Entwicklung Interaktiver Anwendungen I

#6

MKB 1. Semester Bachelor

Inhalt

Variablen, Datentypen, Operatoren

Praktikum *(Fr 15.11.2019)* ↓

Kleine Mathestunde

6. **Seminar** *(Mi 20.11.2019)* ↓

   Funktionen, DOM Manipulation ↓

   Ereignisgesteuerte Programmierung

Praktikum *(Fr 22.11.2019)* ↓

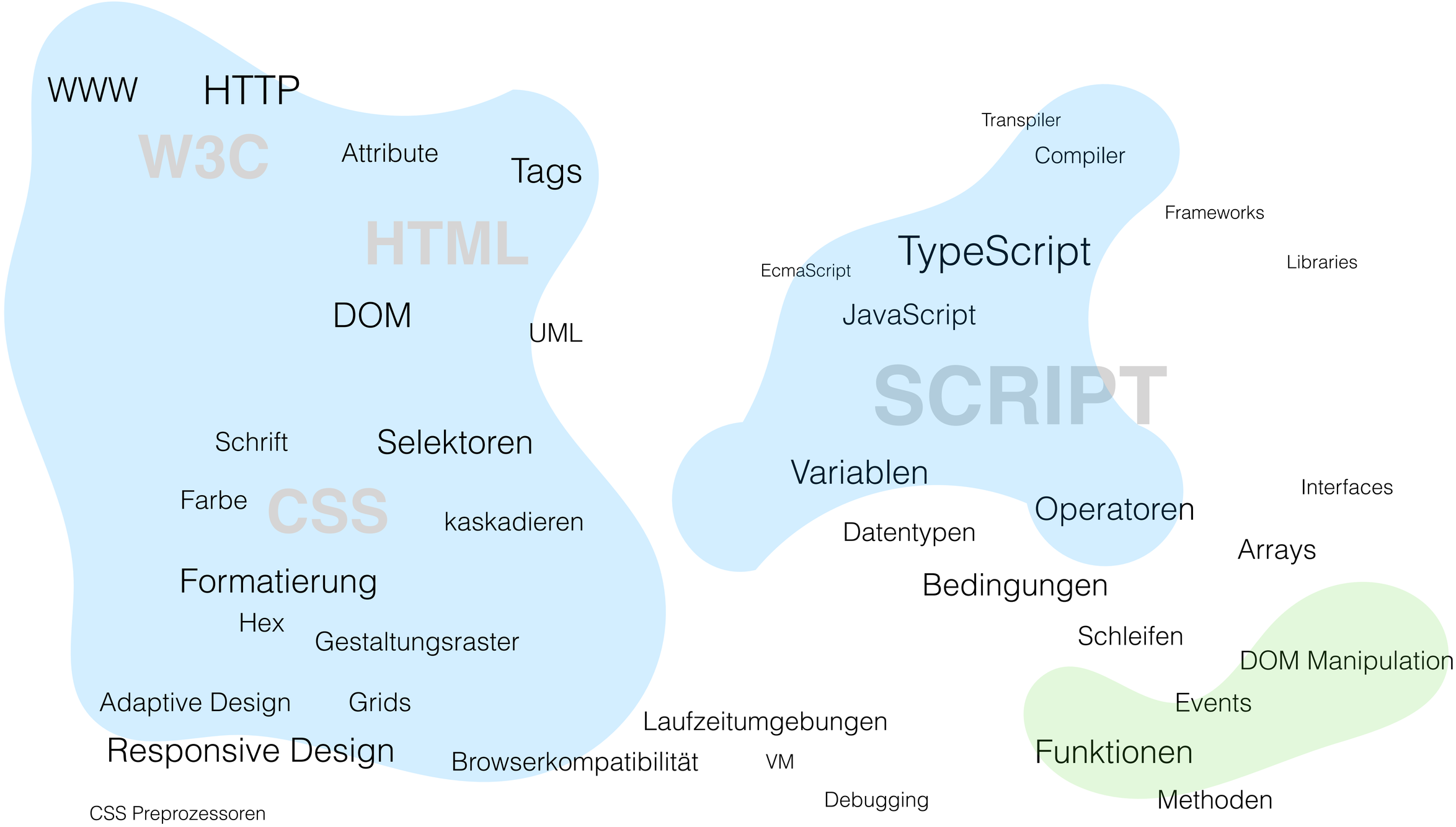
Interaktive Infografik ↓

Intermediate Review mit Jirka

7. **Seminar** *(Mi 27.11.2019)* ↓

   Datentypen im binären ↓

   und hexadezimalen System und Objekte


WWW
HTTP
W3C
Attribute
Tags
HTML
DOM
UML
Schrift
Selektoren
Farbe
CSS
kaskadieren
Formatierung
Hex
Gestaltungsraster
Adaptive Design
Grids
Responsive Design
Browserkompatibilität
CSS Preprozessoren
Transpiler
Compiler
Frameworks
TypeScript
Libraries
EcmaScript
JavaScript
SCRIPT
Variablen
Interfaces
Operatoren
Datentypen
Arrays
Bedingungen
Schleifen
DOM Manipulation
Events
Laufzeitumgebungen
VM
Funktionen
Debugging
Methoden

Parsen des DOMs und des Skripts

.html

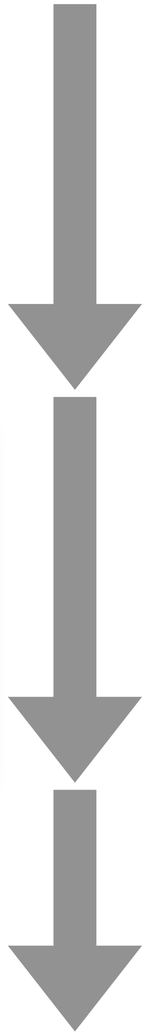
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <script src="script/script.js"></script>
```

.js

