

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 (*Identifizier*)

(Sehr vergleichbar mit Java.)

- Buchstaben (Unicode), Ziffern, Unterstriche, Dollarzeichen
- Ein Identifizier 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 | ■ alert |
| ■ document (bzw. window.document) | ■ navigator |
| | ■ location |

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

- | | |
|-----------|-----------|
| ■ module | ■ process |
| ■ exports | ■ crypto |
| ■ require | |

Deklaration von Variablen (const und let)

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

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 Bibliotheken (log.mjs und Queue.mjs) finden Sie auf:

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

Datentypen

```
1 // Undefined
2 let u = undefined;
3
4 // Number
5 let i = 1; // double-precision 64-bit binary IEEE 754 value
6 let f = 1.0; // double-precision 64-bit binary IEEE 754 value
7 let l = 10_000;
8 let binary = 0b1010;
9 console.log("0b1010", binary);
10 let octal = 0o12;
11 console.log("0o12", octal );
12 let hex = 0xA;
13 console.log("0xA", hex);
14 console.log(
15   Number.MIN_VALUE,
16   Number.MIN_SAFE_INTEGER,
17   Number.MAX_SAFE_INTEGER,
18   Number.MAX_VALUE,
19 );
20 let x = NaN;
21 let y = Infinity;
22 let z = -Infinity;
23
24 // Standard Operatoren: +, -, *, /, %, ++, --, **
25 // Bitwise Operatoren: &, |, ^, ~, <<, >>, >>> (operieren auf dem Ganzzahlwert der Bits)
26 console.log("i=", i, "i++ ", i++); // 1 oder 2?
27 console.log("i=", i, "++i ", ++i); // 2 oder 3?
28 console.log("2 ** 4 === 0 ", 2 ** 4);
29 console.log("7 % 3 === ", 7 % 3);
30 console.log("1 / 0 === ", 1 / 0);
31
32 // BigInt
33 let ib = 1n; // Number.MAX_SAFE_INTEGER 9007199254740991n
34 console.log(100n === BigInt(100));
35 console.log(Number.MAX_SAFE_INTEGER + 2102); // 9007199254743093
36 console.log(BigInt(Number.MAX_SAFE_INTEGER) + 2102n);
37
38
39 // Boolean
40 let b = true; // oder false
41 console.log("Boolean(undefined)", Boolean(undefined)); // true oder false?
42
43
44 // Strings
45 let _s = "42";
46 console.log("Die Antwort ist " + _s); // Template literals (Template strings)
47 console.log(`Die Antwort ist ${_s}.`); // Template literals (Template strings)
48 console.log(`
49   Die Antwort mag ${_s} sein,
50   aber was ist die Frage?`);
51
```



```

52 console.log(String(42)); // "42"
53
54 // Zugriff auf die Eigenschaften eines Objekts
55 let anonymousObj = {
56   i: 1,
57   u: { j: 2, v: { k: 3 } },
58   toString: function () {
59     return "anonymousObj";
60   },
61 };
62 anonymousObj.j = 2; // mittels Bezeichner ("j") (eng. Identifier)
63 anonymousObj["j"] = 4; // mittels String ("j")
64 anonymousObj["k"] = 3;
65 console.log("anonymousObj:           ", anonymousObj);
66 console.log("anonymousObj.toString(): ", anonymousObj.toString());
67 delete anonymousObj.toString;
68 console.log("anonymousObj.toString() [original]", anonymousObj.toString());
69 // Der Chain-Operator kann verwendet werden, um auf Eigenschaften (Properties)
70 // von Objekten zuzugreifen, ohne dass eine Fehlermeldung ausgegeben wird,
71 // wenn eine (höher-liegende) Eigenschaft nicht definiert ist.
72 console.log("anonymousObj.u?.v.k", anonymousObj.u?.v.k);
73 console.log("anonymousObj.u.v?.k", anonymousObj.u.v?.k);
74 console.log("anonymousObj.u.v?.z", anonymousObj.u.v?.z);
75 console.log("anonymousObj.u.q?.k", anonymousObj.u.q?.k);
76 console.log("anonymousObj.p?.v.k", anonymousObj.p?.v.k);
77
78 // Neben den Standardoperatoren: ""&&"" , ""||"" , ""!"" gibt es auch noch ""??""
79
80 // Nützliche Zuweisungen, um den Fall undefined und null gemeinsam zu behandeln:
81 anonymousObj.name ||= "Max Mustermann";
82
83 /* Operator Madness */
84 console.log('1 && "1": ', 1 && "1");
85 console.log('null && "1": ', null && "1");
86 console.log("null && true: ", null && true);
87 console.log("true && null: ", true && null);
88 console.log("null && false: ", null && false);
89 console.log("{} && true: ", {} && true);
90
91 // nullish coalescing operator (??) (vergleichbar zu ||)
92 console.log('1 ?? "1": ', 1 ?? "1");
93 console.log('null ?? "1": ', null ?? "1");
94 console.log("null ?? true: ", null ?? true);
95 console.log("true ?? null: ", true ?? null);
96 console.log("null ?? false: ", null ?? false);
97 console.log("{} ?? true: ", {} ?? true);
98
99 console.log('undefined ?? "1": ', undefined ?? "1");
100 console.log('undefined ?? "1": ', undefined ?? "1");
101 console.log("undefined ?? true: ", undefined ?? true);
102 console.log("true ?? undefined: ", true ?? undefined);
103 console.log("undefined ?? false: ", undefined ?? false);
104 console.log("undefined ?? undefined: ", undefined ?? undefined);
105
106

