

JavaScript Einführung

nick.schneeberger@fhgr.ch

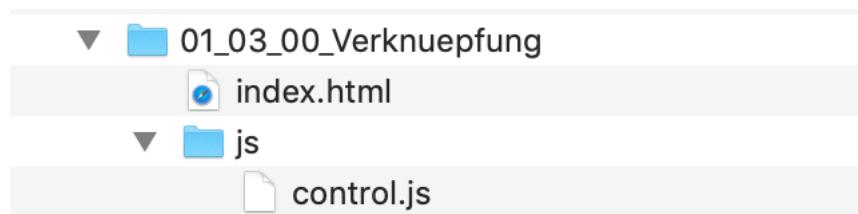


10 Minuten

Übung 01.00 Verknüpfung

Erstelle einen neuen Ordner 'JS', erstelle ein neues Dokument mit dem Namen 'control.js' und verknüpfe das HTML mit dem JavaScript Dokument über einen Skript-Tag.

Teste die Verknüpfung mit `console.log("Hello World!");`



Übung 01.01 Alter

 15 Minuten

Programmiere ein Skript, das dir ausrechnet, in welchem Jahr du wie alt sein wirst. Verwende Variablen für dein Geburtsjahr und das aktuelle Jahr. Gib dein Resultat in der Konsole in einem Satz aus.

Beispiellösung:

Im Jahr 2020 werde ich 23 oder 24 Jahre alt sein.



20 Minuten

Übung 01.02 Versorgung

Ein ängstlicher Bürger plant, wegen des Coronavirus sein Haus nicht mehr zu verlassen. Entwickle einen Versorgungsrechner, der ihm in der Konsole ausgibt, wie viele Beutel Tee oder Packungen Pasta er heute kaufen muss, um bis an sein Lebensende auszuhalten.

Verwende Variablen für Alter, Lebenserwartung, Konsum pro Tag sowie Artikel (Tee) und Einheit (Beutel).

Beispiellösung:

Du brauchst 56210 Beutel Tee bis zum Alter von 100 Jahren.



15 Minuten

Übung 01.03 Fahrenheit

Du möchtest deinen amerikanischen Freunden vom ausserordentlich warmen Winter hier erzählen. Programmiere ein Skript, das dir Grad Celsius in Grad Fahrenheit umrechnet.

Formel:

<https://www.mathsisfun.com/temperature-conversion.html>

Beispiellösung:

15°C entsprechen 59°F

20°F entsprechen -6.66°C

Übung 02.01 Notenrechner

10 Minuten

Programmiere ein Skript, das dir je nach Punktzahl (0-100) jeweils die entsprechende Note (3-6) in der Konsole ausgibt. Verwende für die Lösung dieser Aufgabe eine If / Else Bedingung.

Exkurs: Atom Shortcuts

cmd / ctrl + click

→ Mehrere Cursor



markieren + cmd + d

→ Gleichen Ausdruck markieren (und ersetzen)

cmd / ctrl + shift + 7

→ Eine oder mehrere Zeilen auskommentieren

cmd / ctrl + shift + i

→ Aufräumen, Code automatisch einrücken

Dafür muss das Plugin 'Auto-Indent' installiert sein



Übung 02.02 Übersetzer

15 Minuten

Programmiere ein Skript, das jeweils in der Sprache deines Browsers «Hallo Welt» respektive «Hello World» ausgibt.

Finde die Browsersprache heraus:

https://www.w3schools.com/jsref/prop_nav_language.asp

Beispiellösung:

Hallo Welt!



25 Minuten

Übung 02.03 Pluralisator

Im Englischen werden die meisten Pluralformen gebildet, indem ein 's' an das jeweilige Nomen angehängt wird (cat → cats).

Entwickle mit den Variablen `animal` und `number` ein Skript, das englische Tiernomen automatisch ins Plural setzt, wenn die Anzahl (`number`) grösser als 1 ist. Berücksichtige, dass jemand aus Jux auch 0 Tiere oder 'abc' Tiere eingeben könnte. Gib die Lösung als Satz in der Konsole aus.

Baue in dein Skript folgende Ausnahmen ein:

Mouse → Mice, Sheep → Sheep, Goose → Geese

Beispiellösung:

2 cats

Übung 03.01 Create

15 Minuten

Der Versorgungsrechner aus Übung 01.02 soll neu im DOM visuell dargestellt werden. Erstelle dazu einen Titel sowie eine Zeile, in der die Nachricht ausgegeben wird, dynamisch mittels JavaScript.

Beispiellösung:

Versorgungsrechner

Du brauchst 29930 Tassen Tee bis zum Alter von 66 Jahren.



10 Minuten

Übung 03.02 innerHTML

Schreibe alle statischen Elemente des Versorgungsrechners (Titel, Teile der Nachricht) direkt ins HTML. Verwende `` Elemente, um dynamische Inhalte (Variablen) mittels JavaScript zu füllen.

Beispiellösung:

Versorgungsrechner

Du brauchst 29930 Tassen Tee bis zum Alter von 66 Jahren.



10 Minuten

Übung 03.03 mit Style

Ändere die Farbe des Titels abhängig davon, wie viele Einheiten (z.B. Tassen Tee) errechnet wurden.

Style Object Properties:

https://www.w3schools.com/jsref/dom_obj_style.asp

Beispiellösung:

Versorgungsrechner

Du brauchst 10950 Tassen Tee bis zum Alter von

Versorgungsrechner

Du brauchst 1460 Tassen Tee bis zum Alter von 27 Jahren.



Übung 03.04 CSS Klassen

10 Minuten

Definiere Style-Anpassungen (z.B. Farbänderungen) neu in der CSS Datei als Klassen. Setze Übung 03.03 um, indem du diese Klassen dynamisch hinzufügst oder entfernst, statt direkt auf das Style Attribut eines HTML Elements zuzugreifen.

Beispiellösung:

Versorgungsrechner

Du brauchst 10950 Tassen Tee bis zum Alter von

Versorgungsrechner

Du brauchst 1460 Tassen Tee bis zum Alter von 27 Jahren.

⌚ 15 Minuten

Übung 04.01 Ruf den Koch

In Übung 03.04 haben wir gelernt, wie Klassen mit JavaScript hinzugefügt werden können. Programmiere in dieser Aufgabe den Button so, dass auf Knopfdruck der Koch erscheint und bei erneutem Drücken wieder verschwindet.

Tipp:

Verwende die vorgegebene Klasse .hidden in style.css





30 Minuten

Übung 03.02 Dimmer

Programmiere einen Dimmer, der je nach Position des Sliders 1-5 Glühbirnen einblendet. Zeige den aktuellen Wert des Sliders dabei unterhalb der Glühbirnen an.

Beispiellösung:



15



45



Verwende `element.value`, um den Wert (Position) des Sliders auszulesen.



Übung 03.03 Audioplayer

Programmiere einen Audioplayer, den du über Play / Pause Buttons steuern kannst. Zeige jeweils die aktuelle Abspielzeit in Sekunden unterhalb des Players an.

Event zur Abspielzeit:

https://www.w3schools.com/tags/av_event_timeupdate.asp

Beispiellösung:



Bereits gespielt: 18 Sec.



Übung 04.04 Layer

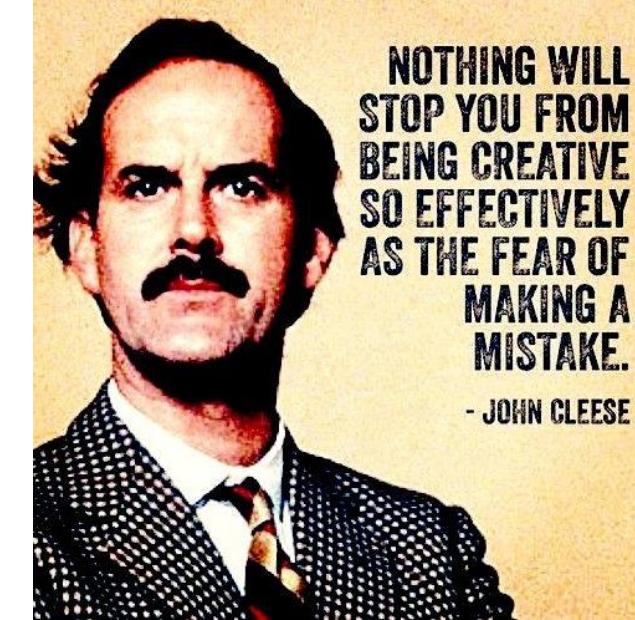
Vorgegeben sind verschiedene HTML Elemente / Layers.
Programmiere ein Skript so, dass bei jedem Klick auf ein
Element, dessen Tag-Name (h1, p, b ...) und dessen ID in der
Konsole ausgegeben wird.



and
now
for something
completely different

Aufgabe Bildgenerator

- Speichere mindestens 5 Bilder in einem **Array**
- **Platziere** Bilder mit `createElement & appendChild`
 - Verpacke das Platzieren der Bilder in einer Funktion
- Definiere **Position, Breite & Rotation** der Bilder mit **inline CSS**
 - Um Bilder frei zu positionieren, verwende `position: absolute` (in CSS)
 - Verwende die Methode **Math.random()** für zufällige Werte
- Erstelle einen Button, der auf **Klick** die gleichen Bilder nochmals hinzufügt



NOTHING WILL
STOP YOU FROM
BEING CREATIVE
SO EFFECTIVELY
AS THE FEAR OF
MAKING A
MISTAKE.

- JOHN CLEEESE

IM II

MULTIMEDIA IM WEB

AGENDA

- **Multimedia im Web**
- **Multimedia Typen**
- **HTML5-Player**
- **Einbinden und Optionen**
- **Funktionen und Eigenschaften**
- **Knigge**
- **Übung 1**
- **Events bei Audio/Video-Elementen**
- **Übung 2**

MULTIMEDIA IM WEB

- **Audio und Video machen das Internet interessant**
- **Immer mehr Videos, Bilder und Animationen werden auf Webseiten integriert**
- **Moderne Webseiten nutzen eine Vielzahl von multimedialen Inhalten**

PROS CONS

- | | |
|---|---|
| <ul style="list-style-type: none">• lockern textlastiges Web auf• grössere Immersion auf wenig Platz (Grafiken, Stimmungen, Erklärungen, ect)• Audio bietet bildschirmloser Kanal• Besseres Seitenranking (SEO)• Mehr Branding möglich | <ul style="list-style-type: none">• Viel Daten (=Ladezeit) nötig• Inhalte müssen gut optimiert werden• Schlecht optimierte Inhalte schaden SEO• Kompatibilität nicht bei allen Browsern mit allen Dateitypen gewährleistet |
|---|---|

MULTIMEDIA IM WEB

Beispiele:

Video Background:

<https://artsandculture.withgoogle.com/en-us/>

<https://gangfilms.com/>

<https://www.designdisruptors.com/>

Video als Scrollytelling:

<https://www.apple.com/chde/airpods-pro/>

MULTIMEDIA TYPEN

- **Die häufigsten Multimedia-Formate sind: Video und Audio**
- **Die Kompatibilität hängt vom Browser und dem Betriebssystem ab**
- **Moderne Datei-Formate sind sehr breit unterstützt**

EMPFEHLUNG

Video: Von allen gängigen Browsern wird *.mp4 unterstützt

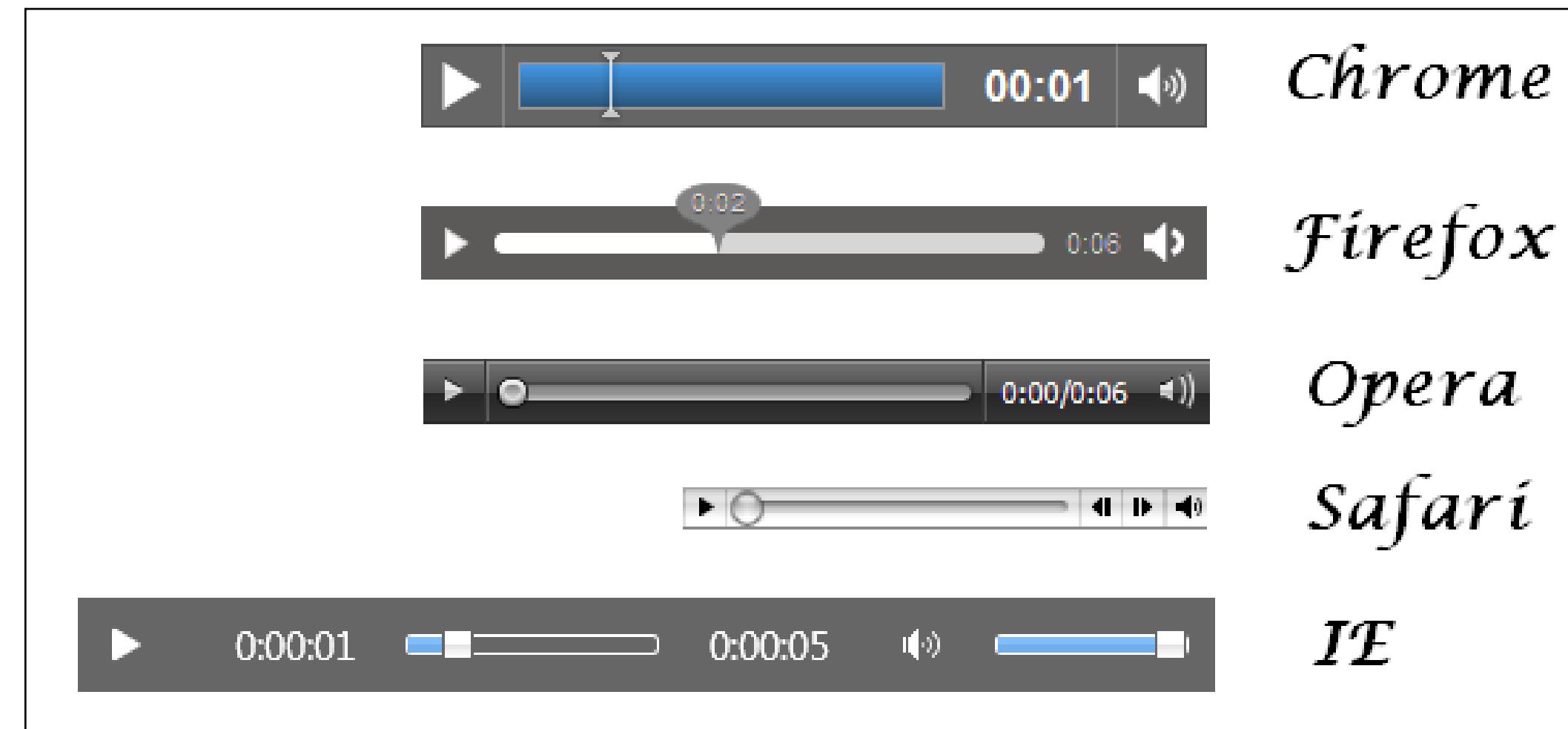
Audio: Von allen gängigen Browsern wird *.mp3 unterstützt

HTML5

- **Frühes Internet basierte ausschliesslich auf Text**
- **Bald kamen Bilder hinzu**
- **Um Videos abzuspielen brauchte es mehr als nur HTML; Technologien wie z.B. Flash wurden entwickelt**
- **Die Unterstützung der Browser war nicht weit verbreitet, Flash (und andere) waren langsam, fehleranfällig und unsicher**
- **Seit 2014 werden die Browser mit HTML5 ausgestattet**
- **Mit HTML5 kamen die Tags <video> und <audio>**
- **Somit ist man nicht mehr auf Drittsoftware angewiesen und das HTML unterstützt von sich aus diese Dateiformate (mit einem eigenen Video-/Audio-Player)**

HTML5

- **Browserhersteller entwickeln seither ihre eigenen Player**
- **HTML5 schreibt vor, welche Funktionen, Eigenschaften und Events enthalten sein müssen**
- **Das Design ist von Browser zu Browser unterschiedlich:**



Quelle: <https://html5tutorial.info/html5-audio.php>

SYNTAX

- **Kurze Schreibweise**
- **Lassen sich via CSS stylen**
- **Bis auf das "src"-Attribut sind die Attribute optional**

Einbinden von einem Video:

```
<video src="video/film.mp4" width="320" height="240" controls>
</video>
```

Einbinden von einem Audio-File:

```
<audio src="audio/musik.mp3" controls></audio>
```

