**27. How do you add an external stylesheet in your HTML?**

An **external stylesheet** is a separate .css file that contains CSS rules and is linked to the HTML document. This allows you to **separate content (HTML) from design (CSS)** and reuse the same style across multiple pages.

**Steps to add an external stylesheet:**

* First, create a CSS file. For example:  
  styles.css
* body {
* background-color: lightblue;
* }
* h1 {
* color: navy;
* font-size: 24px;
* }
* Then, link this file in the <head> section of your HTML document using the <link> tag:
* <!DOCTYPE html>
* <html>
* <head>
* <title>External Stylesheet Example</title>
* <link rel="stylesheet" href="styles.css">
* </head>
* <body>
* <h1>Hello World!</h1>
* </body>
* </html>

**Explanation of <link> tag attributes:**

* rel="stylesheet" — specifies the relationship between the HTML file and the linked file.
* href="styles.css" — provides the path to the CSS file.
* Optionally, you can also add type="text/css" but in modern HTML5, it’s not mandatory.

✅ **Good practice**: Place your <link> inside the <head> tag to ensure styles are loaded before the page content is displayed.

**28. How do you open a link in a new tab?**

To open a hyperlink in a **new browser tab**, you use the target attribute with the value \_blank inside the <a> (anchor) tag.

**Example:**

<a href="https://www.openai.com" target="\_blank">Visit OpenAI</a>

**Explanation of attributes:**

* href="URL" — the URL you want the user to go to.
* target="\_blank" — instructs the browser to open the link in a **new tab or window** (depending on browser settings).

**Additional Tip:**  
For security reasons, it's good practice to also include:

rel="noopener noreferrer"

so the full link becomes:

<a href="https://www.openai.com" target="\_blank" rel="noopener noreferrer">Visit OpenAI</a>

* noopener — prevents the new page from controlling the original page.
* noreferrer — prevents the browser from sending the address of the referring page.

✅ **Why use it?**  
It protects against certain types of phishing attacks (like tab-nabbing).

**29. How do you create an Email Link?**

An **email link** allows users to click and open their default mail application (like Gmail, Outlook, etc.) with a new email draft automatically addressed to a specified email.

You use the <a> tag with the href attribute starting with mailto: followed by the email address.

**Example:**

<a href="mailto:someone@example.com">Send Email</a>

When the user clicks it, their email app will open with:

* "To" field pre-filled with someone@example.com.

**You can also pre-fill subject and body like this:**

<a href="mailto:someone@example.com?subject=Hello&body=This%20is%20a%20test%20email.">Send Custom Email</a>

* subject=Hello — fills in the email subject line.
* body=This%20is%20a%20test%20email. — %20 represents a space in the URL encoding.

✅ **Note:** Email links depend on the user having an email client configured.

**30. What are the different Types of Lists in HTML?**

There are **three** main types of lists in HTML:

| **Type** | **Tag Used** | **Description** |
| --- | --- | --- |
| **Ordered List** | <ol> | A list of items with numbers (1, 2, 3, ...) or letters (a, b, c, ...) |
| **Unordered List** | <ul> | A list of items with bullet points |
| **Description List** | <dl> | A list of terms and their descriptions |

**1. Ordered List (<ol>) Example:**

<ol>

<li>Wake up</li>

<li>Brush Teeth</li>

<li>Eat Breakfast</li>

</ol>

**Output:**

1. Wake up

2. Brush Teeth

3. Eat Breakfast

**2. Unordered List (<ul>) Example:**

<ul>

<li>Apple</li>

<li>Banana</li>

<li>Cherry</li>

</ul>

**Output:**

• Apple

• Banana

• Cherry

**3. Description List (<dl>) Example:**

<dl>

<dt>HTML</dt>

<dd>HyperText Markup Language</dd>

<dt>CSS</dt>

<dd>Cascading Style Sheets</dd>

</dl>

* <dt> = Definition Term (the title)
* <dd> = Definition Description (the explanation)

✅ **Important:** Lists help organize information in a readable format.

**31. What is a Nested List in HTML?**

A **nested list** means having a list **inside** another list.  
You can nest:

* <ul> inside <ul>,
* <ol> inside <ol>,
* or mix <ul> and <ol>.

