## WBE Zusammenfassung



## Zusammenfassung - WBE

## Contents

# JavaScript

# Grundlagen

## Web-Konsole

In JavaScript kann man die Web-Konsole ganz einfach mit dem Keyword «console» ansprechen.

| | |
|---|---|
| console.log(message) | Meldung in der Konsole loggen |
| console.clear() | Konsole löschen |
| console.trace(message) | Stack trace ausgeben |
| console.time() | Timer starten |
| console.timeEnd() | Timer stoppen |
| console.error(message) | stderr ausgeben |

Webseite für Konsolen-API: https://nodejs.org/api/console.html

## Datentypen

- Bekannte Datentypen wie string und int etc...
- Spezielle Zahlen
- Infinity (1/0)
- -Infinity
- NaN (0/0)
- Mit dem keyword typeof wird der Typ zurückgegeben

| | | |
|---|---|---|
| typeof 12 | // | 'number' |
| typeof(12) | // 'number' | |
| typeof 2 n | // | 'bigint' |
| typeof Infinity | // 'number' | |
| typeof NaN | // 'number' !! | |
| typeof 'number' | // 'string' | |

- Speziell Werte
- Null
- Undefined

Vergleich mit == oder ===

- == : Vergleich mit automatischer Typkonvertierung
- === : Vergleich ohne Typkonvertierung (o vorzuziehen)
- Ebenso: != und !==

## Variablenbindung

```
let width = 10
console.log(width * width)          // → 100

let answer = true, next = false
let novalue
console.log(novalue)                // → undefined
```

Var und const als keywords funktionieren auch.

```
switch (<ausdruck>) {
    case <wert1>:
    break
    default:
        ...
        break
}
```

```
if (<ausdruck>) \{
\} else \{

   }
```

```
for (let i=1; i<50; i*=2) {
    console.log(i)
}
4 // -> 1, -> 2, -> 4, > 8, > 16, -> 32
```

Funktionsdefinition

```
1 const square = function (x) {
    return x * x
}
4
5 console.log(square(12)) // -> 144
```

```
1 const square1 $=(x) \Rightarrow$ \{return $x * x$ \}
2 const square2 $=x=x^{*} x$
```

Objekte und Arrays

| Was | Objekt | Array |
|---|---|---|
| Art | Attribut-Wert-Paare | Sequenz von Werten |
| Literalnotation | werte = { a: 1, b: 2} | liste = [1, 2, 3] |
| Ohne Inhalt | werte = {} | liste = [] |
| Elementzugriff | werte[ä"] oder werte.a | liste[0] |

# Objekte

- Objekt Attribute sind dynamisch und können einfach erweitert werden:
- Objekt Attribute können auch einfach mit dem delete keyword entfernt werden.
- Mit in kann überprüft werden, ob ein Attribut existiert

```
let person = {
    name: "John Baker",
    age: 23,
    "exam results": [5.5, 5.0, 5.0, 6.0, 4.5]
}
```

```
    let obj = { message: "not yet implemented" }
    obj.ready = false
> obj
{ message: 'not yet implemented', ready: false }
> obj.attr
undefined
```

```
> let obj = { message: "ready", ready: true, tasks: 3 }
> delete obj.message
> obj.tasks = undefined
> obj
{ ready: true, tasks: undefined }
> "message" in obj
false
> "tasks" in obj
true
```

