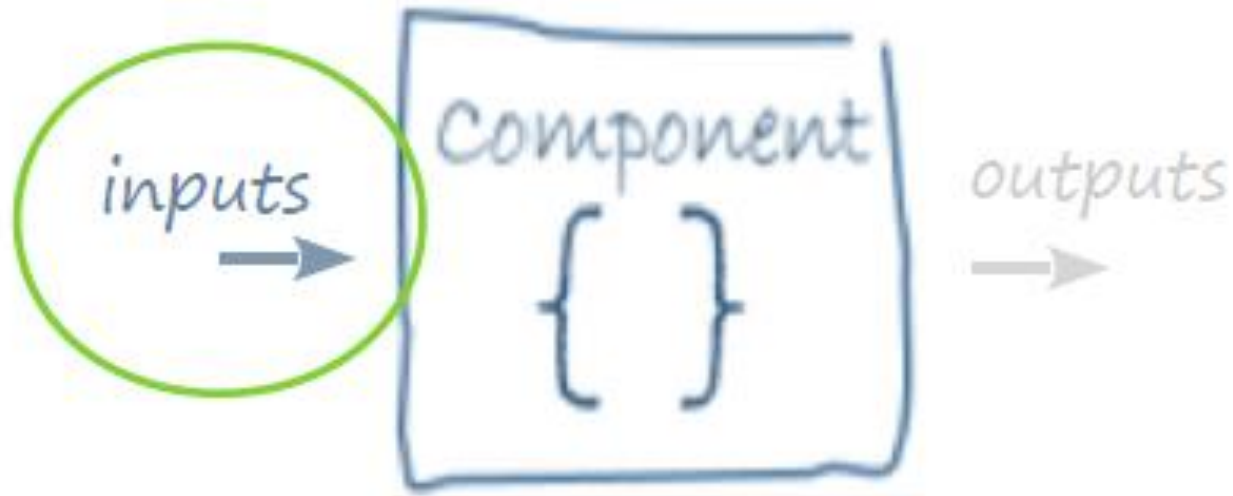


COMPONENTES

» ENTRADAS y SALIDAS

Componentes

Componente > Entradas



Componentes. Entradas padre->hijo

- ▶ Es muy normal usar un componente dentro de otro.
 - El componente en el cual insertamos otro (*se inserta en su **plantilla***), se llama componente **PADRE** (Ej: *ParentComponent*).
 - El componente insertado es el **HIJO** (Ej: *UserComponent*).

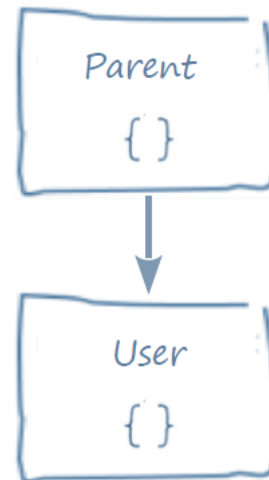
▶ Ej:

parent.component.html

```
<div>
  <h1>User details</h1>
  <user></user>
</div>
```

user.component.ts

```
@Component({
  selector: 'user'
  ...
})
export class UserComponent {
  ...
}
```



En estos casos de uso, es muy normal que **el componente hijo necesite datos de entrada para poder trabajar**.

Por ejemplo, si es un usuario podría necesitar:

- Nombre
- Edad
- Domicilio
- Si es hombre o mujer,...

...para poder mostrarlo en pantalla.

Será el PADRE quién debe darle esa ENTRADA.

Componentes. Pasar datos padre->hijo

¿Cómo pasar los datos del PADRE al HIJO? = PROPERTY BINDING

- ▶ En la plantilla del **padre** pondremos: → *Estamos pasando DATOS*

`<user [name] = "Pepe" [age]="x*2"></user>`

Debe ser una propiedad de `UserComponent` marcada como `@Input`.

Es la **ENTRADA**

Debe ser:

- una propiedad existente en el padre
- una constante (ej: "Pepe")
- Cualquier expresión evaluable.

- ▶ En el controlador del **hijo** indicaremos: → *recogemos los DATOS*

- Qué propiedad es una ENTRADA con el decorador `@Input`

parent.component.html

```
<div>
  <h1>User details</h1>
  <user [name]="Pepe" [age]="18"></user>
</div>
```

user.component.ts

```
import { Input } from '@angular/core';
@Component({
  selector: 'user'
  ...
})
export class UserComponent {
  @Input() age: number;
  @Input() name: string;
  ...
}
```

Componentes. IMPORTANTE

- ▶ Cuando tenemos:

```
<user [name] = "Pepe" [age]="x*2"></user>
```

- **Cualquier cambio en los valores de la parte derecha de la asignación (*en verde*) automáticamente cambian los valores de la parte izquierda y refrescarán la pantalla automáticamente.**
- ▶ Las entradas se recogen en el **controlador** del **hijo** en propiedades marcadas como **@Input**.

