

Curso Avanzado de Angular

Plataforma: Udemy | Instructor: Fernando Herrera | Fecha: Marzo 2023

Nota del Autor:

El siguiente documento constituye una exhaustiva recopilación de los conocimientos y prácticas derivados del curso de Angular ofrecido en la plataforma Udemy. Su propósito fundamental radica en servir como un recurso detallado y de fácil acceso para futuras referencias en mi trayectoria profesional.

Además de presentar el código proporcionado por el instructor, este documento incluye explicaciones detalladas de ciertos conceptos que, aunque se abordaron en el curso de Angular Avanzado, no fueron explorados en profundidad. Se ha procurado enriquecer la comprensión de estos temas mediante análisis más detallados. Esta extensión va más allá de la enseñanza estándar del curso, proporcionando una perspectiva más completa y facilitando la asimilación de conceptos clave. Así, este material no solo actúa como una recopilación de lo aprendido, sino también como un recurso complementario que busca ofrecer una comprensión más holística de los temas tratados en el curso.

Quisiera enfatizar que este material no tiene como finalidad generar lucro alguno. En lugar de ello, busca únicamente consolidar y mantener frescos los conocimientos adquiridos durante el curso. Es importante señalar que la mayor parte del código presente en este documento ha sido proporcionado por el instructor, **Fernando Herrera**. Solo en casos excepcionales se han incorporado modificaciones o funcionalidades adicionales como resultado de prácticas complementarias.

Esta recopilación se presenta como una herramienta personal, creada con el objetivo de fortalecer y consolidar los conceptos aprendidos en el curso de Angular Avanzado. Agradezco profundamente al instructor por compartir su experiencia y conocimientos, los cuales han sido fundamentales para mi desarrollo en esta tecnología.

Espero que este documento no solo sirva como recordatorio para mí, sino también como una fuente de conocimiento para otros estudiantes interesados en profundizar en Angular. Cabe destacar que cualquier beneficio derivado de este material debe ser atribuido principalmente al esfuerzo y dedicación del instructor y la plataforma Udemy, a quienes agradezco por facilitar este valioso aprendizaje.

Mas información Aquí: <https://www.udemy.com/course/angular-pro-siguiente-nivel/>

20/Septiembre 2024 - ...

CONTENIDO

Instalación

Angular Pro

Descargar esta hoja de atajos: [Guías de atajos - Angular](#)

1. [Node JS](#)
2. [VSCode - Visual Studio Code](#)
3. [Postman](#)
4. [Git](#)

```
git config --global user.name "Tu nombre"
git config --global user.email "Tu correo"
```

5. [Docker Desktop](#)

AngularCLI

Documentación [oficial de Angular CLI](#)

Ejecutar el siguiente comando como **administrador**

```
npm install -g @angular/cli
```

Extensiones de VSCode

- [Angular Language Service](#)
- [Angular Snippets](#)
- [Angular Schematics](#)
- [Angular 2 Inline](#)
- [Auto Close Tag](#)
- [Auto Rename Tag](#)
- [Console Ninja](#)
- [Error Lens](#)
- [Paste JSON as Code](#)
- [Editor Config for VSCode](#)

- [Better Comments](#)
- [Terminal](#)
- [Tailwind CSS IntelliSense](#)

Tema que estoy usando en VSCode y Wallpaper del curso:

- [Aura Theme](#)
- [Tokyo Night](#)
- [Tokyo Night Dark](#)
- [Material Icons](#)
- [Bearded Icons](#)
- [Wallpapers Developer](#)

Nueva Sección: Zoneless Calculator:

¿Qué veremos en esta sección?

En esta sección vamos a trabajar con una estructura HTML hecha en *Tailwind*, que nos enseñe los problemas estructurales a los que vamos a caer cuando queramos recrear un diseño en componentes de Angular.

