

## Laboratorio - Pipes en Angular

### ¿Qué son los *pipes*?

Los *pipes* son una herramienta de Angular que nos permite **transformar** visualmente la información, por ejemplo, cambiar un texto a mayúsculas o minúsculas, o darle formato de fecha y hora.

Angular trae una serie de *pipes* **por defecto** pero también nos permite construir nuestros propios *pipes*.

Estos son los *pipes* de Angular que vamos a ver:

- **uppercase y lowercase**

Ejemplo:

```
let name = 'sheldon';
{{ name | uppercase }}
Result --> SHELDON
```

- Slice
- Decimal
- Percent
- Currency
- Json
- Async
- Date

Ejemplo:

```
let birthday = new Date(1980, 6, 31);
{{ birthday | date:'dd/mm/yy' }}
Result --> 31/07/1980
```

Vamos a ver cada *pipe* en profundidad construyendo una pequeña **demo-app** que cogerá una variable y la transformará en algo distinto dependiendo del *pipe* que se le aplique. Será algo así:

VARIABLE	PIPE	RESULT
Variable value	type of pipe	Variable after applying the pipe

### Creando la aplicación

1. Vamos a **crear** una sencilla app de Angular con [Angular CLI](#), usando el comando:  
ng new <nombre-de-tu-app>  
en la carpeta donde quieras que se encuentre tu aplicación.

2. Incluimos **bootstrap** para poder diseñar más rápido nuestra app y centrarnos en el tema que nos ocupa: aprender sobre los *pipes*. Lo incluimos dentro del archivo *index.html*, aunque esa no es la manera ideal de incluirlo, sí es la más **rápida**, y es suficiente para este caso.

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Uso de Pipes en Angular 12</title>
  <base href="/">
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/boo
tstrap.min.css"
  integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQU0hcWr7x9JvoRxT2MZw1T
" crossorigin="anonymous">
  <meta name="viewport" content="width=device-width, initial-
scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>

<body>
  <app-root></app-root>

  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
  integrity="sha384-
q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo
" crossorigin="anonymous">
  </script>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd
/popper.min.js"
  integrity="sha384-
U02eT0CpHqdSJQ6hJty5KVphtPhzWj9W01clHTMGa3JDZwrnQq4sF86dIHNDz0W1
" crossorigin="anonymous">
  </script>
```

```

    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootst
trap.min.js"
    integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy60rQ6VrjIEaFf/nJGzIxFDs4x0xIM+B07jRM
" crossorigin="anonymous">
    </script>
</body>

</html>

```

Con estos cambios, bootstrap ya se debería estar aplicando a nuestra app.

3. Vamos a utilizar una de las **tablas** de bootstrap para construir esta app. En este caso, se copiara uno de los ejemplos de bootstrap que más se ajuste a lo que se necesite, y posteriormente adaptarlo.

Se trabajara sobre el archivo *app.component.html*, así que **borramos** el código que trae Angular por defecto y añadimos la tabla dentro de un *bootstrap container*.

- Limpiamos la tabla para que quede sólo un **esqueleto** al que luego le iremos añadiendo contenido.

```

<div class="container my-3">
  <h2>Uso de Pipes en Angular</h2>
  <hr>

  <table class="table table-hover">
    <thead class="thead-dark">
      <tr>
        <th scope="col">Variable</th>
        <th scope="col">Pipe code</th>
        <th scope="col">Resultado</th>
      </tr>
    </thead>
    <tbody>

      <tr>
        <td></td>
        <td></td>

```

```

        <td></td>
    </tr>

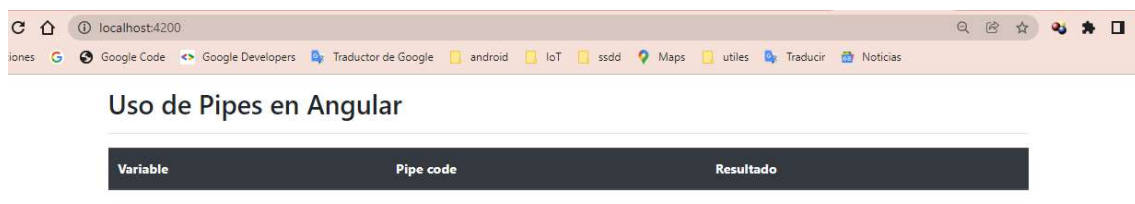
    <tr>
        <td></td>
        <td></td>
        <td></td>
    </tr>

</tbody>
</table>

</div>

```

Hecho esto, tu app debería tener este aspecto:



## Pipes uppercase y lowercase

1. Vamos al *app.component.ts* y añadimos nuestra primera **variable** para poder aplicarle estos dos *pipes*.

