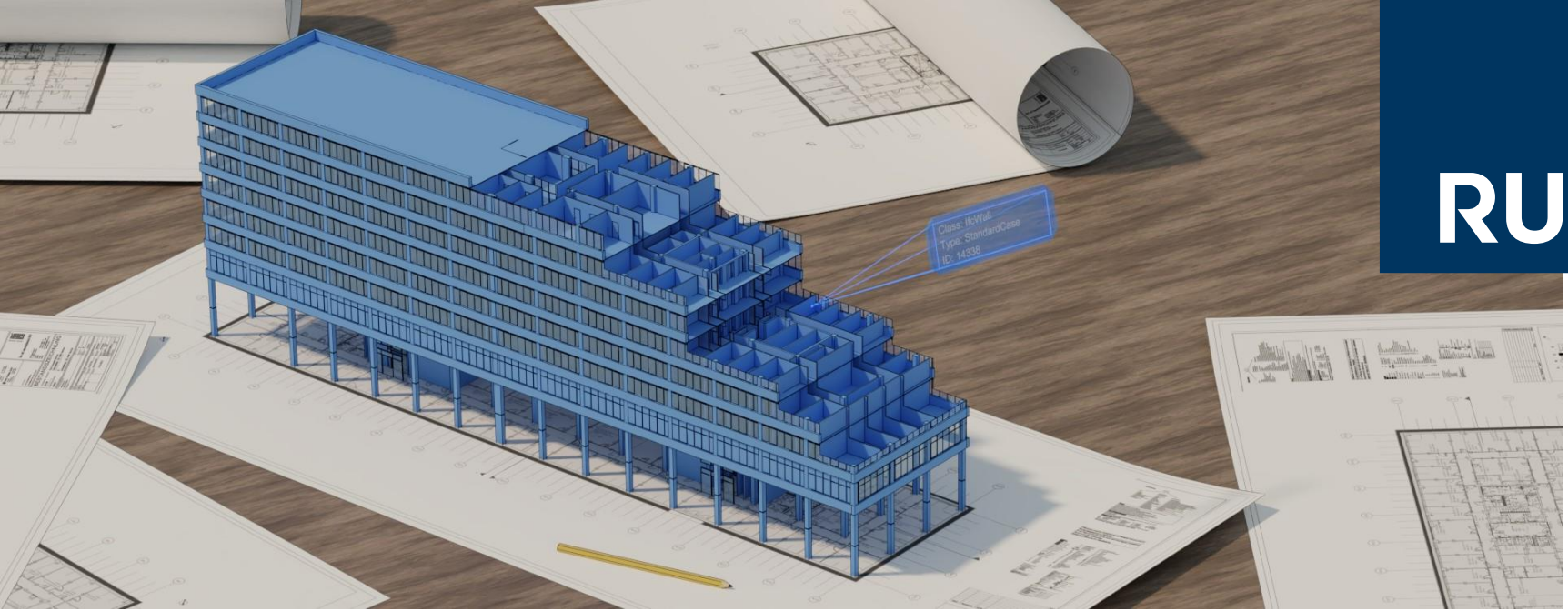


RUB

RUHR-UNIVERSITÄT BOCHUM

WEB-ENGINEERING

Informatik
im Bauwesen

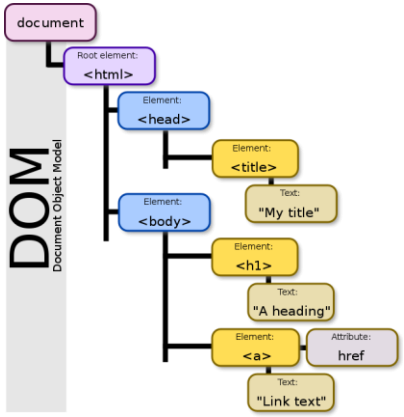
Sommersemester 2023

Web-Engineering
Dynamische Webinhalte

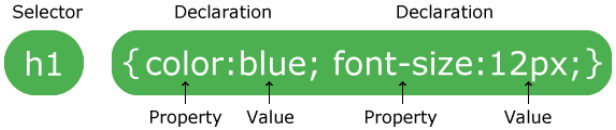
Dynamische Webinhalte

Rückblick

- HMTL dient zur Strukturierung vom Webseiteninhalt
- CSS definiert das Erscheinungsbild von Webseiten
- Über CSS und HTML lassen sich keine neuen Inhalte dynamisch nachladen