Puntualmente veremos:

- Tailwind
- Zoneless
- OnPush
- ViewEncapsulation
- ng-deep (Deprecated)
- Content Projection
- input Signals
- Standalone components
- Angular Schematics
- Host bindings
- Entre otros temas

Nueva APP

```
$ ng new zoneless-calculator
```

Configurar Paths

Para hacer más fácil los imports, en el tsconfig.json

```
"compilerOptions": {  
  "baseUrl": ".",  
  "paths": {  
    "@/*": ["src/app/*"],  
    "@app/*": ["src/app/*"],  
  },  
}
```

Instalamos y configuramos **tailwindCss**

```
$ npm install -D tailwindcss postcss autoprefixer  
$ npx tailwindcss init
```

El init crea nuestro archivo de configuración ****tailwind.config.js****

Agregamos en el archivo **tailwind.config.js** los Paths a todos nuestros archivos de plantilla

```
content: [  
  "./src/**/*.{html,ts}",  
],
```

Agregamos en el **style.css** las directivas de tailwind

```
@tailwind base;  
@tailwind components;  
@tailwind utilities;
```

En el caso de obtener el error:

```
Unknown at rule @tailwindcss(unknownAtRules)
```

Consultar [Aca](#)

En VS Code:

File > Preference > Settings

Buscar **files.associations**

Agregar un nuevo ítem

Key: *.css

Value: tailwindcss

provideZoneChangeDetection vs provideExperimentalZonelessChangeDetection

Por default, Angular usa ZoneJS para la detección de cambios, para este ejercicio, usaremos un algoritmo de detección de cambios que no usa ZoneJS.

En el archivo App.Config.js eliminamos la línea

```
provideZoneChangeDetection({ eventCoalescing: true } ),
```

y usamos en su lugar **provideExperimentalZonelessChangeDetection**

```
import { ApplicationConfig, provideExperimentalZonelessChangeDetection }  
from '@angular/core';
```

```
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [
    provideExperimentalZonelessChangeDetection(),
    provideRouter(routes)]
};
```

Luego del cambio veremos este mensaje en el console

```
core.mjs:32762 NG0914: The application is using zoneless change detection,
but is still loading Zone.js. Consider removing Zone.js to get the full
benefits of zoneless. In applications using the Angular CLI, Zone.js is
typically included in the "polyfills" section of the angular.json file.
```

Para eliminar por completo el Zone.js, en el angular.json, buscamos las referencias a zone.js y las removemos, por ejemplo

```
"polyfills": [
  "zone.js",
  "zone.js/testing"
],
```

Lo cambiamos por:

```
"polyfills": []
```

Crear estructura base:

Creamos directorios:

```
├── calculator
│   ├── components
│   ├── services
│   └── views
```

View, es lo que normalmente conocemos como pages, componentes de páginas completas que son usados en los routers y que agrupan otros componentes.

Creamos el primer Componente

```
$ ng g c calculator/views/calculatorView
```

Rutas

En el app.routes.ts agregamos

```
import { Routes } from '@angular/router';

export const routes: Routes = [
  {
    path: 'calculator',
    loadComponent: () => import('@app/calculator/views/calculator-view/calculator-view.component'),
  },
  {
    path: '**',
    redirectTo: 'calculator',
  }
];
```

Dos cosas a notar: Primero estamos usando el **@app/** del path configurado en el tsconfig.ts file, y lo segundo es que para que este import funcione, necesitamos agregar el key **default** en la definición de la clase del componente:

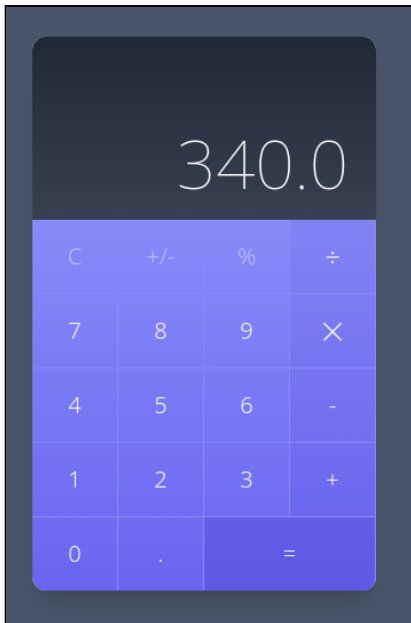
```
export default class CalculatorViewComponent {

}
```

Dado que el componente no está definido dentro de un módulo, podemos importarlo directamente de esa forma.