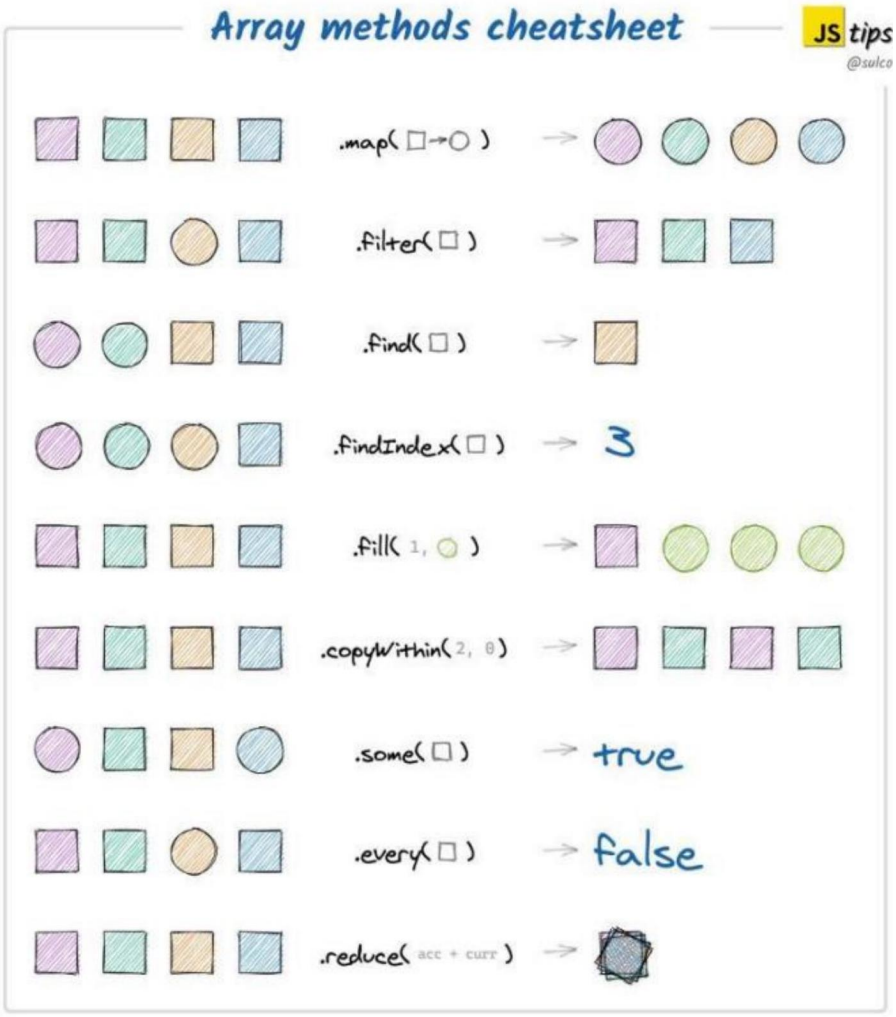
## Methoden

Ein Objekt kann auch Methoden enthalten:

```
> let cat = { type: "cat", sayHello: () => "Meow" }
> cat.sayHello
[Function: sayHello]
    > cat.sayHello()
'Meow'
```

Arrays
Verschiedene Hilfsfunktionen:

- Array.isArray()
- .push()
- .pop()
- Indexof, lastIndexOf
- Concat
- slice
- Shift, unshift
- .forEach(item => ....)
- .filter(item => .....)
- .map(item => ...)




```
// einfachere Variante für Arrays
for (let entry of myArray) {
  doSomethingWith(entry)
}
```

Json (JavaScript Object Notation)

- Daten-Austauschformat, nicht nur für JavaScript
- Orientiert an Notation für JavaScript-Objektliterale

```
> JSON.stringify({ type: "cat", name: "Mimi", age: 3})
'{"type":"cat", "name":"Mimi", "age":3}'
> JSON.parse('{"type":"cat","name":"Mimi","age":3}')
{ type: 'cat', name: 'Mimi', age: 3 }
```

## https://www.json.org/json-en.html

## Funktionen

- Funktionen sind spezielle, aufrufbare Objekte
- Man kann ihnen jederzeit Attribute oder Methoden hinzufügen
- Sie haben bereits vordefinierte Methoden

```
const add = (x, y) => x + y
add.doc = "This function adds two values"
    add(3,4)
7
    add.doc
'This function adds two values'
```

## Modulsystem in JavaScript

```
1 /* car-lib.js */
2}\mathrm{ const car = {
3 brand: 'Ford',
4 model: 'Fiesta'
5 }
6
7 module.exports = car
1 /* other js file */
2 const car = require('./car-lib')
```

## Prototypen von Objekten

- Die meisten Objekte haben ein Prototyp-Objekt.
- Dieses fungiert als Fallback für Attribute und Methoden.

```
> Object.getPrototypeOf(Math.max) == Function.prototype
true
> Object.getPrototypeOf([]) == Array.prototype
true
> Object.getPrototypeOf(Function.prototype) == Object.prototype
```

```
true
> Object.getPrototypeOf(Array.prototype) == Object.prototype
true
```

# PROTOTYPEN-KETTE

```
function Employee (name, salary) {
    Person.call(this, name)
    this.salary = salary
}
Employee.prototype = new Person()
Employee.prototype.constructor = Employee
let e17 = new Employee("Mary", 7000)
console.log(e17.toString()) /* -> Person with name 'Mary' */
console.log(e17.salary) /* -> 7000 */
```

Call, apply, bind

- Weitere Argumente von call : Argumente der Funktion
- Weiteres Argument von apply : Array mit den Argumenten
- Erzeugt neue Funktion mit gebundenem this

## Klassen

```
class Person {
    constructor (name) {
        this.name = name
    }
    toString () {
        return `Person with name '${this.name}'
    }
}
let p35 = new Person("John")
console.log(p35.toString()) // -> Person with name 'John'
```

## Vererbung

```
class Employee extends Person {
    constructor (name, salary) {
        super(name)
        this.salary = salary
    }
    toString () {
```

```
        return `${super.toString()} and salary ${this.salary}
    }
}
let e17 = new Employee("Mary", 7000);
console.log(e17.toString()) /* -> Person with name 'Mary' and salary 7000 */
console.log(e17.salary) /* -> 7000 */
```

Getter und Setter

```
class PartTimeEmployee extends Employee {
    constructor (name, salary, percentage) {
        super(name, salary)
        this.percentage = percentage
    }
    get salary100 () { return this.salary * 100 / this.percentage}
    set salary100 (amount) { this.salary = amount * this.percentage / 100 }
}
let e18 = new PartTimeEmployee("Bob", 4000, 50)
console.log(e18.salary100) /* -> 8000 */
e18.salary100 = 9000
console.log(e18.salary) /* \ 4500 */
```

Asynchrones Programmieren
File API
Mit require('fs') wird auf die File-Api zugegriffen.