```js
var fuelConsumption:number = 7;
var kilometers:number = 30;
var emissionPetrol = fuelConsumption * kilometers * 0.238;

var result = "CO2-Ausstoß für Benzin: " + emissionPetrol;

console.log(result);
```

.html

```html
</head>
<body>
    <h1>Lorem Ipsum</h1>
</body>
</html>
```



Der DOM und externe Ressourcen werden Zeile für Zeile gelesen und interpretiert

Das Skript wird also beim Parsen abgearbeitet. Die Konsolenausgabe erfolgt, sobald diese Zeile in der Skript-Datei gelesen wurde.

Wie kann man verhinden, dass alle Skriptanweisungen automatisch nacheinander abgearbeitet werden?

Funktionen

Funktionen

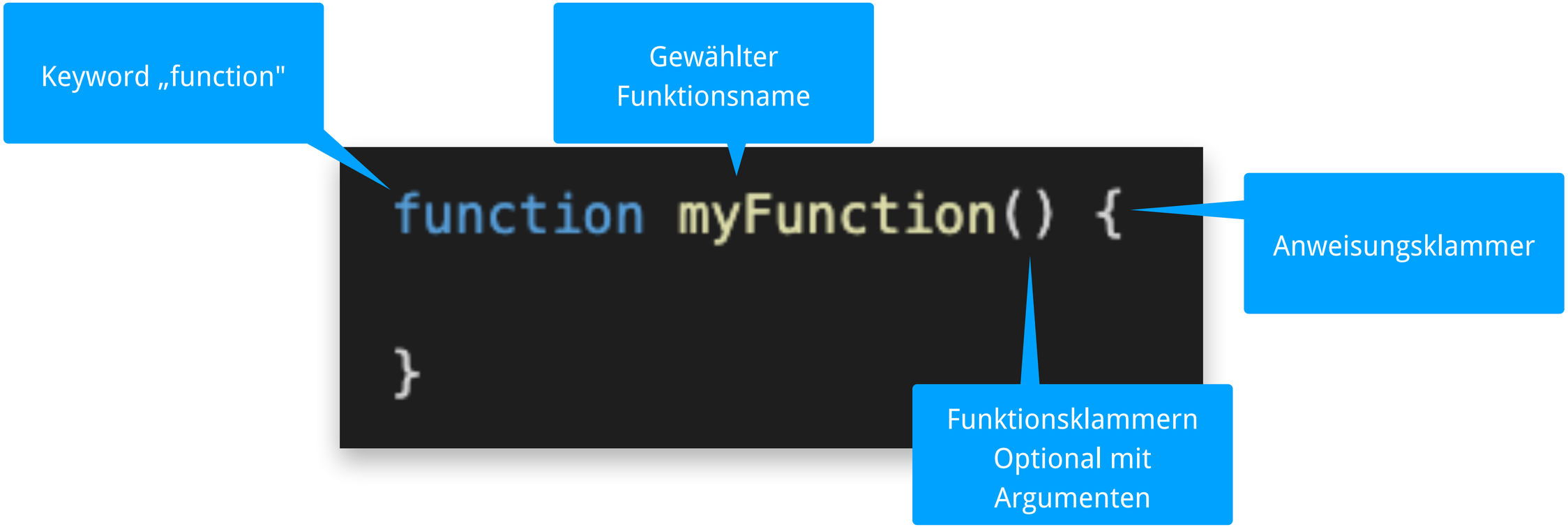
"A function is a […] procedure—a set of statements that performs a task or calculates a value. To use a function, you must define it somewhere in the scope from which you wish to call it.“

— MDN, 2019

Funktionen Deklaration

- Auch „Funktionsdefinition“ oder „Funktionsstatement“ genannt
- Wird mit dem Keyword „function“ eingeleitet
- Empfängt optional eine Liste (kommasepariert) an Argumenten
- Die Anweisungen in einer Funktion werden mit {} umschlossen

Funktionen Deklaration



Funktionen Deklaration

```
function myFunction() {
    var fuelConsumption:number = 7;
    var kilometers:number = 30;
    var emissionPetrol = fuelConsumption * kilometers * 0.238;

    var result = "CO2-Ausstoß für Benzin: " + emissionPetrol;

    console.log(result);
}
```

Funktionen Deklaration und Aufruf

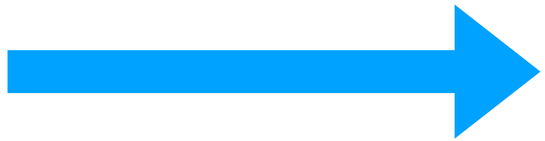
```
function myFunction() {
    var fuelConsumption:number = 7;
    var kilometers:number = 30;
    var emissionPetrol = fuelConsumption * kilometers * 0.238;

    var result = "CO2-Ausstoß für Benzin: " + emissionPetrol;