SYNTAX

Diese optionalen Werte konfigurieren den “Video-Player”.

```
<video src="film.mp4"  
[width="320"]  
[height="240"]  
[controls]  
[autoplay]  
[loop]  
[muted]  
[poster=""]  
[preload]  
></video>
```

Breite des Videos

Höhe des Videos

Buttons wie Play/Fullscreen/Timeline einblenden

Sobald Seite geladen, Video abspielen

Nach Video-Ende wieder von vorne anfangen

Lautlos

Eigenes Bild als Thumbnail verwenden

**Inhalt vorladen (wenn man sich ziemlich sicher ist,
der User wird das Video anschauen)**

SYNTAX

Diese optionalen Werte konfigurieren den “Audio-Player”.

```
<audio src="audio.mp3"  
[controls]  
[autoplay]  
[loop]  
[muted]  
[preload]  
></audio>
```

Buttons wie Play/Lautstärke/Timeline einblenden
Sobald Seite geladen, Video abspielen
Nach Video-Ende wieder von vorne anfangen
Lautlos
**Inhalt vorladen (wenn man sich ziemlich sicher ist,
der User wird das Video anschauen)**

FUNKTIONEN

- Alle Aktionen bei einem Video/Audio die der User auslösen kann, kann man auch via JavaScript.
- Mit einer Referenz zum Video-Objekt (per ID) kann die Funktion ausgelöst werden.
- Auswahl der wichtigsten Funktionen (sowohl für Video- wie Audio-Elemente):

```
video.play(); // Starte Wiedergabe  
video.pause(); // Stoppe Wiedergabe  
video.load(); // Lade Element neu (Stop)
```

Referenz online:

https://www.w3schools.com/tags/ref_av_dom.asp

EIGENSCHAFTEN

- **Diese Eigenschaften kann man entweder nur Lesen (gibt an) oder auch Schreiben (setzen)**
- **Die Eigenschaften zeigen den Status des Elementes**
- **Auswahl der wichtigsten Eigenschaften:**

```
video.duration // gibt die Länge in Sekunden zurück  
video.muted // setzt oder gibt an, ob das Element lautlos ist  
video.volume // setzt oder gibt die Lautstärke an  
video.paused // gibt an, ob das Element pausiert ist
```

Referenz online:

https://www.w3schools.com/tags/ref_av_dom.asp

ZUSAMMENFASSEND

```
....  
<video src="video/film.mp4" id="mein-video"></video>  
<script>  
let video = document.querySelector("#mein-video");  
  
video.muted = true;  
video.play();  
</script>  
....
```

Das Video wird lautlos abgespielt, sobald die Seite geladen wurde.

KNIGGE

- **Autoplay**
(Lautes) Autoplay verärgert Seitenbesucher
Autoplay muss nicht immer negativ sein > Der User muss es aber erwarten!
Beispiel: YouTube
Autoplay wird von manchen Browsern blockiert (Firefox)
Autoplay von tonlosen Videos ist okay
- **Steuerung**
Multimedia-Inhalte sollten vom User gesteuert werden können
(Lautstärke, Mute, Play/Pause, Loop, ect.)
- **Interaktion**
Interaktion mit Medien ist der Schlüssel zu einem ausserordentlichen Erlebnis auf der Webseite

IM II - MULTIMEDIA IM WEB

ERSTE SCHRITTE MIT

MULTIMEDIA-INHALTEN

VIDEO '20_mm_inhalte_first-steps_abgabe'

- 1. Video mit 'controls' und 'preload' einfügen**
- 2. Video mit 'loop' und 'preload' einfügen, welches mit einem externen Button gesteuert wird.**
- 3. Zusatz: Button soll sich dem gegenteiligen 'paused'-Status (true / false) anpassen**

AUDIO '20_mm_inhalte_first-steps_abgabe'

- 1. Audio-Datei mit 'controls' und 'preload' einfügen**
- 2. Audio mit 'loop' und 'preload' einfügen, welches mit einem externen Button gesteuert wird (Audio-Inhalt wird somit 'unsichtbar').**
- 3. Zusatz: Button soll sich dem gegenteiligen 'paused'-Status (true / false) anpassen**

IM II - MULTIMEDIA IM WEB

ÜBUNG 1: AUDIO-PLAYER

AUFGABE '30_mm_inhalte_player_1_abgabe'

Ziel ist einen Video-Player mit folgenden Funktionen zu bauen:

- 1. Button mit "Play"/"Pause" (je nach dem wie gerade der Abspiel-Zustand ist)**
- 2. Button "Stop"**
- 3. Button "Lautlos"**

Der aktuelle Zustand soll in der "Statusbar" ausgegeben werden. Dabei soll bei jedem Klick der Text in <p id="statusbar"></p> überschrieben werden.

- 4. Zusatzaufgabe für Schnelle: Nummern-Feld, mit welchem man die Lautstärke steuern kann. (mit entsprechendem Eventlistener auf einen neuen Wert)**

IM II - MULTIMEDIA IM WEB

LÖSUNG: AUDIO-PLAYER

IM II - MULTIMEDIA IM WEB

PAUSE

IM II - MULTIMEDIA IM WEB

MULTIMEDIA-EVENTS

EVENTS

- Interaktivität kann bei Videos oder Audio-Files mit Events erzeugt werden.
- HTML5 stellt viele Events bereit, welche wir sofort nutzen können.

Eine Auswahl von möglichen Events:

| | |
|--------------|---|
| 'ended' | wird ausgelöst, wenn das Video fertig ist |
| 'timeupdate' | wird ausgelöst, wenn sich die Zeit aktualisiert |
| 'pause' | wird ausgelöst, wenn der User das Video pausiert |
| 'play' | wird ausgelöst, wenn das Video abgespielt wird |
| 'loadeddata' | wird ausgelöst, wenn alle Daten geladen wurden |

Referenz online:

https://www.w3schools.com/tags/ref_av_dom.asp

IM II - MULTIMEDIA IM WEB

ERSTE SCHRITTE MIT

EVENTS

VIDEOPLAYER '10_mm_inhalte_events_abgabe'

- 1. Ein Video einfügen, welches mit 'controls' versehen ist**

- 2. Folgende Events für das Video-Element registrieren und was passiert jeweils in der Statusbar ausgeben ("Das Video ist fertig.", ect).**
 - 'ended'
 - 'pause'
 - 'play'

- 3. Den Event 'timeupdate' einfügen und die Zeit gerundet in <p id="verbleibende-zeit"></p> ausgeben**

IM II - MULTIMEDIA IM WEB

ÜBUNG 2: VIDEO-PLAYER

AUFGABE '40_mm_inhalte_player_2_abgabe'

Ziel ist den Video-Player aus Übung 1 mit folgenden Funktionen zu ergänzen:

- 0. Alle "Statusbar"-Befehle aus Übung 1 löschen (oder Vorlage für Player 2 verwenden)**
- 1. Auf die Events 'ended' und 'pause' hören und den Status in der Statusbar ausgeben**
- 2. Auf Event 'timeupdate' hören und die aktuelle sowie totale Zeit im Format "4s/18s" (gerundet mit Math.round([wert])) ausgeben.**
- 4. Zusatzaufgabe für Schnelle: Auch auf Events 'play' und 'loadeddata' hören.**

IM II - MULTIMEDIA IM WEB

LÖSUNG: VIDEO-PLAYER

IM II - MULTIMEDIA IM WEB

FRAGEN?

DANKE.

IM II

LOCALSTORAGE

AGENDA

- **Einführung localStorage**
- **Anwendungsfälle**
- **Alternativen**
- **Syntax**
- **Erste Schritte**
- **Übung (inklusive Repetition)**

EINFÜHRUNG LOCALSTORAGE

- Der **localStorage** speichert Daten lokal auf dem Gerät
- Der **localStorage** ist auch ohne Request (Cookies) zugreifbar
- Max. 5MB Daten speicherbar
- Jeweils ein Schlüssel und einen Wert speicherbar (wie eine Variable)
- Es sind nur 'Strings' speicherbar
- Der Speicher ist nur für die jeweilige Domain sichtbar (z.B. *.fhgr.ch/*)
- Die Daten werden auf unbegrenzte Zeit auf dem aktuellen Gerät gespeichert (ausser der User löscht sie absichtlich)

Reference:

https://www.w3schools.com/html/html5_webstorage.asp

ANWENDUNGEN

- **Wenn Daten über mehrere Besuche gespeichert werden sollen**
- **Daten für den Offlinebetrieb speichern**
- **Wenn Daten nur für den jeweiligen Nutzer verfügbar sein müssen und somit keine Datenbank nötig ist (z.B. lokale Todo-Liste, Notizen, ...)**
- **Prototyping von Anwendungen, wenn noch kein Backend vorhanden ist**
- **Performance von einer Seite erhöhen, indem Daten schon vorgeladen wurden.**

ALTERNATIVEN

sessionStorage

Funktioniert gleich wie der 'localStorage'

Speichert Daten so lange, bis der Tab geschlossen wird

indexedDB

Strukturierte Daten lokal speichern (lokale Datenbank)

sehr viel Daten können gespeichert werden (mehr als 50MB), sehr komplex!

Cookies

Nur verwenden, wenn die Daten serverseitig verwendet werden

Gemacht, um Userdaten zu speichern

SYNTAX

Daten speichern

```
localStorage.setItem('nameKatze', 'Tom');
```

Daten aufrufen und in Variable speichern

```
let name = localStorage.getItem('nameKatze');  
let name2 = localStorage.nameKatze;
```

Daten löschen

```
localStorage.removeItem('nameKatze'); // dieser Eintrag  
localStorage.clear(); // alle Einträge
```

IM II - LOCALSTORAGE

ERSTE SCHRITTE

ERSTE SCHRITTE '10_localStorage_erneute-schritte_abgabe'

- 1. Speichere deinen Name, Nachname und Alter (als Zahl) im localStorage**
- 2. Untersuche die Webseite und gehe zum Register 'Web-Speicher'**
- 3. Gib die Daten in der Konsole und in einem Paragraph aus**
- 4. Speichere ein Array ab (nutze dafür `JSON.stringify([array])`)**
- 5. Gib das Array in der Konsole aus (nutze dafür `JSON.parse([string])`)**

IM II - LOCALSTORAGE

ÜBUNG

AUFGABE '20_localStorage_formular_abgabe'

Ziel ist mit jedem Enter in einem Textfeld das Array im localStorage zu ergänzen

- 1. Erstelle ein Formular mit einer ID und höre auf den Event 'submit'**

- 2. Die Event-Funktion soll der Reihe nach folgendes machen:**
 - Array aus dem localStorage holen und parsen**
 - Array mit neuem Eintrag füllen**
 - Array mit einer Schleife in einer Liste ausgeben (Liste vorher leeren)**
 - Array 'stringifyen' und wieder in den localStorage kopieren**
 - Textfeld leeren**

- 3. Für Schnelle: Einen Button, um den localStorage komplett zu löschen.**

IM II - LOCALSTORAGE

HILFESTELLUNG BEI

PROBLEMEN

HILFESTELLUNG: STEP BY STEP

- 1. Formular-ID, Textfeld & Notiz-Liste selektieren und in einer Variable speichern**
- 2. Eventlistener auf den 'submit' des Formulares hinzufügen**
- 3. In der Event-Funktion: event.preventDefault(); hinzufügen**
- 4. Überprüfen, ob das Textfeld leer ist (dann nichts machen)**
- 5. Überprüfen, ob der localStorage-Eintrag existiert. Wenn nicht, hinzufügen**
- 6. Liste aus dem localStorage holen und parsen (JSON.parse())**
- 7. Den aktuellen Text dem Array hinzufügen (array.push())**
- 8. Die Notiz--Liste löschen (notizListe.innerHTML = ‘ ’)**
- 9. For-Schleife, welche ein -Element erstellt und dieses mit dem Text füllt.
Dieses Element muss dann der Liste hinzugefügt werden (notizListe.appendChild([li-Element]))**
- 10. Array stringifyen und wieder im localStorage speichern**
- 11. Notizfeld leeren mit notizFeld.value = ‘ ’**

IM II - LOCALSTORAGE

LÖSUNG

**IM II - LOCALSTORAGE
FRAGEN?**

IM II - LOCALSTORAGE

PAUSE

JavaScript-Objekte

AJAX

JSON

Interaktive Medien II
FS 20

Einführung

Die Themen JavaScript-Objekte, AJAX und JSON gehören zusammen.

Ziel

Daten vom Server laden und auf der Webseite anzeigen

Ziel

Daten vom Server laden und auf der Webseite anzeigen

JSON AJAX JS-Objekte

Ziel

Daten vom Server laden und auf der Webseite anzeigen

| | | |
|------|------|------------|
| JSON | AJAX | JS-Objekte |
|------|------|------------|

JavaScript-Objekte ?

Bündelung zusammengehörender Daten

Daten werden nach einfachen Regeln in JavaScript strukturiert zusammengefasst.

Mit JavaScript lässt sich der Inhalt von JavaScript-Objekten sehr leicht auf der Webseite anzeigen.

Beispiel

```
let person1 = {  
    alter: 42,  
    anrede: 'Herr',  
    name: {  
        vorname: 'Urs',  
        nachname: 'Thöny'  
    },  
    interessen: ['Film', 'Fahrrad fahren']  
};
```

Lernziele: JavaScript-Objekte

Regeln für JavaScript-Objekte kennen

Daten im korrekten JavaScript-Objekt-Syntax speichern können

Inhalte von JavaScript-Objekten in der Webseite darstellen können

Ziel

Daten vom Server laden und auf der Webseite anzeigen

| | | |
|------|------|------------|
| JSON | AJAX | JS-Objekte |
|------|------|------------|

Ziel

Daten vom Server laden und auf der Webseite anzeigen

JSON AJAX JS-Objekte

AJAX ?



AJAX ?

[...]

Die Technologie ermöglicht es, einzelne Teile einer Webseite bei Bedarf asynchron zu laden, so dass sie dynamisch wird. Der angezeigte Inhalt lässt sich gezielt so manipulieren, ohne die komplette Seite neu zu laden.

<https://www.dev-insider.de/was-ist-ajax-a-782233/>

AJAX ?

Mit JavaScript Daten vom Server laden.

Lernziele: AJAX

Mit AJAX Daten vom Server laden können
Kontrollieren, ob Ladenvorgang erfolgreich war

geladene Daten in der Webseite darstellen

Ziel

Daten vom Server laden und auf der Webseite anzeigen

JSON AJAX JS-Objekte

Ziel

Daten vom Server laden und auf der Webseite anzeigen

JSON

AJAX

JS-Objekte

JSON ?

Die Daten werden nach einfachen Regeln in einer Textdatei strukturiert zusammengefasst.

JSON ist ein textbasiertes Daten-Austauschformat.

Daten werden in einer bestimmten Text-Struktur von einem Sender (Server) zu einem Empfänger (Webseite) übertragen.

Beispiel

```
{  
  "anrede": "Herr",  
  "name": {  
    "vorname": "Urs",  
    "nachname": "Thöny"  
  },  
  "interessen": ["Film", "Fahrrad fahren"],  
  "alter": 42  
}
```

```
let person1 = {  
    alter: 42,  
    anrede: 'Herr',  
    name: {  
        vorname: 'Urs',  
        nachname: 'Thöny'  
    },  
    interessen: ['Film', 'Fahrrad fahren']  
};
```

JavaScript-Objekt

```
{  
    "anrede": "Herr",  
    "name": {  
        "vorname": "Urs",  
        "nachname": "Thöny"  
    },  
    "interessen": ["Film", "Fahrrad fahren"],  
    "alter": 42  
}
```

JSON

JSON = JavaScript Object Notation

Lernziele: JSON

JSON-Regeln kennen

Daten im korrekten JSON-Syntax speichern können

JSON mit AJAX vom Server laden können

geladenes JSON in JavaScript-Objekt umwandeln können

Inhalte von JavaScript-Objekten (aus JSON erstellt) in der Webseite darstellen können

JavaScript-Objekte

(JS-Objekte)

Eigenschaften von JS-Objekten

Ein Objekt funktioniert als eine Zuordnungsliste, die unter bestimmten Namen weitere Unterobjekte, auch Member genannt, speichert.

Diese Unterobjekte teilt man in Eigenschaften und Methoden.

