Template syntax

In Angular, a *template* is a chunk of HTML. Use special syntax within a template to build on many of Angular's features.

Prerequisites

Before learning template syntax, you should be familiar with the following:

- Angular concepts
- JavaScript
- HTML
- CSS

Each Angular template in your application is a section of HTML to include as a part of the page that the browser displays. An Angular HTML template renders a view, or user interface, in the browser, just like regular HTML, but with a lot more functionality.

When you generate an Angular application with the Angular CLI, the `app.component.html` file is the default template containing placeholder HTML.

The template syntax guides show you how to control the UX/UI by coordinating data between the class and the template.

Most of the Template Syntax guides have dedicated working example applications that demonstrate the individual topic of each guide. To see all of them working together in one application, see the comprehensive .

Empower your HTML

Extend the HTML vocabulary of your applications With special Angular syntax in your templates. For example, Angular helps you get and set DOM (Document Object Model) values dynamically with features such as built-in template functions, variables, event listening, and data binding.

Almost all HTML syntax is valid template syntax. However, because an Angular template is part of an overall webpage, and not the entire page, you don't need to include elements such as `<html>`, `<body>`, or `<base>`, and can focus exclusively on the part of the page you are developing.

To eliminate the risk of script injection attacks, Angular does not support the `<script>` element in templates. Angular ignores the `<script>` tag and outputs a warning to the browser console. For more information, see the Security page.

More on template syntax

You might also be interested in the following:

- Interpolation—learn how to use interpolation and expressions in HTML.

- Template statements—respond to events in your templates.
- Binding syntax—use binding to coordinate values in your application.
- Property binding—set properties of target elements or directive `@Input()` decorators.
- Attribute, class, and style bindings—set the value of attributes, classes, and styles.
- Event binding—listen for events and your HTML.
- Two-way binding—share data between a class and its template.
- Built-in directives—listen to and modify the behavior and layout of HTML.
- Template reference variables—use special variables to reference a DOM element within a template.
- Inputs and Outputs—share data between the parent context and child directives or components
- Template expression operators—learn about the pipe operator, `|`, and protect against `null` or `undefined` values in your HTML.
- SVG in templates—dynamically generate interactive graphics.