Webprogrammierung mit JavaScript

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

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw-mannheim.de, Raum 149B
Version: 2.0 (alpha)

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, ...)

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
- 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 (Node.js) oder in Desktop-Anwendungen (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`

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 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);
```

```
53 console.log("{} && true: ", {} && true);
54
55 // Neben den Standardoperatoren: ``&&``, ``||``, ``!`` gibt es auch noch ``??``
56 // Der ``??``-Operator gibt den rechten Operanden zurück, wenn der linke Operand
57 // ``null`` oder ``undefined`` ist. Andernfalls gibt er den linken Operanden zurück.
58 // ``??`` ist der *nullish coalescing operator (??) (vergleichbar zu ||)*
59 console.log('1 ?? "1": ', 1 ?? "1");
60 console.log('null ?? "1": ', null ?? "1");
61 console.log("null ?? true: ", null ?? true);
62 console.log("true ?? null: ", true ?? null);
63 console.log("null ?? false: ", null ?? false);
64 console.log("{} ?? true: ", {} ?? true);
65
66 console.log('undefined ?? "1": ', undefined ?? "1");
67 console.log('undefined ?? "1": ', undefined ?? "1");
68 console.log("undefined ?? true: ", undefined ?? true);
69 console.log("true ?? undefined: ", true ?? undefined);
70 console.log("undefined ?? false: ", undefined ?? false);
71 console.log("undefined ?? undefined: ", undefined ?? undefined);
72
73
74 console.log("Strings ----------------------------------------------------------------");
75 let _s = "42";
76 console.log("Die Antwort ist " + _s + "."); // String concatenation
77 console.log(`Die Antwort ist ${_s}.`); // Template literals (Template strings)
78 // multiline Strings
79 console.log(`
80     Die Antwort mag ${_s} sein,
81     aber was ist die Frage?`);
82
83 console.log(String(42)); // "42"
84
85
86 console.log("Objekte ----------------------------------------------------------------");
87 let emptyObject = null;
88 let anonymousObj = {
89   i: 1,
90   u: { j: 2, v: { k: 3 } },
91   toString: function () {
92     return "anonymousObj";
93   },
94   "?" : "question mark"
95 };
96 // Zugriff auf die Eigenschaften eines Objekts
97 anonymousObj.j = 2; // mittels Bezeichner ("j") (eng. Identifier)
98 anonymousObj["j"] = 4; // mittels String ("j")
99 anonymousObj["k"] = 3;
100 console.log("anonymousObj:                    ", anonymousObj);
101 console.log("anonymousObj.toString():         ", anonymousObj.toString());
102 delete anonymousObj["?"]; // delete operator dient dem Löschen von Eigenschaften
103 delete anonymousObj.toString; // delete operator dient dem Löschen von Eigenschaften
104 console.log("anonymousObj.toString() [original]", anonymousObj.toString());
105 // Der Chain-Operator kann verwendet werden, um auf Eigenschaften (Properties)
106 // von Objekten zuzugreifen, ohne dass eine Fehlermeldung ausgegeben wird,
107 // 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:
anonymousObj.name ||= "Max Mustermann";



console.log("Date ----------------------------------------------------------------------");
let date = new Date("8.6.2024"); // ACHTUNG: Locale-Settings
console.log(date);


console.log("Funktionen sind auch Objekte ----------------------------------------------");
let func = function () {
  return "Hello World";
};
console.log(func, func());


console.log("Arrays ------------------------------------------------------------------");
let temp = undefined;
let $a = [1];
console.log("let $a = [1]; $a, $a.length", $a, $a.length);
$a.push(2); // append
console.log("$a.push(2); $a", $a);
temp = $a.unshift(0); // "prepend" -> return new length
console.log("temp = $a.unshift(0); temp, $a", temp, $a);
temp = $a.shift(); // remove first element -> return removed element
console.log("temp = $a.shift(); temp, $a", temp, $a);
// Um zu prüfen ob eine Datenstruktur ein Array ist:
console.log("Array.isArray($a)", Array.isArray($a));
console.log("Array.isArray({})", Array.isArray({}));
console.log("Array.isArray(1)", Array.isArray(1));


console.log("Symbols -------------------------------------------------------------------");
let sym1 = Symbol("1"); // a unique and immutable primitive value
let sym2 = Symbol("1");
let obj1Values = { sym1: "value1", sym2: "value2" };
console.log(obj1Values);
console.log(`sym1 in ${JSON.stringify(obj1Values)}: `, sym1 in obj1Values);
let obj2Values = { [sym1]: "value1", [sym2]: "value2" };
console.log(obj2Values);
console.log(`sym1 in ${JSON.stringify(obj2Values)}: `, sym1 in obj2Values);
console.log(obj1Values, " vs. ", obj2Values);

console.log( { sym1 : "this", sym1 : "that"  }); // ??? { sym1: "that" }
console.log("sym1 == sym2", sym1 == sym2);
```

Funktionsdefinitionen

