**Interview(Q1-Qxx=>xx Questions)**

**Interview 1st Topic**

**Q1: Introduce yourself.**

**Answer:** Sure! I am Md. Shahbaz Kamal Chowdhury. My journey into programming started back in 2013 when I was in higher secondary school, where I learned basic C and HTML. I really enjoyed programming, which led me to explore it further during my Bachelor's in Electrical and Electronic Engineering. There, I took a course on C, got some exposure to C++, and also worked with MATLAB for various coursework and projects.

After graduating, I worked as an adjunct lecturer before transitioning to my current role as a Data Artist. While these roles waren't directly related to web dvelepments or javascript, I developed an interest in JavaScript and started learning it alongside my job. As I explored it further, I decided to pursue a career related to programming.

Since then, I’ve been focusing on full-stack development, primarily using React with Tailwind CSS for the frontend and Express.js with MongoDB for the backend. I love building scalable and user-friendly applications, and I’m always eager to learn and grow as a developer.

**Q2: Explain Different types of for loop in js.**

**Answer:**

1. **Standard *for* Loop:** The classic *for* loop is used when the number of iterations is known. It consists of initialization, a condition, and an increment/decrement statement.

for (let i = 1; i <= 5; i++) {

console.log(i);

}

/\* Output:

1

2

3

4

5

\*/

1. ***for...in* Loop:** This loop is used to iterate over the properties (keys) of an object. It is not recommended for arrays as it iterates over indices as strings.

const person = { name: "John", age: 25, city: "New York" };

for (let key in person) {

console.log(`${key}: ${person[key]}`);

}

/\* Output:

name: John

age: 25

city: New York

\*/

1. ***for...of* Loop:** The *for...of* loop is used to iterate over iterable objects such as arrays, strings, maps, and sets.

const fruits = ["Apple", "Banana", "Mango"];

for (let fruit of fruits) {

console.log(fruit);

}

/\* Output:

Apple

Banana

Mango

\*/

1. ***while* Loop:** A *while* loop runs as long as the given condition is true. It is useful when the number of iterations is unknown.

let i = 1;

while (i <= 5) {

console.log(i);

i++;

}

/\* Output:

1

2

3

4

5

\*/

1. ***do...while* Loop:** The *do...while* loop is similar to while, but it ensures that the loop runs at least once before checking the condition.

let i = 1;

do {

console.log(i);

i++;

} while (i <= 5);

/\* Output:

1

2

3

4

5

\*/

**Q3: What is the difference between** <div> **and <section>?**

**Answer:**

1. <div> is a generic container with no meaning.
2. <section> groups related thematic content and often has a heading.

<!-- Non-semantic div -->

<div class="blog-section">

<h2>Blog Title</h2>

<p>Blog content...</p>

</div>

<!-- Semantic section -->

<section>

<h2>Blog Title</h2>

<p>Blog content...</p>

</section>

**Q4: How does** <article> **differ from** <section>**?**

**Answer:**

1. <article> is **self-contained** content like a news post or blog entry.
2. <section> groups related content inside a webpage.

<article>

<h2>Breaking News</h2>

<p>This is a news article.</p>

</article>

**HTML 2nd Topic: Attributes (Q5-Q8=>4 questions)**

**Q5: What are attributes in HTML?**

**Answer:** Attributes are additional values that configure the elements or adjust their behavior in various ways to meet the criteria the users want. Attributes provide extra information about an element. They are written inside the opening tag.

<a href="https://example.com" target="\_blank">Visit Example</a>

Here,

1. href sets the link URL.
2. target="\_blank" opens the link in a new tab.

**Q6: What is the difference between id and class attributes?**

**Answer:**

1. id is unique and applies to one element.
2. class can be applied to multiple elements.

<!-- Using id -->

<p id="unique-paragraph">This is a unique paragraph.</p>

<!-- Using class -->

<p class="styled-text">This is a paragraph.</p>

<p class="styled-text">This is another paragraph.</p>

**Q7: What are the alt, title, and style attributes in HTML, and how are they used?**

**Answer:**

✅The alt attribute provides alternative text for images, improving accessibility.

Example:

<img src="image.jpg" alt="A beautiful landscape">

✅The title attribute provides tooltip text when hovering over an element.

Example:

<a href="https://example.com" title="Go to Example Website">Click Here</a>

✅The styleattribute: Adds inline CSS to an element for quick styling.

Example:

<p style="color: blue; font-size: 16px;">This is a styled paragraph.</p>

**Q8: What is the href attribute in HTML, and how is it used?**

**Answer:**

The href (Hypertext Reference) attribute is used in the <a> (anchor) tag to define the URL or link destination. It allows users to navigate to internal or external web pages.

Examples:

* Linking to an external website:

<a href="https://www.google.com">Visit Google</a>

* Linking to an internal page:

<a href="about.html">Go to About Page</a>

* Linking to an email address:

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

* Linking to a phone number (for mobile devices):

<a href="tel:+1234567890">Call Us</a>

**HTML 3rd Topic: HTML Elements (Q9-Q12=>4 questions)**

**Q9: What are an HTML elements,** **and how do they differ from HTML tags?**

**Answer:**

✅An HTML element consists of a start tag, content, and an end tag. Elements define the structure and content of a webpage.

<p>This is a paragraph element.</p>

1. <p> is the opening tag.
2. This is a paragraph element. is the content.
3. </p> is the closing tag.

✅Difference Between an HTML Element and an HTML Tag:

1. **HTML tag** refers to the actual markup inside angle brackets (<>). Example: <h1>.
2. **HTML element** consists of the opening tag, content, and closing tag. Example:

<h1>Hello World</h1>

**Q10: What are block-level and inline elements, and how do <div> and <span> differ?**

**Answer:**

✅ Block-level and inline elements:

1. **Block-level elements** take up the full width (e.g., <div>**,** <p>, <section>).
2. **Inline elements** only take up as much space as needed (e.g., <span>**,** <a>, <strong>).

✅ Difference Between <div> and <span>:

1. <div> is a block-level container.
2. <span> is an inline container used for styling small text portions.

**Q11: What are self-closing (void) elements, and what are some common examples?**

**Answer:**

✅ Self-closing elements, also known as void elements, do not have closing tags because they do not contain any content.

