(playerData);
88   ws.onmessage = function (event) {
89     message = JSON.parse(event.data);
90     switch (message.type) {
91       case "registration":
92         const name = message.name;
93         console.log("Registration: " + clientId + "/" + name);
94         playerData.name = name;
95         sendCurrentPlayers();
96         break;
97
98       case "answer":
99         const answer = message;
100        handleAnswer(clientId, answer);
101        break;
102
103      default:
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     console.log("Player error: " + clientId);
115     playerWSs = playerWSs.filter(player => player.id !== clientId);
116     sendCurrentPlayers();
117   };
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     message = JSON.parse(event.data);
125     switch (message.type) {
126       case "start":
127         console.log("Start game");
128         sendNextQuestion();
129         break;
130       default:
131         console.log("Unknown message: " + message);
132         break;
133     }
134   };
135
136   ws.onclose = (event) => {
137     console.log("Admin disconnected");
138     adminWS = null;
139     sendCurrentPlayers();
140   };
141
142   ws.onerror = (event) => {
143     console.log("Admin error: " + event);
144     sendCurrentPlayers();
145   };
146
147 });
148
149
150 const PORT = process.env.QUIZZY_PORT || 8800;
151
152 var server = app.listen(PORT, function () {
153   console.log(`Quizzy running at http://127.0.0.1:${PORT}/`);
154 })

```

Quizzy - Client Code (quizzy.js)

```
1  /**
2   * A small web component which enable us to integrate a small quiz
3   * into a webpage.
4  */
5  const quizzyStyles = new CSSStyleSheet();
6  quizzyStyles.replaceSync(`^
7  :host {
8      display: block;
9      width: 100%;
10     height: 10lh;
11     background-color: #f0f0f0
12 }
13 main {
14     height: 100%;
15
16     .select-mode {
17         height: 100%;
18         display: flex;
19         flex-direction: column;
20         align-items: center;
21         gap: 1em;
22         align-content: space-around;
23         justify-content: space-around;
24
25         button,
26         button[type="button"] {
27             width: 40%;
28             height: 2.5lh;
29             flex-grow: 0;
30             padding: 0.5em;
31             font-size: inherit;
32         }
33     }
34 }
35 `);
36
37 class Quizzy extends HTMLElement {
38     constructor() {
39         super();
40
41         const shadow = this.attachShadow({ mode: "open" });
42
43         shadow.adoptedStyleSheets = [quizzyStyles];
44         shadow.innerHTML = `
45             <main>
46                 <div class="select-mode">
47                     <button id="clientButton">Join Quizzy</button>
48                     <button id="adminButton">Administrate Quizzy</button>
49                 </div>
50             </main>`;
51     }
52 }
```

```
52
53     connectedCallback() {
54         const wsURL = this.getAttribute("ws-url");
55
56         /**
57          * Client logic
58          */
59         this.shadowRoot
60             .getElementById("clientButton")
61             .addEventListener("click", () => {
62                 this.shadowRoot.innerHTML = `
63                     <main>
64                         <form>
65                             <input type="text" id="username" placeholder="Username">
66                             <button id="submit" type="button">Submit</button>
67                         </form>
68                     </main>`;
69
70         const ws = new WebSocket(` ${wsURL}/player`);
71
72         ws.onmessage = (event) => {
73             const data = JSON.parse(event.data);
74             switch (data.type) {
75                 case "question":
76                     showQuestion(data);
77                     break;
78                 case "results":
79                     showResults(data);
80                     break;
81                 default:
82                     console.warn("unknown message", data);
83                     break;
84             }
85         };
86         ws.onclose = (event) => {
87             console.log("connection closed", event);
88         };
89         ws.onerror = (event) => {
90             console.error("fatal error", event);
91         };
92
93         const showResults = (data) => {
94             console.log("show results", data);
95             const main = this.shadowRoot.querySelector("main");
96             main.innerHTML = `<h1>Results</h1>`;
97             for (const result of data.results) {
98                 main.innerHTML += `<p>${result.name}: ${result.wins}</p>`;
99             }
100        };
101
102        const showQuestion = (data) => {
103            const main = this.shadowRoot.querySelector("main");
104            main.innerHTML = `<h1>Question</h1><p>${data.question}</p>`;
105    
```

