JavaScript: Callback Hell - Mitschrift

Was ist die Callback-Hölle?

Die Callback-Hölle tritt auf, wenn mehrere asynchrone Funktionen ineinander geschachtelt werden, was den Code unübersichtlich und schwer wartbar macht. Dies wird auch als 'Pyramide des Untergangs' bezeichnet.

Beispiel für die Callback-Hölle

Hier ein Beispiel für tief verschachtelte Callbacks:

Code:

```
function ersteOperation(callback) {
setTimeout(() => {
console.log('Erste Operation abgeschlossen.');
callback();
}, 1000);
}
function zweiteOperation(callback) {
setTimeout(() => {
console.log('Zweite Operation abgeschlossen.');
callback();
}, 1000);
}
function dritteOperation(callback) {
setTimeout(() => {
console.log('Dritte Operation abgeschlossen.');
callback();
}, 1000);
}
// Verschachtelte Aufrufe
ersteOperation(() => {
zweiteOperation(() => {
dritteOperation(() => {
console.log('Alle Operationen abgeschlossen.');
});
});
});
```

Lösung mit Promises

Durch die Verwendung von Promises wird der Code klarer und besser lesbar:

Code:

```
function ersteOperation() {
return new Promise((resolve) => {
setTimeout(() => {
console.log('Erste Operation abgeschlossen.');
resolve();
}, 1000);
});
}
```

Bessere Lösung mit Async/Await

Async/Await macht den Code noch übersichtlicher und einfacher verständlich.

Code:

```
async function alleOperationen() {
try {
await ersteOperation();
await zweiteOperation();
await dritteOperation();
console.log('Alle Operationen abgeschlossen.');
} catch (error) {
console.error('Fehler:', error);
}
}
alleOperationen();
```

Fazit

Die Callback-Hölle entsteht durch tief verschachtelte Funktionen bei der Arbeit mit asynchronem Code. Durch die Nutzung von Promises oder Async/Await kann der Code wesentlich lesbarer und wartbarer gestaltet werden.