Methoden sind ausführbare Funktionen, die dem Objekt zugeordnet sind.

Eigenschaften sind alle nicht ausführbaren Unterobjekte.

Sie können als mit dem Objekt verbundene Variablen erklärt werden.



Syntax

Ein Objekt wird immer durch geschweifte Klammern eingeschlossen.

Nicht alles, was in geschweiften Klammern steht ist ein Objekt.

```
let beispiel = {  
}
```

Syntax - Objekt

Jede Objekt-Eigenschaft beginnt mit einem Eigenschaftsname, gefolgt von einem Doppelpunkt und dem Eigenschaftswert.

Eigenschaftnamen werden **nicht** in Anführungszeichen geschrieben.

Bei Eigenschaftnamen wird Gross- und Kleinschreibung unterschieden.

Konvention: Eigenschaftnamen werden kleingeschrieben.

```
let beispiel = {  
    eineEigenschaft : 'Der Wert einer Eigenschaft'  
}
```



Syntax

Mehrere Eigenschaften werden durch Kommata getrennt.

Nach dem letzten Eigenschaftswert darf kein Komma stehen.

```
let beispiel = {  
    ganzzahl : 42,  
    fliesskommazahl : 13.37,  
    eigenschaftsname : 'Eigenschaftswert'  
}
```

The code example is annotated with three blue speech bubbles:

- A bubble above the first comma after "ganzzahl" contains the word "Komma".
- A bubble above the second comma after "fliesskommazahl" contains the word "Komma".
- A bubble next to the string 'Eigenschaftswert' contains the text "kein Komma".

Syntax – Zusammenfassung

Ein Objekt wird immer durch geschweifte Klammern eingeschlossen.

Nicht alles, was in geschweiften Klammern steht ist ein Objekt.

Jede Objekt-Eigenschaft beginnt mit einem Eigenschaftsname, gefolgt von einem Doppelpunkt und dem Eigenschaftswert.

Die Eigenschaftnamen werden **nicht** in Anführungszeichen geschrieben.

Mehrere Eigenschaften werden durch Kommata getrennt.

Nach dem letzten Eigenschaftswert darf kein Komma stehen.

Syntax - Objekteigenschaft

Eine Eigenschaft ist wie eine Variable innerhalb eines Objekts. Ebenso wie Variablen kann sie jede Eigenschaft einen unterschiedlichen Datentypen enthalten:

- Zeichenkette in Anführungszeichen (einfach oder doppelt)
- Zahl ohne Anführungszeichen, Punkt als Dezimaltrenner
- Objekt in geschweiften Klammern
- Array in eckigen Klammern
- **undefined** kleingeschrieben, ohne Anführungszeichen
- **null** kleingeschrieben, ohne Anführungszeichen
- **bool** kleingeschrieben, ohne Anführungszeichen

gutes Beispiel

Geschweifte Klammern öffnen ...

```
let gutesBeispiel = {  
    ganzzahl : 42,  
    fliesskommazahl : 13.37,  
    zeichenkette : 'Hallo Welt',  
    unterobjekt : {  
        vorname : 'Urs',  
        nachname : 'Thöny'  
    },  
    einArray : ['foo', 'bar'],  
    undefinierbar : undefined,  
    nix : null,  
    entwederOder : true  
}
```

Zeichenketten können in einfachen oder doppelten Anführungszeichen stehen.

undefined ist korrekt, wird jedoch selten als Wert vergeben.

null ist korrekt, wird jedoch selten als Wert vergeben.

undefined, null und true/false müssen in Kleinbuchstaben geschrieben werden.

... und schliessen das JS-Objekt

SCHLECHTES Beispiel

Eigenschaftnamen werden in JS-Objekten (auch in Unterobjekten) immer ohne Anführungszeichen geschrieben!

```
let schlechtesBeispiel = {  
    ganzzahl : "42",  
    fliesskommazahl : 13,37,  
    zeichenkette : 'Hallo Welt'  
    unterobjekt : ({  
        "vorname" : "'Urs'",  
        "nachname" : Thöny  
    }),  
    einArray : {'foo', 'bar'},  
    undefinierbar : "undefined",  
    nix : NULL,  
    entwederOder : FALSE  
}
```

Das ist keine Zahl, sondern eine Zeichenkette.

Der Dezimaltrenner in Fliesskommawerten muss ein Punkt sein.

Hier fehlt das Komma.

Auch Unterobjekte stehen in geschweiften Klammern.

Hier fehlen die Anführungszeichen.

Diese Zeichenkette beinhaltet die einfachen Anführungszeichen.
Korrekt, aber unschön!

Arrays stehen immer in eckigen Klammern.

Das ist nicht undefined, sondern eine Zeichenkette.

null muss in Kleinbuchstaben geschrieben werden,
undefined übrigens auch.

true und false müssen auch in Kleinbuchstaben geschrieben werden.

Dot.Syntax – Zugriff auf Eigenschaften

Auf Eigenschaften und ihre Werte zuzugreifen ist sehr einfach. Wir verwenden den **Dot.Syntax**.

Mit dem Objektnamen rufen wir das komplette **Objekt** ab.

Um auf den **Wert einer Eigenschaft** zuzugreifen schreiben wir den Objektnamen und hängen getrennt durch einen Punkt den Eigenschaftnamen an.

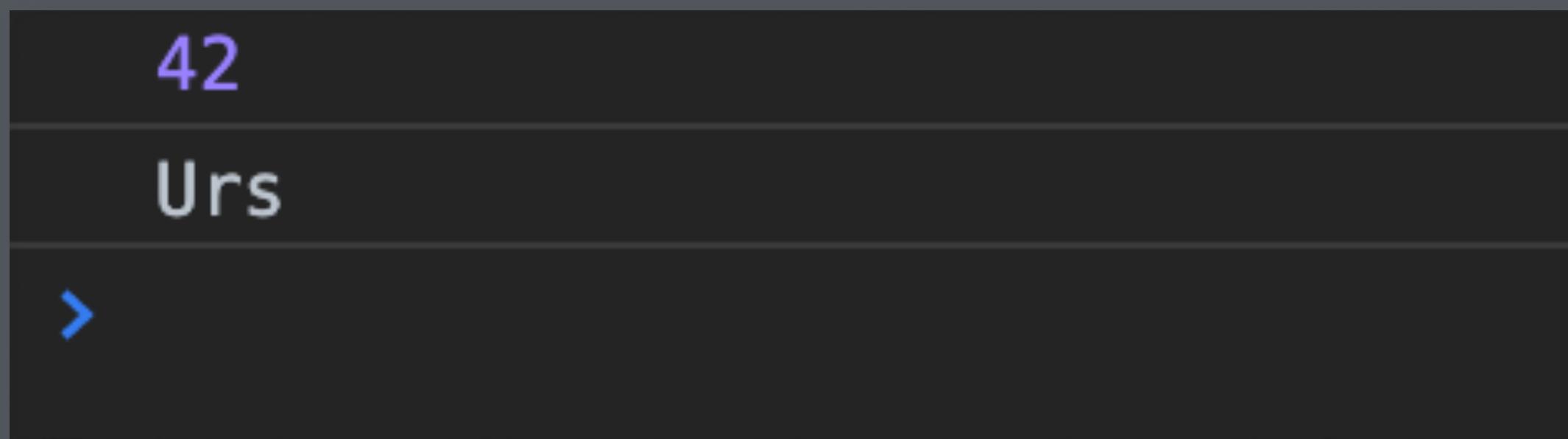
```
let wert1 = objekt.eigenschaft;
```

Um auf den **Wert einer Untereigenschaft** zuzugreifen schreiben wir den Objektnamen und hängen getrennt durch einen Punkt den Eigenschaftnamen an, woran wir wiederum durch einen Punkt getrennt den Untereigenschaftnamen anhängen.

```
let wert2 = objekt.eigenschaft.untereigenschaft;
```

Beispiel

```
let person1 = {  
    alter: 42,  
    anrede: 'Herr',  
    name: {  
        vorname: 'Urs',  
        nachname: 'Thöny'  
    },  
    interessen: ['Film', 'Fahrrad fahren'],  
};  
  
console.log(person1.alter);  
console.log(person1.name.vorname);
```



```
42  
Urs  
>
```

A dark terminal window showing the output of the JavaScript code. The first line '42' is in blue, indicating it's a number. The second line 'Urs' is in green, indicating it's a string. A blue '>' symbol at the bottom left indicates the prompt.

JS-Objekte – Übung 1

```
/* JS-Objekte | Übung 1 | 01_objekt_grundlagen
* 1. Erstellen Sie für den Kanton Bern ein JavaScript-Objekt mit dem Namen "kanton1"
* Das Objekt soll folgende Eigenschaften haben:
*   name,
*   amtssprache,
*   hauptort,
*   flaeche und für die Eigenschaft
*   bevoelkerung die Untereigenschaften
*   einwohner,
*   quote.
* Die Informationen entnehmen Sie der Seite:
*   https://de.wikipedia.org/wiki/Kanton\_Bern
* 2. Rufen Sie in der Konsole des Browsers die einzelnen Objektwerte auf.
*/
```

```
> kanton1
< ▶ {name: "Bern", amtssprache: Array(2), hauptort: "Bern", flaeche: 5959.24, bevoelkerung: {...}}
> kanton1.flaeche
< 5959.24
> kanton1.bevoelkerung.einwohner
< 1034977
>
```

Eigenschaftswerte in HTML anzeigen

Jetzt, wo wir auf Eigenschaftswerte eines Objektes zugreifen können, wollen wir sie auch in der Webseite anzeigen.

Dazu müssen wir die Werte in ein HTML-Element schreiben.

Vorgehen:

HTML-Element mit `document.querySelector()` identifizieren und ...

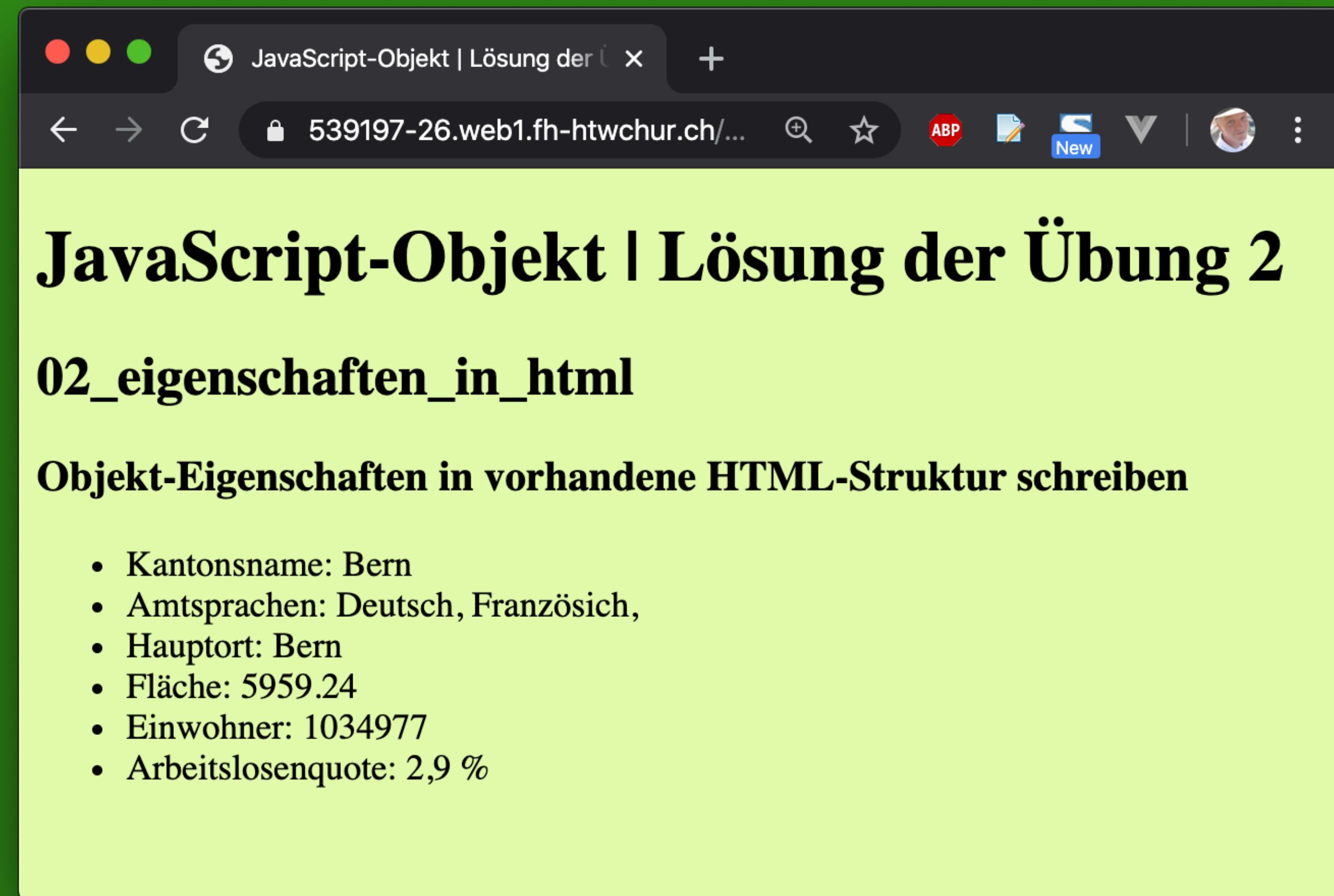
```
document.querySelector('#person_anrede')
```

... Eigenschaftswert mit `.textContent` in das Element schreiben.

```
document.querySelector('#person_anrede').textContent = person1.anrede;
```

JS-Objekte – Übung 2

```
/* JS-Objekte | Übung 2 | 02_eigenschaften_in_html
* Übertragen Sie die Werte aus dem JS-Objekt kanton1 in die li-Elemente der HTML-Liste.
*/
```



Liste aus Eigenschaftswerten erstellen

In Übung 1 haben wir die Eigenschaftswerte in eine vorbereitete HTML-Liste geschrieben.

Jetzt werden wir die Angaben zur Amtssprache in als eine eigene Liste innerhalb der vorhandenen Liste anzeigen.

Verschachtelte Listen sind nichts anderes, als eine eigene Liste innerhalb eines vorhandenen Listenpunktes.

siehe Wiki-selfHTML: [HTML/Textstrukturierung/ul](#)

- Kantonsname: Bern
- Amtssprachen:
 - Deutsch
 - Französisch
- Hauptort: Bern
- Fläche: 5959.24
- Einwohner: 1034977
- Arbeitslosenquote: 2,9 %

Liste aus Eigenschaftswerten erstellen

Vorgehen:

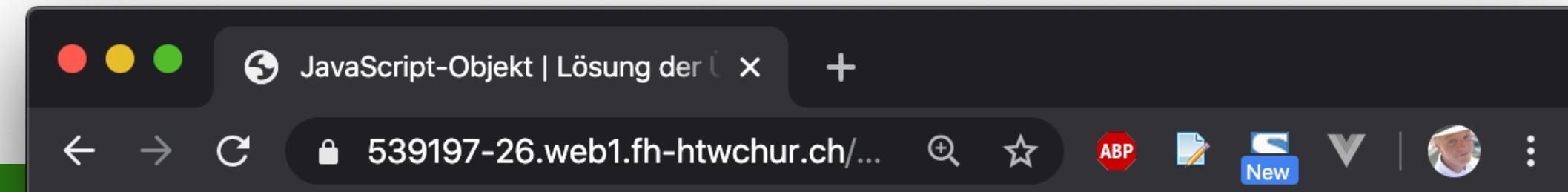
Mit `document.createElement('ul')` in JavaScript eine neue Liste erzeugen und in einer Variablen speichern.

Das Array mit den Amtssprachen durchlaufen und für jedes Element einen neuen Listenpunkt erstellen, den aktuellen Arraywert in den Listenpunkt schreiben und mit `.appendChild()` den Listenpunkt in die gerade erstellte Liste schreiben.

Zuletzt die neue Liste mit `.appendChild()` in den richtigen Listenpunkt schreiben.

