*Clase 04. INTERPOLACIÓN Y DIRECTIVAS*

INTERPOLACIÓN & EXPRESIONES

Una de las características fundamentales en Angular es separar la vista del modelo de datos 👀. En el modelo de datos tenemos las variables y en la vista implementamos cómo se muestran dichos datos.

Esto quiere decir que las variables que tengamos declaradas a nivel global en el componente pueden ser utilizadas en la vista ya sea para mostrarlas directamente como para operar con ellas 🙀.

**Interpolación.** Veamos un ejemplo de interpolación para mostrar el valor de una variable 👇

//componente.ts

export class InterpolacionComponent {

constructor() {}

nombre: string = ‘Marta’;

}

//componente.html

<p>Nombre: {{ nombre }}</p>  
  
Y ahora, interpolación para trabajar con una variable 👇  
//componente.ts

export class InterpolacionComponent {

constructor() {}

edad: number = 30;

}  
//componente.html

<p>Este año Marta cumplirá {{ edad + 1 }} años</p>

DIRECTIVAS

Las directivas son un tipo de componente que nos brinda Angular para manipular la vista HTML.

Existen varios tipos de directivas: las que actúan sobre un atributo de un elemento HTML, las estructurales, condicionales, data binding y personalizadas. En estas últimas nos enfocaremos especialmente 🚀.  
Fuente: [Angular - TypeScript](https://books.google.com.ar/books?id=6aHTDwAAQBAJ&pg=PA22&lpg=PA22&dq=Las+directivas+son+un+tipo+de+componente+que+nos+brinda+Angular+para+manipular+la+vista+HTML.+Existen+varios+tipos+de+directivas:+Las+que+act%C3%BAan+sobre+un+atributo+de+un+elemento+HTML+Las+estructurales+Las+personalizadas,+que+nosotros+desarrollamos&source=bl&ots=MH8LhWJNPz&sig=ACfU3U0q1DffoFI6czZlcPdXHDXvkSRKIw&hl=es&sa=X&ved=2ahUKEwisz-DR4NHzAhXZlZUCHdu6DVoQ6AF6BAgMEAM)

**TIPOS DE DIRECTIVAS  
De atributo**

\*ngClass

\*ngStyle

La sintaxis de estas directivas es

[directiva]=”...“

Del lado izquierdo ponemos la directiva que queremos utilizar encerrada entre corchetes [...]

Del lado derecho, el parámetro que estará procesando.

**NgClass**

Permite asignar dinámicamente la clase CSS a un elemento HTML. En este ejemplo podemos ver que, de acuerdo con la propiedad del componente asociado a la vista llamada hayError de tipo boolean (true|false), se asignará la clase CSS error o success al <div> (Puciarelli, 2020, p. 22)

<div [*ngClass*]="hayError ? 'error' : 'success'">Resultado obtenido</div>

**NgStyle**

Permite asignar dinámicamente un estilo específico a un elemento HTML.

Podemos ver que el estilo font-size del <div> se va a asignar de acuerdo con la propiedad del componente llamada size de tipo boolean (true|false)

<div [*style.font-size*]="size ? 'x-large' : 'smaller'">

Mensaje en letra grande o pequeña

</div>

**Fuente:** [Directivas y Bindings](https://books.google.com.ar/books?id=6aHTDwAAQBAJ&pg=PA22&lpg=PA22&dq=Las+directivas+son+un+tipo+de+componente+que+nos+brinda+Angular+para+manipular+la+vista+HTML.+Existen+varios+tipos+de+directivas:+Las+que+act%C3%BAan+sobre+un+atributo+de+un+elemento+HTML+Las+estructurales+Las+personalizadas,+que+nosotros+desarrollamos&source=bl&ots=MH8LhWJNPz&sig=ACfU3U0q1DffoFI6czZlcPdXHDXvkSRKIw&hl=es&sa=X&ved=2ahUKEwisz-DR4NHzAhXZlZUCHdu6DVoQ6AF6BAgMEAM)

**Directivas personalizadas**

Puede ocurrir que todas las directivas ofrecidas por Angular no se adapten a lo que necesitamos resolver. Para estos casos, podremos desarrollar nuestras propias directivas 😍.

Crear una directiva es muy similar a la creación de un componente, solo que el decorador utilizado será @Directive.