✅ Examples of Self-Closing Elements:

1. <img> → Displays an image
2. <input> → Creates an input field
3. <br> → Inserts a line break
4. <hr> → Adds a horizontal line

<img src="image.jpg" alt="Example image">

<input type="text" placeholder="Enter your name">

<br>

<hr>

**Q12: What are different types of lists in HTML, and how are they used?**

**Answer:**

✅ There are three types of lists in HTML:

1. Ordered List (<ol>) → Items are numbered.
2. Unordered List (<ul>) → Items use bullet points.
3. Description List (<dl>) → Used for key-value pairs.

<!-- Ordered List -->

<ol>

<li>First item</li>

<li>Second item</li>

</ol>

<!-- Unordered List -->

<ul>

<li>Apple</li>

<li>Banana</li>

</ul>

<!-- Description List -->

<dl>

<dt>HTML</dt>

<dd>HyperText Markup Language</dd>

</dl>

✅ When to Use Each List Type?

1. <ol> is used when the order of items matters (e.g., steps in a process).
2. <ul> is used for unordered items (e.g., a shopping list).
3. <dl> is used for descriptions or definitions (e.g., glossary terms).

**HTML 4th Topic: Forms And Inputs (Q13-Q16=>4 questions)**

**Q13: Explain the purpose of HTML forms and how they work, including the onsubmit event.**

**Answer:**

✅ Purpose and How They Work:

HTML forms are used to collect user input. They act as containers for various input elements. When a user submits a form, the collected data is sent to a server for processing. This data is typically used for things like user registration, contact forms, search queries, and e-commerce transactions. The onsubmit event allows you to execute JavaScript code before the form is actually submitted, giving you a chance to validate the data or perform other actions.

<form action="/submit" method="post">

<label for="name">Name:</label>

<input type="text" id="name" name="name">

<button type="submit">Submit</button>

</form>

1. action="/submit" → The URL where form data is sent.
2. method="post" → Defines how the data is sent (GET or POST).

✅ Key Attributes of <form>:

1. action: Specifies the URL where the form data is sent when the form is submitted.
2. method: Defines the HTTP method used to send the data (usually "get" or "post"). "GET" appends the data to the URL, while "POST" sends the data in the request body.
3. target: Specifies where to display the response received after submitting the form (e.g., in a new window or tab).
4. enctype: Specifies how the form data should be encoded when submitting it to the server (e.g., application/x-www-form-urlencoded for standard form data or multipart/form-data for file uploads).
5. onsubmit: Specifies a JavaScript function to be executed when the form is submitted.

**Q14: What are different input types in HTML, and what is their purpose?**

**Answer:**

✅ HTML provides various input types for different user inputs.

**Example of Input Types:**

<input type="text" placeholder="Enter text">

<input type="password" placeholder="Enter password">

<input type="email" placeholder="Enter email">

<input type="number" placeholder="Enter age">

<input type="date">

<input type="file">

<input type="checkbox"> I agree

<input type="radio" name="gender" value="male"> Male

<input type="radio" name="gender" value="female"> Female

<input type="submit" value="Submit">

✅ Key Differences:

|  |  |
| --- | --- |
| **Tag** | **Purpose** |
| text | Single-line text input |
| password | Hidden password input |
| email | Email validation input |
| number | Numeric input with arrows |
| date | Date picker |
| file | File upload |
| checkbox | Select multiple options |
| radio | Select **one** option from a group |
| submit | Submit form data |

**Q15: What are important form-related attributes and tags in HTML?**

**Answer:**

✅ **<label>** Tag → Associates text labels with input fields for better accessibility.

<label for="username">Username:</label>

<input type="text" id="username" name="username">

The for="username" attribute connects the label with the input field (id="username").

✅ **placeholder** Attribute → Displays hint text inside an input field.

<input type="text" placeholder="Enter your name">

✅ **required** Attribute → Ensures the field is filled before submission.

<input type="email" required placeholder="Enter your email">

<button type="submit">Submit</button>

If left empty, the browser prevents form submission.

**Q16: Explain radio buttons and checkboxes with comparisons .What are differences between between <button> and <input type="submit"> and**

**Answer:**

✅ Radio Buttons vs. Checkboxes:

* Radio Buttons (<input type="radio">) → Allow selection of one option.
* Checkboxes (<input type="checkbox">) → Allow selection of multiple options.

<input type="radio" name="gender" value="male"> Male

<input type="radio" name="gender" value="female"> Female

<input type="checkbox" name="subscribe"> Subscribe to newsletter

✅ Difference Between **<button>** and **<input type="submit">**:

* <button> → Can be used inside or outside forms, supports JavaScript interactions.
* <input type="submit"> → Specifically used for form submission.

<form>

<input type="submit" value="Submit">

<button type="submit">Submit</button>

</form>

**HTML 5th Topic: Media (Q17-Q20=>4 questions)**

**Q17: what are the key attributes of the** <img> **tag,and What are the different ways to include images in HTML ?**

**Answer:** In HTML, images are included using the <img> tag, which is self-closing and does not have a closing

✅ **Basic Syntax:**

<img src="image.jpg" alt="Description of image">

📌 Key Attributes of<img>**:**

* **src** (source) → Specifies the image file location (URL or local path).
* **alt** (alternative text) → Provides a text description for accessibility and SEO.
* **width** & **height** → Controls image dimensions.
* **loading** → Lazy loading (**lazy** or **eager**) for performance optimization.
* **title** → Tooltip text when hovering over the image.

Example with Attributes:

<img src="logo.png" alt="Company Logo" width="200" height="100" loading="lazy">

✅ Different Ways to Include Images in HTML:

1. Using Absolute URL (from the web):

<img src="https://example.com/image.jpg" alt="Remote Image">

1. Using a Relative Path (from local storage):

<img src="images/photo.jpg" alt="Local Image">

1. Using CSS Background Image (Alternative to **<img>**):

**CSS Code:**

.background {

background-image: url('image.jpg');

background-size: cover;

}

**HTML Code:**

<div class="background"></div>

📝 Common Interview Follow-Ups(will be added later):

* What is lazy loading (loading="lazy") and how does it optimize performance?
* What are responsive image techniques in HTML?

Q18) How does the alt attribute improve accessibility?

Answer: The alt (alternative text) attribute improves accessibility by providing a textual description of images for visually impaired users who rely on screen readers. It ensures that even if someone cannot see the image, they can understand its purpose through the provided text. Additionally, if an image fails to load due to a broken link or slow connection, the alt text appears in place of the missing image, ensuring users still receive crucial information. It also benefits SEO by helping search engines understand image content. Below are examples demonstrating how the alt attribute enhances accessibility:

✅ Screen Reader Accessibility:

A screen reader will read the alt text aloud, describing the image to visually impaired users.