## Datei-Informationen

```
const fs = require('fs')
fs.stat('test.txt' , (err, stats) => {
    if (err) {
    console.error(err)
    return
    }
    stats.isFile() /* true */
    stats.isDirectory() /* false */
    stats.isSymbolicLink() /* false */
stats.size /* 1024000 = ca 1MB */
})
```

## Pfade der Datei

Um Pfad-informationen einer Datei zu ermitteln muss man dies mit require('path') machen.

```
const path = require('path')
const notes = '/users/bkrt/notes.txt'
path.dirname(notes) /* /users/bkrt */
path.basename(notes) /* notes.txt */
path.extname(notes) /* .txt */
path.basename(notes, path.extname(notes)) /* notes */
```

## Lesen aus einer Datei

```
const fs = require('fs')
fs.readFile('/etc/hosts',"utf8", (err, data) => {
        if (err) throw err
        console.log(data)
})
```

## Dateien schreiben

```
1 const fs = require('fs')
2}\mathrm{ const content = 'Node was here!'
3 fs.writeFile('/Users/bkrt/test.txt', content, (err) => {
4 if (err) {
5 console.error(`Failed to write file: ${err}`)
6 return
7 }
8 ** file written successfully */
9 })
```

Weitere FS Funktionen

| Funktion | Bezeichnung |
|---|---|
| fs.access | Zugriff auf Datei oder Ordner prüfen |
| fs.mkdir | Verzeichnis anlegen |
| fs.readdir | Verzeichnis lesen, liefert Array von Einträgen |
| fs.rename | Verzeichnis umbenennen |
| fs.rmdir | Verzeichnis löschen |
| fs.chmod | Berechtigungen ändern |
| fs.chown | Besitzer und Gruppe ändern |
| fs.copyFile | Datei kopieren |
| fs.link | Besitzer und Gruppe ändern |
| fs.symlink | Symbolic Link anlegen |
| fs.watchFile | Datei auf Änderungen überwachen |

## Callbacks

Ein Callback ist eine Funktion, welche als Argument einer anderen Funktion übergeben wird und erst aufgerufen wird, wenn das Ereignis eingetreten ist. In der folgenden Abbildung wird die KlickFunktion vom Button mit der Id «Button» abonniert.

```
1 document.getElementById('button').addEventListener('click', () => {
2 //item clicked
3 })
```

## SetTimeout

- Mit setTimeout kann Code definiert werden, der zu einem späteren Zeitpunkt ausgeführt werden soll
- Eintrag in die Timer-Liste, auch wenn Zeit auf 0 gesetzt wird
- Kann mit clearTimeout entfernt werden

```
1 setTimeout(() => {
2 /* runs after 50 milliseconds */
3 }, 50)
```

## SetInterval

- Callback alle n Millisekunden in die Callback Queue eingefügt
- Kann mit clearInterval beendet werden

```
1 const id = setInterval(() => {
2 // runs every 2 seconds
3 }, 2000)
4 clearInterval(id)
```

## SetImmediate

```
1 setImmediate(() => {
2 console.log('immediate')
3 })
4
```

## Event-Modul (EventMitter)

- EventEmitter verwaltet Liste von Listeners zu bestimmten Events
- Listener für das Event können hinzugefügt oder entfernt werden

- Event kann ausgelöst werden → Listener werden informiert

## Listener hinzufügen

```
4 const EventEmitter = require('events')
5 const door = new EventEmitter()
6
7 door.on('open', () => \{
8 console.log('Door was opened')
9 \})
```

## Event auslösen

```
1 door.on('open', (speed) => \{
2 console.log(`Door was opened, speed: \$\{speed || 'unknown'\}`)
3 \})
4
5 door.emit('open')
6 door.emit('open', 'slow')
```

## Promises

Ist ein Platzhalter für einen Wert, der erst später voraussichtlich verfügbar sein wird.

```
Funktion mit Promise

function readFilePromise (file) {

    function readFilePromise (file) {
let promise = new Promise(
let promise = new Promise(
function resolver (resolve, reject) {
function resolver (resolve, reject) {
fs.readFile(file, ütf8", (err, data) => {
fs.readFile(file, ütf8", (err, data) => {
if (err) reject(err)
if (err) reject(err)
else resolve(data)
else resolve(data)
})
})
})
})
return promise
```

return promise
}

}

Gibt nun ein Promise-Object zurück

## Promise-Konstruktor erhlt resolver-Funktion

Rückgabe einer Promise: potentieller Wert kann später erfüllt oder zurückgewiesen werden

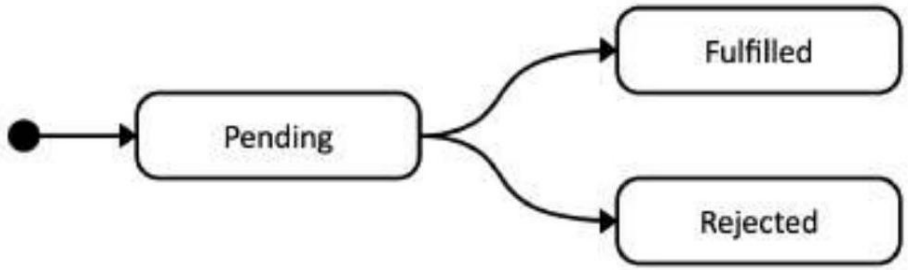
- Rückgabe einer Promise: potentieller Wert
- kann später erfüllt oder zurückgewiesen werden

Aufruf neu:

```
readFilePromise('/etc/hosts')
    .then(console.log)
    .catch(() => {
        console.log("Error reading file")
    })
```

## Promise-Zustände

- pending: Ausgangzustand
- fulfilled: erfolgreich abgeschlossen
- rejected: ohne Erfolg abgeschlossen
- Nur ein Zustandsübergang möglich
- Zustand in Promise-Objekt gekapselt



## Promises Verknüpfen

- Then-Aufruf gibt selbst Promise zurück
- Catch-Aufruf ebenfalls, per Default erfüllt
- So können diese Aufrufe verkettet werden
- Promise, welche unmittelbar resolved wird: Promise.resolve (...)
- Promise, welche unmittelbar rejected wird: Promise.reject (...)

