JAVA FULL STACK DEVELOPMENT PROGRAM

Session 18: Angular Introduction

OUTLINE

- Introduction to Angular
- Angular Versions
- Angular Features
- Development Setup
- Structure of Project
- Angular Architecture
- Data Binding
- Directives

INTRODUCTION

- Angular 2+ is a JavaScript based open-source framework for building client-side web application.
- It is maintained by Google.
- It is a more streamlined framework that allows programmers to focus on simply building JavaScript classes.
- Angular 2+ is a complete rewrite from the same team that built AngularJS but it is completely different from AngularJS
- Improved on that functionality and made it faster, more scalable and more modern.
- Item powers developers to build applications that live on the web, mobile, or the desktop
- Angular 2+ and TypeScript are bringing true object oriented web development to the mainstream

ANGULAR VERSION

Angular Version	Date	Description
Angular 2	14.09.2016	Initial Version of Angular
Angular 4	23.03.2017	Version 4
Angular 5	11.11.2017	Version 5
Angular 6	03-05-2018	Version 6
Angular 7	18-10-2018	Version 7
Angular 8	25-08-2019	Version 8
Angular 9	06-02-2020	Version 9
Angular 10	24-06-2020	Version 10
Angular 10.0.12	24-08-2020	Version 10.0.12

PS: AngularJS: October 20, 2010

ANGULAR

- Angular 2+ is based on
  - ES6/ Typescript
  - DOM (Document Object Model)
  - Web Components
  - RxJS
  - Observables
  - Module Loaders
  - Angular 1.x

TypeScript

- TypeScript is an open source syntactic superset of JavaScript that compiles to JavaScript (EcmaScript 3+).
- TypeScript offers type annotations which provide optional, static type checking at compile time.
- Since it is a superset of JavaScript, all JavaScript is syntactically valid TypeScript. However, that does not mean all JavaScript can actually be processed by the TypeScript compiler

```
let a = 'a';
a = 1; // throws: error TS2322: Type '1' is not assignable to type 'string'.
```

