**JavaScript Interview Questions**

**1. HTML**

**1.1. General Layout of HTML File**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta name="description" content="For SEO purposes.">

<title>Your Page Title</title>

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

<link rel="icon" href="favicon.ico" type="image/x-icon">

<script src="script.js" defer></script>

</head>

<body>

<header>

<h1>Welcome to My Website</h1>

<nav>

<ul>

<li><a href="index.html">Home</a></li>

</ul>

</nav>

</header>

<main>

<section>

<h2>Subheading</h2>

<p>Paragraph</p>

</section>

</main>

<footer></footer>

<!-- Inline JavaScript (optimised performance placed at the end) -->

<script></script>

</body>

</html>

**1.2. Semantic HTML tags?**

* Tags that define the meaning of the content they contain.
* Eg. <header>, <article>, and <footer>
* On the other hand, tags like <div> and <span> are typical examples of non-semantic HTML elements.

**2. CSS**

**2.1. CSS Specificity?**

* Set of rules that browsers use to determine which CSS styles are applied to an element when there are multiple competing styles.
* Goal to make stylesheets predictable and maintainable.
* Can use !important to override specificity but can cause issues with management.

|  |  |  |
| --- | --- | --- |
| **Value** | **Selector** | **Example** |
| a. (1,0,0,0) | Inline styles (inside a style attribute). | <div style="color: red;"></div> |
| b. (0,1,0,0) | IDs. | #header { color: blue; } |
| c. (0,0,1,0) | Classes, attributes, and pseudo-classes (:hover, :focus) | .content { color: green; }  [type="text"] { color: yellow; }  :hover { color: pink; } |
| d. (0,0,0,1) | Elements and pseudo-elements (::before, ::after) | p { color: black; }  ::before { content: ''; } |

**2.2. Box Model?**

* Describes how elements on a web page are structured and how their dimensions are calculated consisting of several components, each contributing to the element's overall size and how it interacts with other elements.
* Content = height, width
* Padding = top, right, bottom, left
* Border = width, style, colour
* Margin = top, right, bottom, left



**3. HTTP**

**3.1. List the main HTTP request methods.**

|  |  |  |
| --- | --- | --- |
| **Methods** | **Purpose** | **Characteristics\*** |
| GET | Retrieve data from the server. | Safe, Idempotent |
| POST | Submit data to the server to create a new resource. | Not Idempotent |
| PUT | Update an existing resource or create a new resource if it does not exist. | Idempotent |
| PATCH | Partially update an existing resource. | Not Necessarily Idempotent |
| DELETE | Remove a resource. | Idempotent |
| HEAD | Get the headers of a resource, like GET but without the response body. | Safe, Idempotent |

\*Safe = does not alter the resource

\*Idempotent = same request with the same data will produce the same result

**4. JavaScript - General**

**4.1. Higher Order Functions?**

* JavaScript functions that take in or return a function.
* Eg. map(), filter()

**4.2. Arrow Functions?**

* Introduced in ES6 and are more precise way of writing functions.
* Key differences from normal functions:
  + Removal of function keyword.
  + If only consist of a return statement following a single line of code, we can remove the curly braces and remove the return keyword.
  + If function has only one argument, we can remove the parenthesis.
  + Cannot use arrow functions as object constructors.
  + Eg. const Car = (color) => { this.color = color;};

**4.3. Difference between == and ===?**

* === ensures that both the value and type are the same.
* Helps in avoiding unexpected behaviour caused by type coercion.
* Makes the code easier to debug and in general it’s in best practice to use.

**4.3. Difference between null and undefined?**

* null == undefined (true because both are considered loosely equal)
* null === undefined (false because they are of different types)
* null is technically an object with no value and undefined are uninitialized variables and function parameters that are not provided.

**5. Rendering**

**5.1. Server-Side Rendering (SSR) vs. Client-Side Rendering (CSR).**

* SSR: The server generates the full HTML (+CSS and JS functions) for a web page and sends it to the client's browser. The browser then renders the HTML to display the content to the user.
* CSR: the browser downloads a minimal HTML shell and then uses JavaScript to build and render the content dynamically on the client side.

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Server-Side Rendering (SSR)** | **Client-Side Rendering (CSR)** |
| Initial Load Time | Faster | Slower |
| Subsequent Navigation | Slower (full page reloads) | Faster (partial updates, no full reloads) |
| SEO | Better (pre-rendered HTML) | Challenging (requires additional setup like pre-rendering or server-side rendering for critical pages) |
| Server Load | Higher (renders HTML on server) | Lower (renders HTML on client) |
| Interactivity | Less dynamic (requires full reloads) | More dynamic (single-page application feel) |
| Browser Compatibility | Better (less reliance on JavaScript) | Relies heavily on JavaScript |
| Frameworks | Next.js, Ruby on Rails | React.js, Vue.js, Angular |

**5.2. In what scenarios would you prefer Client-Side Rendering over Server-Side Rendering?**

* Web pages requiring dynamic page generation eg. online quiz.
* Advanced UI Components: Applications that utilize sophisticated UI components (e.g., drag-and-drop, animations, real-time data visualization) are well-suited for CSR since it allows for more dynamic and responsive interfaces.