<img src="dog.jpg" alt="A golden retriever playing with a ball in the park">

✅ Handling Broken Images:

If the image fails to load, the **alt** text will be displayed instead.

<img src="missing-image.jpg" alt="Product image of a black leather jacket">

✅ Decorative Images (Empty **alt**):

If an image is purely decorative and doesn’t add meaningful content, an empty alt="" ensures screen readers ignore it.

<img src="border-pattern.png" alt="">

Q19) What are HTML <audio> and <video> elements?

Answer: HTML provides built-in support for audio and video playback using the <audio> and <video> tags.

✅ Audio Example:

<audio controls>

<source src="music.mp3" type="audio/mpeg">

Your browser does not support the audio element.

</audio>

✅ Video Example:

<video width="640" height="360" controls>

<source src="video.mp4" type="video/mp4">

Your browser does not support the video tag.

</video>

📌 Key Attributes of <audio> and <video>:

* **Controls** → Adds play, pause, and volume controls
* **autoplay** → Starts playing automatically (not recommended for user experience)
* **loop** → Repeats playback continuously
* **muted** → Mutes the audio/video by default
* **preload** → auto, metadata, none (determines how much data is preloaded).

Example with All Attributes:

<video width="640" height="360" controls autoplay loop muted>

<source src="video.mp4" type="video/mp4">

Your browser does not support this video format.

</video>

Q20) What is the <iframe> element, and how can you secure it to prevent security risks like clickjacking?

Answer: An <iframe> is used to embed external content, such as websites, maps, or YouTube videos, within an HTML page.

Example:

<iframe src="https://example.com" width="600" height="400"></iframe>

📌 Common Use Cases:

* Embedding YouTube videos
* Displaying Google Maps
* Integrating third-party widgets

📝 Common Interview Follow-Ups(will be added later):

* What are the security risks of using <iframe>?
* How does the sandbox attribute help improve security?
* Can an <iframe> interact with the parent page’s JavaScript?

**CSS (Q16-Q35=>15 Questions)**

**CSS 1st Topic: Selectors (Q21-Q23=>3 questions)**

**Q21) What are CSS Selectors, and what are the different types? Can you explain with examples?**

Answer: CSS selectors are patterns that define which HTML elements should be styled by CSS rules. They help target specific elements on a webpage and apply styles to them.

There are several types of CSS selectors:

1. Basic Selectors:

* Universal Selector (\*): Selects all elements on the page

\* { margin: 0; padding: 0; }

* Element (Type) Selector: Selects elements by their tag name

p { font-size: 16px; }

* Class Selector (.): Selects elements with a specific class.

.button { background-color: blue; }

* ID Selector (#): Selects an element with a specific ID.

#header { color: red; }

* ID Selector (#): Selects an element with a specific ID.

#header { color: red; }

* Group Selector (A, B, C): Selects multiple elements at once.

h1, h2, h3 { font-weight: bold; }

1. Combinator Selectors:

* Descendant Selector (A B): Selects elements inside another element, regardless of depth.

div p { color: green; }

* Child Selector (A > B): Selects direct children of an element.

div > p { font-weight: bold; }

* Adjacent Sibling Selector (A + B): Selects an element immediately after another.

h1 + p { margin-top: 10px; }

* Adjacent Sibling Selector (A + B): Selects an element immediately after another.

h1 + p { margin-top: 10px; }

* General Sibling Selector (A ~ B): Selects all siblings after a specific element.

h1 ~ p { color: gray; }

1. Attribute Selectors:

* Matches elements with a specific attribute:

input[type="text"] { border: 1px solid black; }

* Pseudo-classes (:hover, :nth-child(n), :focus): Apply styles based on element state.

a:hover { text-decoration: underline; }

li:nth-child(odd) { background-color: lightgray; }

* Pseudo-elements (::before, ::after): Style parts of elements.

p::first-letter { font-size: 2em; }

**Q22) What is the difference between nth-child(n), nth-of-type(n), first-child, and last-child? Provide examples.**

**Answer:** These are **structural pseudo-classes** used to target elements based on their position within the parent.

1. nth-child(n): Selects the nth child of a parent, regardless of element type.

CSS Code:

p:nth-child(2) { color: blue; } /\* Selects second child if it's a <p> \*/

HTML Code:

<div>

<span>Item 1</span>

<p>Item 2</p>

<p>Item 3</p>

</div>

* p:nth-child(2) will select <p>Item 2</p> because it's the second child and the first is a <span>.

1. nth-of-type(n): Selects the nth element of a specific type within its parent.

CSS Code:

p:nth-child(2) { color: blue; } /\* Selects second child if it's a <p> \*/

* For the above HTML, p:nth-of-type(2) will select <p>Item 3</p> because it is the second <p>

1. first-child and last-child :

* :first-child targets the first child, regardless of type.

p:first-child { font-weight: bold; } /\* Only applies if <p> is the first child \*/

* :last-child targets the last child of the parent.

p:last-child { color: red; }

**Q23 What is the difference between em, rem, vw, and vh units in CSS? How do they help in responsive design?**

**Answer:** CSS units define how sizes are calculated in a webpage. The most common units used for responsive design are:

1. **em (Relative to Parent Font Size)**

* 1em is equal to the font size of the parent element.
* If the parent has font-size: 20px, then 1em = 20px.

.parent { font-size: 20px; }

.child { font-size: 1.5em; } /\* 30px (1.5 \* 20px) \*/

1. **rem (Relative to Root Font Size)**

* 1rem is relative to the **root** (<html>**) font size**.
* Default root size is 16px, so 1rem = 16px unless changed.

html { font-size: 10px; }

p { font-size: 2rem; } /\* 20px \*/

1. **vw (Viewport Width) & vh (Viewport Height)**

* vw: 1vw = 1% of the viewport width.
* vh: 1vh = 1% of the viewport height.

div { width: 50vw; } /\* 50% of the screen width \*/

div { height: 100vh; } /\* Full screen height \*/

📝 Common Interview Follow-Ups(will be added later):

* What are the differences between ::before and :before?
* Can nth-child(n) and nth-of-type(n) be used together? If yes, how?
* What is the difference between position: absolute, relative, and fixed in CSS?
* How can you override CSS specificity?