```
// Die Funktionsdeklaration der Funktion "hello" ist "hochgezogen" (eng. "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); // es werden alle Elemente des Arrays als einzelne Argumente übergeb
  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 - vertraut machen mit JavaScript und Node.js

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

Hello World in Node.js

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

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.

Hello World in Node.js

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

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

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

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 <, >, <=, >=)
// Ungleichheit     !==
// strikt gleich    ===     // 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);
```

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

```
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 for (const value of obj) {
90     log(value);
91 }
92 */
93
94 {
95   ilog("\nIteration über Iterables (here: Map):");
96   const m = new Map();
97   m.set("name", "Elisabeth");
98   m.set("alter", 50);
99   log("Properties of m: ");
100  for (const key in m) {
101    log(key, m[key]);
102  }
103  log("Values of m: ");
104  for (const [key, value] of m) {
105    log(key, value);
106  }
107 }
108
```

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

{
  ilog("\nDo-While Loop: ");
  let c = 0;
  do {
    log(arr[c]);
    c++;
  } while (c < arr.length);
}
```


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

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))`)!

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

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.

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

Einfacher RPN Calculator

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

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

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

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);


```

```
51 // Hier, ist nur die Variablendeklaration (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();
```

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`, ...

```
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 {
9     const p = /[1-9]+H/g;
10    const s = "1H, 2H, 3P, 4C";
11    console.log(s.match(p));
12    console.log(s.replace(p, "XX"));
13 }
```

`this` in Funktionen und partielle Funktionsaufrufe

```
/** Partial function application */

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

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

waitOnInput();

/** "this" in functions */
log('"this" in functions:');

function f(c) {
  const that = this;

  function f() {
    return this === that;
  }

  const fExpr = () => {
    return this === that;
  };

  log(" globalThis === this: " + (globalThis === this));
  log(" this === that (function): " + f());
  log(" f.bind({}); this === that (function): " + f.bind({})());
  log(" this === that (function expression): " + fExpr());
  //log()
}
f();

waitOnInput();

globalThis.x = -1;
this.x = 0;
log(
  "this.x: " + this.x + "(globalThis === this: " + (globalThis === this) + ")",
);

function addToValue(b) {
  return this.x + b;
}
log(addToValue(1));
log(addToValue.call(this, -101));

const obj = { x: 101, addToValue: addToValue };
log("obj.addtoValue(-101): " + obj.addToValue(-101));

