Webprogrammierung mit JavaScript

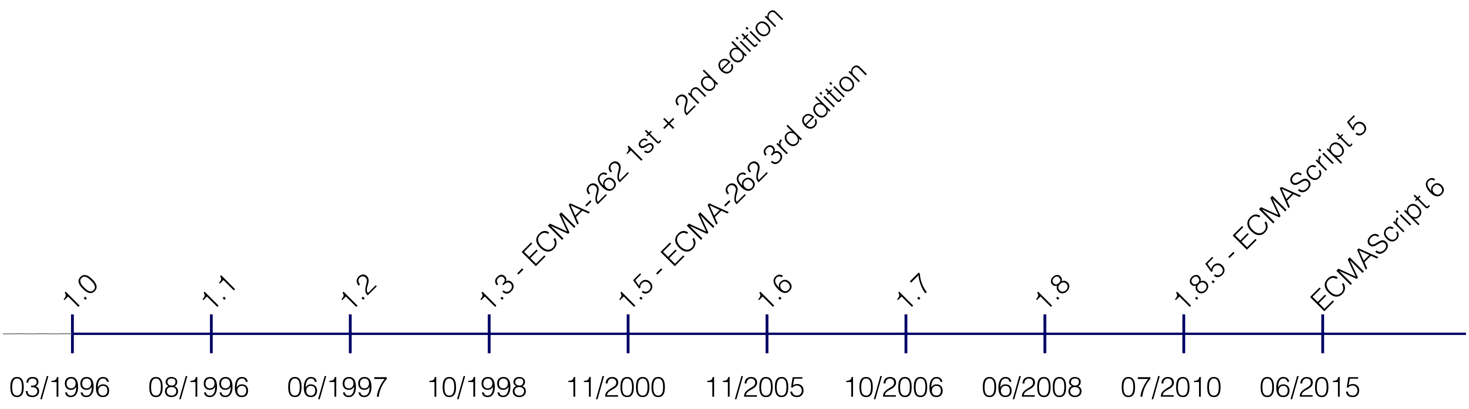
JavaScript und die Webprogrammierung

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw-mannheim.de, Raum 149B
Version: 1.0

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



1.0 – 03/1996

1.1 – 08/1996

1.2 – 06/1997

1.3 - ECMA-262 1st + 2nd edition – 10/1998

1.5 - ECMA-262 3rd edition – 11/2000

1.6 – 11/2005

1.7 – 10/2006

1.8 – 06/2008

1.8.5 - ECMAScript 5 – 07/2010

ECMAScript 6 – 06/2015

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

Datentypen

```
let i = 1;      // double-precision 64-bit binary IEEE 754 value
let f = 1.0;    // double-precision 64-bit binary IEEE 754 value
console.log(
    Number.MIN_VALUE,
    Number.MIN_SAFE_INTEGER,
    Number.MAX_SAFE_INTEGER,
    Number.MAX_VALUE);
let ib = 1n;    // Number.MAX_SAFE_INTEGER 9007199254740991n
console.log(100n === BigInt(100));
let x = NaN;
let y = Infinity;
let z = -Infinity;

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

// we have the standard operators: +, - , *, /, %, ++, --, **
// and the bitwise operators: &, |, ^, ~, <<, >>, >>>
console.log("i++ ", i++); // 1 oder 2?
console.log("++i ", ++i); // 2 oder 3?
console.log("2 ** 4 ", 2 ** 4);
console.log("7 % 3 ", 7 % 3);


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

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


let anonymousObj = {
    i: 1,
    u: { j: 2, v: { k: 3 } },
    toString: function () { return "anonymousObj"; }
};

// 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.toString;
console.log("anonymousObj.toString() [original]", anonymousObj.toString());
console.log("anonymousObj.u?.v.k", anonymousObj.u?.v.k); // Chain-Operator
console.log("anonymousObj.u.v?.k", anonymousObj.u.v?.k);
console.log("anonymousObj.u.q?.k", anonymousObj.u.q?.k);
console.log("anonymousObj.p?.v.k", anonymousObj.p?.v.k);
```

```javascript

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

let $a = [1];

let emptyObject = null;

let func = function () { return "Hello World"; };
console.log(func, func());

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 obj1Values: ", sym1 in obj1Values);
let obj2Values = { [sym1]: "value1", [sym2]: "value2" };
console.log("sym1 in obj2Values: ", sym1 in obj2Values);
console.log(obj1Values, " vs. ", obj2Values);

let u = undefined;


// We have the standard logical operators: &&, ||, ! and also ??

/* Operator Madness */
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);

// 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);

// Nützliche Zuweisungen

anonymousObj.name ||= "Max Mustermann"
```

