Webprogrammierung mit JavaScript

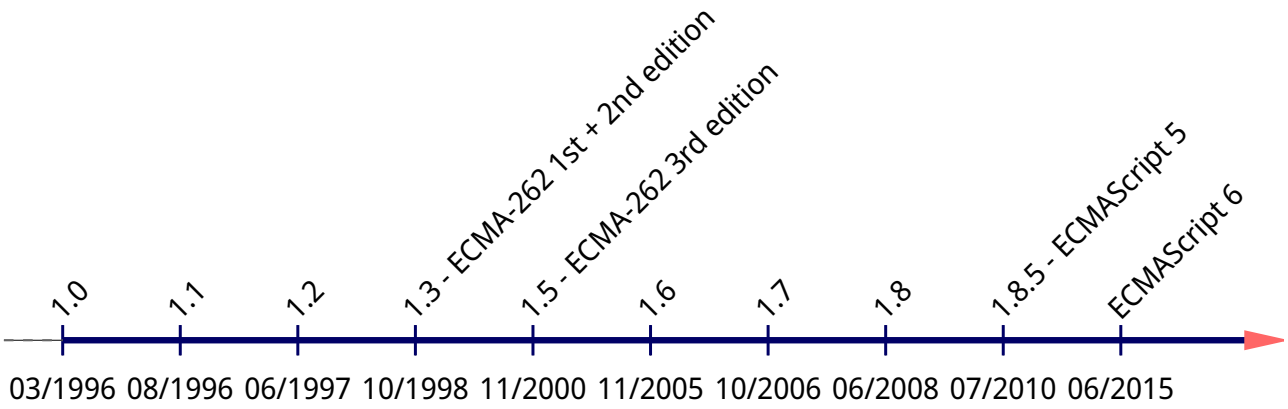
Eine kurze Einführung/eine kurze Übersicht über JavaScript für erfahrene Programmierer.

