

Web Engineering

JavaScript

Adrian Herzog

(basierend auf der Arbeit von Michael Faes, Markus Mächler, Michael Heinrichs & Prof. Dierk König)

JavaScript

- JavaScript (kurz JS) ist die Programmiersprache für Scripting im Browser.
- Wurde 1995 von **Brendan Eich** bei Netscape entwickelt. Der erste Prototyp wurde angeblich in 10 Tagen geschrieben.
- Interpretierte Sprache (oder just-in-time compiled, für bessere Performance).
- Node.js und ähnliche Tools erlauben das Ausführen von JS ohne Browser.
- [JS-Dokumentation auf MDN](#)

Was hat JS mit Java zu tun?

Der Standard hinter JS heisst **ECMAScript**.

Der Name **JavaScript** ist eine Marke von Oracle.

Interview-Auszug zum Namen:

Java is to
JavaScript
what
Car is to Carpet

From an [interview](#) made to its creator [Brendan Eich](#):

InfoWorld: As I understand it, JavaScript started out as Mocha, then became LiveScript and then became JavaScript when Netscape and Sun got together. But it actually has nothing to do with Java or not much to do with it, correct?

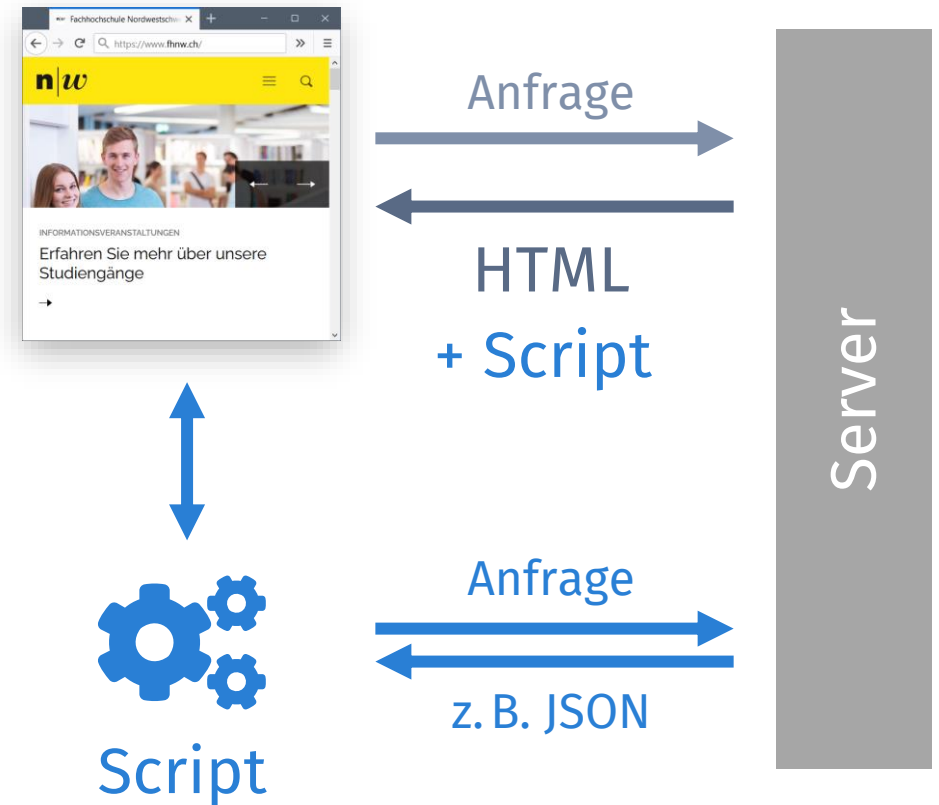
Eich: That's right. It was all within six months from May till December (1995) that it was Mocha and then LiveScript. And then in early December, Netscape and **Sun** did a license agreement and it became JavaScript. And the idea was to make it a complementary scripting language to go with Java, with the compiled language.

Scripting im Browser

1. Script wird vom Server geladen (entweder im HTML eingebettet oder in separater JS-Datei)
2. Client führt Script aus.

JS kann:

- HTML über das **DOM API** ändern
- **HTTP-Requests** ausführen
- **Auf Events reagieren** (z.B. Click auf Element)
- Weitere Beispiele



Mit JavaScript HTML verändern (DOM API)

In einem Browser kann man mittels `document` auf HTML zugreifen:

```
<!DOCTYPE html>
<html lang="en">
<head><title>JS Test</title></head>
<h1>Hello, World!</h1>
<p>No text here...</p>
<script>
  const p = document.querySelector('p');
  p.textContent = 'Text added by JS';
</script>
</html>
```

Hello, World!