**Example of a Nested List (Unordered inside Unordered):**

<ul>

<li>Fruits

<ul>

<li>Apple</li>

<li>Banana</li>

<li>Mango</li>

</ul>

</li>

<li>Vegetables

<ul>

<li>Carrot</li>

<li>Potato</li>

</ul>

</li>

</ul>

**Output:**

* Fruits
  + Apple
  + Banana
  + Mango
* Vegetables
  + Carrot
  + Potato

**Example of a Mixed Nested List (Ordered inside Unordered):**

<ul>

<li>Tasks

<ol>

<li>Morning Run</li>

<li>Breakfast</li>

<li>Work</li>

</ol>

</li>

</ul>

✅ **Why use Nested Lists?**  
They help represent **hierarchical** (tree-like) data structures clearly.

**32. What are <table>, <tr>, <th>, and <td> elements?**

In HTML, to create tables, you use these four important tags:

| **Tag** | **Full Form** | **Purpose** |
| --- | --- | --- |
| <table> | Table | Defines the entire table structure. |
| <tr> | Table Row | Defines a row in the table. |
| <th> | Table Header | Defines a header cell (bold and centered by default). |
| <td> | Table Data | Defines a regular data cell. |

**Example:**

html

CopyEdit

<table border="1">

<tr>

<th>Subject</th>

<th>Marks</th>

</tr>

<tr>

<td>Math</td>

<td>95</td>

</tr>

<tr>

<td>Science</td>

<td>89</td>

</tr>

</table>

✅ **Explanation**:

* <table>: Starts the table.
* <tr>: Defines each row.
* <th>: Defines headings (Subject, Marks).
* <td>: Defines actual data (Math, 95).

**33. What is the colspan attribute in HTML?**

The colspan attribute **combines two or more columns** into a single cell.

✅ It is used inside a <td> or <th> tag.

**Syntax:**

html

CopyEdit

<td colspan="number\_of\_columns">content</td>

**Example:**

html

CopyEdit

<table border="1">

<tr>

<th colspan="2">Student Info</th>

</tr>

<tr>

<td>Name</td>

<td>John Doe</td>

</tr>

</table>

**Explanation**:

* colspan="2" makes the "Student Info" heading spread across two columns.

**34. What is the best way to add a border to a table, column, and cell?**

**Best Practice**: Use **CSS** for styling instead of using the old HTML border attribute.

✅ CSS is cleaner, more powerful, and separates style from content.

**Example using CSS:**

html

CopyEdit

<style>

table, th, td {

border: 2px solid black;

border-collapse: collapse; /\* Makes borders look neat \*/

}

</style>

<table>

<tr>

<th>Subject</th>

<th>Marks</th>

</tr>

<tr>

<td>Math</td>

<td>95</td>

</tr>

</table>

* border-collapse: collapse; — removes double borders between cells.

**35. What is CSS? What are the 3 ways to implement CSS in HTML?**

**CSS** stands for **Cascading Style Sheets**.  
It is used to **style and layout** HTML elements — like setting colors, fonts, spacing, etc.

**Three Ways to Implement CSS:**

| **Way** | **Description** | **Example** |
| --- | --- | --- |
| **Inline CSS** | Add CSS directly inside an HTML tag using style attribute. | <h1 style="color:blue;">Hello</h1> |
| **Internal CSS** | Write CSS rules inside a <style> tag within the <head>. | <style>h1 { color: blue; }</style> |
| **External CSS** | Create a separate .css file and link it. | <link rel="stylesheet" href="styles.css"> |

**36. What is Inline Style in CSS? When to use it in real applications?**

**Inline Style**: CSS that is written directly inside an HTML element using the style attribute.

**Example:**

html

CopyEdit

<p style="color:green; font-size:20px;">This is a paragraph.</p>

**When to use Inline CSS:**

* Quick testing.
* Styling a single element.
* Overriding external/internal styles (higher specificity).
* Small emails or newsletters (where CSS files cannot be attached easily).

✅ **However:** It's **not recommended** for large projects because it mixes content and style, making the code messy and hard to maintain.

**37. What is Internal Stylesheet in CSS? When to use it in real applications?**

**Internal Stylesheet**: CSS written inside a <style> tag in the <head> section of the HTML document.

**Example:**

html

CopyEdit

<head>

<style>

body {

background-color: lightyellow;

}

h1 {

color: green;

}

</style>

</head>

**When to use Internal CSS:**

* When a single page has its unique style.
* For small projects or simple one-page websites.
* When you don't want to create a separate CSS file.