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



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`

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`)

```
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.

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
let l = 10_000;
let binary = 0b1010;
console.log("0b1010", binary);
let octal = 0o12;
console.log("0o12", octal );
let hex = 0xA;
console.log("0xA", hex);
console.log(
  Number.MIN_VALUE,
  Number.MIN_SAFE_INTEGER,
  Number.MAX_SAFE_INTEGER,
  Number.MAX_VALUE,
);
let x = NaN;
let y = Infinity;
let z = -Infinity;

// Standard Operatoren: +, - , *, /, %, ++, --, **
// Bitwise Operatoren: &, |, ^, ~, <<, >>, >>>
//                     (operieren immer auf dem Ganzzahlwert der Bits)
console.log("i =", i, "; i++  ", i++); // 1 oder 2?
console.log("i =", i, "; ++i ", ++i); // 2 oder 3?
console.log("2 ** 4 === 0 ", 2 ** 4);
console.log("7 % 3 === ", 7 % 3);
console.log("1 / 0 === ", 1 / 0);


console.log("BigInt -----------------------------------------------------------");
let ib = 1n;
console.log(100n === BigInt(100));
console.log(Number.MAX_SAFE_INTEGER + 2102); // 9007199254743092
console.log(BigInt(Number.MAX_SAFE_INTEGER) + 2102n); // 9007199254743093n


console.log("Boolean ---------------------------------------------------------");
let b = true; // oder false
console.log("Boolean(undefined)", Boolean(undefined)); // true oder false?
console.log(null == true ? "true" : "false"); // true oder false?


console.log("(Quasi-)Logische Operatoren ------------------------------------");
console.log('1 && "1": ', 1 && "1");
console.log('null && "1": ', null && "1");
console.log("null && true: ", null && true);
console.log("true && null: ", true && null);
console.log("null && false: ", null && false);
console.log("{} && true: ", {} && true);

// Neben den Standardoperatoren: ``&&``, ``||``, ``!`` gibt es auch noch ``??``
// Der ``??``-Operator gibt den rechten Operanden zurück, wenn der linke Operand
```

```
// ``null`` oder ``undefined`` ist. Andernfalls gibt er den linken Operanden
// zurück.
// ``??`` ist der *nullish coalescing operator (??) (vergleichbar zu ||)*
console.log('1 ?? "1": ', 1 ?? "1");
console.log('null ?? "1": ', null ?? "1");
console.log("null ?? true: ", null ?? true);
console.log("true ?? null: ", true ?? null);
console.log("null ?? false: ", null ?? false);
console.log("{} ?? true: ", {} ?? true);

console.log('undefined ?? "1": ', undefined ?? "1");
console.log('undefined ?? "1": ', undefined ?? "1");
console.log("undefined ?? true: ", undefined ?? true);
console.log("true ?? undefined: ", true ?? undefined);
console.log("undefined ?? false: ", undefined ?? false);
console.log("undefined ?? undefined: ", undefined ?? undefined);


console.log("Strings ------------------------------------------------------");
let _s = "42";
console.log("Die Antwort ist " + _s + "."); // String concatenation
console.log(`Die Antwort ist ${_s}.`); // Template literals (Template strings)
// multiline Strings
console.log(`
    Die Antwort mag ${_s} sein,
    aber was ist die Frage?`);

console.log(String(42)); // "42"


console.log("Objekte -----------------------------------------------------");
let emptyObject = null;
let anonymousObj = {
  i: 1,
  u: { j: 2, v: { k: 3 } },
  toString: function () {
    return "anonymousObj";
  },
  "?" : "question mark"
};
// Zugriff auf die Eigenschaften eines Objekts
anonymousObj.j = 2; // mittels Bezeichner ("j") (eng. Identifier)
anonymousObj["j"] = 4; // mittels String ("j")
anonymousObj["k"] = 3;
console.log("anonymousObj:                    ", anonymousObj);
console.log("anonymousObj.toString():         ", anonymousObj.toString());
delete anonymousObj["?"]; // delete dient dem Löschen von Eigenschaften
delete anonymousObj.toString; // delete dient dem Löschen von Eigenschaften
console.log("anonymousObj.toString() [original]", anonymousObj.toString());
// Der Chain-Operator kann verwendet werden, um auf Eigenschaften (Properties)
// von Objekten zuzugreifen, ohne dass eine Fehlermeldung ausgegeben wird,
// wenn eine (höher-liegende) Eigenschaft nicht definiert ist.
// Besonders nützlich beim Verarbeiten von komplexen JSON-Daten.
console.log("anonymousObj.u?.v.k", anonymousObj.u?.v.k);
console.log("anonymousObj.u.v?.k", anonymousObj.u.v?.k);
console.log("anonymousObj.u.v?.z", anonymousObj.u.v?.z);
console.log("anonymousObj.u.q?.k", anonymousObj.u.q?.k);
console.log("anonymousObj.p?.v.k", anonymousObj.p?.v.k);

// Nützliche Zuweisungen, um den Fall undefined und null gemeinsam zu behandeln:
```

```
118 anonymousObj.name ||= "Max Mustermann";
119
120
121
122 console.log("Date ------------------------------------------------------------");
123 let date = new Date("8.6.2024"); // ACHTUNG: Locale-Settings
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];
137 console.log("let $a = [1]; $a, $a.length", $a, $a.length);
138 $a.push(2); // append
139 console.log("$a.push(2); $a", $a);
140 temp = $a.unshift(0); // "prepend" -> return new length
141 console.log("temp = $a.unshift(0); temp, $a", temp, $a);
142 temp = $a.shift(); // remove first element -> return removed element
143 console.log("temp = $a.shift(); temp, $a", temp, $a);
144 // Um zu prüfen ob eine Datenstruktur ein Array ist:
145 console.log("Array.isArray($a)", Array.isArray($a));
146 console.log("Array.isArray({})", Array.isArray({}));
147 console.log("Array.isArray(1)", Array.isArray(1));
148
149
150 console.log("Symbols ---------------------------------------------------------");
151 let sym1 = Symbol("1"); // a unique and immutable primitive value
152 let sym2 = Symbol("1");
153 let obj1Values = { sym1: "value1", sym2: "value2" };
154 console.log(obj1Values);
155 console.log(`sym1 in ${JSON.stringify(obj1Values)}: `, sym1 in obj1Values);
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");

function hello(person = "World" /* argument with default value */) {
  log(`fun: Hello ${person}!`);
}
hello();

waitOnInput();

const helloExpr = function () { // Anonymer Funktionsausdruck
  log("expr: Hello World!");
};

// Arrow Functions
const times3 = (x) => x * 3;
log("times3(5)", times3(5)); // 15

const helloArrow = () => log("arrow: Hello World!");
const helloBigArrow = () => {
  const s = "Hello World!";
  log("arrow: " + s);
  return s;
};
helloExpr();
helloArrow();

var helloXXX = function helloYYY() { // benannter Funktionsausdruck
  // "helloYYY" ist _nur_ innerhalb der Funktion sichtbar und verwendbar
  // "arguments" ist ein Arrays-vergleichbares Objekt
  // und enthält alle Argumente der Funktion
  log(`Hello: `, ...arguments); // "..." ist der "Spread Operator"
};
helloXXX("Michael", "John", "Jane");

waitOnInput();

function sum(...args) {
  // rest parameter
  log("typeof args: " + typeof args + "; isArray: "+ Array.isArray(args));
  log("args: " + args);
  log("args:", ...args); // die Arraywerte werden als einzelne Args. übergeben
  return args.reduce((a, b) => a + b, 0); // function nesting
}
log(sum(1, 2, 3, 4, 5)); // 15
log(sum());

/* Generator Functions */
function* fib() {
  // generator
  let a = 0,
    b = 1;
  while (true) {
    yield a;
    [a, b] = [b, a + b];
  }
```

```
}
const fibGen = fib();
log(fibGen.next().value); // 0
log(fibGen.next().value); // 1
log(fibGen.next().value); // 1
log(fibGen.next().value); // 2
/* Will cause an infinite loop:
  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 `parseInt` verwenden

Vergleich von Werten und implizite Typumwandlung

```
// Gleichheit       ==      // mit Typumwandlung (auch bei <, >, ⩽, ⩾)