¿Cómo crear una directiva personalizada? 👣 Veamos el paso a paso:

📄 app.module.ts

*import* Directive, ElementRef, Renderer2 *from* '@angular/core';

@*Directive*(

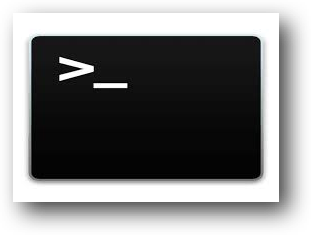
selector: '[appRoundBlock]'

)

*export* class RoundBlockDirective

constructor(renderer: Renderer2, elmRef: ElementRef)

renderer.setStyle(elmRef.nativeElement, 'border-radius', '100px');

ng generate directive roundBlock

📄 app.module.ts

*import* BrowserModule *from* '@angular/platform-browser';

*import* NgModule *from* '@angular/core';

*import* AppComponent *from* './app.component';

*import* RoundBlockDirective *from* './round-block.directive';

@*NgModule*(

declarations: [AppComponent, RoundBlockDirective],

imports: [BrowserModule],

providers: [],

bootstrap: [AppComponent]

)

*export* class AppModule

Hacemos uso de la directiva de atributo presente en la plantilla como se muestra a continuación:

📄 app.module.ts



<div *class*="demo-block" *appRoundBlock*></div>

Fuente: [Angular - TypeScript](https://books.google.com.ar/books?id=6aHTDwAAQBAJ&pg=PA22&lpg=PA22&dq=Las+directivas+son+un+tipo+de+componente+que+nos+brinda+Angular+para+manipular+la+vista+HTML.+Existen+varios+tipos+de+directivas:+Las+que+act%C3%BAan+sobre+un+atributo+de+un+elemento+HTML+Las+estructurales+Las+personalizadas,+que+nosotros+desarrollamos&source=bl&ots=MH8LhWJNPz&sig=ACfU3U0q1DffoFI6czZlcPdXHDXvkSRKIw&hl=es&sa=X&ved=2ahUKEwisz-DR4NHzAhXZlZUCHdu6DVoQ6AF6BAgMEAM)

**Directivas estructurales**

Son aquellas que permiten generar código HTML a partir de un conjunto de valores 😎. Para referirnos a este tipo de directivas, siempre tenemos que anteponer un \* (asterisco).

**\*ngFor**

Se utiliza para iterar sobre colecciones de objetos.

Aquí, por ejemplo, se muestran todas las frutas del arreglo declarado en línea (in-line).

<ul>

<li \**ngFor*="let fruta of ['Banana', 'Pera', 'Naranja']">

{{fruta}}

</li>

</ul>

**Directivas condicionales**

**\*NgIf**

Sirve para mostrar u ocultar el elemento HTML que la SE referencie junto con sus elementos hijos. Vemos que el <div> y su contenido se mostrarán según el valor de la propiedad mostrarDiv de tipo boolean del componente asociado a la vista 👇



<div \**ngIf*="mostrarDiv">

<span>Este texto se mostrará de acuerdo al valor de mostrarDiv</span>

<p>

<span>Este texto también se mostrará de acuerdo al valor de mostrarDiv</span>

</p>

</div>

**\*NgSwitch**

Esta directiva es similar a la sentencia switch de JavaScript, pero el

resultado dependerá del valor que tenga la propiedad valorSwitch del componente asociado a la vista.

<div [*ngSwitch*]="valorSwitch">

<span \**ngSwitchCase*="'valor1'">Valor 1</span>

<span \**ngSwitchCase*="'valor2'">Valor 2</span>

<span \**ngSwitchCase*="'valor3'">Valor 3</span>

</div>

**Directivas binding**

Cada componente contiene la lógica de la porción de la página que representa y, además, tiene asociada la vista HTML 📝.

El proceso de vincular o comunicar un componente (class) con la vista HTML se llama binding.

Los elementos que nos ofrece Angular son:

* template expression
* template statements
* property binding.

**TEMPLATE EXPRESSION**

Nos permiten reemplazar en tiempo de ejecución una referencia dentro del HTML con un valor de una propiedad del componente que se llama interpolación 🎯. La forma más simple de implementar la técnica de binding en Angular es con la interpolación.

Mediante la inserción del nombre de una propiedad de un componente

{{ propiedad }} dentro de la vista, así podremos embeber un string que se actualizará dinámicamente según el valor que va tomando la propiedad.

📄interpolacion.components.ts

*import* { Component, OnInit } *from* "@angular/core";

@*Component*({

selector: 'app.interpolacion',

templateUrl: './interpolacion.component.html',

styleUrls: './interpolacion.component.css'

})

*export* class interpolacionComponent *implements* OnInit {

*public* valor: string = 'Prueba de Interpolacion';

constructor() {}

*ngOnInit*() {}

}

📄interpolacion.components.ts

<h2>Ejemplo de interpolación</h2>

<h3>{{ valor }}</h3>

Si observamos el código del componente, notaremos que está formado por una clase class y su decorador @Component. Estas dos palabras reservadas de TypeScript serán suficientes y necesarias para definir un componente🙌.

Dicho componente cuenta con una propiedad de tipo string llamada valor. Por el lado de la vista, vemos el código HTML estándar al cual se le agregan las dobles llaves {{ valor }} para indicar la interpolación y leer la propiedad.

**TEMPLATE STATEMENTS**

Permiten relacionar un evento generado por nuestra vista, con una función de nuestro componente.

Al igual que template expressions, se inserta en la vista HTML y va asociado a un tag HTML (Puciarelli, 2020, p.27).