Diseño

El código del template descargado del sitio del curso, genera la siguiente calculadora



Contiene mucho código repetido que vamos a transformar en componentes. El template completo es el siguiente:

```
<div class="min-w-screen min-h-screen bg-gray-100 flex items-center
justify-center px-5 py-5">
  <div class="w-full mx-auto rounded-xl bg-gray-100 shadow-xl text-gray-
800 relative overflow-hidden" style="max-width:300px">
    <div class="w-full h-40 bg-gradient-to-b from-gray-800 to-gray-700
flex items-end text-right">
      <div class="w-full py-5 px-6 text-6xl text-white font-
thin">340.0</div>
    </div>
    <div class="w-full bg-gradient-to-b from-indigo-400 to-indigo-500">
      <div class="flex w-full">
        <div class="w-1/4 border-r border-b border-indigo-400">
          <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-opacity-50
text-xl font-light">C</button>
        </div>
        <div class="w-1/4 border-r border-b border-indigo-400">
          <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-opacity-50
text-xl font-light">+/-</button>
        </div>
        <div class="w-1/4 border-r border-b border-indigo-400">
          <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-opacity-50
text-xl font-light">%</button>
        </div>
        <div class="w-1/4 border-r border-b border-indigo-400">
          <button class="w-full h-16 outline-none focus:outline-
none bg-indigo-700 bg-opacity-10 hover:bg-opacity-20 text-white text-2xl
font-light">÷</button>
        </div>
      </div>
    </div>
  </div>
```

```

        <div class="flex w-full">
            <div class="w-1/4 border-r border-b border-indigo-400">
                <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-xl font-
light">7</button>
            </div>
            <div class="w-1/4 border-r border-b border-indigo-400">
                <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-xl font-
light">8</button>
            </div>
            <div class="w-1/4 border-r border-b border-indigo-400">
                <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-xl font-
light">9</button>
            </div>
            <div class="w-1/4 border-r border-b border-indigo-400">
                <button class="w-full h-16 outline-none focus:outline-
none bg-indigo-700 bg-opacity-10 hover:bg-opacity-20 text-white text-xl
font-light">X</button>
            </div>
        </div>
        <div class="flex w-full">
            <div class="w-1/4 border-r border-b border-indigo-400">
                <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-xl font-
light">4</button>
            </div>
            <div class="w-1/4 border-r border-b border-indigo-400">
                <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-xl font-
light">5</button>
            </div>
            <div class="w-1/4 border-r border-b border-indigo-400">
                <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-xl font-
light">6</button>
            </div>
            <div class="w-1/4 border-r border-b border-indigo-400">
                <button class="w-full h-16 outline-none focus:outline-
none bg-indigo-700 bg-opacity-10 hover:bg-opacity-20 text-white text-xl
font-light">-</button>
            </div>
        </div>
        <div class="flex w-full">
            <div class="w-1/4 border-r border-b border-indigo-400">
                <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-xl font-
light">1</button>
            </div>
            <div class="w-1/4 border-r border-b border-indigo-400">
                <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-xl font-
light">2</button>

```

```

        </div>
        <div class="w-1/4 border-r border-b border-indigo-400">
            <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-xl font-
light">3</button>
        </div>
        <div class="w-1/4 border-r border-b border-indigo-400">
            <button class="w-full h-16 outline-none focus:outline-
none bg-indigo-700 bg-opacity-10 hover:bg-opacity-20 text-white text-xl
font-light">+</button>
        </div>
    </div>
    <div class="flex w-full">
        <div class="w-1/4 border-r border-indigo-400">
            <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-xl font-
light">0</button>
        </div>
        <div class="w-1/4 border-r border-indigo-400">
            <button class="w-full h-16 outline-none focus:outline-
none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-xl font-
light">.</button>
        </div>
        <div class="w-2/4 border-r border-indigo-400">
            <button class="w-full h-16 outline-none focus:outline-
none bg-indigo-700 bg-opacity-30 hover:bg-opacity-40 text-white text-xl
font-light">=</button>
        </div>
    </div>
</div>
</div>
</div>
</div>

```

Un componente Button que reciba el texto a desplegar y algunas clases podría evitar esta repetición de código.

Componentes

Creemos:

```
$ ng g c calculator/components/calculator
```

Crearemos la siguiente estructura de componentes

```

├─ App-Component
│   └─ Calculator-View
│       └─ Calculator
│           ├── Buttons
│           └─ Other Components

```

Primeramente el App Component unicamente contiene el wrapper de toda la app.

```
<div class="min-w-screen min-h-screen bg-slate-600 flex items-center
justify-center px-5 py-5">
  <router-outlet></router-outlet>
</div>
```

Por medio de las rutas, el **router-outlet** mostrará el componente indicado, en este caso, el **path: 'calculator'**, mostrará el ****Calculator-View**