Quelle: Birger Eriksson CC BY-SA 3.0



Quelle : https://www.w3schools.com/css/css_syntax.asp

Dynamische Webinhalte

Es gibt mehrere Möglichkeiten in einer Webseite dynamisch Inhalt anzuzeigen

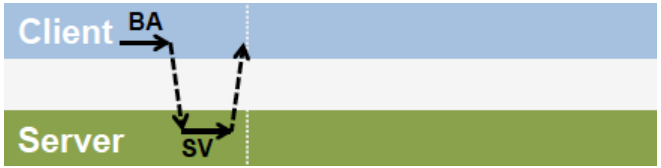
- Alle Informationen werden bei dem Seitenaufruf an den Client übermittelt
  - Vielleicht benötigt der Benutzer nicht alle Informationen, dennoch wird alles geladen
  - Je nach Größe der Seite kann dies zu einer langen Ladezeit führen
- Die Seite wird mit neuen Informationen neu geladen
    - Alte Informationen werden dabei ersetzt
- Es werden neue Inhalte dynamisch nachgeladen und die bisherige Seite wird um neue Informationen ergänzt
    - Nur neue Webseiteninhalte werden bei Bedarf neu geladen und ergänzt

Dynamische Webinhalte - Beispiel

Client-Server Kommunikation anhand der Google Online-Umfrage

- Es ist eine Verarbeitung auf dem Server notwendig
  - Request → Response

- `BA` = Benutzeranfrage
- `SV` = Server Verarbeitung
- `CV` = Client Verarbeitung

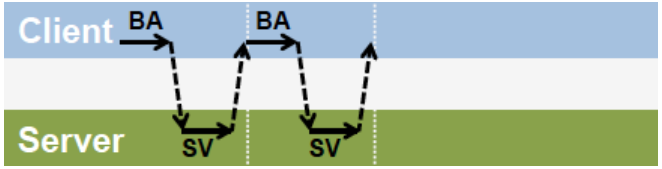


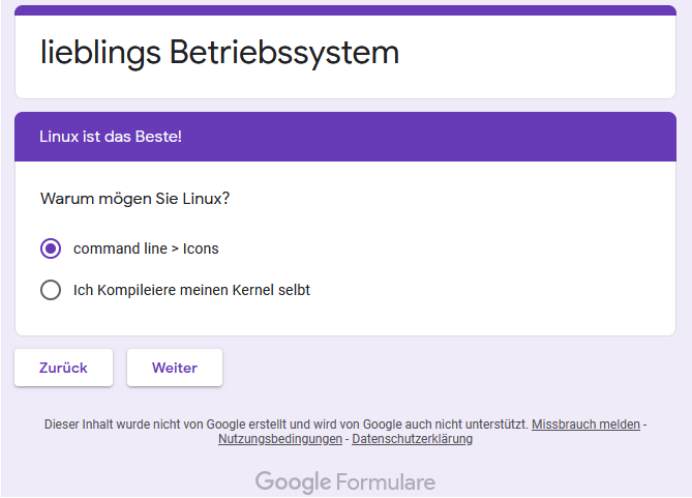
Dynamische Webinhalte - Beispiel

Client-Server Kommunikation anhand der Google Online-Umfrage

Es ist eine Verarbeitung auf dem Server notwendig

- Request → Response
  - `BA` = Benutzeranfrage
  - `SV` = Server Verarbeitung
  - `CV` = Client Verarbeitung



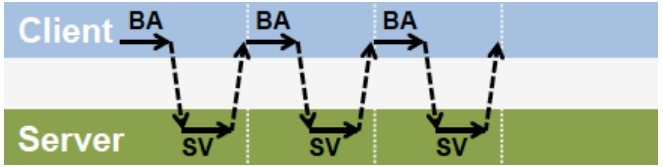


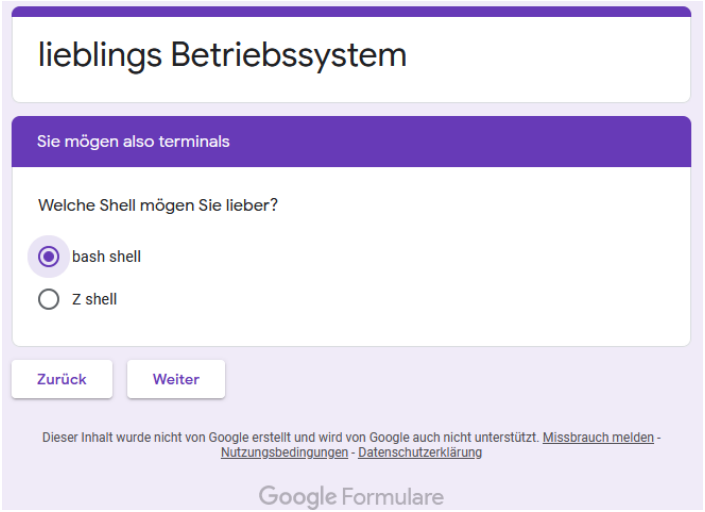
Dynamische Webinhalte - Beispiel

Client-Server Kommunikation anhand der Google Online-Umfrage

Es ist eine Verarbeitung auf dem Server notwendig

- Request → Response
  - `BA` = Benutzeranfrage
  - `SV` = Server Verarbeitung
  - `CV` = Client Verarbeitung



