Timers (setTimeout, setInterval)

Timers are used to **run code after a delay** or **repeatedly** after intervals.

setTimeout()

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

Example:

```
setTimeout(function() {
  console.log("This message appears after 3 seconds");
}, 3000);
```

setInterval()

Executes a function **repeatedly** after a specified interval.

Example:

```
let count = 1;

let timer = setInterval(function() {
  console.log("Count:", count);
  count++;

  if (count > 5) {
    clearInterval(timer); // stop after 5 times
  }
}, 1000);
```

clearTimeout()

Cancels a timer set with `setTimeout()`.

```
let timeout = setTimeout(() => console.log("Will not run"), 5000);
clearTimeout(timeout);
```

Popup Boxes (alert, prompt, confirm)

JavaScript provides **three popup boxes** for simple user interaction.

alert()

Displays a message box with an **OK** button.

```
alert("Welcome to JavaScript!");
```

prompt()

Displays a box asking for **user input**.
You can store the entered value in a variable.

```
let name = prompt("Enter your name:");
alert("Hello, " + name + "!");
```

confirm()

Displays a box asking for **Yes/No confirmation**.
Returns `true` if OK is clicked, otherwise `false`.

```
let result = confirm("Are you sure you want to exit?");
if (result) {
  alert("Exiting...");
} else {
  alert("Cancelled!");
}
```

Summary

Concept	Description
BOM	Interacts with browser (window, navigator, history, location)
Timers	Delay or repeat actions (`setTimeout`, `setInterval`)
Popups	User messages or confirmations (`alert`, `prompt`, `confirm`)

Escape Sequence Characters in JavaScript

Escape sequences are special characters used inside strings when you want to represent something that cannot be typed directly.

1. `\n` – New Line

Starts a new line.

```
console.log("Hello\nWorld");
```

2. \t – Tab

Inserts a horizontal tab space.

```
console.log("Name:\tNeeraj");
```

3. \\ – Backslash

Prints a real backslash.

```
console.log("This is a backslash: \\");
```

4. \' – Single Quote

Used to print a single quote inside a string.

```
console.log('It\'s a good day');
```

5. \" – Double Quote

Used to print double quotes inside a string.

```
console.log("He said \"Hello\"");
```

Destructuring (Array & Object)

Destructuring allows you to **extract values** from arrays or objects easily.

Array Destructuring

```
let numbers = [10, 20, 30];

let [a, b, c] = numbers;

console.log(a); // 10
console.log(b); // 20
console.log(c); // 30
```

Skipping values:

```
let [x, , z] = [1, 2, 3];
console.log(x, z); // 1 3
```

With default values:

```
let [p, q = 5] = [10];
console.log(p, q); // 10 5
```

Object Destructuring

```
let student = { name: "Amit", age: 21, city: "Lucknow" };

let { name, city } = student;

console.log(name); // Amit
console.log(city); // Lucknow
```

Renaming while destructuring:

```
let { name: fullName, age: years } = student;
console.log(fullName, years); // Amit 21
```

Spread & Rest Operators (...)

Both use `...` but work **differently** depending on the context.

Spread Operator

Used to **expand** arrays or objects.

Example (Array):

```
let arr1 = [1, 2, 3];
let arr2 = [...arr1, 4, 5];
console.log(arr2); // [1, 2, 3, 4, 5]
```

Example (Object):

```
let obj1 = { name: "Neeraj", age: 23 };
let obj2 = { ...obj1, city: "Lucknow" };
console.log(obj2);
// { name: "Neeraj", age: 23, city: "Lucknow" }
```

Rest Operator

Used to **collect remaining values** into an array or object.

Example (Array):

```
let [a, b, ...rest] = [10, 20, 30, 40, 50];
console.log(a);    // 10
console.log(b);    // 20
console.log(rest); // [30, 40, 50]
```

Example (Function Parameters):

```
function sum(...numbers) {
  return numbers.reduce((total, num) => total + num, 0);
}
console.log(sum(1, 2, 3, 4)); // 10
```

Console Object (console)

The **console object** is used to display messages in the browser's console (mainly for debugging).