El template del Calculator-View es:

```
<div class="w-full mx-auto rounded-xl bg-gray-100 shadow-xl text-gray-800
relative overflow-hidden">
  <calculator></calculator>
</div>
```

En este caso tomamos el siguiente contenedor y dentro de este mostraremos el componente calculator

Calculator por el momento contiene todo lo demás del template original

Dado que estamos usando standalone component en este proyecto, no es necesario crear los módulos, simplemente creamos los componentes y los importamos donde sean necesarios

El Calculator Component

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

@Component({
  selector: 'calculator',
  standalone: true,
  imports: [],
  templateUrl: './calculator.component.html',
  styleUrls: ['./calculator.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class CalculatorComponent {

}
```

El Calculator-View Component, lo importa y lo usa en su template

```
import { CalculatorComponent } from
'@/calculator/components/calculator/calculator.component';
import { ChangeDetectionStrategy, Component } from '@angular/core';

@Component({
  selector: 'calculator-view',
  standalone: true,
  imports: [ CalculatorComponent],
  templateUrl: './calculator-view.component.html',
  styleUrls: ['./calculator-view.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export default class CalculatorViewComponent {

}
```

Cambios en la estructura del HTML

Al agregar componentes, el HTML original se renderiza con DIVS adicionales, tal como se nota en la siguiente imagen, esto puede generar que el diseño original cambie debido a que se rompe la secuencia de elementos hijos.

```
<app-root _ngcontent-ng-c2413495075 ng-version="18.2.5">
  <div _ngcontent-ng-c2413495075 class="min-w-screen min-h-screen bg-slate-50">
    <router-outlet _ngcontent-ng-c2413495075></router-outlet>
    <calculator-view _ngcontent-ng-c2972209345>
      <div _ngcontent-ng-c2972209345 class="w-full mx-auto rounded-xl bg-gradient-to-r from-blue-500 to-purple-500">
        <calculator _ngcontent-ng-c2972209345 _ngcontent-ng-c193239480>
          <div _ngcontent-ng-c193239480 class="w-screen h-40 bg-gradient-to-r from-blue-500 to-purple-500" style="max-width: 300px;"> ... </div>
          <div _ngcontent-ng-c193239480 class="w-full bg-gradient-to-b from-blue-500 to-purple-500"> ... </div>
        </calculator>
      </div>
    </calculator-view>
    <!--container-->
  </div>
</app-root>
```

Esto implica que a veces se tenga que modificar el HTML original para mostrarse tal como se hacía al inicio, antes de introducir componentes.

En amarillo se muestran los cambios aplicados para mantener el diseño original.

Al introducir los siguiente componentes, como los botones, vamos a tener el mismo escenario.

El problema con esta solución es que a veces no podremos cambiar el estilo facilmente, y si deseamos mantener el mismo diseño original, debemos buscar otra solución, aca es donde aparecen los host-components.

Host Element

La calculadora cuenta con una serie de botones:

```
<div class="w-1/4 border-r border-b border-indigo-400">
  <button
    class="w-full h-16 outline-none focus:outline-none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-opacity-50 text-xl font-light">
    C
  </button>
</div>
```

Transformar esto a un componente normal, introducirá DIV's adicionales que van a romper el estilo original.

Creamos el componente:

```
$ ng g c calculator/components/calculator-button
```

Copiamos (cortamo) el HTML del boton (incluyendo el DIV) y lo agregamos al template del nuevo componente

```
<div class="w-1/4 border-r border-b border-indigo-400">
  <button class="w-full h-16 outline-none focus:outline-none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-opacity-50 text-xl font-light">C</button>
</div>
```

Importamos el componente en el **CalculatorComponent**

```
import { ChangeDetectionStrategy, Component } from '@angular/core';
import { CalculatorButtonComponent } from '../calculator-button/calculator-button.component';

@Component({
  selector: 'calculator',
  standalone: true,
  imports: [CalculatorButtonComponent],
  templateUrl: './calculator.component.html',
  styleUrls: ['./calculator.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class CalculatorComponent {

}
```

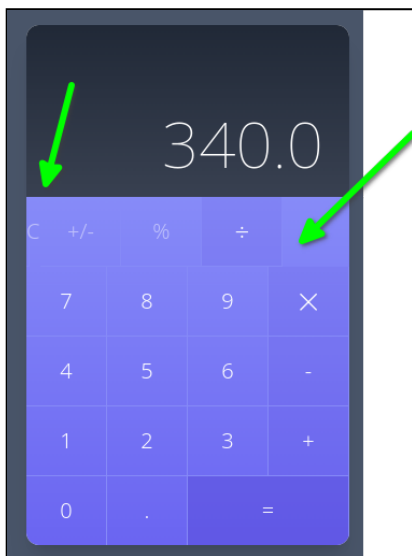
Y lo agregamos en lugar del HTML cortado.