**CSS 2nd Topic: Box Model (Q24-Q26=>3 questions)**

**Q24 What is the CSS Box Model? Can you explain its components and how it affects layout?**

**Answer:** The CSS Box Model is a fundamental concept in CSS that defines how elements are displayed and how spacing is calculated. Every HTML element is treated as a box, consisting of the following components:

1. **Content**

* The actual content inside the element (text, images, etc.).
* Its size is defined by properties like width and height.

div { width: 200px; height: 100px; }

1. **Padding**

* Space between the content and the border.
* Expands the area inside the border but does not affect the element’s position.

div { padding: 20px; } /\* Adds 20px space inside the border \*/

1. **Border**

* A visible outline around the element.
* It adds to the total size of the element.

div { border: 5px solid black; }

1. **Margin:**

* The space **outside** the border, creating distance between elements.
* **Collapses** with adjacent margins (margin collapsing).

div { margin: 15px; } /\* Adds 15px space outside the border \*/

**How Does the Box Model Affect Layout?**

The total width and height of an element is calculated as:

Total Width = Content Width + Padding + Border + Margin

Total Height = Content Height + Padding + Border + Margin

For example:

div {

width: 200px;

height: 100px;

padding: 10px;

border: 5px solid black;

margin: 20px;

}

Total width = 200px (content) + 10px \* 2 (padding) + 5px \* 2 (border) = 230px

Total height = 100px + 10px \* 2 + 5px \* 2 = 130px

**Q25 What is box-sizing in CSS? How does it affect the Box Model?**

**Answer:** The box-sizing property controls how width and height are calculated in the CSS Box Model. It has two values:

1. **content-box (Default Box Model)**

* width and height only include the content.
* Padding and border are added separately, increasing the element's actual size.

div {

width: 200px; /\* Content width only \*/

padding: 10px;

border: 5px solid black;

box-sizing: content-box;

}

Total width = 200px + 10px \* 2 (padding) + 5px \* 2 (border) = 230px

1. **border-box (Better for Responsive Design)**

* width and height **include padding and border**.
* The actual size remains fixed.

div {

width: 200px;

padding: 10px;

border: 5px solid black;

box-sizing: border-box;

}

Total width remains 200px (padding and border are included).

Why Use border-box?

* Prevents unwanted layout shifts.
* Makes width calculation easier, useful in responsive design.

\* {

box-sizing: border-box; /\* Recommended for consistent layouts \*/

}

**Q26) What is Margin Collapsing? When does it happen, and how can you prevent it?**

**Answer:** Margin Collapsing occurs when vertical margins of adjacent elements merge into a single margin instead of adding up.

**When Does Margin Collapsing Happen?**

1. Between Adjacent Elements:

div { margin-bottom: 20px; }

p { margin-top: 30px; }

Instead of 20px + 30px = 50px, the margin becomes 30px (largest margin wins).

1. Parent and First/Last Child:

div { margin: 20px; }

p { margin-top: 20px; }

Instead of 20px + 20px = 40px, the margin remains 20px.

1. Empty Elements:

* If an element has no border, padding, or content, its margins may collapse.

**How to Prevent Margin Collapsing?**

* Use padding or border to separate margins.

div { padding-bottom: 1px; }

* Use display: flex; (Flexbox does not collapse margins).
* Use overflow: hidden; on the parent element.

📝 Common Interview Follow-Ups(will be added later):

* What is the difference between padding and margin, and when should you use each?
* Can the Box Model be applied to inline elements? If yes, how?
* How does display: inline-block; affect the Box Model?
* What are the different types of borders in CSS, and how do they interact with the Box Model?

**CSS 3rd Topic: Positioning And Layout (Q27-Q29=>3 questions)**

**Q27) What are the different CSS position values, and how do they affect element placement? Can you explain with examples?**

**Answer:** CSS provides five different position values, which define how an element is placed in the document flow.

1. **static (Default Positioning)**

* Elements are positioned according to the normal document flow.
* No effect when using top, left, right, or bottom.

div {

position: static; /\* Default \*/

top: 20px; /\* Has no effect \*/

}

1. **relative (Relative to Itself)**

* The element remains in the normal document flow but can be moved using top, left, right, or bottom.
* Space is still reserved for the element in the document.

div {

position: relative;

top: 20px; /\* Moves 20px down \*/

left: 10px; /\* Moves 10px right \*/

}

1. **absolute (Relative to the Nearest Positioned Ancestor)**

* The element is **removed** from the normal document flow.
* Positioned relative to the nearest relative, absolute, or fixed ancestor (if none, then relative to <html>).

.parent {

position: relative;

}

.child {

position: absolute;

top: 50px;

left: 50px;

}

Here, .child will be positioned relative to .parent, not the document.

1. **fixed (Relative to the Viewport)**

* The element is removed from the document flow.
* Positioned relative to the viewport (browser window).

.fixed-header {

position: fixed;

top: 0;

width: 100%;

background: black;

}

A fixed header remains at the top even when scrolling.

1. **sticky (Hybrid of relative and fixed)**

* The element behaves like relative **until** a certain scroll position is reached, then behaves like fixed.

.sticky-menu {

position: sticky;

top: 10px;

}

This makes the element stick to the top when scrolling past it.

**Q28) What is Flexbox, and how does it help in layout design? Can you explain important properties?**

**Answer:** Flexbox (Flexible Box Layout) is a layout model designed to arrange elements in a single row or column efficiently, even if their sizes are dynamic. It simplifies alignment, spacing, and distribution.

**Key Flexbox Properties:**

1. **Container Properties (display: flex;):**

* flex-direction: Defines the main axis (row or column).

.container {

display: flex;

flex-direction: row; /\* Default: row \*/

}

* justify-content: Aligns items along the main axis.

.container {

justify-content: space-between; /\* Even spacing between items \*/

}

* align-items: Aligns items along the cross-axis.

.container {

align-items: center; /\* Centers items vertically \*/

}

1. **Item Properties (For Child Elements):**

* flex-grow: Defines how much an item should expand.

.item {

flex-grow: 1; /\* Equal space for all items \*/

}

* flex-shrink: Defines how much an item should shrink.
* align-self: Aligns individual items differently than others.

**Examples:** This creates a horizontally and vertically centered layout.

.container {

display: flex;

flex-direction: row;

justify-content: center;

align-items: center;

}

Why Use Flexbox?