✅ **Good** for small-scale styling but **bad** for large websites (because styles can't be reused across multiple pages).

**38. What is External Stylesheet in CSS? When to use it in real applications?**

**External Stylesheet**: CSS rules are written in a separate file (e.g., style.css) and linked to HTML using a <link> tag.

**Example:**

html

CopyEdit

<link rel="stylesheet" href="style.css">

**style.css:**

css

CopyEdit

body {

background-color: lightgrey;

}

h1 {

color: navy;

}

**When to use External CSS:**

* For medium to large websites.
* When you want the same styling across multiple pages.
* To separate structure (HTML) and design (CSS) properly.
* To make maintenance easier.

✅ **Best practice** for professional websites.

**39. What are CSS Selectors and what are their types?**

**CSS Selectors** are used to **select HTML elements** so you can apply styles to them.

**Types of CSS Selectors:**

| **Selector** | **Syntax** | **Example** | **Meaning** |
| --- | --- | --- | --- |
| **Universal** | \* | \* {} | Selects all elements |
| **Element** | element | p {} | Selects all <p> tags |
| **Class** | .classname | .menu {} | Selects elements with class "menu" |
| **ID** | #idname | #header {} | Selects element with id "header" |
| **Group** | selector1, selector2 | h1, h2 {} | Applies same style to multiple tags |
| **Descendant** | ancestor descendant | div p {} | Selects <p> inside <div> |
| **Child** | parent > child | ul > li {} | Selects direct children only |
| **Pseudo-classes** | selector:pseudo-class | a:hover {} | Selects based on user interaction (e.g., hover, focus) |
| **Attribute** | [attribute=value] | input[type="text"] {} | Selects based on attributes |

✅ Selectors make CSS powerful and flexible.

**40. How do you Include CSS in a webpage or HTML?**

You can include CSS into HTML in three ways:

| **Way** | **How** |
| --- | --- |
| **Inline** | Using style inside any HTML element. |
| **Internal** | Using <style> tag in the <head> section. |
| **External** | Using <link> tag to link a separate CSS file. |

✅ **Best practice**: Always prefer **External CSS** for maintainability.

**41. Explain Box Model in CSS.**

The **CSS Box Model** is how every HTML element is structured and spaced.  
Think of every element as a **box** with layers around it.

The Box Model has **four main parts**:

1. **Content** — The actual text or image inside the element.
2. **Padding** — Space **inside** the border but **around** the content.
3. **Border** — The visible edge around padding and content.
4. **Margin** — Space **outside** the border (separates element from others).

**Visual Diagram:**

lua

CopyEdit

+-------------------------------+

| Margin |

| +-------------------------+ |

| | Border | |

| | +---------------------+ | |

| | | Padding | | |

| | | +-----------------+ | | |

| | | | Content | | | |

| | | +-----------------+ | | |

| | +---------------------+ | |

| +-------------------------+ |

+-------------------------------+

**Example in CSS:**

css

CopyEdit

div {

width: 200px;

padding: 20px;

border: 5px solid black;

margin: 30px;

}

✅ **Total space** taken = width + padding + border + margin.

✅ **Important** for layouts and spacing between elements.

**42. Explain Padding, Margin, and Border.**

In CSS:

| **Term** | **Description** | **Affects** | **Example** |
| --- | --- | --- | --- |
| **Padding** | Space between the content and the element’s border. | Inside the border | padding: 10px; |
| **Border** | A line surrounding the padding and content. | Surrounds padding | border: 2px solid black; |
| **Margin** | Space outside the border, separating elements. | Outside the border | margin: 20px; |

✅ **Simple visual:**

css

CopyEdit

[ Margin ]

[ Border ]

[ Padding ]

[ Content ]

✅ **Example:**

css

CopyEdit

div {

padding: 10px;

border: 2px solid black;

margin: 20px;

}

This means:

* 10px padding inside,
* 2px border,
* 20px space outside the element.

**43. What are the different data types available in JavaScript? Provide examples of each.**

JavaScript has **two main types**: **Primitive** and **Non-Primitive (Objects)**.