Vergleich von Werten

```javascript
/* requires node.js */
"use strict";
const Queue = require("./Queue");

const messages = new Queue();

function log(message, ...args) {
  messages.enqueue([message]);
  messages.enqueue(args);
}

// Gleichheit       ==      // mit Typumwandlung (auch bei <, >, <=, >=)
// Ungleichheit     !==
// strikt gleich    ===     // ohne Typumwandlung
// strikt ungleich  !===

log('1 == "1": ', 1 == "1");
log('1 === "1": ', 1 === "1");
log("1.0 == 1: ", 1 == 1.0);
log("1 === 1n: ", 1 === 1n);

log("asdf" === "as" + "df");

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

let firstJohn = { person: "John" };
let secondJohn = { person: "John" };
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);
```

```javascript

let sym1 = Symbol("1"); // a unique and immutable primitive value
log(sym1, sym1, "===", sym1 === sym1); // true
let sym2 = Symbol("1");
let objValues = { sym1: "value1", sym2: "value2" };
log(objValues);
let obj2Values = { [sym1]: "value1", [sym2]: "value2" };
log(objValues, " === ", obj2Values, " vs. ", objValues === obj2Values);
let obj1Value = { [sym1]: "value1", [sym1]: "value2" };
log(obj2Values, " vs. ", obj1Value);
log(sym1, sym2, "===", sym1 === sym2); // false
log(sym1, sym2, "==", sym1 == sym2); // false
log(Symbol.for("1"), sym1, "==", Symbol.for("1") === sym1);

{
  const obj = {
    name: "John",
    age: 30,
    city: "Berlin",
  };
  log("\nTyptests und Feststellung des Typs:");
  log("typeof obj", typeof obj);
  log("obj instanceof Object", obj instanceof Object);
  log("obj instanceof Array", obj instanceof Array);
}
{
  const obj = { a: "lkj" };
  const obj2 = Object.create(obj);
  log(obj2 instanceof obj.constructor);
}

log("\n?-Operator and Truthy and Falsy Values:");
log('""', "" ? "is truthy" : "is falsy");
log("f()", (() => {}) ? "is truthy" : "is falsy");
log("Array ", Array ? "is truthy" : "is falsy");
log("obj ", obj ? "is truthy" : "is falsy");
log("undefined ", undefined ? "is truthy" : "is falsy");
log("null ", null ? "is truthy" : "is falsy");
log("0", 0 ? "is truthy" : "is falsy");
log("1", 1 ? "is truthy" : "is falsy");

process.stdin.on("data", () => {
  const args = messages.dequeue();
  for (const arg of args) {
    process.stdout.write(String(arg));
    process.stdout.write(" ");
  }
  if (messages.isEmpty()) {
    process.exit();
  }
});
```

Bedingungen und Schleifen

```javascript
'use strict';

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

console.log("\If-elseif-else:");
if (arr.length == 7) {
    console.log("arr.length == 7");
} else if (arr.length < 7) {
    console.log("arr.length < 7");
} else {
    console.log("arr.length > 7");
}

console.log("\nSwitch:");
switch (arr.length) {
    case 7:
        console.log("arr.length == 7");
        break;
    case 6:
        console.log("arr.length == 6");
        break;
    default:
        console.log("arr.length != 6 and != 7");
}

switch ("foo") {
    case "bar":
        console.log("it's bar");
        break;
    case "foo":
        console.log("it's foo");
        break;
    default:
        console.log("not foo, not bar");
}

switch (1) { // Vergleich auf strikte Gleichheit (===)
    case "1":
        console.log("string(1)");
        break;
    case 1:
        console.log("number(1)");
        break;
}



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

console.log("\nBreak with label:");
outer: for (let i = 0; i < arr.length; i++) {
    for (let j = 0 ; j < i; j++) {
        if (j == 3) break outer;
        console.log(arr[i], arr[j]);
    }
}

console.log("\nin (properties of Arrays):");
for (const key in arr) {
    console.log(key, arr[key]);
}

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

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

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

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

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


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