✅Automatically adjusts elements for different screen sizes.  
✅Replaces float and inline-block for better alignment.  
✅Simplifies responsive web design.

**Q29) What is CSS Grid, and how does it compare to Flexbox? When should you use Grid instead of Flexbox?**

**Answer:** CSS Grid is a two-dimensional layout system, unlike Flexbox, which is one-dimensional. Grid is best for designing entire page layouts, while Flexbox is better for arranging items in a single row or column..

**Key Grid Properties:**

* display: grid; - Enables Grid layout.
* grid-template-columns - Defines column structure.
* grid-template-rows - Defines row structure.
* gap - Adds spacing between grid items.

**Examples:** This creates a **3-column layout** with two rows.

.container {

display: grid;

grid-template-columns: repeat(3, 1fr); /\* 3 equal columns \*/

grid-template-rows: 100px 200px; /\* Two rows with different heights \*/

gap: 10px;

}

**Grid vs. Flexbox: When to Use What?**

|  |  |  |
| --- | --- | --- |
| **Feature** | Flexbox | Grid |
| Layout | 1D (Row/Column) | 2D (Rows & Columns) |
| Alignment | Easier for single lines | Best for complex layouts |
| Use Case | |  | | --- | |  |  |  | | --- | | Navigation bars, cards, forms | | Whole page structure |

If you need a **structured layout (e.g., a dashboard)**, use **CSS Grid**. If you need a **fluid, row-based design**, use **Flexbox**.

📝 Common Interview Follow-Ups(will be added later):

* What is the difference between absolute, relative, and fixed positioning in real-world usage?
* How does z-index work in CSS, and when should it be used?
* Can you explain the difference between inline, block, and inline-block elements in CSS?
* How can Flexbox and Grid be used together in a responsive layout?

**CSS 4th topic: Responsive Design (Q30-Q32=>3 questions)**

**Q30) What is responsive web design, and how is it implemented in CSS?**

**Answer:** Responsive web design ensures that a website adjusts smoothly across different screen sizes and devices, improving user experience on desktops, tablets, and mobile phones. In CSS, responsiveness is implemented using flexible layouts, media queries, and responsive units. A key technique is using relative units like percentages (%), viewport units (vw, vh), and flexible grid systems instead of fixed pixels. Another important method is using media queries, which apply specific styles based on the screen width.

**For example,** the following media query changes the layout when the screen width is 768px or smaller:

@media (max-width: 768px) {

body {

font-size: 14px;

}

}

Responsive images are also essential, ensuring they scale within their containers using max-width: 100% and height: auto. Another powerful approach is using Flexbox and CSS Grid for layout structures that automatically adjust based on available screen space.

**Q31) What are media queries, and how do they work?**

**Answer:** Media queries are a CSS feature that allows developers to apply different styles based on device properties such as screen width, resolution, and orientation. They help create responsive designs by adjusting the appearance of elements for different screen sizes. The basic syntax of a media query consists of the @media rule followed by conditions that define when the styles should be applied.

**For example**, the following media query changes the background color when the screen width is 600px or smaller:

@media (max-width: 600px) {

body {

background-color: lightgray;

}

}

Media queries can also be combined using logical operators such as and, or, and not. A practical example is modifying the layout based on both screen width and device orientation:

@media (max-width: 768px) and (orientation: portrait) {

.container {

display: block;

}

}

Advanced media queries allow targeting high-resolution (Retina) displays using min-resolution: 2dppx or enabling dark mode detection using prefers-color-scheme: dark. These techniques make web pages more adaptable to modern devices and user preferences.

**Q32) How do you make images and videos responsive in CSS?**

Answer: Making images and videos responsive ensures they scale properly across different screen sizes without overflowing their containers. The most common approach for images is using max-width: 100% and height: auto, which prevents distortion while allowing the image to shrink within its parent container.

For example:

img {

max-width: 100%;

height: auto;

display: block;

}

For videos, the issue is that embedded <iframe> or <video> elements often have fixed dimensions. A common solution is wrapping them inside a responsive container using the aspect-ratio property or CSS tricks like padding hacks. The following example ensures an embedded YouTube video remains responsive while maintaining a 16:9 aspect ratio:

.video-container {

position: relative;

width: 100%;

padding-top: 56.25%; /\* 16:9 aspect ratio \*/

}

.video-container iframe {

position: absolute;

top: 0;

left: 0;

width: 100%;

height: 100%;

}

Alternatively, using the aspect-ratio property in modern browsers simplifies this:

.video-container {

aspect-ratio: 16 / 9;

width: 100%;

}

These techniques ensure media elements adjust dynamically across different devices without breaking the layout, improving accessibility and user experience.

📝 Common Interview Follow-Ups(will be added later):

* What is the difference between em, rem, %, vw, and vh in responsive design?
* How does the viewport meta tag help in making websites responsive?
* What is clamp() in CSS, and how does it improve responsive typography?
* How does min-width and max-width affect media queries?
* What are the differences between relative, absolute, fixed, and sticky positioning in responsive layouts?
* What are some common challenges in responsive design, and how do you solve them?
* Can you explain how aspect-ratio in CSS helps with responsive images and videos?
* How does flex-wrap work in Flexbox, and when should it be used?

CSS 5th topic: Styling (Q33-Q35=>3 questions)

**Q33) What is the difference between inline, internal, and external CSS?**

Answer: CSS can be applied to HTML elements in three ways: inline, internal, and external styles.

Inline CSS is applied directly to an HTML element using the style attribute. It has the highest specificity but is not recommended for maintainability. Example:

<p style="color: blue; font-size: 16px;">This is an inline-styled paragraph.</p>

Internal CSS is placed inside a <style> tag within the <head> section of an HTML file. It is useful for applying styles to a single page. Example:

<head>

<style>

p {

color: blue;

font-size: 16px;

}

</style>

</head>

External CSS is written in a separate .css file and linked using the <link> tag. This is the best practice for large projects as it promotes reusability and clean code. Example:

<head>

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

</head>

Using external CSS ensures a structured approach, making it easier to manage styles across multiple pages. However, inline CSS should be avoided unless necessary for quick styling adjustments.

**Q34) What is the difference between absolute, relative, and fixed units in CSS?**

Answer: CSS uses both absolute and relative units to define element sizes, spacing, and positioning.

