ChatGPT

Q: What is HTML and how do `<div>` and `<span>` differ?
A: HTML (HyperText Markup Language) is the standard language for creating and structuring web pages. A `<div>` tag is a **block-level** element (it creates a block that takes up the full available width), used to group larger sections of content. A `<span>` tag is an **inline** element (it only takes up as much width as its content) and is used to group inline content (like text within a paragraph) ① .

Q: What is the difference between `id` and `class` in HTML?
A: An `id` attribute uniquely identifies a single element on a page (each `id` value can be used only once), whereas a `class` can be applied to multiple elements. In CSS, `#idName` selects an element by its id, and `.className` selects elements by class. In short, "id" is unique per page; "class" can be reused by many elements ② .

Q: What are semantic elements in HTML5?
A: Semantic elements are tags that have meaningful names reflecting their purpose, such as `<header>`, `<footer>`, `<nav>`, `<article>`, etc. They clearly describe their content (for example, `<article>` represents an independent piece of content). HTML5 introduced many semantic tags to make code easier to read for developers and to help browsers understand how to treat each part of a webpage ③ .

Q: What is CSS and why do we use it?
A: CSS (Cascading Style Sheets) is the style sheet language used to describe the presentation of HTML content. It lets you apply styling (like fonts, colors, layouts) across one or many pages. Using CSS separates content from design: you write styles once and reuse them, which saves time and ensures consistency. It also cleans up HTML markup (making pages easier for search engines to index) and makes global design changes simple (update one stylesheet to affect the whole site) ④ ⑤ .

Q: What is the CSS box model?
A: The CSS box model is how browsers render elements as rectangular boxes. Each element is treated as a box with **content** in the middle, surrounded by optional **padding**, **border**, and **margin** areas. For example, text or images are the content; padding adds space around the content inside the border; the border wraps around padding; and the margin is space outside the border. Together, these layers determine the element's size and spacing ⑥ .

Q: What is JavaScript?
A: JavaScript is a high-level, interpreted programming language used to make web pages interactive. It runs in web browsers (and on servers via platforms like Node.js) and can change HTML/CSS in response to user actions. In other words, it's the scripting language that allows webpages to respond to user input and create dynamic content ⑦ .

Q: What is hoisting in JavaScript?
A: Hoisting is a JavaScript mechanism where variable and function **declarations** are moved to the top of their containing scope during the compile phase. In practice, it means you can call a function or access a `var`-declared variable before its written declaration in code (though the `var` variable will be `undefined` until assignment). For example, functions and `var` variables act as if their declaration were at the top of their scope ⑧ .

Q: What is the difference between `==` and `===` in JavaScript?
A: The `==` operator compares two values for equality after converting them to a common type (type coercion). The `===` operator (strict equality) compares both value **and** type without converting. For example, `5 == "5"` is true (string coerces to number), but `5 === "5"` is false because the types differ [9].

Q: What is the difference between `var`, `let`, and `const` in JavaScript?
A: `var` declares a variable with function scope (or global if not in a function) and allows redeclaration. `let` and `const` (introduced in ES6) declare block-scoped variables (limited to `{ }` scope). `let` can be reassigned, while `const` cannot be reassigned once initialized. Unlike `var`, `let`/`const` do not hoist in the same way, which helps avoid certain bugs [10].

Q: What are closures in JavaScript?
A: A closure is a function that "remembers" the environment in which it was created. In practice, a closure is an inner function that has access to variables from an outer function even after that outer function has finished executing. This allows the inner function to use and modify those variables later. For example, if an inner function returns a reference to an outer variable, that variable persists via the closure [11].

Q: What is a callback function in JavaScript?
A: A callback is a function passed as an argument to another function. The receiving function can then invoke (call) the callback at a later time. Callbacks are commonly used for asynchronous tasks or events. For example, when making an AJAX request, you might pass a callback to handle the response. The callback won't run until the main function (like an event or data fetch) has completed [12].

Q: What are Promises in JavaScript?
A: A Promise is an object that represents the eventual success or failure of an asynchronous operation. Instead of using nested callbacks, you get a Promise and attach `.then()` for success and `.catch()` for errors. For example, `fetch(url).then(response => …)` returns a Promise that resolves when the data is ready. Promises allow cleaner, more readable handling of async results [13].

Q: What are arrow functions in JavaScript?
A: Arrow functions are a concise syntax for writing functions, using `=>`. They are shorter to write than normal function expressions. Unlike normal functions, arrow functions inherit the `this` context from their surrounding scope (they do not have their own `this`). Example: `const add = (a, b) => a + b;` defines a function that returns `a+b`. This syntax is more compact and avoids common pitfalls with `this` binding [14].