```

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Uso de Pipes';

  name = 'Jorge Guerra';
}

```

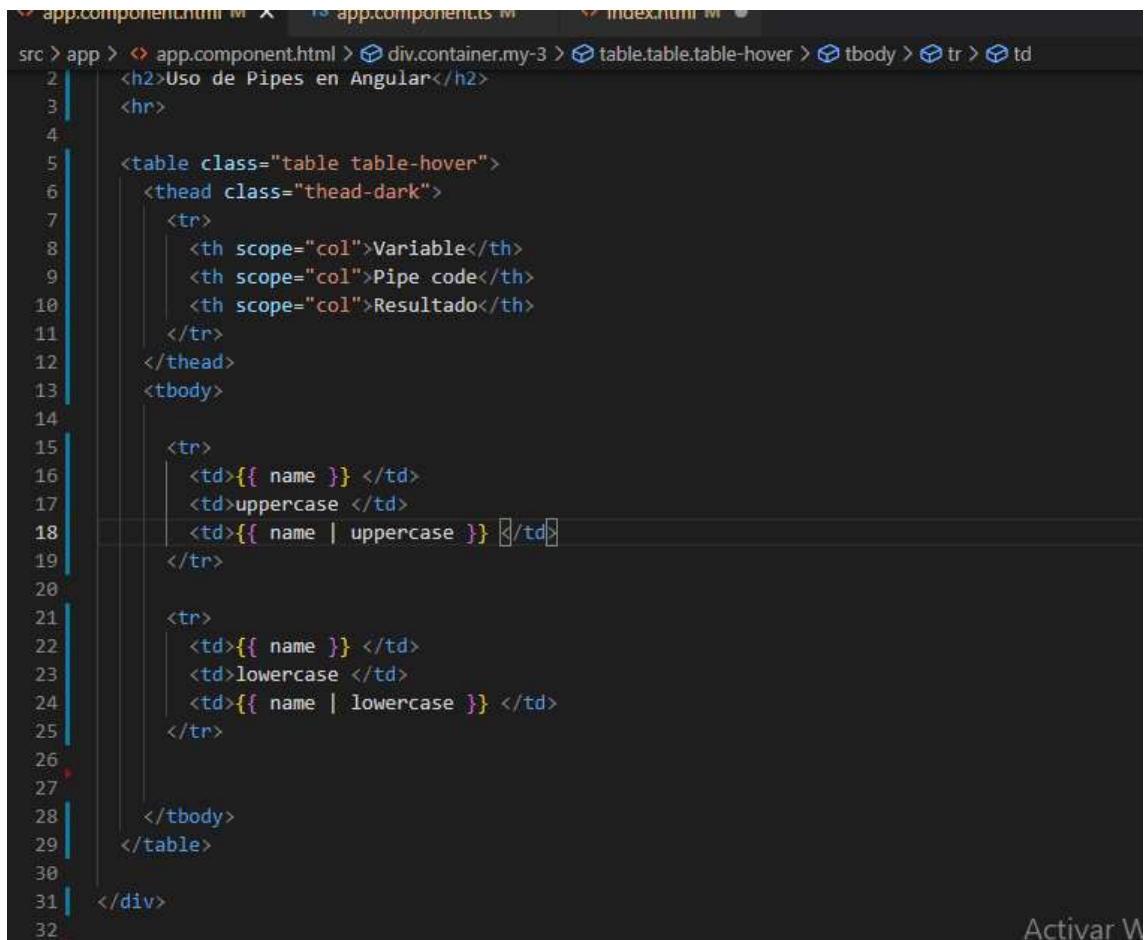
2. Volvemos al `app.component.html` y se vincula la **variable** mediante *string interpolation* (primera columna). En la segunda columna especificamos el nombre y **código** del *pipe* (que en este caso es simplemente *uppercase*) y en la tercera, volvemos a vincular la variable, pero aplicándole el *pipe*:

```
{{ myVariable | thePipe }}
```

3. Replicamos este paso pero con el **pipe lowercase**. Y así nos queda el bloque de código que acabamos de añadir:

```
<tr>
  <td>{{ name }} </td>
  <td>uppercase </td>
  <td>{{ name | uppercase }} </td>
</tr>

<tr>
  <td>{{ name }} </td>
  <td>lowercase </td>
  <td>{{ name | lowercase }} </td>
</tr>
```



The screenshot shows a code editor with the file `app.component.html` open. The breadcrumb navigation at the top indicates the path: `src > app > app.component.html > div.container.my-3 > table.table.table-hover > tbody > tr > td`. The code in the editor is as follows:

```
2 <h2>Uso de Pipes en Angular</h2>
3 <hr>
4
5 <table class="table table-hover">
6   <thead class="thead-dark">
7     <tr>
8       <th scope="col">Variable</th>
9       <th scope="col">Pipe code</th>
10      <th scope="col">Resultado</th>
11    </tr>
12  </thead>
13  <tbody>
14
15    <tr>
16      <td>{{ name }} </td>
17      <td>uppercase </td>
18      <td>{{ name | uppercase }} </td>
19    </tr>
20
21    <tr>
22      <td>{{ name }} </td>
23      <td>lowercase </td>
24      <td>{{ name | lowercase }} </td>
25    </tr>
26
27  </tbody>
28 </table>
29
30
31 </div>
32
```

The breadcrumb navigation at the bottom right of the editor shows the path: `Activar W`.

A continuación, se muestra el resultado del código. A partir de aquí, lo que haremos será ir añadiendo **filas**, una por cada *pipe* o cosa nueva que aprendamos.

### Uso de Pipes en Angular

Variable	Pipe code	Resultado
Jorge Guerra	uppercase	JORGE GUERRA
Jorge Guerra	lowercase	jorge guerra

### Pipe: slice

El *slice* es un *pipe* que requiere mínimo un parámetro.

#### Con un parámetro

Su sintaxis sería:

```
{{ myVariable | slice:param }}
```

El parámetro que pongamos después de los dos puntos (un número) será la cantidad de elementos que **eliminará** de nuestra variable, empezando por el principio. Por ejemplo:

```
{{ myVariable | slice:3 }}
```

quitaría las letras "myV", dando un resultado de "**ariable**".

#### Con dos parámetros

Este *pipe* también admite dos parámetros, y su sintaxis sería:

```
{{ myVariable | slice:paramOne:paramTwo }}
```

El primer parámetro (un número) recoge la posición desde donde queremos empezar a cortar y el segundo, hasta dónde queremos **cortar** nuestra variable. El resultado serán los elementos que hayan quedado entre el primer y el segundo parámetro, por ejemplo:

```
{{ myVariable | slice:0:4 }}
```

Resultaría en "**myVa**".

1. Vamos a aplicar estos cambios a nuestro código, en el archivo *app.component.html*.

```
<tr>
  <td>{{ name }} </td>
  <td>slice:0:4 </td>
  <td>{{ name | slice:0:4 }} </td>
</tr>
```

## Uso de Pipes en Angular

Variable	Pipe code	Resultado
Jorge Guerra	uppercase	JORGE GUERRA
Jorge Guerra	lowercase	jorge guerra
Jorge Guerra	slice:0:4	Jorg

2. Este *pipe* funciona igualmente con *arrays*. Vamos a declarar un *array* de ejemplo en el *app.component.ts*, asignándole números.

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Uso de Pipes';
  name = 'Jorge Guerra';

  myArray = [1, 2, 3, 4, 5, 6, 7, 8];
}
```

3. Y ahora, en el *app.component.html*, podemos utilizar el *slice* de la misma manera que con nuestra variable *name*. Aquí el bloque de código que hemos añadido:

```
<tr>
  <td>{{ myArray }} </td>
  <td>slice:0:4 </td>
  <td>{{ myArray | slice:0:4 }} </td>
</tr>
```

Observese que el *slice* empieza a contar desde el primer parámetro e incluye todos los elementos entre el primer parámetro y el **último elemento** antes del segundo parámetro.

