# HTML5-Tutorium: JavaScript: Hello World 03

Wechseln zu:Navigation, Suche

Dieser Artikel erfüllt die GlossarWiki-Qualitätsanforderungen **nur teilweise**:

| | | | | |
|---|---|---|---|---|
| **Korrektheit**: 3 (zu größeren Teilen überprüft) | **Umfang**: 4 (unwichtige Fakten fehlen) | **Quellenangaben**: 3 (wichtige Quellen vorhanden) | **Quellenarten**: 5 (ausgezeichnet) | **Konformität**: 3 (gut) |

**Vorlesung MMProg**

Inhalt | Teil 1 | Teil 2 | Teil 3 | Teil 4

**Musterlösung**: `index.html`, `index2.html` (Git-Repository)

## Inhaltsverzeichnis



# 1 Anwendungsfälle (Use Cases)

Gegenüber dem zweiten Teil des Tutoriums ändern sich die die Anwendungsfälle deutlich. Es soll nun nicht mehr die ganze Welt begrüßt werden, sondern der Benutzer, der die Web-Anwendung gestartet hat. Dazu muss er zunächst nach seinem Namen gefragt werden. Anschließend wird das HTML-Dokumnt mit Hilfe von JavaScript umgestaltet: Das Eingabeformular wird ausgeblendet und stattdessen wird die Begrüßungsformel angezeigt.

# 2 Erstellen eines neuen Projektes

Erstellen Sie ein neues Projekt `HelloWorld03`.

Kopieren Sie anschließend die Projektdateien (insbesondere den Ordner `web` mit den Dateien `index.html` und `main.css` sowie die Datei `.gitignore`) aus dem zweiten Teil des Tutoriums und passen Sie den Titel in der HTML-Datei an.

Sofern sich die Datei `.gitignore` noch nicht in Ihrem HelloWorld02-Projekt befindet, sollten Sie die Datei https://glossar.hs-augsburg.de/beispiel/tutorium/2018/.gitignore in das Wurzelverzeichnis Ihres Web-Projektes kopieren. Speichern Sie sie unbedingt unter dem Namen .gitignore (mit einem Punkt

als erstes Zeichen). Diese Datei enthält die Namen und Endungen zahlreicher Dateien und Ordner, die üblicherweise nicht auf einem Git-Server gespeichert werden sollten.

Erstellen Sie für Ihr Projekt ein Git-Repository, committen Sie und pushen Sie es dann auf den Git-Server.

# 3 Single-Page-Web-Anwendung

Die Anwendung wird als Single-Page-Web-Anwendung (Onepager) realisiert. Das HTML-Dokument `index.html` enthält zwei Abschnitte (sections), eines mit einem Formular zur Eingabe des Namens und ein zweites zur Begrüßung des Benutzers, nachdem er seinen Namen eingegeben hat.

Für jede Seite fügen wir in `index.html` einen HTML-Abschnitt „`<section id="..."> ...</section>`“ ein. Das Section-Element gruppiert einen logischen Abschnitt oder ein Kapitel eines HTML-Dokuments. Es soll laut Spezifikation eine Überschrift enthalten, die das Thema des Abschnitts beschreibt. Man hätte auch ein Div-Element „`<div id="..."> ...</div>`“ anstelle des Section-Elements verwenden können. Ein Div-Element hat keinerlei Semantik (Bedeutung), es dient lediglich der Strukturierung einer HTML-Datei. Die (spezifikationsgemäße) Verwendung von Section- und Article-Elementen, die in HTML5 eingeführt wurden, hat den Vorteil, dass Browser deren Bedeutung kennen und daher geeignete Defaultstyles verwenden können, falls der Entwickler keine entsprechenden Styles angibt. Ein Beispiel sind Browser für Blinde. Derartige Browser lesen die Inhalte entweder vor oder geben sie textuell über eine sogenannte Braillezeile aus. Für derartige Browser werden meist keine CSS-Layout-Vorgaben gemacht (obwohl dies problemlos möglich wäre) und daher ist es wichtig, Strukturelementen eine Semantik zuzuordnen. Ein Blindenbrowser könnte beim Vorlesen eines Dokuments beispielsweise stets das Wort „Kapitel“ vor eine H1-Überschrift einfügen, die als erstes Element innerhalb eines Section-Elements steht.

```
<body>
  <section id="section_form">
    <h1>Hello, Stranger!</h1>
  </section>
  <section id="section_hello" class="hidden">
    <h1 id="heading_hello">Hello, ...!</h1>
    <p>Welcome to Multimedia Programming!</p>
  </section>
</body>
```

Wenn Sie diese Datei ausführen, stellen Sie fest, dass sich nicht viel geändert hat. Anstelle einer Überschrift werden nun zwei angezeigt.

