WBE: BROWSER-TECHNOLOGIEN

JAVASCRIPT IM BROWSER (TEIL 2)

ÜBERSICHT

- Event Handling
- Kleiner Exkurs: jQuery
- Bilder und Grafiken
- Weitere Browser-APIs

ÜBERSICHT

- Event Handling
- Kleiner Exkurs: jQuery
- Bilder und Grafiken
- Weitere Browser-APIs

EVENT HANDLING IM BROWSER

- Im Browser können Event Handler registriert werden
- Methode: `addEventListener`
- Erstes Argument: Ereignistyp
- Zweites Argument: Funktion, die beim Eintreten des Events aufgerufen werden soll

```
<p>Click this document to activate the handler.</p>
<script>
  window.addEventListener("click", () => {
    console.log("You knocked?")
  })
</script>
```

EVENT REGISTRIEREN

- Events können auch an DOM-Elementen registriert werden
- Nur Ereignisse, die im Kontext dieses Elements auftreten, werden dann berücksichtigt

```
<button>Click me</button>
<p>No handler here.</p>
<script>
  let button = document.querySelector("button")
  button.addEventListener("click", () => {
    console.log("Button clicked.")
  })

  /* oder: button.onclick = () => {...} */
</script>
```

HANDLER ENTFERNEN

- Entfernen von Handlern mit `removeEventListener`
- Beispiel – Ereignis nur einmal behandeln:

```
<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>
```

EVENT-OBJEKT

- Nähere Informationen über das eingetretene Ereignis
- Wird dem Event Handler automatisch übergeben
- Je nach Ereignistyp verschiedene Attribute
- Bei Mouse Events z.B. x und y (Koordinaten)

```
<script>
  let button = document.querySelector("button")
  button.addEventListener("click", (e) => {
    console.log("x="+e.x+", y="+e.y)
  })
  // z.B.: x=57, y=14
</script>
```

https://developer.mozilla.org/en-US/docs/Web/API/Event

EVENT-OBJEKT

```
window.addEventListener("mousedown", console.log)
```

EVENT-WEITERLEITUNG

- Ereignisse werden für Knoten im DOM-Baum registriert
- Reagieren auch, wenn Ereignis an untergeordnetem Knoten auftritt
- Alle Handler nach oben bis zur Wurzel des Dokuments ausgeführt
- Bis ein Handler `stopPropagation()` auf dem Event-Objekt aufruft

```
<p>A paragraph with a <button>button</button>.</p>

<script>
  document.querySelector("p").addEventListener("mousedown", () => {
    console.log("Handler for paragraph.")
  })
  document.querySelector("button").addEventListener("mousedown", event => {
    console.log("Handler for button.")
    if (event.button == 2) event.stopPropagation()
  })
</script>
```

EVENT-WEITERLEITUNG

- Element, bei welchem das Ereignis ausgelöst wurde: `event.target`
- Element, bei welchem das Ereignis registriert wurde: `event.currentTarget`

```
<button>A</button>
<button>B</button>
<button>C</button>
<script>
  document.body.addEventListener("click", event => {
    if (event.target.nodeName == "BUTTON") {
      console.log("Clicked", event.target.textContent)
    }
  })
</script>
```