TypeScript

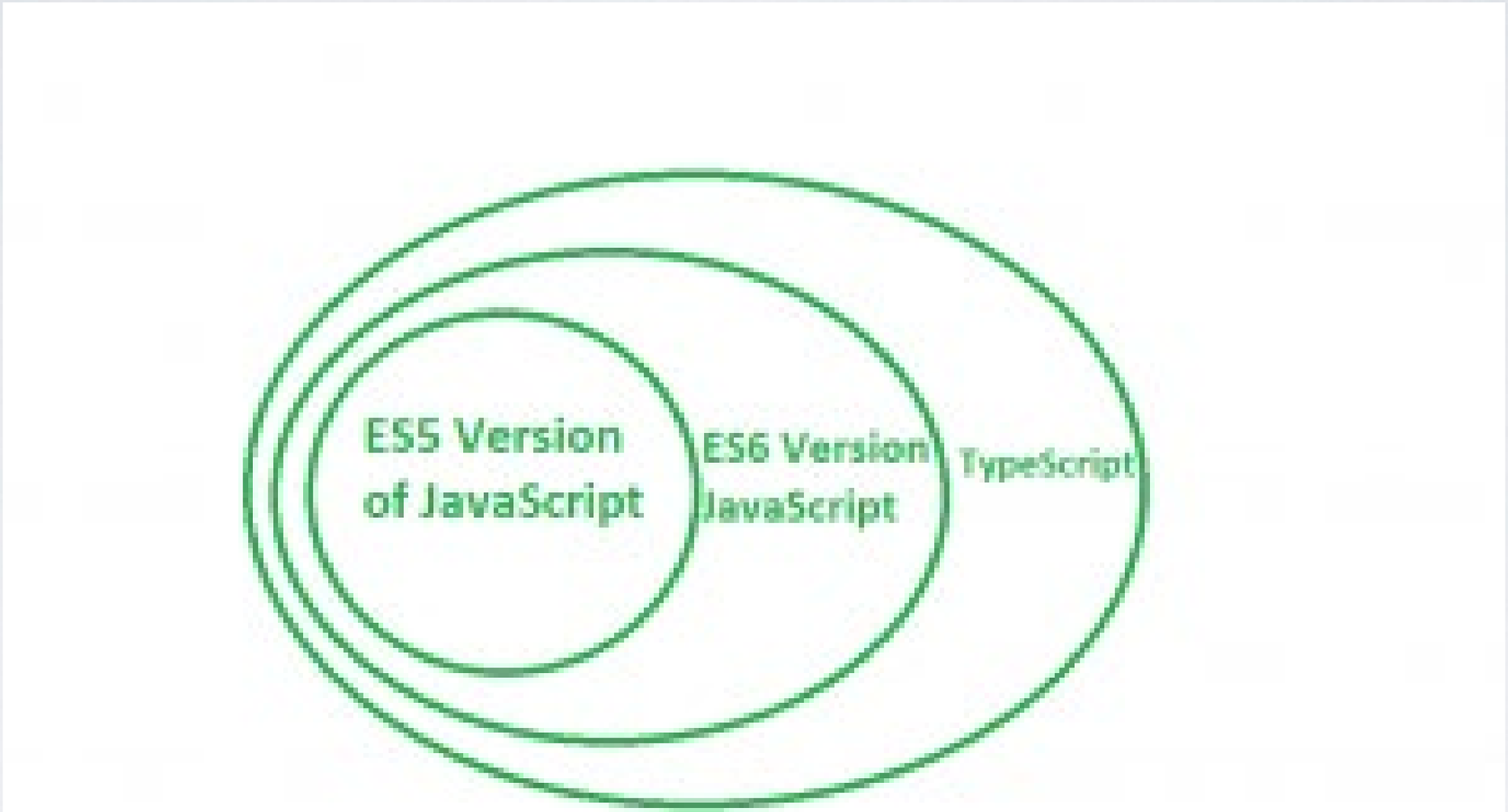


Fig: TypeScript is a SuperSet of JavaScript

TypeScript

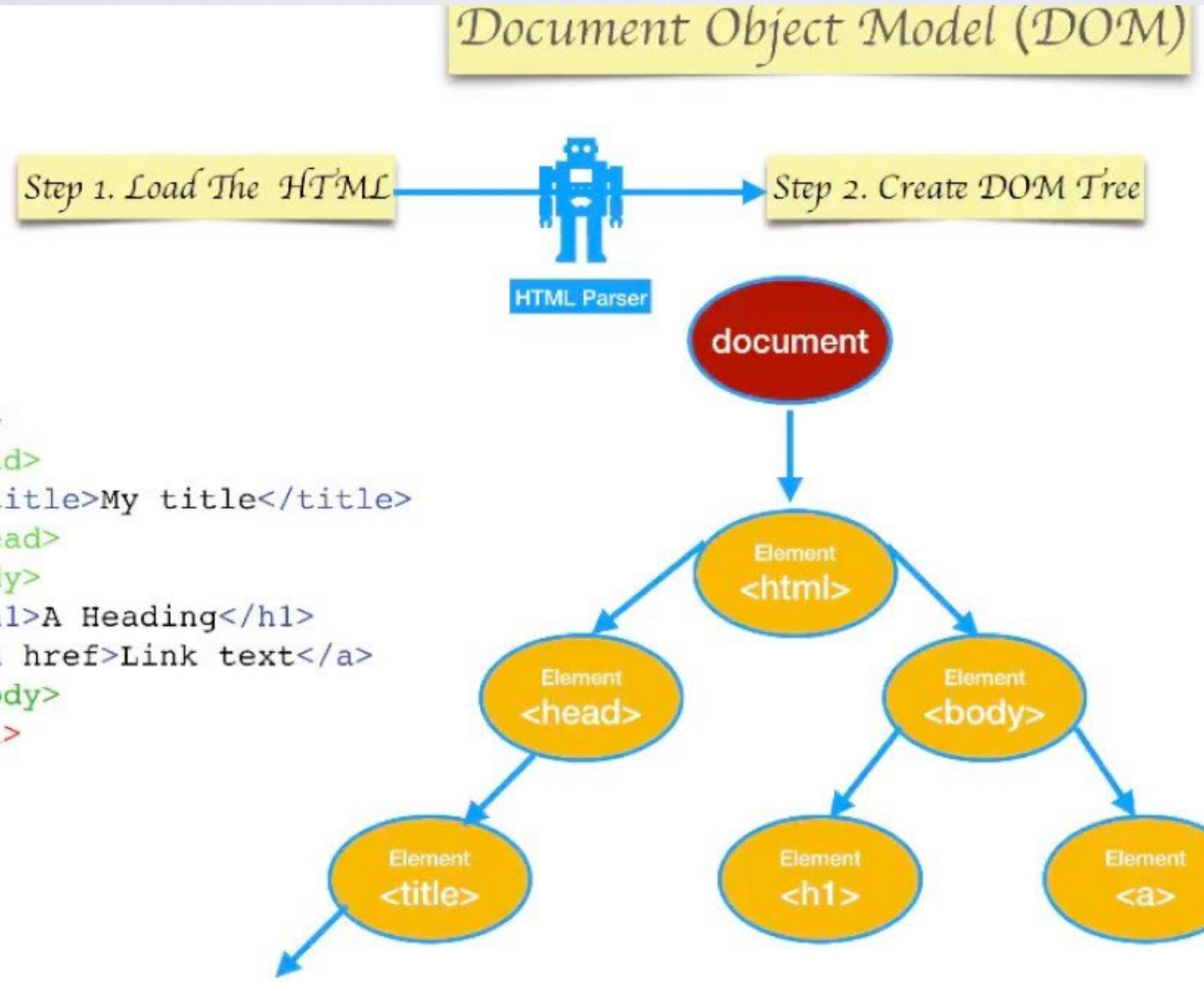
```
// Basic JavaScript
function getPassword(clearTextPassword) {
    if(clearTextPassword) {
        return 'password';
    }
    return '********';
}

let password = getPassword('false'); // "password"
```

```
// Written with TypeScript
function getPassword(clearTextPassword: boolean) : string {
    if(clearTextPassword) {
        return 'password';
    }
    return '********';
}

let password = getPassword('false'); // throws: error TS2345: Argument of type '"false"' i
s not assignable to parameter of type 'boolean'.
```

DOM




```
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <h1>A Heading</h1>
    <a href>Link text</a>
  </body>
</html>
```