```
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
```

```
const createAnswerButton = (answer) => {
    const button = document.createElement("button");
    button.innerText = answer;
    button.onclick = submitAnswer(answer);
    return button;
};

for (const answer of data.answers) {
    main.appendChild(createAnswerButton(answer));
}

const submitAnswer = (answer) => {
    return () => {
        ws.send(
            JSON.stringify({
                type: "answer",
                answer: answer,
            }),
        );
        doWait();
    };
};

setTimeout(() => {
    this.shadowRoot
        .getElementById("submit")
        .addEventListener("click", () => {
            submitUsername();
        });
});

const submitUsername = () => {
    const name =
        this.shadowRoot.getElementById("username").value;
    ws.send(
        JSON.stringify({
            type: "registration",
            name: name,
        }),
    );
    doWait();
};

const doWait = () => {
    const main = this.shadowRoot.querySelector("main");
    main.innerHTML = "Waiting for other players...";
};

/** 
 * Admin logic
```

```

159     */
160     this.shadowRoot
161     .getElementsByClassName("adminButton")
162     .addEventListener("click", () => {
163       this.shadowRoot.innerHTML = `
164       <main>
165         <h1>Players</h1>
166         <p id="players"></p>
167         <button id="startGame" type="button">Start Game</button>
168       </main>`;
169
170     const ws = (this.ws = new WebSocket(` ${wsURL}/admin`));
171
172     ws.onmessage = (event) => {
173       const data = JSON.parse(event.data);
174       console.log("Received: " + event.data);
175       switch (data.type) {
176         case "players":
177           const players =
178             this.shadowRoot.getElementById("players");
179           players.innerText =
180             "[" +
181               data.players.length +
182                 " players] " +
183                   data.players
184                     .map(
185                       (player) =>
186                         player.id + ": " + player.name,
187                           )
188                         .join(", ");
189           break;
190         case "question":
191           this.shadowRoot.querySelector("main").innerText =
192             question: ${data.question}; correct answer: ${data.correct}
193             `;
194           break;
195         case "results":
196           const main = this.shadowRoot.querySelector("main");
197           main.innerText = "Result: " + event.data;
198           break;
199         default:
200           console.log("unknown: " + event.data);
201           break;
202         }
203     };
204
205     ws.onclose = (event) => {
206       console.log("Connection closed: " + event);
207       const main = this.shadowRoot.querySelector("main");
208       main.innerText = "Connection closed - you need to restart.";
209     };
210     ws.onerror = (event) => {
211       console.log("Connection error", event);

```

```
212     };
213
214     function startGame() {
215       ws.send(JSON.stringify({ type: "start" }));
216     }
217
218     setTimeout(() => {
219       this.shadowRoot
220         .getElementById("startGame")
221         .addEventListener("click", () => {
222           startGame();
223         });
224     });
225   });
226
227   console.log("Quizzy component connected", this.wsURL);
228 }
229
230 disconnectedCallback() {
231   console.log("Quizzy component disconnected");
232   // Clean up any resources, event listeners, or connections,... TODO
233 }
234
235
236 customElements.define("ld-quizzy", Quizzy);
```

Verwendung der Quizzy Komponente

```
<script src="code/quizzy-component/quizzy.js"></script>
<div onclick="event.stopPropagation();">
  <ld-quizzy ws-url="ws://localhost:5557">
    ← When used as a real component, we have to either specify the
    questions here (somehow encrypted) and send them to the server when
    required, or refer to a set of questions stored on the server by
    means of a parameter/attribute... →
  </ld-quizzy>
</div>
```