    console.log(result);
}
```

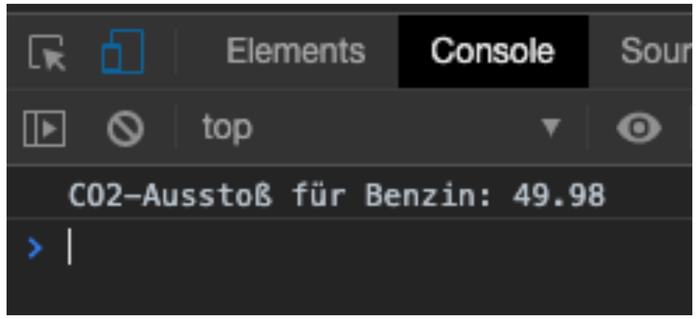
```
myFunction();
```

Funktionsaufruf mit Funktionsklammern, Argumenten und Semikolon

Funktionen Aufruf und Ausgabe

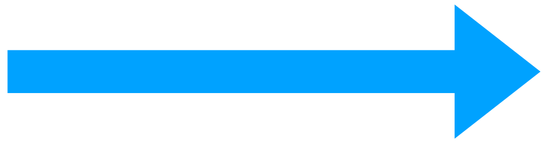
```
myFunction();
```

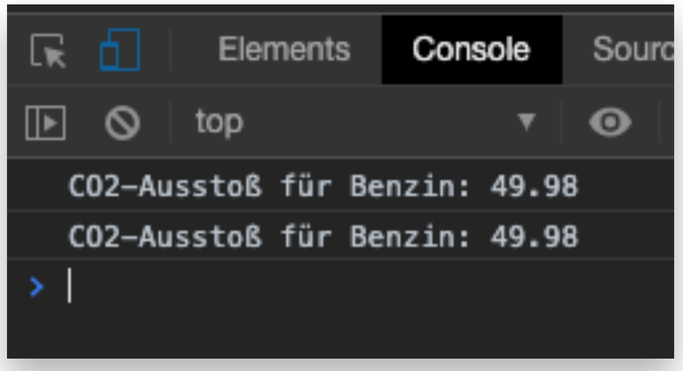




Funktionen Aufruf und Ausgabe

```
myFunction();
myFunction();
```





Argumente

```
function myFunction() {
    var fuelConsumption:number = 7;
    var kilometers:number = 30;
    var emissionPetrol = fuelConsumption * kilometers * 0.238;

    var result = "CO2-Ausstoß für Benzin: " + emissionPetrol;

    console.log(result);
}
```

```
function myFunction(kilometers:number) {
    var fuelConsumption:number = 7;
    var emissionPetrol = fuelConsumption * kilometers * 0.

    var result = "CO2-Ausstoß für Benzin: " + emissionPetr

    console.log(result);
}
```

Argument in Funktionsdeklaration

Argument wird wie eine Variable weiterverwendet

Argumente

```
function myFunction(kilometers:number) {
    var fuelConsumption:number = 7;
    var emissionPetrol = fuelConsumption * kilometers * 0.238;

    var result = "CO2-Ausstoß für Benzin: " + emissionPetrol;

    console.log(result);
}
```

```
myFunction(30);
```

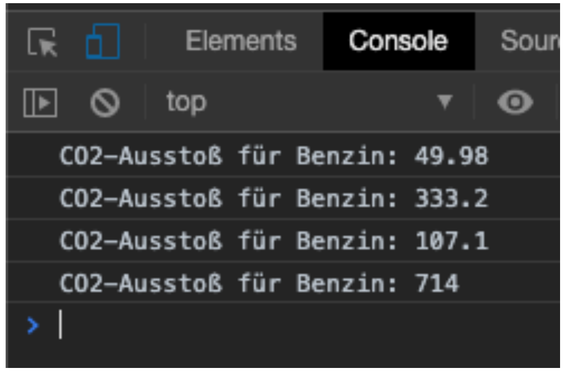
Argument wird mitgegeben beim Funktionsaufruf

Mehrere **Argumente**

```
function myFunction(kilometers:number, fuelConsumption:number) {
    var emissionPetrol = fuelConsumption * kilometers * 0.238;

    var result = "CO2-Ausstoß für Benzin: " + emissionPetrol;

    console.log(result);
}

myFunction(30, 7);
```

Kommasepariert

```
// Auto A
myFunction(30, 7);
myFunction(200, 7);

// Auto B
myFunction(30, 15);
myFunction(200, 15);
```



Funktionen

- Die Ausführung der Anweisungen einer Funktion (bzw. das Ausführen einer Funktion) kann gesteuert werden
- Eine Funktion kann immer wieder verwendet und flexibel mit Argumenten aufgerufen werden

Funktionsbereich

- Variablen, die **außerhalb** einer Funktion **(gobal)** deklariert wurden, sind innerhalb einer Funktion **verfügbar**
- Variablen, die **innerhalb** einer Funktion **(lokal)** deklariert wurden, sind außerhalb der Funktion **nicht verfügbar**

Funktionsbereich

```
var initialText:string = "Die Berechnung ergibt: ";

function myFunction(kilometers:number, fuelConsumption:number) {
    var emissionPetrol = fuelConsumption * kilometers * 0.238;

    var result = initialText + " CO2-Ausstoß für Benzin beträgt " + emissionPetrol;

    console.log(result);
}

console.log(initialText);

myFunction(30, 7);
```

```
Die Berechnung ergibt:
Die Berechnung ergibt:  CO2-Ausstoß für Benzin beträgt 49.98
>
```

Innerhalb der Funktion wird auf eine globale Variable zugegriffen

```
function myFunction(kilometers:number, f
    var initialText:string = "Die Berech
    var emissionPetrol = fuelConsumptior

    var result = initialText + " CO2-Aus

    console.log(result);
}

console.log(initialText);

myFunction(30, 7);
```

Falsch: Eine eine lokale Variable wird außerhalb des Funktionsscopes zugegriffen

Wie kann ein Funktionsblock bei Nutzerinteraktion aufgerufen werden?

Events

Event-Listener Inline

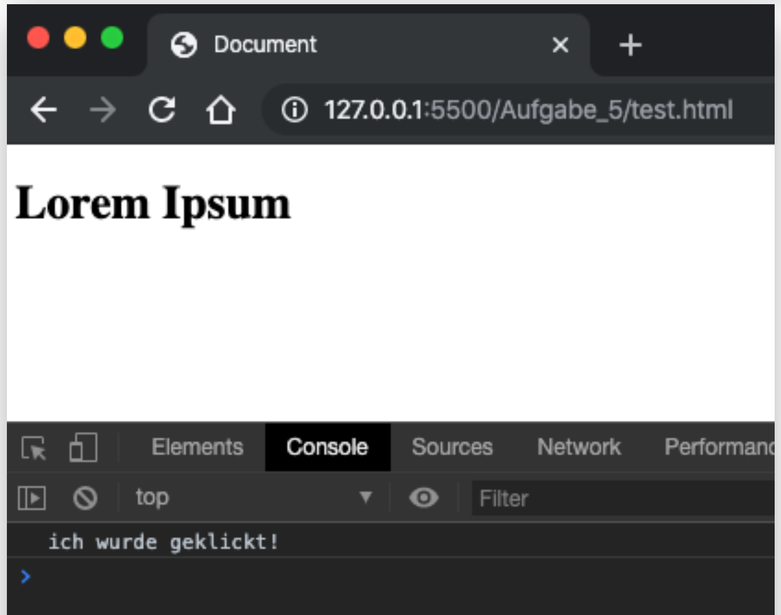
```
<body>
    <h1 onclick="myFunction()">Lorem Ipsum</h1>
</body>
```

- Die Event-Listener direkt als Inline-Attribut an ein HTML-Element ergänzen ist einfach…
- …aber nicht zu empfehlen!!
  Vermischung von Inhalt, Funktion und Gestaltung

Event-Listener Inline