Allerdings wurden drei `id`-Attribute in das Dokument eingefügt: „`id="section_form"`“, „`id="section_hello"`“ und „`id="heading_hello"`“. Jedes öffnende HTML-Element darf mit einem derartigen Attribut versehen werden. **Allerdings darf es in einer HTML-Datei keine zwei `id`-Attribute mit demselben Namen geben.**

Die Vergabe von `id`-Attributen bringt zwei Vorteile mit sich: Zum einen können so bestimmte HTML-Element gezielt mittels CSS gestylt werden, und zum anderen können bestimmte HTML-Element gezielt mittels JavaScript modifiziert werden.

Im Browser soll zunächst nur der erste Abschnitt mit dem Formular angezeigt werden

(`id="section_form"`). Um das zu erreichen, fügen Sie zunächst folgenden Code in die CSS-Datei ein:

```
.hidden
{ display: none; }
```

Fügen Sie nun in das öffnende Tag des Section-Elements mit dem Identifikator `section_hello` das Attribut-Wert-Paar „`class="hidden"`“ ein. Ein `class`-Attribut darf im Gegensatz zu einem `id`-Attribut beliebig vielen Elementen zugeordnet werden, ohne dass sich der zugehörige Wert unterscheiden muss. `id`-Attribute werden verwendet, um HTML-Elemente eindeutig zu kennzeichnen, `class`-Attribute werden verwendet, um diversen HTML-Elementen gleiche CSS-Eigenschaften zuzuordnen. In einem Onepager sind üblicherweise die meisten Seiten unsichtbar. Daher ist hier ein `class`-Attribut angebracht.

In der CSS-Datei werden Identifikator-Attributewerte mit einer Raute „`#`“ gekennzeichnet (z. B. `#section_form`) und Klassen-Attributwerte mit einem Punkt „`.`“ (z. B. `.hidden`):

Führen Sie diese Änderungen durch und testen Sie die Web-Anwendung erneut. Nun sollte nur noch die Überschrift der Formular-Section zu sehen sein.

# 4 Definition eines HTML5-Formulars

Als nächstes muss das Formular erstellt werden, mittels dem der Name des Besuchers erfragt wird. Es soll folgende Elemente haben:

ein Label, das beschreibt, welche Information vom Benutzer eingegeben werden soll
ein Text-Feld, in das der Benutzer seinen Namen eingeben kann
einen Reset-Button, um den Inhalt des Namensfeldes zu löschen
einen Submit-Button, um dem Browser mitzuteilen, dass der Name vollständig eingegeben wurde

Der zugehörige HTML-Code sieht folgendermaßen aus:

```
<form>
  <div>
    <label for="input_name">What's your name?</label>
    <input id="input_name" autofocus="autofocus"/>
  </div>
  <div>
    <input id="button_reset"  type="reset"  value="Reset"/>
    <input id="button_submit" type="button" value="Say hello"/>
  </div>
</form>
```

Er wird in die erste Section hinter die zugehörige Überschrift eingefügt.

In diesem Formular („`<form> ... </form>`“) sind die vier zuvor genannten Elemente enthalten. Je zwei davon sind mittels eines Div-Elements zu einer Gruppe zusammengefasst. Damit ist die Struktur des Formulars vorgegeben: Jedes Div-Element steht in einer eigenen Zeile, die darin enthaltenen Elemente stehen jeweils hintereinander in einer Zeile. Mittels CSS können die Elemente nun besser

angeordnet werden: mehr Abstand von der Überschrift, mehr vertikaler Abstand zwischen den Elementen, einheitliche Breite der Elemente etc. Wie sagt Captain Picard ganz richtig: „Machen Sie es so.“