Absolute units (px, cm, mm, in, pt) have a fixed size that does not change based on the screen size or parent elements. For example:

p {

font-size: 16px;

}

Here, the font size remains 16 pixels regardless of the parent element or screen width.

Relative units (em, rem, %, vw, vh) are flexible and adapt based on the context. For example:

p {

font-size: 1.5em; /\* 1.5 times the size of the parent element \*/

}

Using em makes the font size relative to its parent, while rem is relative to the root element (html).

Viewport-based units (vw, vh) are relative to the screen size. For example:

.container {

width: 80vw; /\* 80% of the viewport width \*/

height: 50vh; /\* 50% of the viewport height \*/

}

Using relative units enhances responsiveness and makes layouts more flexible across different devices.

**Q35) How does the z-index property work in CSS?**

Answer: The z-index property in CSS controls the stacking order of elements along the z-axis (depth). Elements with a higher z-index appear in front of those with a lower value. However, z-index only works on elements with a position value other than static (e.g., relative, absolute, fixed, or sticky).

For example, in the following code, the red box appears in front of the blue box due to a higher z-index:

.box1 {

width: 100px;

height: 100px;

background-color: blue;

position: absolute;

top: 50px;

left: 50px;

z-index: 1;

}

.box2 {

width: 100px;

height: 100px;

background-color: red;

position: absolute;

top: 70px;

left: 70px;

z-index: 2;

}

If two elements have the same z-index, the one appearing later in the HTML code will be placed on top. A common mistake is trying to use z-index without setting a position, which prevents it from working.

📝 Common Interview Follow-Ups(will be added later):

* What is the difference between em and rem, and when should you use each?
* How does opacity differ from visibility: hidden and display: none?
* What is the difference between auto, inherit, and initial in CSS properties?
* How does the position property (static, relative, absolute, fixed, sticky) affect element behavior?
* What is the difference between min-width, max-width, and width?
* How does box-shadow work, and how can you create a 3D effect using it?
* How can CSS variables (--var-name) improve styling in a project?
* What is the difference between hover, focus, and active pseudo-classes?

**Javascript (Q36-Q60=>25 Questions)**

**Javascript 1st Topic: DOM Manipulation (Q36-Q40=>5 questions)**

**Q36) What is DOM in JavaScript?**

Answer: The Document Object Model (DOM) is an interface that allows programs to manipulate the structure, style, and content of web documents. It represents the page so that programs can change the document structure, style, and content. The DOM is a tree-like structure where each element is an object that can be accessed and manipulated using JavaScript. It provides a structured way to interact with HTML and XML documents and allows dynamic updates to the content of a webpage.

**Q37) How can you select an element from the DOM using JavaScript?**

Answer: To select an element from the DOM in JavaScript, you can use various methods. The most common ones are getElementById(), getElementsByClassName(), getElementsByTagName(), querySelector(), and querySelectorAll().

For example:

let element = document.getElementById('myElement');

getElementById() selects an element with a specific ID, returning a single element. On the other hand, querySelector() allows selecting an element using CSS selectors. If you need to select multiple elements, you can use querySelectorAll() or getElementsByClassName().

**Q38) What are the different ways to manipulate DOM elements using JavaScript?**

Answer: There are multiple ways to manipulate DOM elements in JavaScript. You can change the content, modify styles, add or remove classes, and even change attributes. Here are a few examples:

1. Changing Content: you can change the inner content of an element using innerHTML or textContent.

document.getElementById('myElement').innerHTML = 'New Content';

1. Changing Styles: You can modify an element's style using the style property.

document.getElementById('myElement').style.color = 'blue';

1. Adding or Removing Classes: You can add or remove classes from an element using classList..

document.getElementById('myElement').classList.add('active');

document.getElementById('myElement').classList.remove('inactive');

1. Manipulating Attributes: You can change an element's attributes using setAttribute() and getAttribute().

document.getElementById('myElement').setAttribute('src', 'image.jpg');

**Q39) What is the difference between innerHTML and textContent?**

Answer: innerHTML and textContent are both used to get or set the content of an element, but there are key differences:

* innerHTML: This property returns or sets the HTML content of an element. If used to set the content, it can inject HTML elements, including scripts and other tags, into the element. It can be a security risk if you're working with untrusted content (e.g., XSS attacks).

document.getElementById('myElement').innerHTML = '<b>Bold Text</b>';

* textContent: This property returns or sets the text content of an element, ignoring any HTML tags within the element. It is generally safer when you're only working with text and don’t want to risk injecting malicious code.

document.getElementById('myElement').textContent = 'This is plain text';

**Q40) How can you add a new element to the DOM?**

**Answer:** You can add a new element to the DOM by creating a new element and appending it to an existing element. First, use document.createElement() to create the new element, and then use methods like appendChild() or insertBefore() to add it to the document.

Examples:

let newElement = document.createElement('div');

newElement.innerHTML = 'New Element Content';

document.body.appendChild(newElement);

Here, a new <div> element is created and its content is set. The appendChild() method then adds it as the last child of the <body> element.

📝 Common Interview Follow-Ups(will be added later):

* How can you handle events using JavaScript in the DOM?
* Can you explain event delegation in JavaScript?
* What is the purpose of createElement() and how is it used?
* What is the classList property in JavaScript, and how can it be used for DOM manipulation?
* How would you remove an element from the DOM?
* What is the difference between parentNode and parentElement in DOM manipulation?
* Can you explain how to use querySelector() and querySelectorAll() to select multiple elements?
* What are the performance considerations when manipulating large DOM elements?

**Javascript 2nd Topic: Control Flow (Q41-Q45=>5 questions)**

**Q41) What is the role of if-else statements in JavaScript?**

Answer: In JavaScript, if-else statements are used to control the flow of execution based on specific conditions. The condition within the if block is evaluated, and if it is true, the code within the if block runs. If the condition is false, the else block, if present, gets executed. It is commonly used to make decisions in the code based on different conditions. Example:

let age = 18;

if (age >= 18) {

console.log('You are an adult.');

} else {

console.log('You are a minor.');

}

In this example, the condition checks if age is greater than or equal to 18. If it is, it logs "You are an adult." If not, it logs "You are a minor."

**Q42) What is a switch statement in JavaScript and how does it work?**

Answer: A switch statement in JavaScript provides a way to perform multiple comparisons based on the value of a variable. Unlike multiple if-else conditions, it checks a variable against several possible values and executes the corresponding block of code when a match is found. If no match is found, the default case, if provided, will run.