// strikt gleich    ===     // ohne Typumwandlung
// strike Ungleichheit  !== // ohne Typumwandlung

log('1 == "1": ', 1 == "1");
log('1 === "1": ', 1 === "1");
log("1.0 == 1: ", 1 == 1.0);
log("1.0 === 1: ", 1 === 1.0);
log("1 === 1n: ", 1 === 1n);
log("1 == 1n: ", 1 == 1n);
log('1 < "1"', 1 < "1");
log('0 < "1"', 0 < "1");
log('0 ⩽ "0"', 0 ⩽ "0");
log('"abc" ⩽ "d"', "abc" ⩽ "d");

log('"asdf" === "as" + "df"', "asdf" === "as" + "df"); // unlike Java!

log("NaN === NaN: ", NaN === NaN);
log("NaN == NaN: ", NaN == NaN);
log("null === NaN: ", null === NaN);
log("null == NaN: ", null == NaN);
log("null === null: ", null === null);
log("null == null: ", null == null);
log("undefined === undefined: ", undefined === undefined);
log("undefined == undefined: ", undefined == undefined);
log("null === undefined: ", null === undefined);
log("null == undefined: ", (null == undefined)+ "!");


const a1 = [1, 2, 3];
const a2 = [1, 2, 3];
log("const a1 = [1, 2, 3]; a1 == [1, 2, 3]: ", a1 == [1, 2, 3]);
log("const a1 = [1, 2, 3]; a1 == a1: ", a1 == a1);
log("const a1 = [1, 2, 3]; a1 === a1: ", a1 === a1);
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);
log(
  "flatEquals(a1,a2):",
  a1.length == a2.length && a1.every((v, i) => v === a2[i])
);


let firstJohn = { person: "John" };
show('let firstJohn = { person: "John" };');
let secondJohn = { person: "John" };
show('let secondJohn = { person: "John" };');
let basedOnFirstJohn = Object.create(firstJohn);
show("let basedOnFirstJohn = Object.create(firstJohn)");
log("firstJohn == firstJohn: ", firstJohn == firstJohn);
log("firstJohn === secondJohn: ", firstJohn === secondJohn);
log("firstJohn == secondJohn: ", firstJohn == secondJohn);
log("firstJohn === basedOnFirstJohn: ", firstJohn === basedOnFirstJohn);
log("firstJohn == basedOnFirstJohn: ", firstJohn == basedOnFirstJohn);


{
```

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

Bedingungen und Schleifen

```
const arr = [1, 3, 4, 7, 11, 18, 29];

log("if-else_if-else:");
if (arr.length == 7) {
  ilog("arr.length == 7");
} else if (arr.length < 7) {
  ilog("arr.length < 7");
} else {
  ilog("arr.length > 7");
}

log("\nswitch (integer value):");
switch (arr.length) {
  case 7:
    ilog("arr.length == 7");
    break;
  case 6:
    ilog("arr.length == 6");
    break;
  default:
    ilog("arr.length != 6 and != 7");
}

log("\nswitch (string value):");
switch ("foo") {
  case "bar":
    ilog("it's bar");
    break;
  case "foo":
    ilog("it's foo");
    break;
  default:
    ilog("not foo, not bar");
}

log("\nswitch (integer - no type conversion):");
switch (
  1 // Vergleich auf strikte Gleichheit (===)
) {
  case "1":
    ilog("string(1)");
    break;
  case 1:
    ilog("number(1)");
    break;
}

ilog("\nfor-continue:");
for (let i = 0; i < arr.length; i++) {
  const v = arr[i];
  if (v % 2 == 0) continue;
  log(v);
}

ilog("\n(for)-break with label:");
outer: for (let i = 0; i < arr.length; i++) {
  for (let j = 0; j < i; j++) {
```

```
    if (j == 3) break outer;
    log(arr[i], arr[j]);
  }
}

ilog("\nin (properties of Arrays; i.e. the indexes):");
for (const key in arr) {
  log(key, arr[key]);
}

ilog("\nof (values of Arrays):");
for (const value of arr) {
  log(value);
}

ilog("\nArray and Objects - instanceof:");
log("arr instanceof Object", arr instanceof Object);
log("arr instanceof Array", arr instanceof Array);

const obj = {
  name: "John",
  age: 30,
  city: "Berlin",
};

ilog("\nin (properties of Objects):");
for (const key in obj) {
  log(key, obj[key]);
}

/* TypeError: obj is not iterable
for (const value of obj) {
    log(value);
}
*/

{
  ilog("\nIteration über Iterables (here: Map):");
  const m = new Map();
  m.set("name", "Elisabeth");
  m.set("alter", 50);
  log("Properties of m: ");
  for (const key in m) {
    log(key, m[key]);
  }
  log("Values of m: ");
  for (const [key, value] of m) {
    log(key, value);
  }
}

{
  ilog("\nWhile Loop: ");
  let c = 0;
  while (c < arr.length) {
    const v = arr[c];
    if (v > 10) break;
    log(v);
    c++;
  }
}
```

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

Die Tatsache, dass insbesondere `null` als auch `undefined` falsy sind, wird of 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(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))`)!

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,"*"])` ⇒ `20`

Variables (var)

(Neuer Code sollte var nicht mehr verwenden!)