Beachten Sie, dass zwei weitere `id`-Attribute eingeführt wurden: „`id="input_name"`“ und „`id="button_submit"`“. Diese werden für den Zugriff von JavaScript aus auf das Dokument benötigt. Über den Identifikator `input_name` kann auf den Inhalt des Textfeldes, d. h. auf den Namen, den der Benutzer eingegeben hat, zugegriffen werden. Außerdem kann mit Hilfe dieses Identifikators das Label-Element über das Attribut `for` an das Textfeld gekoppelt werden. Damit weiß der Browser, auf welches Input-Element sich das Label-Element bezieht. Auch dies ist wieder besonders wichtig, wenn der Zusammenhang nicht optisch (per CSS) hergestellt werden kann bzw. hergestellt wird. Damit lassen sich aber auch Checkboxes realisieren, die durch einen Klick auf den zugehörigen Label aktiviert und wieder deaktiviert werden können.

Der Identifikator für den Submit-Button wird benötigt, um in JavaScript die Klick-Aktion des Benutzers abfangen zu können. Normalerweise im Form-Element in einem Attribut namens `action` ein URI angegeben. Dieser verweist auf eine Serveradresse, an den die vom Benutzer erfassten Daten bei einem Klick auf einen echten Submit-Button (`type="submit"`) übermittelt werden. In dieser Web-Anwendung werden keine Daten an einen Server übermittelt. Alle Benutzereingaben werden direkt im Browser (per JavaScript) verarbeitet. Deshalb wird das Action-Attribut nicht benötigt und als Submit-Button wird ein einfacher Button (`type="button"`) eingesetzt.

Wenn alles zu Ihrer Zufriedenheit ausgefallen ist, sollten Sie den aktuellen Stand in Ihr Repository einspielen.

# 5 Interaktion mittels JavaScript

Erzeugen Sie eine leere JavaScript-Datei:

Rechtsklick auf die Projektwurzel `HelloWorld03` im Dateibrowser → `New` → `JavaScript file` Name: `main.js` → `OK`

Fügen Sie folgenden JavaScript-Code in diese Datei ein

```
/**
 * Welcomes the user of the web app by displaying a welcome message
 * that includes his name. The name is fetched from a text input field.
 */
function sayHello()
{ document.getElementById('heading_hello').innerHTML =
    'Hello, ' + document.getElementById("input_name").value + '!';

  document.getElementById('section_form') .classList.add   ('hidden');
  document.getElementById('section_hello').classList.remove('hidden');
}

/**
 * Initializes the web app.
 * Is to be called when the web app has been loaded completely.
 */
function init()
{ document.getElementById('button_submit')
          .addEventListener('click', sayHello);
}

// So call init, when you are ready with loading.
window.addEventListener('load', init);
```

Im diesem Stückchen JavaScript-Code sind mehrere interessante Dinge zu entdecken.

1. Alle Befehle innerhalb einer JavaScript-Datei werden der Reihe nach abgearbeitet, sobald sie geladen wird. In der Datei `main.js` gibt es insgesamt drei Befehle: Zwei Funktionsdefinitionen und eine Wertzuweisung.
2. Achtung: Die beiden Funktionen `sayHello` und `init` werden nur definiert, aber nicht sofort ausgeführt! Damit sie eine Wirkung entfalten, müssen sie aufgerufen werden.
3. Bei `sayHello` und `init` handelt es sich um sogenannte Observer-Funktionen. Derartige Funktionen werden immer dann aktiviert, wenn ein bestimmtes Ereignis eintritt.
4. Mittels des letzten JavaScript-Befehls, wird festgelegt, dass die Init-Funktion ausgeführt wird, sobald die Web-Anwendung vollständig geladen ist, d. h., sobald der Browser-Fenster `window` das Ereignis `'load'` signalisiert.
5. Sobald Init-Funktion aufgerufen wird, wird festgelegt, dass die Funktion `sayHello` ausgeführt wird, wenn der Benutzer den Submit-Button drückt. Das heißt, solange die Init-Funktion nicht ausgeführt wurde, d. h., solange das Dokument und alle seine assoziierten Dokumente nicht vollständig geladen wurden, hat ein Klick des Benutzers auf den Submit-Button keinerlei Auswirkungen.
6. Beide Funktionen machen regen Gebrauch vom JavaScript-Objekt `document`, das das Document Object Model, d. h. die interne Darstellung des HTML-Dokuments als sogenannten DOM-Baum beinhaltet (siehe MDN-API-Dokumentation).
7. Mit Hilfe der Methode `getElementById` kann man besonders elegant auf bestimmte Elemente des DOM-Baus zugreifen, sofern man zuvor im HTML-Dokument für die entsprechenden Elemente `id`-Attribute definiert hat (was wir gemacht haben).
8. Für jedes Element des DOM-Baums gibt es diverse elementspezifische Attribute:
   So kann man mittels `innerHTML` auf den HTML-Code eines Elements wie `p` (Paragraph), `h1`