ANGULAR

- Web Components
  - Web components are a set of standard APIs that make it possible to natively create custom HTML tags that have their own functionality and component lifecycle. The main goal of web components is to encapsulate the code for the components into a nice, reusable package for maximum interoperability.
- RxJS
  - Reactive programming is an asynchronous programming paradigm concerned with data streams and the propagation of change. RxJS (Reactive Extensions for JavaScript) is a library for reactive programming using observables that makes it easier to compose asynchronous or callback-based code

Web Components

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="./main.js"></script>
</head>
<body>
    <show-text text="hello world"></show-text>
    <show-text text="Simple Example"></show-text>
</body>
</html>
```

```js
class ShowText extends HTMLElement {
    constructor() {
        super()
    }

    connectedCallback() {
        const text = this.getAttribute( qualifiedName: 'text');
        this.outerHTML = `<div style="color: red;">The content is: <label>${text}</label></div>`;

    }
}

customElements.define( name: 'show-text', ShowText);
```

ANGULAR

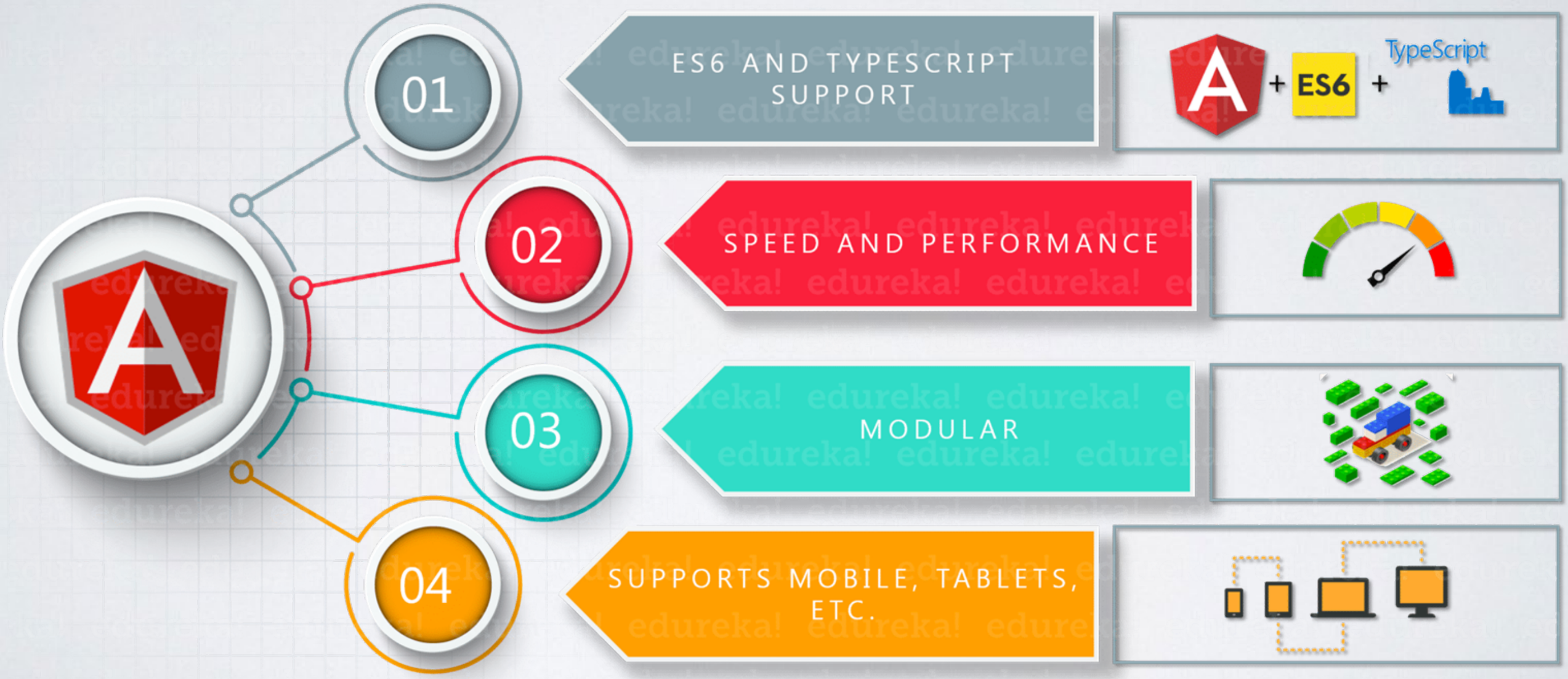
- Observable
  - Observables open up a continuous channel of communication in which multiple values of data can be emitted over time. From this we get a pattern of dealing with data by using array-like operations to parse, modify and maintain data. Angular uses observables extensively - you'll see them in the HTTP service and the event system.