```
let y = "yyy"; // wie zuvor
const z = "zzz";

// Der Gültigkeitsbereich von var ist die umgebende Funktion oder der
// globale Gültigkeitsbereich.
// Die Definition ist hochgezogen (eng. "hoisted") (initialisiert mit undefined);
var x = "xxx";

function sumIfDefined(a, b) {
  // ⚠ Der folgende Code ist NICHT empfehlenswert!
  //   Er dient der Visualisierung des Verhaltens von var.
  if (parseInt(a)) {
    var result = parseInt(a);
  } else {
    result = 0;
  }
  const bVal = parseFloat(b);
  if (bVal) {
    result += bVal;
  }
  return result;
}

ilog("sumIfDefined()", sumIfDefined()); // 0
ilog("sumIfDefined(1)", sumIfDefined(1)); // 1
ilog("sumIfDefined(1, 2)", sumIfDefined(1, 2)); // 3
ilog('sumIfDefined(1, "2")', sumIfDefined(1, "2")); // 3
ilog("undefined + 2", undefined + 2);
ilog('sumIfDefined(undefined, "2")', sumIfDefined(undefined, "2")); // 2

function global_x() {
  ilog("global_x():", x, y, z);
}

function local_var_x() {
  ilog("local_var_x(): erste Zeile (x)", x);

  var x = 1; // the declaration of var is hoisted, but not the initialization
  let y = 2;
  const z = 3;

  ilog("local_var_x(): letzte Zeile (x, y, z)", x, y, z); // 1 2 3
}

global_x();
local_var_x();

ilog("nach global_x() und local_var_x() - x, y, z:", x, y, z);


// Hier, ist nur die Variablendeklaration (helloExpr) "hoisted", aber nicht
// die Definition. Daher kann die Funktion nicht vorher im Code aufgerufen
// werden!
try {
  helloExpr();
} 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:");

let [val1, val2] = [1, 2, 3, 4];
ilog("[val1, val2] = [1, 2, 3, 4]:", "val1:", val1, ", val2:", val2); // 1

log("Object Destructuring:");

let { a, b } = { a: "aaa", b: "bbb" };
ilog('let { a, b } = { a: "aaa", b: "bbb" }: ', "a:", a, ", b:", b); // 1

let { a: x, b: y } = { a: "aaa", b: "bbb" };
ilog('let { a: x, b: y } = { a: "aaa", b: "bbb" }: ', "x:", x, ", y:", y); // 1

let { a: u, b: v, ...w } = { a: "+", b: "-", c: "*", d: "/" };
ilog(
  'let { a: u, b: v, ...w } = { a: "+", b: "-", c: "*", d: "/" }:',
  "u:",
  u,
  ", v:",
  v,
  ", w:",
  JSON.stringify(w), // just for better readability/comprehension
);

let { k1, k2 } = { a: "a", b: "b" };
ilog('let { k1, k2 } = { a: "a", b: "b" }:', "k1:", k1, ", k2:", k2);
// "undefined undefined", weder k1 noch k2 sind definiert
```

JSON (JavaScript Object Notation)

```
const someJSON = `{
    "name": "John",
    "age": 30,
    "cars": {
        "American": ["Ford"],
        "German": ["BMW", "Mercedes", "Audi"],
        "Italian": ["Fiat","Alfa Romeo", "Ferrari"]
    }
}
`

// 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., `RegExp.test(String)`).
- Methoden auf Strings, die reguläre Ausdrücke verarbeiten: `search`, `match`, `replace`, `split`, ...

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

Klassen und Vererbung

```
class Figure {
  calcArea() {
    throw new Error("calcArea is not implemented");
  }
}
class Rectangle extends Figure {
  height;
  width;

  constructor(height, width) {
    super();
    this.height = height;
    this.width = width;
  }

  calcArea() {
    return this.height * this.width;
  }

  get area() {
    return this.calcArea();
  }

  set area(value) {
    throw new Error("Area is read-only");
  }
}

const r = new Rectangle(10, 20);
console.log("r instanceof Figure", r instanceof Figure); // true
console.log(r.width);
console.log(r.height);
console.log(r.area); // 200

try {
  r.area = 300; // Error: Area is read-only
} catch (e) {
  console.error(e.message);
}
```

Grundlagen von ECMAScript Modulen

Queue.mjs exportiert die Klasse Queue

```
/* Modul für den Datentyp Warteschlange (Queue). */
export class Queue {
  #last = null; // private field
  #first = null;
  constructor() {} // "default constructor"
  enqueue(elem) {
    if (this.#first === null) {
      const c = { e: elem, next: null };
      this.#first = c;
      this.#last = c;
    } else {
      const c = { e: elem, next: null };
      this.#last.next = c;
      this.#last = c;
    }
  }
  dequeue() {
    if (this.#first === null) {
      return null;
    } else {
      const c = this.#first;
      this.#first = c.next;
      return c.e;
    }
  }
  head() {
    if (this.#first === null) {
      throw new Error("Queue is empty");
    } else {
      return this.#first.e;
    }
  }
  last() {
    if (this.#first === null) {
      throw new Error("Queue is empty");
    } else {
      return this.#last.e;
    }
  }
  isEmpty() {
    return this.#first === null;
  }
}
```