```html
<body>
    <h1 onclick="myFunction()">Lorem Ipsum</h1>
</body>
```

```js
function myFunction() {
    console.log("ich wurde geklickt!");
}
```



Event-Listener

```
function myFunction() {
    console.log("ich wurde geklickt!");
}

document.querySelector("h1").addEventListener('click', myFunction);
```

Event-Listener

```
function myFunction() {
    console.log("ich wurde geklickt!");
}

document.querySelector("h1").addEventListener('click', myFunction);
```

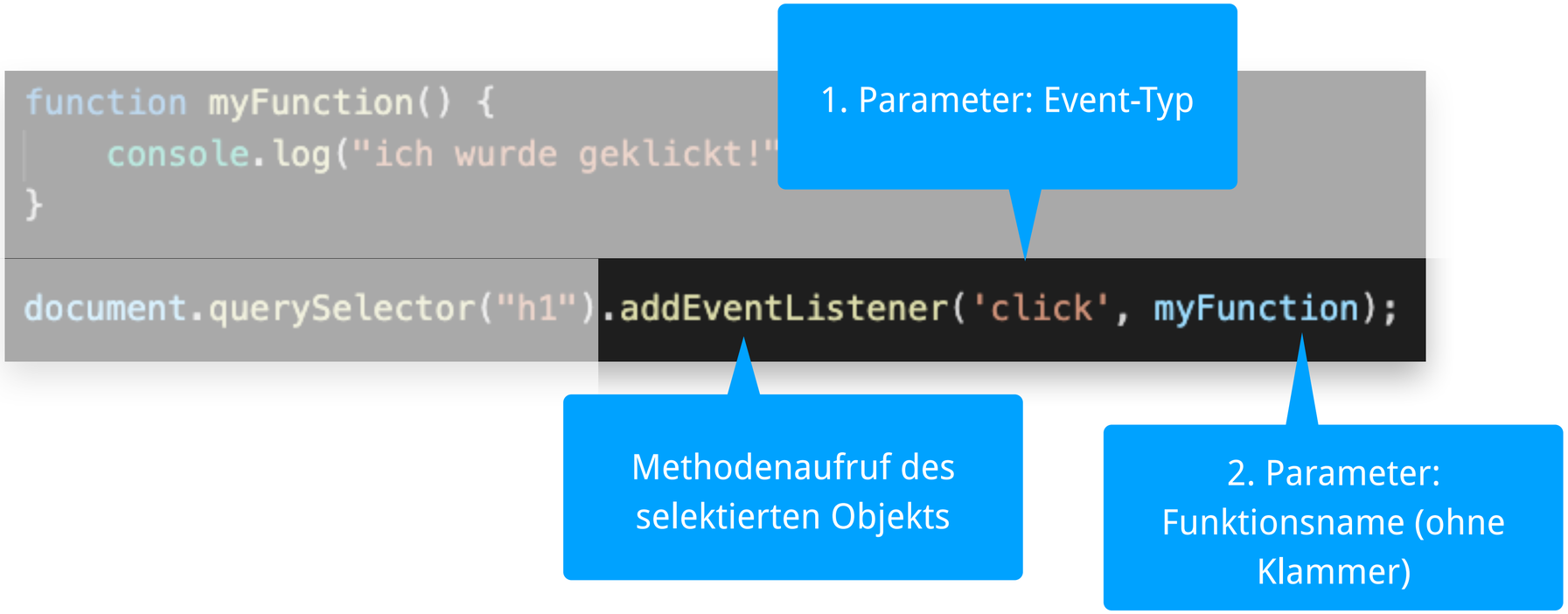
Eine Methode, durch die ein Element im DOM Selektiert werden kann (vgl. CSS Selektion)

Tag, Klassen oder ID Selektor Möglich