```
<calculator-button></calculator-button>
```

Esto genera un DIV adicional que rompe el estilo

```
<div _ngcontent-ng-c3594720438 class="flex w-full"> (flex)
  <calculator-button _ngcontent-ng-c3594720438 _ngghost-ng-c742991328>
    <div _ngcontent-ng-c742991328 class="w-1/4 border-r border-b border-indigo-400"> ... </div>
  </calculator-button>
  <div _ngcontent-ng-c3594720438 class="w-1/4 border-r border-b border-indigo-400"> ... </div>
  <div _ngcontent-ng-c3594720438 class="w-1/4 border-r border-b border-indigo-400"> ... </div>
  <div _ngcontent-ng-c3594720438 class="w-1/4 border-r border-b border-indigo-400"> ... </div>
```

El resultado es que el boton "C" no ocupa el 25% del ancho (Clase **w-1/4**)



ngContent

La proyección de contenido es un patrón en el que se inserta o proyecta el contenido que se desea utilizar dentro de otro componente.

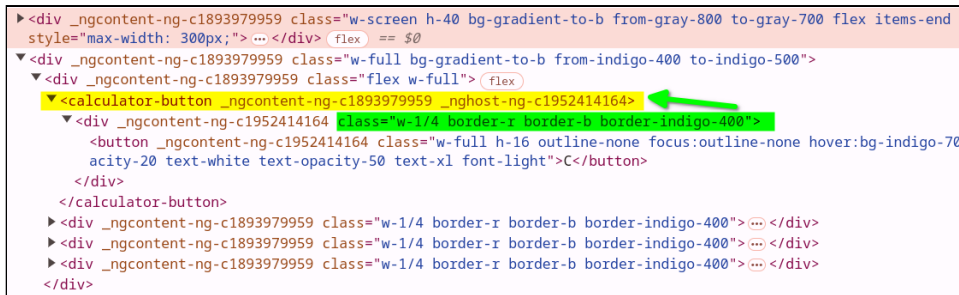
En nuestro ComponentButton podemos usar **ng-content**

```
<button class="...">
  <ng-content></ng-content>
</button>
```

Y llamarlo de esta forma:

```
<calculator-button>C</calculator-button>
```

Aun tenemos el problema original, pero ya podemos enviar el label del botón usando el ng-content, en este momento el HTML se ve de esta forma



```

<div _ngcontent-ng-c1893979959 class="w-screen h-40 bg-gradient-to-b from-gray-800 to-gray-700 flex items-end
style="max-width: 300px;">...</div> (flex) == $0
▼ <div _ngcontent-ng-c1893979959 class="w-full bg-gradient-to-b from-indigo-400 to-indigo-500">
  ▼ <div _ngcontent-ng-c1893979959 class="flex w-full"> (flex)
    ▼ <calculator-button _ngcontent-ng-c1893979959 _ngghost-ng-c1952414164>
      ▼ <div _ngcontent-ng-c1952414164 class="w-1/4 border-r border-b border-indigo-400">
        <button _ngcontent-ng-c1952414164 class="w-full h-16 outline-none focus:outline-none hover:bg-indigo-700
        acity-20 text-white text-opacity-50 text-xl font-light">C</button>
      </div>
    </calculator-button>
  </div>
  <div _ngcontent-ng-c1893979959 class="w-1/4 border-r border-b border-indigo-400">...</div>
  <div _ngcontent-ng-c1893979959 class="w-1/4 border-r border-b border-indigo-400">...</div>
  <div _ngcontent-ng-c1893979959 class="w-1/4 border-r border-b border-indigo-400">...</div>
</div>

```

Lo que necesitamos para tener el diseño original es que las clases marcadas en verde, puedan aplicarse al DIV generado por Angular en amarillo.

Para lograr esto, cambiaremos el template del boton