Q: What is the difference between `null` and `undefined` in JavaScript?
A: In JavaScript, `null` is a value that represents "no value" and is explicitly assigned by the programmer. `undefined` means a variable has been declared but has not been assigned a value yet. For instance, if you write `let x;`, then `x` is `undefined`. If you later set `x = null;`, then `x` has a `null` value. In summary, `null` is an intentional absence of value, while `undefined` usually means a value is missing [15].

Q: What is ReactJS?
A: ReactJS is an open-source JavaScript library for building user interfaces. It allows developers to create

reusable UI components that manage their own state. React uses a virtual DOM to efficiently update only parts of the page when data changes, which makes web apps fast. It's widely used for single-page applications because it can dynamically update the view without full page reloads [16].

Q: What is the difference between props and state in React?
A: In React, *props* (short for properties) are inputs passed from a parent component to a child component. They are read-only in the child. *State* is data maintained within a component that can change over time (often via user input or async events). So props allow data flow between components, while state is local to a component. Props make components reusable by giving them external data, whereas state makes components interactive with internal data [17].

Q: What is JSX in React?
A: JSX is a syntax extension for JavaScript used in React. It looks like HTML/XML and lets you write HTML elements inside JavaScript code. For example, you can write `<h1>Hello</h1>` directly in a React component's `render` method. Browsers don't understand JSX natively; a transpiler (like Babel) converts it into regular JavaScript calls (`React.createElement`) before running. JSX makes defining component layouts more intuitive [18].

Q: What are React components and their types?
A: React components are the building blocks of React apps—independent, reusable pieces of UI. The two main types are *functional components* (simple functions that return JSX) and *class components* (ES6 classes with a `render()` method). With React Hooks (like `useState`), functional components can now have state and lifecycle features that were once only in class components. Class components manage state with `this.state` and lifecycle methods, whereas functional components use hooks to handle state and effects [19].

Q: What is Next.js?
A: Next.js is an open-source React framework created by Vercel for building modern web applications. It provides features like **Server-Side Rendering (SSR)** and **Static Site Generation (SSG)** out of the box, which help with performance and SEO. Next.js also supports file-based routing, API routes (serverless functions), and automatic code splitting. In short, Next.js simplifies building fast, scalable React apps with built-in optimizations [20] [21].

Q: How is Next.js different from other JavaScript frameworks?
A: Next.js is specifically tailored for React and stands out by handling server-side rendering and static generation without extra setup. Its built-in SSR means pages can be pre-rendered on the server for better SEO and faster first load. Next.js also automatically splits JavaScript bundles per page, provides an optimized image component, and lets you easily create backend API routes. These features (SSR, SSG, image optimization, easy API endpoints) come by default in Next.js, whereas many other frameworks require manual configuration [22] [23].

Q: What is Tailwind CSS?
A: Tailwind CSS is a **utility-first** CSS framework. Instead of writing custom CSS rules, you use its small utility classes directly in your HTML. For example, to apply margin or font styles, you might use classes like `mt-4` or `text-blue-500`. This approach allows rapid development of responsive designs without leaving the HTML. Tailwind is open-source, highly customizable via a configuration file, and is designed to minimize CSS file size by purging unused classes [24].

Q: Why use Tailwind CSS?
A: Tailwind speeds up styling by letting you apply ready-made utility classes rather than writing new CSS. It's great for prototyping and keeping styles consistent. Tailwind is highly customizable (you can configure themes and breakpoints) and supports responsive design right in class names. Because you build designs by composing utilities, it often reduces the final CSS size (unused classes can be purged). Big projects and companies (like GitHub or Mozilla) use Tailwind for its speed and flexibility [24] [25].

Q: What does "utility-first" mean in Tailwind CSS?
A: "Utility-first" means Tailwind provides lots of single-purpose classes (utilities) for styling. Instead of writing custom class names, you combine utilities in your HTML. For example, instead of defining `.btn { padding: 1rem; background: blue; }` in CSS, you can directly write `<button class="p-4 bg-blue-500">`. This modular approach keeps styling inline with markup and avoids writing unique CSS for every component [25].

Q: How do you install Tailwind CSS in a project?
A: You can install Tailwind via npm or use a CDN for quick prototypes. With npm, you run `npm install tailwindcss`, then create a CSS file that includes the Tailwind directives:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Finally, build your CSS (for example with `npx tailwindcss build input.css -o output.css`). The CDN approach simply involves adding a `<link>` to the Tailwind CDN in your HTML. Both methods will give you access to all of Tailwind's utility classes [26] [27].