(Hauptüberschrift), `h2` (Überschrift der 2. Stufe) etc. lesend und schreibend zugreifen.
Das Attribut `value` ermöglicht einen lesenden und schreibenden Zugriff auf den Inhalt von Formularfeldern.
Über das Attribut `classList` hat man Zugriff auf `class`-Attribute, die einem HTML-Element zugeordnet sind. Man kann jedem Element neue `class`-Attribut-Werte zuordnen und bestehende Werte entfernen.
Wir nutzen das aus, indem wir die Formularabschnitt `section_form` unsichtbar und dafür den Begrüßungsabschnitt `section_hello` sichtbar machen, sobald die Funktion `sayHello` ausgeführt wird.

9. Der dritte Befehl „`window.addEventListener('load', init);`“ übergibt die Funktion `init` (und nicht etwa den Funktionsaufruf `init()`) dem JavaScript-Objekt `window` mit der Bitte, diese Methode auszuführen, sobald das `load`-Ereignis eintritt. Das hat zur Folge, dass die Funktion `init` nicht sofort aufgerufen wird, sondern erst – durch das Observer-Objekt `window` – sobald das Ereignis „der Inhalt des aktuelle Browserfensters wurde vollständig geladen“ eintritt (siehe MDN-API-Dokumentation).
10. Auch für den Submitbutton wird ein Eventlistener registriert. Die Funktion `sayHello` soll ausgeführt werden, sobald für diesen Button das `click`-Ereignis eintritt. Diese Zuordnung kann allerdings erst passieren, wenn sich der Submit-Button auch im DOM-Baum befindet. Das ist jedoch sicher erst der Fall, wenn das gesamte Dokument geladen wurde. Daher wird diese Zuordnung nicht sofort, sondern in der Init-Funktion durchgeführt. (Das `window`-Objekt existiert dagegen von Anfang an, d. h. auch wenn der DOM-Baum noch nicht geladen wurde.)
11. Alle Funktionen wurden mit Kommentaren im JSDoc-Format versehen.[1] Machen Sie das auch immer. Andere Entwickler und auch Sie selbst werden das schätzen, wenn sie bzw. Sie den Code zu einem späteren Zeitpunkt lesen und verstehen müssen. Das JSDoc-Format bring den Vorteil mit sich, dass Sie automatisch eine Schnittstellen-Dokumentation für Ihre Anwendung erstellen können

    Wenn Sie jetzt Ihre Anwendung testen, hat sich nichts geändert. Der Grund ist wie bei der Datei `main.css` auch, dass die Datei `index.html` nichts davon weiß, das ihr dieser JavaScript-Code zugeordnet ist. Das muss ihr erst bekannt gegeben werden. Fügen Sie folgende Zeile in der Head-Bereich der `index.html` ein:

    ```
    <script src="main.js" async="async"></script>
    ```

    Nun wird nicht nur die CSS-Datei, sondern auch die JavaScript-Datei geladen, bevor der Body-Bereich der `index.html` eingelesen wird. Die Angabe von `async` bewirkt Folgendes: Das Rendering der Seite wird durch das Laden des Skripts **nicht** unterbrochen. Das heißt, wenn ein langes Skript ohne Angabe von `async` geladen wird, kann sich der Seitenaufbau merklich verzögern. Mit Angabe dieser Option ist dies nicht der Fall.

    Sobald das Skript geladen wurde, wird es ausgeführt. Zu diesem Zeitpunkt ist der Rendervorgang möglicherweise noch nicht abgeschlossen. Das heißt, man muss darauf achten, dass man von Skript aus nicht zu früh auf die Elemente des HTML-Dokuments zugreift. Aus diesem Grund wird die `init`-Funktion am Ende der Datei `main.css` nicht direkt mittels `init()` gestartet, sondern mittels eines Eventlisteners `window.addEventListener('load', init)`.