```

<div class="w-1/4 border-r border-b border-indigo-400">
  <button class="w-full h-16 outline-none focus:outline-none hover:bg-
indigo-700 hover:bg-opacity-20 text-white text-opacity-50 text-xl font-
light">
    <ng-content></ng-content>
  </button>
</div>

```

Eliminamos el DIV y dejamos únicamente el button, y las clases del DIV las vamos a agregar al componente directamente

```

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

@Component({
  selector: 'calculator-button',
  standalone: true,
  imports: [],
  templateUrl: './calculator-button.component.html',
  styleUrls: ['./calculator-button.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush,
  host: {
    class: 'w-1/4 border-r border-b border-indigo-400',
  },
})
export class CalculatorButtonComponent {

}

```

Notar como hemos agregado el host

```

host: {
  class: 'w-1/4 border-r border-b border-indigo-400',
},

```


De esta forma, el diseño origina se reestablece y el HTML queda de la siguiente manera

```
<div _ngcontent-ng-c1893979959 class="flex w-full">
  <calculator-button _ngcontent-ng-c1893979959 class="w-1/4 border-r border-b border-indigo-400" _ngghost-ng-c141004325>
    <button _ngcontent-ng-c141004325 class="w-full h-16 outline-none focus:outline-none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-opacity-50 text-xl font-light">C</button>
  </calculator-button>
  <div _ngcontent-ng-c1893979959 class="w-1/4 border-r border-b border-indigo-400">
    <button _ngcontent-ng-c1893979959 class="w-full h-16 outline-none focus:outline-none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-opacity-50 text-xl font-light">+/-</button>
  </div>
  <div _ngcontent-ng-c1893979959 class="w-1/4 border-r border-b border-indigo-400">
    <button _ngcontent-ng-c1893979959 class="w-full h-16 outline-none focus:outline-none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-opacity-50 text-xl font-light">%</button>
  </div>
  <div _ngcontent-ng-c1893979959 class="w-1/4 border-r border-b border-indigo-400">
    <button _ngcontent-ng-c1893979959 class="w-full h-16 outline-none focus:outline-none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-opacity-50 text-xl font-light">÷</button>
  </div>
</div>
```

Ahora podemos usar nuestro nuevo componente

```
<div class="flex w-full">
  <calculator-button>C</calculator-button>
  <calculator-button>+/-</calculator-button>
  <calculator-button>%</calculator-button>
  <calculator-button>÷</calculator-button>
</div>
```

Y de esta forma, hemos simplificado el HTML.

El único problema que tenemos es que el último botón, tiene un color de fondo diferente, y ahora mismo estamos aplicando las mismas clases a los botones por medio del host.

InputSignal y HostBidings

Antes de usar los Inputs y HostBiding moveremos el CSS del Boton a su CSS correspondiente

```
<button class="w-full h-16 outline-none focus:outline-none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-opacity-50 text-xl font-light">
  <ng-content></ng-content>
</button>
```

Todas las class dle button la agregamos al archivo **calculator-button.component.css**

```
button {
  @apply w-full h-16 outline-none focus:outline-none hover:bg-indigo-700
  hover:bg-opacity-20 text-white text-opacity-50 text-xl font-light
}
```

Luego aplicamos estos cambios en el componente

```
export class CalculatorButtonComponent {

  public isCommand = input(
    false, // default value
  )
}
```

```
    transform: (value: string) => typeof value === 'string' ? value ===
'' : value,
  }
)

@HostBinding('class.bg-indigo-700') get CommandStyle() {
  return this.isCommand();
}
}
```

Además de estos cambios en el typescript, ahora podemos usar el atributo en el componente

```
<calculator-button isCommand>÷</calculator-button>
```

Ahora el componente tiene un atributo llamado **isCommand**. Este atributo se usa para indicar que este botón es un "comando" en la calculadora (como los operadores ÷, ×, +, etc.), en lugar de un número.

