

Session 9

Javascript Asynchronous Programming

JavaScript is inherently a **single-threaded, synchronous language**, meaning it executes code line-by-line from top to bottom. However, modern web applications frequently require tasks that take time to complete (e.g., fetching data from a server, waiting for user input, or running timers). If JavaScript waited for these tasks to finish before moving to the next line, the entire browser UI would freeze. This session introduces **Asynchronous Programming**, which allows JavaScript to initiate a long-running task, continue executing other code, and handle the result later using **callbacks**. This session covers the foundational async patterns required for responsive, non-blocking web applications.

Objectives

✗ Do not copy. Read for understanding and the viva

- Understand the difference between synchronous and asynchronous execution
- Master callback functions as the primary mechanism for async flow control
- Implement built-in async methods: `setTimeout()` and `setInterval()`
- Load external resources dynamically using `XMLHttpRequest`
- Solve real-world async navigation and timing exercises

Synchronous vs. Asynchronous Execution

Feature	Synchronous	Asynchronous
Execution Flow	Blocks until task completes	Starts task, moves to next line immediately
UI Impact	Freezes browser during long tasks	Keeps UI responsive
Example	<pre>console.log("Start"); console.log("End");</pre>	<pre>setTimeout(() => console.log("End"), 2000);</pre>

Callback Functions

A **callback** is a function passed as an argument to another function. It is **not executed immediately**; instead, it's "called back" at a later time when the parent function completes its task.

```
1 // Function that accepts a callback
2 function fetchData(callback) {
3   // Simulate delay
4   setTimeout(() => {
5     const data = "User Data Loaded";
6     callback(data); // Execute callback when ready
7   }, 1000);
8 }
9
10 // Pass callback WITHOUT parentheses
11 fetchData((result) => console.log(result)); // Logs after 1 second
```

JS

Built-in Async Timer Methods

Method	Syntax	Behavior
<code>setTimeout()</code>	<code>setTimeout(callback, delayInMs)</code>	Executes callback once after <code>delay</code>
<code>setInterval()</code>	<code>setInterval(callback, intervalInMs)</code>	Executes callback repeatedly every <code>interval</code>
<code>clearTimeout()</code> / <code>clearInterval()</code>	<code>clearInterval(timerID)</code>	Stops the scheduled execution

```
1 // Example: Live Clock
2 function updateClock() {
3     const now = new Date();
4     document.getElementById('clock').textContent = now.toLocaleTimeString();
5 }
6 const clockTimer = setInterval(updateClock, 1000); // Runs every second
7 clearInterval(clockTimer); // Stops it
```

JS

Loading External Resources (XMLHttpRequest)

Before `fetch()` became standard, `XMLHttpRequest` (XHR) was the primary way to load external files asynchronously.

```
1 const xhr = new XMLHttpRequest();
2 xhr.open("GET", "external.html", true); // true = async
3 xhr.onload = function() {
4     if (xhr.status === 200) {
5         document.getElementById('container').innerHTML = xhr.responseText;
6     }
7 };
8 xhr.send(); // Triggers the request
```

JS

Best Practices & Common Pitfalls

✗ Do not copy. Read for understanding and the viva

Pitfall	Best Practice
Passing <code>callback()</code> with parentheses	Pass function reference only: <code>setTimeout(myFunc, 1000)</code> not <code>setTimeout(myFunc(), 1000)</code>
Assuming code after <code>setTimeout</code> runs later	<code>setTimeout</code> schedules execution; code immediately after it runs first
Forgetting to clear intervals	Always store <code>setInterval</code> ID and use <code>clearInterval()</code> to prevent memory leaks
Nesting callbacks deeply ("Callback Hell")	Keep nesting shallow; use named functions or modern <code>Promise</code> / <code>async-await</code> in production
Ignoring XHR <code>status</code> codes	Always check <code>xhr.status === 200</code> before processing <code>responseText</code>

Quick Guide

✖ Do not copy. Read for understanding and the viva

JS

```
1 // 🕒 Timers
2 const timeoutID = setTimeout(() => console.log("Once"), 2000);
3 clearTimeout(timeoutID);
4
5 const intervalID = setInterval(() => console.log("Repeated"), 1000);
6 clearInterval(intervalID);
7
8 // 🔄 Callback Pattern
9 function asyncTask(param, callback) {
10   // Do work...
11   callback(result);
12 }
13 asyncTask("data", (res) => console.log(res));
14
15 // 🌐 XMLHttpRequest (Lab Requirement)
16 const xhr = new XMLHttpRequest();
17 xhr.open("GET", "file.html", true);
18 xhr.onload = () => { if(xhr.status===200) console.log(xhr.responseText); };
19 xhr.send();
```

Question 1

Problem Statement

■ Write in lab record

Implement a clock in HTML which shows the live time in the format `HH:MM:SS`. Use the JavaScript `setInterval()` method and a callback function by displaying the system time every second.

