

# 04b JavaScript II

Multimedia Engineering (PS siehe 36609b)

Nils Hellwig, M.Sc.

Lehrstuhl für Medieninformatik

**FAKULTÄT FÜR INFORMATIK UND DATA SCIENCE**



Universität Regensburg

# Überblick Themen Nachtrag

- Klassen
- Export von Variablen, Klassen, Objekte
- Globale Objekte
- Daten im Browser persistieren mit LocalStorage, SessionStorage und Cookies
- Promises
- Sortieren mit `.sort()`
- Events

## JavaScript Classes

- Klassen in JavaScript sind wie Blaupausen oder Vorlagen, die uns helfen, ähnliche Objekte mit **gemeinsamen Eigenschaften und Methoden** zu erstellen

## JavaScript Classes

- Um ein Objekt zu erstellen, müssen wir eine Instanz der Klasse erstellen
- Eine Klasse kann Vererbung unterstützen, was bedeutet, dass sie Eigenschaften und Methoden von einer anderen Klasse erben kann.

## JavaScript Classes

- Man verwendet das Schlüsselwort `class`, um eine Klasse zu erstellen.

Beispiel:

```
class Car {  
  constructor(name, year) {  
    this.name = name;  
    this.year = year;  
  }  
}
```

```
let myCar1 = new Car("Ford", 2014); // Car Object: {name: "Ford", year: 2014}  
let myCar2 = new Car("Audi", 2019); // Car Object: {name: "Audi", year: 2019}
```

Im obigen Beispiel wird die Klasse `Car` verwendet, um zwei `Car`-Objekte zu erstellen.

## JavaScript Classes: Konstruktor

```
class Car {  
  constructor(name, year) {  
    this.name = name;  
    this.year = year;  
  }  
}
```

- Die Konstruktormethode wird automatisch aufgerufen, wenn ein neues Objekt erstellt wird
- Es wird verwendet, um Objekteigenschaften zu initialisieren
- Wenn Sie keine Konstruktormethode definieren, fügt JavaScript eine leere Konstruktormethode hinzu

## JavaScript Classes: Methoden

```
class Car {  
  constructor(name, year) {  
    this.name = name;  
    this.year = year;  
  }  
  getCarInformation() {  
    return "A " + this.name + " from " + this.year;  
  }  
}  
  
let myCar = new Car("Ford", 2014);  
document.getElementById("demo").innerHTML = myCar.getCarInformation();
```

## JavaScript Classes: Vererbung

- Klassen in JavaScript können von anderen Klassen erben.
- Die erbende Klasse erhält alle Eigenschaften und Methoden der vererbenden Klasse (auch Basisklasse genannt)
- Die erbende Klasse kann die Methoden und Eigenschaften der Basisklasse überschreiben oder neue hinzufügen.



# JavaScript Classes: Vererbung

```
class Tier {
  constructor(name) {
    this.name = name;
  }

  sprechen() {
    console.log("Ein Tier spricht.");
  }
}

class Hund extends Tier {
  sprechen() {
    console.log("Der Hund bellt.");
  }
}

const fido = new Hund("Fido");
console.log(fido.name); // Output: "Fido"
fido.sprechen(); // Output: "Der Hund bellt."
```

# Export und Import in JavaScript

- Mit `export` können Funktionen, Objekte oder Klassen aus einer Datei bereitgestellt werden.
- Mit `import` können diese in anderen Dateien geladen werden.
- Es gibt zwei Arten von Exporten: **Named Exports** und **Default Export**.

- **Named Exports:**

```
// toolbox.js
export function greet() { ... }
export const PI = 3.14;
export class Person { ... }

// script.js
import { greet, PI, Person } from './toolbox.js';
```

Beim Import muss der Name exakt angegeben werden

- **Default Exports:**

```
// toolbox.js
export default function greet() { ... }

// script.js
import greet from './toolbox.js';
```

Jede Datei kann genau einen Default Export haben.

## Globale Objekte/Methoden/Klassen

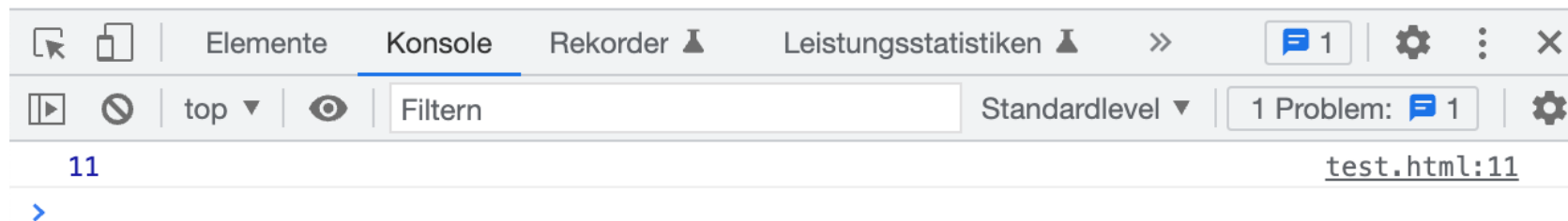
- Globale Objekte sind vordefinierte Objekte, die im globalen Namensraum **überall** verfügbar sind.
- Sie bieten zugängliche Funktionen, Konstanten und Eigenschaften, **ohne dass man sie explizit importieren muss.**
- Wieso gibt es globale Objekte?: JavaScript stellt **häufig genutzte Funktionalitäten** standardisiert bereit.
- Beispiele (Klassen und Objekte ohne Konstruktor): window(.document), Math, JSON, console, Date, RegExp, Promise, LocalStorage
- Liste: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects)