```

```
107 // Date
108 let date = new Date("8.6.2024"); // ACHTUNG: Locale-Settings
109 console.log(date);
110
111 let emptyObject = null;
112
113 let func = function () {
114     return "Hello World";
115 };
116 console.log(func, func());
117
118
119 // Array
120 let temp = undefined;
121 let $a = [1];
122 console.log("let $a = [1]; $a, $a.length", $a, $a.length);
123 $a.push(2);
124 console.log("$a.push(2); $a", $a);
125 temp = $a.unshift(0); // "prepend"
126 console.log("temp = $a.unshift(0); temp, $a", temp, $a);
127 temp = $a.shift(); // remove first element
128 console.log("temp = $a.shift(); temp, $a", temp, $a);
129 // Um zu prüfen ob eine Datenstruktur ein Array ist:
130 console.log("Array.isArray($a)", Array.isArray($a));
131 console.log("Array.isArray({})", Array.isArray({}));
132 console.log("Array.isArray(1)", Array.isArray(1));
133
134
135
136 // Symbols
137 let sym1 = Symbol("1"); // a unique and immutable primitive value
138 let sym2 = Symbol("1");
139 let obj1Values = { sym1: "value1", sym2: "value2" };
140 console.log(obj1Values);
141 console.log(`sym1 in ${JSON.stringify(obj1Values)}: `, sym1 in obj1Values);
142 let obj2Values = { [sym1]: "value1", [sym2]: "value2" };
143 console.log(obj2Values);
144 console.log(`sym1 in ${JSON.stringify(obj2Values)}: `, sym1 in obj2Values);
```

Funktionsdefinitionen

```
1 // Die Funktionsdeklaration von "hello" ist "hoisted" und kann hier
2 // verwendet werden.
3 hello("Michael");
4
5 function hello(person = "World") {
6   // argument with default value
7   log(`fun: Hello ${person}!`);
8 }
9
10 waitOnInput();
11
12 const helloExpr = function () {
13   log("expr: Hello World!");
14 };
15
16 // Arrow Functions
17 const times3 = (x) => x * 3;
18 log("times3(5)", times3(5)); // 15
19 const helloArrow = () => log("arrow: Hello World!");
20 const helloBigArrow = () => {
21   const s = "Hello World!";
22   log("arrow: " + s);
23   return s;
24 };
25
26 log("Hello World!");
27 helloExpr();
28 helloArrow();
29
30 var helloXXX = function helloXXX() {
31   // *Function expression* mit Name
32   // arguments ist ein Arrays-vergleichbares Objekt
33   // und enthält die Argumente der Funktion
34   log(`Hello: `, ...arguments);
35   // Wir könnten hier (und nur hier) helloXXX verwenden,
36   // um die Funktion rekursiv aufzurufen.
37 };
38 helloXXX("Michael", "John", "Jane");
39
40 waitOnInput();
41
42 function sum(...args) {
43   // rest parameter
44   log("args: " + args);
45   log(...args); // we use the spread operator here
46   return args.reduce((a, b) => a + b, 0); // function nesting
47 }
48 log(sum(1, 2, 3, 4, 5)); // 15
49 log(sum());
50
51 /* Generator Functions */
52 function* fib() {
```

```
53 // generator
54 let a = 0,
55     b = 1;
56 while (true) {
57     yield a;
58     [a, b] = [b, a + b];
59 }
60 }
61 const fibGen = fib();
62 log(fibGen.next().value); // 0
63 log(fibGen.next().value); // 1
64 log(fibGen.next().value); // 1
65 log(fibGen.next().value); // 2
66 /* Will cause an infinite loop: for (const i of fib()) console.log(i);
67    // 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 ... */
```

