Lesson Outline: Asynchronous JavaScript – Callbacks, Promises, and Async/Await

Lesson Duration: ~ 2 hours

Lesson Goals:

- Understand asynchronous programming and its importance.
- Learn about callbacks, promises, and async/await syntax in JavaScript.
- Apply asynchronous programming concepts through practical examples.

---

1. Introduction to Asynchronous Programming (5 min.)

- Explanation:
  - Synchronous vs. asynchronous programming.
  - Why asynchronous programming matters (e.g., non-blocking operations).
- Demonstration:

```
console.log('Start');
setTimeout(() => {
  console.log('2 Second Timer');
}, 2000);
console.log('End');
```

- Interactive Question:
  - Ask students to predict the output and explain why it happens.

---

2. Callback Functions (10 min.)

- Concept Explanation:
  - Definition: A callback is a function passed into another function as an argument, executed after some event or action.
  - Common use-cases (e.g., event handling, timers, async operations).
- Example:

```
function foo (msg, callback) {
  callback("Welcome to: " + msg);
};

foo("Big Blue Marble Academy", function (msg) {
  console.log(msg);
});
```

- Interactive Exercise:
  - Modify the example to add another callback function that performs a different operation.

---

3. Promises (20 min.)

- Concept Explanation:
  - Definition: A Promise represents the eventual completion or failure of an asynchronous operation.
  - Promise states: Pending, Fulfilled (resolved), and Rejected.
  - The purpose of using promises over callbacks (avoiding callback hell).
- Example:

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('Promise resolved');
  }, 2000);
});

promise.then(value => {
  console.log(value);
});
console.log('This line will print before the promise is resolved.');
```

- Interactive Exercise:
  - Create a Promise that resolves after 3 seconds and rejects under certain conditions. Handle both cases.

---

4. Async/Await (20 min.)

- Concept Explanation:

    - Simplifying promises with async/await.
    - async keyword makes a function return a Promise.
    - await keyword pauses async function execution until Promise settles.
- Example:

```
async function asyncFunction() {
  let promise = new Promise((resolve) => {
    setTimeout(() => {
      resolve('Promise resolved');
    }, 2000);
  });
  console.log(await promise);
}

asyncFunction();
console.log('This line will print before the asyncFunction finishes.');
```

- Interactive Challenge:
    - Rewrite the previous Promise exercise using async/await syntax.

---

5. TASK 1 student try first (30 min.)

Here's the given JavaScript comment converted into a markdown format for better readability:

Problem Statement:

The following code runs really slow because of the `sleep` function. Your job is to make the code run within **5 seconds**.

Conditions for Full Credit:

- **Do NOT** change the waiting time to make the code run faster.
- **Do NOT** change the function signature.
- You **CAN** change the implementation of the existing functions.
- You **CAN** add new functions or variables if needed.

Requirements:

- Both oven preheat and pie-making time are **5 seconds** each.

Hint:

- Utilizing **Promise**, **async/await** might help you solve this problem.

Additional Note:

- After finishing the task, refer to the `helper_code.js` file to understand how the `testTime` function works, enabling you to test your code.

Code:

```
function sleep(ms) {
  let start = Date.now();
  while (Date.now() - start < ms) {}
}

function preheatOven() {
  console.log("Oven is preheating...");
  sleep(5000);
  console.log("Oven preheat complete!");
}

function makePie() {
  console.log("Start making the pie...");
  sleep(5000);
  console.log("Pie is ready!");
}

async function run() {
  console.log("Starting tasks...");
  preheatOven();
  makePie();
  console.log("All tasks done!");
}

// testTime(run); // TODO: Uncomment this line to test your code
```

---

6. TASK 1 take up solution (30 min.)

- Now is time to take up solution

7. Recap & Discussion (10 min.)

- Review key concepts:
  - Callback functions
  - Promises
  - Async/Await
- Open Q&A:
  - Ask students about any difficulties or points needing clarification.
  - Discuss the advantages and potential pitfalls of each approach.