# 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 auf Folien melden:**

https://github.com/Delors/delors.github.io/issues

# Historie



| 1.0 | 1.1 | 1.2 | 1.3 - ECMA-262 1st + 2nd edition | 1.5 - ECMA-262 3rd edition | 1.6 | 1.7 | 1.8 | 1.8.5 - ECMAScript 5 | ECMAScript 6 |
|-----|-----|-----|------|------|-----|-----|-----|------|------|
| 03/1996 | 08/1996 | 06/1997 | 10/1998 | 11/2000 | 11/2005 | 10/2006 | 06/2008 | 07/2010 | 06/2015 |
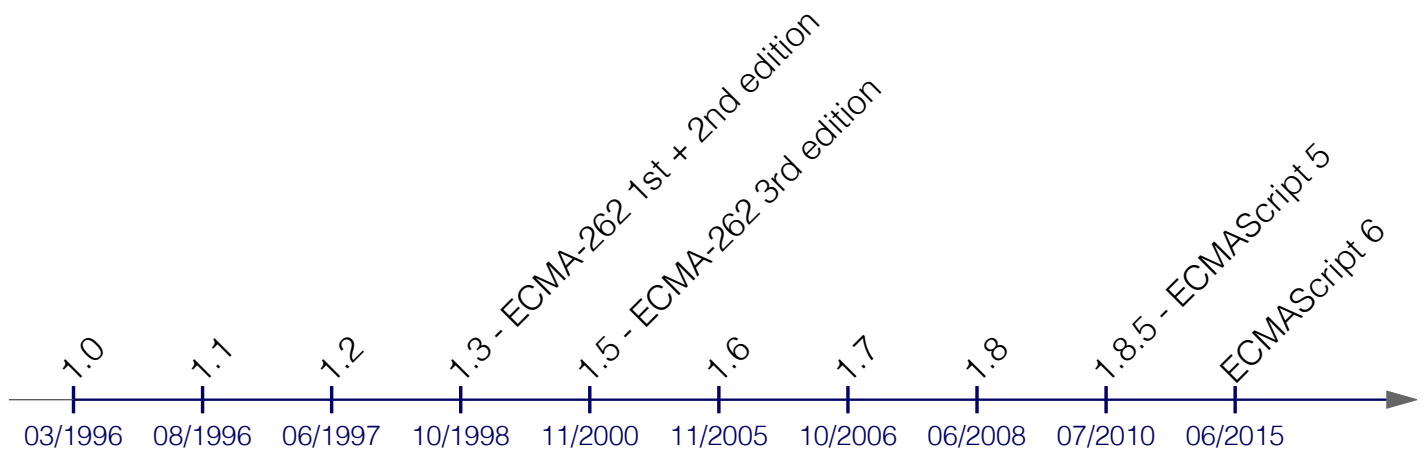
Seit 2016 gibt es jährliche Updates (ECMAScript 2016, 2017, 2018, 2019, 2020, 2021, 2022, ...)