## Globale Objekte: localStorage, sessionStorage und Cookies

Merkmal	SessionStorage	LocalStorage	Cookies
<b>Speicherort</b>	Im Browser (Session-bezogen)	Im Browser (persistent)	Im Browser (persistent) und werden mit Requests an Server gesendet
<b>Art der Speicherung</b>	Schlüssel-Wert-Paare (key-value) als String (im Speicher des Browsers)	Schlüssel-Wert-Paare (key-value) als String (im Speicher des Browsers)	Textbasierte Strings im HTTP-Header und Browser-Speicher
<b>Speicherdauer</b>	Bis Browser- oder Tab-Schließung	Persistiert, bis gelöscht	Je nach Ablaufdatum, oft persistent
<b>Speichergröße</b>	Ca. 5–10 MB pro Origin	Ca. 5–10 MB pro Origin	Ca. 4 KB pro Cookie
<b>Automatische Übertragung an Server</b>	Nein	Nein	Ja (bei jedem HTTP-Request)
<b>Zugriff</b>	Nur clientseitig (JavaScript)	Nur clientseitig (JavaScript)	Client- und serverseitig möglich
<b>Typischer Anwendungsfall</b>	Temporäre Daten während Sitzung (z.B. Form-Daten)	Persistente Daten (z.B. Benutzereinstellungen)	Sitzungsverwaltung, Authentifizierung
<b>API</b>	sessionStorage.setItem(), getItem() usw.	localStorage.setItem(), getItem() usw.	document.cookie (stringbasiert, komplexer)
<b>Sicherheit</b>	Nur Same-Origin zugreifbar	Nur Same-Origin zugreifbar	Kann mit Flags wie HttpOnly, Secure geschützt werden

## JavaScript Debugging: `console.log()`

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      console.log(5 + 6);
    </script>
  </body>
</html>
```



## JavaScript Kommentare

- Code nach doppelten Schrägstrichen `//` oder zwischen `/*` und `*/` wird wie ein Kommentar behandelt

```
let x = 5;    // wird ausgeführt  
// x = 6;    wird nicht ausgeführt
```

```
/*  
  Dies ist ein mehrzeiliger Kommentar  
*/
```

## Promises: Warum asynchrone Programmierung?

- JavaScript läuft im Browser (und in Node.js) standardmäßig in **einem einzigen Thread**.
- Es gibt **nur einen Ausführungskontext** zur selben Zeit – blockierende Operationen (z. B. lange Netzwertkanfragen, komplexe Berechnungen) halten die komplette Anwendung auf.
- **Beispiel:** Wenn JavaScript eine zeitaufwändige Aufgabe (z. B. Laden von Daten oder Warten auf eine Antwort) direkt ausführt, kann nichts anderes mehr reagieren – kein Klicken, Scrollen oder Tippen.
- **Lösung:** Statt zu blockieren, kann die Aufgabe asynchron ausgeführt werden – sie läuft im Hintergrund, während der Rest der Anwendung weiter funktioniert.

## Promises: Typische Anwendungsfälle Asynchronität

- **Netzwerkoperationen:** Daten von einer API abrufen (`fetch()`)
- **Zeitverzögerungen:** z. B. Animationen mit `setTimeout()`, Countdown-Funktionen.
- **Dateioperationen (v. a. in Node.js):** Lesen/Schreiben großer Dateien.
- **Benutzerinteraktion:** z.B. Dynamische Ladeindikatoren.



## Promises: Typische Anwendungsfälle Asynchronität

- Promises bieten eine moderne Möglichkeit, solche asynchronen Aufgaben zu verwalten.
- Promises implementieren eine `.then()`-Methode, die eine Funktion als Callback annimmt.
- Zustände:

`pending` – das Versprechen wird gerade bearbeitet

`fulfilled` – das Versprechen wurde erfolgreich eingehalten

`rejected` – es gab einen Fehler, das Versprechen konnte nicht eingehalten werden

- **Beispiel:**