    Beachten Sie, dass folgender Code zwar SGML/XML-konform, aber in Standard-HTML5-Dokumenten nicht erlaubt ist:

```
<script src="main.js" async="async"/>
```

Die tieferen Gründe dafür werden sehr schön auf Stack Overflow beschrieben. (Ich verstehe es trotzdem nicht.)

Testen Sie Ihre Anwendung nochmals. Sie sollten jetzt Ihren Namen in das Textfeld eingeben und dann „`Say hello`“ klicken können. Das Ergebnis sollte sein, dass Sie von Ihrer Anwendung persönlich begrüßt werden. Wenn alles funktioniert, sollten Sie wieder committen.

# 6 Barrierefreiheit

Gemäß den „Richtlinien für barrierefreie Webinhalte“[2], Abschnitt 2.1 soll eine Web-Entwickler dafür Sorge tragen, „dass alle Funktionalitäten per Tastatur zugänglich sind“. Das ist bei unserer Web-Anwendung nicht in voller Schönheit der Fall. Der Cursor befindet sich wegen des Attributes „`autofocus="autofocus"`“ im Text-Input-Feld des HTML-Dokument bei Start der Anwendung automatisch an der richtigen Position. Man muss also den Cursor nicht erst mit Hilfe der Maus platzieren. Auch die Weiterschaltung mittels Tab-Taste funktioniert. Man kann zunächst seinen Namen eingeben, dann zweimal die Tab-Taste betätigen, um den Submit-Button zu aktivieren und dann Return drücken, um ein Klick-Event für diesen Button auszulösen.

Schöner wäre es jedoch, wenn man nach Eingabe des Namens gleich die Returntaste nutzen könnte, um die Eingabe abschließen zu können. Zurzeit passiert nichts, wenn Sie Ihren Namen eingeben und dann die Returntaste betätigen. Versuchen Sie es.

Damit begeben wir uns in die Untiefen der Benutzerinteraktion mit dem Browser...

Um diese Funktionalität zu realisieren, brauchen wir einen weiteren Eventlistener. Die Observer-Methode heiße `sayHelloOnEnter`. Diese muss wieder registriert werden. Die Tastaturereignisse werden vom `window`-Objekt gemeldet (und nicht etwa vom `dokument`-Objekt, da das Dokument nichts mit Tastatureingaben zu schaffen hat). Die Registrierung des Listeners erfolgt wie üblich mit Hilfe der `addEventListener`-Methode

```
window.addEventListener('keydown', sayHelloOnEnter);
```

Dieser Befehl kann sowohl innerhalb der `init`-Funktion, als auch außerhalb stehen (da das `window`-Objekt von Anfang an existiert). Man sollte es in den Rumpf der `init`-Funktion einfügen. Die Anzahl der globalen Befehle sollte so gering wie möglich gehalten werden. (Dieser Punkt wird später noch wesentlich genauer behandelt.)

Die (in diesem Teil des Tutoriums noch globale) Funktion `sayHelloOnEnter` wird folgendermaßen definiert:

```
/**
 * A keyboard event observer. It tests whether the enter key has been
pressed.
 * If so, the sayHello method is activated. Default reactions of the
browser are
 * disabled.
 * @param {KeyboardEvent} p_event - A standard JavaScript keyboard event
object
 *   (https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent)
 */
function sayHelloOnEnter(p_event)
{ if (p_event.key === 'Enter')  // The enter key has been pressed.
  { // If the input text field or the submit button is active say hello.
    if (document.activeElement === document.getElementById('input_name')
||
        document.activeElement ===
document.getElementById('button_submit')
       )
    { sayHello(); }
    // If the reset button is active clear the input text field.
    else if (document.activeElement ===
document.getElementById('button_reset'))
    { document.getElementById('input_name').value = ''; }
  }
}
```

Fünf Dinge fallen bei dieser Definition auf:

1. Die Funktion hat einen Parameter namens `p_event` (Parameternamen können beliebig gewählt werden. Ich beginne Parameternamen stets mit einem `p_`, um rein optisch klar zu machen, dass es sich um einen Parameter handelt; siehe Multimedia-Programmierung: Style Guide). In diesem Parameter übermittelt das `window`-Objekt dem Eventhandler nähere Informationen zum aktuellen Tastaturereignis: welche Taste gedrückt wurde, ob die Schift-Taste dabei gehalten wurde etc. Details finden sich in der ausgezeichneten MDN-JavaScript-Dokumentation.
2. Im Funktionsrumpf wird getestet, ob die Returntaste gedrückt wurde. Falls das Ergebnis des Test false lautetet, macht sie nichts (da das Zeichen einfach ins Textfeld eingefügt werden soll), anderenfalls ruft sie die Funktion `sayHello` auf, um die Eingabe abzuschließen.
3. Die Gleichheit wird mit „`===`“ anstelle von „`==`“ getestet. Bei dem ersten Test handelt es sich um einen strikten Gleichheitstest. Der zweite Test liefert dagegen manchmal ziemlich eigenwillige Ereignisse. So wird beispielsweise bei den Tests „`0==''`“ „`0=='0'`“ jeweils `true` als Ereignis ausgegeben. Der strikte Gleichheitstest liefert dagegen das erwartete Ergebnis `false`.
4. Die Bedeutung der Enter-Taste anhängig muss bhängig vom Fokus verändert werden, da dieser mittels der Tab-Taste geändert werden kann. Wenn der Fokus auf dem Texteingabefeld oder dem Submit-Button liegt, soll der Benutzer gegrüßt werden. Liegt er dagegen auf der Reset-Taste, wird das Eingabefeld geleert.
5. Im Code wird `p_event.key === 'Enter'` anstelle von `p_event.keyCode ===13` verwendet. Die letztere Methode gilt als deprecated, da sie die Position der gedrückten Taste auf der Tastatur ausgibt. Diese Position ist jedoch sprachabhängig. Beispielsweise hat die Z-Taste auf einer

amerkikansischen Tastatur eine andere Position als auf einer deutschen. Mit der folgenden JSFiddle-Anwendung können Sie die Unterschiede zwischen `p_event.key`, `p_event.code` und `p_event.keyCode` untersuchen. Öffen Sie die Anwendung in verschiedenen Brwosern und drücken Sie unterschiedliche Tasten (z. B. auch im Nummernblock.

So weit so gut. Aber wenn man `sayHelloOnEnter` so definiert, dass sie bei einem Druck der Returntaste lediglich die Funktion `sayHello` aufruft oder das Eingabefeld löscht, stellt man schnell fest, dass ein Betätigen der Returntaste nicht wie gewünscht funktioniert. Das Problem ist, dass der Browser Tastatur-Ereignisse bereits anderweitig verarbeitet. Je nach Tastendruck werden Textfelder befüllt, Formularelemente aktiviert oder irgendwelche Spezialfunktionen ausgeführt. Man muss den JavaScript-Interpreter klar machen, dass er dies im Falle der Returntaste nicht machen soll. Das ist Aufgabe der beiden folgenden Befehle:

```
p_event.preventDefault();
p_event.stopPropagation();
```

Der erste verlangt, dass bei Drücken dieser Taste keine Default-Aktionen ausgeführt werden sollen und der zweite bewirkt, dass das Ereignis nicht auch noch von irgendwelchen anderen Eventhandlern behandelt wird. Fügen Sie also die beiden Befehle noch an den Anfang des Rumpfes der Funktion `sayHelloOnEnter` ein.

Vergessen Sie nicht, Ihr aktuelles Projekt auf dem Git-Server zu speichern, sobald alles funktioniert.

# 7 Quellen


1. **JSDoc (GitHub)**: [[Contributors]]; @use JSDoc; Organisation: The JSDoc 3 documentation project; http://usejsdoc.org/; Quellengüte: 5 (Web)
2. **W3C (2009)**: Web Content Accessibility Guidelines (WCAG) 2.0; Hrsg.: Ben Caldwell, Michael Cooper, Loretta Guarino Reid und Gregg Vanderheiden; Band: 2014; Organisation: W3C; http://www.w3.org/Translations/WCAG20-de/WCAG20-de-20091029/; 2009; Quellengüte: 5 (Web)
1. **Kowarschick (MMProg)**: Wolfgang Kowarschick; Vorlesung „Multimedia-Programmierung“; Hochschule: Hochschule Augsburg; Adresse: Augsburg; Web-Link; 2018; Quellengüte: 3 (Vorlesung)



Kategorien:
Multimedia-Programmierung/Tutorium
Praktikum:MMProg
HTML5-Tutorium: JavaScript: Hello World
HTML5-Beispiel
Kapitel:Multimedia-Programmierung:Beispiele