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

```
import { Queue } from "./Queue.mjs"; // import des Moduls "Queue.mjs"

const messages = new Queue();

export function log(...message) {
  if (messages.isEmpty()) {
    messages.enqueue(message);
  } else {
    message.unshift("\n");
    messages.last().push(...message);
  }
```

```
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)

```
1  //"use strict";
2
3  function counter () {
4      // console.log(this === globalThis); // true
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
11     return this.count;
12 }
13
14 const counterExpr = function () {
15     if(this.count)
16         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);
27     this.count = this.count ? this.count + 1 : 1;
28     return this.count;
29 }
30
31 console.log("\nCounter");
32 console.log(counter()); // 1
33 console.log(counter()); // 2
34 console.log(`Counter (${globalThis.count})`);
35
36 console.log("\nCounterExpression");
37 console.log(counterExpr()); // 3
38 console.log(counterExpr()); // 4
39
40 console.log("\nCounter");
41 const obj = {};
42 console.log(counter.apply(obj)); // 1 - we set a new "this" object!
43 console.log(counterExpr.apply(obj)); // 2
44
45 console.log(`\nCounterArrow (${this.count})`);
46 console.log(counterArrow.apply(obj)); // 1
47 console.log(counterArrow.apply(undefined)); // 2
48 console.log(counterArrow.apply()); // 3
49 console.log(counterArrow.apply(obj)); // 4
50 console.log(counterArrow.apply({})); // 5
51
52 console.log("\nCounter (global)");
53 console.log(counter());
```

```
54 console.log(counterExpr());
```

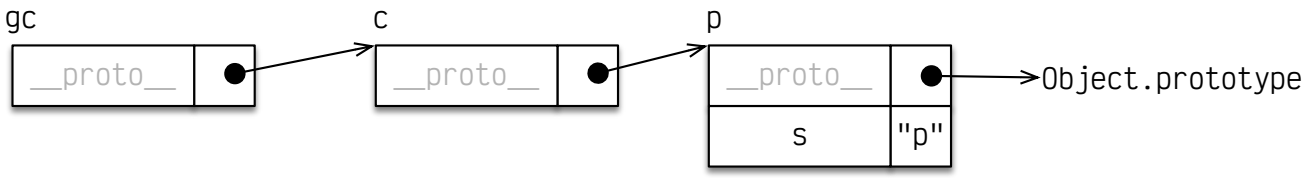
Partial Function Application

```
function add(x, y) {
  return x + y;
}

// Partial function application:
const add2 = add.bind(null, 2); // "null" is the value of "this"
console.log(add2(3));


function addToValue(b) {
  return this.x + b;
}
console.log(addToValue.call({x : 0}, -101));
```

Prototype basierte Vererbung

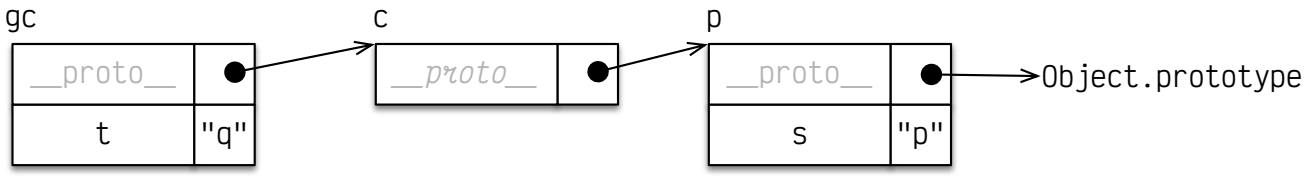
Verwendung von `Object.create` zur Initialisierung der *Prototype Chain*:

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



Verwendung der Eigenschaften von Prototypen:

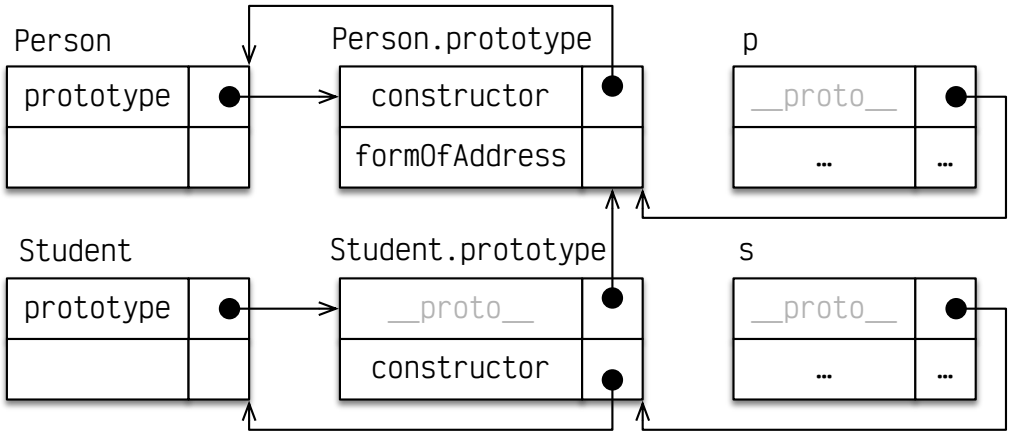
```
const p = { s : "p" };
const c = Object.create(p);
const gc = Object.create(c);
gc.t = "q";
```