const add100 = addToValue.bind({ x: 100 });
log("add100: " + add100(100));
```

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)

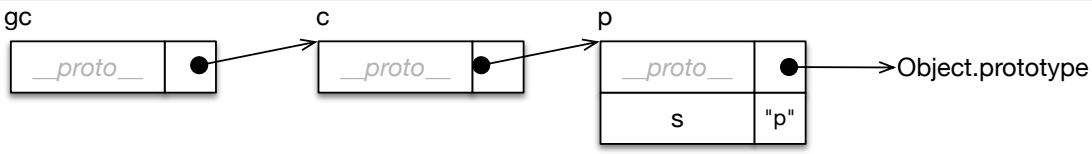
```
1  //"use strict";
2
3  function counter () {
4      // console.log(this === globalThis); // true
5      if(this.count)  // this is the global object if we don't use strict mode
6          this.count ++;
7      else {
8          this.count = 1;
9      }
10
11     return this.count;
12 }
13
14 const 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());
```

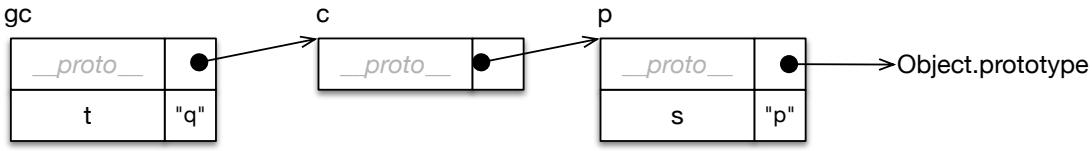
Prototype basierte Vererbung

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



1

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



2

Pseudoclassical Inheritance

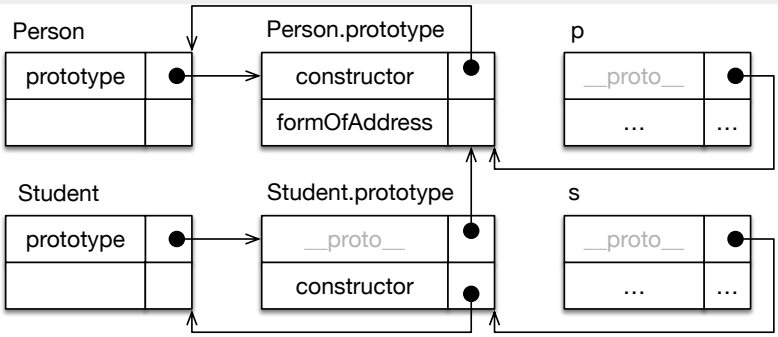
```
1  function Person(name, title){ this.name = name; this.title = title; } // constructor
2  Person.prototype.formOfAddress = function (){
3     const foa = "Dear ";
4     if(this.title){ foa += this.title+" "; }
5     return foa + this.name;
6  }
7  function Student(name, title, id, email) { // constructor
8     Person.call(this, name, title);
9     this.id = id;
10    this.email = email;
11 }
12 Student.prototype = Object.create(Person.prototype);
13 Student.prototype.constructor = Student;
14
15 const aStudent = new Student("Emilia Galotti", "Mrs.", 1224441, 'emilia@galotti.com');
```

3

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(…); const s = new Student(…);
```



4

Prototype basierte Vererbung

```
1  console.log({}.__proto__);
2  console.log(Array.prototype);
3  console.log(Array.prototype.__proto__);
4  console.log(Object.prototype);
5  console.log(Object.__proto__);
6
7  try {
8    let a = [1];
9    console.log(a.fold());
10 } catch (error) {
11   console.log("error: ", error.message);
12 }
13
14 // ADDING FUNCTIONS TO Array.prototpye IS NOT RECOMMENDED!
15 // IF ECMAScript EVENTUALLY ADDS THIS METHOD (I.E. fold)
16 // TO THE PROTOTYPE OF ARRAY OBJECTS, IT MAY CAUSE HAVOC.
17 Array.prototype.fold = function (f) {
18   if (this.length === 0) {
19     throw new Error("array is empty");
20   } else if (this.length === 1) {
21     return this[0];
22   } else {
23     let result = this[0];
24     for (let i = 1; i < this.length; i++) {
25       result = f(result, this[i]);
26     }
27     return result;
28   }
29 };
30
31 let a = [1, 10, 100, 1000];
32 console.log(a.fold((u, v) => u + v));
33
34 let o = { created: "long ago" };
35 var p = Object.create(o);
36 console.log(Object.getPrototypeOf(o));
37 console.log(o.isPrototypeOf(p));
38 console.log(Object.prototype.isPrototypeOf(p));
```

Classes

```
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);
}

class Queue {
  #last = null;
  #first = null;

  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;
    }
  }

  isEmpty() {
    return this.#first === null;
  }
}

const q = new Queue();
q.enqueue(1);
console.log("new Queue().enqueu(1).dequeue()", q.dequeue());
q.enqueue("first");
try {
  /* q.first is not our private field: */ console.log("q.first", q.first);

  // Compile(!) time error: console.log("q.#first", q.#first);
} catch (error) {
  console.error(error);
}
```

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>
```

```
53     </body>
54 </html>
```

Interaktion mit Server

```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>
            const box = document.getElementById("box");
            let color = 0;
            const setColor = () => {
                color = (color + 1) % 512;
                let rgb = color;
                if (rgb > 255) rgb = 256 - (rgb - 255);
                // console.log(rgb);
                document.body.style.backgroundColor = `rgb(${rgb}, ${rgb}, ${rgb})`;
            };
            setInterval(setColor, 10); // the function setColor is called every 10ms

            /* 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://127.0.0.1:4080/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>
```

Referenzen

- HTML DOM API