Lightning Web Component How to wait for long running operation in JS LWC

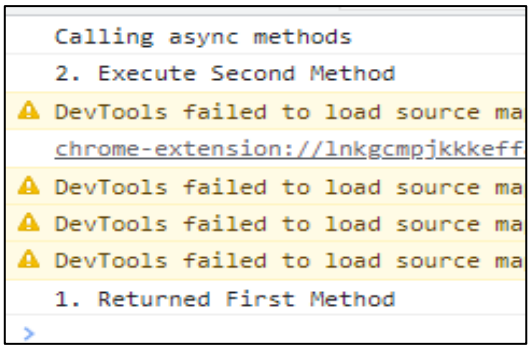
The Context

In context, JavaScript is synchronous language I.e. only single operation is performed at a time. But certain actions can be performed asynchronously such as fetching data from server using Apex (wire functions) and such actions could take some time. During this time JS will not wait for the operation to finish rather it will start executing the next operation. In general if you need the output of any long running operation then ideal solution is to **wait** for such operation to finish and this can be achieved using **async await.**

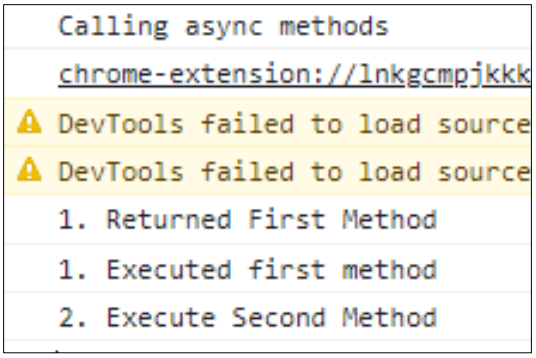
Use Case

In the example below, Sequence of execution of two functions is not controlled. First function is taking time to execute, as a result causing second function (dependent on first) to execute prior to first. Compare below outputs.

Synchronous output (without async await):



Expected output:



Approach (How to serialize your multiple synchronous/async-sync calls)

1. Bind long running operation to return it as a **Promise**.

firstmethod takes one second to finish the execution and we wait till we receive response.

```
async firstmethod(){
  return new Promise( resolve=>{
      //Long running operation such as fetching data from server
      setTimeout(() => {
        resolve("1. Executed first method");
        console.log('1. Returned First Method');
      }, 1000)
    }
  );
}
```

2. Promise will return **fulfill** state if operation is complete. Execute further statements after making a call to this async method.

```
console.log('Calling async methods');
const firstpromise = await this.firstmethod();
console.log(firstpromise);
this.secondMethod();
```

3. While making a call to asynchronous function (in this case **firstmethod**), calling method needs to be denoted by async and calling statement needs to be denoted by await.

4. Imperative and wired calls to Apex are asynchronous in nature. Usually we use `**.then** ` to access the result returned by calls. Implicit behaviour of these calls is similar to **Promises.**

5. To serialize such multiple asynchronous calls it is important to chain these events

6. Practical use case -
    a) There are 4 tabs , each tab rendering a child component displaying specific data.
    b) There is one search box where use can search data among these four tabs.
    c) Data to each tab is loaded when that tab is selected (active).
    d) Search functionality is specific to child component and when each tab is active (clicked) search result specific to each tab is displayed.
    e) When tab is active(clicked) two events must occur synchronously
        i. Initialized data in active tab
        ii. And call search functionality to perform the search logic on initialized data
    f) To do this we return a Promise for step 1 and using **.then** we call the search function.

Complete JS Code

```
import { LightningElement } from 'lwc';

export default class AsyncawaitExample extends LightningElement {

    connectedCallback(){
        this.init();
    }

    async init(){
        console.log('Calling async methods');
        const firstpromise = await this.firstmethod();
        console.log(firstpromise);
        this.secondMethod();
    }

    async  firstmethod(){
        return new Promise( resolve=>{
                //Long running operation such as fetching data from server
                setTimeout(() => {
                    resolve("1. Executed first method");
                    console.log('1. Returned First Method');
                }, 1000)
            }
        );
    }

    secondMethod(){
        console.log('2. Execute Second Method');
    }
}
```