

JAVASCRIPT CALLBACKS / PROMISES

CALLBACK FUNCTIONS



- Deutsch Rückruffunktion
- Funktionen die im Parameter einer Funktion aufgerufen werden
- Diese Funktionen werden erst nach dem Ereignis der Funktion aufgerufen und wird deswegen Callback-Function genannt

```
document.getElementById("close_button").addEventListener("click", function() {
        document.getElementById("menu").classList.remove("active");
});

document.getElementById("close_button").addEventListener("click", closeMenu());

function closeMenu() {
        document.getElementById("menu").classList.remove("active");
}
```

FUNKTIONSAUFRUF IN ES6



- Statt function()
- () =>

```
document.getElementById("close_button").addEventListener("click", function() {
        document.getElementById("menu").classList.remove("active");
});

document.getElementById("close_button").addEventListener("click", () => {
        document.getElementById("menu").classList.remove("active");
});
```

CALLBACK FUNCTION



```
setTimeout (callback, delay);
setTimeout (function () {
  console.log ("Aufruf nach 5 Sekunden!");
}, 5000);
```

ASYNCHRONE EVENTS



- Wenn Daten geladen werden dauert das oft eine Weile
 - Bilder oder JSON-Dateien
- Erst nach dem Warten sollen die nächsten Aktionen gestartet werden
- Kann über
 - Callback-Functions
 - Promises

gelöst werden

EINFACHE CALLBACKS - BSP



```
const url = "file.json";

const xhr = new XMLHttpRequest(url);
xhr.addEventListener ("readystatechange", (url) => {
    if (xhr.readyState === 4) {
        console.log (xhr.responseText);
    }
})
xhr.open ("GET", url);
xhr.send();
```

https://www.mediaevent.de/javascript/promise.html

VERSCHACHTELTE CALLBACKS



- Was ist aber wenn die n\u00e4chste Datei erst geladen werden sollte nachdem die erste Datei geladen wurde
 - Man muss Callbacks verschachteln
- Callbacks Hell
- Verschachtelte asynchrone Events werden sehr unübersichtlich
 - Lösung: Promises

VERSCHACHTELTE CALLBACKS const url = "../fetch.json";



```
const url2 = "../file.json";
const url3 = "../data.json";
const xhr = new XMLHttpRequest(url);
xhr.addEventListener("readystatechange", (url) => {
   if (xhr.readyState === 4) {
      console.log (xhr.responseText);
      const xhr2 = new XMLHttpRequest(url2);
      xhr2.addEventListener ("readystatechange", (url2) => {
         if (xhr2.readyState === 4) {
             console.log ("xhr2 " + xhr2.responseText);
             const xhr3 = new XMLHttpRequest(url3);
             xhr3.addEventListener ("readystatechange", (url3) => {
                //console.log ("xhr3 " + xhr3.responseText);
             });
             xhr3.open ("GET", url3);
             xhr3.send();
      });
     xhr2.open ("GET", url2);
      xhr2.send();
});
xhr.open ("GET", url);
xhr.send();
```

https://www.mediaevent.de/java script/promise.html

PROMISES



- Deutsch Versprechen
- Ein Objekt das entweder einen Wert oder ein Fehler-Objekt zurückbekommt
 - Je nach dem ob der Code funktioniert
- Sicherer (bei asynchronem Verhalten) und lesbarer als Callback-Funktionen
 - Besser lesbare Struktur
 - Verlässlicher, dass der Ablauf abgewartet wird
 - Wird sicher nur einmal aufgerufen

https://www.w3schools.com/js/js _promise.asp

PROMISES FUNKTIONEN



- .then()
 - Behandlung nach Erfolg
- .catch()
 - Behandlung nach Error
- .resolve()
 - Returniert ein Promise-Objekt
- .reject()
 - Returniert ein Error-Objekt

PROMISES THEN



- Promises stellen die Methode .then() zur Verfügung, die ausgeführt wird nachdem das Versprechen eingelöst wurde
- 2 optionale Parameter
 - Success-Callback
 - Error-Callback (auch mit .catch() abzubilden)
- .then() Aufrufe können aneinander gehängt werden um zusätzliche asynchrone Aktionen nacheinander auszuführen
 - Chaining verketten

PROMISE + THEN



```
const url = "../fetch.json";
const url2 = "../file.json";
const url3 = "../data.json";
fetch(url).then(function(response){
   return response.json();
}).then(function (data) {
   console.log ("url 1 " + data)
}).then(fetch(url2).then(function(response){
  return response.json();
}) .then(function (data) {
   console.log ("url 1 " + data)
}).then(fetch(url3).then(function(response){
   return response.json();
}) .then(function (data) {
   console.log ("url 1 " + data)
}).catch(function(error));
```

https://www.mediaevent.de/java script/promise.html

AWAIT / ASYNC



- Die asynch-Funktion definiert eine asynchrone Funktion
- Geben ein Promise-Objekt zurück
- Await stoppt den weiteren Code
- Wartet auf ein Promise-Objekt
- Macht nur in einer asynchronen Funktion Sinn

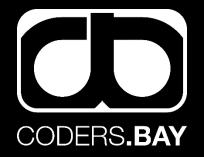
```
function resolveAfter2Seconds(x) {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve(x);
        }, 2000); });
    }

async function f1() {
    var x = await resolveAfter2Seconds(10);
    console.log(x); // 10
}

f1();
```

Beispiel:

https://developer.mozilla.org/de/docs/Web/JavaScript/Ref erence/Statements/async function



ENDE