**Primitive Data Types:**

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| **Number** | Numeric values | let num = 42; |
| **String** | Text values | let name = "John"; |
| **Boolean** | True or False | let isActive = true; |
| **Undefined** | Variable declared but not assigned | let a; |
| **Null** | Empty or non-existent value | let b = null; |
| **Symbol** | Unique identifiers | let sym = Symbol('id'); |
| **BigInt** | Large integers beyond normal limits | let big = 123456789012345678901234567890n; |

**Non-Primitive (Object) Data Types:**

| **Type** | **Example** |
| --- | --- |
| **Object** | let person = {name: "Alice", age: 30}; |
| **Array** | let colors = ["red", "blue", "green"]; |
| **Function** | function greet() { alert("Hello!"); } |

**44. Explain the difference between var, let, and const. When would you use each?**

| **Keyword** | **Scope** | **Redeclaration** | **Reassignment** | **Hoisting Behavior** | **Best Use** |
| --- | --- | --- | --- | --- | --- |
| **var** | Function scoped | Allowed | Allowed | Hoisted with undefined | Legacy code / older projects |
| **let** | Block scoped | Not allowed | Allowed | Hoisted but not initialized | When you need a variable that will change |
| **const** | Block scoped | Not allowed | Not allowed | Hoisted but not initialized | When you want a constant value |

✅ **Example:**

javascript

CopyEdit

var x = 10;

let y = 20;

const z = 30;

✅ **When to use:**

* **var**: Avoid unless maintaining old code.
* **let**: Use for variables that **might change** (e.g., counters, flags).
* **const**: Use for variables that **should not change** (e.g., configuration, constant references).

**45. What is Hoisting in JavaScript? How does it affect var, let, and const?**

**Hoisting** is JavaScript’s behavior of **moving variable and function declarations to the top** of their scope **before execution**.

✅ **How it affects:**

* **var**: Hoisted and initialized as undefined.
* **let** and **const**: Hoisted but **NOT initialized**. Accessing them before declaration gives a **ReferenceError**.

✅ **Example:**

javascript

CopyEdit

console.log(a); // undefined

var a = 5;

console.log(b); // ReferenceError

let b = 10;

✅ **Important:**

* var → Safe to declare after use but **bad practice**.
* let/const → Must declare **before using**.

**46. What is a callback function in JavaScript? Give an example of how and when to use a callback function.**

✅ **Callback function** is a function passed **as an argument** to another function and is **executed later**.

**When to use:**

* For **asynchronous operations** like API calls, reading files, event handling.

**Example:**

javascript

CopyEdit

function greet(name, callback) {

console.log('Hi ' + name);

callback();

}

function callMe() {

console.log('I am a callback function');

}

greet('John', callMe);

✅ **Explanation**:

* callMe is passed as a callback to greet, which is called after greeting.

**47. What are arrow functions? How do they differ from traditional functions in JavaScript?**

✅ **Arrow Functions** are a shorthand way to define functions, introduced in ES6.

**Syntax:**

javascript

CopyEdit

const add = (a, b) => a + b;

✅ **Differences from Traditional Functions:**

| **Traditional** | **Arrow Function** |
| --- | --- |
| Uses function keyword | Uses => (fat arrow) |
| this keyword refers to its own object | this is inherited from parent |
| More lines of code | Less/clean code |
| Suitable for object methods | Suitable for callbacks, small functions |

✅ **Example:**

javascript

CopyEdit

// Traditional

function sum(a, b) {

return a + b;

}

// Arrow

const sum = (a, b) => a + b;

✅ **Note**:  
Arrow functions **cannot** be used as constructors and **do not** have their own this, arguments, super, or new.target.

**48. Explain the concept of 'callback hell' in JavaScript. Why does it occur, and what problems does it cause?**

✅ **Callback Hell** refers to the situation where callbacks are nested within callbacks many levels deep, making the code **difficult to read, maintain, and debug**.

✅ **Occurs when:**

* You perform **multiple asynchronous operations** one after another.
* Each step depends on the previous one.

✅ **Example:**

javascript

CopyEdit

doTask1(function(result1){

doTask2(result1, function(result2){

doTask3(result2, function(result3){

doTask4(result3, function(result4){

console.log(result4);

});

});

});

});

✅ **Problems caused:**

* Hard to understand ("Pyramid of Doom" shape).
* Difficult to maintain.
* Hard to handle errors properly.

✅ **Solution to callback hell:**

* Use **Promises**.
* Use **async/await** (modern and clean).

