Series 1: Next.js Fundamentals & Core Concepts
with **interview-style Q&A**, **simple code examples**, and **real-world analogies**.

🔹 Series 1: Next.js Fundamentals & Concepts

---

1. What is Next.js and why would you use it over React?

Interview Answer: Next.js is a React framework that adds features like server-side rendering (SSR), static site generation (SSG), file-based routing, API routes, and performance optimizations out of the box. Unlike React, which is just the UI library, Next.js handles full-stack concerns like routing and data fetching, making it better for SEO and production-ready apps.

🧪 **Example Code:**

```
// pages/index.js
export default function Home() {
  return <h1>Hello from Next.js!</h1>
}
```

🧠 **Real-world Analogy:** React is like a raw engine; you can build anything, but need to handle everything (routing, SSR). Next.js is like a car with built-in GPS, AC, and a dashboard — much easier to drive to production.

---

2. What is File-based Routing in Next.js?

Interview Answer: Next.js uses file-based routing where each `.js` or `.tsx` file in the `pages/` folder automatically becomes a route. There's no need to use `react-router` or create a route config manually.

🧪 **Example Code:**

```
pages/
├── index.js     →  "/"
├── about.js     →  "/about"
└── blog/
    └── [id].js  →  "/blog/:id"
```

🧠 **Real-world Analogy:** Imagine your website as a filing cabinet. Each file in the `pages/` folder is like a labeled tab in the drawer. You open it, and boom — there's the content.

---

3. What's the difference between a page and a component in Next.js?

Interview Answer: A page is a top-level route under `pages/`, directly mapped to a URL. A component is a reusable UI piece that can be used inside pages or other components. Pages are entry points; components are building blocks.

🧪 **Example Code:**

```
// pages/about.js
import Header from '../components/Header';

export default function About() {
  return (
    <>
      <Header />
      <p>About Page</p>
    </>
  );
}
```

🧠 **Real-world Analogy:** Think of a **page** like a house (complete unit with an address), and **components** like the furniture inside — reusable and movable.

---

4. What is `Link` in Next.js and why use it over `<a>`?

Interview Answer: Next.js provides a `Link` component for client-side navigation without full page reloads. Using `<a>` directly causes full reloads. `Link` improves performance and preserves state.

🧪 **Example Code:**

```
import Link from 'next/link';

<Link href="/about">Go to About</Link>
```

🧠 **Real-world Analogy:** Using `Link` is like switching TV channels with a remote (fast, no delay), whereas `<a>` is like turning off the TV and booting it up again (slow reload).

---

5. What is Pre-rendering in Next.js?

Interview Answer: Pre-rendering means HTML is generated in advance, either at build time (SSG) or on each request (SSR), instead of waiting for client-side JavaScript. It improves performance and SEO.

🧪 **Example Code:**

```
// getStaticProps for pre-rendering at build time
export async function getStaticProps() {
  return { props: { message: 'Pre-rendered!' } };
}
```

🧠 **Real-world Analogy:** Pre-rendering is like preparing food before guests arrive. When they come (users), it's ready to serve instantly — not cooked from scratch (CSR).

---

6. What's the difference between SSR and CSR?

Interview Answer: SSR (Server-Side Rendering) means HTML is generated on the server per request. CSR (Client-Side Rendering) means HTML is a blank shell, and JavaScript fetches data after page load. SSR is better for SEO; CSR is better for dynamic dashboards.

🧪 **Example Code:**

```
// SSR using getServerSideProps
export async function getServerSideProps() {
  return { props: { data: 'Hello SSR' } };
}
```

🧠 **Real-world Analogy:** SSR is like a waiter bringing a ready meal to your table. CSR is like a hotpot — you cook it yourself after getting ingredients.
