Angular is a development platform, built on TypeScript.

**Components:** Components are the building blocks that compose an application.

import { Component } from '@angular/core';

@Component({

selector: 'hello-world',

template: `

<h2>Hello World</h2>

<p>This is my first component!</p>

`

})

export class HelloWorldComponent {

// The code in this class drives the component's behavior.

}

A CSS selector that defines how the component is used in a template. HTML elements in your template that match this selector become instances of the component.

An HTML template that instructs Angular how to render the component

An optional set of CSS styles that define the appearance of the template's HTML elements

**Templates:** Every component has an HTML template that declares how that component renders. You define this template either inline or by file path.

import { Component } from '@angular/core';

@Component ({

selector: 'hello-world-interpolation',

templateUrl: './hello-world-interpolation.component.html'

})

export class HelloWorldInterpolationComponent {

message = 'Hello, World!';

}

<p>{{ message }}</p>

**double curly braces**—they instruct Angular to interpolate the contents within them.

Angular also supports property bindings, to help you set values for properties and attributes of HTML elements and pass values to your application's presentation logic.

<p

[id]="sayHelloId"

[style.color]="fontColor"

(click)="sayMessage()">

You can set my color in the component!

</p>

**square brackets**—that syntax indicates that you're binding the property or attribute to a value in the component class.

**Round brackets**— respond to user actions such as keystrokes, mouse movements, clicks, and touches

**Directives:** most popular directives in Angular are \*ngIf and \*ngFor

.html

<button type="button" (click)="onEditClick()">Make text editable!</button>

<div \*ngIf="canEdit; else noEdit">

<p>You can edit the following paragraph.</p>

</div>

<ng-template #noEdit>

<p>The following paragraph is read only. Try clicking the button!</p>

</ng-template>

.ts

message = "I'm read only!";

canEdit = false;

onEditClick() {

this.canEdit = !this.canEdit;

if (this.canEdit) {

this.message = 'You can edit me!';

} else {

this.message = "I'm read only!";

}

}

**Dependency injection** lets you declare the dependencies of your TypeScript classes without taking care of their instantiation. Instead, Angular handles the instantiation for you. This design pattern lets you write more testable and flexible code.

consider the following example. The first file, logger.service.ts, defines a Logger class. This class contains a writeCount function that logs a number to the console.

import { Injectable } from '@angular/core';

@Injectable({providedIn: 'root'})

export class Logger {

writeCount(count: number) {

console.warn(count);

}

}

Next, the hello-world-di.component.ts file defines an Angular component.

import { Component } from '@angular/core';

import { Logger } from '../logger.service';

@Component({

selector: 'hello-world-di',

templateUrl: './hello-world-di.component.html'

})

export class HelloWorldDependencyInjectionComponent {

count = 0;

constructor(private logger: Logger) { }

onLogMe() {

this.logger.writeCount(this.count);

this.count++;

}

}

**Angular CLI:**

The Angular CLI is the fastest, straightforward, and recommended way to develop Angular applications.

[ng build](https://angular.io/cli/build): Compiles an Angular application into an output directory.

[ng serve](https://angular.io/cli/serve): Builds and serves your application, rebuilding on file changes.

[ng generate](https://angular.io/cli/generate): Generates or modifies files based on a schematic.

[ng test](https://angular.io/cli/test): Runs unit tests on a given project.

[ng e2e](https://angular.io/cli/e2e): Builds and serves an Angular application, then runs end-to-end tests.

**First-party libraries** that are provided by the Angular it self.

[Angular Router](https://angular.io/guide/router): Advanced client-side navigation and routing based on Angular components. Supports lazy-loading, nested routes, custom path matching, and more.

[Angular Forms](https://angular.io/guide/forms-overview): Uniform system for form participation and validation.

[Angular HttpClient](https://angular.io/guide/http): Robust HTTP client that can power more advanced client-server communication.

[Angular Animations](https://angular.io/guide/animations): Rich system for driving animations based on application state.

[Angular PWA](https://angular.io/guide/service-worker-intro): Tools for building Progressive Web Applications (PWA) including a service worker and Web application manifest.

[Angular Schematics](https://angular.io/guide/schematics): Automated scaffolding, refactoring, and update tools that simplify development at large scale.

Generat new Component: **ng generate component [component-name]**

Create SErvice in Angular: **ng generate service [service-name]**

**Deploy Your Project:**

npm install -g @angular/cli

npm install

ng serve --port 4201

ng build: building the application for the production environment.

**Angular Setup in Local:**

Node.js (<https://github.com/nodejs/release#release-schedule>)

npm package manager (which is installed with Node.js by default, check npm -v)

Install Angular CLI: npm install -g @angular/cli

Create new Project: ng new [app-name]

**Components LifeCycle:**

A component instance has a lifecycle that starts when Angular instantiates the component class and renders the component view along with its child views. The lifecycle continues with change detection, as Angular checks to see when data-bound properties change, and updates both the view and the component instance as needed. The lifecycle ends when Angular destroys the component instance and removes its rendered template from the DOM. Directives have a similar lifecycle, as Angular **creates, updates, and destroys** instances in the course of execution.

implementing one or more of the *lifecycle hook* interfaces in the Angular core library to achive the lifecycle methods. Angular creates, updates, or destroys that instance.

=> Lifecycle Methods:

ngOnChanges(): Called before ngOnInit()

ngOnInit(): Called once, after the first ngOnChanges(). ngOnInit() is still called even when ngOnChanges() is not

ngDoCheck() : Called immediately after ngOnChanges() on every change detection run, and immediately after ngOnInit() on the first run.

ngAfterContentInit() : Called once after the first ngDoCheck().

ngAfterContentChecked(): Called after ngAfterContentInit() and every subsequent ngDoCheck().

ngAfterViewInit(): Called once after the first ngAfterContentChecked().

ngAfterViewChecked(): Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked().

ngOnDestroy(): Called immediately before Angular destroys the directive or component.