La sintaxis es:

<tag ...(origenEvento)=”template\_statement()” ...>

**TEMPLATE STATEMENTS**Permiten relacionar un evento generado por nuestra vista, con una función de nuestro componente. Al igual que template expressions, se inserta en la vista HTML y va asociado a un tag HTML (Puciarelli, 2020, p.27). La sintaxis es la siguiente:

<tag ...(origenEvento)=”template\_statement()” ...>



<button *(click)*=”guardarCambios()”>Guardar</button>

En este caso podemos ver la relación creada entre el evento clic del botón con la función guardarCambios del componente la cual sería nuestro template statement 😉.

<form *#formDatos* *(ngSubmit)*=”guardarForm(formDatos)”>

...

</form>

Este caso se diferencia del anterior porque, además de invocar a la función guardarForm, le estamos pasando como parámetro los datos del formulario mediante la variable de tipo Template reference llamada formDatos.

**PROPERTY BINDING**

Permite relacionar una propiedad de un objeto del DOM (Document Object Model) con nuestro componente. Esta relación puede ser en un solo sentido (one-way) o en ambos (two-way).

**one-way.** La relación en una dirección será desde una propiedad de nuestro componente, a una propiedad de un objeto del DOM

**two-way.** Los bindings bidireccionales se utilizan para compartir datos entre el componente y su vista. Incluyen, además, se usan para actualizar el modelo a medida que interactuamos con la vista.

**one-way**

📄component.ts

*export* class CompComponent *implements* OnInit {

urlImagen = 'https://afa.com/Lionel\_Messi.jpg';

constructor() {}

*ngOnInit*() {}

}

📄vista.html

<img [*src*]="urlImagen">

**two-way**

📄componente.ts

…

…

*verDatos*() {

console*.log*('Nombre: ', this*.*nombre);

console*.log*("Apellido: ", this*.*apellido)

}

*cargarDatos*() {

this*.*nombre = "Lionel"

this*.*apellido = "Messi"

}

📄component.vista.html

<p>

<label *for*="nombre">Nombre: </label>

<input *id*="nombre" [(*ngModel*)]="nombre" />

</p>

<p>

<label *for*="apellido">Apellido: </label>

<input *id*="apellido" [(*ngModel*)]="apellido" />

</p>

<button (*click*)="*verDatos*()">Mostrar</button>

<button (*click*)="*cargarDatos*()">Cargar</button>

**Componente**  
El ejemplo anterior contiene dos propiedades nombre y apellido que luego serán utilizadas para comunicarse con la vista. También están las funciones verDatos y cargarDatos

**Vista**

Los <input> con id nombre y apellido están vinculados a las propiedades nombre y apellido respectivamente, mediante la notación [(...)]

vista

FILTROS DE VISTA

**PIPES**

Son pequeñas funciones separadas a un componente que te servirá en cualquier parte de la web.

Las vamos a utilizar en 1 o varios componentes para no programar varias veces la misma función o crear un componente solo para esto una tarea (Flores, 2018).

Además de invocar a la función guardarForm, le estamos pasando como parámetro los datos del formulario mediante la variable de tipo Template reference llamada formDatos:

📄component.ts

birthday = new Date(1988, 3, 15);

📄component.html

<p>Mi cumpleaño es el {{ birthday | *date* }}</p>

**¿QUIERES SABER MÁS? TE DEJAMOS MATERIAL AMPLIADO DE LA CLASE**

[Angular - TypeScript](https://books.google.com.ar/books?id=6aHTDwAAQBAJ&pg=PA22&lpg=PA22&dq=Las+directivas+son+un+tipo+de+componente+que+nos+brinda+Angular+para+manipular+la+vista+HTML.+Existen+varios+tipos+de+directivas:+Las+que+act%C3%BAan+sobre+un+atributo+de+un+elemento+HTML+Las+estructurales+Las+personalizadas,+que+nosotros+desarrollamos&source=bl&ots=MH8LhWJNPz&sig=ACfU3U0q1DffoFI6czZlcPdXHDXvkSRKIw&hl=es&sa=X&ved=2ahUKEwisz-DR4NHzAhXZlZUCHdu6DVoQ6AF6BAgMEAM#v=onepage&q=Las%20directivas%20son%20un%20tipo%20de%20componente%20que%20nos%20brinda%20Angular%20para%20manipular%20la%20vista%20HTML.%20Existen%20varios%20tipos%20de%20directivas%3A%20Las%20que%20act%C3%BAan%20sobre%20un%20atributo%20de%20un%20elemento%20HTML%20Las%20estructurales%20Las%20personalizadas%2C%20que%20nosotros%20desarrollamos&f=false) | Luciano Puciarelli

Resumen de lo visto en clase hoy:

* Interpolación & Expresiones
* Directivas: De atributo, personalizadas, estructurales, condicionales y de binding.
* Filtros de vista & Pipes.