Text added by JS

JavaScript als Sprache

(Basics, mehr Details im Modul Web Programming)

JavaScript im Vergleich zu Java

	Java	JavaScript
Distribution	Als Java ByteCode (Kompilation von Source Code zu ByteCode)	Als Source Code (allenfalls minifiziert, etc., aber immer noch gültigs JS). <i>Viele Fehler erst zur Laufzeit erkennbar.</i>
Ausführung	In Java Virtual Machine (JVM)	JS Engine im Browser oder z.B. in node.js
Syntax	Inspiriert von C	Inspiriert von C
Typisierung	Statisch typisiert. Typ wird spätestens beim Kompilieren festgelegt und geprüft.	Dynamisch typisiert. Das ist erlaubt: let a = 5; a = ['a', 'b', 'c'];
Vererbung	Klassenbasiert	<u>Prototypenbasiert</u>

Dynamische Typisierung

Keine statischen Typen für Variablen, etc.

64-bit Float! Kein eigener Int-Typ

```
let x = 'Hello, World!'; // x enthält einen String
x = 42;                  // jetzt eine Zahl
x = false;               // jetzt einen Boolean
x = null;                // jetzt 'null'
x = undefined;           // jetzt 'undefined'
```

Kein Compiler, der prüft, ob Methoden/Eigenschaften existieren!

x hat keinen Typ

```
function foo(x) { console.log(x.textContent); }
```

?

```
foo(document.querySelector('p'));
```

Text added by JS

```
foo(42);
```

undefined

!

Typen

JavaScript unterscheidet grob folgende Typen:

- ***Primitive Typen***

string, number, bigint, boolean, undefined, null, symbol

- ***Objekte***

Object, Array, Set, Map, Date, ...

- ***Funktionen***

Primitive Typen (1)

string

```
const singleQuotes = 'Hello World';  
const doubleQuotes = "Hello World";  
const template1 = `2 * 5 = ${2 * 5}`; // '2 * 5 = 10'  
const name = "Jane Doe";  
const template2 = `Name: ${name.toUpperCase()}`; // 'Name: JANE DOE'
```

number

als double
gespeichert!

```
const integer = 42;  
const float = 42.1234;  
const infinity1 = Infinity;  
const infinity2 = 1 / 0; // division by zero returns Infinity  
const nan1 = NaN; // NaN = Not a Number  
const nan2 = parseInt('one'); // failed conversion returns NaN
```

bigint

seit ca. 2021

```
const answer = 42n;
```

Primitive Typen (2)

boolean

```
const enableLogging = true;  
const logPassword = false;
```

undefined

Wert nicht
gesetzt oder
Eigenschaft
existiert nicht

```
let a;  
console.log(a === undefined); // true  
const b = [1, 2, 3];  
console.log(b[3]); // undefined
```

null

Referenz auf
nicht existierendes
Objekt

```
const invoice = {  
  amount: 100,  
  currency: 'CHF',  
  paymentDate: null  
};  
console.log(typeof null); // object (!)
```

Übung 1: Typkonvertierung in JavaScript

Gewisse Operatoren und Anweisungen wandeln Werte automatisch um. Verwende die JavaScript-Konsole im Browser, um zu bestimmen, welche Ausgabe folgende Codestücke erzeugen. Überlege dir jeweils vorher, was du erwartest.

a) `console.log(false + 5);`

b) `console.log('1' == 1);`

c) `console.log(4 + '2' - 2);`

d)

```
if ('Hello, World!') {  
    console.log('yes');  
} else {  
    console.log('no');  
}
```

Übung 1: Typkonvertierung in JavaScript

Gewisse Operatoren und Anweisungen wandeln Werte automatisch um. Verwende die JavaScript-Konsole im Browser, um zu bestimmen, welche Ausgabe folgende Codestücke erzeugen. Überlege dir jeweils vorher, was du erwartest.

a) `console.log(false + 5);`

5

d)

```
if ('Hello, World!') {  
    console.log('yes');  
} else {  
    console.log('no');  
}
```

yes

besser: ===

b) `console.log('1' == 1);`

true

c) `console.log(4 + '2' - 2);`

40

Truthy, Falsy und Nullish

Werte jeglicher Typen können für Boolesche Ausdrücke verwendet werden. Diese werden dann nach `true` oder `false` konvertiert.