## Promise.all()

- Erhält Array von Promises
- Erfüllt mit Array der Result, wenn alle erfüllt sind
- Zurückgewiesen sobald eine Promise zurückgewiesen wird

## Promise.race()

- Erhält Array von Promises
- Erfüllt sobald eine davon erfüllt ist
- Zurückgewiesen sobald eine davon zurückgewiesen wird

## ASYNC/AWAIT

Beispiel 1

```
/* Bekanntes Beispiel */
const readHosts =() => {
    readFilePromise('/etc/hosts')
        .then(console.log)
        . catch(() => {
            console.log("Error reading file")
        })
}
/* Mit async/await */
const readHosts = async () => {
    try {
        console.log(await readFilePromise('/etc/hosts'))
    }
    catch (err) {
        console.log("Error reading file")
    }
}
```

## Beispiel 2

```
function resolveAfter2Seconds (x) {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve(x)
        }, 2000)
    })
}
async function add1(x) {
    var a = resolveAfter2Seconds(20)
    var b = resolveAfter2Seconds(30)
    return x + await a + await b
}
add1(10).then(console.log)
```

## Webserver

Die Standard-Ports von einem Webserver sind 80 und 443. Der Webserver wartet auf eine Anfrage vom Client.

## Server im Internet

- Wartet auf Anfragen auf bestimmtem Port
- Client stellt Verbindung her und sendet Anfrage
- Server beantwortet Anfrage

| Port | Service |
|---|---|
| **20** | FTP - Data |
| **21** | FTP - Control |
| **22** | SSH Remote Login Protocol |
| **23** | Telnet |
| **25** | Simple Mail Transfer Protocol (SMTP) |
| **53** | Domain Name System (DNS) |
| **80** | HTTP |
| **443** | HTTPs |

File-Transfer (File Server)
Um Dateien auf einem File-Server auszutauschen, werden die Protokolle FTP (File Transfer Protocol) und SFTP (SSH File Transfer Protocol) verwendet.
HTTP
HTTP-Requests

| Methode | Beschreibung |
|---|---|
| GET | Ressource laden |
| POST | Information senden |
| PUT | Ressource anlegen, überschreiben |
| PATCH | Ressource anpassen |
| DELETE | Ressource löschen |

HTTP-Response Codes

| Code | Beschreibung |
|---|---|
| **1xx** | Information (101 Switching protocols) |
| **2xx** | Erfolg (200 OK) |
| **3xx** | Weiterleitung (301 Moved permanently) |
| **4xx** | Fehler in Anfrage (403 Forbidden, 404 Not Found) |
| **5xx** | Server-Fehler (501 Not implemented) |

Node.js Webserver

Einfacher Webserver

```
const {createServer} = require("http")
Let server = createServer((request, response) => {
    response.writeHead(200, {"Content-Type": "text/html"})
    response.write(‘
        <h1>Hello!</h1>
        <p>You asked for <code>${request.url}</code></p>‘)
    response.end()
})
server.listen(8000)
console.log("Listening! (port 8000)")
```

## Einfacher Webclient

```
onst {request} = require("http")
let requestStream = request({
    hostname: "eloquentjavascript.net",
        path: "/20_node.html",
        method: "GET"
        headers: {Accept: "text/html"}
}, response => {
        console.log("Server responded with status code", response.statusCode)
})
requestStream.end()
```

## Server und Client mit Streams

```
const {createServer} = require("http")
```

```
createServer((request, response) => {
    response.writeHead(200, {"Content-Type": "text/plain"})
        request.on("data", chunk =>
            response.write(chunk.toString().toUpperCase()))
        request.on("end" , () => response.end())
    }).listen(8000)
```

```
const \{request\} = require("http")
Let $\mathrm{rq}=$ request(\{
    hostname: "localhost",
    port: 8000,
    method: "POST"
\}, response => \{
    response.on("data", chunk =>
    process.stdout.write(chunk.toString()));
\})
rq.write("Hello server\n")
rq.write("And good bye\n")
rq.end()
```

# REST API

- REST: Representational State Transfer
- Zugriff auf Ressourcen über ihre Adresse (URI)
- Kein Zustand: jede Anfrage komplett unabhängig
- Kein Bezug zu vorhergehenden Anfragen
- Alle nötigen Informationen in Anfrage enthalten
- Verwenden der HTTP-Methoden: GET , PUT , POST , ...
  (WBE_Alles.pdf) Page 363.
  Express.js
  Express.Js ist ein minimales, aber flexibles Framework für Web-apps. Es hat zahlreiche Utilities und Erweiterungen. Express.js basiert auf Node.js.