## Uso de Pipes en Angular

Variable	Pipe code	Resultado
Jorge Guerra	uppercase	JORGE GUERRA
Jorge Guerra	lowercase	jorge guerra
Jorge Guerra	slice:0:4	Jorg
1,2,3,4,5,6,7,8	slice:0:4	1,2,3,4

### Aplicando el *pipe* a la directiva *ngFor*

Los *pipes* también se pueden aplicar a elementos de lógica de programación, como la [directiva \*ngFor\*](#).

1. Vamos a crearnos una `<ul>` para que nos imprima cada elemento de nuestro *array* en un `<li>`. Esto se puede hacer fácilmente usando el *ngFor*.
2. Le añadimos el *pipe slice* al *ngFor* y le pasamos **dos** parámetros, ya sabes, el primero indica desde dónde empezaremos a cortar y el segundo, hasta dónde. La sintaxis sería así:

```
<li *ngFor="let number of myArray | slice:paramOne:paramTwo">{{
number }}</li>
```

En este caso, si atribuimos al segundo parámetro un número **mayor** que la cantidad de elementos que hay en nuestro array, Angular no nos dará ningún error. Por ejemplo, si le pedimos que imprima los elementos de primero al quinto pero solo hay cuatro, imprimirá del primero al cuarto.

Aquí el bloque de código que hemos añadido:

```
<tr>
  <td>{{ myArray }} </td>
  <td>*ngFor="let number of myArray | slice:0:3" </td>
  <td>
    <ul>
      <li *ngFor="let number of myArray | slice:0:3">{{
number }} </li>
    </ul>
  </td>
</tr>
```





## Uso de Pipes en Angular

Variable	Pipe code	Resultado
Jorge Guerra	uppercase	JORGE GUERRA
Jorge Guerra	lowercase	jorge guerra
Jorge Guerra	slice:0:4	Jorg
1,2,3,4,5,6,7,8	slice:0:4	1,2,3,4
1,2,3,4,5,6,7,8	*ngFor="let number of myArray   slice:0:3"	<ul style="list-style-type: none"><li>• 1</li><li>• 2</li><li>• 3</li></ul>

### Pipe: decimal

Podemos utilizar este *pipe* para trabajar con números y decimales. Su sintaxis sería:

```
{{ myVariable | number: '(string con 3 valores numéricos) N1.N2-N3' }}
```

O, sin **parámetros**:

```
{{ myVariable | number }}
```

Sí, el *pipe* se llama "decimal" pero al usarlo en el código se usa la palabra clave

"number".

Si no especificamos parámetros, Angular los **creará** igualmente *behind the scenes*, adquiriendo los siguientes valores por defecto:

- N1 = 1      cantidad de números enteros
- N2 = 0      cantidad mínima de números decimales
- N3 = 3      cantidad máxima de números decimales

1. En el archivo *app.component.ts* declaramos una variable equivalente al número PI, mediante el uso del *Math object*. Aquí la línea de código que hemos añadido:

```
PI = Math.PI;
```

2. Si ahora le aplicamos este *pipe* a la variable *PI* sin ningún añadido extra, verás que en tu navegador se muestra un número integral (3) y tres decimales (142). Esto ocurre

porque si no le especificamos nada, esos serán los valores que tome el *pipe* por **defecto**: un número integral y tres decimales.

Realizamos estos cambios en el archivo *app.component.html*. Aquí el bloque de código que hemos añadido:

```
<tr>
  <td>{{ PI }} </td>
  <td>number</td>
  <td>{{ PI | number }} </td>
</tr>
```

### Uso de Pipes en Angular

Variable	Pipe code	Resultado
Jorge Guerra	uppercase	JORGE GUERRA
Jorge Guerra	lowercase	jorge guerra
Jorge Guerra	slice:0:4	Jorg
1,2,3,4,5,6,7,8	slice:0:4	1,2,3,4
1,2,3,4,5,6,7,8	*ngFor="let number of myArray   slice:0:3"	<ul style="list-style-type: none"><li>• 1</li><li>• 2</li><li>• 3</li></ul>
3.141592653589793	number	3.142

3. Vamos a **personalizar** nuestro *pipe* especificándole que queremos sólo un número entero y dos decimales. Aquí el bloque de código que hemos añadido:

```
<tr>
  <td>{{ PI }} </td>
  <td>number:'1.0-2'</td>
  <td>{{ PI | number:'1.0-2' }} </td>
</tr>
```

