Introduction to TypeScript & Single Page Applications (SPA)

PART 1: INTRODUCTION TO TYPESCRIPT

1. What is TypeScript?
TypeScript is an open-source, strongly typed programming language developed and maintained by Microsoft. It builds on JavaScript by adding static type definitions. Because browsers and runtime environments cannot execute TypeScript directly, TypeScript code is compiled (transpiled) down into standard, plain JavaScript files using the TypeScript compiler (tsc).

2. Core Benefits of TypeScript
- Static Typing: Catches type-related bugs and syntax errors at compile-time during development rather than at runtime in production.
- Enhanced Tooling: Empowers modern IDEs (like VS Code) with advanced features such as intelligent code completion (IntelliSense), robust refactoring, and accurate navigation.
- Modern ECMAScript Features: Allows developers to write cutting-edge ES6+ code that targets older browser environments seamlessly.

3. Basic Types in TypeScript
- Primitive Types: string, number, boolean, null, undefined.
- Arrays & Tuples: string[] or Array<number> for homogenous collections; tuples for fixed-length arrays with distinct types.
- Special Types: any (opts out of type checking), unknown (safer alternative to any), void (used for functions returning no value).

4. Program: TypeScript Interfaces and Functions

```
// Save as user.ts
interface User {
  id: number;
  name: string;
  email: string;
  isActive?: boolean; // Optional property
}

function formatUserGreeting(user: User): string {
  const status = user.isActive ?? true ? 'Active' : 'Inactive';
  return `Welcome back, ${user.name} (${user.email}) - Status: ${status}`;
}

const appUser: User = {
  id: 101,
  name: 'Asha Sharma',
```

```
  email: 'asha@example.com'
};

console.log(formatUserGreeting(appUser));
```

PART 2: SINGLE PAGE APPLICATIONS (SPA)

1. What is a Single Page Application (SPA)?
A Single Page Application (SPA) is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, rather than loading entire new pages from scratch. This results in a fast, fluid, desktop-like user experience resembling native applications.

2. How SPAs Work vs Traditional Multi-Page Apps (MPA)
- Traditional MPA: Every user action, click, or navigation triggers a full page request to the server, reloading HTML, CSS, and JS assets.
- Single Page Application: The browser loads a single HTML shell (e.g., index.html) once. Subsequent views and content are rendered dynamically via client-side JavaScript routers and fetched via AJAX/Fetch API.

3. Advantages and Challenges of SPAs
- Advantages: Lightning-fast navigation after initial load, reduced server load, smooth animations, and excellent developer experience using component-based frameworks (React, Angular, Vue).
- Challenges: Initial load time can be heavier, SEO optimization requires server-side rendering (SSR) or pre-rendering, and client-side memory management requires careful handling.

4. Program: Vanilla JavaScript SPA Router Simulation

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>SPA Router Demo</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 40px; }
    nav a { margin-right: 15px; text-decoration: none; color: #0066cc; font-weight: bold; }
    #app { margin-top: 20px; padding: 20px; border: 1px solid #ccc; border-radius: 6px;
background: #f9f9f9; }
  </style>
</head>
<body>
  <nav>
    <a href="#home" onclick="loadRoute('home')">Home</a>
    <a href="#about" onclick="loadRoute('about')">About</a>
```

```
    <a href="#contact" onclick="loadRoute('contact')">Contact</a>
  </nav>
  <div id="app">Loading content...</div>
  <script>
    const routes = {
      home: '<h2>Home View</h2><p>Welcome to our Single Page Application homepage.</p>',
      about: '<h2>About View</h2><p>This page explains how client-side routing works without full reloads.</p>',
      contact: '<h2>Contact View</h2><p>Get in touch with us at support@example.com.</p>'
    };
    function loadRoute(route) {
      const content = routes[route] || '<h2>404</h2><p>Page not found.</p>';
      document.getElementById('app').innerHTML = content;
    }
    // Initial load
    loadRoute('home');
  </script>
</body>
</html>
```