DEFAULT-VERHALTEN

- Viele Ereignisse haben ein Default-Verhalten
- Beispiel: auf einen Link klicken
- Eigene Handler werden vorher ausgeführt
- Aufruf von `preventDefault()` auf Event-Objekt verhindert Default-Verhalten

```
<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-EREIGNISSE

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

- Ereignisse `keydown` und `keyup`
- Modifier-Tasten als Attribute des Event-Objekts
- Achtung: `keydown` kann bei längerem Drücken mehrfach auslösen

ZEIGER-EREIGNISSE

- Mausklicks:
  `mousedown`, `mouseup`, `click`, `dblclick`
- Mausbewegung:
  `mousemove`
- Berührung (Touch-Display):
  `touchstart`, `touchmove`, `touchend`

SCROLL-EREIGNISSE

- Ereignis-Typ: `scroll`
- Attribute des Event-Objekts: `pageYOffset`, `pageXOffset`

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

https://eloquentjavascript.net/15_event.html#h_xGSp7W5DAZ

FOKUS- UND LADE-EREIGNISSE

- Fokus erhalten/verlieren: `focus`, `blur`
- Seite wurde geladen: `load`
  - Ausgelöst auf `window` und `document.body`
  - Elemente mit externen Ressourcen (`img`) unterstützen ebenfalls `load`-Events
  - Bevor Seite verlassen wird: `beforeunload`
- Diese Ereignisse werden nicht propagiert

BEISPIEL: BILD LADEN

- Funktion `loadImage` soll Bild laden
- Callbacks für Erfolg und Fehler

```
1  loadImage('zhaw.png',
2
3    function onsuccess (img) {
4      document.body.appendChild(img)
5    },
6
7    function onerror (e) {
8      console.log('Error occured while loading image')
9      console.log(e)
10   }
11
12 )
```

BEISPIEL: BILD LADEN

```
function loadImage (url, success, error) {
  var img = new Image()
  img.src = url

  img.onload = function () {
    success(img)
  }

  img.onerror = function (e) {
    error(e)
  }
}
```

BEISPIEL: MIT PROMISE

```
1  function loadImage (url) {
2    var promise = new Promise(
3      function (resolve, reject) {
4        var img = new Image()
5        img.src = url
6
7        img.onload = function () {
8          resolve(img)
9        }
10
11       img.onerror = function (e) {
12         reject(e)
13       }
14     }
15   )
16   return promise
17 }
```

BEISPIEL: MIT PROMISE

```
1 loadImage('zhaw.png')
2   .then(function (img) {
3     document.body.appendChild(img)
4   })
5   .catch(function (e) {
6     console.log('Error occured while loading image')
7     console.log(e)
8   })
```

VERZÖGERTES BEARBEITEN

- Bestimmte Ereignisse in schneller Folge ausgelöst
- Zum Beispiel: `mousemove`, `scroll`
- Ereignisbearbeitung auf Wesentliches reduzieren
- Oder jeweils mehrere Ereignisse zusammenfassen

```
<textarea>Type something here...</textarea>
<script>
  let textarea = document.querySelector("textarea")
  let timeout
  textarea.addEventListener("input", () => {
    clearTimeout(timeout)
    timeout = setTimeout(() => console.log("Typed: " + textarea.value), 500)
  })
</script>
```

ANIMATION

- Anpassen zum Beispiel der `position`-Eigenschaft
- Synchronisieren mit der Browser-Anzeige: `requestAnimationFrame`

```
function animate (time, lastTime) {
  /* calculate new position */
  /* ... */
  requestAnimationFrame(newTime => animate(newTime, time))
}
requestAnimationFrame(animate)
```

ÜBERSICHT

- Event Handling
- Kleiner Exkurs: jQuery
- Bilder und Grafiken
- Weitere Browser-APIs

jQuery

- DOM-Scripting ist oft mühsam
- Grund: unübersichtliche, inkonsistente API
- Abhilfe für lange Zeit: jQuery
  - DOM-Element mit CSS-Selektor auswählen
  - Einfache Anpassungen am DOM
  - Asynchrone Serverzugriffe (Ajax)

JQUERY: DOM UND EVENTS

```
$("button.continue").html("Next Step...")