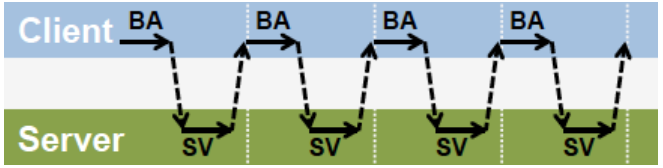


Dynamische Webinhalte - Beispiel

Client-Server Kommunikation anhand der Google Online-Umfrage

- Rechenintensive Operationen können auf leistungsstarke Server ausgelagert werden

  - `BA` = Benutzeranfrage
  - `SV` = Server Verarbeitung
  - `CV` = Client Verarbeitung

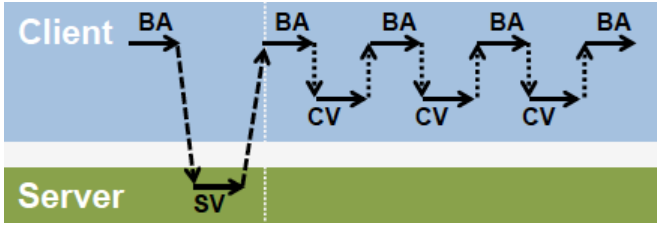


Dynamische Webinhalte - Beispiel

Client-Server Kommunikation anhand der Google Online-Umfrage

- JavaScript (JS) ermöglicht es Webseiteninhalte clientseitig zu generieren, zu verändern zu validieren, und nachzuladen
- Weniger Datentransfer, schnellere Reaktionszeiten und mögliche asynchrone Kommunikation mit dem Server

  - `BA` = Benutzer Anfrage
  - `SV` = Server Verarbeitung
  - `CV` = Client Verarbeitung



Beispiel für eine Clientseitige Verarbeitung von Webseiten mit dynamischen Inhalten.

Web-Engineering
Java-Script (js)

Popularität von JavaScript

third quarter plot for 2022

https://redmonk.com/sogrady/2022/10/20/language-rankings-6-22/

Popularität von JavaScript

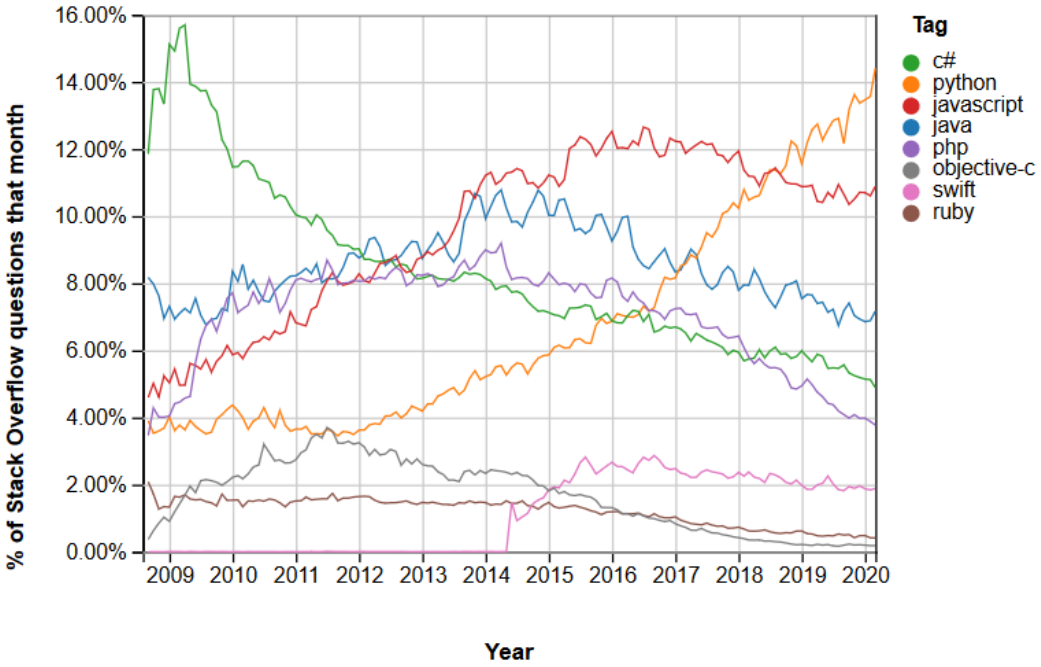
Warum ist JavaScript so populär?

