

Web-basierte Systeme – Übung

01: Einführung in JavaScript, Teil 1

Wintersemester 2025

Arne Vogel, Maxim Ritter von Onciul



Lehrstuhl für Informatik 4
Systemsoftware



Friedrich-Alexander-Universität
Technische Fakultät

Übersicht

JavaScript Überblick

JavaScript Programmierung

Basics

Variablen

Vergleiche

Funktionen

Arrays und Objekte

Scopes

this

Objekte und Klassen

JavaScript und HTML

JavaScript ausprobieren

Ausblick

JavaScript Überblick

■ Interpretierte Skriptsprache

- Ursprünglich für kleinere Programme und Prototyping gedacht

■ Objektorientiert

- Alles ist ein Objekt!

■ Funktional

- Funktionen ohne Seiteneffekte (pure functions)
- Funktionen können Parameter oder Rückgabewerte sein

■ Imperativ

- Schrittweise Abarbeitung von Befehlen
- Schleifen und Verzweigungen

■ Typisierung: schwach, dynamisch, duck

- **Schwach:** Keine Unterscheidung von Datentypen
- **Dynamisch:** Datentypen werden erst zur Laufzeit festgelegt
- **Duck:** Vorhandensein bestimmter Methoden oder Attribute bestimmt Typ eines Objekts

JavaScript: Typische Anwendungsgebiete

- Ursprünglich für Einsatz in **Webbrowsern** entwickelt, neu:
 - Node.js
 - IoT
- Manipulation von Webseiten über DOM (Document Object Model)
- Validierung von Benutzereingaben (Syntax von E-Mail korrekt?)
- Asynchrone Datenübertragung ohne Seite neu zu laden (AJAX)
- Mit Node.js **beliebige Serveranwendungen**

Historischer Rückblick

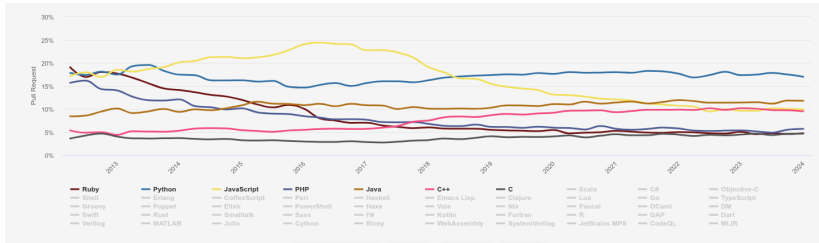
- 1995** Brendan Eich entwickelt eine **Skriptsprache** für den Netscape Navigator. Name: erst Mocha, dann LiveScript, später JavaScript
- 1997** Standardisierung von JavaScript 1.1 der ECMA → **ECMAScript**
- 1998** Standardisierung des **DOM** durch das W3C
- 2005** Aufsatz *Ajax: A New Approach to Web Applications*
- 2009** Einsatz von JavaScript auf Serverseite mit **NodeJS**
- 2011** Browser-zu-Browser Kommunikation in Echtzeit mit **WebRTC**
- 2014** Einführung von **HTML5**, neue APIs für JavaScript
- 2017** **WebAssembly** als Ergänzung zu JavaScript

- **ECMAScript-262**: 1997 Version 1, 2017 Version 8, 2019 Version 9
- `caniuse.com`
- Implementiert von **JavaScript engines**:
 - V8 (Google)
 - NodeJS (V8 + Event Loop + I/O API)
 - SpiderMonkey (Mozilla)
 - Chakra (Microsoft)
 - JavaScriptCore/Nitro (Apple)
 - Carakan (Opera)
 - Nashorn (Oracle)
 - Tamarin (Adobe)
 - kleinere Engines: Duktape, MuJS, ...

JavaScript: eine interpretierte Sprache

- Quellcode von **kompilierten Sprachen** wird von Compilern übersetzt
 - Kompilat direkt von CPU ausführbar
 - Fehler zur Kompilier- und Laufzeit möglich
 - Beispiele: C, C++
- Quellcode von **interpretierten Sprachen** wird von **Interpretern** analysiert und ausgeführt
 - Code wird zeilenweise eingelesen und ausgeführt
 - Instruktionen bereits in Interpreter enthalten
 - Fehler erst zur Laufzeit erkennbar
 - Beispiele: Python, **JavaScript**
- **Java** ist sowohl eine kompilierte als auch ein interpretierte Sprache
 - Kompilierung zu Bytecode
 - Bytecode kann von JVM interpretiert oder kompiliert werden (JIT)
 - Auch moderne **JavaScript engines** unterstützen JIT!