```
public isCommand = input(
  false, // default value
  {
    transform: (value: string) => typeof value === 'string' ? value ===
'' : value,
  }
)
```

isCommand es una propiedad del componente que puede ser configurada desde el HTML cuando el componente es utilizado. Se está utilizando una función **input()** para definir el valor de esta propiedad.

false: Este es el valor predeterminado de **isCommand**, lo que significa que, si no se proporciona este atributo en el HTML, el botón no será tratado como un comando.

transform: Esta es una transformación que se aplica al valor de **isCommand** cuando se pasa desde el HTML. Si **isCommand** es una cadena vacía (""), se considerará como true (ya que en Angular, cuando un atributo se presenta sin valor, se lo trata como verdadero). Si no es una cadena vacía, entonces **isCommand** tomará el valor correspondiente.

De modo que esto `<calculator-button isCommand>÷</calculator-button>` definirá **isCommand** en true y esto `<calculator-button isCommand='false'>÷</calculator-button>` y también esto `<calculator-button>÷</calculator-button>` definirá **isCommand** en false.

Host Binding (Decorador @HostBinding):

El decorador **@HostBinding** vincula una clase CSS al componente cuando la propiedad **isCommand** es verdadera. En este caso, cuando el valor de **isCommand** es verdadero (true), se aplica la clase CSS **bg-indigo-700**.

El método **CommandStyle** usa **isCommand()** para determinar si debe aplicar la clase **bg-indigo-700**.

Multiples class en el HostBiding

Si necesitaramos agregar más clases al HostBiding, lo ideal es definir una clase en el CSS, por ejemplo:

```
.is-command {  
  @apply bg-indigo-700 bg-opacity-20 text-opacity-100  
}
```

View Encapsulation

El cambio anterior aplica la clase **is-command** al componente hosting, este es el HTML renderizado

```
<calculator-button _ngcontent-ng-c90236445="" iscommand="" class="w-1/4  
border-r border-b border-indigo-400 is-command" _nghost-ng-c4087423074=""  
ng-reflect-is-command="">  
  <button _ngcontent-ng-c4087423074="" class="w-full h-16 outline-none  
focus:outline-none hover:bg-indigo-700 hover:bg-opacity-20 text-white text-  
opacity-50 text-xl font-light">  
    ÷  
  </button>  
</calculator-button>
```

Recordemos que el ÷ está definido en el componente externo **Calculator-Component** mientras que el **is-command** lo estamos definiendo en a nivel del componente interno **Calculator-Button** por lo tanto, si bien agregamos la clase **is-command**, el estilo no se aplica, porque no está en su scope.

Una forma de solucionar es definir la clase **is-command** en el style.css global de la APP.

otra forma de solucionarlo es mantener la clase **is-command** en nuestro archivo **calculator-button.component.css** pero definirla con el atributo **::ng-deep**

```
::ng-deep .is-command {  
  @apply bg-indigo-700 bg-opacity-20 text-opacity-100  
}
```

Esto no se recomienda porque Angular ha deprecado ese atributo.

La otra opción es indicarle a Angular que ese componente no encapsule nada, esto lo logramos si agregamos **encapsulation: ViewEncapsulation.None**, a la definición del componente:

```
Component({  
  selector: 'calculator-button',  
  standalone: true,
```

```
imports: [],
templateUrl: './calculator-button.component.html',
styleUrl: './calculator-button.component.css',
changeDetection: ChangeDetectionStrategy.OnPush,
host: {
  class: 'w-1/4 border-r border-b border-indigo-400',
},
encapsulation: ViewEncapsulation.None,
})
```

Esto sigue siendo una solución no ideal, porque podemos filtrar ciertos estilos a otros componentes, ya que estos estilos estarían disponibles a nivel global.

La solución ideal es colocar el `is-command` en el CSS del componente padre, en el archivo `calculator.component.css`

Aún con esta solución, estamos aplicando un estilo a un nivel superior, la solución final ideal debe ser aplicar el estilo directamente en el ámbito del `ComponentButton`, es decir directamente al botón.

Primero regresamos el CSS siguiente al `calculator-button.component.css`

```
.is-command {
  @apply bg-indigo-700 bg-opacity-20 text-opacity-100
}
```

Segundo, eliminamos el `HostBinding` del componente `CalculatorButtonComponent`

```
@HostBinding('class.is-command') get CommandStyle() {
  return this.isCommand();
}
```

Finalmente en el template

```
<button [class.is-command]="isCommand()">
  <ng-content/>
</button>
```

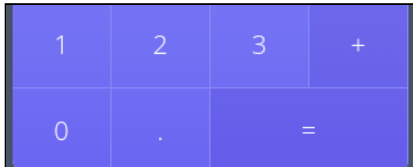
Agregaré la clase `is-command` cuando `isCommand()`, nuestro public input, sea true. Esto genera el HTML siguiente para el botón ÷