- Eine Cross-Plattform Scriptsprache die auch auf mobilen und eingebetteten Geräten einfach funktioniert
- Mit steigender Zahl mobiler Endgeräte steigen auch die Einsatzmöglichkeiten und somit die Beliebtheit von Javascript
- Es gibt wenige Alternativen für dynamische Inhalte im Frontend
- Stetige Weiterentwicklung der Sprache
  - In Form von Frameworks und anderen Erweiterungen
- Ermöglicht dynamische Anpassungen der Inhalte an entsprechende Endgeräte
- Möglichkeiten für clientseitige Berechnungen

Popularität von JavaScript

Warum ist JavaScript so populär?

- JS gewinnt seit den letzten 10 Jahren immer weiter an Beliebtheit
- Mit steigender Anzahl Webbasierter Anwendungen steigt auch die Popularität der entsprechenden Technologien wie Javascript
- Javascript kann auf fast allen mobilen Endgeräten genutzt werden, diese machen teilweise 50% der Seitenaufrufe aus



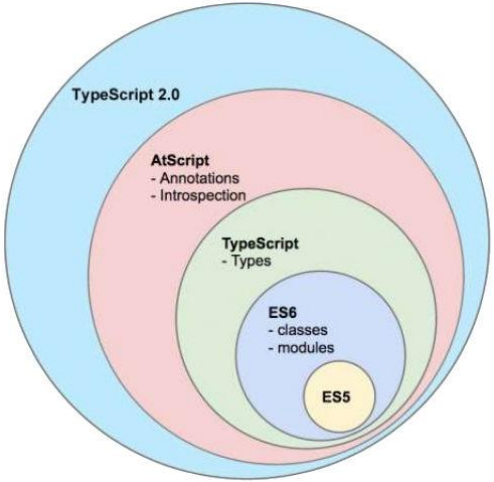
Quelle: https://insights.stackoverflow.com/trends

Aktuelle Entwicklungen

TypeScript

”JavaScript that scales.”

- Typescript ist ein Superset von JavaScript
- JS-Anweisung sind eine Teilmenge von TypeScript, dass bedeutet jede JS-Anweisung ist ebenso gültig in TypeScript
- Gibt die Möglichkeit statische Variablen zu nutzen



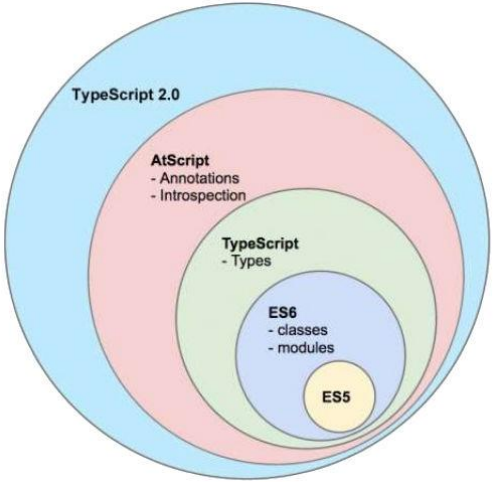
Quelle: https://devopedia.org/typescript

Aktuelle Entwicklungen

CoffeeScript

“It’s just a plain JavaScript”

- Programme werden in JavaScript transkompiliert
- Verbessert die Lesbarkeit und Prägnanz von JavaScript Code
- Kompakterer Programm-Code (ca. 30 – 55% weniger Programmzeilen



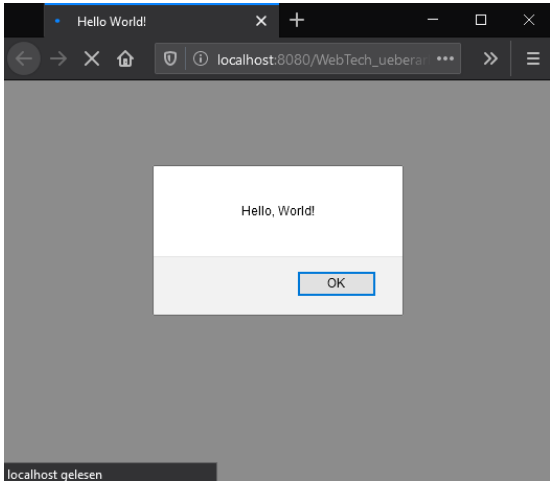
Quelle: https://devopedia.org/typescript