**49. What are some common solutions to avoid callback hell in JavaScript? Provide code examples for one solution.**

✅ **Common Solutions:**

* **Using Promises** instead of nested callbacks.
* **Using async/await** to write asynchronous code in a synchronous style.
* **Modularizing code** (breaking large functions into smaller ones).
* **Using named functions** instead of anonymous functions.

✅ **Example using Promises (Solution for Callback Hell):**

function task1() {

return new Promise((resolve) => {

setTimeout(() => {

console.log("Task 1 completed");

resolve();

}, 1000);

});

}

function task2() {

return new Promise((resolve) => {

setTimeout(() => {

console.log("Task 2 completed");

resolve();

}, 1000);

});

}

function task3() {

return new Promise((resolve) => {

setTimeout(() => {

console.log("Task 3 completed");

resolve();

}, 1000);

});

}

// Chaining promises to avoid callback hell

task1()

.then(task2)

.then(task3)

.then(() => {

console.log("All tasks completed");

});

✅ **Explanation:**

* Each task returns a Promise.
* We chain .then() methods instead of nesting multiple callbacks.
* Code becomes cleaner and readable.

**50. Describe the scope of variables in JavaScript.**

How does the scope differ between **var**, **let**, and **const**?

✅ **Scope** refers to **where a variable can be accessed**.

**Types of Scope in JavaScript:**

| **Type** | **Meaning** |
| --- | --- |
| **Global Scope** | Accessible everywhere. |
| **Function Scope** | Accessible inside a function. |
| **Block Scope** | Accessible only inside a block {}. |

✅ **var**:

* **Function scoped**.
* If declared inside a function, accessible only inside that function.
* If declared outside, becomes **global**.

function example() {

var x = 10;

console.log(x); // 10

}

console.log(x); // Error: x is not defined

✅ **let** and **const**:

* **Block scoped** (inside {}).
* Limited to the block, statement, or expression.

{

let a = 20;

const b = 30;

console.log(a, b); // 20 30

}

console.log(a, b); // Error: a is not defined, b is not defined

✅ **Summary Table:**

| **Feature** | **var** | **let** | **const** |
| --- | --- | --- | --- |
| Scope | Function | Block | Block |
| Redeclaration | Allowed | Not allowed | Not allowed |
| Reassignment | Allowed | Allowed | Not allowed |

**51. How does JavaScript handle asynchronous operations?**

Briefly explain the role of the **event loop** in this process.

✅ **JavaScript** is **single-threaded** (one task at a time) but it handles **asynchronous tasks** (e.g., network requests, timers) using:

* **Call Stack**
* **Web APIs** (provided by browser like setTimeout, fetch)
* **Callback Queue**
* **Event Loop**

✅ **Role of Event Loop:**

* Keeps checking if the **call stack is empty**.
* If empty and there are tasks in the **callback queue**, it pushes them to the stack for execution.

✅ **Process:**

1. Normal (synchronous) code runs first.
2. Asynchronous tasks are sent to **Web APIs**.
3. When done, callbacks move to **Callback Queue**.
4. **Event Loop** sends them to **Call Stack** when it’s empty.

✅ **Simple Diagram:**

Code → Call Stack → Web APIs → Callback Queue → Event Loop → Call Stack

✅ **Example:**

console.log('First');

setTimeout(() => {

console.log('Second');

}, 2000);

console.log('Third');

**Output:**

First

Third

Second

✅ **Explanation:**

* First and Third are synchronous.
* Second is delayed by the event loop.

**52. What is the difference between synchronous and asynchronous code in JavaScript? Provide an example of each.**

✅ **Synchronous Code**:

* Code that **executes sequentially**.
* One task must **finish** before the next starts.

✅ **Example:**

console.log('Task 1');

console.log('Task 2');

console.log('Task 3');

**Output:**

Task 1

Task 2

Task 3

✅ **Asynchronous Code**:

* Code that **executes independently** of the main thread.
* Does **not block** the execution of other tasks.

✅ **Example:**

console.log('Task 1');

setTimeout(() => {

console.log('Task 2 (after 2 sec)');

}, 2000);

console.log('Task 3');

**Output:**

Task 1

Task 3

Task 2 (after 2 sec)

✅ **Summary:**

| **Synchronous** | **Asynchronous** |
| --- | --- |
| Blocks execution | Non-blocking |
| Tasks run one by one | Tasks can run in the background |
| Easier to read | Can cause callback hell if not handled properly |