# http://expressjs.com

# Installation

- Der Schritt npm init fragt eine Reihe von Informationen (Projektname, Version, ...) zum Projekt ab
- Als Entry Point ist hier index.js voreingestellt

- Das kann zum Beispiel in app.js geändert werden.

```
$ mkdir myapp
$ cd myapp
$ npm init
$ npm install express --save
```

## Beispiel: Express Server

```
const express = require('express')
const app = express()
const port = 3000
app.get('/', (req, res) => {
        res.send('Hello World!')
})
app.listen(port, () => {
    console.log(`Example app listening at http://localhost:${port}`)
}})
```

```
Routing
```

```
app.get('/', function (req, res) {
```

app.get('/', function (req, res) {
res.send('Hello World!')
res.send('Hello World!')
})
})
app.post('/', function (req, res) {
app.post('/', function (req, res) {
res.send('Got a POST request')
res.send('Got a POST request')
})
})
app.put('/user', function (req, res) {
app.put('/user', function (req, res) {
res.send('Got a PUT request at /user')
res.send('Got a PUT request at /user')
})
})
app.delete('/user', function (req, res) {
app.delete('/user', function (req, res) {
res.send('Got a DELETE request at /user')
res.send('Got a DELETE request at /user')
})

```
})
```

Jasmine (Testing)

## BEISPIEL (ZUGEHÖRIGE TESTS)

```
/* PlayerSpec.js - Auszug */
describe("when song has been paused", function() {
    beforeEach(function() {
        player.play(song)
        player.pause()
    })
    it("should indicate that the song is currently paused", function() {
        expect(player.isPlaying).toBeFalsy()
        /* demonstrates use of 'not' with a custom matcher */
        expect(player).not.toBePlaying(song)
    })
    it("should be possible to resume", function() {
        player.resume()
        expect(player.isPlaying).toBeTruthy()
        expect(player.currentlyPlayingSong).toEqual(song)
    })
})
```

## JASMINE: MATCHER

```
expect([1, 2, 3]).toEqual([1, 2, 3])
expect(12).toBeTruthy()
expect("").toBeFalsy()
expect("Hello planet").not.toContain("world")
expect(null).toBeNull()
expect(8).toBeGreaterThan(5)
expect(12.34).toBeCloseTo(12.3, 1)
expect("horse_ebooks.jpg").toMatch(/\w+.(jpg|gif|png|svg)/i)
```

## JASMINE: TESTS DURCHFÜHREN

$ npx jasmine
Randomized with seed 03741
Started
    5 specs, 0 failures
Finished in 0.014 seconds
Randomized with seed 03741 (jasmine –random=true –seed=03741)

## Browser-Technologien

Vordefinierte Objekte

- Die allgemeinen Objekte sind in JavaScript vordefiniert
- Tatsächlich handelt es sich um Funktionen/Konstruktoren
- Die Browser-Objekte existieren auf der Browser-Plattform
- Sie beziehen sich auf das Browser-Fenster, das angezeigte Dokument, oder den Browser selbst
  document
- Repräsentiert die angezeigte Webseit
- Einstieg ins DOM (Document Object Model)
- Diverse Attribute und Methoden, zum Beispiel:

```
1 document.cookie /* Zugriff auf Cookies */
2 document.lastModified /* Zeit der letzten Änderung */
3 document.links /* die Verweise der Seite */
4 |document.images /* die Bilder der Seite */
```

window

- Repräsentiert das Browserfenster
- Zahlreiche Attribute und Methoden, u.a.:
- Alle globalen Variablen und Methoden sind hier angehängt
- Neue globale Variablen landen ebenfalls hier

```
1 window.document /* Zugriff auf Dokument */
2 window.history /* History-Objekt */
3 window.innerHeight /* Höhe des Viewports */
4 window.pageYOffset /* vertikal gescrollte Pixel */
5 window.alert === alert /* -> true */
6 window.setTimeout === setTimeout /* -> true */
7 window.parseInt === parseInt /* true */
```

## navigator

Konsolen-eingabe auf dem folgenden Bild:

```
> navigator.userAgent
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:91.0) Gecko/20100101 Firefox/91.0"
> navigator.language
"de"
> navigator.platform
"MacIntel"
> navigator.onLine
true
location
> location.href
"https://gburkert.github.io/selectors/"
> location.protocol
"https:"
> document.location.protocol
"https:"
```

## Document Object Model

- Element erzeugen: document.createElement
- Attribute erzeugen: document.createAttribute
- Und hinzufügen: .setAttribute
- Element in Baum einfügen: .appendChild

Element auffinden

```
1 let aboutus = document.getElementById("aboutus")
2 let aboutlinks = aboutus.getElementsByTagName("a")
3 let aboutimportant = aboutus.getElementsByClassName("important")
4 let navlinks = document.querySelectorAll("nav a")
```

## Textknoten erzeugen

```
<p>The <img src="img/cat.png" alt="Cat"> in the
<img src="img/hat.png" alt="Hat">.</p>
<p><button onclick="replaceImages()">Replace</button></p>
<script>
    function replaceImages () {
        let images = document.body.getElementsByTagName("img")
        for (let i = images.length - 1; i >= 0; i--) {
                let image = images[i]
                if (image.alt) {
                let text = document.createTextNode(image.alt)
                image.parentNode.replaceChild(text, image)
```

```
            }
        }
    }
</script>
```

Elementknoten erzeugen

```
<blockquote id="quote">
    No book can ever be finished. While working on it we learn ...
</blockquote>
<script>
/* definition of elt ... */
document.getElementById("quote").appendChild(
    elt("footer", "-",
    elt("strong", "Karl Popper"),
    ", preface to the second edition of ",
    elt("em", "The Open Society and Its Enemies"),
    ", 1950"))
</script>
```