JS-Objekte – Übung 3

```
/* JS-Objekte | Übung 3 | 03_liste_aus_eigenschaften_erstellen
* 1. Erstellen Sie für die Werte des Array kanton1.amtssprache mit Hilfe von JavaScript
eine ungeordnete Liste (ul-Element) und ...
* 2. fügen in diese Liste für jedes Array-Element ein HTML-Listen-Elemen (li-Element) mit
dem entsprechenden Wert ein.
* 3. Fügen Sie die vollständige HTML-Liste (ul-Element) in das HTML-Element mit der id
"kanton_hauptort" ein.
*/
```



The screenshot shows a web browser window titled "JavaScript-Objekt | Lösung der Übung 3". The URL in the address bar is "539197-26.web1.fh-htwchur.ch/...". The page content is as follows:

JavaScript-Objekt | Lösung der Übung 3

03_liste_aus_eigenschaften_erstellen

ungeordnete Liste aus dynamisch in JavaScript aus Untereigenschaften des Objekts erstellen und in HTML einfügen

- Kantonsname: Bern
- Amtsprachen:
 - Deutsch
 - Französisch
- Hauptort: Bern
- Fläche: 5959.24
- Einwohner: 1034977
- Arbeitslosenquote: 2,9 %

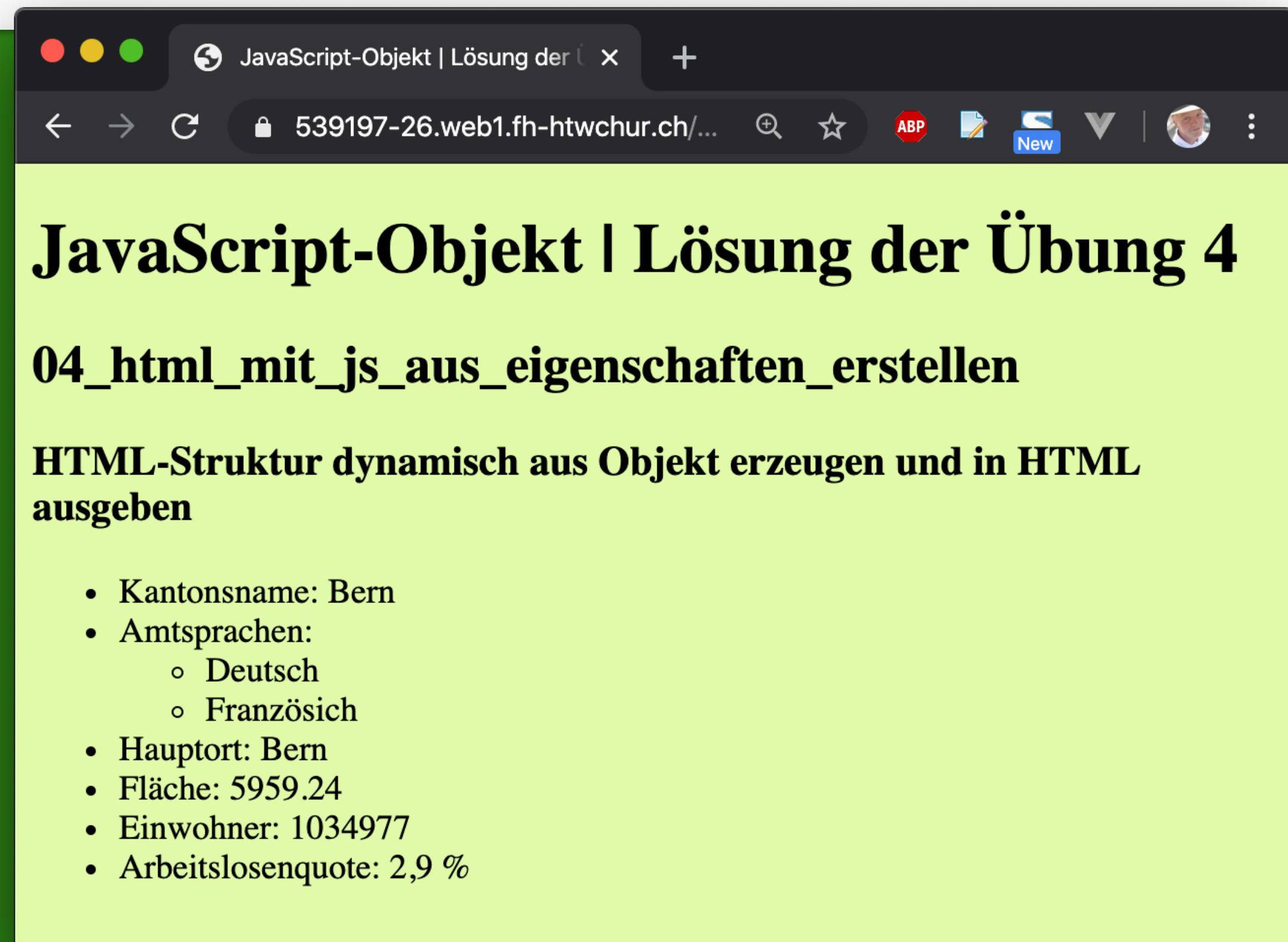
HTML mit JavaScript aus Eigenschaften erstellen

Wir können auch alle Angaben aus dem Objekt durch JavaScript in HTML darstellen.

So können wir flexibel auf unterschiedliche Angaben innerhalb verschiedener Objekte reagieren.

JS-Objekte – Übung 4

```
/* JS-Objekte | Übung 4 | 04_html_mit_js_aus_eigenschaften_erstellen
 * 1. Erstellen Sie die HTML-Struktur mit Hilfe von JavaScript ...
 * 2. und schreiben Sie die Eigenschaftswerte aus "kanton1" in die entsprechenden Elemente
 * 3. Fügen Sie zuletzt die ganze, in JS erstellte HTML-Struktur als ChildNode in das HTML-
Element mit der id "container" ein.
 */
```

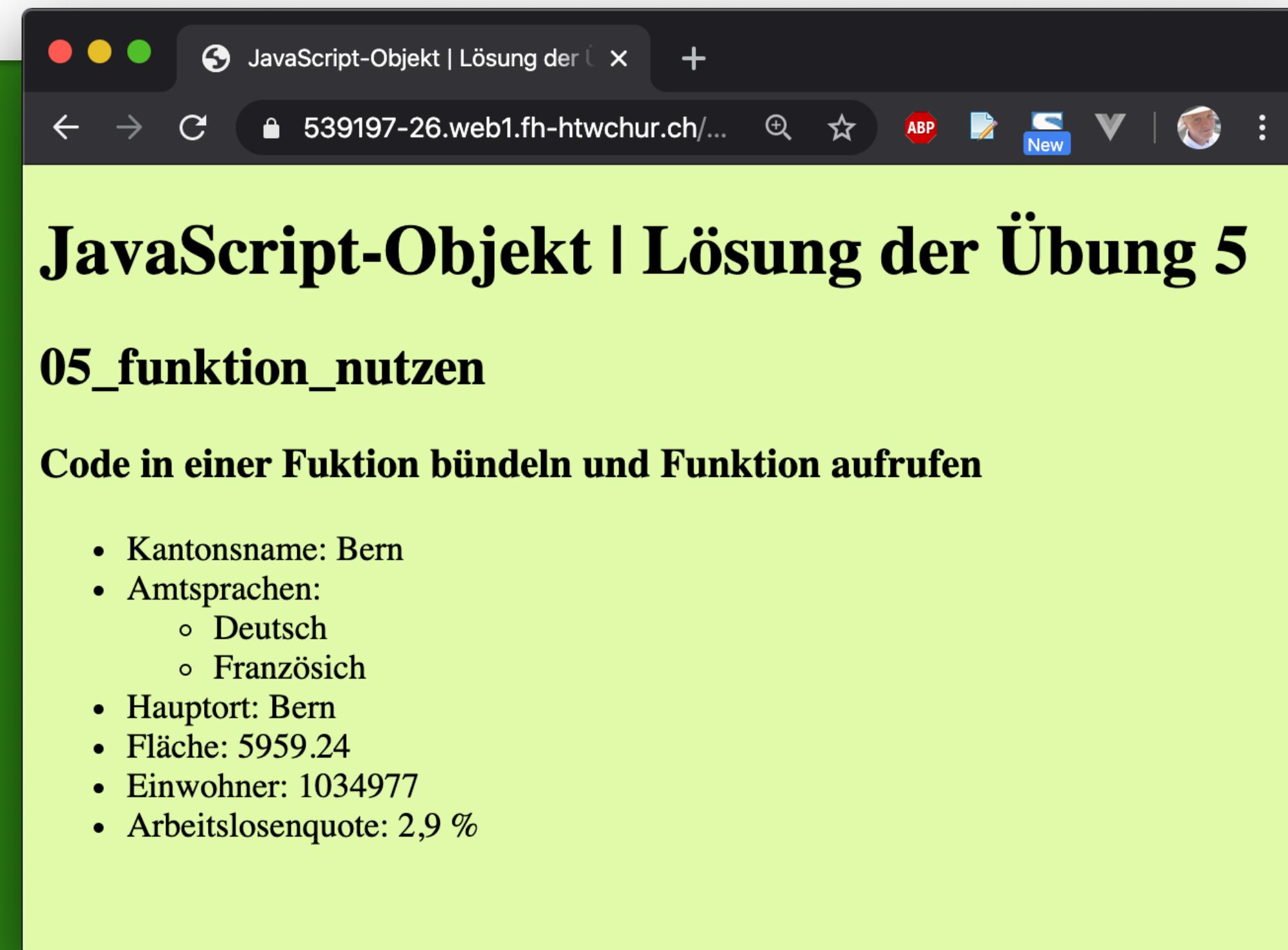


eine Funktion zur Darstellung nutzen

Noch flexibler wird es, wenn wir alle Befehle zur Erzeugung und Darstellung des Arrays in einer Funktion bündeln.
(Vorbereitung für Übung 6)

JS-Objekte – Übung 5

```
/* JS-Objekte | Übung 5 | 05_funktionen_nutzen
 * 1. Packen Sie den kompletten Aufbau in eine Funktion mit dem Namen "kanton_anzeigen()"
 * mit dem Funktionsparameter "kanton_param)".
 *      Dem Funktionsparameter "kanton_param" übergeben Sie später beim Funktionsaufruf das
 * Objekt "kanton1".
 * 2. Ersetzen Sie innerhalb der Funktion "kanton1" durch "kanton_param".
 * 3. Schreiben Sie "anzeige" als Kind-Element in das HTML-Element mit der id="container".
 * 4. Rufen Sie die Funktion auf mit dem JS-Objekt "kanton1" als Parameter auf.
 */
```



mehrere Objekte mit einer Funktion anzeigen

Mit Hilfe der Funktion können wir weitere Objekte ganz einfach anzeigen.

JS-Objekte – Übung 6

```
/* JS-Objekte | Lösung der Übung 6 | 06_mehrere_objekte_mit_funktionen_anzeigen
* 1. Erstellen Sie für den Kanton Graubünden ein JavaScript-Objekt mit dem Namen "kanton2"
* Das Objekt soll die selben Eigenschaften haben wie "kanton1"
* Die Informationen entnehmen Sie der Seite:
*     https://de.wikipedia.org/wiki/Kanton\_Graub%C3%BCnden
* 2. Rufen Sie zusätzlich die Funktion auf mit dem JS-Objekt "kanton2" als Parameter auf.
*/
```



JS-Objekte – Aufgabe 1

```
/* JS-Objekte | Aufgabe 1 | 01_bus_objekt_erreichen
/* Bilden Sie den Verlauf der Churer Buslinien 6 und 9 als JS-Objekt ab.
/* Jedes Bus-Objekt muss die Eigenschaften liniennummer, kurzbeschreibung und haltestellen
beinhalten.
/* Den Linienverlauf der Buslinien entnehmen Sie den Seiten
/*      https://churbus.ch/linie/linie-6 und
/*      https://churbus.ch/linie/linie-9
/* ***** */
```

Keine Anzeige im Browser

JS-Objekte – Aufgabe 2

```
/* JS-Objekte | Aufgabe 2 | 02_bus_objekt_anzeigen
/* Erstellen Sie eine Funktion "busobjekt_in_html_anzeigen()" mit dem Funktionsparameter
"bus_parameter",
/* Die Funktion gibt ein alle Angaben zur Buslinie als HTML zurück.
/* Wenden Sie die Funktion mit den JS-Objekten an und geben Sie das Ergebnis als Child-
Elemente des HTML-Elements mit der id "container" im Browser aus.
/* ***** */
```



Feedback

anonymes Feedback

AJAX

AJAX war nie neu

AJAX war nie eine neue Erfindung.

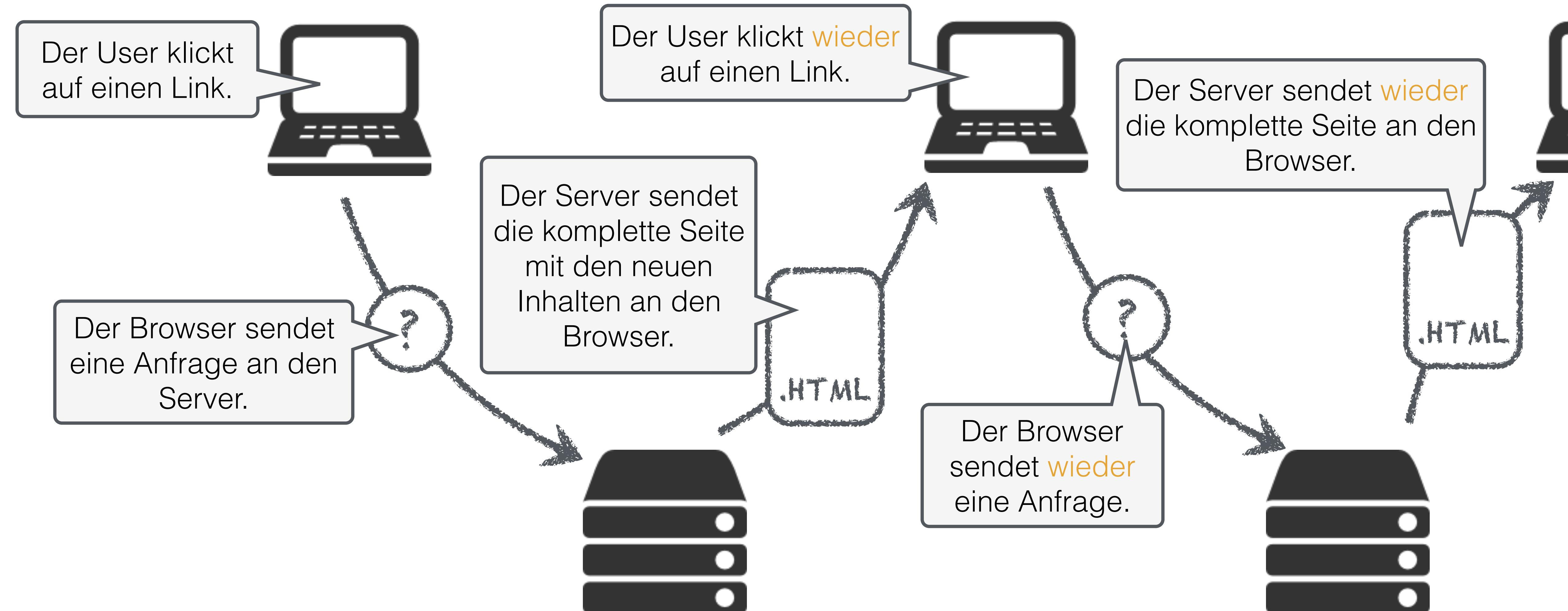
Die Techniken von AJAX existieren schon seit ca. 1998.

Zusammenfassung der Techniken erst ca. 2004.