Al ejecutarse se verá el resultado formateado

## Uso de Pipes en Angular

Variable	Pipe code	Resultado
Jorge Guerra	uppercase	JORGE GUERRA
Jorge Guerra	lowercase	jorge guerra
Jorge Guerra	slice:0:4	Jorg
1,2,3,4,5,6,7,8	slice:0:4	1,2,3,4
1,2,3,4,5,6,7,8	*ngFor="let number of myArray   slice:0:3"	<ul style="list-style-type: none"> <li>1</li> <li>2</li> <li>3</li> </ul>
3.141592653589793	number	3.142
3.141592653589793	number:'1.0-2'	3.14

También es posible **formatear** un número de manera que muestre una coma cada 3 cifras, sin necesidad de especificar una cantidad de números integrales. La sintaxis sería:

```
{{ myVariable | number: '.N1-N2' }}
```

Es decir, si cogemos este número 123456 y le aplicamos un *pipe number* como este:

```

13 |   PI = Math.PI;
14 |
15 |   myVariable = 123456789
16 | }
17 |

```

```
{{ myVariable | number: '.0-0' }}
```

```

<tr>
  <td>{{ myVariable }} </td>
  <td>number:'1.0-2'</td>
  <td>{{ myVariable | number:'.0-0' }} </td>
</tr>
</tbody>
</table>

```

El resultado sería 123,456.

## Uso de Pipes en Angular

Variable	Pipe code	Resultado
Jorge Guerra	uppercase	JORGE GUERRA
Jorge Guerra	lowercase	jorge guerra
Jorge Guerra	slice:0:4	Jorg
1,2,3,4,5,6,7,8	slice:0:4	1,2,3,4
1,2,3,4,5,6,7,8	*ngFor="let number of myArray   slice:0:3"	<ul style="list-style-type: none"> <li>1</li> <li>2</li> <li>3</li> </ul>
3.141592653589793	number	3.142
3.141592653589793	number:'1.0-2'	3.14
123456789	number:'1.0-2'	123,456,789

Activar Windows

## Pipe: percent

Utilizamos este *pipe* para mostrar números en forma de porcentaje, y su sintaxis es muy parecida al *pipe decimal*:

```
{{ myVariable | percent: '(string con 3 valores numéricos) N1.N2-N3' }}
```

- N1 = 1 --> cantidad de números enteros (valor por defecto)
- N2 = 0 --> cantidad mínima de números decimales (valor por defecto)
- N3 = 0 --> cantidad máxima de números decimales (valor por defecto)

1. Se creará una variable (llamada *myNum*, por ejemplo) en el archivo *app.component.ts* con la que trabajar. Aquí la línea de código que hemos añadido:

```
myNum = 0.589;
```

2. En el archivo *app.component.html*, le aplicamos el *pipe* a nuestra variable *myNum*. Se observará que si no le añadimos ningún parámetro al *pipe*, el resultado final es 59%. Esto es así porque:

- el *pipe* **multiplica** nuestra variable por 100.
- por defecto el *pipe* no aplica ningún decimal, y **redondea** hacia el integral más próximo, que en este caso es 59.

Pero si le añadimos algún **parámetro**, por ejemplo, si queremos ver dos enteros y dos decimales, el resultado sería 58.90%. Aquí el bloque de código que hemos añadido:

```
<tr>
  <td>{{ myNum }} </td>
  <td>percent</td>
  <td>{{ myNum | percent }} </td>
</tr>

<tr>
```

```

    <td>{{ myNum }} </td>
    <td>percent:'2.2-2'</td>
    <td>{{ myNum | percent:'2.2-2' }} </td>
</tr>

```

1,2,3,4,5,6,7,8

\*ngFor="let number of myArray | slice:0:3"

- 1
- 2
- 3

3.141592653589793	number	3.142
3.141592653589793	number:'1.0-2'	3.14
123456789	number:'1.0-2'	123,456,789
0.589	percent	59%
0.589	percent:'2.2-2'	58.90%

## Pipe: currency

Este *pipe* se utiliza cuando queremos mostrar números acompañados de una **divisa** (euros, dólares, yenes, etc). Tiene una sintaxis parecida a los *pipes decimal* y *percent* (en la parte final).