Common Console Methods

a) `console.log()`

Prints general messages or values.

```
console.log("Hello JavaScript");
```

b) `console.error()`

Shows an error message in red.

```
console.error("Something went wrong!");
```

c) `console.warn()`

Displays a warning message in yellow.

```
console.warn("This is a warning");
```

d) `console.table()`

Displays data in a table form.

```
console.table([1, 2, 3]);
```

e) `console.clear()`

Clears the console.

```
console.clear();
```

f) `console.info()`

Shows informational messages.

```
console.info("This is some useful info.");
```

Quick Summary Table

Function	What it does	Return Value
`alert()`	Shows a message popup	No return (undefined)
`prompt()`	Takes user input	String or `null`
`confirm()`	Asks Yes/No	true / false
`console.log()`	Prints message	void
`console.error()`	Prints error	void
`console.warn()`	Prints warning	void
`console.table()`	Prints table	void

Window Object

The **window object** is the **top-level object** in the browser.
Everything in the browser (BOM, DOM, console, alert, etc.) is part of `window`.

Key Points

- Every global variable and function automatically becomes a property of `window`.
- It represents the browser window/tab.
- Many functions like `alert()`, `prompt()`, `confirm()`, `setTimeout()` belong to the window object.

Examples

```
window.alert("Hello");   // same as alert("Hello")
console.log(window.innerWidth);   // width of the browser window
console.log(window.location.href); // current page URL
```

Common Window Properties

- `window.innerWidth` → width of viewport
- `window.innerHeight` → height of viewport
- `window.location` → URL information
- `window.history` → browser history
- `window.navigator` → browser information

BOM (Browser Object Model)

BOM refers to **browser-specific objects** that allow JavaScript to interact with the browser environment.

BOM = Browser Features (NOT HTML elements)

BOM Includes:

- ✓ **window** (root)
- ✓ **location**
- ✓ **history**
- ✓ **navigator**
- ✓ **screen**
- ✓ **alert, prompt, confirm**
- ✓ **setTimeout, setInterval**

Examples

a) window.location

```
console.log(window.location.href);
window.location.reload();   // reload the page
```

b) window.history

```
history.back();   // go to previous page
history.forward(); // next page
```

c) navigator

```
console.log(navigator.userAgent);
```

d) setTimeout

```
setTimeout(() => console.log("Hello"), 2000);
```

DOM (Document Object Model)

DOM represents the **HTML document** as a tree of objects (nodes).
Using DOM, we can **access, modify, add, or delete HTML elements.**

DOM = HTML + CSS as Objects

The browser converts your HTML into a JavaScript object called the **document**.

Common DOM Operations

a) Selecting Elements

```
document.getElementById("title");
document.querySelector(".box");
document.getElementsByClassName("item");
```

b) Changing Content

```
document.getElementById("title").innerHTML = "New Title";
```

c) Changing Styles

```
document.querySelector("h1").style.color = "red";
```

d) Creating New Elements

```
let btn = document.createElement("button");
btn.innerText = "Click Me";
document.body.appendChild(btn);
```

e) Event Handling

```
document.getElementById("btn").addEventListener("click", function() {
    alert("Button clicked!");
});
```

Clear Difference – Window vs BOM vs DOM

Feature	Window	BOM	DOM
Meaning	Main global browser object	Browser features	HTML document
Contains	BOM + DOM	location, history, navigator	HTML elements
Purpose	Control browser tab	Control browser features	Manipulate webpage
Example	`window.alert()`	`window.location.href`	`document.getElementById()`

In Simple Words

- **Window** → The boss (main container of everything)
- **BOM** → Browser features (URL, history, screen, alert, prompt)
- **DOM** → HTML elements (document, tags, text, attributes)

JavaScript DOM Nodes

In the DOM, everything inside an HTML document is represented as a **node**.

