JavaScript Synchronous & Asynchronous Programming — Complete Master Guide

This document explains **synchronous vs asynchronous JavaScript** in simple English with: - Definitions - Syntax - Input → Output - Step-by-step explanations - Real-time examples - setTimeout, Callbacks, Promises, async/await - Promise methods (Promise, Promise.all, Promise.race, Promise.allSettled) - Interview questions with answers

---

1. What is Synchronous JavaScript?

Definition

Synchronous JavaScript means **code runs line-by-line**, in order. One line must finish before the next one starts.

Simple Explanation

- JavaScript has only **one thread**.
- So it executes code **one step at a time**.
- If one step takes time → everything waits.

Example

```
console.log("Start");
console.log("Middle");
console.log("End");
```

Output

```
Start
Middle
End
```

Everything runs in exact order.

---

2. What is Asynchronous JavaScript?

Definition

Asynchronous JavaScript means code **can start**, but JavaScript continues running without waiting for it to finish.

Simple Explanation

- Long tasks run "in the background".
- When they finish, JavaScript handles the result.
- Used for: API calls, timers, reading files, DB calls.

Basic Example (setTimeout)

```
console.log("Start");

setTimeout(function() {
  console.log("Timeout finished");
}, 2000);

console.log("End");
```

Output

```
Start
End
Timeout finished
```

Because setTimeout is **asynchronous**, it runs after everything else.

---

3. How JavaScript Achieves Asynchronous Behavior

JavaScript uses: - **Callback functions** - **Promises** - **async/await**

These do not block the main thread.

---

4. Callback Functions

Definition

A function passed as an argument to another function, to run later.

Syntax

```
function task(callback) {
  callback();
}
```

Example

```
function greet(name, callback) {
  console.log("Hello " + name);
  callback();
}

greet("John", function() {
  console.log("Callback executed");
});
```

Output

```
Hello John
Callback executed
```

Real-time Example

```
setTimeout(function() {
  console.log("Data loaded");
}, 1000);
```

Problem with Callbacks

- Callback Hell (nested callbacks)
- Hard to debug

---

5. Promises

Definition

A Promise represents a **value that will be available in the future**.

States: - **pending** → still working - **resolved** → success - **rejected** → error

Syntax

```
let p = new Promise(function(resolve, reject) {
  resolve("Done");
});
```

Basic Promise Example

```
let p = new Promise(function(resolve, reject) {
  setTimeout(() => resolve("Task completed"), 2000);
});

p.then(function(result) {
  console.log(result);
});
```

Output

```
Task completed
```

Promise with Error

```
let p = new Promise(function(resolve, reject) {
  reject("Something went wrong");
});

p.catch(function(error) {
  console.log(error);
});
```

---

6. Promise Methods

6.1 Promise.all()

Runs all promises together. Returns result **only when all are resolved**.

Example

```
let p1 = Promise.resolve(10);
let p2 = Promise.resolve(20);
let p3 = Promise.resolve(30);

Promise.all([p1, p2, p3]).then(function(values) {
  console.log(values);
});
```

Output

```
[10, 20, 30]
```

6.2 Promise.race()

Returns result of **first completed promise** (resolved or rejected).

Example

```
let p1 = new Promise(res => setTimeout(res, 100, "A"));
let p2 = new Promise(res => setTimeout(res, 50, "B"));

Promise.race([p1, p2]).then(console.log);
```

Output

```
B
```

6.3 Promise.allSettled()

Waits for all promises to finish (resolved or rejected).

Example

```
let p1 = Promise.resolve("Done");
let p2 = Promise.reject("Error");

Promise.allSettled([p1, p2]).then(console.log);
```

Output

```
[{status:"fulfilled", value:"Done"}, {status:"rejected", reason:"Error"}]
```

---

7. async / await

Definition

Syntax that makes asynchronous code look like synchronous code.

Syntax

```
async function demo() {
  let result = await promise;
}
```

Example

```
function getData() {
  return new Promise(resolve => {
    setTimeout(() => resolve("Data Loaded"), 2000);
  });
}

async function show() {
  let output = await getData();
  console.log(output);
}

show();
```

Output

```
Data Loaded
```

Why async/await?

