Anngular
===========
Anugular is a TypeScript based full-stack web framework for building web and mobile applications. One of the major advantage is that the Angular 8 support for web application that can fit in any screen resolution. Angular application is fully compatible for mobiles, tablets, laptops or desktops. Angular 8 has an excellent user interface library for web developers which contains reusable UI components.

This functionality helps us to create Single Page Applications (SPA). SPA is reactive and fast application. For example, if you have a button in single page and click on the button then the action performs dynamically in the current page without loading the new page from the server. Angular is Typescript based object oriented programming and support features for server side programming as well.

Angular framework is based on four core concepts and they are as follows:

ï · Components.
ï · Templates with Data binding and Directives.
ï · Modules.
ï · Services and dependency injection

Component
=========
The core of the Angular framework architecture is Angular Component. Angular Component is the building block of every Angular application.
Every angular application is made up of one more Angular Component.
It is basically a plain JavaScript / Typescript class along with a HTML template and an associated name.

The HTML template can access the data from its corresponding JavaScript / Typescript class.
Componentâ s HTML template may include other component using its selectorâ s value (name).
The Angular Component may have an optional CSS Styles associated it and the
HTML template may access the CSS Styles as well.

Template
========

Template is basically a super set of HTML. Template includes all the features of HTML and
provides additional functionality to bind the component data into the HTML and to dynamically generate HTML DOM elements.
The core concept of the template can be categorised into two items and they are as follows:

Data binding

Used to bind the data from the component to the template.
{{ title }}
Here, title is a property in AppComponent and it is bind to template using Interpolation.

Directives

Used to include logic as well as enable creation of complex HTML DOM elements.

Event binding

Events are actions like mouse click, double click, hover or any keyboard and mouse actions. If a user interacts with an application and performs some actions, then event will be raised. It is denoted by either parenthesis () or on-. We have different ways to bind an event to DOM element. Letâ s understand one by one in brief.

```
export class TestComponent {
showData($event: any){
 console.log("button is clicked!");
 if($event) {
 console.log($event.target);
 console.log($event.target.value);
 }
 }
}
```

alternatively :

Property Binding

Property binding is used to bind the data from property of a component to DOM elements. It is denoted by [].

```
 valueDisabled = true;
```

Attribute Binding

Attribute binding in Angular allows you to bind values to standard HTML attributes and custom attributes that don't have corresponding DOM properties.
While Angular usually prefers property binding for interacting with DOM properties, attribute binding can be useful when you need to work with attributes directly.
To perform attribute binding in Angular, you use square brackets [...] in your template, just like with property binding. Here's how to use attribute binding:

```
url = "www.javatpoint.com";
isDisabled = true;
```

Class binding

Class binding is used to bind the data from component to HTML class property. The
syntax is as follows:

```
  MyClass = "red";

  MyStyles  = true; // if value is true, blue color will be added to text, if it is false , blur color will not be added to
text.
```

```
        .red{
                color: red;
        }

        .blue{
                color: blue;
        }
```

Two-way data binding
=====================
Two-way data binding is a two-way interaction, data flows in both ways (from component to views and views to component). Simple example is ngModel. If you do any changes in your property (or model) then, it reflects in your view and vice versa. It is the combination of property and event binding.

```
  fullName : string | undefined = "Nani Babu Pallapu";
```

2. Angular-Directives
==================
The Angular 8 directives are used to manipulate the DOM.
By using Angular directives, you can change the appearance, behavior or a layout of a DOM element. It also helps you to extend HTML.

Angular 8 directives can be classified in 3 categories based on how they behave:

Component Directives
Structural Directives
Attribute Directives

1) Component Directives: Component can be used as directives. Every component has Input and Output option to pass between component and its parent HTML elements.
2) Structural Directives: Structural directives start with a * sign. These directives are used to manipulate and change the structure of the DOM elements.
For example, *ngIf directive, *ngSwitch directive, and *ngFor directive.

*ngIf Directive: gIf directive is used to display or hide data in your application based on the condition becomes true or false. We can add this to any tag in your template.
*ngSwitch Directive: The *ngSwitch allows us to Add/Remove DOM Element. It is similar to switch statement of C#.
*ngFor Directive: The *ngFor directive is used to repeat a portion of HTML template once per each item from an iterable list (Collection).

3) Attribute Directives: Attribute directives are used to change the look and behavior of the DOM elements. For example: ngClass directive, and ngStyle directive etc.

ngClass Directive: The ngClass directive is used to add or remove CSS classes to an HTML element.
ngStyle Directive: The ngStyle directive facilitates you to modify the style of an HTML element using the expression. You can also use ngStyle directive to dynamically change the style of your HTML element.

Structural Directives example:
==============================

```
    check : boolean = true;

    loggedAs : string  = "user";
```

Attribute Directives example:
===============================

in css :

```
.highlight
{
 color: red;
}
```

in html :

Component Directives
====================

In Angular, you can use the @Input and @Output decorators to pass data into and emit events from components. These decorators are essential for building parent-child communication between components.
Here's how you can use @Input and @Output decorators in Angular components:

Using @Input Decorator:
----

1. @Input Decorator Definition: @Input is used to pass data into a component from its parent component.
Parent Component:

2. In the parent component template or TypeScript file, you can bind data to a child component's input property using property binding.

Example (Parent Component HTML):

Example (Parent Component TypeScript):

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

@Component({
  selector: 'app-parent',
  template: `

  `,
})
export class ParentComponent {
  parentData = 'Data from parent';
}
```

In this example, the inputData property in the ChildComponent is decorated with @Input(), allowing it to receive data from the parent component.

Using @Output Decorator:
----
1. @Output Decorator Definition: @Output is used to emit events from a child component to its parent component.

2. Child Component: In the child component TypeScript file, you define an output property using the @Output decorator. You also create an EventEmitter to emit events.

Example (Child Component TypeScript):

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

@Component({
  selector: 'app-child',
  template: '',
})
export class ChildComponent {
  @Output() myEvent = new EventEmitter
```