Types of DOM Nodes

1. **Element Node**
2. **Text Node**
3. **Comment Node**
4. **Document Node**
5. **Attribute Node** (not commonly used directly)

Element Node

Represents an HTML element.

Example:

```
<p>Hello</p>
```

Here `<p>` is an **element node**.

JavaScript Example:

```
let para = document.querySelector("p");
console.log(para.nodeType); // 1 (Element node)
```

2. Text Node

Represents the **text inside an element**.

Example:

```
<p>Hello</p>
```

The word **"Hello"** is a **text node**.

JavaScript Example:

```
let text = document.querySelector("p").firstChild;
console.log(text.nodeType); // 3 (Text node)
```

3. Comment Node

Represents an HTML comment.

Example:

```
<!-- This is a comment -->
```

JavaScript Example:

```
let comment = document.body.childNodes[1];
console.log(comment.nodeType); // 8 (Comment node)
```

Walking the DOM

“Walking the DOM” means moving through nodes like:

- parent → child
- child → parent
- element → next sibling
- element → previous sibling
- accessing descendants

4. Parent Node

The element **above** another node.

Example:

```
let child = document.querySelector("p");
console.log(child.parentNode);   // returns the parent element
```

5. Child Nodes

Nodes that are **directly inside** a parent.

Properties:

- `element.childNodes` → returns all child nodes (including text + comment)
- `element.children` → only element nodes

Example:

```
let list = document.querySelector("ul");
console.log(list.children);   // only <li> elements
console.log(list.childNodes); // includes text nodes also
```

6. First Child & Last Child

`firstChild`

Returns the first node (can be a text node).

`firstElementChild`

Returns the first **element** node only.

Example:

```
let div = document.querySelector("div");
console.log(div.firstChild);        // maybe text node
console.log(div.firstElementChild); // always element
```

`lastChild`

Returns the last node.

`lastElementChild`

Returns the last element node.

7. Sibling Nodes

Nodes that share the same parent.

Useful Properties:

- `nextSibling` (may return text node)
- `previousSibling`
- `nextElementSibling`
- `previousElementSibling`

Example:

```
let item = document.querySelector("li");
console.log(item.nextElementSibling);
console.log(item.previousElementSibling);
```

8. Descendant Nodes

Nodes that are inside an element at **any level** (child, grandchild, great-grandchild, etc.).

Example:

```
let wrapper = document.querySelector("#box");
console.log(wrapper.querySelectorAll("*")); // all descendants
```

Node Type Values (Important for Exam)

Node Type	Meaning
`1`	Element Node
`3`	Text Node
`8`	Comment Node
`9`	Document Node

Example:

```
console.log(node.nodeType);
```

Short Summary for Quick Revision

- **Element Node** → `<p>`, `<div>`, `<h1>`
- **Text Node** → Text inside elements
- **Comment Node** → `<!-- comment -->`
- **parentNode** → parent element
- **childNodes** → all children including text
- **children** → only element children
- **firstChild / lastChild** → first/last node
- **firstElementChild / lastElementChild** → first/last element
- **nextSibling / previousSibling** → may include text nodes
- **nextElementSibling / previousElementSibling** → only elements
- **descendants** → all nested nodes inside an element

```
Document
  │
  └── <html>  (Element Node)
        │
        ├── <head>  (Element Node)
        │     │
        │     └── <title>  (Element Node)
        │           │
        │           └── "My Page"  (Text Node)
        │
        └── <body>  (Element Node)
              │
              ├── <!-- This is a comment -->  (Comment Node)
              │
              ├── <h1>  (Element Node)
              │     │
              │     └── "Welcome"  (Text Node)
              │
              ├── <p>  (Element Node)
              │     │
              │     └── "This is a paragraph."  (Text Node)
              │
              └── <ul>  (Element Node)
                     │
                     ├── <li>  (Element Node)
                     │      └── "Item 1" (Text Node)
                     │
                     ├── <li>  (Element Node)
                     │      └── "Item 2" (Text Node)
                     │
                     └── <li>  (Element Node)
                            └── "Item 3" (Text Node)
```