Demo

✗ Do not copy. Read for understanding and the viva

Live Clock

11:23:55

Updates every second via setInterval()

Code

■ Write in lab record

time.html

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Ex1: Live Clock</title>
6     <style>
7       body {
8         font-family: sans-serif;
9         text-align: center;
10        padding: 50px;
11        background: #f4f4f4;
12      }
13    </style>
14  </head>
15  <body>
16    <h1>Live Clock</h1>
17    <div
18      id="clock"
19      style="
20        font-size: 4rem;
21        font-weight: bold;
22        font-family: monospace;
23        margin-top: 20px;
24      "
25    >
26      00:00:00
27    </div>
28
29    <script>
30      // Callback function to update time
31      function updateClock() {
32        const now = new Date();
33        const h = String(now.getHours()).padStart(2, "0");
34        const m = String(now.getMinutes()).padStart(2, "0");
35        const s = String(now.getSeconds()).padStart(2, "0");
36        document.getElementById("clock").textContent = `${h}:${m}:${s}`
37      }
38
39      // Set interval to execute callback every 1000ms
40      setInterval(updateClock, 1000);
41      updateClock(); // Initial call to avoid 1s delay
42    </script>
```

```
43     </body>
44 </html>
```

HTML View

Question 2

Problem Statement

■ Write in lab record

Sometimes we use external resource files in our HTML page. The content of an external file cannot be used until loaded completely. This can be implemented with JavaScript Callback. Design a web page which loads an external HTML file on the event: `onLoad`. Hint: use the inbuilt class `XMLHttpRequest()` to use its functions: `open()`, `onLoad()` etc. to load an external HTML file.

Demo

✕ Do not copy. Read for understanding and the viva

External Content Loader (onMount)

Loading external.html...

Code

■ Write in lab record

■ Lab record: every tab is one file of the answer. Write all of them.

request.html

request.html

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Ex2: XHR Loader</title>
6     <style>
7       body {
8         font-family: sans-serif;
9         padding: 20px;
10      }
11      #output {
12        border: 2px dashed #aaa;
13        padding: 20px;
14        margin-top: 15px;
15        background: #fafafa;
16        min-height: 50px;
17      }
18    </style>
19  </head>
20  <body>
21    <h2>External Content Loader (onLoad Event)</h2>
22    <div id="output">Loading external.html...</div>
23
24    <script>
25      window.addEventListener("load", () => {
26        const xhr = new XMLHttpRequest();
27        // true = asynchronous
28        xhr.open("GET", "external.html", true);
29
30        xhr.onload = function () {
31          if (xhr.status === 200) {
32            document.getElementById("output").innerHTML =
33              xhr.responseText;
34          } else {
35            document.getElementById("output").textContent =
36              `❌ Failed to load (Status: ${xhr.status})`;
37          }
38        };
39
40        xhr.onerror = () => {
41          document.getElementById("output").textContent =
42            "❌ Network error. Ensure server is running.";
43        };
44      });
45    </script>
46  </body>
47 </html>
```

```
44
45         xhr.send();
46     });
47     </script>
48 </body>
49 </html>
```

HTML View

Question 3

Problem Statement

■ Write in lab record

The village crows own an old scalpel that they occasionally use on special missions say, to cut through screen doors or packaging. To be able to quickly track it down, every time the scalpel is moved to another nest, an entry is added to the storage of both the nest that had it and the nest that took it, under the name "scalpel", with its new location as the value. This means that finding the scalpel is a matter of following the breadcrumb trail of storage entries, until you find a nest where that points at the nest itself.

Write an async function `locateScalpel` that does this, starting at the nest on which it runs. You can use the `anyStorage` function defined earlier to access storage in arbitrary nests. The scalpel has been going around long enough that you may assume that every nest has a "scalpel" entry in its data storage.

Demo

✗ Do not copy. Read for understanding and the viva



Scalpel Tracker

Find Scalpel

Code

■ Write in lab record

scalpel.html

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Ex3: Locate Scalpel</title>
6     <style>
7       body {
8         font-family: sans-serif;
9         padding: 20px;
10        max-width: 600px;
11        margin: auto;
12      }
13      pre {
14        background: #f4f4f4;
15        padding: 15px;
16        border-radius: 5px;
17        white-space: pre-wrap;
18      }
19    </style>
20  </head>
21  <body>
22    <h2>🦋 Scalpel Tracker</h2>
23    <button id="findBtn">Find Scalpel</button>
24    <pre id="output">Waiting...</pre>
25
26    <script>
27      // Mock nest data (simulating distributed storage)
28      const nestData = {
29        CrowNestA: { scalpel: "CrowNestB" },
30        CrowNestB: { scalpel: "CrowNestC" },
31        CrowNestC: { scalpel: "CrowNestC" }, // Points to itself → f
32      };
33
34      // Simulated async anyStorage function
35      function anyStorage(nest, name) {
36        return new Promise((resolve) => {
37          setTimeout(
38            () => resolve(nestData[nest]?.[name] || null),
39            600,
40          );
41        });
42      }
43
```

```
44      // Async locateScalpel implementation
45      async function locateScalpel(startNest) {
46          let current = startNest;
47          let trail = [current];
48
49          while (true) {
50              const next = await anyStorage(current, "scalpel");
51              if (next === current) break; // Found the self-referenci
52              trail.push(next);
53              current = next;
54          }
55          console.log({ foundAt: current, trail: trail.join(" → ") });
56          return { foundAt: current, trail: trail.join(" → ") };
57      }
58
59      document
60          .getElementById("findBtn")
61          .addEventListener("click", async () => {
62              document.getElementById("output").textContent =
63                  "🔍 Tracing scalpel breadcrumbs...";
64              const result = await locateScalpel("CrowNestA");
65              document.getElementById("output").textContent =
66                  `Found at: ${result.foundAt}\n👣 Trail: ${result.trail}
67          });
68      </script>
69  </body>
70 </html>
```

HTML View

[✎ Edit this page](#)

[< Previous](#)
Session 8

Session 10 [Next >](#)