Übung

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. 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. 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:

- `document.body` enthält den gesamten Inhalt des Dokuments
- die aktuellen Abmaße des Dokuments können Sie mit der Funktion `window.getComputedStyle(<HTML Element>).height` ermitteln
- 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:

- `document.body` enthält den gesamten Inhalt des Dokuments
- die aktuellen Abmaße des Dokuments können Sie mit der Funktion `window.getComputedStyle(<HTML Element>).height` ermitteln
- 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

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

```

52 let sym1 = Symbol("1"); // a unique and immutable primitive value
53 log(sym1, sym1, "=== ", sym1 === sym1); // true
54 let sym2 = Symbol("1");
55 let objValues = { sym1: "value1", sym2: "value2" };
56 log('let objValues = { sym1: "value1", sym2: "value2" };');
57 let obj2Values = { [sym1]: "value1", [sym2]: "value2" };
58 log(objValues, " === ", obj2Values, " vs. ", objValues === obj2Values);
59 let obj1Value = { [sym1]: "value1", [sym1]: "value2" };
60 log(obj2Values, " vs. ", obj1Value);
61 log(sym1, sym2, "=== ", sym1 === sym2); // false
62 log(sym1, sym2, "=== ", sym1 == sym2); // false
63 log(Symbol.for("1"), sym1, "=== ", Symbol.for("1") === sym1);
64
65 {
66   const obj = {
67     name: "John",
68     age: 30,
69     city: "Berlin",
70   };
71   log("\nTyptests und Feststellung des Typs:");
72   log("typeof obj", typeof obj);
73   log("obj instanceof Object", obj instanceof Object);
74   log("obj instanceof Array", obj instanceof Array);
75 }
76 {
77   const obj = { a: "lkj" };
78   const obj2 = Object.create(obj);
79   log(obj2 instanceof obj.constructor);
80 }
81
82 log("\n?-Operator and Truthy and Falsy Values:");
83 log('""', "" ? "is truthy" : "is falsy");
84 log("f()", (() => {}) ? "is truthy" : "is falsy");
85 log("Array ", Array ? "is truthy" : "is falsy");
86 log("obj ", {} ? "is truthy" : "is falsy");
87 log("undefined ", undefined ? "is truthy" : "is falsy");
88 log("null ", null ? "is truthy" : "is falsy");
89 log("0", 0 ? "is truthy" : "is falsy");
90 log("1", 1 ? "is truthy" : "is falsy");

```

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 }
```

```

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

```

```
109 {
110   ilog("\nWhile Loop: ");
111   let c = 0;
112   while (c < arr.length) {
113     const v = arr[c];
114     if (v > 10) break;
115     log(v);
116     c++;
117   }
118 }
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 }
```

Fehlerbehandlung

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

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!

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!

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,"*"])` \Rightarrow 20

Variables (var)

```
1 import { ilog, log, done } from "./log.mjs";
2
3 let y = "yyy"; // wie zuvor
4 const z = "zzz";
5
6 // Der Gültigkeitsbereich von var ist die umgebende Funktion oder der
7 // globale Gültigkeitsbereich.
8 // Die Definition ist ("hoisted") hochgezogen (initialisiert mit undefined);
9 // In modernem JavaScript sollte var nicht mehr verwendet werden.
10 var x = "xxx";
11
12 function sumIfDefined(a, b) {
13   // ⚠ Der folgende Code ist NICHT empfehlenswert!
14   // Er dient der Visualisierung des Verhaltens
15   // von var.
16   if (parseInt(a)) {
17     var result = parseInt(a);
18   } else {
19     result = 0;
20   }
21   const bVal = parseFloat(b);
22   if (bVal) {
23     result += bVal;
24   }
25   return result;
26 }
27
28 ilog("sumIfDefined()", sumIfDefined()); // 0
29 ilog("sumIfDefined(1)", sumIfDefined(1)); // 1
30 ilog("sumIfDefined(1, 2)", sumIfDefined(1, 2)); // 3
31 ilog('sumIfDefined(1, "2")', sumIfDefined(1, "2")); // 3
32 ilog("undefined + 2", undefined + 2);
33 ilog('sumIfDefined(undefined, "2")', sumIfDefined(undefined, "2")); // 2
34
35 function global_x() {
36   ilog("global_x():", x, y, z);
37 }
38
39 function local_var_x() {
40   ilog("local_var_x(): erste Zeile (x)", x);
41
42   var x = 1; // the declaration of var is hoisted, but not the initialization
43   let y = 2;
44   const z = 3;
45
46   ilog("local_var_x(): letzte Zeile (x, y, z)", x, y, z); // 1 2 3
47 }
48
49 global_x();
50 local_var_x();
51
52 ilog("nach global_x() und local_var_x() - x, y, z:", x, y, z);
```