```
document.querySelector("h2")
document.querySelector("#meineID")
document.querySelector(".meineKlasse")
```

Das Kleingedruckte: als Einsteiger bitte nur einzelne Objekte auswählen. Hinter h2 oder .meineKlasse sollte für den Einstieg nur EIN Element selektiert werden können. Ansonsten müssen wir mit einer fortgeschritteneren Programmier-Strategie eine Liste an selektierten Elementen an die EventListener übergeben. Das lernen wir später noch.

Event-Listener



Basic Mouse-**Event-Types**

Mouse events

Event Name	Fired When
`auxclick`	A pointing device button (ANY non-primary button) has been pressed and released on an element.
`click`	A pointing device button (ANY button; soon to be primary button only) has been pressed and released on an element.
`contextmenu`	The right button of the mouse is clicked (before the context menu is displayed).
`dblclick`	A pointing device button is clicked twice on an element.
`mousedown`	A pointing device button is pressed on an element.
`mouseenter`	A pointing device is moved onto the element that has the listener attached.
`mouseleave`	A pointing device is moved off the element that has the listener attached.
`mousemove`	A pointing device is moved over an element. (Fired continously as the mouse moves.)
`mouseover`	A pointing device is moved onto the element that has the listener attached or onto one of its children.
`mouseout`	A pointing device is moved off the element that has the listener attached or off one of its children.
`mouseup`	A pointing device button is released over an element.
`pointerlockchange`	The pointer was locked or released.
`pointerlockerror`	It was impossible to lock the pointer for technical reasons or because the permission was denied.
`select`	Some text is being selected.
`wheel`	A wheel button of a pointing device is rotated in any direction.

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

Basic Keyboard-**Event-Types**

Keyboard events

Event Name	Fired When
`keydown`	ANY key is pressed
`keypress`	ANY key except Shift, Fn, CapsLock is in pressed position. (Fired continously.)
`keyup`	ANY key is released

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

Basic Window-**Event-Types**

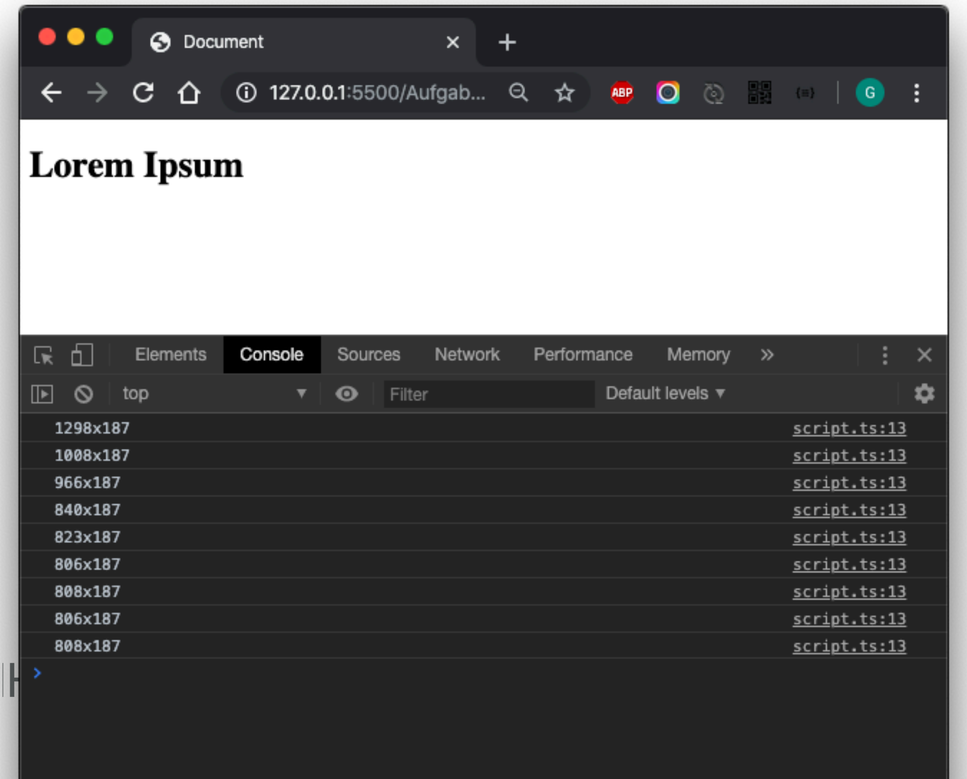
View events

Event Name	Fired When
`fullscreenchange`	An element was turned to fullscreen mode or back to normal mode.
`fullscreenerror`	It was impossible to switch to fullscreen mode for technical reasons or because the permission was denied.
`resize`	The document view has been resized.
`scroll`	The document view or an element has been scrolled.

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

Beispiel Window-**Event-Types**

```
window.addEventListener('resize', function() {
    console.log(window.innerWidth + "x" + window.innerHeight)
});
```



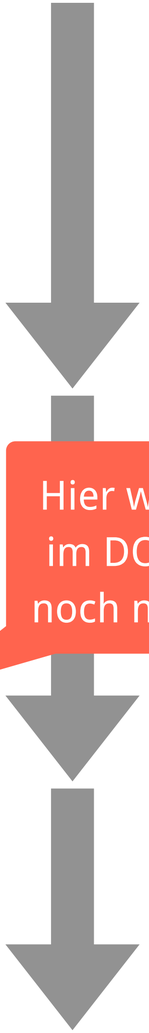
Eine kleine, aber wichtige Anmerkung zu der Ladereihenfolge

Skript-Ladereihenfolge beachten, wenn mit dem DOM genutzt wird!

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <script src="script/script.js"></script>
```