Utiliza de referencia el documento [ISO 4217](#), una lista con el **código** asociado a cada divisa. Su sintaxis es la siguiente, sin parámetros:

```
{{ myVariable | currency }}
```

Lo que resultará en un formato en dólares (\$) por defecto. Pero también admite **parámetros**:

```
{{ myVariable | currency:'currencyCharacter' }}
```

- currencyCharacter --> lo podemos encontrar en el ISO 4217. Son los EUR, USD, etc.
- Por defecto mostrará el *symbol* de la divisa, es decir, \$, €, etc.

```
{{ myVariable | currency:'currencyCharacter':'symbol/code' }}
```

- symbol/code --> o uno u otro.

Podemos usar una de esas dos sintaxis y **combinarla** con una parte final para dar formato a los números. Esa parte final tiene el mismo estilo que para los *pipes decimal* y *percent*.

```
{{ myVariable | currency:'currencyCharacter':'symbol/code':N1.N2-N3' }}
```

- N1 = 1 (valor por defecto) --> cantidad de números enteros
- N2 = 2 (valor por defecto) --> cantidad mínima de números decimales
- N3 = 2 (valor por defecto) --> cantidad máxima de números decimales

1. Se crea una variable (llamada *salary*, por ejemplo) en el archivo *app.component.ts*. Aquí la línea de código que hemos añadido:

```
salario = 3500.5;
```

2. Y ahora en el *app.component.html*, aplicamos el *pipe* a la variable. Si únicamente le aplicamos el *pipe* sin ningún añadido, verás que el *pipe* transforma la variable en una cantidad en dólares americanos.

\$3,500.50

Aquí el bloque de código que hemos añadido:

```
<tr>
  <td>{{ salario }} </td>
  <td>currency</td>
  <td>{{ salario | currency }} </td>
</tr>
```

3.141592653589793	number	3.142
3.141592653589793	number:'1.0-2'	3.14
123456789	number:'1.0-2'	123,456,789
0.589	percent	59%
0.589	percent:'2.2-2'	58.90%
3500.5	currency	\$3,500.50

*Angular Value*

3. Si queremos mostrar nuestra variable en formato de Euros, por ejemplo, vamos al documento ISO 4217 y buscamos el **código** para el Sol Peruano (PEN). Se lo aplicamos al *pipe* (tenemos la sintaxis arriba) y eso nos da un resultado de 3,500.50.

Esto puede resultar algo confuso, porque:

- código = code = EUR, USD, GBP, etc...
- símbolo = symbol = €, \$, etc...

Y, si al *pipe* le pasamos por parámetro el código del Euro (EUR), Angular nos devolverá un resultado con el **símbolo del euro (€)**. Porque, como hemos mencionado arriba, Angular devuelve por defecto el **símbolo** de la divisa.

Aquí el bloque de código que hemos añadido al *app.component.html*:

```
<tr>
  <td>{{ salario }} </td>
  <td>currency:'PEN'</td>
  <td>{{ salario | currency:'PEN' }} </td>
</tr>
```

3.141592653589793	number:'1.0-2'	3.14
123456789	number:'1.0-2'	123,456,789
0.589	percent	59%
0.589	percent:'2.2-2'	58.90%
3500.5	currency	\$3,500.50
3500.5	currency:'PEN'	PEN3,500.50

Activar Windo

4. También podemos intentar que muestre el **código** de la divisa (EUR) en lugar del símbolo (€), y pasarle un parámetro al *pipe* para que no muestre ningún decimal. Aquí el bloque de código que hemos añadido:

```
<tr>
  <td>{{ salario }} </td>
  <td>currency:'EUR': 'code'</td>
  <td>{{ salario | currency:'EUR': 'code' }} </td>
</tr>

<tr>
  <td>{{ salario }} </td>
  <td>currency:'EUR': 'symbol': '4.0-0'</td>
  <td>{{ salario | currency:'EUR': 'symbol': '4.0-0' }}
</td>
</tr>
```

3500.5	currency:'PEN'	PEN3,500.50
3500.5	currency:'EUR': 'code'	EUR3,500.50
3500.5	currency:'EUR': 'symbol': '4.0-0'	€3,501

Activar Windo

## Pipe: JSON

El *pipe JSON* es muy útil combinado con la etiqueta **<pre>**, cuando queremos mostrar código en formato JSON. Es especialmente útil cuando queremos mostrar [objetos](#) en el navegador, porque si intentamos mostrarlos sin este *pipe*, lo único que veremos será algo así:

```
[object Object]
```

Esto pasa porque un objeto es un *reference type*.