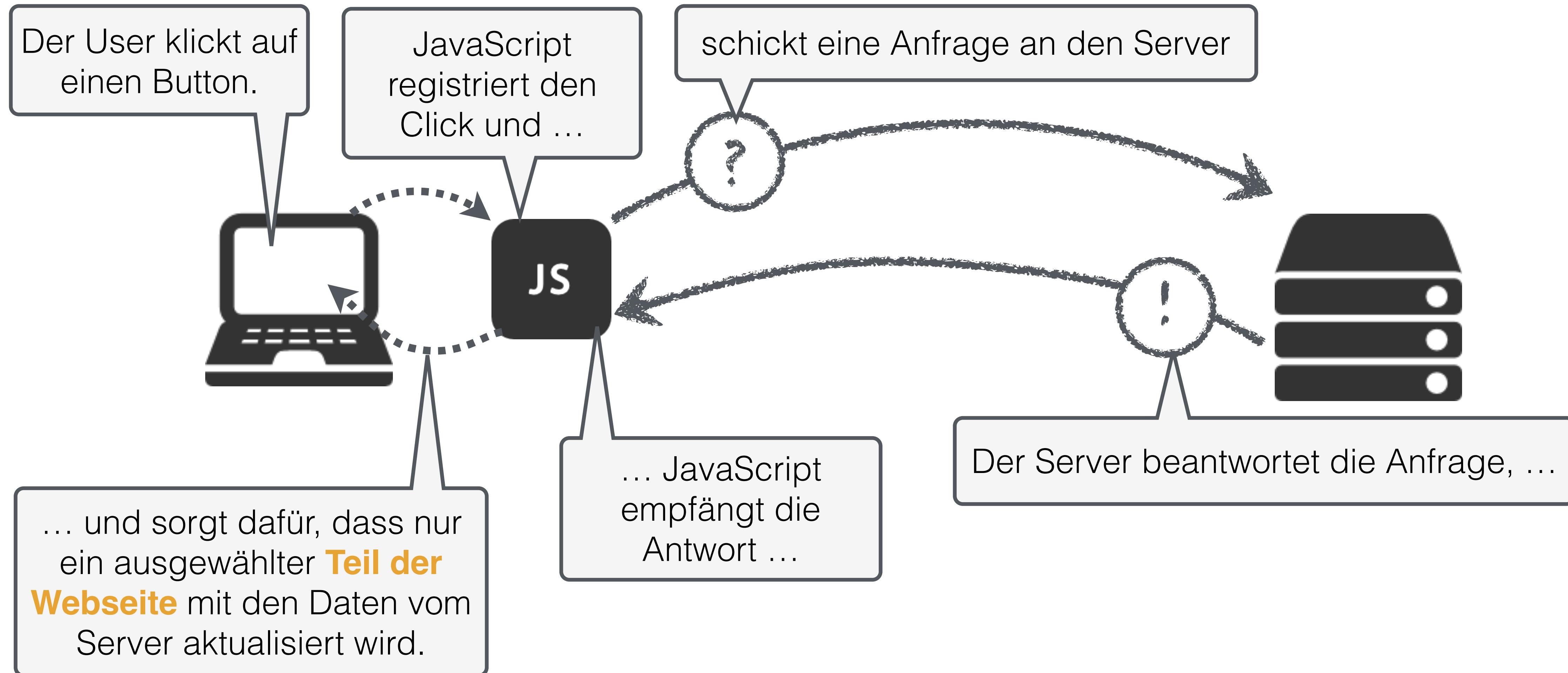
Was bedeutet AJAX

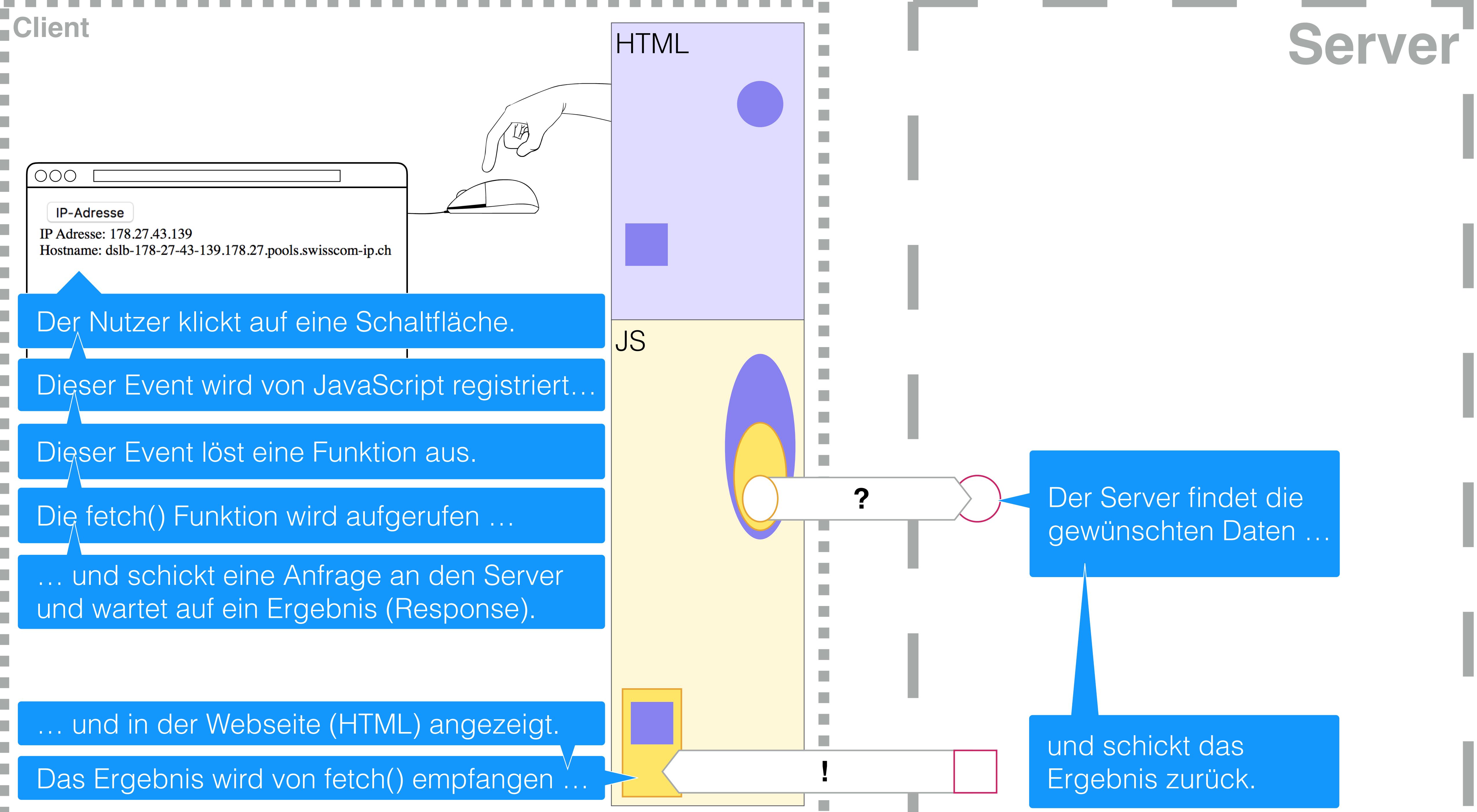
Aynchronous
JavaScript
And
XML

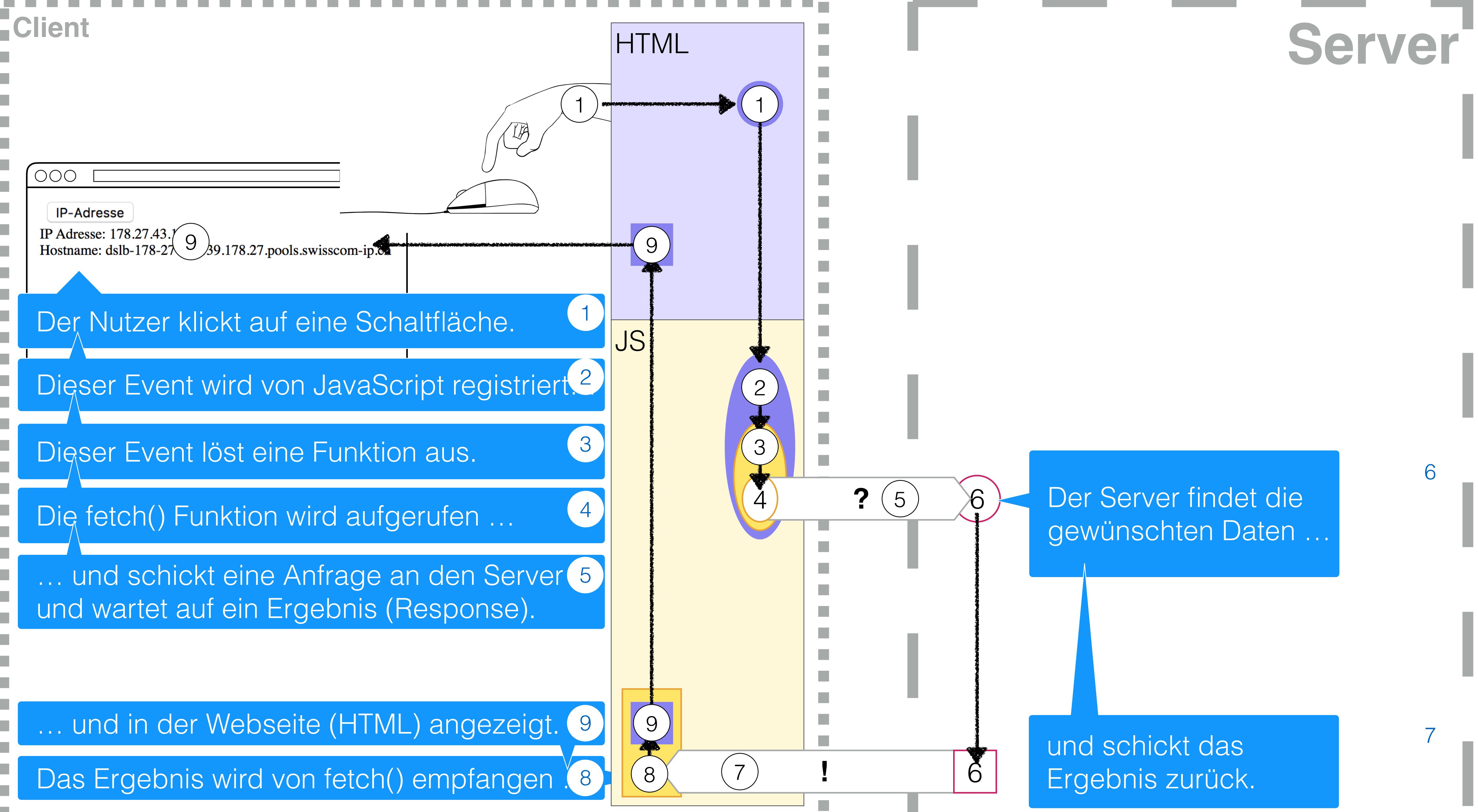
Serverkommunikation bisher



Serverkommunikation mit AJAX







fetch() Funktionsweise

Mit der `fetch()`-Funktion können wir mit JavaScript Dateien von einem Server laden.

`fetch()` ist ein relativ neuer Befehl, und er macht uns das Leben sehr viel einfacher.

Die `fetch()`-Funktion arbeitet mit sog. Promises (Versprechen).

Als Parameter benötigt `fetch()` lediglich die URL der zu ladenden Datei.

Promises

Die `fetch()`-Funktion arbeitet mit sog. Promises (Versprechen).

Promises sind eine erweiterbare Kette von Funktionen.

Promises am Beispiel von fetch()

fetch() benötigt als Parameter die URL der zu ladenden Datei.

Wenn fetch() mit der URL aufgerufen wurde, verspricht fetch() eine Antwort (response) zu liefern.

Wir müssen angeben, welches Datenformat zu erwarten ist.

Wenn die Antwort vom Server eintrifft, verspricht fetch() Daten zu liefern.
Mit diesen Daten können wir jetzt Arbeiten.

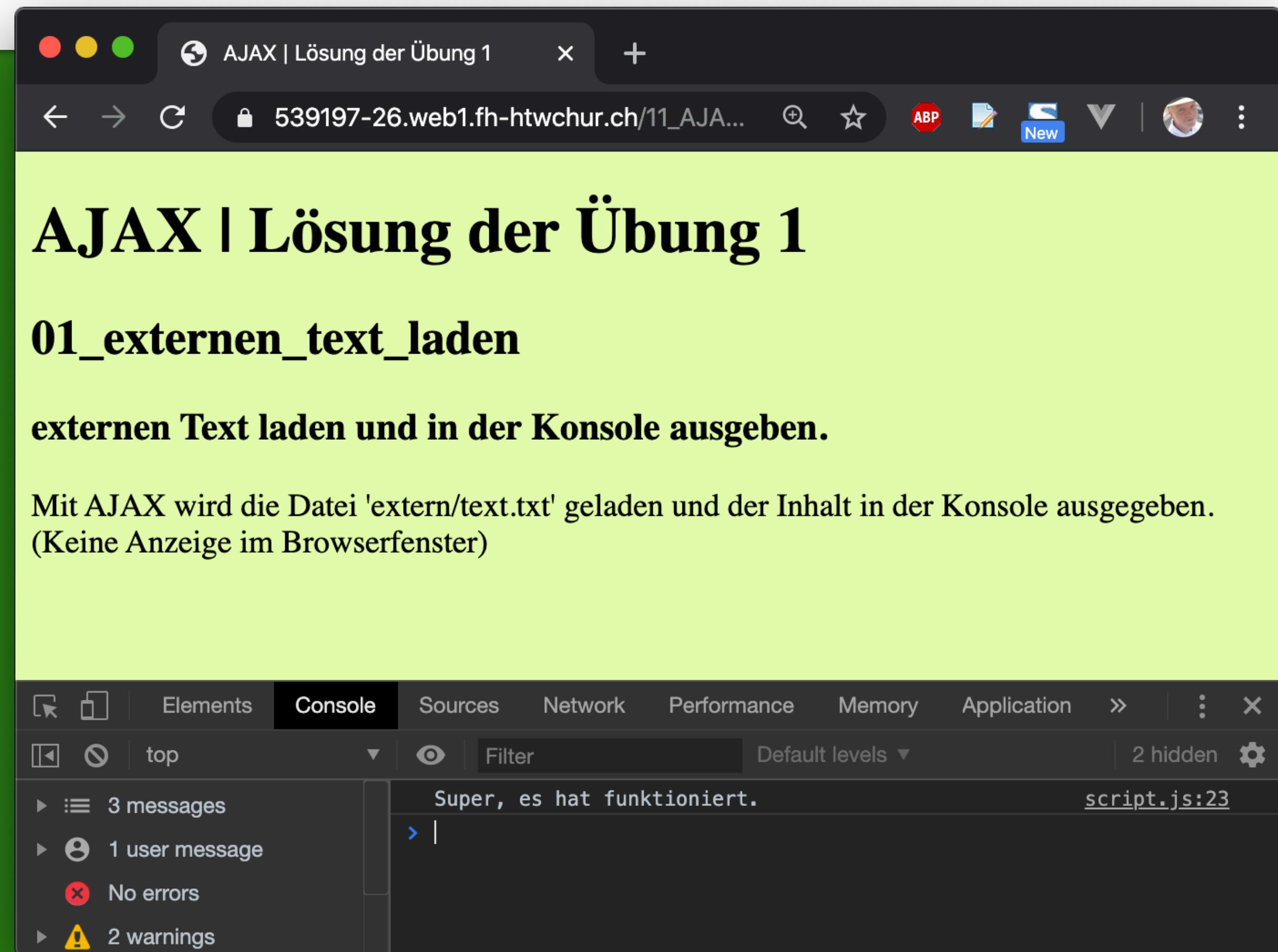
Sollte etwas nicht funktionieren,
können wir den Fehler einfangen (catchen).

fetch() Funktionsweise

```
// mit fetch() das Laden einer externen Datei starten.  
// Als Parameter benötigt fetch() den Pfad zur externen Datei.  
fetch('extern/text.txt')  
  // Der fetch() Aufruf erwartet eine Antwort (response)  
  .then((response) => {  
    // Definieren, welches Format die Antwort hat (wichtig für den nächsten Teil)  
    // hier text  
    return response.text();  
  })  
  // Wenn die Antwort eintrifft ...  
  .then((data) => {  
    // ... wird sie weiterverarbeitet (hier: Ausgabe in die Konsole)  
    console.log(data);  
  })  
  // Nur wenn etwas nicht funktioniert hat ...  
  .catch(function(error) {  
    // ... wird eine Fehlermeldung ausgegeben.  
    console.log('Error: ' + error.message);  
  });
```

AJAX – Übung 1

```
/* AJAX | Übung 1 | 01_externen_text_laden
/* 1. Laden Sie den Inhalt aus der Datei 'extern/text.txt' mit Hilfe der fetch()-API
/* 2. Geben Sie im ersten Promise an, welches Datenformat als Antwort erwartet wird.
/* 3. Geben Sie den geladenen Text in der Konsole aus.
/* Kontrollieren Sie das Ergebnis in der Konsole ihres Browsers.
/* *****
```

