JavaScript Practical Lecture: Asynchronous Programming

Asynchronous programming allows JavaScript to perform long-running tasks (like network requests or file I/O) without blocking the main thread. Key approaches are Callbacks, Promises, and async/await.

1. Callbacks

A callback is a function passed as an argument to another function, executed after an operation finishes.

```
// Simulating an async task with setTimeout
function fetchData(callback) {
  console.log("Fetching data...");
  setTimeout(() => {
    const data = { user: "Alice", age: 25 };
    callback(data); // call the provided function
  }, 2000);
}

// Usage
fetchData((result) => {
  console.log("Data received:", result);
});
```

2. Promises

A Promise represents a value that may be available now, later, or never. It provides .then() and .catch() methods.

```
// Function returning a Promise
function fetchData() {
  return new Promise((resolve, reject) => {
    console.log("Fetching data...");
    setTimeout(() => {
      const success = true;
      if (success) {
        resolve({ user: "Bob", age: 30 });
      } else {
        reject("Error fetching data");
      }
    }, 2000);
  });
}

// Usage
fetchData()
  .then(result => {
    console.log("Data received:", result);
  })
  .catch(error => {
    console.error(error);
  });
```

3. async/await

async/await provides a cleaner way to write asynchronous code. It looks like synchronous code but works asynchronously.

```
// Function using async/await
function fetchData() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve({ user: "Charlie", age: 35 });
    }, 2000);
  });
}

async function getUser() {
  console.log("Fetching data...");
  const data = await fetchData(); // waits for promise to resolve
  console.log("Data received:", data);
}

// Usage
getUser();
```

Complete Example (HTML + JS)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Async Programming Demo</title>
</head>
<body>
  <h1>Async Programming: Callbacks, Promises, async/await</h1>
  <button id="callbackBtn">Run Callback Example</button>
  <button id="promiseBtn">Run Promise Example</button>
  <button id="asyncBtn">Run Async/Await Example</button>
  <pre id="output"></pre>

  <script>
    const output = document.getElementById("output");

    // Callback Example
    function fetchDataWithCallback(callback) {
      output.innerText += "Fetching with callback...\n";
      setTimeout(() => {
        callback({ user: "Alice", age: 25 });
      }, 1000);
    }

    document.getElementById("callbackBtn").addEventListener("click", () => {
      fetchDataWithCallback((data) => {
        output.innerText += "Callback received: " + JSON.stringify(data) + "\n";
      });
    });

    // Promise Example
    function fetchDataWithPromise() {
      return new Promise((resolve, reject) => {
        output.innerText += "Fetching with promise...\n";
        setTimeout(() => resolve({ user: "Bob", age: 30 }), 1000);
      });
    }

    document.getElementById("promiseBtn").addEventListener("click", () => {
      fetchDataWithPromise().then(data => {
        output.innerText += "Promise received: " + JSON.stringify(data) + "\n";
      });
    });

    // Async/Await Example
    function fetchDataAsync() {
      return new Promise(resolve => {
        output.innerText += "Fetching with async/await...\n";
        setTimeout(() => resolve({ user: "Charlie", age: 35 }), 1000);
      });
    }

    async function runAsync() {
      const data = await fetchDataAsync();
      output.innerText += "Async/Await received: " + JSON.stringify(data) + "\n";
    }

    document.getElementById("asyncBtn").addEventListener("click", runAsync);
  </script>
</body>
</html>
```