Interpolation

Interpolation allows you to incorporate calculated strings into the text between HTML element tags and within attribute assignments. Template expressions are what you use to calculate those strings.

With interpolation, calculated values or variables can be displayed in the HTML template.

Interpolation {{...}}

Interpolation in Angular is used to display variables or calculated values in the template. This is done using the two curly braces `{{…}}` as shown In the following code snippet, {{currentUser}} and {{itemImageUrl}} is an example of interpolation

```
<h3>Current User: {{currentUser}}</h3>
<div><img src="{{itemImageUrl}}"></div>
```

Template Expressions

In Angular, template expressions are computations or assignments done in the template inside the interpolation curly braces. This expression is considered as local and only exist inside the template.

Complex expressions or assignment are discouraged to be done inside the template expressions as complex logics are advised to only be done in the components.

```
<p>The sum of 1 + 1 is {{1 + 1}}.</p>
<p>The sum of 1 + 1 is not {{1 + 1 + getVal()}}.</p>
```

Template statements

A template **statement** responds to an **event** raised by a binding target such as an element, component, or directive. Template statements are written in the format (event)="statement"

```
<button (click)="deleteUser()">Delete hero</button>
```

Binding syntax

Data binding is a mechanism for coordinating what users see, with application data values. While you could push values to and pull values from HTML, the application is easier to write, read, and maintain if you turn these chores over to a binding framework. You simply declare bindings between binding sources and target HTML elements and let the framework do the work.

Binding types can be grouped into three categories distinguished by the direction of data flow:

- *source-to-view*
- *view-to-source*
- *view-to-source-to-view*

source-to-view: One way, from the data source to the view data. The Syntax is this:

```
[target]="expression"
bind-target="expression"
```

view-to-source: One way, from the view target to the data source. The Syntax is this:

```
(target)="statement"
on-target="statement"
```

view-to-source-to-view: Two way, from the view target to the data source, and vice versa The Syntax is this:

```
[(target)]="statement"
bindon-target="expression"
```

The target name is the name of a property. It may look like the name of an attribute but it never is. To appreciate the difference, you must develop a new way to think about template HTML.

Structural directives

Structural directives alter layout by adding, removing, and replacing elements in the DOM. The example template uses two built-in structural directives to add application logic to how the view is rendered

<li *ngFor="let item of items; index as i; trackBy: trackByFn">...

```
<li *ngFor="let hero of heroes"></li>
<app-hero-detail *ngIf="selectedHero"></app-hero-detail>
```

Track in ngfor

To avoid this expensive operation, you can customize the default tracking algorithm. by supplying the `trackBy` option to `NgForOf`. `trackBy` takes a function that has two arguments: `index` and `item`. If `trackBy` is given, Angular tracks changes by the return value of the function.

NgIf :

```
<div *ngIf="condition">Content to render when condition is true. </div>
```

NgSwitch :

```
<container-element [ngSwitch]="switch_expression">
```

NgSwitch Casese :

```
<container-element [ngSwitch]="switch_expression">

   <some-element
*ngSwitchCase="match_expression_1">...</some-element>

...

   <some-element *ngSwitchDefault>...</some-element>

</container-element>
```