ANGULAR

- Module Loaders
  - Modules allow code to be compartmentalized to provide logical separation for the developers. We need a way to have many Angular files in dev, but they need to be loaded into the browser in bulk (not a script tag for each one).
  - This is why people look to module loaders. Typescript uses the ES6 import and export syntax to load modules. Angular uses following Module loaders
    - SystemJS
    - Webpack
    - RequireJS

ANGULAR FEATURES

edureka!



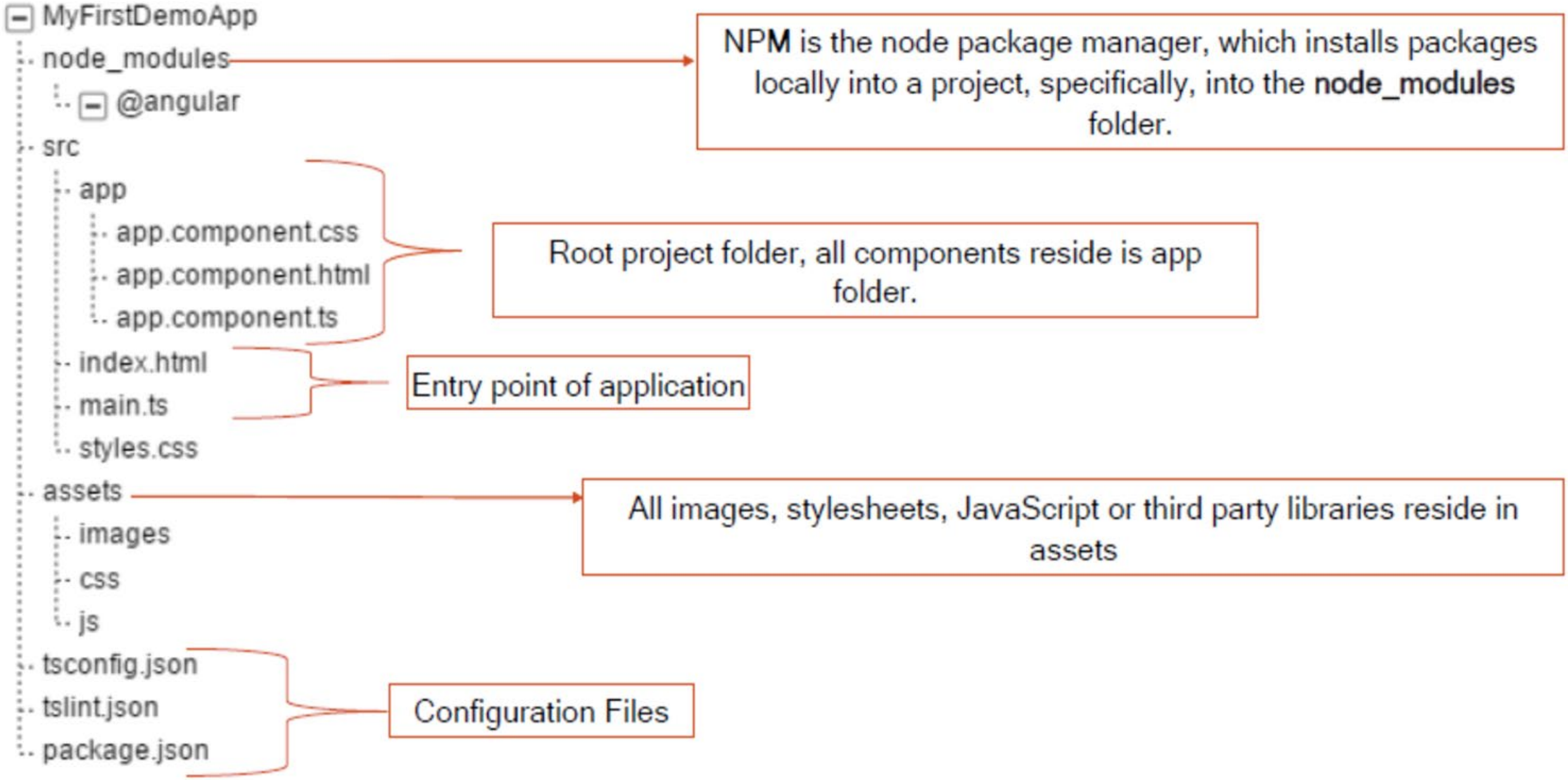
ENVIRONMENT SETUP

- The first step is to install Node.js and npm
  - https://nodejs.org/en/download
    - Node.js is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side.
    - Download latest version of Node.js
- Why it is Required
  - We are going to use NPM to install Angular, TypeScript, SystemJS and any other packages/modules required by our application. NPM is can be used to upgrade these packages as and when necessary. Without NPM, we have to download and install all these packages manually.

ENVIRONMENT SETUP

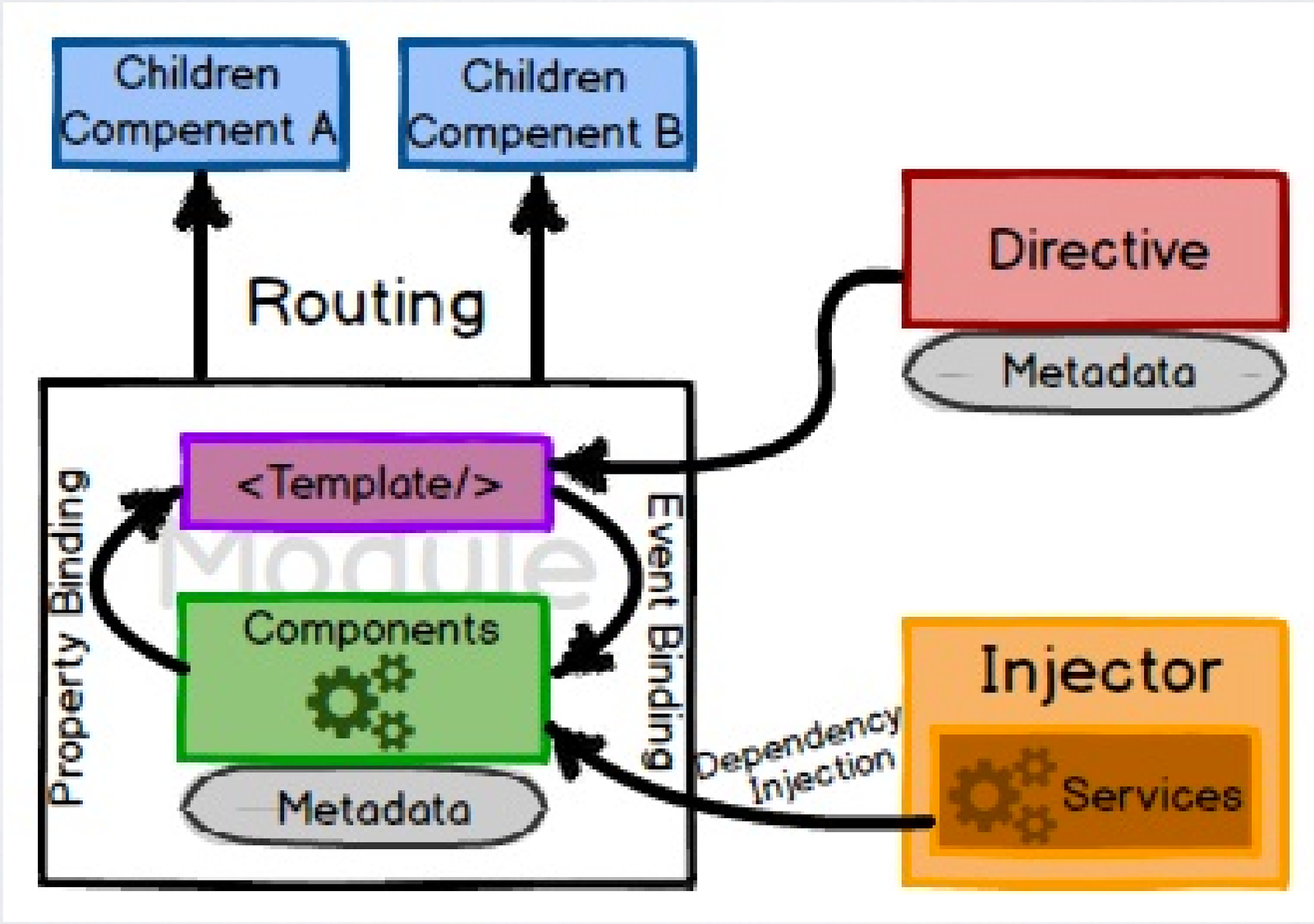
- Check version of Node.js and npm.
  - Syntax : node –v
  - Syntax : npm –v
- Install Angular CLI
  - Syntax: npm install --save-dev @angular/cli@latest
- Create first Angular App
  - ng new my-first-app
  - cd my-first-app
  - ng serve

ANGULAR PROJECT



ANGULAR ARCHITECTURE

- The Architecture of an Angular Application is based on the idea of Components.



MODULE

- An Angular module is a container for a group of related components, services, directives, and so on.
- All elements of a small application can be located in one module (the root module), whereas larger apps may have more than one module.
- All apps must have at least a root module that is bootstrapped during the app launch.

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

MODULE

- The @ngModule Metadata above has four fields.
  - Imports Metadata — tells the angular list of other modules used by this module.(Import)
  - Declaration Metadata — lists the components, directives & pipes that are part of this module. (Bean)
  - Providers — are the services that the other components can use. (DI)
  - Bootstrap Metadata — identifies the root component of the module. When Angular loads the appModule it looks for bootstrap Metadata and loads all the components listed here. We want our module to load AppComponent , hence we have listed it here. (@SpringBootApplication)

MODULE

- NgModules and JavaScript modules
  - The NgModule system is different from and unrelated to the JavaScript (ES2015) module system for managing collections of JavaScript objects. These are complementary module systems that you can use together to write your apps.
  - In JavaScript each file is a module and all objects defined in the file belong to that module. The module declares some objects to be public by marking them with the export keyword. Other JavaScript modules use import statements to access public objects from other modules.