```
53
54
55 // Hier, ist nur die Variablendeklaration (helloExpr) "hoisted", aber nicht
56 // die Definition. Daher kann die Funktion nicht vorher im Code aufgerufen
57 // werden!
58 try {
59     helloExpr();
60 } catch ({error, message}) {
61     log("calling helloExpr() failed:", error, "; message: ", message);
62 }
63 var helloExpr = function () {
64     log("expr: Hello World!");
65 };
66 // ab jetzt funktioniert es
67 helloExpr();
68
69 done();
```

this in Funktionen und partielle Funktionsaufrufe

```
1 /** Partial function application */
2
3 function add(x, y) {
4   return x + y;
5 }
6
7 const add2 = add.bind(null, 2); // the "null" is the value of "this"
8 log(add2(3));
9
10 waitOnInput();
11
12 /** "this" in functions */
13 log('"this" in functions:');
14
15 function f(c) {
16   const that = this;
17
18   function f() {
19     return this === that;
20   }
21
22   const fExpr = () => {
23     return this === that;
24   };
25
26   log(" globalThis === that: " + (globalThis === this));
27   log(" this === that (function): " + f());
28   log(" f.bind({}); this === that (function): " + f.bind({})();
29   log(" this === that (function expression): " + fExpr());
30 }
31 f();
32
33 waitOnInput();
34
35 globalThis.x = -1;
36 this.x = 0;
37 log(
38   "this.x: " + this.x + "(globalThis === this: " + (globalThis === this) + ")",
39 );
40
41 function addToValue(b) {
42   return this.x + b;
43 }
44 log(addToValue(1));
45 log(addToValue.call(this, -101));
46
47 const obj = { x: 101, addToValue: addToValue };
48 log("obj.addToValue(-101): " + obj.addToValue(-101));
49
50 const add100 = addToValue.bind({ x: 100 });
51 log("add100: " + add100(100));
```


Destructuring

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

JSON

```
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
12 const someObject = JSON.parse(someJSON);
13 someObject.age = 31;
14 someObject.cars.German.push("Porsche");
15 someObject.cars.Italian.pop();
16 console.log(someObject);
17 console.log(JSON.stringify(someObject, null, 2));
```

Reguläre Ausdrücke

- Built-in support by means of regular expression literals and an API
- Use Perl syntax
- Methods on regular expression objects: test (e.g., `RegExp.test(String)`).
- Methods on strings that take RegExps: 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 }
```

Alles ist ein Objekt

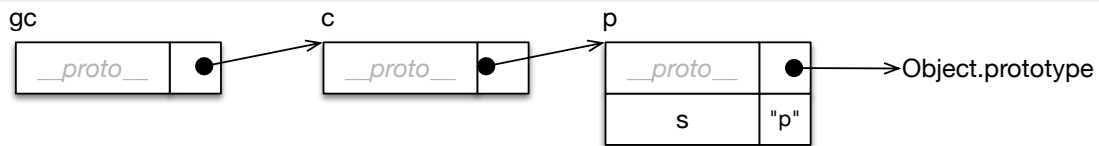
- **this** ist ein "zusätzlicher" Parameter, dessen Wert von der aufrufenden Form abhängt
- **this** ermöglicht den Methoden den Zugriff auf ihr Objekt
- **this** wird zum Zeitpunkt des Aufrufs gebunden (außer bei Arrow-Funktionen)

```
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());
```