1. Vamos a crear un objeto (llamado *nerd*, por ejemplo) en el *app.component.ts* sobre el que trabajar. Aquí el bloque de código que hemos añadido:

```
novato = {
  name: 'Elena Soto',
  alias: 'prima',
```

```

    song: 'Hurt So Good',
    skills: ['divertida y graciosa', 'Come mucha pizza'],
    youtubeChannel: 'Riete con Elena',
    address: {
      street: 'Av Universitaria',
      number: 3567,
      city: 'Lima'
    }
  };

```

2. Y ahora vamos a aplicarle el *pipe JSON combinado* con la etiqueta `<pre>`. Aquí el código que hemos añadido en el `app.component.html`.

```

<tr>
  <td> {{ novato }} </td>
  <td>json</td>
  <td><pre>{{ novato | json }} </pre> </td>
</tr>

```

Mostrando la respuesta a continuación:

3500.5	currency:'PEN'	PEN3,500.50
3500.5	currency:'EUR':'code'	EUR3,500.50
3500.5	currency:'EUR':'symbol':'4.0-0'	€3.501
[object Object]	json	<pre> {   "name": "Elena Soto",   "alias": "prima",   "song": "Hurt So Good",   "skills": [     "divertida y graciosa",     "Come mucha pizza"   ],   "youtubeChannel": "Riete con Elena",   "address": {     "street": "Av Universitaria",     "number": 3567,     "city": "Lima"   } } </pre>

## Pipe: async

Este *pipe* nos permite mostrar información proveniente de código basado en ***promises***, entre otras cosas. Su sintaxis es muy sencilla:

```

{{ myVar | async }}

```

`myVar` = siempre será o una *promise* o un *observable*.

1. Vamos a crearnos una *promise* en el archivo `app.component.ts`. **Simularemos**, mediante un `setTimeout`, que estamos realizando una *http request* para que nos devuelva una información X. Aquí el bloque de código que hemos añadido:

```

promiseValue = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve(dato X se muestra!);
  }, 3000);
});

```



```
});  
  
}
```

2. Volvemos al **app.component.html** y le aplicamos el *pipe* a nuestra variable *promiseValue*. Este *pipe* no requiere ningún parámetro, por lo que al aplicárselo tal cual a nuestra variable, verás en tu navegador cómo aparece el **valor** de la promesa al cabo de 3 segundos.

Puedes probar también a pasarle el *pipe json* a la variable, para ver qué te devuelve. Aquí el bloque de código que hemos añadido:

```
<tr>  
  <td> {{ promiseValue }} </td>  
  <td>json</td>  
  <td><pre>{{ promiseValue | json }} </pre> </td>  
</tr>  
  
<tr>  
  <td> {{ promiseValue }} </td>  
  <td>async</td>  
  <td><pre>{{ promiseValue | async }} </pre> </td>  
</tr>
```

Variable	Pipe code	Resultado
[object Promise]	json	{ "__zone_symbol__state": null, "__zone_symbol__value": [] }
[object Promise]	async	

Después de 3 segundos:

Variable	Pipe code	Resultado
[object Promise]	json	{ "__zone_symbol__state": true, "__zone_symbol__value": "dato X se muestra!" }
[object Promise]	async	dato X se muestra!

## Pipe: date

Este *pipe* nos permite mostrar fechas de manera más legible, admitiendo múltiples formatos. Su sintaxis sería:

```
{{ myVar | date:'parameter' }}
```

El *parameter* es opcional.

1. Vamos a crear una variable en el archivo *app.component.ts* que será una **instance** del *Date object*. Aquí el bloque de código que hemos creado:

```
myDate = new Date();
```

2. Le aplicamos el *pipe date* a nuestra variable en el archivo *app.component.html*. Sin parámetros, para observar su comportamiento por defecto. Aquí el código que hemos añadido:

```
<tr>
  <td> {{ myDate }} </td>
  <td>date</td>
  <td>
    <pre>{{ myDate | date }} </pre>
  </td>
</tr>
```

En tu navegador verás cómo una fecha tan larga como esta:

```

    "youtubeChannel": "Riete con Elena",
    "address": {
      "street": "Av Universitaria",
      "number": 3567,
      "city": "Lima"
    }
  }
}

Mon Jan 17 2022 17:09:17 GMT-0500 (hora estándar de Perú)    date    Jan 17, 2022
Activar Window
```