JavaScript Trends



Quelle: <https://madnight.github.io/github/#/pushes/2024/1>

JavaScript Programmierung

- Ähnliche Schreibweise wie die Sprachen aus der **C-Familie** (C, C++, Java, Perl, PHP)
- Anweisungen sollten mit **Semikolons** getrennt werden
 - Syntax schreibt das nicht vor
 - verhindern aber Programmierfehler
- Blöcke werden mit **geschwungenen Klammern** gebildet

```
1 // document.location speichert die aktuelle URL
2 document.location = "https://sys.cs.fau.de";
3
4 // die Methode write (über)schreibt die webseite
5 document.write("hi");
```

Variablen

- Deklaration mit den Keywords **let** und **const**
- **let** definiert eine überschreibbare Variable
- **const** definiert unveränderbare Konstanten

```
1  let a = "Hallo";
2  a = "Salut";
3  a = 1234;    // Auch der Typ von a ist veränderbar
4  a = [1, 2, 3, 4];
5
6  const b = 1;
7  let b = 2;   // SyntaxError: Identifier 'b' has already been declared
8  b = 3;      // TypeError: Assignment to constant variable
9  const c;    // SyntaxError: Missing initializer in const declaration
```

- Es gibt noch **var**, aber das ist stark veraltet und sollte heutzutage nicht mehr benutzt werden!
 - Siehe: <https://stackoverflow.com/a/11444416>

Sonderfälle: const

- `const` funktioniert anders für Arrays und Objekte
- Beide können verändert werden
- Nur die Referenz darf nicht verändert werden

```
1  const numbers = [1, 2];
2  numbers.push(3); // [1, 2, 3]
3  numbers.pop();   // [1, 2]
4  // gibt: TypeError: Assignment to constant variable.
5  numbers = [1, 2, 3, 4];
6
7  const car = { color: "red", speed: "zoom", brand: "VW" };
8  car.model = "Polo";
9  // gibt selben TypeError
10 car = { color: "pink", model: "Multipla", brand: "Fiat" };
11
```

- Es gibt Zahlen, Strings, Booleans, Arrays, Objekte, Funktionen
- Speziellere Typen: Null, Undefined

```
1 let a;  
2 a = 10;           // typeof(a) === "number"  
3 a = 3.141;        // typeof(a) === "number"  
4 a = "ein text";   // typeof(a) === "string"  
5 a = true;         // typeof(a) === "boolean"  
6 a = undefined;    // typeof(a) === "undefined"  
7 a = {};           // typeof(a) === "object"  
8 a = function() {}; // typeof(a) === "function"
```

www.tutorialrepublic.com/javascript-tutorial/javascript-data-types.php

- Drei Arten Strings zu schreiben:

```
1 let a, b, c;  
2 a = "Hello World!";  
3 b = 'Tom';  
4 // Template Literal  
5 c = `Hallo ${b}, der brutto Preis ist ${100 * 1.5}`;
```

- **Template literals** erlauben das einbinden von Variablen und das Auswerten von Javascript-Expressions innerhalb von Strings.

- nicht boolesche Werte können auch true/false sein
- vieles macht intuitiv Sinn, anderes nicht so...

```
1 // binäre Representation von true/false
2 0 //=> false
3 1 //=> true
4
5 "" //=> false
6 "nonEmptyStrings" //=> true
7
8 2, 12.2, -42 //=> true
9 null, undefined //=> false
10
11 // just accept it..
12 [], {} //=> true
13 function () {} //=> true
```

Truthy & Falsy



Vergleichsoperator

- Auch JS benutzt das Gleichheitszeichen für Vergleiche
- Jedoch gibt es 2 Versionen!

```
1 a == b    // bzw: a != b
2 a === b   // bzw: a !== b
```

- Das doppelte Gleichheitszeichen prüft ob der Wert derselbe ist
- Das dreifache prüft außerdem auf die Gleichheit des Datentypen!

```
1 12 == "12" // ergibt true
2 12 === "12" // ergibt false
3
```

- Es gilt als gute Praxis das dreifache zu benutzen

- Erstellung von Funktionen mit dem `function` Keyword
- 3 Arten eine Funktion zu definieren

```
1 function a(x, y) { ... }  
2 const b = function(x, y) { ... }    // anonyme Funktion  
3 const c = (x, y) => { ... }         // Arrow-Funktion
```

- Funktionsaufruf immer gleich: `a(1,2)`
- Funktionen müssen keinen Rückgabewert haben
 - Dann returned die Funktion `undefined`

```
1 function a() { /* does nothing */ }  
2 // Das loggt "undefined"  
3 console.log(a());  
4 // Das auch, weil console.log an sich kein return Wert hat!  
5 console.log(console.log())
```

