Q.2 How to access and update object properties using dot notation and bracket notation?

Ans : 1. Dot Notation

Use a dot (.) followed by the property name.

```
let student = { name: "Rahul", age: 21 };

// Access
console.log(student.name); // Output: Rahul

// Update
student.age = 22;
console.log(student.age); // Output: 22
```

2. Bracket Notation

Use square brackets ([]) with the property name in quotes.

```
let student = { name: "Rahul", age: 21 };

// Access
console.log(student["name"]); // Output: Rahul

// Update
student["age"] = 23;
console.log(student["age"]); // Output: 23
```

JavaScript Events

Q.1 What are JavaScript events? Explain the role of event listeners.

Ans : What are JavaScript events?
Events are actions or occurrences that happen in the browser, which JavaScript can respond to.
Examples of events:

- A user clicks a button (click event)
- A user hovers over an element (mouseover event)
- A form is submitted (submit event)
- A web page **loads** (load event)

Q.2 How does the addEventListener() method work in JavaScript?

Ans : What it does:
addEventListener() attaches an event listener to an element without overwriting existing event handlers.

Syntax:

```
element.addEventListener(event, function, useCapture);
```

- event → the event type (e.g., "click", "mouseover")
- function → the code (callback) to run when event happens
- useCapture → optional, true/false (default is false)

- ## DOM Manipulation

Q.1 What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?

Ans : Definition:
The DOM (Document Object Model) is a tree-like structure that represents the entire content of an HTML or XML document.

- In the browser, every element (like <p>, <div>, <img>) becomes a node in this tree.
- This structure allows programming languages (like JavaScript) to access and manipulate the page dynamically.

How JavaScript interacts with the DOM:
JavaScript can:

- **Select** elements from the DOM (e.g., headings, buttons, forms),
- **Change** content (text, HTML),
- **Change styles** (CSS properties),
- **Add or remove elements**,
- **Listen to user events** (like clicks, input, mouseover).

Example:

```
document.getElementById("demo").innerText = "Hello World!";
```

Q.2 Methods to select elements from the DOM.

Ans :

Method	What it does	Returns	Example
getElementById()	Selects a single element by its unique id.	A single element (or null if not found).	document.getElementById("header")
getElementsByClassName()	Selects all elements that have a given class name.	An **HTMLCollection** (array-like).	document.getElementsByClassName("menu-item")
querySelector()	Selects the **first element** that matches a CSS selector.	A single element (or null).	document.querySelector(".menu-item") or document.querySelector("#header")

Quick Code Examples:

html

CopyEdit

```
<h1 id="title">My Title</h1>
<p class="info">Paragraph 1</p>
<p class="info">Paragraph 2</p>


<script>
// Using getElementById
let title = document.getElementById("title");
title.style.color = "blue";
```

```
// Using getElementsByClassName
let infos = document.getElementsByClassName("info");
infos[0].innerText = "First paragraph changed!";
infos[1].style.fontWeight = "bold";

// Using querySelector
let firstInfo = document.querySelector(".info");
firstInfo.style.backgroundColor = "yellow";
</script>
```

- **JavaScript Timing Events (setTimeout, setInterval)**

Q.1 **Explain setTimeout() and setInterval() in JavaScript. How are they used for timing events?**

Ans : setTimeout()

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

Syntaxt :

setTimeout(function, delay)

When to use :

When you want to run code after some time.

setInterval()

Executes a function repeatedly at a specified time interval (in milliseconds) until you stop it.

Syntax :

setInterval(function, delay)

When to use :

When you want to run code repeatedly after every interval.

Usage in timing events:

- To **delay** actions (like showing a message after 2 seconds) → setTimeout().
- To **repeat** actions (like updating a clock every second) → setInterval().

Q.2 Example of setTimeout() to delay an action by 2 seconds.

Ans : Example: Show an alert message after 2 seconds.

javascript

CopyEdit

```
setTimeout(function() {
  alert("Hello! This message appeared after 2 seconds.");
}, 2000); // 2000 milliseconds = 2 seconds
```

✅ How it works:

- setTimeout() waits 2000 ms (2 seconds),
- Then executes the function inside it only once.

• JavaScript Error Handling

Q.1 What is error handling in JavaScript?

Ans : Error handling in JavaScript means detecting and managing runtime errors (errors that happen when the script is running) so that your application does not crash unexpectedly.
Instead of stopping execution, you can handle the error gracefully and keep the app running.

JavaScript provides try…catch…finally blocks for error handling.

try block

- Place the code that might throw an error inside try.
- If no error occurs → code runs normally.

catch block

- If an error happens inside try, execution jumps to catch.
- catch receives the error object and you can handle or log the error.

finally block (optional)

- Code inside finally always runs **after try/catch**, whether an error occurred or not.
- Useful for cleanup tasks (closing connections, clearing intervals, etc.).

Q.2 Why is error handling important in JavaScript applications?

Ans : Prevents application crashes:
Without error handling, one unexpected error can stop your whole script.

Improves user experience:
Instead of showing a broken page, you can show a friendly message like *"Something went wrong, please try again."*

Easier debugging:
You can log errors for developers to investigate later.

Keeps critical parts running:
Even if one part fails, the rest of the app can continue to work.