PROSENSIA PVT LTD

M FAROOQ SAJID

Asynchronous JavaScript

1.Introduction

Asynchronous JavaScript allows for **non-blocking execution**, meaning the program can continue running while waiting for operations like timers, user interactions, or server responses.

This report covers three core asynchronous features in JavaScript:

setTimeout

setInterval

Promises

These tools are essential for responsive, event-driven programming in modern web applications.

2.setTimeout()

Purpose

Executes a function **once** after a specified delay (in milliseconds).

Syntax

javascript
CopyEdit
setTimeout(function, delay)

Example

javascript
CopyEdit
setTimeout(() => {

```
  console.log('Executed after 2 seconds');
}, 2000);
```

---

3. setInterval()

Purpose

Executes a function **repeatedly** at specified intervals.

Syntax

javascript
CopyEdit
setInterval(function, interval)

Example

javascript
CopyEdit
```
setInterval(() => {
  console.log('Runs every 1 second');
}, 1000);
```

---

4. Promises

Purpose

A Promise is an object representing the eventual **completion or failure** of an asynchronous operation and its resulting value.

Promise States

Pending – initial state

Fulfilled – operation completed successfully

Rejected – operation failed

Creating a Promise

javascript
CopyEdit

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => resolve('Success!'), 2000);
});
```

Using a Promise

```
javascript
CopyEdit
myPromise
  .then(result => {
    console.log(result); // Success!
  })
  .catch(error => {
    console.error(error);
  });
```

---

5. Summary and Use Cases

✅ Use `setTimeout()` for delayed execution (e.g., splash screens, notifications)

Use `setInterval()` for repeating actions (e.g., clocks, live updates)

Use `Promises` to manage API requests, async tasks, and clean up callback hell