Falsy (Wert wird zu `false` konvertiert)

`false`, `0`, `-0`, `0n`, `"`, `null`, `undefined`, `NaN`, and `document.all`.

Truthy (Wert wird zu `true` konvertiert)

All values that are not falsy

Nullish (Wert wird wie `null` behandelt)

`null` and `undefined`

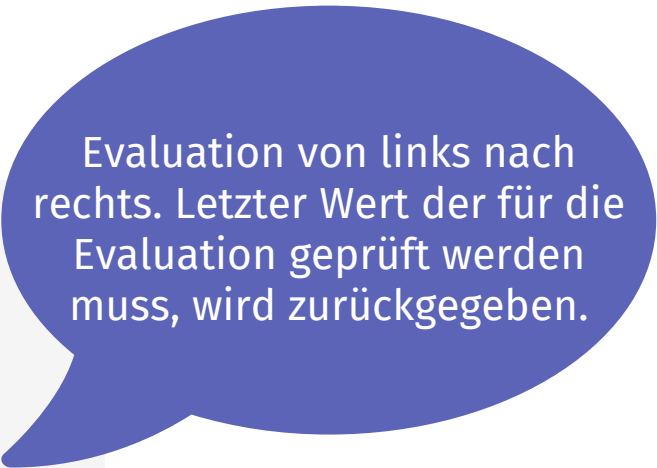
Logische Operatoren

Verhalten sich wie erwartet für Boolean-Werte. Unterstützen aber auch andere Typen und liefern dann interessante Überraschungen als Ergebnis.

Siehe auch [MDN](#).

```
// AND
const a1 = "Cat" && "Dog"; // returns Dog
const a2 = false && "Cat"; // returns false
const a3 = "Cat" && false; // returns false
```

```
// OR
const o1 = "Cat" || "Dog"; // returns Cat
const o2 = false || "Cat"; // returns Cat
const o3 = "Cat" || false; // returns Cat
```



Evaluation von links nach rechts. Letzter Wert der für die Evaluation geprüft werden muss, wird zurückgegeben.

Objekte ohne Klassen

JavaScript ist u.a. objekt-orientiert, funktioniert aber ohne Klassen!

```
let person = {};  
person.name = 'Fritz';  
person.age = 42;  
person.greet = function() {  
    console.log('Hello, I am ' + this.name);  
}  
  
person.greet();
```

Hello, I am Fritz

Objekte sind eigentlich nur HashMaps:

```
console.log(person['name']);  
console.log(person['age']);
```

Fritz
42

Spezielle Syntax für Objekte und Arrays

JavaScript macht es einfach, Objekte und Arrays zu definieren:

```
let person = { name: 'Fritz', age: 42 };  
console.log(person.name);
```

Fritz

```
let list = ['Fritz', 'Fränzi', 'Freddy'];  
console.log(list[1]);
```

Fränzi

Kombiniert:

```
let mother = {  
  name: 'Elsa',  
  children: [  
    { name: 'Fritz', age: 42 },  
    { name: 'Fränzi', age: 38 }  
  ]  
};  
console.log(mother.children[1].age);
```

fast JSON

38

Funktionen als Werte

Funktionen können ähnlich verwendet werden wie Methoden in Java:

```
function foo() {  
    console.log('Hello, World!');  
}
```

```
foo();
```

Hello, World!

Aber: Können auch als Werte verwendet werden:

```
let foo = function() {  
    console.log('Hello, World!');  
};  
let bar = foo;  
bar();
```

Hello, World!

Arrow Functions

Kurzschreibweise für Funktionen (seit ca. 2016) mit feinen Unterschieden und Limitierungen.

```
const f1 = a => console.log(a);
```

Minimale
Variante

```
const f2 = (a, b) => {  
  console.log(a);  
  console.log(b);  
};
```

Mehrere
Argumente und
mehrzeiliger
Body

```
const a = [1, 2, 3, 5, 8, 13, 21];  
const b = a.filter(x => x % 2 === 0);  
console.log(b);
```

Typische Anwendung

JavaScript im Browser

JavaScript + HTML

Wohin mit dem Code?