- Cleaner than promises
- Readable like normal code
- No callback hell

---

8. Synchronous vs Asynchronous — Comparison Table

Feature	Synchronous	Asynchronous
Execution	Line by line	Non-blocking
Waiting	Yes	No
Used for	Normal tasks	API calls, timers
Example	console.log	setTimeout, fetch

---

9. Summary

- JavaScript is single-threaded.
- Async behavior achieved using callbacks, promises, and async/await.
- Promises help manage future values.
- async/await makes async code cleaner.

- JavaScript is single-threaded.
- Async behavior achieved using callbacks, promises, and async/await.
- Promises help manage future values.
- async/await makes async code cleaner.

---

11. Advanced Asynchronous Concepts (Detailed)

11.1 What is a Promise? (Deep Explanation)

A **Promise** is an object that represents a task that: - is happening now (pending), - will finish successfully (resolved), or - will fail (rejected).

A Promise has **two handlers**: - `then()` → runs when promise resolves. - `catch()` → runs when promise fails. - `finally()` → runs always.

Example (Step-by-step)

```
let p = new Promise(function(resolve, reject) {
  let success = true;

  if (success) {
    resolve("Task success");
  } else {
    reject("Task failed");
  }
});

p.then(res => console.log(res))
 .catch(err => console.log(err))
 .finally(() => console.log("Completed"));
```

11.2 All Promise Methods (Complete List)

✔️then()

Used to read the resolved value.

```
p.then(value => console.log(value));
```

✔️catch()

Handles errors.

```
p.catch(err => console.log(err));
```

✔️finally()

Runs whether success or failure.

```
p.finally(() => console.log("done"));
```

✔️Promise.resolve()

Creates a resolved promise.

```
let p = Promise.resolve(100);
p.then(v => console.log(v));
```

✔️Promise.reject()

Creates a rejected promise.

```
let p = Promise.reject("Error");
p.catch(e => console.log(e));
```

✔️Promise.all()

Waits for **all** promises.

✔️Promise.race()

Returns **first settled** promise.

✔️Promise.allSettled()

Waits for all, returns success + failures.

✔️Promise.any()

Returns **first resolved** promise, ignores rejections.

```
let p1 = Promise.reject("Error 1");
let p2 = Promise.resolve("Success");

Promise.any([p1, p2]).then(console.log); // Success
```

11.3 Callback vs Promise vs Async/Await

Callback

- Function passed inside another function.
- Can become messy → callback hell.

Promise

- Cleaner than callbacks.
- Handles future values.
- Uses `then()`, `catch()`.

Async/Await

- Built on top of promises.
- Looks like synchronous code.
- Easier to read.

---

12. Deep Examples (Real Scenarios)

12.1 Simulating an API Call (Promise)

```
function fetchUser() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve({ id: 1, name: "John" });
    }, 2000);
  });
}

fetchUser().then(data => console.log(data));
```

12.2 Same Example Using async/await

```
async function loadUser() {
  let user = await fetchUser();
  console.log(user);
}

loadUser();
```

---

13. Interview Questions (Expanded)

Q1: What is synchronous JavaScript?

JavaScript executes one line at a time. Next line waits for previous to finish.

Q2: What is asynchronous JavaScript?

JavaScript can start long tasks but does not wait — continues running.

Q3: What is a callback?

A function passed to another function to run later.

Q4: What is a promise?

An object representing a value that will come in the future.

Q5: Promise states?

Pending, fulfilled, rejected.

Q6: What is async/await?

A syntactic feature that allows writing asynchronous code like synchronous code.

Q7: Difference between callback, promise, async/await?

Callbacks → messy. Promises → cleaner. Async/Await → simplest & readable.

Q8: Difference between Promise.all & Promise.any?

all waits for all to succeed. any returns first successful.

Q9: What is the event loop?

Mechanism that handles asynchronous tasks, using callback queue & microtask queue.

---

14. End of Document