```
gc.s = "gc"
console.log(gc.s); // gc
delete gc.s;
console.log(gc.s); // p
```

Pseudoclassical Inheritance

```
// constructor for Person objects:
function Person(name, title){ this.name = name; this.title = title; }
Person.prototype.formOfAddress = function (){
   const foa = "Dear ";
   if(this.title){ foa += this.title+" "; }
   return foa + this.name;
}
function Student(name, title, id, email) {
   Person.call(this, name, title); // super constructor call
   this.id = id;
   this.email = email;
}
Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;

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.

```
// 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 o = { 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);
}

// - ATTENTION! ------------------------------------------------------------
// 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.
Array.prototype.fold = function (f) {
  if (this.length == 0) {
    throw new Error("array is empty");
  } else if (this.length == 1) {
    return this[0];
  } else {
    let result = this[0];
    for (let i = 1; i < this.length; i++) {
      result = f(result, this[i]);
    }
    return result;
  }
};

console.log(a.fold((u, v) => u + v));
```

DOM Manipulation

```html
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>DOM Manipulation with JavaScript</title>
        <script>
            function makeScriptsEditable() {
                const scripts = document.getElementsByTagName("script");
                for (const scriptElement of scripts) {
                    scriptElement.contentEditable = false;
                    const style = scriptElement.style;
                    style.display = "block";
                    style.whiteSpace = "preserve";
                    style.padding = "1em";
                    style.backgroundColor = "yellow";
                }
            }
        </script>
    </head>
    <body>
        <h1>DOM Manipulation with JavaScript</h1>
        <p id="demo">This is a paragraph.</p>
        <button
            type="button"
            onclick="
                    document.getElementById('demo').style.color = 'red';
                    makeScriptsEditable();
                    document.querySelector('button').style.display = 'none';"
        >
            Magic!
        </button>

        <script>
            const demoElement = document.getElementById("demo");
            const style = demoElement.style;
            demoElement.addEventListener(
                "mouseover",
                () => (style.color = "green"),
            );
            demoElement.addEventListener(
                "mouseout",
                () => (style.color = "unset"),
            );
        </script>

        <p>Position der Mouse: <span id="position"></span></p>
        <script>
            window.addEventListener("mousemove", () => {
                document.getElementById("position").innerHTML =
                    `(${event.clientX}, ${event.clientY})`;
            });
        </script>
    </body>
</html>
```

