Asynchrone Programmierung

Eines der grundlegenden Prinzipien bei der Entwicklung in JavaScript ist die asynchrone Programmierung. Sei es bei der Formulierung von Ajax-Anfragen oder beim Lesen von Dateien unter Node.js.

Wichtig ist hierbei Klarheit bezüglich der Begriffe synchrone und asynchrone Kommunikation zu haben.

Definition **asynchrone** Kommunikation:

Unter asynchroner Kommunikation versteht man einen Modus der Kommunikation, bei dem das Senden und Empfangen von Daten zeitlich versetzt und ohne Blockieren des Prozesses durch bspw. Warten auf die Antwort des Empfängers stattfindet.

Definition **synchrone** Kommunikation:

Unter synchroner Kommunikation versteht man in der Informatik und Netzwerktechnik einen Modus der Kommunikation, bei dem die Kommunikationspartner (Prozesse) beim Senden oder beim Empfangen von Daten immer synchronisieren, also warten (blockieren), bis die Kommunikation abgeschlossen ist.[1]

Sync/Await

In Version ES2016 wurde mit den sogenannten *Async Funktionen* eine Möglichkeit eingeführt, die das Formulieren von asynchronem Code vereinfacht. Über das Schlüsselwort *async* können dabei asynchrone Funktionen als solche gekennzeichnet werden, wobei das Schlüsselwort vor die Deklaration der entsprechenden Funktion geschrieben wird.

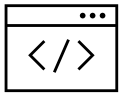
Arbeitsauftrag

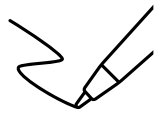
Betrachten Sie nachfolgendes Video (ab Minute 22:46) und erstellen Sie sich im nachfolgenden Code handschriftliche Ergänzungen:

Erklärvideo https://t1p.de/30dfj	

[1] Vgl. https://de.wikipedia.org/wiki/Synchrone_Kommunikation

Code-Beispiel





```
async function getTodos() {
  try{
    const response = await fetch("https://jsonplaceholder.typicode.com/todos/1")
    const data = await response.json()
    console.log(data)
  }
  catch (err) {
    console.log("Something went wrong...")
    console.log(err)
  }
}

getTodos()
```

Beispiel für den Ablauf zweier asynchroner Aufrufe

```
// First async Function
async function loadFirst() {
      const response = await
      fetch('https://jsonplaceholder.typicode.com/posts/1')
      .then((response) => response.json());

        // simulation of Delay
    setTimeout(() => {
        console.log(response)
    }, 500);
}

// Second async Function
async function loadSecond() {
    const response = await
fetch('https://jsonplaceholder.typicode.com/posts/2')
    .then((response) => response.json());
    console.log(response);

}

loadFirst();
loadSecond();
```

Die setTimeout-Methode im Beispiel dient der Simulation einer Verzögerung, um die Asynchronität besser zu verdeutlichen.

Arbeitsauftrag

Welche Funktion wird als erstes ausgeben und warum?

da es asynchron ist, wird loadSecond zuerst ausgegeben,
da loadFirst einen zu großen delay hat, um vor loadSecond fertig zu sein

Quelle: Ackermann, Philip: JavaScript – Das umfassende Handbuch, 3. Auflage, 2021 Rheinwerk, S. 849ff.