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

- `$(<selector>)` erzeugt jQuery Objekt, das eine Sammlung von DOM-Elementen enthält
- Darauf sind zahlreiche Methoden anwendbar
- DOM-Traversal und -Manipulation sehr einfach

https://api.jquery.com
http://jqapi.com

JQUERY: ÜBERBLICK

Aufruf	Bedeutung	Beispiel
$(*Funktion*)	DOM ready	$(function() { .... });
$("*CSS Selektor*") .aktion(arg1, .,..) .aktion(...)	Wrapped Set - Knoten, die Sel. erfüllen - eingepackt in jQuery Obj.	$(".toggleButton").attr("title") $(".toggleButton").attr("title", "click here") $(".toggleButton").attr({title : "click here", ...}) $(".toggleButton").attr("title", function(){...}) .css(...) .text(...) .on("click", function(event) { ...})
$("*HTML-Code*")	Wrapped Set - neuer Knoten - eingepackt in jQuery Obj. - noch nicht im DOM	$(" ... ").addClass(...) .appendTo("*Selektor*") $(" ... ").length $(" ... ")[0]
$(DOM-Knoten)	Wrapped Set - dieser Knoten - eingepackt in jQuery Obj.	$(document.body) $(this)

JQUERY: BEDEUTUNG ABNEHMEND

- DOM-Element mit CSS-Selektor auswählen
  → `querySelector`, `querySelectorAll`

- Einfache Anpassungen am DOM
  → mit Framworks wie React.js weniger nötig

- Asynchrone Serverzugriffe (Ajax)
  → Fetch API (spätere Lektion)

ÜBERSICHT

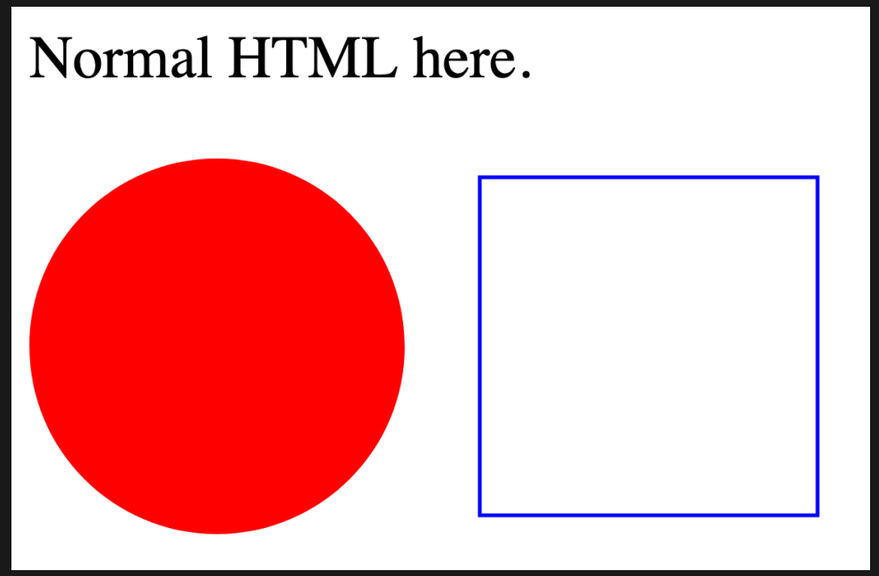
- Event Handling
- Kleiner Exkurs: jQuery
- Bilder und Grafiken
- Weitere Browser-APIs

WEB-GRAFIKEN

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

SVG

```
<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>
```



SVG

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

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

```
▼ children:                    HTMLCollection { 0: circle , 1: rect , length: 2 }
  ▼ 0:                         <circle r="50" cx="50" cy="50" fill="cyan">
      assignedSlot:            null
    ▼ attributes:              NamedNodeMap(4) [ r="50", cx="50", cy="50", … ]
      ▶ 0:                     r="50"
      ▶ 1:                     cx="50"
      ▶ 2:                     cy="50"
      ▶ 3:                     fill="cyan"
        length:                4
```

CANVAS

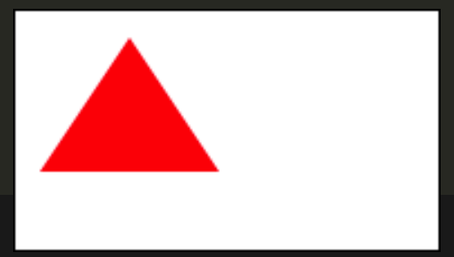
- Element `canvas` als Zeichenbereich im Dokument
- API zum Zeichnen auf dem Canvas

```
<p>Before canvas.</p>
<canvas width="120" height="60"></canvas>
<p>After canvas.</p>
<script>
  let canvas = document.querySelector("canvas")
  let context = canvas.getContext("2d")
  context.fillStyle = "red"
  context.fillRect(10, 10, 100, 50)
</script>
```

