Angular Concepts (Detailed Explanation)

1. Component-Based Architecture

Angular applications are built from components — small, reusable building blocks that represent parts of the user interface (like a header, sidebar, or form).
Each component combines logic (TypeScript), layout (HTML), and styling (CSS), making it easy to manage, reuse, and test parts of your app independently.

---

2. Modules

Modules are containers that organize related parts of an Angular application — components, directives, pipes, and services.
They help structure an app logically, enable lazy loading (loading only what's needed), and make large projects scalable.

---

3. Data Binding

Data binding connects the component's data (TypeScript) with the user interface (HTML).
It ensures that when data changes in the logic, the UI updates automatically, and vice versa.
There are four types:

- **Interpolation:** Displays dynamic data inside the template.
- **Property binding:** Passes data from component to element attributes.
- **Event binding:** Sends user actions (like clicks) back to the component.
- **Two-way binding:** Syncs data both ways between component and template.

---

4. Directives

Directives let you change or extend HTML's behavior.

- **Structural directives** (like `*ngIf`, `*ngFor`) change the DOM structure — adding, removing, or repeating elements.
- **Attribute directives** (like `ngClass`, `ngStyle`) modify how elements look or behave without changing structure.

---

5. Services and Dependency Injection (DI)

Services store reusable logic or data that can be shared across components — for example, fetching data from an API.
Dependency Injection automatically provides these services to components without manually creating new instances, improving modularity and testing.

---

6. Routing

Angular's Router manages page navigation within a single-page application (SPA).
Instead of reloading the whole page, it switches components dynamically, creating the illusion of multiple pages.

---

7. RxJS (Reactive Extensions for JavaScript)

RxJS is a library for handling asynchronous data streams using Observables.
It's used in Angular for handling API responses, user input streams, and real-time updates efficiently.

---

8. Forms

Angular supports two ways to handle forms:

- **Template-driven forms:** Simple and quick, ideal for small forms.
- **Reactive forms:** More robust and scalable; form logic is handled in TypeScript, suitable for complex validation and dynamic forms.

---

9. Lifecycle Hooks

Components in Angular go through a lifecycle — creation, update, and destruction.
Hooks like `ngOnInit`, `ngOnChanges`, and `ngOnDestroy` let you run custom logic at specific points in that lifecycle (e.g., fetching data when the component loads).

---

10. Change Detection

Angular automatically updates the view when data changes in the component.
You can use strategies like **Default** (automatic detection) or **OnPush** (manual, performance-optimized) to control how updates happen.

---

11. Pipes

Pipes transform data before displaying it — for example, formatting a date, currency, or filtering a list.
They make templates cleaner and more readable.

---

12. HTTP Client

`HttpClientModule` allows communication with backend APIs.
It supports GET, POST, PUT, DELETE, and error handling, all built around RxJS Observables for asynchronous operations.

---

13. Guards and Interceptors

- **Guards:** Control route access (for example, blocking unauthorized users).
- **Interceptors:** Modify or monitor HTTP requests and responses globally (for example, adding authentication tokens or logging errors).

---

14. Performance Optimization

Angular offers tools for faster apps:

- **Lazy loading:** Load modules only when needed.
- **OnPush strategy:** Detect changes only when necessary.
- **Ahead-of-Time (AOT) compilation:** Compiles templates during build, not runtime.
- **Minification and bundling:** Reduce file size for faster loading.

□ HTML Fundamentals

1. HTML Elements

HTML provides the structure of a webpage using tags like `<div>`, `<a>`, `<p>`, and `<img>`.
Each element defines a different part of the page (text, image, links, containers, etc.).

---

2. HTML5 Features

HTML5 introduced semantic tags like `<header>`, `<footer>`, `<nav>`, `<article>`, and built-in support for media (`<audio>`, `<video>`).
It also added APIs for geolocation, local storage, and offline web apps.

---

3. Document Structure

An HTML page has three main sections:

- `<html>`: The root element.
- `<head>`: Metadata, links, scripts, and titles.
- `<body>`: The visible content.
  This structure ensures browsers interpret the page correctly.

---

4. Forms and Inputs

Forms collect user input using fields like text boxes, dropdowns, and checkboxes.
They are the main way users interact with a web application.

---

5. Accessibility

Accessibility ensures everyone, including users with disabilities, can use your site.
This includes adding `alt` text for images, using semantic tags, and applying ARIA roles for screen readers.

□ CSS Essentials

1. Selectors

Selectors target specific HTML elements to style them.
Examples include IDs (`#header`), classes (`.button`), and element selectors (`p`, `div`).

---

2. Box Model

Every element in CSS has four layers: content, padding, border, and margin.
Understanding this model helps you control spacing and alignment.

---

3. Specificity

When multiple styles target the same element, CSS decides which one to apply using **specificity rules**:
Inline styles > ID selectors > Class selectors > Element selectors.

---

4. Flexbox
A layout model that arranges elements in flexible rows or columns, automatically adjusting to screen sizes.
Useful for aligning items horizontally and vertically with ease.

---

5. Grid
A two-dimensional layout system ideal for complex page structures, allowing precise control over rows and columns.

---

6. Media Queries
Used to apply different styles depending on screen size, device orientation, or resolution — essential for responsive design.

---

7. Positioning
CSS positioning controls where elements appear:

- **Static:** Default flow.
- **Relative:** Moved relative to its normal position.
- **Absolute:** Positioned relative to the nearest positioned ancestor.
- **Fixed:** Stays in place even when scrolling.
- **Sticky:** Behaves like relative until scrolled past.

---

8. Responsive Design
Ensures web pages look good on all screen sizes using fluid layouts, flexible images, and adaptive CSS units (`rem`, `%`, `vw`, `vh`).

---

9. Preprocessors (Sass/Less)
They extend CSS with variables, nesting, and mixins — making large stylesheets cleaner and easier to maintain.

⚙ JavaScript Concepts
1. Variables and Scope
`var`, `let`, and `const` define variables.

- `var` is function-scoped,
- `let` and `const` are block-scoped,
- `const` prevents reassignment.

---

2. Data Types
Includes primitives (string, number, boolean, null, undefined) and complex types (object, array, function).
Understanding types avoids unexpected bugs.

---

3. Functions and Closures
Functions group logic into reusable units.
Closures allow functions to “remember” variables from their parent scope, even after that scope ends.

---

4. Event Handling
JavaScript responds to user actions (clicks, hovers, input).
Event delegation attaches one listener to a parent element to manage many child elements efficiently.

---

5. DOM Manipulation
JavaScript can dynamically create, modify, or remove HTML elements using the Document Object Model (DOM).
This is the foundation of dynamic web pages.

---

6. Promises and Async Programming
Promises and async/await handle tasks that take time (like API calls).
They make asynchronous operations easier to read and manage.

---

7. Equality Operators
- == compares values with type conversion.
- === compares values and types (strict equality).
  Always prefer === to avoid unexpected results.

---

8. Event Bubbling and Capturing
When an event happens on a nested element, it travels:
- **Bubbling:** From inner to outer elements.
- **Capturing:** From outer to inner.
  Developers can control this for precise event handling.

---

9. Single-Page Applications (SPAs)
SPAs load a single HTML page and dynamically update the content using JavaScript — improving speed and user experience without full page reloads.

---

10. Security
Prevent attacks by validating and sanitizing input, avoiding `eval()`, using HTTPS, and implementing Content Security Policy (CSP).

---

11. Web APIs
APIs like `localStorage`, `fetch`, and `geolocation` extend what browsers can do — store data, make network requests, and access device features.