# 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

```javascript
 1 let i = 1;        // double-precision 64-bit binary IEEE 754 value
 2 let f = 1.0;      // double-precision 64-bit binary IEEE 754 value
 3 console.log(
 4     Number.MIN_VALUE,
 5     Number.MIN_SAFE_INTEGER,
 6     Number.MAX_SAFE_INTEGER,
 7     Number.MAX_VALUE);
 8 let ib = 1n;      // Number.MAX_SAFE_INTEGER 9007199254740991n
 9 console.log(100n === BigInt(100));
10 let x = NaN;
11 let y = Infinity;
12 let z = -Infinity;
13
14 let b = true; // oder false
15 console.log("Boolean(undefined)", Boolean(undefined));
16
17 // we have the standard operators: +, - , *, /, %, ++, --, **
18 // and the bitwise operators: &, |, ^, ~, <<, >>, >>>
19 console.log("i++ ", i++); // 1 oder 2?
20 console.log("++i ", ++i); // 2 oder 3?
21 console.log("2 ** 4 ", 2 ** 4);
22 console.log("7 % 3 ", 7 % 3);
23
24
25 let _s = "42";
26 console.log('Die Antwort ist ' + _s); // Template literals (Template strings)
27 console.log(`Die Antwort ist ${_s}.`); // Template literals (Template strings)
28 console.log(`
29     Die Antwort mag ${_s} sein,
30     aber was ist die Frage?`);
31
32 console.log(String(42)); // "42"
33
34
35 let anonymousObj = {
36     i: 1,
37     u: { j: 2, v: { k: 3 } },
38     toString: function () { return "anonymousObj"; }
39 };
40
41 // Zugriff auf die Eigenschaften eines Objekts
42 anonymousObj.j = 2;      // mittels Bezeichner ("j") (eng. Identifier)
43 anonymousObj["j"] = 4;   // mittels String ("j")
44 anonymousObj["k"] = 3;
45 console.log("anonymousObj", anonymousObj);
46 console.log("anonymousObj.toString()", anonymousObj.toString());
47 delete anonymousObj.toString;
48 console.log("anonymousObj.toString() [original]", anonymousObj.toString());
49 console.log("anonymousObj.u?.v.k", anonymousObj.u?.v.k); // Chain-Operator
50 console.log("anonymousObj.u.v?.k", anonymousObj.u.v?.k);
51 console.log("anonymousObj.u.q?.k", anonymousObj.u.q?.k);
52 console.log("anonymousObj.p?.v.k", anonymousObj.p?.v.k);
```

```javascript
53
54  let date = new Date("8.6.2024") // ACHTUNG: Locale-Settings
55  console.log(date);
56
57  let $a = [1];
58
59  let emptyObject = null;
60
61  let func = function () { return "Hello World"; };
62  console.log(func, func());
63
64  let sym1 = Symbol("1"); // a unique and immutable primitive value
65  let sym2 = Symbol("1");
66  let obj1Values = { sym1: "value1", sym2: "value2" };
67  console.log(obj1Values);
68  console.log("sym1 in obj1Values: ", sym1 in obj1Values);
69  let obj2Values = { [sym1]: "value1", [sym2]: "value2" };
70  console.log("sym1 in obj2Values: ", sym1 in obj2Values);
71  console.log(obj1Values, " vs. ", obj2Values);
72
73  let u = undefined;
74
75
76  // We have the standard logical operators: &&, ||, ! and also ??
77
78  /* Operator Madness */
79  console.log("1 && \"1\": ", 1 && '1');
80  console.log("null && \"1\": ", null && '1');
81  console.log("null && true: ", null && true);
82  console.log("true && null: ", true && null);
83  console.log("null && false: ", null && false);
84  console.log("{} && true: ", {} && true);
85
86  // nullish coalescing operator (??) (vergleichbar zu ||)
87  console.log("1 ?? \"1\": ", 1 ?? '1');
88  console.log("null ?? \"1\": ", null ?? '1');
89  console.log("null ?? true: ", null ?? true);
90  console.log("true ?? null: ", true ?? null);
91  console.log("null ?? false: ", null ?? false);
92  console.log("{} ?? true: ", {} ?? true);
93
94  // Nützliche Zuweisungen
95
96  anonymousObj.name ||= "Max Mustermann"
```

# Vergleich von Werten

```javascript
'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
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(objValues);
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
67 log("\nTyptests und Feststellung des Typs:");
68 log("typeof obj", typeof obj);
69 log("obj instanceof Object", obj instanceof Object);
70 log("obj instanceof Array", obj instanceof Array)
71
72
73 log("\n?-Operator 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 ", 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");
82
83
84
85
86 process.stdin.on('data', () => {
87     const args = messages.dequeue();
88     for (const arg of args) {
89         process.stdout.write(String(arg));
90         process.stdout.write(' ');
91     }
92     if (messages.isEmpty()) {
93         process.exit();
94     }
95 });
```

# Bedingungen und Schleifen

```javascript
1  'use strict';
2
3  const arr = [1, 3, 4, 7, 11, 18, 29];
4
5  console.log("\If-elseif-else:");
6  if (arr.length == 7) {
7      console.log("arr.length == 7");
8  } else if (arr.length < 7) {
9      console.log("arr.length < 7");
10 } else {
11     console.log("arr.length > 7");
12 }
13
14 console.log("\nSwitch:");
15 switch (arr.length) {
16     case 7:
17         console.log("arr.length == 7");
18         break;
19     case 6:
20         console.log("arr.length == 6");
21         break;
22     default:
23         console.log("arr.length != 6 and != 7");
24 }
25
26 switch ("foo") {
27     case "bar":
28         console.log("it's bar");
29         break;
30     case "foo":
31         console.log("it's foo");
32         break;
33     default:
34         console.log("not foo, not bar");
35 }
36
37 switch (1) { // Vergleich auf strikte Gleichheit (===)
38     case "1":
39         console.log("string(1)");
40         break;
41     case 1:
42         console.log("number(1)");
43         break;
44 }
45
46
47
48 console.log("\nContinue:");
49 for (let i = 0; i < arr.length; i++) {
50     const v = arr[i];
51     if (v % 2 == 0) continue;
52     console.log(v);
```

```javascript
53 }
54
55 console.log("\nBreak 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         console.log(arr[i], arr[j]);
60     }
61 }
62
63 console.log("\nin (properties of Arrays):");
64 for (const key in arr) {
65     console.log(key, arr[key]);
66 }
67
68 console.log("\nof (values of Arrays):");
69 for (const value of arr) {
70     console.log(value);
71 }
72
73 console.log("\nArray and Objects - instanceof:");
74 console.log("arr instanceof Object", arr instanceof Object);
75 console.log("arr instanceof Array", arr instanceof Array);
76
77 const obj = {
78     name: "John",
79     age: 30,
80     city: "Berlin"
81 };
82
83 console.log("\nin (properties of Objects):");
84 for (const key in obj) {
85     console.log(key, obj[key]);
86 }
87
88 /* TypeError: obj is not iterable
89 for (const value of obj) {
90     console.log(value);
91 }
92 */
93
94
95 {
96     console.log("\nIteration über Iterables (here: Map):");
97     const m = new Map();
98     m.set("name", "Elisabeth");
99     m.set("alter", 50);
100    console.log("Properties of m: ");
101    for (const key in m) {
102        console.log(key, m[key]);
103    }
104    console.log("Values of m: ");
105    for (const [key, value] of m) {
106        console.log(key, value);
107    }
108 }
```

```
109
110 {
111     console.log("\nWhile Loop: ");
112     let c = 0;
113     while (c < arr.length) {
114         const v = arr[c]
115         if (v > 10) break;
116         console.log(v);
117         c++;
118     }
119 }
120
121
122 {
123     console.log("\nDo-While Loop: ");
124     let c = 0
125     do {
126         console.log(arr[c]);
127         c++;
128     } while (c < arr.length);
129 }
```