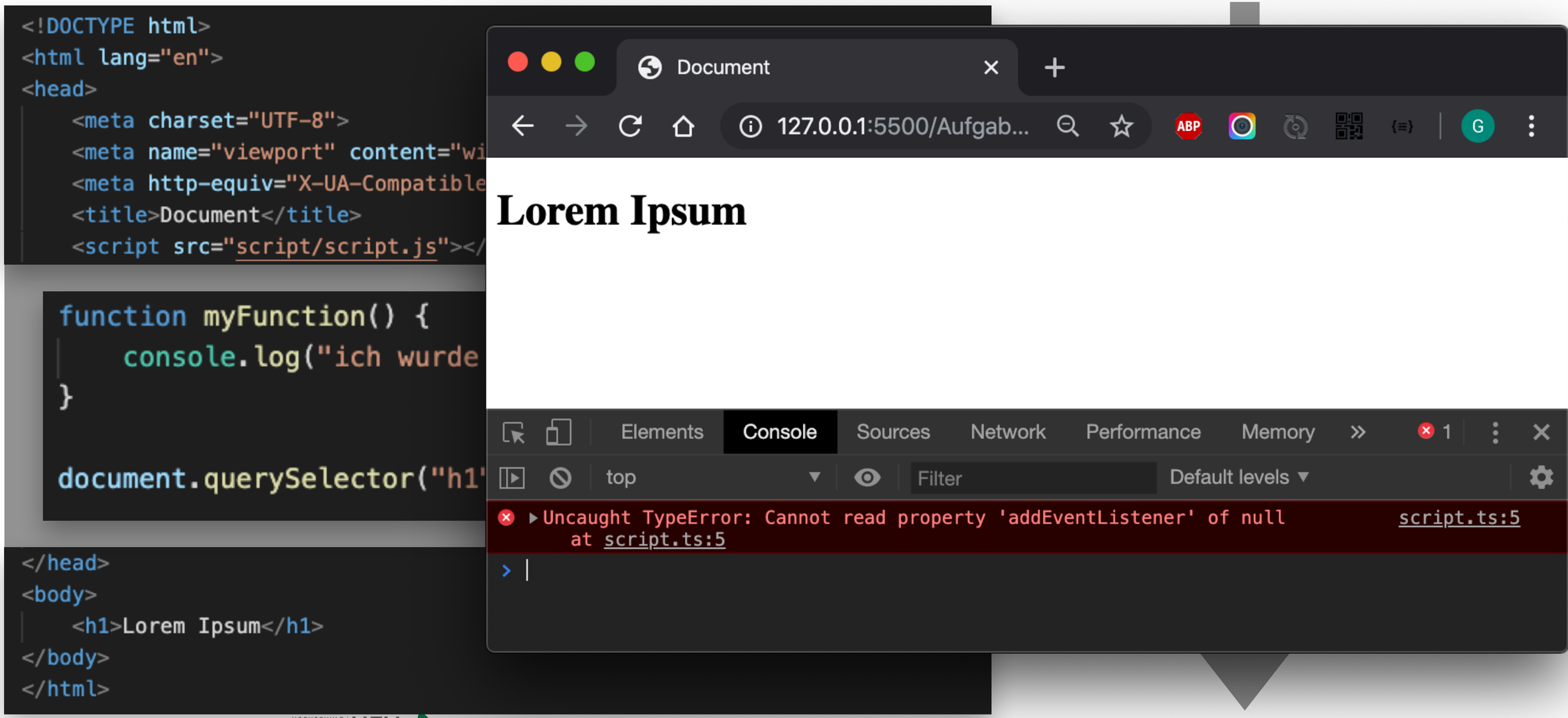
```js
function myFunction() {
    console.log("ich wurde geklickt!");
}

document.querySelector("h1").addEventListener('click', myFunction);
```

```html
</head>
<body>
    <h1>Lorem Ipsum</h1>
</body>
</html>
```



Hier wird auf ein Element im DOM zugegriffen, das noch nicht gelesen wurde.

Skript-Ladereihenfolge beachten, wenn mit dem DOM genutzt wird!



Skript-Ladereihenfolge beachten, wenn der DOM genutzt wird!

1. Lösung (Quick and Dirty)

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <h1>Lorem Ipsum</h1>

    <script src="script/script.js"></script>
</body>
</html>
```

Eine mögliche Lösung: die Script-Einbindung vor dem Schließen des Body-Tags

Skript-Ladereihenfolge beachten, wenn der DOM genutzt wird!

2. Lösung (Quick and Dirty)

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-sc
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <script src="script/script.js" async></script>
</head>
<body>
    <h1>Lorem Ipsum</h1>
</body>
</html>
```

Eine andere quick-and-dirty Lösung: das Script als dynamisch ladendes Script einbinden (jetzt wird der DOM linear gelesen und parallel die externe Script-Datei gelesen).

Skript-Ladereihenfolge beachten, wenn der DOM genutzt wird!

3. Lösung (empfohlen)

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-wi
    <meta http-equiv="X-UA-Compatible" content="ie
    <title>Document</title>
    <script src="script/script.js"></script>
</head>
<body>
    <h1>Lorem Ipsum</h1>
</body>
</html>
```

```js
function myFunction() {
    console.log("ich wurde geklickt!");
}

window.addEventListener('load', function() {
    document.querySelector("h1").addEventListener('click', myFunction);
});
```

Ein Event Listener wird an das Window gehängt. Wenn das Window das Event „load“ (oder onload) triggert, dann wurden alle DOM-Elemente eingelesen und die initialen Anweisungen ausgeführt.

Die Konsolenausgabe ist ja schön und gut... Aber kann ich nicht den DOM manipulieren?

DOM Manipulation

DOM Manipulation

- DOM Manipulation bezeichnet ganz allgemein das verändern der HTML Elemente zur Anwendungslaufzeit
- Umfasst das hinzufügen, löschen, ergänzen von Elementen

DOM Manipulation

Den Inhalt von DOM-Elementen ändern

```
document.querySelector("h1").innerHTML = "Meine neue Überschrift";
```

DOM Manipulation

Den Inhalt von DOM-Elementen ändern

```
document.querySelector("h1").innerHTML = "Meine neue Überschrift";
```

Selektion des entsprechenden DOM-Elements
Hier per Tag-Selektor.
Ansonsten auch Klassen- oder ID-Selektor (siehe Ausführung auf Event-Folie)

DOM Manipulation

Den Inhalt von DOM-Elementen ändern

```
document.querySelector("h1").innerHTML = "Meine neue Überschrift";
```

Methode „innerHTML“ mit Wertzuweisung

DOM Manipulation

Den Inhalt von DOM-Elementen ändern

```
document.querySelector("h1").innerHTML = "Meine neue <i>Überschrift</i>";
```

Zeichenketten-Wert kann auch HTML-Elemente beinhalten.

DOM Manipulation

Attribute eines DOM-Element ändern