JavaScript Beispiel

Ein erstes Beispiel – dynamisches HTML

- JavaScript Skripte können innerhalb des <script>-Tags in eine Webseite eingebunden werden
- Dieses Beispiel ruft bei jedem laden der Seite einen Alarm-Dialog (alert) mit der Ausgabe „Hello, World!“ auf:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World!</title>
</head>
<body>
        <script>
                alert("Hello, World!");
        </script>
</body>
</html>
```

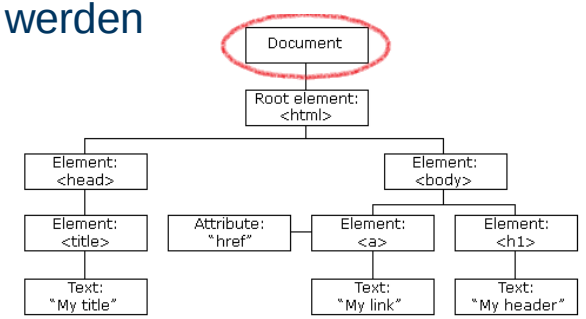


JavaScript Beispiel

Ein erstes Beispiel – dynamisches HTML

- In diesem Beispiel wird eine wichtige Funktionalität von JavaScript gezeigt: `Manipulation des DOM durch JavaScript`
- Innerhalb des `script`-Elements wird über `document` auf das DOM zugegriffen
  - Der Funktion `getElementsByTagName` wird der Tag eines Absatzes übergeben (hier `p`)
  - Die Hintergrundfarbe des ersten HTML Elementes wird dann über die `style`-Eigenschaft abgeändert und auf `yellow` (gelb) gesetzt werden

```
<body>
    <h1>JavaScript Kurs für Einsteiger</h1>
    <p>JavaScript lernen einfach gemacht ...</p>
    <script type="text/javascript">
        document.getElementsByTagName('p')[0].
                                style.backgroundColor =
'yellow';
    </script>
</body>
```



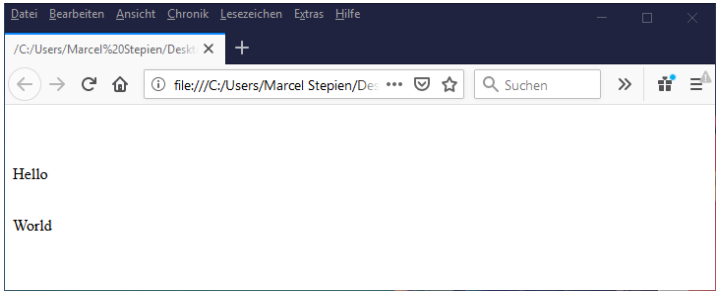
Quelle: https://www.w3schools.com/js/js_htmldom.asp

JavaScript Beispiel

Ein erstes Beispiel – dynamisches HTML

- Im letzten Beispiel wurde ein bestehendes DOM-Element angepasst
- In diesem Beispiel wird über die append() Methode der String „World“ an den bestehenden body des DOMs angehangen

```
<body>
    <main>
        <p>Hello</p>
        <script>
            document.body.append
('World');
        </script>
    </main>
</body>
```



Web-Engineering
Datentypen & Werte

Datentypen & Werte

Allgemeine Informationen zur Typisierung in JS

- JavaScript verwendet eine dynamische Typisierung, d.h. alle Datentypen werden zur Laufzeit ermittelt

- Variablen werden durch die Schlüsselwörter `var`, `let` oder `const` definiert
  - Wird kein Schlüsselwort angegeben, wird es eine globale Variable (sollte vermieden werden)

Datentypen & Werte

Insgesamt gibt es in JavaScript sechs primitive Datentypen:

- `boolean`, `null`, `undefined`, `number`, `string` und `symbol`
  - `Symbol` wurde 2015 mit ECMAScript 2015 eingeführt

Außerdem gibt es verschiedene Objekttypen z.B.:

- `Object`, `Data`, `Array`

Alle Zahlen sind 64-Bit Fließkommazahlen:

```
var ganzZahl = 5;
let fließkommaZahl = 5.4;
```

Datentypen & Werte

Gültigkeitsbereich von Variablen (Scope)

- Ein Block ist ein von Klammern eingeschlossener Bereich, beispielsweise der von einer if-Anweisung eingeschlossene Bereich
- Variablendeklaration mittels `var` oder `let` macht einen Unterschied
  - `var` → Außerhalb einer Funktion deklariert global gültig, ansonsten im entsprechenden Block einer Funktion
  - `let` → Variable ist nur innerhalb des Blocks sichtbar
- `const` → Variable ist wie `let` nur im Block sichtbar und kann zudem keinen neuen Wert zugewisen bekommen

```
if ( … )
{
  …
} else {
  …
}

var x = 1;
let y = 1;
const z = 1;
```

Datentypen & Werte

Gültigkeitsbereich von Variablen (Scope)

```
function foo() {

  var x = 1;

  function bar() {
    var y = 2;
    console.log(x); // 1
    console.log(y); // 2
  }

  bar();
  console.log(x); // 1
  console.log(y); // ReferenceError im strict mode
}

foo();
```

Datentypen & Werte

Zeichenketten (Strings)

- Zeichenketten sind in JavaScript 16-Bit-Zeichen
- Es gibt keinen Datentyp für ein einzelnes Zeichen
- Es können einfach oder doppelte Anführungszeichen zur Definition eines Strings verwendet werden