- Weitere Schreibweise für anonyme Funktionen
- Erlauben für kürzere Funktionssyntax, insbesondere Einzeiler

```
1  let a;  
2  a = () => { return "arrow functions are awesome" };  
3  // Die {}-Klammern sind bei Einzeilern optional  
4  // Dann entfällt aber auch das return Statement!  
5  a = (a, b) => a + b;  
6  // Die ()-Klammern sind bei einem Parameter optional  
7  a = text => console.log(text);
```

- Für das Erzeugen eines Arrays gibt es zwei Schreibweisen:

```
1 let a;  
2 a = ["eins", 2, 3.141, true];           // JSON-Schreibweise  
3 a = new Array("eins", 2, 3.141, true); // Objekt-Schreibweise
```

- Ein Array kann **verschiedene Datentypen** beinhalten

- Zugriff auf Arraywerte mittels Index

```
1 const colors = ["red", "green", "grey"];  
2 const green = colors[1];
```

Arrays: Iteration

- Länge von Arrays **nicht beschränkt** und wird **automatisch erweitert**
- Länge von Arrays mit der Eigenschaft `.length` auslesbar

```
1  const numbers = [1, 2, 3, 4];
2
3  for(let number of numbers) {
4      console.log(number);
5  }
6
7  for(let index in numbers) {
8      console.log(numbers[index]);
9  }
10
11 for(let i = 0; i < numbers.length; i++) {
12     console.log(numbers[i]);
13 }
```

Arrays: Methoden

- Vereinfachen die Arbeit mit Arrays
- Man gibt eine Funktion (genannt Callback) an, welche auf die Elemente des Arrays angewandt wird.
- Die meisten geben ein neues Array zurück
- Es gibt: `forEach`, `filter`, `map`, `reduce`, ...

```
1  const numbers = [1, 2, 3, 4];  
2  //wie ein for-Loop  
3  numbers.forEach(number => console.log(number));  
4  // filtert alle ungeraden Zahlen heraus  
5  oddNumbers = numbers.filter(number => number % 2);  
6  // quadriert alle Zahlen  
7  squaredNumbers = numbers.map(number => number ** 2);  
8  // summiert alle Zahlen, sprich return Wert ist eine Zahl  
9  sum = numbers.reduce((a, v) => a + v);
```

- werden mit { }-Klammern erzeugt
- bestehen aus Key-Value Pairs mit beliebigen Datentypen
- kann man beliebig verschachteln

```
1  const object = { key: "value" };
2
3  const tom = {
4    name: "Tom Holland",
5    age: 25,
6    phoneNumbers: {
7      mobile: [0123456789, 9876543210],
8      landline: [6574839201]
9    };
10 };
```

Zugriff auf Eigenschaften von Objekten mittels

- Punkt-Schreibweise
- eckige Klammern

```
1  const tshirt = { color: "yellow", size: 12 };  
2  console.log(`Das Shirt ist ${tshirt.color}`);  
3  console.log(`Das Shirt ist ${tshirt["color"]}`);
```

- String in eckigen Klammern kann in einer Variable gespeichert sein:

```
1  keyName = "size";  
2  console.log(`Es kostet ${tshirt[keyName]}`);
```

- JSON Schreibweise kann verkürzt werden
- wenn Variablen und Eigenschaften gleichen Namen haben

```
1  const id = 1, color = "yellow", tshirtSize = 12;
2
3  // lang
4  const tshirt1 = { id: id, color: color, size: tshirtSize };
5
6  // kurz
7  const tshirt2 = { id, color, size: tshirtSize };
```

Zuweisung mit Destructuring

- Auf der linken Seite einer Zuweisung kann die Array- oder Objekt-Schreibweise von JSON verwendet werden um mehrere Variablen auf einmal zuzuweisen

```
1 let [x] = [1, 2, 3];    // x === 1
2 let [x,y] = [1, 2, 3];  // x === 1 & y === 2
3 let person = { name: "Tom", bday: "19.08", age: 22, addresses: [...] };
4 let { name, addresses } = person;
5
6 // besonders praktisch für imports!
7 const { whatINeedFromMyModule } = require('myModule');
8 import { whatINeedFromMyModule } from 'myModule';
```

- `let` & `const` benutzen einen sog. Block Scope
- Ein Block kann mittels `{ }`-Klammern erstellt werden

```
1 { /* ich bin ein neuer Block */ }  
2 function a() {  
3   // ich bin auch ein neuer Block  
4 }
```

- Variablen die innerhalb eines Blocks definiert werden, können nicht von außerhalb zugegriffen werden

```
1 function a() {  
2   const b = "scopedVariable";  
3 }  
4 // ReferenceError: b is not defined  
5 console.log(b)
```