```
document.querySelector('h1').setAttribute('attributeName', 'value');
```

Methode, um ein Attribut zu setzen

1. Parameter: Name des Attributs

2. Parameter: Wert des Attributs

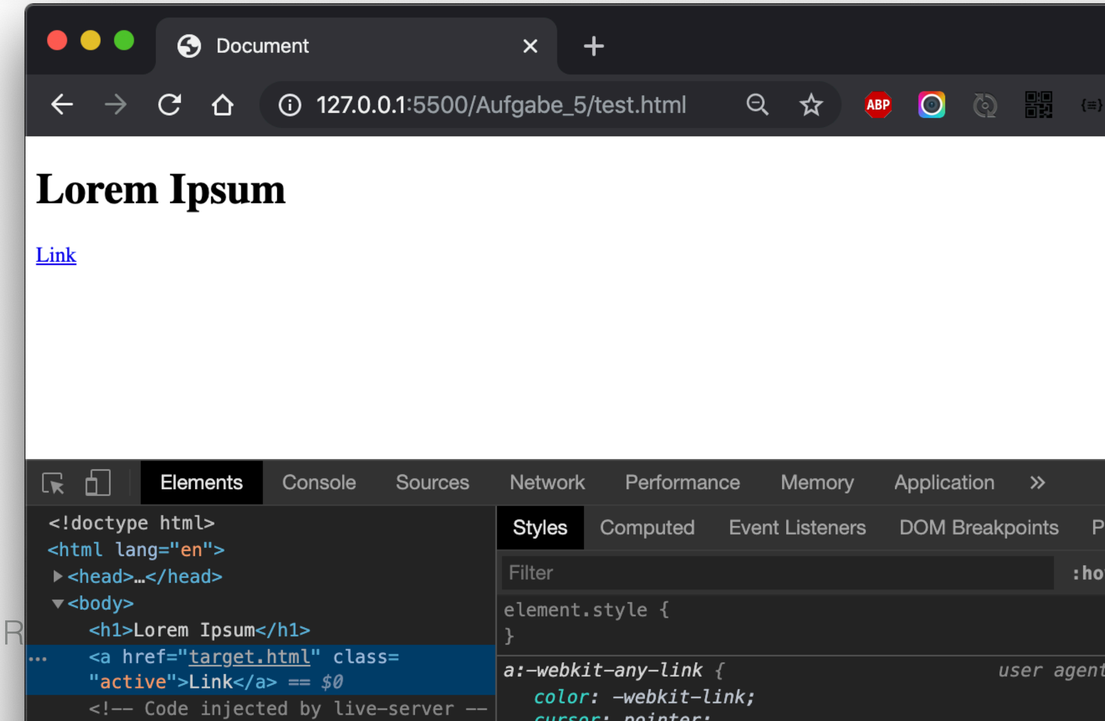
DOM Manipulation

Attribute eines DOM-Element ändern

Beispiel Klassen-Attribut

```
document.querySelector('a').setAttribute('class', 'active');
```

```
<a href="target.html">Link</a>
```



DOM Manipulation

Attribute eines DOM-Element ändern

Beispiel Style-Attribut

```
document.querySelector('.chart').setAttribute('style', 'height:100px');
```

Inline-Style Attribut ist zur Manipulation mit TypeScript eine valide Art, um CSS zu generieren.

Das CSS Fragment als Zeichenkette

DOM Manipulation

Attribute eines DOM-Element ändern

Beispiel Style-Attribut

```
var num: number= Math.random() * 100;

document.querySelector('.chart').setAttribute('style', 'height:' + num + 'px');
```

Das Attribut kann natürlich auch aus Variablen generiert werden.

DOM Manipulation

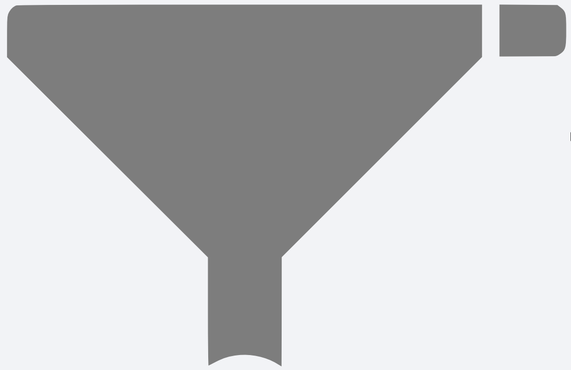
Neue Elemente erstellen