```
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';


export class AppModule { }
```

Angular libraries

For example, import Angular's Component decorator from the @angular/core library like this.

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

You also import NgModules from Angular libraries using JavaScript import statements. For example, the following code imports the BrowserModule NgModule from the platform-browser library.

```
import { BrowserModule } from '@angular/platform-browser';
```

In the example of the simple root module above, the application module needs material from within BrowserModule. To access that material, add it to the @NgModule metadata imports like this.

```
imports:      [ BrowserModule ],
```

Frequently used modules

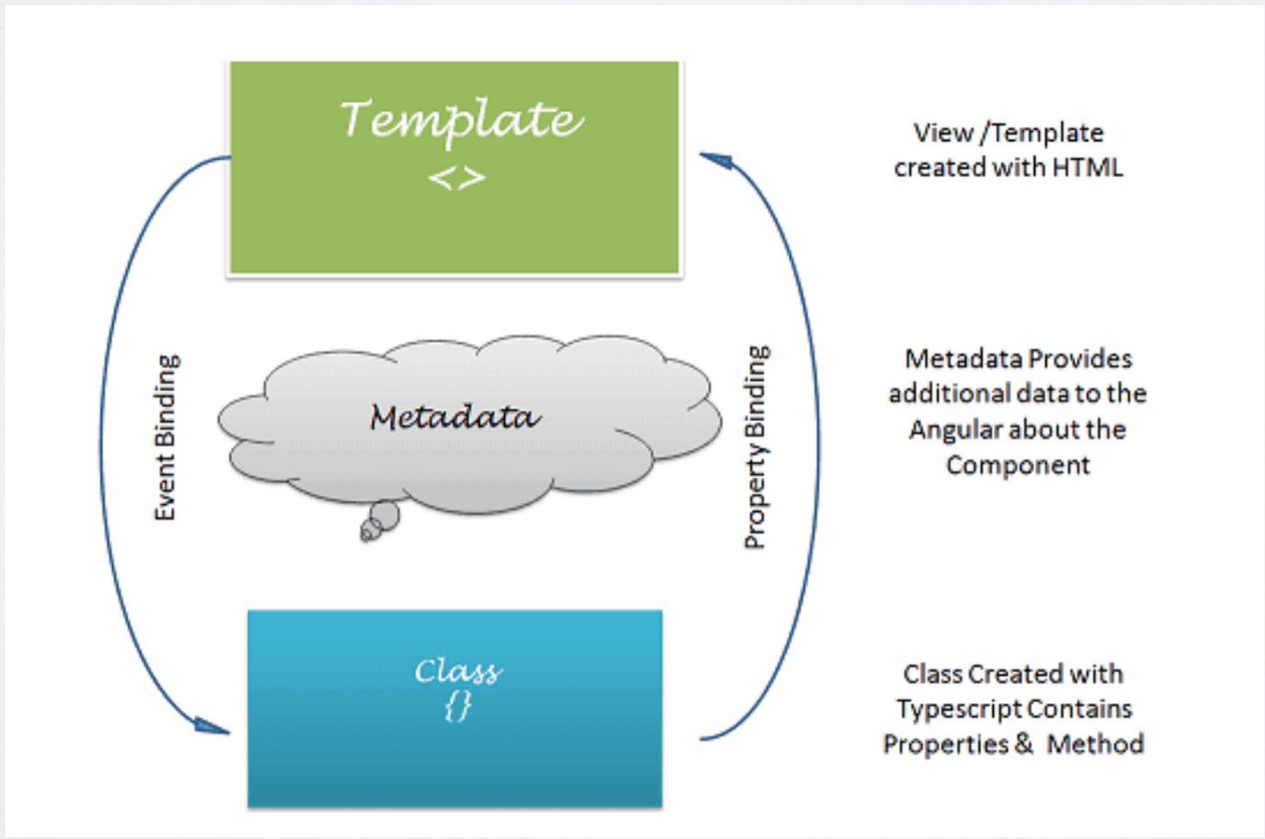
NgModule	Import it from	Why you use it
`BrowserModule`	`@angular/platform-browser`	When you want to run your app in a browser
`CommonModule`	`@angular/common`	When you want to use `NgIf`, `NgFor`
`FormsModule`	`@angular/forms`	When you want to build template driven forms (includes `NgModel`)
`ReactiveFormsModule`	`@angular/forms`	When you want to build reactive forms
`RouterModule`	`@angular/router`	When you want to use `RouterLink`, `.forRoot()`, and `.forChild()`
`HttpClientModule`	`@angular/common/http`	When you want to talk to a server

COMPONENT

- The Component is the main building block of an Angular Application.

- The Components consists of three main building blocks

Template, Class and Metadata



COMPONENT

- The Angular Components are plain JavaScript classes and defined using @component Annotation.
  - This Decorator provides the component with the View to display & Metadata about the class
- An app must have at least one module and one component, which is called the root component.

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'first-demo-app';
}
```

COMPONENT

- Each @Component annotation must define selector and template (or templateUrl) properties, which determine how the component should be discovered and rendered on the page.

- Properties and methods of the component class are available to the template

- The selector property is similar to a CSS selector. Each HTML element that matches the selector is rendered as an Angular component.
    - For example, if an app's HTML contains <app-hero-list></app-hero-list>, then Angular inserts an instance of the HeroListComponent view between those tags

- The template property defines user interface(UI)