- **@Input debe SER IMPORTADO:**

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

- ▶ **Una ENTRADA es “algo” que NOS VENDRÁ DADO DESDE FUERA DEL COMPONENTE.**
 - Si lo recibimos o le damos valor desde el propio componente NO ES ENTRADA, sería UNA PROPIEDAD NORMAL.
 - Normalmente lo proporciona un componente PADRE.

Componentes. **IMPORTANTE**


- ▶ Cuando se instancia el controlador, las entradas son inyectadas automáticamente.
- ▶ Son accesibles después de ngOnInit().
- ▶ En el **constructor NO TENDREMOS ACCESO A LAS ENTRADAS.**

parent.component.html

```
<div>  
  <h1>User details</h1>  
  <user [name]="Pepe" [age]="18"></user>  
</div>
```

user.component.ts

```
@Component({ ... })  
export class UserComponent implements OnInit {  
  @Input() age: number;  
  @Input() name: string;  
  constructor() { console.log('Name: ${this.name}, age: ${this.age}'); }  
  ngOnInit() { console.log('Name: ${this.name}, age: ${this.age}'); }  
}
```

A green curved arrow originates from the `[age]="18"` attribute in the HTML template and points to the `@Input() age: number;` property in the TypeScript class. Another green curved arrow originates from the `[name]="Pepe"` attribute in the HTML template and points to the `@Input() name: string;` property in the TypeScript class.

Componentes. ENTRADAS


- ▶ **Todo** son componentes en Angular:
 - Incluso los elementos del DOM!!!!: `<p>`, ``, `<button>`, `<h1>`,...
 - Así, las **propiedades del DOM** pueden ser **fijadas** o tratadas como **ENTRADAS**.
 - Eso **nos permite modificar DINÁMICAMENTE EL HTML**.
- ▶ Ej de uso de propiedades como ENTRADAS:

example.component.html

```
<img [src]="person.photo" [alt]="person.name" [title]="person.name">  
<button [disabled]="isDisabled" [hidden]="isHidden">Click me!!</button>  
<p [innerHTML]="Hello world!!"></p>
```

example.component.ts

```
@Component({ ... })  
export class ExampleComponent {  
  person: Person;  
  isDisabled: boolean;  
  isHidden: boolean;  
  ...  
}
```



Componentes. ¡CUIDADO!

HTML atributo vs DOM propiedad

- ▶ Los **atributos** son definidos por **HTML**. Las **propiedades** por el **DOM** (Document Object Model).
- ▶ Cuando hacemos:

``

- Estamos accediendo a la propiedad "src" del DOM del elemento "img".
- Algunos atributos del HTML se convierten en una propiedad del mismo nombre. Ej: *id*
Pero **NO TODOS**
- Algunos atributos HTML no tienen propiedades equivalentes en el DOM. Ej: *colspan*
- Algunas propiedades del DOM no tienen atributos equivalentes en HTML. Ej: *textContent*.
- Muchos atributos HTML se corresponden con propiedades del DOM, pero con OTRO SIGNIFICADO:
 - ❑ Los **atributos inicializan las propiedades del DOM**, pero su valores **NO PUEDEN CAMBIAR LUEGO**.
 - ❑ Los **valores de las propiedades del DOM SÍ PUEDEN CAMBIAR**.

Componentes. ¡CUIDADO!

HTML atributo vs DOM propiedad

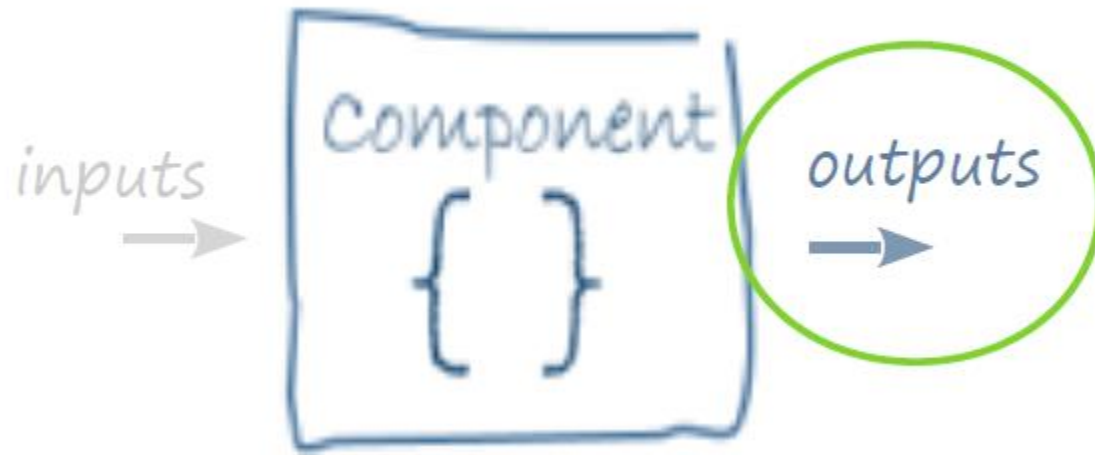
- ▶ Ejemplo: cuando el navegador renderiza:

`<input type="text" value="María">`

- crea el correspondiente *nodo DOM con una propiedad "value" inicializada a "María".*
- Si el **usuario** luego introduce "Juan" en el "input box", *la propiedad "value" del DOM cambia a "Juan".*
 - Pero el **atributo value del HTML** sigue valiendo "María". NO HA CAMBIADO.
 - Se puede mostrar el valor del atributo HTML con: `input.getAttribute('value')`
 - El atributo **value** del HTML indica el VALOR INICIAL; la propiedad **value** del DOM indica el valor actual.
- ▶ **Toda propiedad del DOM va a estar disponible como entrada en ANGULAR.**

Componentes

Componente > Salidas




Componentes. Salidas: hijo -> padre

- ▶ Un componente **puede notificar “algo” a su padre**; puede generar datos de salida.
- ▶ Las **salidas en Angular** se modelan como **EVENTOS**:
 - Si un componente debe **notificar** algo, lo va a hacer con un **evento** de forma asíncrona.

Componentes. Salidas: hijo -> padre

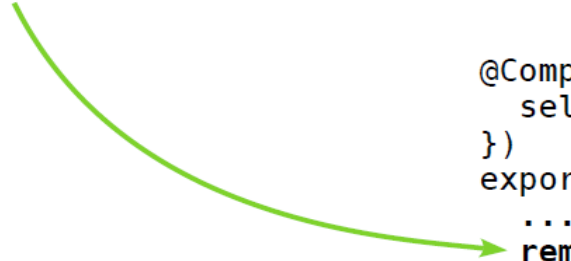
- ▶ **Escuchamos** las salidas en la **plantilla padre** con la sintaxis: **(evento)**

```
<user (removed)="removeUser()"></user>                                @Component({
                                                                           selector: 'parent'
                                                                           })
                                                                           export class ParentComponent {
                                                                           ...
                                                                           removeUser() { ... }
                                                                           }
```



- ▶ Las **salidas también podrían emitir datos**. Ejemplo:

```
<users (removed)="removeUser($event)"></users>                                @Component({
                                                                           selector: 'parent'
                                                                           })
                                                                           export class ParentComponent {
                                                                           ...
                                                                           removeUser(name: string){ ... }
                                                                           }
```



donde “**\$event**” representa el evento ocurrido, con toda su información:

- *Si es un evento del DOM:* click, mouseover,..., **\$event** será MouseEvent, KeyboardEvent,...
- *Si es EventEmitter*, el valor emitido estará en **\$event**.
- Todos los objetos del DOM tienen la **propiedad “target”** que representa el **elemento que lanzó el evento**.

Componentes. Salidas: hijo -> padre

- ▶ Cuando hacemos:

`<componente (evento) = "método()" ></componente>`

estamos registrando un listener para ese evento.

Por tanto:

`(evento)` → es el **nombre de la salida** que genera el componente

`"método()"` → debe ser una **SENTENCIA**; no una **EXPRESIÓN**.

Esa sentencia, **se evalúa contra el contexto actual** que, como siempre es el **controlador del componente actual**.

Por tanto, **ese método debe existir en ese controlador**.

Componentes. Salidas: hijo -> padre

► Definimos salidas en el controlador generando eventos con: @Output y EventEmitter.

- @Output

- permite *marcar como salida una variable de instancia.*
- Está definido en '@angular/core'

- EventEmitter

- es una **clase** definida en '@angular/core' también.
- Permite **emitir eventos**.
- Permite **crear objetos que pueden emitir eventos** mediante el método: "*emit(...)*".

PASOS A SEGUIR:

1. Se deben importar @Output y EventEmitter con la sentencia:

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

2. Se declara la variable de instancia con @Output y se crea el objeto EventEmitter:

```
@Output() removed = new EventEmitter();
```

3. Cuando queramos generar el evento, lo emitimos a nuestro padre pasándole los datos que queramos (*él lo tratará luego*):

```
remove() { this.removed.emit(this.name);}
```

Componentes. Salidas: hijo -> padre

▶ Ejemplo:

user.component.ts

```
import { Component, Input, Output, EventEmitter } from '@angular/core';
@Component({
  selector: 'user'
})
export class UserComponent {
  @Input() name: string; @Input() age: string;
  @Output() removed = new EventEmitter();
  remove() { this.removed.emit(this.name); }
}
```

parent.component.html

```
<div>
  <h1>User details</h1>
  <user [name]="Pepe" [age]="18" (removed)="removeUser($event)"></user>
</div>
```



Componentes. ENTRADAS

- ▶ **RECORDAD: Todo** son componentes en Angular!!!!:
- Incluso los elementos del DOM!!!!: `<p>`, ``, `<button>`, `<h1>`,...
- Así, los **eventos del DOM** pueden ser **fijados** o tratados como **SALIDAS**.
- ▶ Ej de uso de eventos DOM como SALIDAS:

```
<button (click)="clicked()">
  Click me!
</button>
<div
  (click)="onEvent($event)"
  (mouseenter)="onEvent($event)"
  (mouseleave)="onEvent($event)"
  (mousemove)="onEvent($event)"
</div>
```

```
@Component({ ... })
export class ExampleComponent {
  ...
  clicked() { ... }
  onEvent(event: MouseEvent) { ... }
}
```