```
loadData()
```

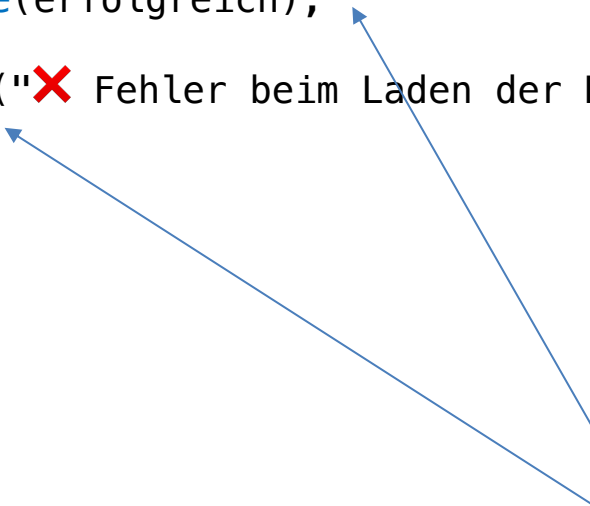
```
.then(result => console.log(result))
```

```
.catch(error => console.error("Error:", error));
```

## Promises: Typische Anwendungsfälle Asynchronität

```
function loadData() {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      const data = Math.random() > 0.5; // 50/50 Erfolg oder Fehler  
      if (erfolgreich) {  
        resolve(erfolgreich);  
      } else {  
        reject("✗ Fehler beim Laden der Daten.");  
      }  
    }, 2000);  
  });  
}
```

```
loadData()  
  .then(result => console.log(result))  
  .catch(error => console.error("Error:",  
    error));
```



## Arrays: Sortieren mit .sort()

- Die Methode `.sort()` sortiert Strings alphabetisch (lexikographisch) nach Unicode-Werten.
- Problem: `.sort()` sortiert Zahlen nicht numerisch, sondern als Strings (Unicode-Vergleich).

```
const fruits = ["Banane", "Apfel", "Orange"];  
fruits.sort();  
console.log(fruits); // ["Apfel", "Banane", "Orange"]
```

```
const numbers = [10, 2, 5, 1];  
numbers.sort();  
console.log(numbers); // [1, 10, 2, 5]
```

## Arrays: Zahlen korrekt sortieren mit Comperator-Funktion

- `.sort()` kann eine Comparator-Funktion als Argument bekommen:

```
numbers.sort((a, b) => a - b);  
console.log(numbers); // [1, 2, 5, 10]
```

Rückgabewert  $< 0$   $\rightarrow$  a kommt vor b

Rückgabewert  $= 0$   $\rightarrow$  Reihenfolge bleibt

Rückgabewert  $> 0$   $\rightarrow$  b kommt vor a

Die `.sort()`-Methode sortiert intern das Array mit einem bestimmten Algorithmus (meist eine Variante von **Quicksort**, **Mergesort** oder **Timsort** – abhängig von der JavaScript-Engine).

## Arrays: Comparator-Funktion auch für Objekte (und Strings)

```
const personen = [
  { name: "Anna", alter: 28 },
  { name: "Bernd", alter: 35 },
  { name: "Clara", alter: 22 },
  { name: "David", alter: 35 }
];

// Sortieren nach Name (alphabetisch)
personen.sort((a, b) => {
  if (a.name < b.name) return -1;
  if (a.name > b.name) return 1;
  return 0; // Namen sind gleich
});
console.log("Nach Name sortiert:", personen);
```

```
/* Ausgabe:
[
  { name: "Anna", alter: 28 },
  { name: "Bernd", alter: 35 },
  { name: "Clara", alter: 22 },
  { name: "David", alter: 35 }
]
*/
```

# Events in JavaScript

- Events treten auf, wenn der Benutzer mit der Webseite interagiert.
- Beispiele:

**click** – Wird ausgelöst, wenn ein Element angeklickt wird.

```
button.addEventListener("click", () => console.log("Button geklickt"));
```

**mouseover** – Wenn der Mauszeiger über ein Element fährt.

**mouseout** – Wenn der Mauszeiger ein Element verlässt.

**keydown** – Wenn eine Taste gedrückt wird (Gedrückte Taste kann mit event.key abgerufen werden).

```
document.addEventListener("keydown", (e) => console.log(e.key));
```

**submit** – Beim Absenden eines Formulars.

**change** – Wenn sich der Wert eines Eingabefeldes ändert.

## Weiterführende Literatur / Quellen

- Literatur: Eloquent JavaScript:  
<https://eloquentjavascript.net/>
- Ausführliches Tutorial zu JavaScript mit anschaulichen Beispielen:  
<https://www.w3schools.com/js/>
- Daten im Browser speichern:  
[https://www.w3schools.com/html/html5\\_webstorage.asp](https://www.w3schools.com/html/html5_webstorage.asp)
- LocalStorage, Leseempfehlung: [https://eloquentjavascript.net/2nd\\_edition/18\\_forms.html](https://eloquentjavascript.net/2nd_edition/18_forms.html)