1. Semantic Tag use

Semantic tags like `<header>`, `<article>`, and `<footer>` in HTML enhance SEO by providing structured information to search engines about the content's hierarchy and relevance. They also improve accessibility by helping assistive technologies interpret and navigate web pages more effectively.

2. Attributes in HTML

Attributes in HTML provide additional information about elements and are specified within their opening tags. They control element behavior, appearance (via CSS classes), define links and image sources, handle events, and support custom data storage (`data-` attributes) for enhanced functionality and styling.

3. Ways to Write CSS

i) External CSS: Allows for separation of concerns, making it easier to manage and maintain styles across multiple pages. It promotes reusability and scalability by centralizing styles in one or more CSS files that can be cached by the browser, leading to faster page load times.

ii) Internal CSS: Useful for small projects or situations where styles are specific to a single HTML file. It provides more flexibility than inline CSS and keeps styles within the HTML document, but it doesn't promote reuse or separation of concerns as effectively as external CSS.

iii) Inline CSS: Should generally be avoided for larger projects because it mixes style information with content, making it harder to maintain and update styles across multiple elements or pages. However, it can be useful for quick styling changes on individual elements or in certain dynamically generated content scenarios.

- Priority: Inline > Internal > External

4. Difference between Inline and Block Elements

i) Inline Elements: Flow inline with surrounding content, ignoring explicit width or height settings. Examples: `<span>`, `<a>`, `<strong>`, `<em>`, `<img>`, `<input>`, `<button>`.

ii) Block Elements: Start on a new line and take up the full width available, can have width and height specified, and can contain both inline and block-level elements. Examples: `<div>`, `<p>`, `<h1>` to `<h6>`, `<ul>`, `<ol>`, `<li>`, `<form>`.

Key Differences:

- Layout: Inline elements flow inline with surrounding content, while block elements start on a new line and take up the full width available.

- Width and Height: Inline elements typically do not have width and height properties applied directly to them, whereas block elements can have both width and height specified.

- Margins and Padding: Inline elements respect horizontal margins and padding but ignore vertical margins. Block elements can have margins, padding, and borders applied to all sides.

- Content Model: Inline elements cannot contain block-level elements inside them, while block elements can contain both inline and block-level elements.

CSS

1. Cascading in CSS

Cascading in CSS means combining and prioritizing multiple style rules to determine how elements are styled on a webpage, considering specificity, order, and inheritance.

2. Specificity in CSS

Specificity in CSS determines which style rule is applied to an element when multiple conflicting rules exist, based on selectors' hierarchy:

- Inline styles have the highest specificity.

- ID selectors have higher specificity than classes or attributes.

- Elements and pseudo-elements have the lowest specificity.

3. Default value of `position` property: Static

4. Difference between `position: absolute` and `position: relative`

i) Flow: Relative elements remain in the normal flow of the document, affecting other elements' positions, while absolute elements are taken out of the normal flow and do not affect other elements' positions.

ii) Offset: Relative elements can be offset from their normal position using `top`, `right`, `bottom`, `left` properties, whereas absolute elements are positioned relative to their nearest positioned ancestor.

iii) Use Cases: Use relative positioning for minor adjustments to element positions while keeping them in the flow. Use absolute positioning for precise positioning of elements within their containing elements.

Javascript

1. Difference between `let`, `var`, and `const`

- var: Function-scoped, hoisted to the top of the function, can be redeclared and reassigned.

- let: Block-scoped, not hoisted (in TDZ until declared), can be reassigned but not redeclared in the same scope.

- const: Block-scoped, not hoisted (in TDZ until declared), cannot be reassigned or redeclared after initialization.

Category	var	let	const
Scope	Functional or Global scope	Block	Block
Update and Declaration	Updated and re-declared in the same scope.	Updated but cannot be re-declared in the same scope.	Neither be updated or re-declared in any scope.
Intialization	Can be declared without initialization	Can be declared without initialization	Cannot be declared without initialization
Access	Can be accessed without initialization as its default value is “undefined”	Cannot be accessed without initialization otherwise it will give ‘referenceError’	Cannot be accessed without initialization, as it cannot be declared without initialization.
Hoist	These variables are hoisted.	These variables are hoisted but stay in the temporal dead zone untill the initialization	These variables are hoisted but stays in the temporal dead zone until the initialization.

2. Promises

Promises in JavaScript represent the eventual completion or failure of an asynchronous operation, allowing handling of its result asynchronously through chained `.then()` and `.catch()` methods. They simplify callback-based code, providing a more readable and manageable way to work with asynchronous tasks.

3. Difference between return value of `querySelectorAll` and `getElementsByClassName`

- querySelectorAll: Returns a `NodeList` object that contains all elements matching the specified CSS selector.

- getElementsByClassName: Returns a live `HTMLCollection` of elements with the specified class name.