Q: How do you make text bold in Tailwind CSS?
A: Use the `font-bold` utility class on any element. For example: `<p class="font-bold">Bold text</p>` will render the text inside as bold. Tailwind's utility classes like `font-bold` directly correspond to common CSS properties (here `font-weight: 700`) [28].

Q: How do you center content horizontally and vertically with Tailwind?
A: To center horizontally, use `mx-auto` on a block-level element (it applies `margin-left: auto; margin-right: auto;`). To center vertically inside a flex container, use `self-center` on the element (it aligns itself along the cross-axis). For example:

```
<div class="flex h-screen">
  <div class="mx-auto self-center">Centered both ways</div>
</div>
```

This centers the inner div both horizontally and vertically [29].

Q: What is Django?
A: Django is a high-level Python web framework that encourages rapid development and clean design. It is

open-source and follows the Model-Template-View (MTV) architectural pattern (similar to MVC). Django handles many web development tasks out of the box (like database ORM, authentication, admin interface), helping you build robust, secure websites quickly [30] .

Q: What is the MTV (Model-Template-View) architecture in Django?
A: In Django's MTV pattern, the *Model* is the data-access layer (your database tables/models), the *Template* is the presentation layer (HTML files that define the UI), and the *View* is the business logic layer (Python functions that handle requests and return responses). Together, they separate concerns: Models define data structure, Views process user input and interact with models, and Templates format the output HTML [30] .

Q: What is FastAPI and why use it?
A: FastAPI is a modern, fast (high-performance) web framework for building APIs with Python 3.7+. It uses standard Python type hints to automatically validate data and generate API documentation (Swagger/OpenAPI). Unlike older frameworks, FastAPI is built on asynchronous ASGI servers (like Uvicorn), which allows it to handle many requests efficiently. It's known for being easy to use and for providing automatic client generation and schema validation [31] .

Q: How do type hints work in FastAPI?
A: FastAPI uses Python's type hints on path or query parameters and request bodies to perform automatic request validation and serialization. For example, if you write `def read_item(item_id: int): ...`, FastAPI ensures `item_id` is an integer from the request. It uses these type annotations to check input data and convert it to the right type before your code runs. This means many potential bugs are caught early, and FastAPI can generate documentation based on these types [32] .

Q: What is dependency injection in FastAPI?
A: FastAPI's dependency injection lets you define "dependencies" (like database connections, authentication, etc.) as parameters in your path operation functions. You annotate these parameters with dependency providers (functions or classes). FastAPI then knows to resolve these dependencies automatically when handling a request. This makes code more modular and testable, because you can easily swap or mock these dependencies for different routes or tests [33] .

Q: How can you test file uploads in a FastAPI app?
A: You can test file uploads using `TestClient` from `fastapi.testclient`. For example, you can write:

```python
from fastapi.testclient import TestClient
client = TestClient(app)
with open("test.txt", "rb") as f:
    response = client.post("/upload", files={"file": ("test.txt", f, "text/plain")})
```

This simulates sending a file in a multipart/form-data request. FastAPI's `UploadFile` type will receive the file. In tests, this lets you verify that your endpoint correctly processes file uploads [34] .

---

(1) (2) (3) HTML Interview Questions and Answers (2024) - Intermediate Level - GeeksforGeeks
https://www.geeksforgeeks.org/html/html-interview-questions-answers-set-2/

(4) (5) (6) CSS Interview Questions and Answers - GeeksforGeeks
https://www.geeksforgeeks.org/css/css-interview-questions/

(7) (8) (9) (10) (11) (12) (13) (14) (15) 55 Top JavaScript Interview Questions with Example Answers | Built In
https://builtin.com/software-engineering-perspectives/javascript-interview-questions

(16) (17) (18) (19) React Interview Questions and Answers - GeeksforGeeks
https://www.geeksforgeeks.org/reactjs/react-interview-questions/

(20) (21) (22) (23) Next.js Interview Questions and Answers - 2025 - GeeksforGeeks
https://www.geeksforgeeks.org/reactjs/next-js-interview-questions-answers/

(24) (25) (26) (27) (28) (29) Tailwind CSS Interview Questions and Answers -2025 - GeeksforGeeks
https://www.geeksforgeeks.org/css/tailwind-css-interview-questions-and-answers/

(30) Django - Creating a Multi-Page Website - GeeksforGeeks
https://www.geeksforgeeks.org/python/django-creating-a-multi-page-website/

(31) (32) (33) (34) Top asked FastAPI Interview Questions | Vskills
https://www.vskills.in/interview-questions/fastapi-interview-questions