Minimaler Server mit Express JS

```
// "express" and "cors" are CommonJS modules, which requires us to use the
// "default import" syntax.
import express from "express";

// Cross-Origin Resource Sharing (CORS); This is required to allow the browser
// using a different domain to load the HTML to make requests to this server.
// I. e., we can use the HTML file from the "web-javascript" project to make
// requests to this server.
import cors from "cors";
const APP_PORT = 5080;

const app = express();

app.get("/users", cors(), function (req, res) {
  res.set("Content-Type", "application/json");
  res.end(`{
        "user1" : {
           "name" : "dingo",
           "password" : "1234",
           "profession" : "chef",
           "id": 1
        },

        "user2" : {
           "name" : "ringo",
           "password" : "asdf",
           "profession" : "boss",
           "id": 3
        }
     }`);
});

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

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
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Eventhandling</title>
    </head>
    <body>
        <script>
            /* Using Promises:
            function getUsers() {
                fetch('http://127.0.0.1:4080/users')
                    .then(response => response.json())
                    .then(users => {
                        const usersElement = document.getElementById('users');
                        usersElement.innerText = JSON.stringify(users);
                    });
                    }
            */

            /* Using async/await: */
            async function getUsers() {
                let response = await fetch("http://127.0.0.1:5080/users");
                let users = await response.json();
                const usersElement = document.getElementById("users");
                usersElement.innerText = JSON.stringify(users);
            }
        </script>

        <div id="users"></div>
        <button onclick="getUsers()">Get Users</button>
    </body>
</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();

const expressWs = require('express-ws')(app);

let clients = 0;
let playerWSs = [];

let adminWS = null;
let answersCount = 0;
let correctAnswersCount = 0;

app.use(express.static('.')); // required to serve static files


function sendCurrentPlayers() {
    if (adminWS && playerWSs.length > 0) {
        allPlayers = playerWSs
            .filter(player => player.name)
            .map(player => { return { "id": player.id, "name": player.name } })
        console.log("Sending current players: " + JSON.stringify(allPlayers));
        adminWS.send(JSON.stringify({ "type": "players", "players": allPlayers }));
    }
}

function sendNextQuestion() {
    answersCount = 0;
    correctAnswersCount = 0;
    const question = "What is the capital of France?";
    const answers = ["Paris", "London", "Berlin", "Madrid"];
    const correct = "Paris";

    const nextQuestion = JSON.stringify({
        "type": "question",
        "question": question,
        "answers": ["Paris", "London", "Berlin", "Madrid"]
    })
    playerWSs.forEach(player => player.ws.send(nextQuestion));
    adminWS.send(JSON.stringify({
        "type": "question",
        "question": question,
        "answers": answers,
        "correct": correct
    }));
}

function sendResults() {
    const results = playerWSs.map(player => {
        return { "id": player.id, "name": player.name, "wins": player.wins }
    });
    const sortedResults = results.sort((a, b) => b.wins - a.wins);
    const resultsMsg = JSON.stringify({
        "type": "results",
        "results": sortedResults
```

```
    });
    playerWSs.forEach(player => player.ws.send(resultsMsg));
    adminWS.send(resultsMsg);

}


function handleAnswer(clientId, answer) {
    const correct = answer.answer === "Paris";
    const player = playerWSs.find(player => player.id === clientId);
    if (correct) {
        if (correctAnswersCount === 0) {
            player.wins++;
        }
        correctAnswersCount++;
    }
    answersCount++;
    if (answersCount === playerWSs.length) {
        // sendNextQuestion();
        sendResults();
    } else {
        adminWS.send(JSON.stringify({
            "type": "answers",
            "count": answersCount,
            "correctAnswersCount": correctAnswersCount
        }));
    }
}


app.ws('/player', function (ws, request) {
    const clientId = clients++;
    const playerData = { "ws": ws, "id": clientId, "wins": 0 }
    playerWSs.push(playerData);
    ws.onmessage = function (event) {
        message = JSON.parse(event.data);
        switch (message.type) {
            case "registration":
                const name = message.name;
                console.log("Registration: " + clientId + "/" + name);
                playerData.name = name;
                sendCurrentPlayers();
                break;

            case "answer":
                const answer = message;
                handleAnswer(clientId, answer);
                break;

            default:
                console.log("Unknown message: " + message);
                break;
        }
    };
    ws.onclose = function () {
        console.log("Player disconnected: " + clientId);
        playerWSs = playerWSs.filter(player => player.id !== clientId);
        sendCurrentPlayers();
    };
    ws.onerror = function () {
```

```
        console.log("Player error: " + clientId);
        playerWSs = playerWSs.filter(player => player.id != clientId);
        sendCurrentPlayers();
    };
});

app.ws('/admin', function (ws, req) {
    adminWS = ws;
    sendCurrentPlayers(); // when admin registers her/himself, send current players
    ws.onmessage = function (event) {
        message = JSON.parse(event.data);
        switch (message.type) {
            case "start":
                console.log("Start game");
                sendNextQuestion();
                break;
            default:
                console.log("Unknown message: " + message);
                break;
        }
    };

    ws.onclose = (event) => {
        console.log("Admin disconnected");
        adminWS = null;
        sendCurrentPlayers();
    };

    ws.onerror = (event) => {
        console.log("Admin error: " + event);
        sendCurrentPlayers();
    };

});


var server = app.listen(8800, function () {
    console.log("Quizzy running at http://127.0.0.1:8800/");
})
```

Client - Players

```
<!DOCTYPE html>
<html lang="en">

<head>
    <script>
        const ws = new WebSocket("ws://localhost:8800/player");
        ws.onmessage = (event) => {
            const data = JSON.parse(event.data);
            switch (data.type) {
                case "question":
                    console.log("Question: " + data.question);
                    showQuestion(data);
                    break;
                case "results":
                    const main = document.getElementById("main")
                    main.innerText = "Results: " + event.data;
                    break;
                default:
                    console.log("Unknown message: " + data);
```

```
                    break;
                }
            };
            ws.onclose = (event) => {
                console.log("Connection closed: " + event);
            }
            ws.onerror = (event) => {
                console.error("Error: " + event);
            }

            function showQuestion(data) {
                const main = document.getElementById("main")
                main.innerHTML = `<h1>Question</h1><p>${data.question}</p>`;

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

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

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

            function submitUsername() {
                const name = document.getElementById("username").value;
                ws.send(JSON.stringify({
                    "type": "registration",
                    "name": name
                }));

                doWait();
            }

            function doWait() {
                const main = document.getElementById("main");
                main.innerHTML = "Waiting for other players...";
            }
        </script>

    <body>

        <main id="main">
            <form>
                <input type="text" id="username" placeholder="Username">
                <button type="button" onclick="submitUsername();">Submit</button>
            </form>
        </main>
```

```
</body>

</html>
```