1. In `<script>`-Tag

```
<h1>Hello, World!</h1>  
<script> console.log('Hello, console!') </script>
```

ausgeführt, wenn
`<script>` geparkt

2. In externe Datei:

```
<script src="path/to/script.js"></script>
```

wie oben

wenn Seite
geladen ist

```
<script src="path/to/script.js" defer></script>
```

```
<script src="path/to/script.js" async></script>
```

irgendwann

3. In Event-Attribut:

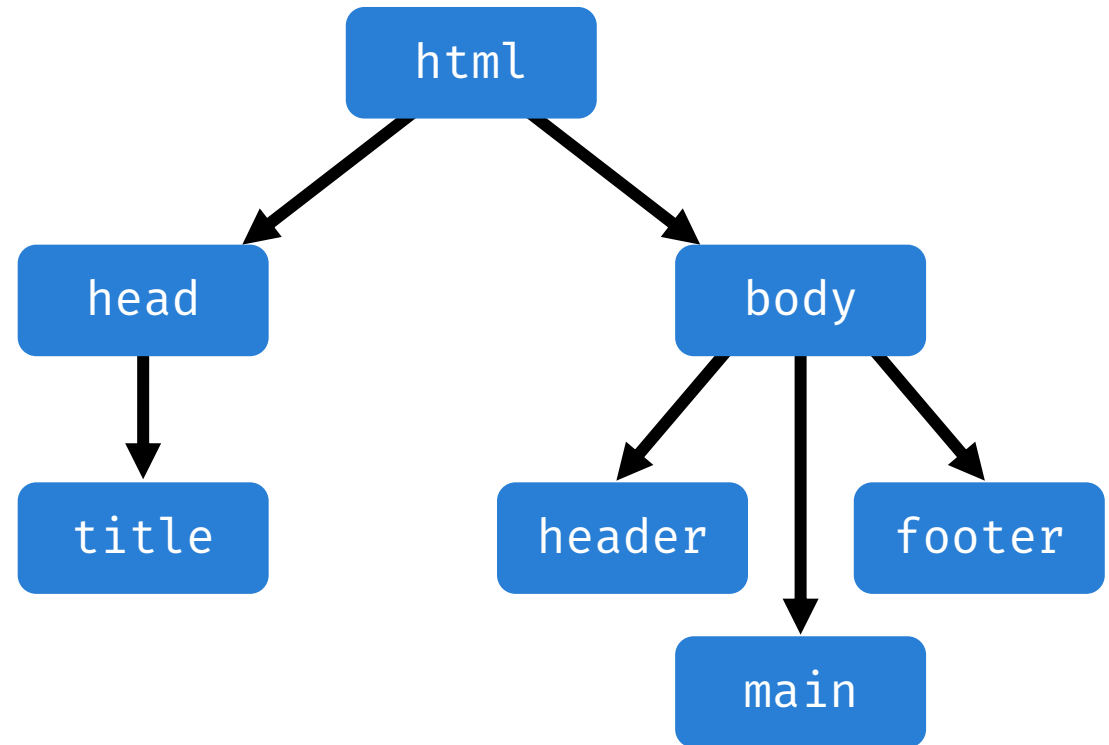
```
<h1 onclick="this.textContent = 'Clicked'">  
  Click me  
</h1>
```

wenn geklickt

Zugriff auf das «DOM»

DOM: Objekt-basierte Repräsentation des HTML-Dokuments

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>...</title>
  </head>
  <body>
    <header>...</header>
    <main>...</main>
    <footer>...</footer>
  </body>
</html>
```



In JavaScript: Zugriff über globale `document`-Variable

Wichtige document-Methoden

Methode	Beschreibung
<code>querySelector(selector)</code>	Findet das erste Element, das dem gegebenen CSS-Selektor entspricht
<code>querySelectorAll(selector)</code>	Findet alle Elemente, die dem gegebenen CSS-Selektor entsprechen

Beispiele:

```
document.querySelector('h1.title')
```

```
document.querySelector('main > p')
```

```
for (let li of document.querySelectorAll('#menu li')) {  
    // do something with 'li'  
}
```

Element-Methoden/Eigenschaften

Methode/Eigenschaft	Beschreibung
<code>element.id</code>	Enthält die ID des Elements
<code>element.value = newValue</code>	Schreibt einen neuen Wert in das Element (z. B. in <code><input></code> -Element)
<code>element.textContent = newText</code>	Schreibt einen neuen Text in ein Element (z. B. in <code><p></code> -Element)
<code>element.innerHTML = newHTML</code>	Schreibt einen neuen HTML-Inhalt in ein Element
<code>element.classList.add(newStyle)</code>	Fügt eine CSS-Klasse zum Element hinzu

Beispiel:

```
let p = document.querySelector('main > p');  
p.innerHTML = 'Oh <em>yeah!</em>';
```


Auf Events reagieren

Um User-Interaktionen zu ermöglichen, muss man auf *Events* reagieren. Statt `onclick=...` besser mit `addEventListener`:

Beispiele:

```
<input type="text" id="name-field">  
<button type="button" id="add-button">Add</button>
```

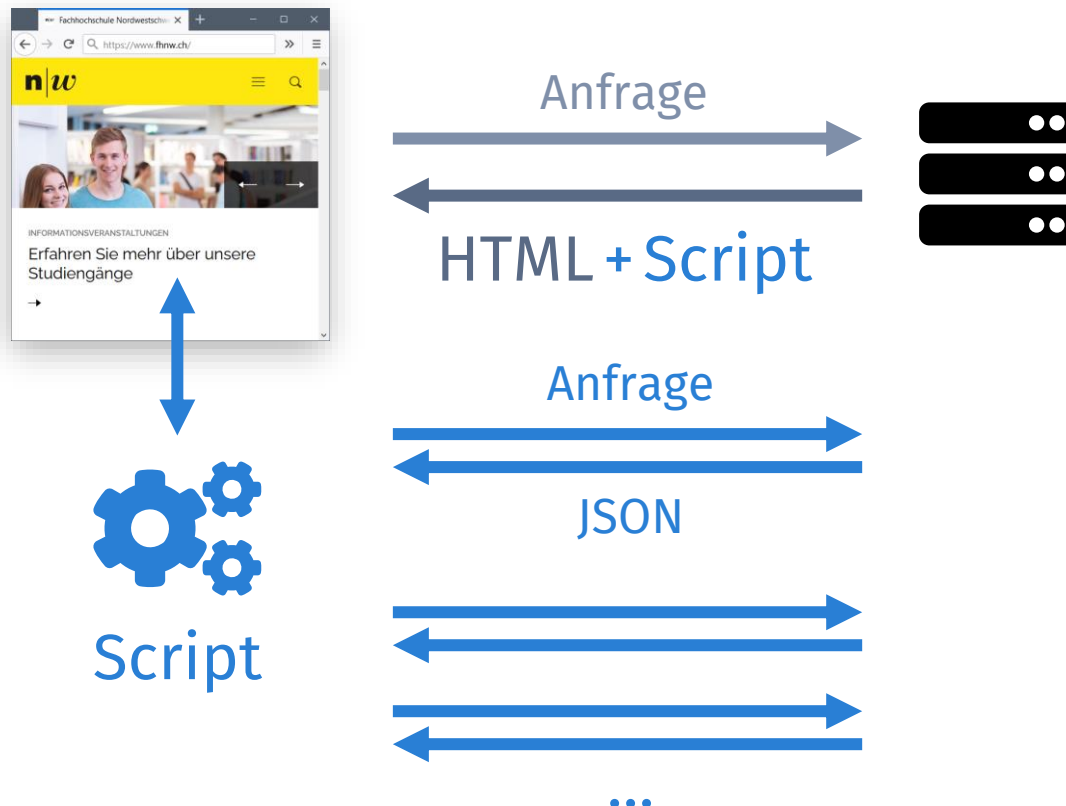
```
let button = document.querySelector('#add-button');  
button.addEventListener('click', function() {  
    console.log('Add button was clicked!');  
});
```

```
let field = document.querySelector('#name-field');  
field.addEventListener('input', function() {  
    console.log('Name was changed!');  
});
```

Kommunikation mit Server

Unsere Web-Apps bisher: Bei jeder Interaktion mit dem Server wird neue Seite geladen...

Mit JS sind mehr/bessere Interaktionen möglich!



Fetch API

Das [Fetch API](#) ist der HTTP Client in JavaScript ([seit 2017](#), vorher wurde das XMLHttpRequest Objekt verwendet).

```
async function getData() {  
  const url = "https://example.org/products.json";  
  try {  
    const response = await fetch(url);  
    if (!response.ok) {  
      throw new Error(`Response status: ${response.status}`);  
    }  
  
    const json = await response.json();  
    console.log(json);  
  } catch (error) {  
    console.error(error.message);  
  }  
}
```

Fragen?