- this ist ein reserviertes Keyword
- Äquivalent zu this in Java und self in Python
- In der Konsole im Browser zeigt es auf das window Objekt:

```
1 console.log(`this = ${this}`);  
2 // this = [object Window]
```

- In Funktionen eines Objekts verweist this auf das zugehörige Objekt
- Gleich mit Klassen

```
1  const tom = {  
2    name : "Tom",  
3    surname : "Holland",  
4    fullName : function () {  
5      return `${this.name} ${this.surname}`  
6    };  
7  };  
8  
9  tom.fullName();    // returns Tom Holland
```

- Funktionen können auch Objekte zurückgeben!

```
1
2 function Person(name, surname) {
3   return {
4     name,
5     surname,
6     fullName: function() { return `${this.name} ${this.surname}` }
7   };
8 };
9
10 const tom = Person("Tom", "Holland");
11 // returns Tom Holland
12 tom.fullName();
```

Objekterzeugung mittels Klasse

- JS bietet auch ein `class` Keyword an, um Objekte zu erzeugen
- Eine eigene Klasse macht syntaktisch Sinn, wenn das Objekt viele Funktionen hat

```
1  class Person {
2      constructor(name, surname) {
3          this.name = name
4          this.surname = surname
5      };
6      fullName() {
7          return `${this.name} ${this.surname}`
8      };
9  };
10
11  const tom = new Person("Tom", "Holland");
12  // returns Tom Holland... surprise
13  tom.fullName();
```

```
1 class Pet {  
2     constructor() {  
3         this.status = "sleeping";  
4     };  
5 };  
6  
7 class Mammal extends Pet {  
8     constructor() {  
9         super();  
10        this.legs = 4;  
11    };  
12 };  
13  
14 class Dog extends Mammal {  
15     constructor(breed) {  
16         super();  
17         this.breed = breed;  
18         this.isGoodBoy = true;  
19     };  
20     sit() {  
21         this.status = "sitting";  
22     };  
23 };
```

Instantiierung und Benutzung der Klasse
Dog:

```
1 const beagle = new Dog("beagle");  
2 beagle.legs;    // 4  
3 beagle.sit();  
4 beagle.status;  // sitting
```

- JavaScript wurde als Sprache für dynamische/aktive Elemente in Webseiten entwickelt
- „Dynamisches HTML“ / DHTML
- Verbindung von JavaScript und HTML nötig
- HTML bietet bestimmte **Tags** dazu an

Einbindung von JavaScript in HTML

```
1 <head>
2 <body>
3 <h1>Farbwahl</h1>
4 <form>
5   <input type="button" value="Rot"  onclick="setcolor('red')">
6   <input type="button" value="Grün" onclick="setcolor('#0F0')">
7   <input type="button" value="Blau" onclick="setcolor('blue')">
8 </form>
9 <script>
10 function setcolor(color) {
11   const body = document.body;
12   body.style.backgroundColor = color;
13 }
14 </script>
15 </body>
```

Einbindung von JavaScript in HTML

■ lokale externe Javascript-Datei

```
1 <script src="jquery.js"></script>
```

■ entfernte externe Javascript-Datei (CDN)

```
1 <script src="https://ajax.googleapis.com/[...]/jquery.min.js"></script>
```

■ mit <script>-Tag

```
1 <script>alert("Hello World!");</script>
```

■ onevent-Attribute, Beispiele:

```
1 <body onload="...">
2 <a href="..." onmouseover="...">
3 <input type="button" onclick="...">
4 <form onsubmit="...">
```

```
1 <a href="..." onclick="...">
2 <input onchange="...">
3 <div ontouchstart="..." ontouchend="..." ontouchmove="...">
4 <body onoffline="..." ononline="...">
```

- Node.js ist auf allen Rechnern im WinCIP installiert

- nodejs auf der Konsole eingeben

- JavaScript Konsole, gut für Minimalbeispiele
- Frage: Warum wird undefined ausgegeben?

```
goltzsch@iz01:~ % nodejs
> var x = 5;
undefined
> console.log(x);
5
undefined
> 
```

- Mit dem Aufruf von `nodejs datei.js` können JavaScript Dateien ausgeführt werden

Ausblick

- Asynchrone Programmierung mit JavaScript
- Browser Entwicklertools
- Ausgabe von Aufgabe 1

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- https://en.wikipedia.org/wiki/List_of_ECMAScript_engines
- <https://web-development.github.io/> CC BY-NC-SA von Brigitte Jellinek
- <https://developer.mozilla.org/docs/Web/JavaScript>
- https://developer.mozilla.org/en/docs/Web/JavaScript/Guide/Using_promises
- <https://medium.com/codebuddies/17e0673281ee>