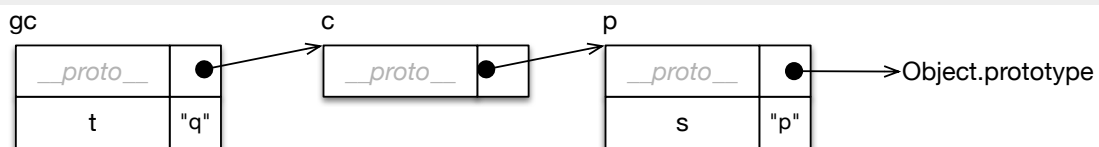
Prototype basierte Vererbung

```
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

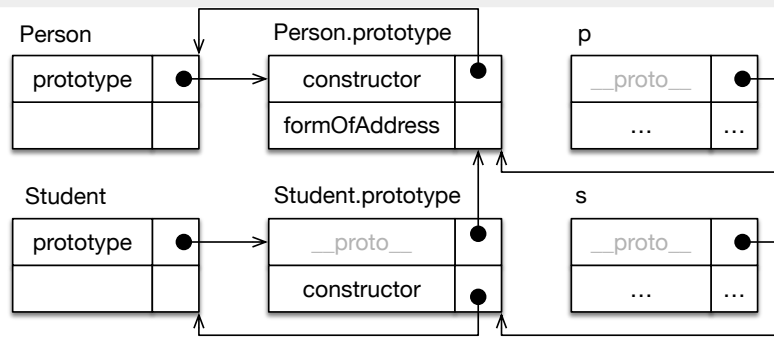
Objektabhängigkeiten

```
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(...);

```



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

Classes

```
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    this.height = height;
12    this.width = width;
13  }
14
15  calcArea() {
16    return this.height * this.width;
17  }
18
19  get area() {
20    return this.calcArea();
21  }
22
23  set area(value) {
24    throw new Error('Area is read-only');
25  }
26 }
27
28 const r = new Rectangle(10, 20);
29 console.log("r instanceof Figure", r instanceof Figure); // true
30 console.log(r.width);
31 console.log(r.height);
32 console.log(r.area); // 200
33
34 try {
35   r.area = 300; // Error: Area is read-only
36 } catch (e) {
37   console.log(e.message);
38 }
39
40 class Queue {
41   #last = null;
42   #first = null;
43   constructor() { }
44   enqueue(elem) {
45     if (this.#first === null) {
46       const c = { e: elem, next: null };
47       this.#first = c
48       this.#last = c
49     } else {
50       const c = { e: elem, next: null };
51       this.#last.next = c;
52       this.#last = c;
```

```
53     }
54 }
55 dequeue() {
56     if (this.#first === null) {
57         return null;
58     } else {
59         const c = this.#first;
60         this.#first = c.next;
61         return c.e;
62     }
63 }
64 isEmpty() {
65     return this.#first === null;
66 }
67 }
```

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 = false;
11           scriptElement.style.display = 'block';
12           scriptElement.style.whiteSpace = 'preserve';
13           scriptElement.style.padding = '1em';
14           scriptElement.style.backgroundColor = 'yellow';
15         }
16       }
17     </script>
18   </head>
19   <body>
20     <h1>DOM Manipulation with JavaScript</h1>
21     <p id="demo">This is a paragraph.</p>
22     <button type="button"
23       onclick="
24         document.getElementById('demo').style.color = 'red';
25         makeScriptsEditable();">
26       Magic!
27     </button>
28
29     <script>
30       const demoElement = document.getElementById('demo');
31       demoElement.addEventListener(
32         'mouseover',
33         () => demoElement.style.color = 'green'
34       );
35       demoElement.addEventListener(
36         'mouseout',
37         () => demoElement.style.color = 'unset'
38       );
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
49   </body>
50 </html>
```

Interaktion mit Server

```
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
9
10    <script>
11      const box = document.getElementById('box');
12      let color = 0;
13      const setColor = () => {
14        color = (color + 1) % 512 ;
15        let rgb = color
16        if (rgb > 255) rgb = 256-(rgb-255);
17        // console.log(rgb);
18        document.body.style.backgroundColor =
19          `rgb(${rgb}, ${rgb}, ${rgb})`;
20      };
21      setInterval(setColor,10); // the function setColor is called every 10ms
22
23      function getUsers() {
24        fetch('http://127.0.0.1:4080/users')
25          .then(response => response.json())
26          .then(users => {
27            const usersElement = document.getElementById('users');
28            usersElement.innerText = JSON.stringify(users);
29          });
30      }
31    </script>
32
33    <div id=users> </div>
34    <button onclick="getUsers()">Get Users</button>
35  </body>
36 </html>
```

Referenzen

- [HTML DOM API](#)