# Functions

```javascript
1  // the function (see below) is hoisted, so it can be called before it is defined
2  hello('Michael');
3
4  function hello(person = "World") { // argument with default value
5      console.log(`fun: Hello ${person}!`);
6  }
7
8  // helloExpr(); // the variable declaration is hoisted, but not the definition!
9                  // So it cannot be called here!
10 var helloExpr = function() {
11     console.log('expr: Hello World!');
12 }
13
14 // Arrow Functions
15 const times3 = x => x * 3;
16 console.log("times3(5)", times3(5)); // 15
17 const helloArrow = () => console.log('arrow: Hello World!');
18 const helloBigArrow = () =>  {
19     const s = "Hello World!";
20     console.log('arrow: '+s);
21     return s;
22 }
23
24 console.log('Hello World!');
25 helloExpr();
26 helloArrow();
27
28 var helloXXX = function helloXXX () { // Function Expression with (useless) Name
29     // console.log(arguments); // arguments is an array-like object
30     console.log(`Hello: `,...arguments);
31 }
32 helloXXX('Michael', 'John', 'Jane');
33
34 function sum(...args) { // rest parameter
35     console.log("args: " + args);
36     process.stdout.write("...args: ");
37     console.log(...args); // we use the spread operator here
38     return args.reduce((a, b) => a + b, 0); // function nesting
39 }
40 console.log(sum(1, 2, 3, 4, 5)); // 15
41 console.log(sum());
42
43
44 function* fib() { // generator
45     let a = 0, b = 1;
46     while(true) {
47         yield a;
48         [a, b] = [b, a + b];
49     }
50 }
51 const fibGen = fib();
```

```
52 console.log(fibGen.next().value); // 0
53 console.log(fibGen.next().value); // 1
54 console.log(fibGen.next().value); // 1
55 console.log(fibGen.next().value); // 2
56 /* Will cause an infinite loop: for (const i of fib()) console.log(i);
57     // 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 ... */
58
59 console.log("Done.");
```

# Variables

```javascript
 1 `use strict`;
 2
 3 // scope is limited to the enclosing function;
 4 // the definition is hoisted (initialized with undefined);
 5 // in modern JS, use let or const instead of var!
 6 var x = "x";
 7
 8 // scope is limited to the enclosing block;
 9 // reference before definition throws an error
10 let y = "y";
11
12 // scope is limited to the enclosing block
13 const z = "z";
14
15
16 function sumIfDefined(a, b) {
17     if (parseInt(a)) {
18         var result = parseInt(a); // don't do this in your real code!
19     } else {
20         result = 0;
21     }
22     const bVal = parseFloat(b);
23     if (bVal) {
24         result  += bVal
25     }
26     return result;
27 }
28
29 console.log(sumIfDefined()); // 0
30 console.log(sumIfDefined(1)); // 1
31 console.log(sumIfDefined(1, 2)); // 3
32 console.log(sumIfDefined(1, "2")); // 3
33 console.log(sumIfDefined(undefined, "2")); // 2
34
35
36 function global_x() {
37     console.log(x,y);
38
39     // const y = ''; // => the previous line throws an error because y is not defined
40
41     console.log(x, y, z); // 1 2 3
42 }
43
44 function local_var_x() {
45     console.log(x);
46     // console.log(y); // y is not defined
47
48     var x = 1; // the declaration of var is hoisted, but not the initialization
49     let y = 2;
50     const z = 3;
51
```

```javascript
        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

```javascript
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

```javascript
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,...