CANVAS: PFADE

```
<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()
</script>
```



CANVAS: WEITERE MÖGLICHKEITEN (1)

- Quadratische Kurven: `quadraticCurveTo`
- Bezier-Kurven: `bezierCurveTo`
- Kreisabschnitte: `arc`
- Text: `fillText`, `strokeText` (und: `font`-Attribut)
- Bild: `drawImage`

CANVAS: BILD EINFÜGEN

```
<canvas></canvas>

<script>
  let cx = document.querySelector("canvas").getContext("2d")
  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: WEITERE MÖGLICHKEITEN (2)

- Skalieren: `scale`
- Koordinatensystem verschieben: `translate`
- Koordinatensystem rotieren: `rotate`
- Transformationen auf Stack speichern: `save`
- Letzten Zustand wiederherstellen: `restore`

HTML, SVG, CANVAS

HTML

- einfach
- Textblöcke mit Umbruch, Ausrichtung etc.

SVG

- beliebig skalierbar
- Struktur im DOM abgebildet
- Events auf einzelnen Elementen

Canvas

- einfache Datenstruktur: Ebene mit Pixeln
- Pixelbilder verarbeiten

ÜBERSICHT

- Event Handling
- Kleiner Exkurs: jQuery
- Bilder und Grafiken
- Weitere Browser-APIs

WEB STORAGE

- Speichern von Daten clientseitig
- Einfache Variante: Cookies (s. spätere Lektion)
- Einfache Alternative: LocalStorage
- Mehr Features: IndexedDB (nicht Stoff von WBE)

LOCALSTORAGE

```
localStorage.setItem("username", "bkrt")
console.log(localStorage.getItem("username"))   // → bkrt
localStorage.removeItem("username")
```

- Bleibt nach Schliessen des Browsers erhalten
- In Developer Tools einsehbar und änderbar
- Alternative solange Browser/Tab geöffnet: `sessionStorage`

LOCALSTORAGE

- Gespeichert nach Domains
- Limit für verfügbaren Speicherplatz pro Website (~5MB)
- Attributwerte als Strings gespeichert
- Konsequenz: Objekte mit JSON codieren

```
let user = {name: "Hans", highscore: 234}
localStorage.setItem(JSON.stringify(user))
```

HISTORY

- Zugriff auf den Verlauf des aktuellen Fensters/Tabs
- `length`: Anzahl Einträge inkl. aktueller Seite
- `back()`: zurück zur letzten Seite
- Seit HTML5 mehr Kontrolle über den Verlauf:

`pushState()`, `replaceState()`, `popstate`-Event

```
function goBack () {
  window.history.back()
}
```

MDN: Working with the History API

GEOLOCATION

```
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)
```

MDN: Geolocation API

WEB WORKERS

- Laufen parallel zum Haupt-Script
- Ziel: aufwändige Berechnungen blockieren nicht die Event Loop

```
// squareworker.js
addEventListener("message", event => {
  postMessage(event.data * event.data)
})

// main script
let squareWorker = new Worker("code/squareworker.js")
squareWorker.addEventListener("message", event => {
  console.log("The worker responded:", event.data)
})
squareWorker.postMessage(10)
squareWorker.postMessage(24)
```

QUELLEN

- Marijn Haverbeke: Eloquent JavaScript, 3rd Edition
  https://eloquentjavascript.net/
- Ältere Slides aus WEB2 und WEB3

LESESTOFF

Geeignet zur Ergänzung und Vertiefung

- Kapitel 15 und 17 von:
  Marijn Haverbeke: Eloquent JavaScript, 3rd Edition
  https://eloquentjavascript.net/

Stand:
27.7.2025 15:31