Issues with Callbacks

1. Callback Hell

- **Definition**: A situation in asynchronous programming where nested callbacks make the code difficult to read and maintain.
- **Example**:

```
let cart = [1,2,3];
api.createOrder(cart, function() {
  api.proceedToPayment(function() {
    api.showSummary(function() {
    });
  });
});
```

2. Inversion of Control

- **Definition**: `showSummary` is dependent on `proceedToPayment`. If `proceedToPayment` succeeds, then only `showSummary` will execute.
- **Effect**: The control of `showSummary` is moved to `proceedToPayment`, which is called **inversion of control**.

📌 Watch Video

Promises

What is a Promise?

- A **Promise** is an object representing the eventual completion or failure of an asynchronous operation.

States of a Promise

- **Pending**: Initial state, neither fulfilled nor rejected.
- **Fulfilled**: Operation completed successfully.
- **Rejected**: Operation failed.

Key Features

- **Immutability**: Once a promise is fulfilled or rejected, its state cannot change.
- **Handling Multiple Promises**:

Method	Waits for all promises (Success Scenario)	Stops on failure (Failure Scenario)
`Promise.all`	✅ Yes	❌ No (Returns error if any promise fails)
`Promise.allSettled`	✅ Yes	✅ Yes (Waits for all promises even if some fail)
`Promise.race`	❌ No (Returns the first settled promise, success or failure)	❌ No
`Promise.any`	✅ Waits for the first resolved (successful) promise	❌ No (Fails only if all promises are rejected)

Example: Working with Promises

```
let p1 = new Promise((resolve, reject) => {
  setTimeout(() => { resolve("p1 success") }, 3000);
  // setTimeout(() => { reject("p1 fail") }, 3000);
});

let p2 = new Promise((resolve, reject) => {
  setTimeout(() => { resolve("p2 success") }, 1000);
  // setTimeout(() => { reject("p2 fail") }, 1000);
});

let p3 = new Promise((resolve, reject) => {
  setTimeout(() => { resolve("p3 success") }, 2000);
  // setTimeout(() => { reject("p3 fail") }, 2000);
});
```

1. Using `Promise.all`

```
Promise.all([p1, p2, p3]).then(result => {
  console.log(result);
}).catch(err => console.error(err));
```

- **Case**: `p1` , `p2` , `p3` all resolve → Output: `["p1 success", "p2 success", "p3 success"]` after 3s.
- **Case**: If `p2` rejects → Output: `"p2 fail"` after 1s.

2. Using `Promise.allSettled`

```
Promise.allSettled([p1, p2, p3]).then(result => {
  console.log(result);
});
```

- **All success case**:

```
[
  { "status": "fulfilled", "value": "p1 success" },
  { "status": "fulfilled", "value": "p2 success" },
  { "status": "fulfilled", "value": "p3 success" }
]
```

- **If `p2` is rejected**:

```
[
  { "status": "fulfilled", "value": "p1 success" },
  { "status": "rejected", "reason": "p2 fail" },
  { "status": "fulfilled", "value": "p3 success" }
]
```

3. Using `Promise.race`

```
Promise.race([p1, p2, p3]).then(result => {
  console.log(result);
});
```

- **First settled promise wins** → Output: `"p2 success"` (since `p2` resolves in 1s).

4. Using `Promise.any`

```
Promise.any([p1, p2, p3]).then(result => {
  console.log(result);
});
```

- **Case**: `p1` fulfilled, `p2` rejected, `p3` rejected → Output: `"p1 success"` after 3s.
- **Case**: All rejected → Output: `AggregateError: All promises were rejected` .

📌 Watch Video

Promise Chaining Examples

```
function one() {
  return new Promise((resolve) => {
    setTimeout(() => resolve(1), 1000);
  });
}

function two() {
  return new Promise((resolve) => {
    setTimeout(() => resolve(2), 2000);
  });
}

function three() {
  return new Promise((resolve) => {
    setTimeout(() => resolve(3), 3000);
  });
}
```

Using Async/Await

```
async function test() {
  try {
    // Sequentially calling APIs one after another
    const resp1 = await one();
    const resp2 = await two();
    const resp3 = await three();
    const output = resp1 + resp2 + resp3;
    console.log("Response is " + output);

    // Calling APIs in parallel
    const output1 = await Promise.allSettled([one(), two(), three()]);
    console.log("Response is " + JSON.stringify(output1));
  } catch (error) {
    console.log("Error is " + error);
  }
}

test();
```