```
 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)

```javascript
 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
19          this.count = 1;
20      }
21
22      return this.count;
23  }
24
25  const counterArrow = () => {
26      console.log(this);
27      console.log(this === globalThis);
28      this.count = this.count ? this.count + 1 : 1;
29      return this.count;
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());
```
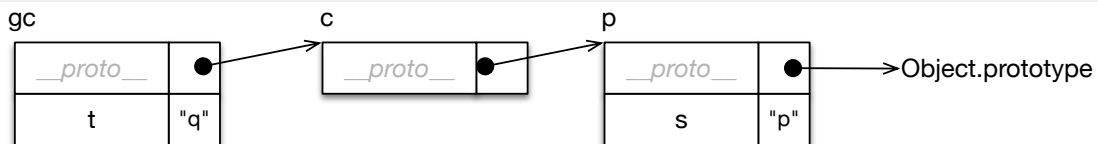
# Prototype basierte Vererbung

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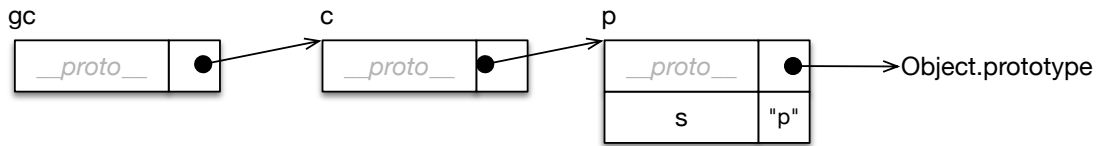


```
1 const p = { s : "p" };
2 const c = Object.create(p);
3 const gc = Object.create(c);
4 gc.t = "q";
```



## 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');
```

## 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(…);
```

| Person | | Person.prototype | | p | |
|---|---|---|---|---|---|
| prototype | ● | constructor | ● | __proto__ | ● |
| | | formOfAddress | | … | … |

| Student | | Student.prototype | | s | |
|---|---|---|---|---|---|
| prototype | ● | __proto__ | ● | __proto__ | ● |
| | | constructor | ● | … | … |

4
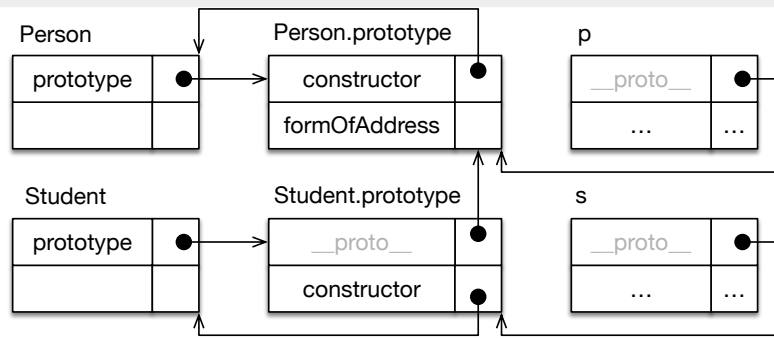
# Prototype basierte Vererbung

```javascript
1  console.log({}.__proto__)
2  console.log(Array.prototype)
3  console.log(Array.prototype.__proto__)
4  console.log(Array.prototype)
5
6  try {
7      let a = [1];
8      console.log(a.fold());
9  } catch (error) {
10     console.log("error: ", error.message)
11 }
12
13 // THIS IS NOT RECOMMENDED!
14 // IF ECMAScript EVENTUALLY ADDS THIS FUNCTIONALITY,
15 // IT MAY CAUSE HAVOC.
16 Array.prototype.fold = function (f) {
17     if (this.length === 0) {
18         throw new Error("array is empty");
19     } else if (this.length === 1) {
20         return this[0];
21     } else {
22         let result = this[0];
23         for (let i = 1; i < this.length; i++) {
24             result = f(result, this[i]);
25         }
26         return result;
27     }
28 }
29
30 let a = [1,10,100,1000];
31 console.log(a.fold((a,b) => a + b));
32
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

```javascript
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

```html
1 <html lang=en>
2     <head>
3         <meta charset="utf-8">
4         <meta name="viewport" content="width=device-width, initial-scale=1.0">
5         <title>DOM Manipulation with JavaScript</title>
6         <script>
7             function makeScriptsEditable() {
8                 const scripts = document.getElementsByTagName('script')
9                 for (const scriptElement of scripts) {
10                    scriptElement.contentEditable = true;
11                    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

```html
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