```

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


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

Functions

```javascript
// The last examples require that "use strict"; is not enabled!

// the function (see below) is hoisted, so it can be called before it is defined
hello("Michael");

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

// helloExpr(); // the variable declaration is hoisted, but not the definition!
// So it cannot be called here!
var helloExpr = function () {
  console.log("expr: Hello World!");
};

// Arrow Functions
const times3 = (x) => x * 3;
console.log("times3(5)", times3(5)); // 15
const helloArrow = () => console.log("arrow: Hello World!");
const helloBigArrow = () => {
  const s = "Hello World!";
  console.log("arrow: " + s);
  return s;
};

console.log("Hello World!");
helloExpr();
helloArrow();

var helloXXX = function helloXXX() {
  // Function Expression with (useless) Name
  // console.log(arguments); // arguments is an array-like object
  console.log(`Hello: `, ...arguments);
};
helloXXX("Michael", "John", "Jane");

function sum(...args) {
  // rest parameter
  console.log("args: " + args);
  process.stdout.write("...args: ");
  console.log(...args); // we use the spread operator here
  return args.reduce((a, b) => a + b, 0); // function nesting
}
console.log(sum(1, 2, 3, 4, 5)); // 15
console.log(sum());

/* Generator Functions */
function* fib() {
  // generator
  let a = 0,
    b = 1;
  while (true) {
    yield a;
    [a, b] = [b, a + b];
  }
}
const fibGen = fib();
console.log(fibGen.next().value); // 0
console.log(fibGen.next().value); // 1
console.log(fibGen.next().value); // 1
console.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 ... */

/** Partial function application */

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

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

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

function f(c) {
  const that = this;

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

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

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

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

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

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

```

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

console.log("Done.");
```

Variables

```javascript
`use strict`;

// scope is limited to the enclosing function;
// the definition is hoisted (initialized with undefined);
// in modern JS, use let or const instead of var!
var x = "x";

// scope is limited to the enclosing block;
// reference before definition throws an error
let y = "y";

// scope is limited to the enclosing block
const z = "z";


function sumIfDefined(a, b) {
    if (parseInt(a)) {
        var result = parseInt(a); // don't do this in your real code!
    } else {
        result = 0;
    }
    const bVal = parseFloat(b);
    if (bVal) {
        result  += bVal
    }
    return result;
}

console.log(sumIfDefined()); // 0
console.log(sumIfDefined(1)); // 1
console.log(sumIfDefined(1, 2)); // 3
console.log(sumIfDefined(1, "2")); // 3
console.log(sumIfDefined(undefined, "2")); // 2


function global_x() {
    console.log(x,y);

    // const y = ''; // => the previous line throws an error because y is not defined

    console.log(x, y, z); // 1 2 3
}

function local_var_x() {
    console.log(x);
    // console.log(y); // y is not defined

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

    console.log(x, y, z); // 1 2 3
}


console.log("Start:", x, y, z); // 0 0 0
global_x();
local_var_x();


console.log("Last:", x, y, z); // 0 0 0
```

Destructuring

```
let [a,b] = [1,2,3,4]; // array destructuring
console.log(a,b); // 1


let {a : x, b : y} = {a: "a", b: "b"};  // object destructuring
console.log(x,y); // 1
let {a : u, b : v, ...w} = {a: "+", b: "-", c :"*", d:"/"};  // object destructuring
console.log(u,v,w); // + - {c: "*", d: "/"}

let {k1 , k2} = {a: "a", b: "b"};  // object destructuring
console.log(k1,k2); // undefined undefined // k1 and k2 are not defined in the object
```

JSON

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

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

Regular Expressions

- 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,...