```
var newElement = document.createElement('h2');
document.body.appendChild(newElement);
newElement.innerHTML = "Ein neues Element";
```

Erstellen eines neuen Elements

Element zum Body hinzufügen

Element-Inhalt setzen



TAKE AWAYS

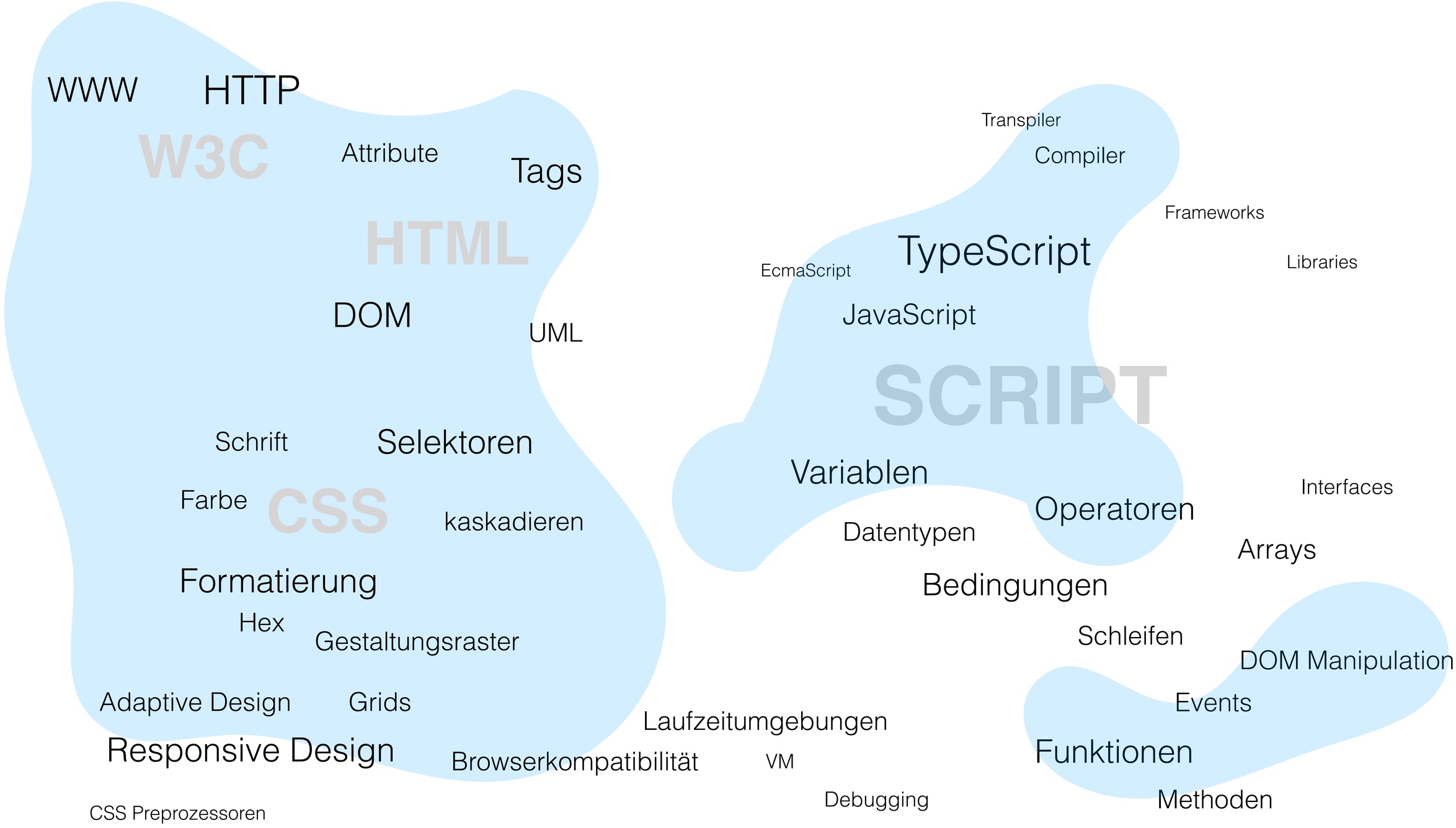
Funktionen sind **Anweisungsblöcke**, die gesteuert **aufgerufen** werden können.

Events sind Ereignisse, die durch **unterschiedlichste Initiatoren** ausgelöst werden können.
Bspw. eine **Nutzereingabe mit der Maus** kann ein Event triggern.

Ein **getriggertes Event** kann eine **Funktion auslösen.**

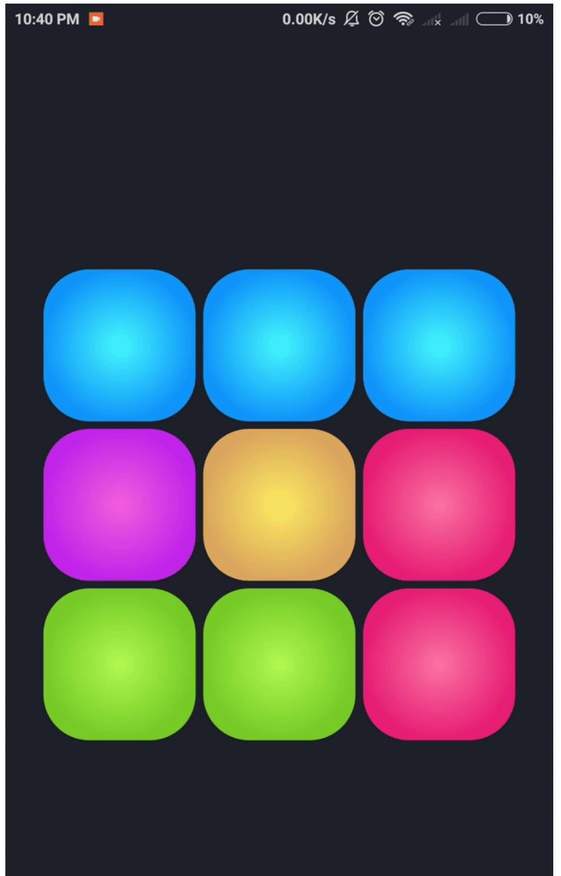
Der **DOM** mit all seinen HTML-Element kann durch verschiedene Methoden **manipuliert** werden.

Eine einfache Herangehensweise stellt die Methode **innerHTML** dar.


WWW
HTTP
W3C
Attribute
Tags
HTML
DOM
UML
Schrift
Selektoren
Farbe
CSS
kaskadieren
Formatierung
Hex
Gestaltungsraster
Adaptive Design
Grids
Responsive Design
Browserkompatibilität
CSS Preprozessoren
Transpiler
Compiler
Frameworks
TypeScript
EcmaScript
Libraries
JavaScript
SCRIPT
Variablen
Interfaces
Operatoren
Datentypen
Arrays
Bedingungen
Schleifen
DOM Manipulation
Events
Laufzeitumgebungen
VM
Funktionen
Debugging
Methoden

Sneak Preview

Mit diesen **Programmiergrundlagen** lassen sich spannende **interaktive Medien entwickeln**



Nächster Termin

Variablen, Datentypen, Operatoren

Praktikum *(Fr 15.11.2019)* ↓

Kleine Mathestunde

6. **Seminar** *(Mi 20.11.2019)* ↓

   Funktionen, DOM Manipulation ↓

   Ereignisgesteuerte Programmierung

   Praktikum *(Fr 22.11.2019)* ↓

   Interaktive Infografik ↓

   Intermediate Review mit Jirka

7. **Seminar** *(Mi 27.11.2019)* ↓

   Datentypen im binären ↓

   und hexadezimalen System und Objekte