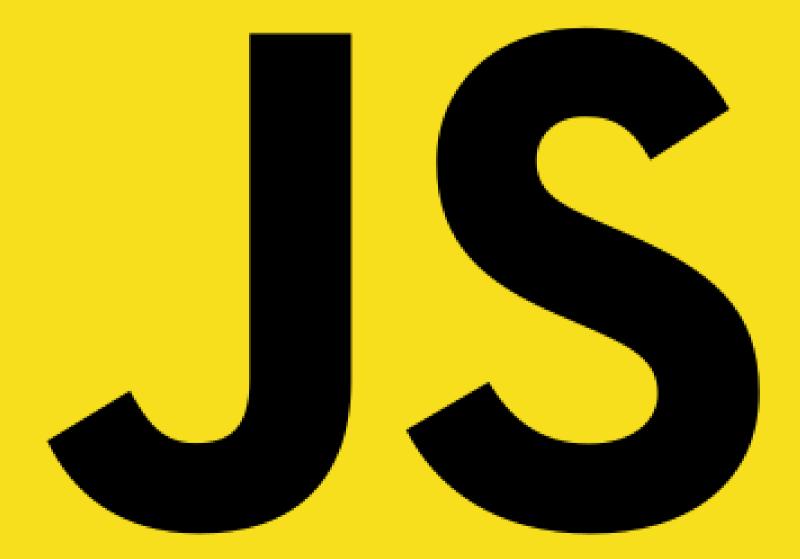
## Asynchronous Programming in



## Callbacks

Callbacks are functions passed as arguments to another function and executed once the operation is complete.

••	JS main.js
1 2	<pre>function fetchData(callback) {    setTimeout(() =&gt; {</pre>
3 4 5	<pre>const data = 'This is some data fetched asynchronously with a callback'; callback(data);</pre>
6 7 8	<pre>}, 2000); }</pre>
9 10 11 12	<pre>// Using the callback function fetchData(function(data) {     console.log(data); });</pre>

## Promises

Promises represent a value that may be available now, in the future, or never. They provide a cleaner alternative to callback-based approaches.

```
- - -
          JS main.js
    function fetchData() {
      return new Promise((resolve, reject) => {
        setTimeout(() => {
          const data = 'This is some data fetched
    asynchronously with a Promise';
         resolve(data);
        }, 2000);
     });
    }
    fetchData()
      .then(data => {
        console.log(data);
      })
      .catch(error => {
      console.error(error);
      });
```

## Async/Await

Async/await provides syntactic sugar on top of Promises, making asynchronous code more readable and easier to understand.

```
- -
          JS main.js
    async function fetchData() {
      return new Promise(resolve => {
        setTimeout(() => {
          const data = 'This is some data fetched
    asynchronously with async/await';
          resolve(data);
        }, 2000);
      });
    }
    async function getData() {
    try {
        const data = await fetchData();
      console.log(data);
      } catch (error) {
        console.error(error);
      }
    getData();
```