Attribut setzen

```
1
   6
   8

3 let att = document.createAttribute("class")
4 att.value ="democlass"
5 h1.setAttributeNode(att)
7 /* oder kürzer: */
let h1 = document.querySelector("h1")
,h1.setAttribute("class","democlass")
```

Style anpassen

```
1 Nice text
2
```

## Event handling

Event abonnieren/entfernen
Mit der Methode addEventListener() wird ein Event abonniert. Mit removeEventListener kann das Event entfernt werden.

```
<button>Act-once button</button>
<script>
    let button = document.querySelector("button")
    function once () {
```

```
        console.log("Done.")
        button.removeEventListener("click", once)
    }
    button.addEventListener("click", once)
</script>
```

Wenn ein Parameter zur Methode hinzugefügt wird, wird dieses als das Event-Objekt gesetzt.

```
<script>
    let button = document.querySelector("button")
    button.addEventListener("click", (e) => {
        console.log("x="+e.x+", y="+e.y)
    })
</script\
```

Das Event wird bei allen abonnierten Handlern ausgeführt bis ein Handler stopPropagation() ausführt.

```
<script>
    let button = document.querySelector("button")
    button.addEventListener("click", (e) => {
                console.log("x="+e.x+", y="+e.y)
                e.stopPropagation()
    })
</script>
```

Viele Ereignisse haben ein Default verhalten. Eigene Handler werden vor Default-Verhalten ausgeführt. Um das Default-Verhalten zu verhindern, muss die Methode preventDefault() ausgeführt werden.

```
<a href="https://developer.mozilla.org/">MDN</a>
<script>
    let link = document.querySelector("a")
    link.addEventListener("click", event => {
            console.log("Nope.")
                event.preventDefault()
        })
,/script>
```

## Tastatur-Events

- keydown
- keyup
- Achtung: bei keydown kann das event mehrfach ausgelöst werden

```
<p>Press Control-Space to continue.</p>
<script>
    window.addEventListener("keydown", event => {
            if (event.key ==" " && event.ctrlKey) {
                console.log("Continuing!")
            }
    })
</script>
```

Mauszeiger-Events

- Mausklicks:
- mousedown
- mouseup
- click
- dblclick
- Mausbewegung
- mousemove
- Touch-display
- touchstart
- touchmove
- touched

## Scroll-Events

Das Scrollevent hat die Attribute des Event-Objekts: pageYOffset, pageXOffset.

```
window.addEventListener("scroll", () => {
    let max = document.body.scrollHeight - innerHeight
    bar.style.width = `${(pageYOffset / max) * 100}%`
,})
```

Fokus- und Ladeereignisse

- Fokus erhalten / verlieren
- focus
- blur
- Seite wurde geladen (ausgelöst auf window und document.body)

- load
- beforeunload

Jquery
JQuery ist eine freie JavaScript-Bibliothek, die Funktionen zur DOM-Navigation und -Manipulation zur Verfügung stellt.

```
$("button.continue").html("Next Step...")
var hiddenBox = $("#banner-message")
$("#button-container button").on("click", function(event) {
        hiddenBox.show()
    .})
```

| Aufruf | Bedeutung | |
|---|---|---|
| $(Funktion) | DOM ready | |
| $("CSS Selektor") .aktion(arg1, ....) .aktion(...) | Wrapped Set | |
| | - Knoten, die Sel. erfüllen | |
| | - eingepackt in jQuery Obj. | |
| | | `$(".toggleButton").at` |
| $("HTML-Code") | Wrapped Set<br>- neuer Knoten | |
| | - eingepackt in jQuery Obj.<br>- noch nicht im DOM | |
| $(DOM-Knoten) | Wrapped Set | |
| | - dieser Knoten<br>- eingepackt in jQuery Obj. | |

## Web-Grafiken

- Einfache Grafiken mit HTML und CSS möglich
- Zum Beispiel: Balkendiagramme
- Alternative für Vektorgrafiken: SVG
- Alternative für Pixelgrafiken: Canvas

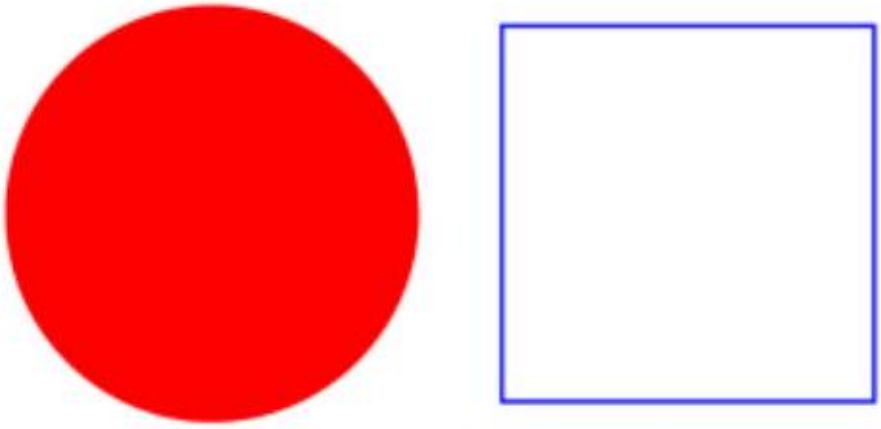
SVG

- Basiert wie HTML auf XML
- Elemente repräsentieren grafische Formen
- Ins DOM integriert und durch Scripts anpassbar

Beispiel:

```
p>Normal HTML here.</p>
<svg xmlns="http://www.w3.org/2000/svg">
    <circle r="50" cx="50" cy="50" fill="red"/>
    <rect x="120" y="5" width="90" height="90" stroke="blue" fill="none"/>
</svg>
```

Ausgabe:
Normal HTML here.