Client - Admin

```
<!DOCTYPE html>
<html lang="en">

<head>
    <script>
        const ws = new WebSocket("ws://localhost:8800/admin");

        ws.onmessage = (event) => {
            const data = JSON.parse(event.data);
            console.log("Received: " + event.data);
            switch (data.type) {
                case "players":
                    const players = document.getElementById("players")
                    players.innerText =
                        "["+data.players.length + " players] " +
                        data.players
                            .map(player => player.id + ": " + player.name)
                            .join(", ");
                    break;
                case "question":
                    showQuestion(data);
                    break;
                case "results":
                    const main = document.getElementById("main")
                    main.innerText = "Result: " + event.data;
                    break;
                default:
                    console.log("unknown: " + event.data);
                    break;
            }
        };

        ws.onclose = (event) => {
            console.log("Connection closed: " + event);
            const main = document.getElementById("main")
            main.innerText = "Connection closed - you need to restart.";
        };
        ws.onerror = (event) => {
            console.log("Connection error: " + event);
        };

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

        function showQuestion(data) {
            document.getElementById("main").innerText = `
                question: ${data.question}; correct answer: ${data.correct}
                `
        }
    </script>
</head>

<body>
    <main id="main">
```

```
56          <h1>Players</h1>
57          <p id="players"></p>
58          <button type="button" onclick="startGame();">Start Game</button>
59      </main>
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

```
import express from "express";
import fs from "fs";
import path from "node:path";
import { fileURLToPath } from "url";
import jwt from "jsonwebtoken";
import crypto from "crypto";
import bodyParser from "body-parser";

const app = express();

const SERVER_SECRET = crypto.randomBytes(64).toString("hex");
const users = JSON.parse(
  fs.readFileSync(
    path.resolve(path.dirname(fileURLToPath(import.meta.url)), "users.json"),
    "utf8",
  ),
);
console.log("Users: " + JSON.stringify(users));

app.use(express.static("."));
app.use(express.json());
app.use(bodyParser.text());

const verifyToken = (req, res, next) => {
  console.log("Headers: " + JSON.stringify(req.headers));

  const token = req.headers["authorization"].split(" ")[1];
  if (!token) {
    return res.status(401).json({ error: "Unauthorized" });
  }

  jwt.verify(token, SERVER_SECRET, (err, decoded) => {
    console.log("Decoded: " + JSON.stringify(decoded));
    if (err) {
      return res.status(401).json({ error: "Unauthorized" });
    }
    req.userIndex = decoded.userIndex;
    next();
  });
};

app.get("/admin/login", function (req, res) {
```

```
  const name = req.query.name;
  const password = req.query.password; // in a real app use hashed passwords!

  if (!name || !password) {
    res.status(400).send("Missing name or password");
    return;
  }

  let userIndex = -1;
  for (let i = 0; i < users.length; i++) {
    if (users[i].name === name && users[i].password === password) {
      userIndex = i;
      break;
    }
  }
  if (userIndex === -1) {
    res.status(401).send("Credentials invalid.");
    return;
  }
  console.log(
    "Authenticated: " + users[userIndex].name + " " + users[userIndex].password,
  );

  // Here, we can use the userIndex to identify the user;
  // but his only works as long as the user list is fixed.
  // In a real app use, e.g., a users's email.
  const token = jwt.sign({ userIndex: userIndex }, SERVER_SECRET, {
    expiresIn: "2h",
  });
  res.status(200).json({ token });
});

app.post("/admin/question", verifyToken, function (req, res) {
  const userIndex = req.userIndex;
  const question = req.body;
  console.log("Received question: " + question + " from user: " + users[userIndex].name);

  res.status(200).send("Question stored. Preliminary answer: 42.");
});

// Attention: a port like 6666 will not work on (most?) browsers
const port = 8080;
var server = app.listen(port, function () {
  console.log(`Running at http://127.0.0.1:${port}/`);
});
```

Client (JavaScript)

```
/*
Initializes the login interface.
*/
document
  .getElementsByTagName("main")[0]
  .replaceChildren(document.getElementById("log-in").content.cloneNode(true));
document.getElementById("login-dialog").showModal();
document.getElementById("login-button").addEventListener("click", login);

let jwt = null; // JSON Web Token for authentication

async function login() {
  const name = document.getElementById("administrator").value;
```

```
  const password = document.getElementById("password").value;
  const urlEncodedName = encodeURIComponent(name);
  const urlEncodedPassword = encodeURIComponent(password);
  const response = await fetch(
    "http://" +
      location.host +
      "/admin/login?name=" +
      urlEncodedName +
      "&password=" +
      urlEncodedPassword,
  );
  if (response.status !== 200) {
    console.error("Login failed: " + response.status);
    return;
  }
  const responseJSON = await response.json();
  jwt = responseJSON.token;
  console.log("Received JWT: " + jwt);

  document.getElementById("login-dialog").close();

  document
    .getElementsByTagName("main")[0]
    .replaceChildren(document.getElementById("logged-in").content.cloneNode(true));
  document.getElementById("enter-question-dialog").showModal();
  document.getElementById("send-question").addEventListener("click", sendQuestion);
}

async function sendQuestion() {
  const question = document.getElementById("question").value;

  const response = await fetch("http://" + location.host + "/admin/question", {
    method: "POST",
    headers: {
      "Content-Type": "text/plain",
      Authorization: `Bearer ${jwt}`,
    },
    body: question,
  });
  const text = await response.text();
  showAnswer(text);
}

function showAnswer(text) {
  document.getElementById("answer-dialog").showModal(false);
  document.getElementById("answer-paragraph").textContent = text;
}
```

Alle Quellen:

admin.js

admin.html

admin.css

server.mjs

start_server.sh

Users.json

Referenzen

- HTML DOM API