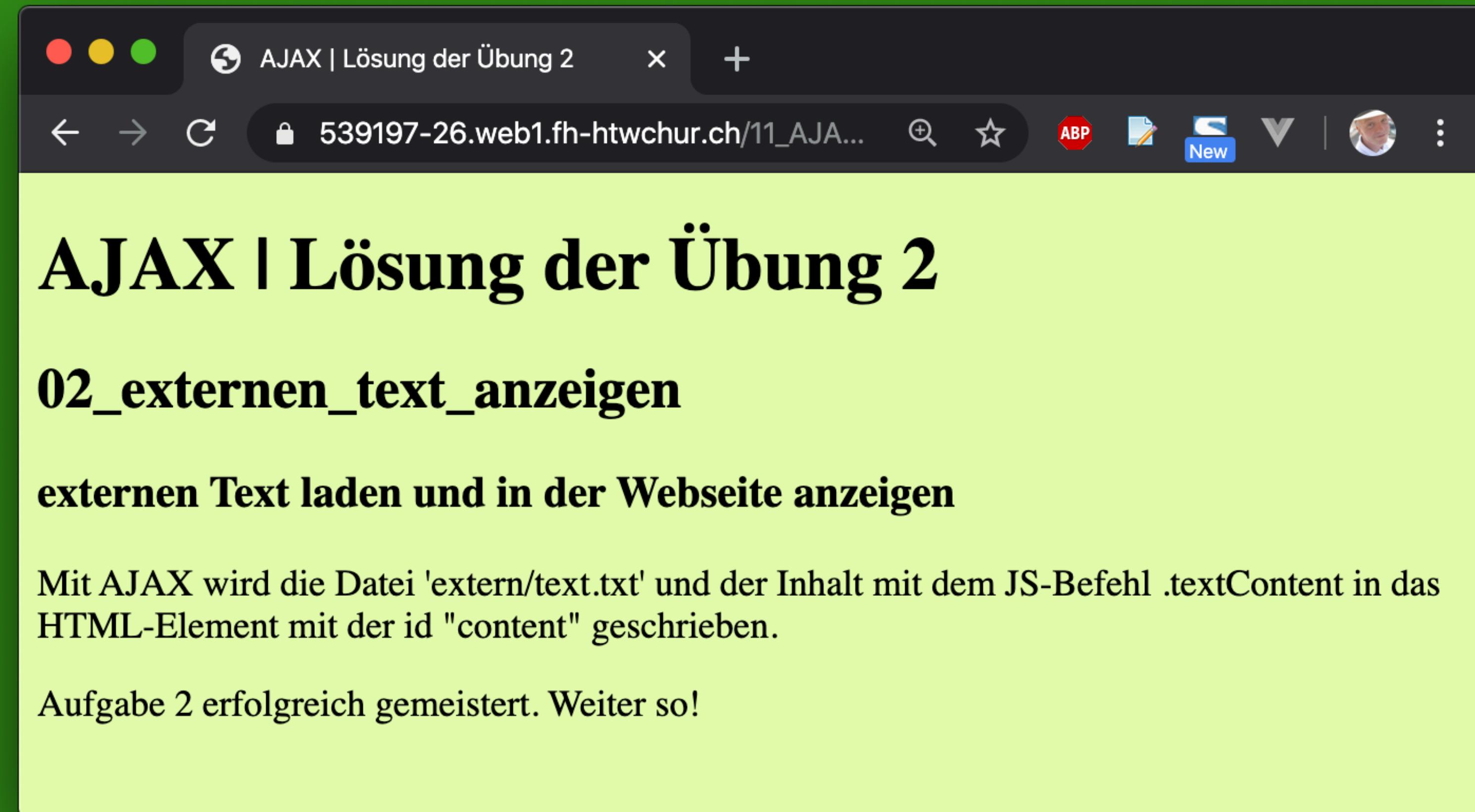


externen Text anzeigen

Den empfangenen Text können wir ganz einfach im Browserfenster ausgeben. Wir müssen ihn lediglich als Text in ein vorhandenes HTML-Element schreiben.

AJAX – Übung 2

```
/* AJAX | Übung 2 | 02_externen_text_anzeigen
/* 1. Geben Sie den geladenen Inhalt als textContent in dem HTML-Element mit der ID
'content' aus.
/* *****
```

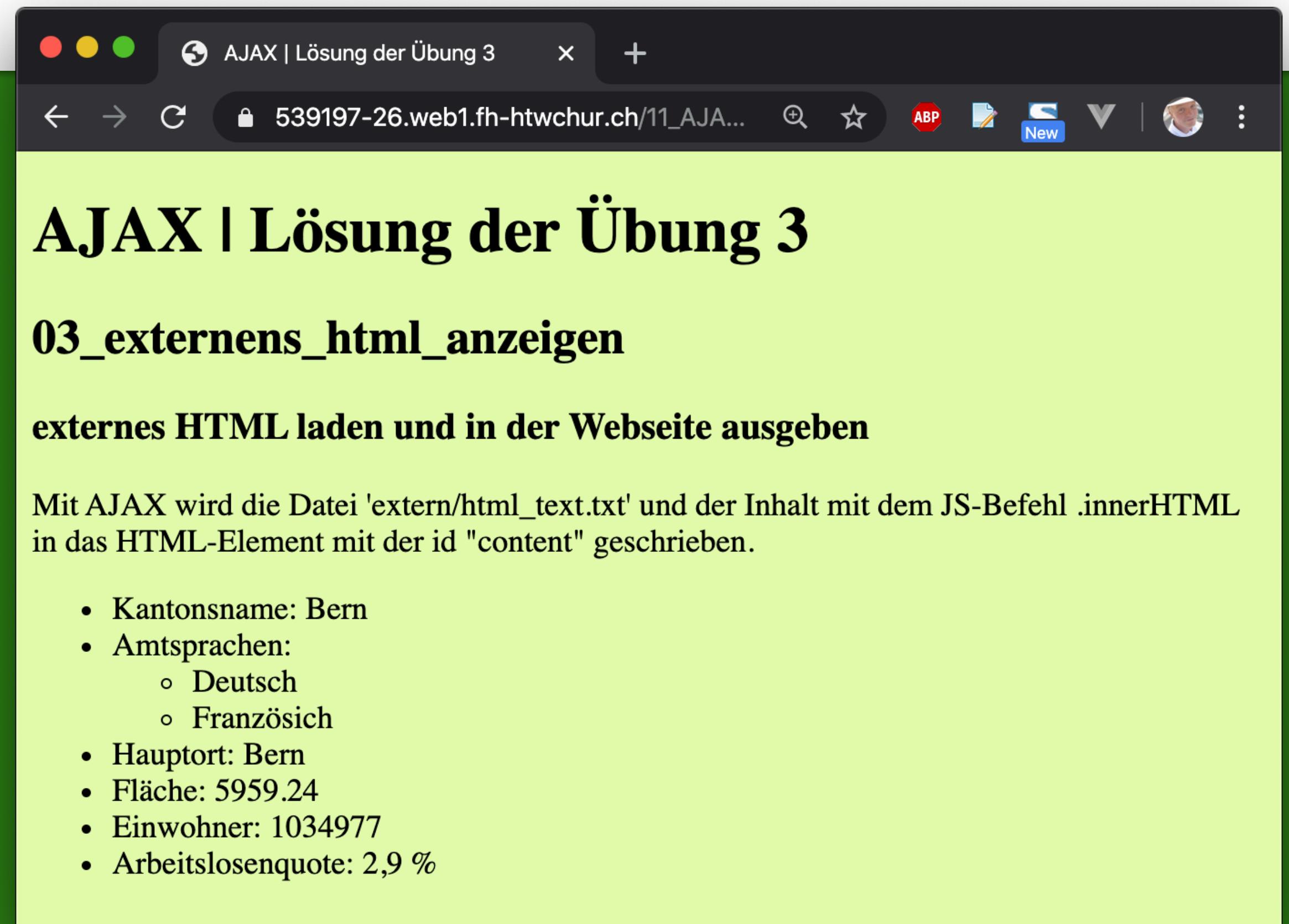


externes HTML anzeigen

Wenn in der geladenen Datei HTML-Code steht, können wir diesen in die Webseite einbauen. Wir müssen dazu den geladenen HTML-Code mit `.innerHTML()` in ein vorhandenes Element laden.

AJAX – Übung 3

```
/* AJAX | Übung 3 | 03_externes_html_anzeigen
/* 1. Laden Sie den Inhalt aus der Datei 'extern/html_kanton1.txt' mit Hilfe der fetch()-API
/* 2. Geben Sie den geladenen Inhalt als innerHTML in dem HTML-Element mit der ID 'content'
aus.
/* *****
```



The screenshot shows a web browser window titled "AJAX | Lösung der Übung 3". The address bar displays the URL "539197-26.web1.fh-htwchur.ch/11_AJA...". The main content area contains the following text:

AJAX | Lösung der Übung 3

03_externens_html_anzeigen

externes HTML laden und in der Webseite ausgeben

Mit AJAX wird die Datei 'extern/html_text.txt' und der Inhalt mit dem JS-Befehl .innerHTML in das HTML-Element mit der id "content" geschrieben.

- Kantonsname: Bern
- Amtssprachen:
 - Deutsch
 - Französisch
- Hauptort: Bern
- Fläche: 5959.24
- Einwohner: 1034977
- Arbeitslosenquote: 2,9 %

Inhalte interaktiv austauschen

Solch kleine Datenmengen beim laden der Seite mit `fetch()` abzurufen erscheint zunächst nicht sinnvoll.

Besser ist es, die Inhalte erst auf Anforderungen des Users zu laden.

Dazu müssen wir auf Ereignisse, die der User macht reagieren.

Es bietet sich an Buttons zu erstellen, die bei `'click'` den Ladeprozess auslösen.

Die Herausforderung besteht darin die Angabe über die zu ladende Datei, z. B. den Dateinamen, innerhalb des HTML-Codes des jeweiligen Buttons mitzugeben.

Hier hilft uns das data-*Attribut.

data-*

Mit **data-***-Attributen (Custom Data Attributes) haben Sie die Möglichkeit, Elementen eigene Attribute mitzugeben, die dann per Script ausgewertet oder mit CSS genutzt werden können.

In diesen Attributen können Sie Informationen, die nicht visuell präsentiert werden sollen, zur Verfügung stellen.

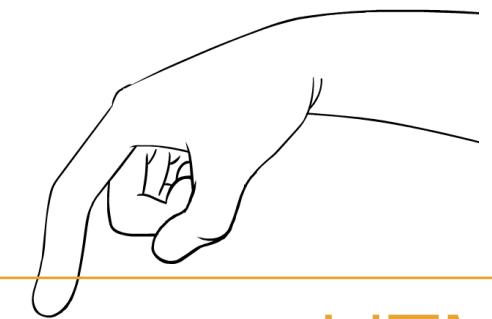
this in EventHandlern

Funktionen, die einen Event wie z. B. einen Mausklick verarbeiten, nennt man EventHandler.

Jeder Event hat einen sog. Auslöser, wie z. B. den Button, auf den geklickt wurde.

Innerhalb eines EventHandlers können wir mit dem Schlüsselwort **this** auf die Eigenschaften des Auslösers zugreifen.

data-* und this kombinieren



```
<div class="buttons">  
  <button class="btn" data-dateiname="html_kanton1">Bern</button><br>  
  <button class="btn" data-dateiname="html_kanton2">Graubünden</button><br>  
</div>
```

Bei Klick auf diesen Button ...

// Eine Liste (Array) mit allen Elementen mit der class "btn" wird in der Variablen "buttons" gespeichert.

```
let buttons = document.querySelectorAll('.btn');
```

... wird der EventHandler ausgelöst ...

// Die Liste "buttons" wird in einer Schleife durchlaufen.

```
for(let i = 0; i < buttons.length; i++){
```

// Jedem Listen-Element (jedem Button) wird ein Click-EventListener zugeordnet.

```
  buttons[i].addEventListener("click", function(){
```

// In der Variablen aktuellerDateiname speichern wir mit this.getAttribute("data-dateiname") ...

// ... den Inhalt des Attributs data-dateiname aus dem Button, der gerade geklickt wurde ...

// ... und dadurch den EventListener ausgelöst hat. (Dateiname ohne Extension)

```
    let aktuellerDateiname = this.getAttribute("data-dateiname");
```

... und in der Variablen aktuellerDateiname der Wert *html_kanton2* gespeichert

```
// Beim Click auf den Button wird die Funktion zeige_externen_inhalt() Aufgerufen.
```

// Als Parameter wird der Inhalt des data-dateiname-Attributs, des jeweiligen Buttons mitgegeben.

```
  zeige_externen_inhalt(aktuellerDateiname);
```