JavaScript:

```
1 let circle = document.querySelector("circle")
2 circle.setAttribute("fill","cyan")
```

## Canvas

- Element canvas als Zeichenbereich im Dokument
- API zum Zeichnen auf dem Canvas

```
<canvas></canvas>
<script>
    Let cx = document.querySelector("canvas").getContext("2d")
    cx.beginPath()
    cx.moveTo(50, 10)
    cx.lineTo(10, 70)
    cx.lineTo(90, 70)
    cx.fill()
    let img = document.createElement("img")
    img.src = "img/hat.png"
    img.addEventListener("load" , () => {
        for (let x = 10; x < 200; x += 30) {
```

```
            cx.drawImage(img, x, 10)
        }
    })
</script>
```

Canvas Methoden

| Methoden | Beschreibung |
|---|---|
| scale | Skalieren |
| translate | Koordinatensystem verschieben |
| rotate | Koordinatensystem rotieren |
| save | Transformationen auf Stack speichern |
| restore | Letzten Zustand wiederherstellen |

Browser-API
Web Storage
Web Storage speichert Daten auf der Seite des Client.

## Local Storage

Local Storage wird verwendet, um Daten der Webseite lokal abzuspeichern. Die Daten bleiben nach dem Schliessen des Browsers erhalten. Die Daten sind in Developer Tools einsehbar und änderbar.

Die Daten werden nach Domains abgespeichert. Es können pro Webseite etwa 5MB abgespeichert werden.

```
1 localStorage.setItem("username","bkrt")
2 console.log(localStorage.getItem("username")) // -> bkrt
3 localStorage.removeItem("username")
```

Die Werte werden als Strings gespeichert, daher müssen Objekte mit JSON codiert werden:
1 Let user = {name: "Hans", highscore: 234}
2 localStorage.setItem(JSON.stringify(user))

## History

History gibt zugriff auf den Verlauf des akutellen Fensters/Tab.

```
1 function goBack() {
2 window.history.back();
3
    ,}
```

| Methoden | Beschreibung |
|---|---|
| length (Attribut) | Anzahl Einträgte inkl. aktueller Seite. Keine Methode! |
| back | zurück zur letzten Seite |

GeoLocation

Mit der GeoLocation-API kann der Standort abgefragt werden.

```
var options = { enableHighAccuracy: true, timeout: 5000, maximumAge: 0 }
function success(pos) {
    var crd = pos.coords
    console.log(‘Latitude : ${crd.latitude}‘)
    console.log(‘Longitude: ${crd.longitude}‘)
    console.log(‘More or less ${crd.accuracy} meters.‘)
}
function error(err) { ... }
navigator.geolocation.getCurrentPosition(success, error, options)
```

## Formulare

Formulare ermöglichen Benutzereingaben. Sie gilt als Grundlade für Interaktion mit dem Web.
Input types:

- submit, number, text, password, email, url , range , date , search , color

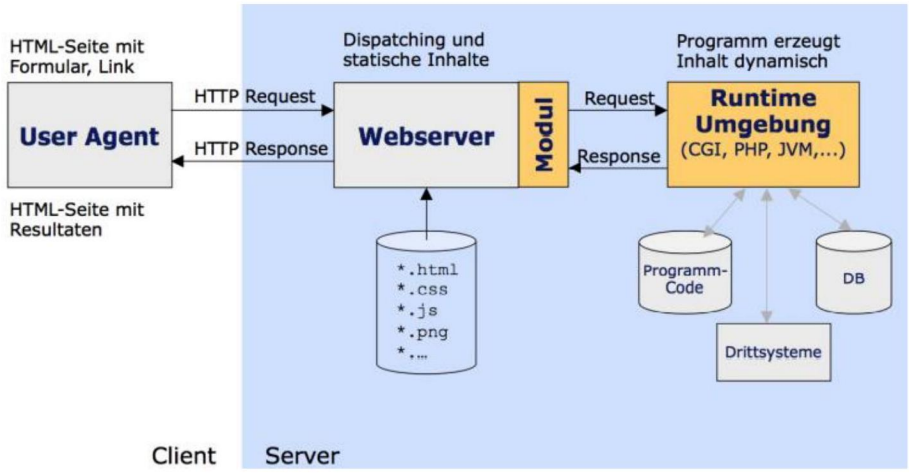
```
<form>
    <fieldset>
        <legend>General information</legend>
        <label>Text field <input type="text" value="hi"></label>
        <label>Password <input type="password" value="hi"></label>
        <label class="area">Textarea <textarea>hi</textarea></label>
    </fieldset>
    <fieldset>
        <legend>Additional information</legend>
        <label>Checkbox <input type="checkbox"></label>
        <label>Radio button <input type="radio" name="demo" checked></label>
        <label>Another one <input type="radio" name="demo"></label>
    </fieldset>
    <form>
    <label>Button <button>Click me</button></label>
    <label>Select menu
    <select name="cars">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
    </select>
    </label>
    <input type="submit" value="Send">
</form>
```

| ’