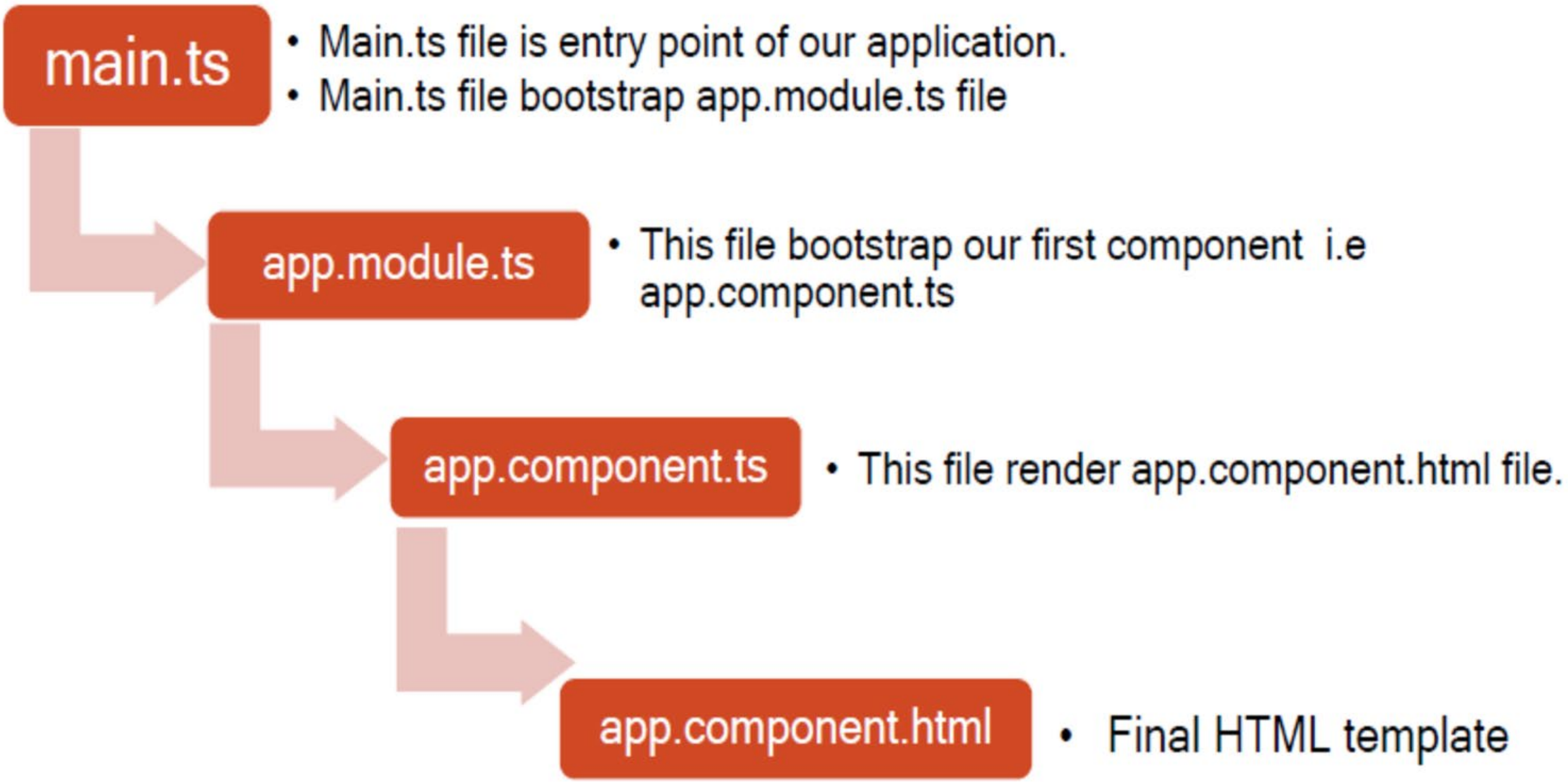
TEMPLATE

- A template is nothing but a form of HTML tags that tells Angular about how to render the component.
- A template looks like regular HTML, except for a few differences.
- Each component must define a view, which is specified in either a template or a templateUrl property of the @Component annotation
- Templates include data bindings as well as other components and directives
  - In-lined template in the component
    - *template=`<h1>This is Angular2 Demo</h1>`*
  - In an external HTML file
    - *templateUrl='template.html'*

METADATA

- Metadata tells Angular how to process a class. To tell that AppComponent is a component, metadata is attached to the class.
- In TypeScript, you attach metadata by using a decorator.
- With metadata we describe parts of the Angular application and bind those parts together:
- @NgModule decorator — Configures module
- @Component decorator — Configures component
- @Service decorator — Configures service
- @Pipe decorator — Configures pipe
- @Directive decorator — Configures directive

HOW ANGULAR START



DATA BINDING

- Data binding is communication between business logic (Component) and views(Template).
- It enables data to flow from the component to template and vice-versa.
- Binding can be used display component class property values to the user, change element styles, respond to a user event etc.
- The data-binding syntax lowers the amount of manual coding.
- The Data binding is not new to Angular. It existed in AngularJS., but the syntax has changed.
- Data binding includes *interpolation*, *property binding*, *event binding*, and *two-way binding*

INTERPOLATION

- Interpolation markup is used to provide data-binding to text and attribute values.

- The content inside the double braces is called *Template Expression* in Angular.

- Using {{ }} we display values from component into template.
  - Reference Variable in the component
  - Perform some mathematical operations
  - Concatenate two string
  - Invoke a method in the component

PROPERTY BINDING

- Property binding allow us to bind values to properties of an element to modify their behavior or appearance.

- This can include properties such as class, disabled, href or textContent.

- Using [ ] we bind valued from component to element’s property.