```
Beispiel:
var name = ‘Müller‘;
var titel = “Müller“;
```

Datentypen & Werte

Boolsche Werte

- In JS können auch nicht boolsche Werte als true oder false interpretiert werden

```
false == 0          // true
false == ""         // true
NaN == NaN      // false
null == false       // false
null == true        // false
null == null    // true
if (null)           // false
if ({})             // true
```

JS Spec. Vergleich mit NaN ist immer false

Alternative:
isNaN(NaN)
//true

Datentypen & Werte

Boolsche Werte

```
x == x// true
x != x// false
```
} Vergleich auf “number”

```
1 == true    // true
1 === true   // false
5 + 5 === 10 // true
```
} Identitätsoperator „===“ Prüfung des Wertes & des Typs

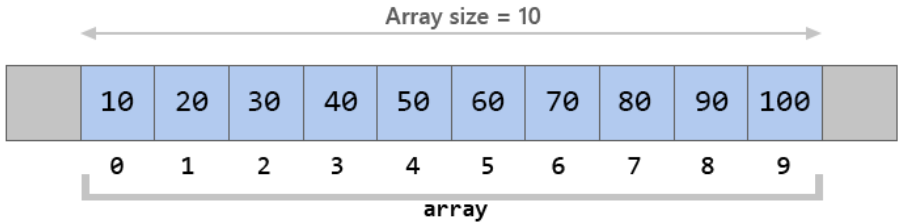
```
0.1 + 0.2 == 0.3   // false
Array(3) == ‘,,‘   // true
```

```
Math.max()>Math.min()// false
Math.min()>Math.max()// true
```
}
```
console.log(Math.min());//Infinity
console.log(Math.max());//-Infinity
```

Datentypen & Werte

Was sind Arrays

- Sind wie ein Schubladenschrank und bieten eine einfache Möglichkeit Daten zu strukturieren
- Die gewünschte Schublade wird über einen Index angegeben
- Eine einzelne Schublade ist wie eine einzelne Variable, die einen Wert enthalten kann



Quelle: https://laptrinhx.com/remove-element-from-an-array-in-java-2026564821/

Quelle: https://www.madress.com/bisley-schubladenschrank-mit-15-schubladen.html

Datentypen & Werte

Arrays in JavaScript

Arrays sind ebenfalls Objekte und können über einen Konstruktor (optional nur [ ]) als auch eine Literal-Schreibweise erzeugt werden.

Beispiel über Konstruktor:

```
var zahlen = new Array(); // = [] auch möglich
zahlen[0] = 1.1;
zahlen[1] = 2.2;
```

Beispiel über Literal-Schreibweise:

```
var zahlen = [1.1, 2.2]
```

Datentypen & Werte

Arrays in JavaScript

- Wird dem Konstruktor eine Zahl übergeben, wird diese als Länge des Arrays angesehen:

```
new Array(10)
```

- Werden mehrere Werte angegeben, wird ein Array mit den Werten initialisiert:

```
new Array (10, 11) // Länge 2
```

- 2D Arrays können wie folgt erzeugt werden:

```
var items = [[1,2],[3,4],[5,6]];
items[1][0]; // 3
```

Multiple Wertzuweisung

Destructuring assignment

Ermöglichen jedes Element eines Arrays oder jede Eigenschaft eines Objektes in verschiedene Variablen zu speichern

- Dies geschieht in einer Zeile

```
let array = [1, 2, 3];

// Zuweisung der Werte 1, 2, 3 an die Array-Werte x, y, z
let [x, y, z] = array;

console.log(x); // Ausgabe: 1;

console.log(y); // Ausgabe: 2;

console.log(z); // Ausgabe: 3;
```

JavaScript Ausgaben

JavaScript kann Daten auf verschiedene Arten ausgeben

- `console.log()`
  - Schreibt die Ausgabe in die Konsole des Browsers → `strg+shift+k` in Firefox
- `document.write()`
  - Schreibt die Daten in ein HTML-Output
- `window.alert()`
  - Schreibt die Ausgabe innerhalb des momentanen Fensters in einen Alarm-Dialog

Web-Engineering
Kontrollstrukturen & Schleifen

Kontrollstrukturen & Schleifen

Kontrollstrukturen und Schleifen regeln den Programmfluss

- Kontrollstrukturen und Schleifen funktionieren in JavaScript ähnlich wie in Java
- Innerhalb einer `if`-Klausel können Werte beliebigen Typs verwendet werden, da jeder Wert zu `true` oder `false` evaluiert wird