Example:

let fruit = 'apple';

switch (fruit) {

case 'apple':

console.log('Apple selected');

break;

case 'banana':

console.log('Banana selected');

break;

default:

console.log('Unknown fruit');

}

Here, the variable fruit is checked against the values in each case. If it matches 'apple', the code logs "Apple selected." If no case matches, the default block is executed.

**Q43) What are the differences between for, while, and do-while loops in Js?**

Answer: JavaScript provides three types of loops for repeating a block of code: for, while, and do-while.

1. for Loop : A for loop is typically used when the number of iterations is known beforehand. It consists of three parts: initialization, condition, and increment/decrement.

for (let i = 0; i < 5; i++) {

console.log(i);

}

1. while Loop : A while loop runs as long as the condition is true. It checks the condition before executing the code block, so if the condition is false initially, the code inside will not run.

let i = 0;

while (i < 5) {

console.log(i);

i++;

}

1. do-while Loop : A do-while loop executes the code block at least once, even if the condition is false, because it checks the condition after running the code.

let i = 0;

do {

console.log(i);

i++;

} while (i < 5);

**Q44) How do break and continue work in JavaScript loops?**

Answer: break and continue are used to control the flow of loops.

1. break : The break statement immediately exits the loop, regardless of the condition. This is useful when you want to terminate a loop early.

for (let i = 0; i < 10; i++) {

if (i === 5) break; // Exits the loop when i equals 5

console.log(i);

}

1. continue : The continue statement skips the current iteration and moves to the next one. It’s useful when you want to skip specific iterations based on a condition.

for (let i = 0; i < 10; i++) {

if (i === 5) continue; // Skips the iteration when i equals 5

console.log(i);

}

**Q45) What is the purpose of the return statement in JavaScript functions?**

Answer: The return statement in JavaScript is used to exit from a function and optionally send a value back to the caller. When a function reaches the return statement, it stops execution and returns the specified value. If no value is provided, it returns undefined.

Example:

function add(a, b) {

return a + b; // Returns the sum of a and b

}

let result = add(2, 3);

console.log(result); // Logs 5

In this example, the add function returns the sum of a and b. The value is then assigned to the variable result.

📝 Common Interview Follow-Ups(will be added later):

* Can you explain the concept of "fall-through" in a switch statement?
* What happens if there’s no break statement in a switch case?
* How do you avoid infinite loops in JavaScript?
* Can you use return in an arrow function, and how does it behave differently from a regular function?
* How would you handle multiple conditions with a switch statement?
* What is a labeled loop in JavaScript, and when might it be used?
* How do forEach loops differ from traditional for loops in JavaScript?
* How can you handle errors within loops in JavaScript?

**Javascript 3rd Topic: ES6 Features (Q46-Q53=>8 questions)**

**Q46) What are arrow functions in JavaScript?**

**Answer:** Arrow functions, introduced in ES6, provide a more concise syntax for writing functions. The syntax removes the need for the function keyword and uses the => syntax instead. They are often used for short functions or callbacks. One of the key differences from regular functions is that arrow functions do not have their own this value; instead, they inherit this from the surrounding context. **Example:**

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

console.log(add(2, 3)); // Outputs 5

In this example, the add function takes two parameters and returns their sum. The syntax is more compact than the traditional function declaration.

**Q47) What is the let keyword in JavaScript?**

**Answer** :The let keyword, introduced in ES6, is used to declare block-scoped variables, as opposed to var, which declares variables globally or function-scoped. This makes let more predictable when working with loops or conditional blocks, as its scope is limited to the block in which it is defined. **Example:**

let x = 10;

if (true) {

let x = 20;

console.log(x); // Outputs 20

}

console.log(x); // Outputs 10

Here, the value of x inside the if block is different from the value outside, demonstrating the block scope of let.

**Q48) What are template literals in JavaScript?**

**Answer:** Template literals, introduced in ES6, allow for easier string interpolation. They use backticks (`) instead of quotes and enable the inclusion of expressions inside strings using ${expression}. This feature simplifies the creation of strings that involve dynamic values. **Example:**

const name = 'John';

const greeting = `Hello, ${name}!`;

console.log(greeting); // Outputs "Hello, John!"

In this example, the value of name is dynamically inserted into the string using template literals.

**Q49) What are default parameters in JavaScript?**

**Answer:** Default parameters in ES6 allow you to assign default values to function parameters if they are not provided when the function is called. This makes it easier to handle optional parameters without needing to check if they are undefined. **Example:**

function greet(name = 'Guest') {

console.log(`Hello, ${name}`);

}

greet(); // Outputs "Hello, Guest"

greet('Alice'); // Outputs "Hello, Alice"

In this example, the name parameter has a default value of 'Guest', which is used if no argument is passed to the function.

**Q50) What is destructuring in JavaScript?**

**Answer:** Destructuring, introduced in ES6, allows you to unpack values from arrays or objects into variables in a more convenient and concise way. It makes it easier to extract multiple properties from an object or elements from an array in a single statement. **Examples:**

* **Object Destructuring:**

const person = { name: 'John', age: 30 };

const { name, age } = person;

console.log(name, age); // Outputs "John 30"

* **Array Destructuring:**

const numbers = [1, 2, 3];

const [a, b] = numbers;

console.log(a, b); // Outputs "1 2"

In these examples, the values of name and age are extracted from the person object, and the values 1 and 2 are extracted from the numbers array.

**Q51) What are Promise objects in JavaScript?**

**Answer:** A Promise in JavaScript represents an asynchronous operation that can be either resolved or rejected. Promises are used to handle asynchronous operations, allowing you to handle successful or failed results with then() and catch() methods. They simplify asynchronous code and make it more readable. **Example:**

const myPromise = new Promise((resolve, reject) => {

let success = true;

if (success) {

resolve('Operation successful');

} else {

reject('Operation failed');

}

});

myPromise

.then(result => console.log(result)) // Outputs "Operation successful"

.catch(error => console.log(error)); // Outputs if rejected

In this example, the Promise resolves with a success message and handles it using then(). If it fails, it is caught by catch().