```
{
    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"));
}
```

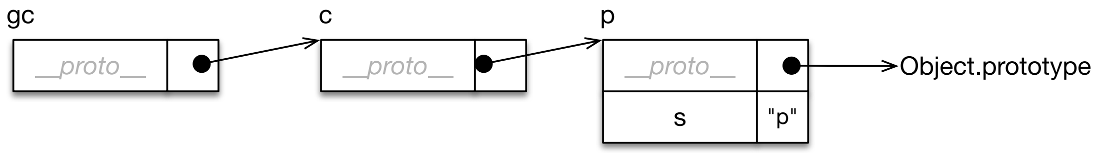
Alles ist ein Objekt

- `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)

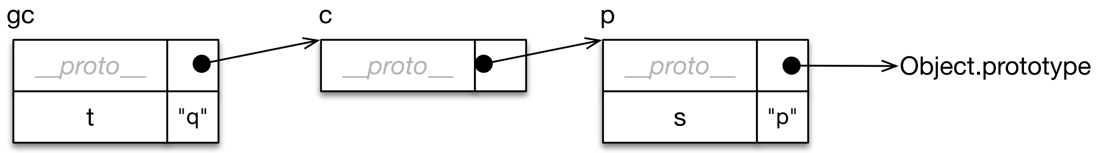
```javascript
//"use strict";

function counter () {
    // console.log(this === globalThis); // true
    if(this.count)  // this is the global object if we don't use strict mode
        this.count ++;
    else {
        this.count = 1;
    }

    return this.count;
}

const counterExpr = function () {
    if(this.count)
        this.count ++;
    else {

        this.count = 1;
    }

    return this.count;
}

const counterArrow = () => {
    console.log(this);
    console.log(this === globalThis);
    this.count = this.count ? this.count + 1 : 1;
    return this.count;
}

console.log("\nCounter");
console.log(counter()); // 1
console.log(counter()); // 2
console.log(`Counter (${globalThis.count})`);

console.log("\nCounterExpression");
console.log(counterExpr()); // 3
console.log(counterExpr()); // 4

console.log("\nCounter");
const obj = {};
console.log(counter.apply(obj)); // 1 - we set a new "this" object!
console.log(counterExpr.apply(obj)); // 2

console.log(`\nCounterArrow (${this.count})`);
console.log(counterArrow.apply(obj)); // 1
console.log(counterArrow.apply(undefined)); // 2
console.log(counterArrow.apply()); // 3
console.log(counterArrow.apply(obj)); // 4
console.log(counterArrow.apply({})); // 5

console.log("\nCounter (global)");
console.log(counter());
console.log(counterExpr());
```

Prototype basierte Vererbung

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



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

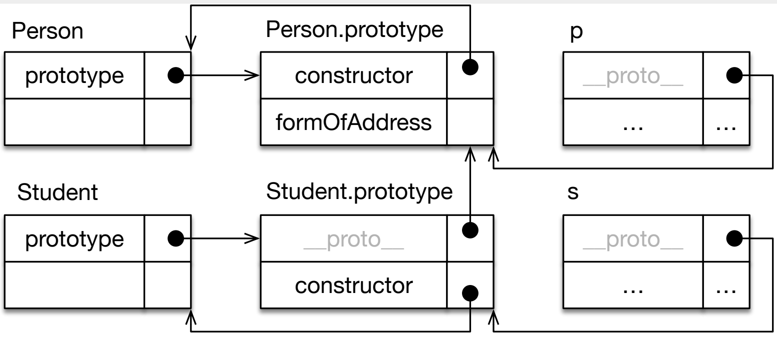


Pseudoclassical Inheritance

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

const aStudent = new Student("Emilia Galotti", "Mrs.", 1224441, 'emilia@galotti.com');
```

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(…);
```



Prototype basierte Vererbung

```javascript
console.log({}.__proto__)
console.log(Array.prototype)
console.log(Array.prototype.__proto__)
console.log(Array.prototype)

try {
    let a = [1];
    console.log(a.fold());
} catch (error) {
    console.log("error: ", error.message)
}

// THIS IS NOT RECOMMENDED!
// IF ECMAScript EVENTUALLY ADDS THIS FUNCTIONALITY,
// 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;
    }
}

let a = [1,10,100,1000];
console.log(a.fold((a,b) => a + b));


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

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 = true;
                    scriptElement.style.display = 'block';
                    scriptElement.style.whiteSpace = 'preserve';
                    scriptElement.style.padding = '1em';
                    scriptElement.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();">
            Magic!
        </button>

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

Interaktion mit Server

```
<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

            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);
                    });
            }
        </script>

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

Referenzen

- HTML DOM API