```
if(parameter)  {
    console.log("definiert und nicht null");
} else {
    alert("nicht definiert");
}
```

- Jede Zahl außer 0 ist true

Kontrollstrukturen & Schleifen

Kontrollstrukturen und Schleifen regeln den Programmfluss

Innerhalb einer `switch`-Anweisung werden `case`-Schlüsselwörter geprüft

- Mit einer `case`-Anweisung prüft man in der Regel `strings` und `numbers`
- Ein `case`-Schlüsselwort kann auch dynamisch über Funktionen ermittelt werden
- `case`-Anweisungen sind in vielen Sprachen leserlicher und manchmal sogar schneller als das `if`-Äquivalent

```
var fruit = "mango";
switch(fruit)  {
    case "apfel":
       console.log("Apfel kosten 1 Euro.");
       break;
    case "mango":

       console.log("Mangos kosten 2 Euro.")
;
       break;
    case "birne":

       console.log("Birne kosten 1.25 Euro.
");
       break;
    default:
       console.log(fruit +
                   " nicht im Sortiment.");
       break;
}
```

Kontrollstrukturen & Schleifen

Kontrollstrukturen und Schleifen regeln den Programmfluss

- Die `while`-Schleife führt eine Anweisung durch, solang die Bedingung `true` ist
- Die `do ... while` Schleife führt erst eine Anweisung durch und prüft die Bedingung am Ende der Iteration
- Die Bedingung wird nach jeder Iteration erneut geprüft
- Es muss selbst gewährleistet werden, dass die Iteration nach gesetzter Bedingung abbricht → Sonst landet das Programm in dem Zustand einer Dauerschleife!

```
var zahlen = [0, 1, 2];
let n = zahlen.length;

while (n > 0) {
        //Abbruchbedingung
        n--;
        //Ausgabe; 2 1 0
        console.log(zahlen[n]);
}
```

```
var zahlen = [0, 1, 2];
let n = zahlen.length;

do {
        //Abbruchbedingung
        n--;
        //Ausgabe; 2 1
        console.log(zahlen[n]);
} while (n > 0);
```

Kontrollstrukturen & Schleifen

Kontrollstrukturen und Schleifen funktionieren in JavaScript ähnlich wie in Java

- Im Allgemeinen iteriert eine `for`-Schleifen mit einem Laufindex über eine Liste von Werten
- Zur Definition einer for-Schleife wird eine `Index-Variable`, eine `Abbruchbedingung` und eine `Inkrement/Dekrement Anweisung` benötigt

```
var zahlen = [1, 2, 3];
zahlen.name = "Zahlen eins bis drei";

for (var i = 0; i < zahlen.length; i++) {

    console.log(zahlen[i]);
}
```

Index-Variable | Abbruchbedingung | Inkrement-Anweisung

Kontrollstrukturen & Schleifen

Kontrollstrukturen und Schleifen funktionieren in JavaScript ähnlich wie in Java

- Neben den Standardschleifen gibt es weitere Schleifenarten, mit denen beispielswiese über Eigenschaften eines Objektes iteriert werden kann

```
var zahlen = [1, 2, 3];
zahlen.name = "Zahlen eins bis drei";
for (var i in zahlen) {
        console.log(i); // 0, 1, 2, name <- Keys
}
for (var i in zahlen) {
        console.log(zahlen[i]);} // 1, 2, 3, Zahlen eins…<- Werte
```

- Die `for … in`-Schleife kann über Strukturen mit `Enumerable`-Objekten iterieren
  - `Enumerable` sind abzählbare Objekte, welche mit einem String verifiziert werden

Kontrollstrukturen & Schleifen

Kontrollstrukturen und Schleifen funktionieren in JavaScript ähnlich wie in Java

- Die `for ... of` Schleife kann über Strukturen mit `iterator` iterieren z.B. `Arrays` oder `Strings`
- Der Laufindex wird automatisch bei jedem Durchlauf entsprechend zugewiesen

```
var zahlen = [1, 2, 3];

for (var i of zahlen) {
        console.log(i); // 1, 2, 3 <- Werte
}
```

Web-Engineering
Funktionen

JavaScript Funktionen

Wozu werden Funktionen genutzt?

- Wiederverwendung von Code
- Strukturierung des Programms
- Mit Ergebnissen einfach weiterarbeiten
- Nutzung von Namespaces
- Berechnungen mit verschiedenen Werten einfach zusammenfassen

```
function blub(a, b = 1) {

  let c = a * b;
  c -= a * a;
  c += a;
  return c;

}

console.log(blub(5)); // -15
```

JavaScript Funktionen

Funktionen können auf verschiedene Weise erstellt werden.

Über einen Funktionsausdruck:

```
var addition = function
(zahl1, zahl2){
       return zahl1 + zahl2;
}
```

Über den Konstruktor des Funktions-Objekts:

```
var addition =
new Function("zahl1", "zahl2",
"return zahl1 + zahl2");
```

Über eine Funktionsanweisung:

```
function addition
(zahl1, zahl2){
       return zahl1 + zahl2
;
}
```

Über eine Arrow-Funktionsanweisung:

```
var
addition = x => { return x + x }
```

JavaScript Funktionen

- Jedes Mal wenn eine Funktion aufgerufen wird, wird ein Objekt mit dem Namen *arguments* erzeugt
  - *arguments* enthält die Funktionsparameter
- Dieses Objekt ist kein richtiges Array, da keine Array-Methoden zur Verfügung stehen

```
function addiereAlle(){
        var ergebnis = 0;
        for (var i=0; i<arguments.length; i++){
                eregbnis += arguments[i];
        }
        return ergebnis;
}


//Funktionsaufruf
var summe = addiereAlle(1, 2, 3, 4); //Ergebnis: 10
```

JavaScript Funktionen

Es gibt seit ECMAScript 6 eine neue Möglichkeit, um eine beliebige Anzahl von Parametern als Array zu übergeben.

Dies ermöglichen die Rest Parameter Syntax

- Damit können 0 bis n Parameter als Array der Funktion übergeben werden
- Der Rest Parameter wird mit "…" eingeleitet und muss bei Benutzung immer als letzter Parameter gelistet werden
- Ersetzt das *arguments* Objekt
- Ist ein Array und es können daher Methoden wie z.B. `forEach` oder `map` darauf angewendet werden

```
function fun(...input) {
        let sum = 0;
        for (let i of input) {
                sum += i;
        }
        return sum;
}
console.log(fun(1, 2)); //3
console.log(fun(1, 2, 3)); //6
console.log(fun(1, 2, 3, 4, 5)); //1
5
```

Objektifizierung durch Funktionen

Im klassischen JavaScript können Funktionen objektifiziert werden

```
Mensch = function(vname, nname){
    this.vname = vname;
    this.nname = nname;
}
m1 = new Mensch("Dennis",
       "Ritchie");
m2 = new Mensch();
```

```
// Zuweisung nach if-Abfrage
Mensch_2 = function(vname, nname){
   if(vname) this.vname = vname;
   if(nname) this.nname = nname;
}
m3 = new Mensch_2("Linus",
       "Torvalds");
m4 = new Mensch_2();
```

```
console.log(m1);
// Mensch { vname: 'Dennis', nname: 'Ritchie' }
console.log(m2);
// Mensch { vname: undefined, nname: undefined }

console.log(m3);
// Mensch_2 { vname: 'Linus', nname: 'Torvalds' }
console.log(m4);
// Mensch_2 {} //hat die Properties vname & nname nicht
```

Web-Engineering
Fehlerbehandlung

Fehlerbehandlung

Fehlerbehandlung mittels `try`, `catch` **und** `finally`

Falls ein Fehler bei der Ausführung einer Operation auftritt, sollte das Programm dieses Problem behandeln ohne den Programmfluss zu unterbrechen.

- JavaScript nutzt dafür den `try-catch-finally` Ausdruck
  - `try` → wird versucht ausgeführt zu werden
  - `catch` → wird ausgeführt wenn `try` einen Fehler wirft
  - `finally` → wird in jedem Fall ausgeführt
- Der `try-catch-finally` Ausdruck kann in verschiedenen Formen genutzt werden
  - `try` … `catch`
  - `try` … `catch` … `finally`

Fehlerbehandlung

Fehlerbehandlung mittels `try`, `catch` und `finally`

In diesem Beispiel wird der Fehler (Ausgabe einer nicht deklarierten Variable) nicht abgefangen:

```
console.log(a); // ReferenceError: a is not defined
```

In diesem Beispiel wird der Fehler abgefangen:

```
try {
    console.log(a);
} catch (e) {
    console.log(e.message); // a is not defined
} finally {
    console.log('Abschließende Ausgabe');
}
```

- Es gibt in Javascript verschiedene integrierte Errortypen, unter anderem
  - `EvalError, RangeError, ReferenceError, SyntaxError, TypeError, URIError, AggregateError`

Fehlerbehandlung

Das `throw` Statement wird benutzt, um eine benutzerdefinierte Ausnahme zu werfen

Definition einer eigenen Exception:

```
function OwnExcept(message) {
    this.message = message;
    this.name = 'OwnException';
}
```

Ausgabe einer nicht deklarierten Variable:

```
try {
    if (typeof a === 'undefined') {
        throw new OwnExcept('a ist nicht definiert');
    }
} catch (e) {
    alert(e.name + ': ' + e.message);
    // OwnException: a ist nicht definiert
}
```