JavaScript: Promises - Mitschrift

Was ist eine Promise?

Eine Promise ist ein Objekt, das einen zukünftigen Wert repräsentiert. Es hilft, asynchrone Operationen zu handhaben, ohne in verschachtelte Callbacks zu geraten. Eine Promise kann drei Zustände haben: Pending (wartend), Fulfilled (erfüllt), Rejected (abgelehnt).

Erstellen einer Promise

Eine Promise wird mit dem `Promise`-Konstruktor erstellt. Sie nimmt eine Funktion mit `resolve` und `reject` entgegen.

Code:

```
const meinePromise = new Promise((resolve, reject) => {
let erfolgreich = true;
if (erfolgreich) {
resolve('Erfolg!');
} else {
reject('Fehlgeschlagen!');
}
});
```

Promise verwenden

Mit `then()` reagieren wir auf Erfolg, mit `catch()` auf Fehler:

Code:

```
meinePromise
.then((ergebnis) => {
console.log(ergebnis); // 'Erfolg!'
})
.catch((fehler) => {
console.error(fehler); // 'Fehlgeschlagen!'
});
```

Daten mit `fetch()` abrufen

Ein häufiger Anwendungsfall ist das Laden von Daten aus einer API.

Code:

```
fetch('https://api.example.com/data')
.then(response => {
if (!response.ok) throw new Error('Netzwerkfehler');
return response.json();
})
.then(data => console.log(data))
.catch(error => console.error('Fehler:', error));
```

Promise-Chaining

Mehrere asynchrone Aufgaben können durch Verkettung (`then()`) organisiert werden.

Code:

```
doSomething()
.then(result => doSomethingElse(result))
.then(newResult => doThirdThing(newResult))
.then(finalResult => console.log('Fertig:', finalResult))
.catch(error => console.error('Fehler:', error));
```

Fazit

Promises machen asynchronen Code besser lesbar und vermeidbare Fehler durch verschachtelte Callbacks. Ein gutes Verständnis von Promises ist essenziell für moderne JavaScript-Entwicklung.