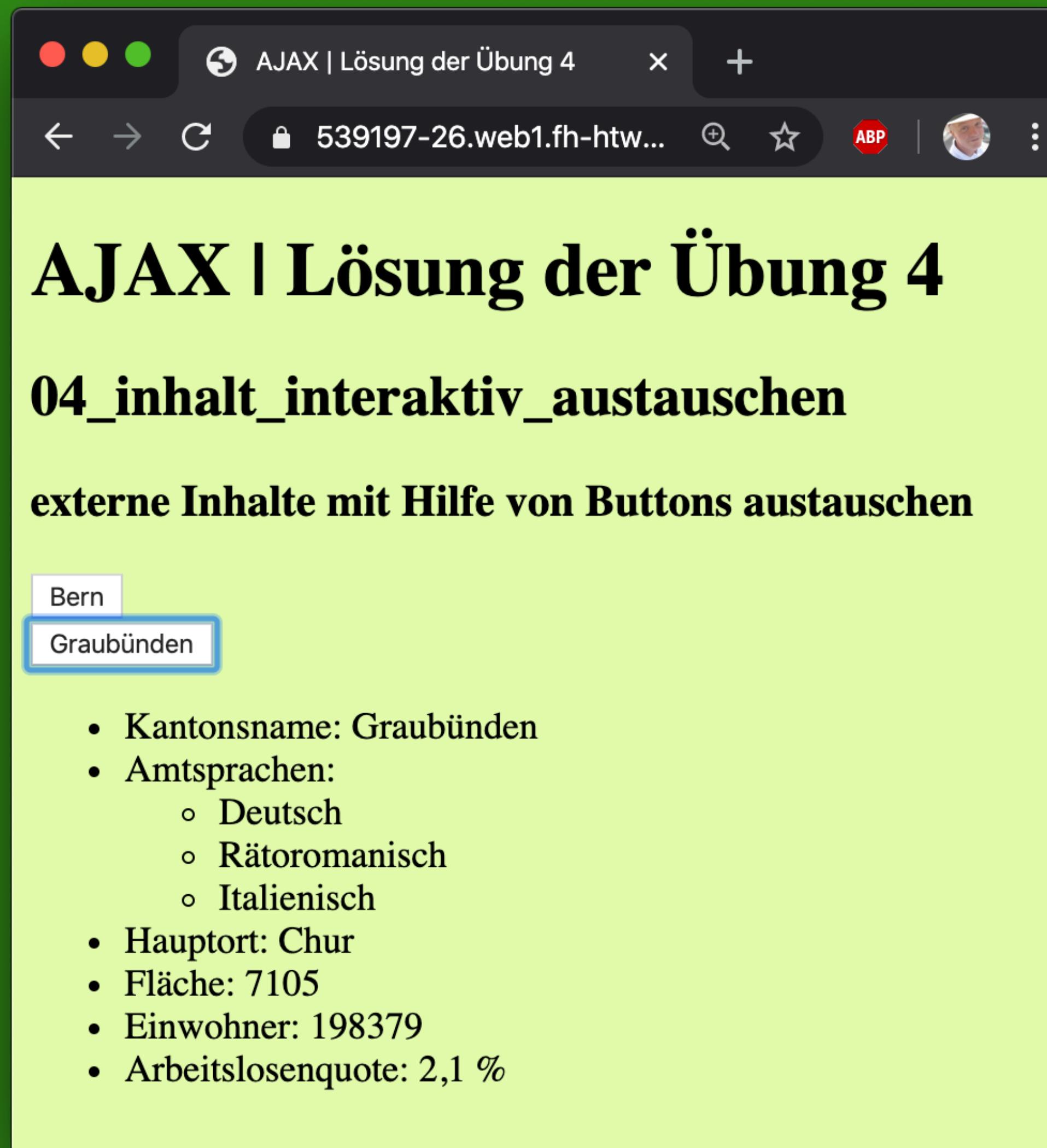
```
}
```

AJAX – Übung 4

```
/* AJAX | Übung 4 | 04_inhalt_interaktiv_austauschen
/* 1. Siehe: zugehörige HTML-Datei. (../index.html)
/* 2. Packen Sie den fetch()-Aufruf in eine Funktion mit dem Namen zeige_externen_inhalt()
/* 3. Als Parameter (dateiname) erhält die Funktion den Dateinamen der externen Datei.
(Dateiname ohne Extension)
/* 4. Setzen Sie den Pfad der zu ladenden Datei aus
/*      dem Ordnernamen mit folgendem Schrägstrich,
/*      dem Parameter "dateiname",
/*      der Extension mit vorstehendem Punkt zusammen,
/*      und speichern Sie die so entstandene Zeichenkette in der Variablen "url".
/* 5. Rufen Sie fetch() mit der Variablen url als Parameter auf
/* 6. Geben Sie den geladenen Inhalt als innerHTML in dem HTML-Element mit der ID 'anzeige'
aus.

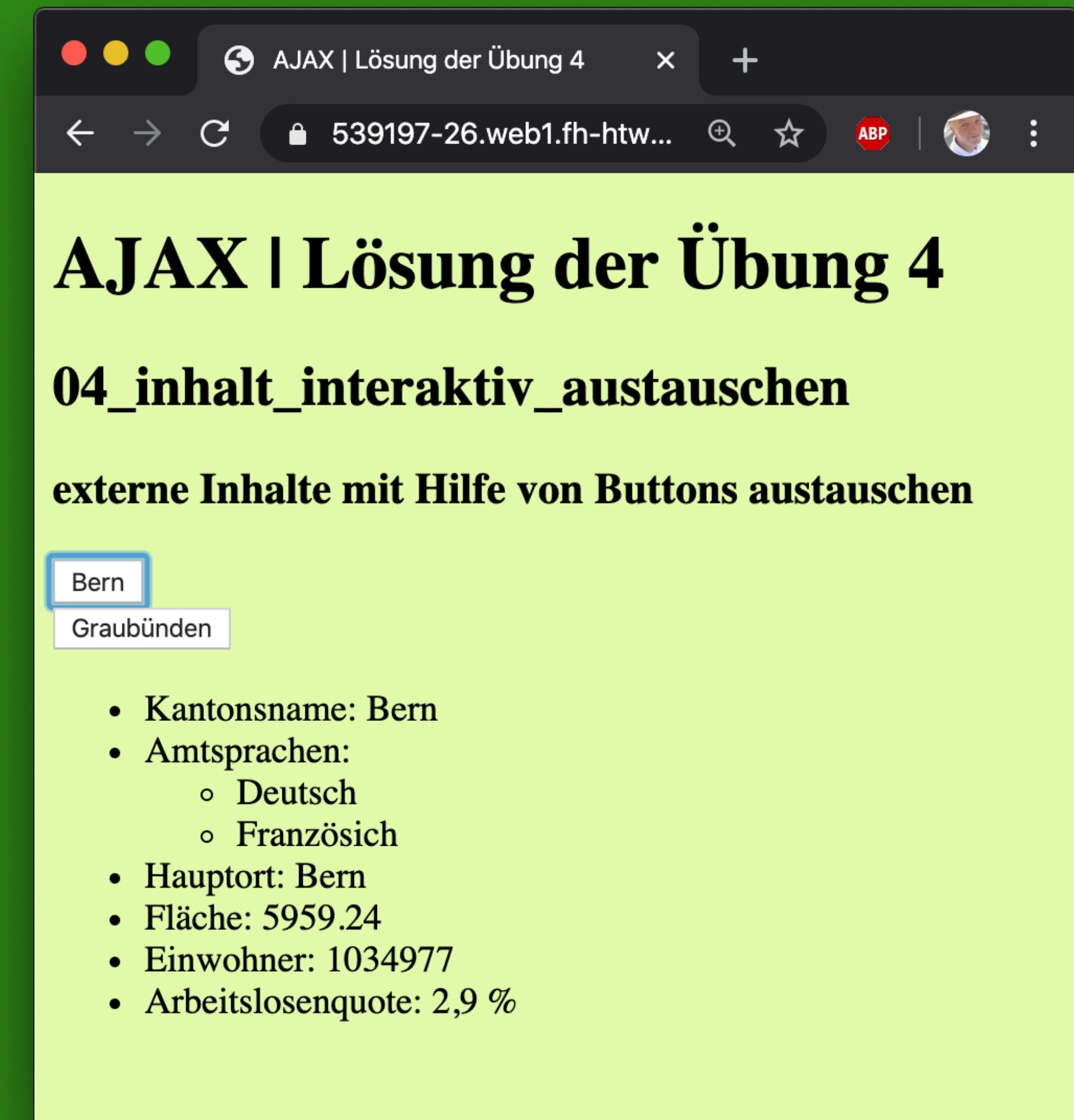
/* 7. Speichern Sie mit Hilfe von .querySelectorAll() alle Buttons mit der class="btn" in
einer Variablen als Array.
/* 8. Durchlaufen Sie das Array mit einer for-Schleife ...
/* 9. ... und geben Sie mit Hilfe von .addEventListener() jedem Array-Element (Button) für
den EventListener "click" eine Funktion mit, ...
/* 10. ... die den Wert des Attributs data-dateiname des geklickten Buttons in einer Variable
speichert und ...
/* 11. ... unsere Funktion zeige_externen_inhalt() aufruft. Als Parameter erhält die Funktion
den Wert der soeben erstellten Variablen.
/* *****
```

AJAX – Übung 4



A screenshot of a web browser window titled "AJAX | Lösung der Übung 4". The URL is "539197-26.web1.fh-htw...". The page content displays the title "AJAX | Lösung der Übung 4" and the section "04_inhalt_interaktiv_austauschen". Below this, the heading "externe Inhalte mit Hilfe von Buttons austauschen" is shown. Two buttons are present: "Bern" and "Graubünden", with "Graubünden" being highlighted by a blue border. A list of facts follows:

- Kantonsname: Graubünden
- Amtssprachen:
 - Deutsch
 - Rätoromanisch
 - Italienisch
- Hauptort: Chur
- Fläche: 7105
- Einwohner: 198379
- Arbeitslosenquote: 2,1 %



A screenshot of a web browser window titled "AJAX | Lösung der Übung 4". The URL is "539197-26.web1.fh-htw...". The page content displays the title "AJAX | Lösung der Übung 4" and the section "04_inhalt_interaktiv_austauschen". Below this, the heading "externe Inhalte mit Hilfe von Buttons austauschen" is shown. Two buttons are present: "Bern" and "Graubünden", with "Bern" being highlighted by a blue border. A list of facts follows:

- Kantonsname: Bern
- Amtssprachen:
 - Deutsch
 - Französich
- Hauptort: Bern
- Fläche: 5959.24
- Einwohner: 1034977
- Arbeitslosenquote: 2,9 %

AJAX– Aufgabe 1

```
/* AJAX | Aufgabe 1 | 01_bus_austauschen
/* Erstellen Sie in der zugehörigen HTML–Datei für jede Churer Buslinei einen Button.
/* Funktionalisieren Sie die Buttons der zugehörigen Datei so,
/* Dass beim Klick auf einen Button die passende Text–Datei geladen
/* und der Inhalt als HTML dargestellt wird.
/* Die txt–Dateien befinden sich im Verzeichnis "extern".
/* ***** */
```

The screenshots show a repeating list of bus stops and their connections, likely generated by an Ajax call. The stops listed are:

- 1 Lachen – Bahnhofplatz – Plankis – Domat/Ems – Rhäzüns
- 1 Lachen – Bahnhofplatz – Plankis – Domat/Ems – Rhäzüns
- 2 Obere Au – Bahnhofplatz – Fürstewald
- 2 Obere Au – Bahnhofplatz – Kleinwaldegg
- 3 Austrasse – Bahnhofplatz – Stelleweg – Haldenstein
- 3 Austrasse – Bahnhofplatz – Trimmis – Untervaz
- 4 Austrasse – Bahnhofplatz – Seniorencentrum Cadonau
- 6 Bahnhofplatz – City West
- 9 Bahnhofplatz – Meiersboden
- 1 Lachen – Bahnhofplatz – Plankis – Domat/Ems – Rhäzüns
- 1 Lachen – Bahnhofplatz – Plankis – Felsberg
- 2 Obere Au – Bahnhofplatz – Fürstewald
- 2 Obere Au – Bahnhofplatz – Kleinwaldegg
- 3 Austrasse – Bahnhofplatz – Stelleweg – Haldenstein
- 3 Austrasse – Bahnhofplatz – Trimmis – Untervaz
- 4 Austrasse – Bahnhofplatz – Seniorencentrum Cadonau
- 6 Bahnhofplatz – City West
- 9 Bahnhofplatz – Meiersboden
- 1 Lachen – Bahnhofplatz – Plankis – Domat/Ems – Rhäzüns
- 1 Lachen – Bahnhofplatz – Plankis – Felsberg
- 2 Obere Au – Bahnhofplatz – Fürstewald
- 2 Obere Au – Bahnhofplatz – Kleinwaldegg
- 3 Austrasse – Bahnhofplatz – Stelleweg – Haldenstein
- 3 Austrasse – Bahnhofplatz – Trimmis – Untervaz
- 4 Austrasse – Bahnhofplatz – Seniorencentrum Cadonau
- 6 Bahnhofplatz – City West
- 9 Bahnhofplatz – Meiersboden
- 1 Lachen – Bahnhofplatz – Plankis – Domat/Ems – Rhäzüns
- 1 Lachen – Bahnhofplatz – Plankis – Felsberg
- 2 Obere Au – Bahnhofplatz – Fürstewald
- 2 Obere Au – Bahnhofplatz – Kleinwaldegg
- 3 Austrasse – Bahnhofplatz – Stelleweg – Haldenstein
- 3 Austrasse – Bahnhofplatz – Trimmis – Untervaz
- 4 Austrasse – Bahnhofplatz – Seniorencentrum Cadonau
- 6 Bahnhofplatz – City West
- 9 Bahnhofplatz – Meiersboden
- 1 Lachen – Bahnhofplatz – Plankis – Domat/Ems – Rhäzüns
- 1 Lachen – Bahnhofplatz – Plankis – Felsberg
- 2 Obere Au – Bahnhofplatz – Fürstewald
- 2 Obere Au – Bahnhofplatz – Kleinwaldegg
- 3 Austrasse – Bahnhofplatz – Stelleweg – Haldenstein
- 3 Austrasse – Bahnhofplatz – Trimmis – Untervaz
- 4 Austrasse – Bahnhofplatz – Seniorencentrum Cadonau
- 6 Bahnhofplatz – City West
- 9 Bahnhofplatz – Meiersboden
- 1 Lachen – Bahnhofplatz – Plankis – Domat/Ems – Rhäzüns
- 1 Lachen – Bahnhofplatz – Plankis – Felsberg
- 2 Obere Au – Bahnhofplatz – Fürstewald
- 2 Obere Au – Bahnhofplatz – Kleinwaldegg
- 3 Austrasse – Bahnhofplatz – Stelleweg – Haldenstein
- 3 Austrasse – Bahnhofplatz – Trimmis – Untervaz
- 4 Austrasse – Bahnhofplatz – Seniorencentrum Cadonau
- 6 Bahnhofplatz – City West
- 9 Bahnhofplatz – Meiersboden
- 1 Lachen – Bahnhofplatz – Plankis – Domat/Ems – Rhäzüns
- 1 Lachen – Bahnhofplatz – Plankis – Felsberg
- 2 Obere Au – Bahnhofplatz – Fürstewald
- 2 Obere Au – Bahnhofplatz – Kleinwaldegg
- 3 Austrasse – Bahnhofplatz – Stelleweg – Haldenstein
- 3 Austrasse – Bahnhofplatz – Trimmis – Untervaz
- 4 Austrasse – Bahnhofplatz – Seniorencentrum Cadonau
- 6 Bahnhofplatz – City West
- 9 Bahnhofplatz – Meiersboden
- 1 Lachen – Bahnhofplatz – Plankis – Domat/Ems – Rhäzüns
- 1 Lachen – Bahnhofplatz – Plankis – Felsberg
- 2 Obere Au – Bahnhofplatz – Fürstewald
- 2 Obere Au – Bahnhofplatz – Kleinwaldegg
- 3 Austrasse – Bahnhofplatz – Stelleweg – Haldenstein
- 3 Austrasse – Bahnhofplatz – Trimmis – Untervaz
- 4 Austrasse – Bahnhofplatz – Seniorencentrum Cadonau
- 6 Bahnhofplatz – City West
- 9 Bahnhofplatz – Meiersboden

JSON

JSON

JSON (JavaScript Object Notation) ist ein schlankes Datenaustauschformat, das für Menschen einfach zu lesen und zu schreiben und für Maschinen einfach zu parsen (Analysieren von Datenstrukturen) und zu generieren ist.

JSON

JSON ist ein universelles Daten-Austauschformat. Es wird von fast allen Programmiersprachen unterstützt.

Die Daten werden nach einfachen Regeln in einer Textdatei strukturiert zusammengefasst.

JSON ist ein textbasiertes Daten-Austauschformat.

Daten werden in einer bestimmten Text-Struktur von einem Sender (Server) zu einem Empfänger (Webseite) übertragen.

JSON-Regeln

Wie ein JavaScript-Objekt mit vier Unterschieden:

1. Eigenschaftnamen müssen in doppelten Anführungszeichen stehen.
2. Zeichenketten müssen in doppelten Anführungszeichen stehen.
Einfache Anführungszeichen funktionieren nicht.
3. Den Eigenschaftswert **undefined** gibt es in JSON nicht.
4. In JSON können keine Methoden definiert werden.