Formulare können auch POST/GET Aktionen ausführen:
Action beschreibt das Skript, welches die Daten annimmt. Method ist die Me-

thode die ausgeführt wird.

```
<form action="/login" method="post">
2 ...
3 </form>
```




Formular Events

| Events | Beschreibung |
|---|---|
| **change** | Formularelement geänder |
| **input** | Eingabe in Textfeld |
| **submit** | Formular absenden |

GET/POST-Methode

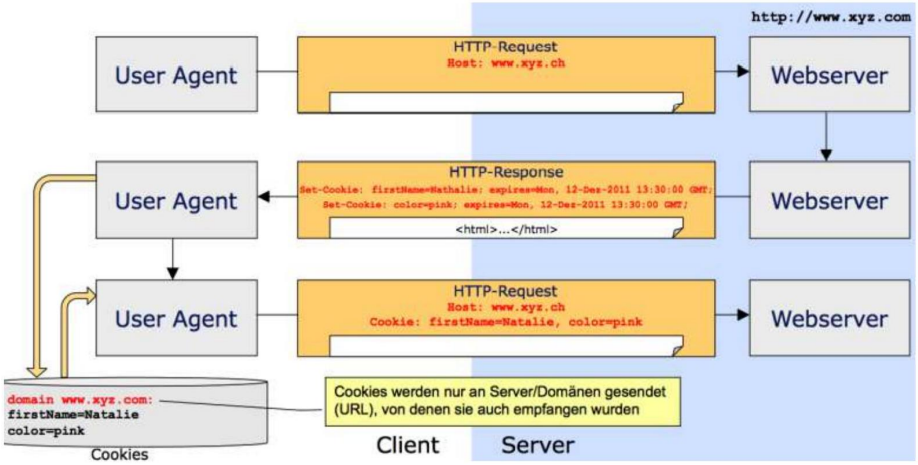
```
1  <form action="http://localhost/cgi/showenv.cgi"method="get">
2      <fieldset>
3          <legend>Login mit GET</legend>
4          <label for="login_get">Benutzername:</label>
5          <input type="text" id="login_get" name="login"/>
6          <label for="password_get">Passwort:</label>
7          <input type="password" id="password_get" name="password"/>
8          <label for="submit_get"></label>
9          <input type="submit" id="submit_get" name="submit" value="Anmelden" />
10     </fieldset>
11 </form>
```

# Cookies und Sessions

Cookies

- HTTP als zustandsloses Protokoll konzipiert
- Cookies: Speichern von Informationen auf dem Client
- Response: Set-Cookie -Header, Request: Cookie -Header

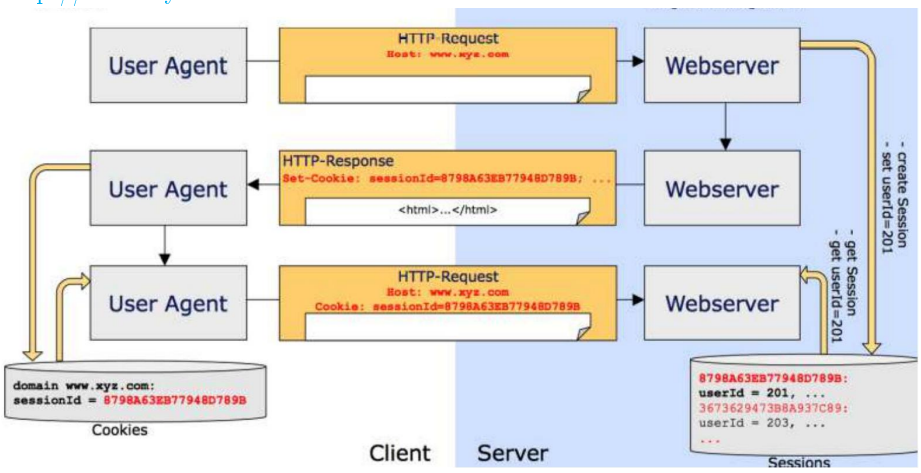
- Zugriff mit JavaScript möglich (ausser HttpOnly ist gesetzt)




## Sessions

- Cookies auf dem Client leicht manipulierbar
- Session: Client-spezifische Daten auf dem Server speichern
- Identifikation des Clients über Session-ID (Cookie o.a.)
- Gefahr: Session-ID gerät in falsche Hände (Session-Hijacking)

Ablauf:
http://www.xyz.com



## Fetch API

- HTTP-Requests von JavaScripts
- Geben Promise zurück
- Nach Server-Antwort erfüllt mit Response-Objekt

```
fetch("example/data.txt")
.then(response => {
            console.log(response.status) // -> 200
    console.log(response.headers.get("Content-Type")) // -> text/plain
})
.then(resp => resp.text())
.then(text => console.log(text))
// -> This is the content of data.txt
```

Response Objekt

- headers : Zugriff auf HTTP-Header-Daten Methoden get, keys, forEach , ...
- status: Status-Code
- json() : liefert Promise mit Resultat der JSON-Verarbeitung
- text() : liefert Promise mit Inhalt der Server-Antwort

## UI-Bibliothek

Dom-Scripting und Abstraktionen
JSX und SJDON
SuiWeb