```
<calculator-button _ngcontent-ng-c90236445="" iscommand="" class="w-1/4
border-r border-b border-indigo-400" ng-reflect-is-command="">
  <button class="is-command">÷</button>
</calculator-button>
```

De esta forma el CSS queda aplicado a nivel del botón directamente y a la vez, dicha regla se define a nivel del mismo componente, respetando el encapsulamiento y evitando conflictos con otras reglas en nuestra app.

Double Size

El botón "=" a parte de ser un comando tiene un ancho doble



Para aplicar la clase "w-2/4" usaremos el **@HostBinding** y agregaremos un nuevo input: **isDoubleSize**

```
export class CalculatorButtonComponent {

  public isCommand = input(
    false, // default value
    {
      transform: (value: string) => typeof value === 'string' ? value ===
'' : value,
    }
  )

  public isDoubleSize = input(
    false, // default value
    {
      transform: (value: string) => typeof value === 'string' ? value ===
'' : value,
    }
  );

  @HostBinding('class') get DoubleSizeStyle() {
    return this.isDoubleSize() ? 'w-2/4' : 'w-1/4';
  }
}
```

Finalmente en el template usamos el nuevo atributo:

```
<calculator-button isCommand isDoubleSize>=</calculator-button>
```

De esta forma logramos aplicar la clase **w-2/4** al botón.

```
<calculator-button
  _ngcontent-ng-c878039076="" iscommand="" isdoublesize=""
  class="border-r border-b border-indigo-400 w-2/4" ng-reflect-is-
```

```
command="" ng-reflect-is-double-size="">
  <button class="is-command">
    =
  </button>
</calculator-button>
```

Para evitar un conflicto de classes css, se aplicó un cambio, la clase w-* ahora es definida por el @HostBinding en lugar del host: {} directamente en el componente

Nota agregada al curso:

La solución propuesta en el curso genera un "conflicto" de clases css, el elemento host puede llegar a contener ambas clases w-1/2 y w-2/4 siendo lo correcto que solo se incluya una de ellas.

Yo haría esto: primero remover el w-1/2 del Host

```
host: {
  class: 'border-r border-b border-indigo-400',
},
```

y luego en el @HostBinding definir una de ellas

```
@HostBinding('class') get DoubleSizeStyle() {
  return this.isDoubleSize() ? 'w-2/4' : 'w-1/4';
}
```

Con esto evitamos el conflicto entre las clases w-1/2 y w-2/4

Nueva Sección: Señales, comportamiento y lógica

¿Qué veremos en esta sección?

- Host Property - Condicional
- Remove Hostlisteners y HostBindings
- Output Emitter Refs
- Signals ViewChild
- Signal ViewChildren
- Servicios con señales
- Computed Signals
- Realizar cálculos y operaciones
- Validaciones y consideraciones

OutputEmitterRef y Signal ViewChild

Normalmente, para agregar un evento click a un botón realizabamos lo siguientes, en el componente **CalculatorComponent** agregabamos un método:

```
public handleClick (key: string): void {  
    console.log({key});  
}
```

y luego lo podíamos usar de esta forma:

```
<calculator-button (click)="handleClick('c')">C</calculator-button>  
<calculator-button (click)="handleClick('+/-')">+/-</calculator-button>
```

Angular recomienda un Event Emitter, primero, a nivel de ComponentButton agregamos una **output**

```
public onClick = output<string>();
```

Y además agregaremos una función para manejar el click

```
public handleClick() {  
    this.onClick.emit(this.contentValue()?.nativeElement.innerText || '');  
}
```

`this.contentValue()?.nativeElement.innerText` captura el texto del botón.

A continuación actualizamos nuestro botón para tener una referencia `#button` y para llamar la función `handleClick`

```
<button
  #button
  [class.is-command]="isCommand()"
  (click)="handleClick()">
  <ng-content/>
</button>
```

Hasta aca solo hemos definido que el botón emitirá un evento, y el parámetro enviado será el texto asociado al botón.

Seguidamente debemos definir en el elemento padre, el código para escuchar el evento click. Definimos un método:

```
export class CalculatorComponent {
  public handleClick (key: string) {
    console.log({key});
  }
}
```

y en su template

```
<calculator-button (onClick)="handleClick($event)">C</calculator-button>
```

"C" es el contentProjection que le mandamos al Componente Button Button, emite un evento con el valor de "C" "C" es capturado de regreso en el componente padre y su valor es pasado como un argumento Finalmente el argumento padre imprime en consola el texto del botón.