JSON – Übung 1

```
/* JSON | Übung 1 | 01_JSON_erreichen
/* Erstellen Sie in der Datei extern/kanton1.json
/* aus dem unten stehenden Code für ein JS-Objekt
/* einen valieden JSON-Code
/*
/* Unterschiede zu JS-Objekten:
/*   Es sind nur doppelte Anführungszeichen erlaubt
/*   Alle Eigenschaftnamen (links vom Doppelpunkt) müssen in Anführungszeichen stehen.
/* ****
{

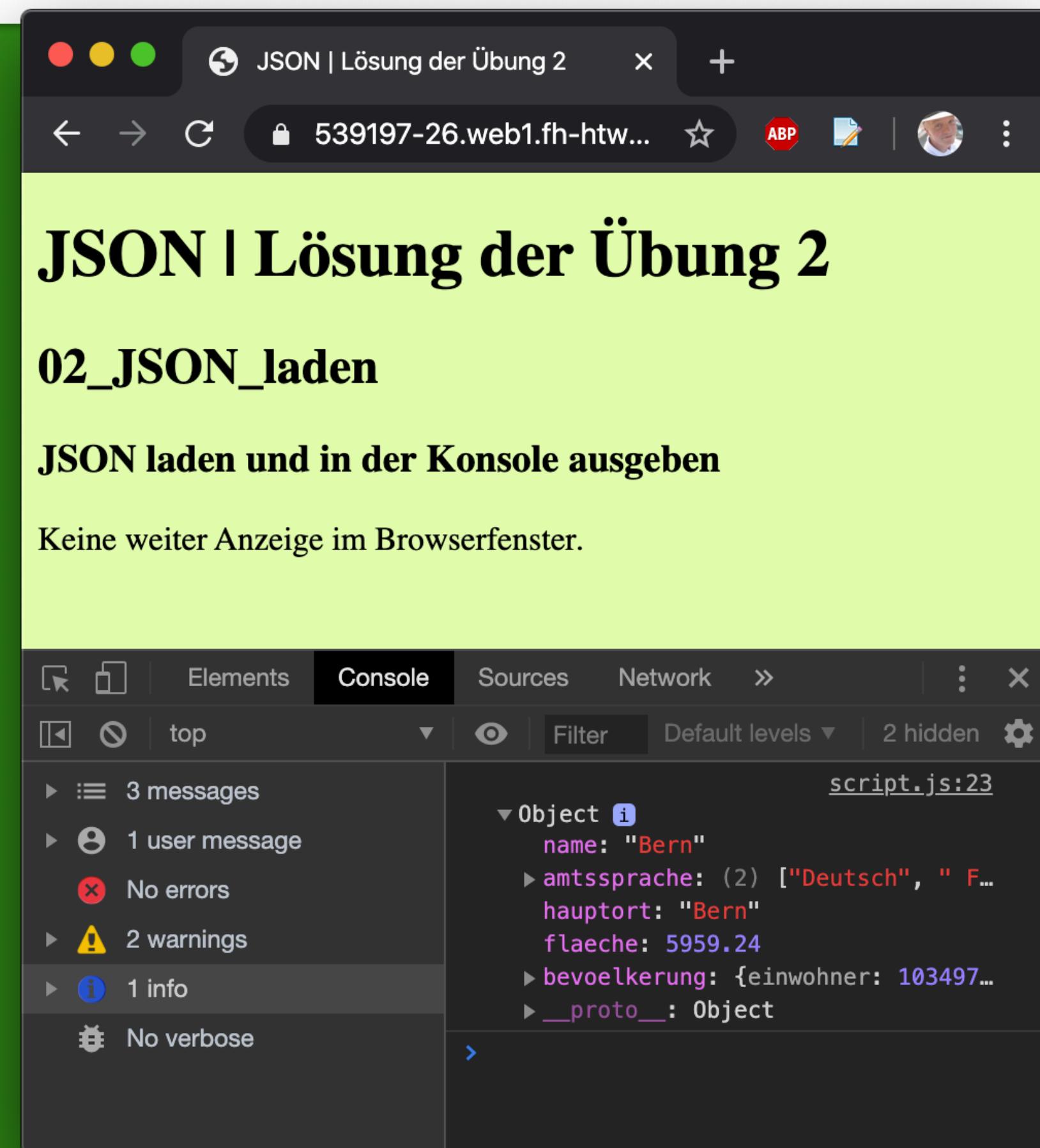
  name : "Bern",
  amtssprache : ["Deutsch", " Französisch"],
  hauptort : "Bern",
  flaeche : 5959.24,
  bevoelkerung : {
    einwohner : 1034977,
    quote : "2,9 %"
  }
}
```

fetch() – JSON response

```
// Mit fetch() -Funktion die externe Datei 'extern/person1.json' laden.  
fetch('extern/person1.json')  
  .then((response) => {  
    // Definiert, welches Format die Antwort hat (wichtig für den nächsten Teil)  
    // !!!!! hier JSON !!!!!!  
    // fetch erwartet jetzt eine validen JSON-Zeichenkette  
    // Die empfangenen JSON-Daten werden von JS direkt in ein Objekt umgewandelt.  
    return response.json();  
})  
  .then((data) => {  
    // Da JavaScript oben mitgeteilt haben, dass der empfangene Text eine JSON-Zeichenkette ist,  
    // können wir die empfangenen Daten direkt als JavaScript-Objekt-Variable speichern.  
    // Ausgabe von data in der Konsole des Browsers.  
    console.log(data);  
})  
  .catch(function(error) {  
    console.log('Error: ' + error.message);  
});
```

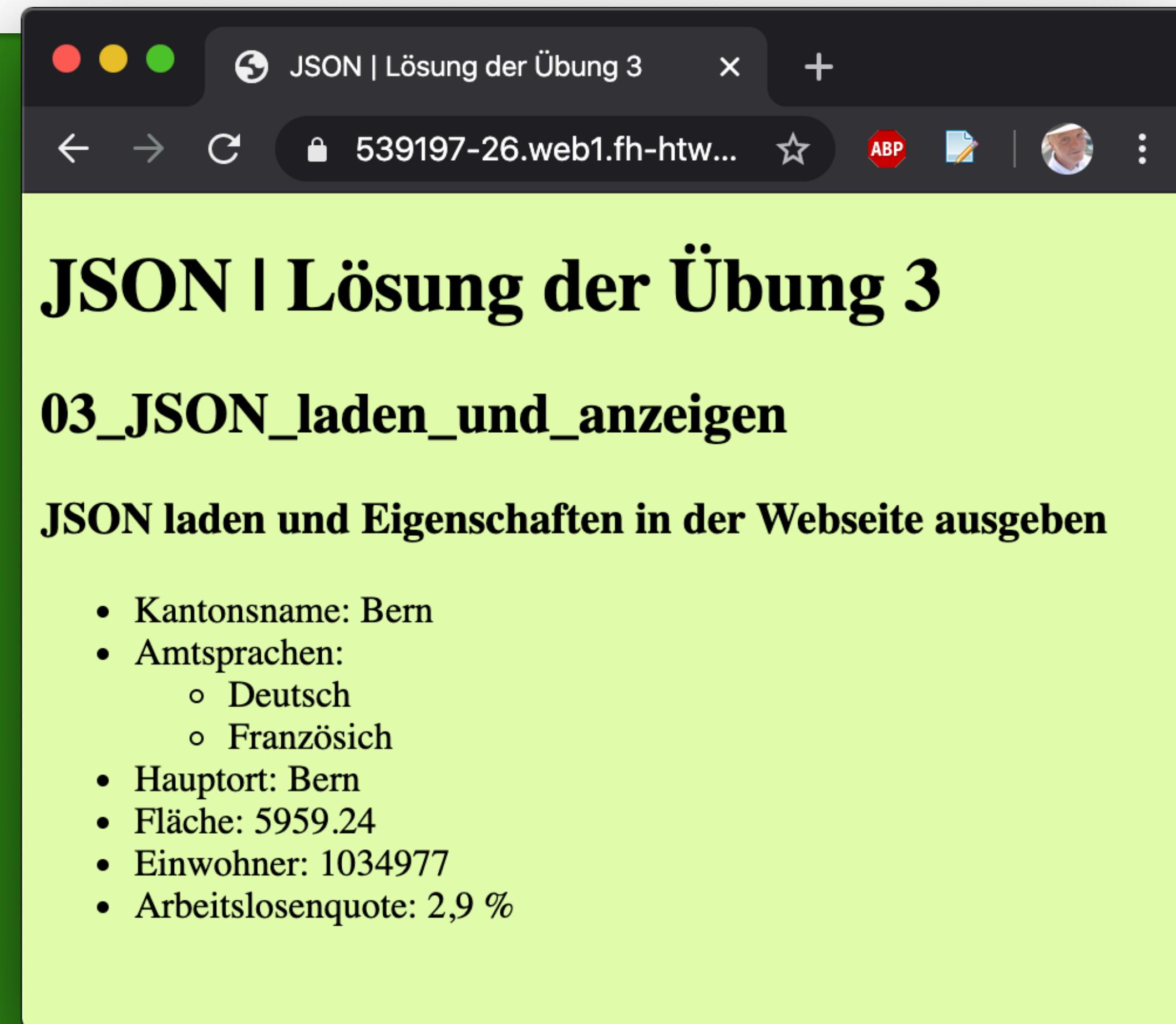
JSON – Übung 2

```
/* JSON | Übung 2 | 02_JSON_laden
/* 1. Laden Sie mit fetch() die Datei 'extern/kanton1.json'.
/* 2. Stellen sie im response ein, welchen Datentyp fetch() zu erwarten hat.
/* 3. Geben Sie mit console.log() die empfangenen Daten in der Konsole Ihres Browsers aus.
/* *****
```



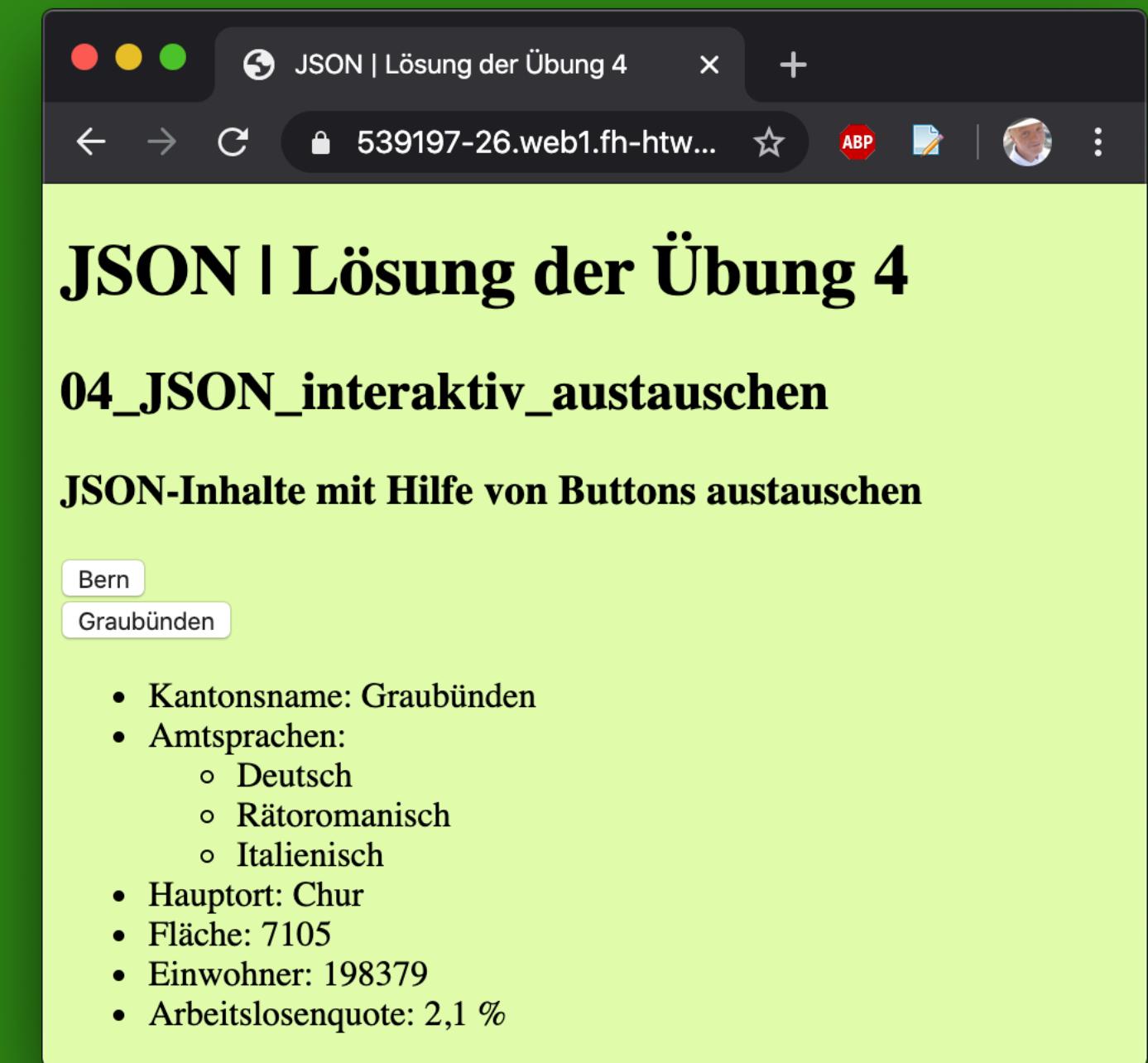
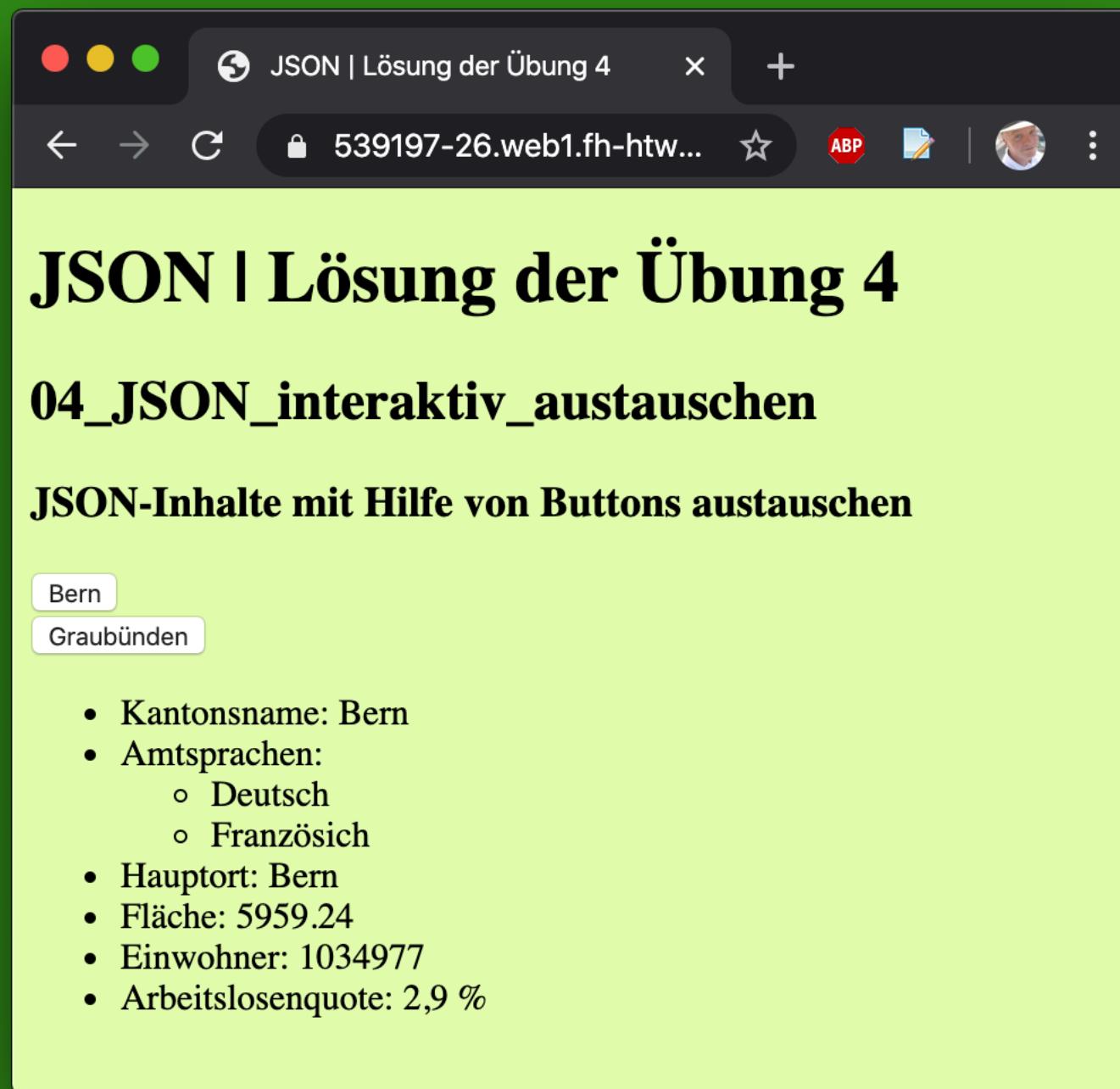
JSON – Übung 3

```
/* JSON | Übung 3 | 03_JSON_laden_und_anzeigen
/* 1. Laden Sie mit fetch() die Datei 'extern/kanton1.json'.
/* 2. Stellen sie im response ein, welchen Datentyp fetch() zu erwarten hat.
/* 3. Geben Sie mit console.log() die empfangenen Daten in der Konsole Ihres Browsers aus.
/* *****
```



JSON – Übung 4

```
/* JSON | Übung 4 | 04_JSON_interaktiv_austauschen
/* 1. Laden Sie mit fetch() die Datei 'extern/kanton1.json'.
/* 2. Stellen sie im response ein, welchen Datentyp fetch() zu erwarten hat.
/* 3. Geben Sie mit console.log() die empfangenen Daten in der Konsole Ihres Browsers aus.
/* *****
```



JSON– Übung 4

```
* JSON | Aufgabe 1 | 01_JSON_Busplan
/* Funktionalisieren Sie die Buttons der zugehörigen Datei so,
/* Dass beim Klick auf einen Button die passende JSON–Datei geladen,
/* der Inhalt aufbereitet und als HTML–Dargestellt wird.
/* Die JSON–Dateien befinden sich im Verzeichnis "extern".
/* *****
```

Feedback

anonymes Feedback