**Q52) What are classes in JavaScript?**

**Answer:** ES6 introduced classes to JavaScript, providing a more structured way of creating objects and handling inheritance. Classes are syntactical sugar over the existing prototype-based inheritance and allow for a more object-oriented approach to defining objects and methods.**Examples:**

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

greet() {

console.log(`Hello, my name is ${this.name}`);

}

}

const john = new Person('John', 30);

john.greet(); // Outputs "Hello, my name is John"

In this example, a Person class is defined with a constructor to initialize name and age, and a method greet() to print a greeting.

**Q53) What is the spread operator in JavaScript?**

**Answer:** The spread operator (...) in JavaScript allows you to expand elements from an iterable (like an array or object) into individual elements. It simplifies copying, merging, or combining arrays and objects..**Example:**

* **Array Spread:**

const arr1 = [1, 2, 3];

const arr2 = [...arr1, 4, 5];

console.log(arr2); // Outputs [1, 2, 3, 4, 5]

* **Object Spread:**

const obj1 = { name: 'John', age: 30 };

const obj2 = { ...obj1, city: 'New York' };

console.log(obj2); // Outputs { name: 'John', age: 30, city: 'New York' }

In these examples, the spread operator is used to copy and merge arrays and objects.

📝 Common Interview Follow-Ups(will be added later):

* How does the const keyword differ from let in JavaScript?
* Can you explain the difference between the call(), apply(), and bind() methods in JavaScript?
* What are the benefits of using arrow functions over regular functions?
* How do you handle asynchronous operations without using Promises in JavaScript?
* What are getter and setter methods in JavaScript classes?
* Can you explain the concept of "hoisting" in JavaScript?
* How do async and await work in JavaScript? What’s their relation to Promises?
* Can you explain the concept of "modules" in JavaScript ES6?

**Javascript 4th Topic: APIs (Q54-Q60=>7 questions)**

**Q54) What is the Fetch API in JavaScript?**

**Answer:** The Fetch API provides a modern way to make asynchronous HTTP requests in JavaScript. It returns a Promise that resolves to the Response object representing the response to the request. The Fetch API is a more flexible and powerful replacement for the older XMLHttpRequest. You can use fetch() to send GET, POST, and other HTTP requests to retrieve or send data from a server. **Example:**

fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.log('Error:', error));

In this example, fetch() is used to send a GET request. The response is converted to JSON format, and the data is logged to the console. Any errors are caught and logged as well.

**Q55) What is the difference between fetch() and XMLHttpRequest?**

**Answer:** The main difference between fetch() and XMLHttpRequest is that fetch() is promise-based, making it easier to work with asynchronous code, while XMLHttpRequest uses callbacks. This makes fetch() more modern, cleaner, and easier to read and maintain.

* **Promises**: With fetch(), you can use .then() and .catch() for handling success and failure, while XMLHttpRequest uses callback functions.
* **Streams**: fetch() supports streams, allowing you to handle large responses as they are received, whereas XMLHttpRequest doesn’t have this feature.
* **No Built-In Support for JSON Parsing**: fetch() requires you to manually parse JSON responses, while XMLHttpRequest parses responses based on their content type.

**Example with fetch():**

fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.log('Error:', error));

**Q56) How does the localStorage API work in JavaScript?**

**Answer** The localStorage API allows you to store key-value pairs in the browser's storage, which persists even after the page is reloaded or the browser is closed. It is synchronous and can store data as strings. You can use localStorage to save user preferences, settings, or other information that should be retained across sessions.**Example:**

localStorage.setItem('username', 'JohnDoe');

let username = localStorage.getItem('username');

console.log(username); // Outputs "JohnDoe"

In this example, the setItem() method stores a value, and the getItem() method retrieves the value. Data stored in localStorage will persist until explicitly removed with removeItem() or cleared using clear().

**Q57) What is the sessionStorage API in JavaScript?**

**Answer:** sessionStorage is similar to localStorage, but with one key difference: it only persists data for the duration of the page session. A session ends when the browser or tab is closed. It is typically used for storing temporary data that only needs to be available during a single session.**Example:**

sessionStorage.setItem('theme', 'dark');

let theme = sessionStorage.getItem('theme');

console.log(theme); // Outputs "dark"

In this example, the setItem() method stores a value in the session storage, and the getItem() method retrieves it. The data will be cleared when the session ends (i.e., when the tab is closed).

**Q58) What is the Geolocation API in JavaScript?**

**Answer:** The Geolocation API allows websites to access the geographical location of a user's device. It is commonly used for applications like maps or location-based services. The API provides methods to get the user's current position or watch for changes in their position..**Example:**

navigator.geolocation.getCurrentPosition(function(position) {

console.log('Latitude: ' + position.coords.latitude);

console.log('Longitude: ' + position.coords.longitude);

});

In this example, getCurrentPosition() is used to retrieve the user's current geographical coordinates. The position object contains the coords property with latitude and longitude.

**Q59) How does the Web Storage API differ from cookies in JavaScript?**

Answer: The Web Storage API (which includes localStorage and sessionStorage) differs from cookies in several ways:

* Storage Size: Web Storage can store larger amounts of data (up to 5-10MB per domain) compared to cookies, which are limited to around 4KB.
* Lifetime: localStorage persists data across sessions, while cookies can have expiration dates set by the server.
* Data Handling: Cookies are sent with every HTTP request, while data in Web Storage is stored on the client side and not transmitted with requests, improving performance.
* Simplicity: Web Storage is easier to use and more efficient for client-side storage compared to cookies.

**Q60) What is the Notification API in JavaScript?**

**Answer:** The Notification API allows web pages to display notifications to the user, even if the page is not in the foreground. This API is often used in conjunction with service workers to send push notifications for real-time updates, such as messages or alerts...**Example:**

if (Notification.permission === 'granted') {

new Notification('Hello, you have a new message!');

} else {

Notification.requestPermission().then(permission => {

if (permission === 'granted') {

new Notification('Hello, you have a new message!');

}

});

}

In this example, the Notification object is used to display a notification. Before sending a notification, the browser must request permission to show notifications.

📝 Common Interview Follow-Ups(will be added later):

* How would you handle errors in the Fetch API?
* What are some common use cases for the localStorage and sessionStorage APIs?
* How do you ensure compatibility for the Geolocation API across different browsers?
* How do you handle JSON responses from APIs using the Fetch API?
* What are some security concerns when using localStorage and sessionStorage?
* How can you set expiration for localStorage data?
* How would you handle sending data through the Fetch API using POST requests?
* How can you use the Notification API to send push notifications with service workers?