INTERPOLATION VS. PROPERTY BINDING

- Everything that can be done from Property binding can be done using the interpolation.

- But the only difference is that Interpolation requires expression to return a string. If you want to set an element property to a non-string data value, you must use property binding.

- interpolation, the button will always be disabled irrespective of isDisabled class property value is true of false.

```
<button disabled='{{isHide}}'>Try Me</button>
<br>
<button [disabled]='isHide'>Try Me</button>
```

EVENT BINDING

- When a user interacts with your app, it's sometimes necessary to know when this happens.

- A click, hover, or a keyboard action are all events that you can use to call component logic within Angular.

- The Name of the event is enclosed in ().

- Event Binding is one way from View to Component

TWO WAY BINDING

- Two way binding means send values from component to template, and returns changed values from template to component
- Using [()] we can achieve two way data binding
- Two-way data binding combines the property and event binding into a single notation using the ngModel directive.
- The ngModel directive is not part of the Angular Core library. It is part of the FormsModule library.
    - You need to import the FormsModule package into your Angular module

DIRECTIVES

- The Angular directive helps us to manipulate the DOM. You can change the appearance, behavior or a layout of a DOM element using the Directives. They help you to extend HTML.

- Directives are the most fundamental unit of Angular applications.

- There are three kinds of directives in Angular:

  - Structural Directives

  - Attribute Directives

  - Custom Directive

STRUCTURAL DIRECTIVES

- Structural directives can change the DOM layout by adding and removing DOM elements. All structural Directives are preceded by Star symbol

- *ngFor

  - The ngFor is an Angular 2 structural directive, which repeats a portion of HTML template once per each item from an iterable list (Collection).

  - The ngFor is similar to ngRepeat in AngularJS

LOCAL VARIABLES IN NGFOR

- ngFor also provides several values to help us manipulate the collection. We can assign the values of these exported values to the local variable and use it in our template.
  - index — This is a zero-based index and set to the current loop iteration for each template context.
  - first — This is a boolean value, set to true if the item is the first item in the iteration.
  - last — This is a boolean value, set to true if the item is the last item in the iteration.
  - even — This is a boolean value, set to true if the item is the even-numbered item in the iteration.
  - odd — This is a boolean value, set to true if the item is the odd-numbered item in the iteration.

STRUCTURAL DIRECTIVES

- ngSwitch
  - The ngSwitch directive lets you add/remove HTML elements depending on a match expression. ngSwitch directive used along with NgSwitchCase and NgSwitchDefault

```
<div [ngSwitch]="Switch_Expression">
    <div *ngSwitchCase="MatchExpression1"> First Template</div>
    <div *ngSwitchCase="MatchExpression2">Second template</div>
    <div *ngSwitchCase="MatchExpression3">Third Template</div>
    <div *ngSwitchCase="MatchExpression4">Third Template</div>
    <div *ngSwitchDefault?>Default Template</div>
</div>
```

STRUCTURAL DIRECTIVES

- *ngIf
  - The ngIf Directives is used to add or remove HTML Elements based on an expression. The expression must return a boolean value. If the expression is false then the element is removed, else the element is inserted.

```
<div *ngIf="condition">
    This is shown if condition is true
</div>
```

ATTRIBUTE DIRECTIVES

- An Attribute or style directive can change the appearance or behavior of an element.

- ngModel
  - The ngModel directive is used the achieve the two-way data binding.

ATTRIBUTE DIRECTIVES

- ngClass
  - The ngClass is used to add or remove the CSS classes from an HTML element. Using the ngClass one can create dynamic styles in HTML pages
- As we know, there is CSS pseudo which can be applied to, for example
  - Styles for identifying an element with the focus, via the :focus pseudo class
  - Hover styles and on-click active state styles (using :hover and :active)
- However, some styles are not actively supported by the browser:
  - styles for identifying the currently selected elements of a list
  - styles for identifying the currently active menu entry in a navigation menu
  - styles to identify a certain feature of a element; for example to identify a new element in an e-commerce site

ATTRIBUTE DIRECTIVES

- ngStyle
  - ngStyle is used to change the multiple style properties of our HTML elements. We can also bind these properties to values that can be updated by the user or our components.

```
<div [ngStyle]="{'color': 'blue', 'font-size': '24px', 'font-weight': 'bold'}">
    some text
</div>
```

CUSTOM DIRECTIVE

- The @Directive decorator allows you to attach custom behavior to an HTML element (for example, you can add an autocomplete feature to an <input> element).
  - Each component is basically a directive with an associated view, but unlike a component, a directive doesn't have its own view.
- The following example shows a directive that can be attached to an input element in order to log the input's value to the browser's console as soon as the value is changed.
- To bind events to event handlers, enclose the event name in parentheses. When the input event occurs on the host element, the onInput() event handler is invoked and the event object is passed to this method as an argument.

CUSTOM DIRECTIVE

- Here's an example of how you can create and attach the directive to an HTML element:

```
@Directive({
  selector: 'input[log-directive]',
  host: {
    '(input)': 'onInput($event)'
  }
})
class LogDirective {
  onInput(event) {
    console.log(event.target.value);
  }
}
```

```
<input type="text" log-directive/>
```

ANY QUESTIONS?