3. Vamos a probar con algún parámetro, por ejemplo, *medium*. En el archivo *app.component.html*, añadimos:

```
<tr>
  <td> {{ myDate }} </td>
  <td>date:'medium'</td>
  <td>
    <pre>{{ myDate | date:'medium' }} </pre>
  </td>
</tr>
```

Y ahora debería salirte el mismo formato + la hora local:

```

    }
  }
}

Mon Jan 17 2022 17:10:49 GMT-0500 (hora estándar de Perú)    date    Jan 17, 2022
Mon Jan 17 2022 17:10:49 GMT-0500 (hora estándar de Perú)    date:'medium'    Jan 17, 2022, 5:10:49 PM
Activar Window
```

## Formatos de fecha personalizados

También podemos generar formatos personalizados de fechas en base a la documentación oficial. Podemos **combinarlos** entre sí. Por ejemplo, si queremos que muestre el día de la semana, el mes y el año, escribiríamos:

```
date: 'EEE, d of LLL of yyyy'
```

Luego, podemos añadirle cualquier palabra o símbolo entre keywords, como hemos hecho con la coma y el of.

1. Vamos a añadir este formato personalizado a nuestro código del archivo `app.component.html`:

```
<tr>
  <td> {{ myDate }} </td>
  <td>date: 'EEE, d of LLL of yyyy'</td>
  <td>
    <pre>{{ myDate | date: 'EEE, d of LLL of yyyy' }}
</pre>
  </td>
</tr>
```

Lo cual nos dará un resultado tipo:

Perú)		
Mon Jan 17 2022 17:13:20 GMT-0500 (hora estándar de Perú)	date:'medium'	Jan 17, 2022, 5:13:20 PM
Mon Jan 17 2022 17:13:20 GMT-0500 (hora estándar de Perú)	date:'EEE, d of LLL of yyyy'	Mon, 17 of Jan of 2022

Activar Wi

## Fechas en otros idiomas

Todo esto está muy bien, pero habrás notado que, por defecto, el resultado siempre se muestra en inglés. Para mostrarlo en cualquier otro **idioma**, como el español, debemos hacer una pequeña configuración en el módulo `app.module.ts`.

1. Importamos una constante llamada `LOCALE_ID` desde el paquete `@angular/core`.
2. En el array `providers`, añadimos un **objeto** donde especificaremos el idioma. Debemos también importar el idioma y el módulo para registrar dicho idioma. Así nos queda el archivo `app.module.ts`:

```
import { BrowserModule } from '@angular/platform-browser';
import { LOCALE_ID, NgModule } from '@angular/core';

import { AppComponent } from './app.component';
```

```

import localeEs from '@angular/common/locales/es';
import { registerLocaleData } from '@angular/common';
registerLocaleData(localeEs);

@NgModule({
  imports: [BrowserModule],
  declarations: [AppComponent],
  providers: [{ provide: LOCALE_ID, useValue: 'es' }],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

De esa manera, **app.module.ts** quedara de la siguiente manera:

```

src > app > TS app.module.ts > AppModule
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6
7  import { LOCALE_ID } from '@angular/core';
8  import localeEs from '@angular/common/locales/es';
9  import { registerLocaleData } from '@angular/common';
10
11 registerLocaleData(localeEs);
12
13 @NgModule({
14   declarations: [
15     AppComponent
16   ],
17   imports: [
18     BrowserModule,
19     AppRoutingModule
20   ],
21   providers: [{ provide: LOCALE_ID, useValue: 'es' }],
22   bootstrap: [AppComponent]
23 })
24 export class AppModule { }
25

```

Es posible que la consola del navegador te lance algún error, pero fíjate que ahora las fechas están en **español**. La última que habíamos incluido te saldrá con el "of" incluido, lógicamente, ya que está *hardcoded*:

		}
Mon Jan 17 2022 17:20:45 GMT-0500 (hora estándar de Perú)	date	17 ene 2022
Mon Jan 17 2022 17:20:45 GMT-0500 (hora estándar de Perú)	date:'medium'	17 ene 2022 17:20:45
Mon Jan 17 2022 17:20:45 GMT-0500 (hora estándar de Perú)	date:'EEE, d of LLL of yyyy'	1un, 17 of ene of 2022

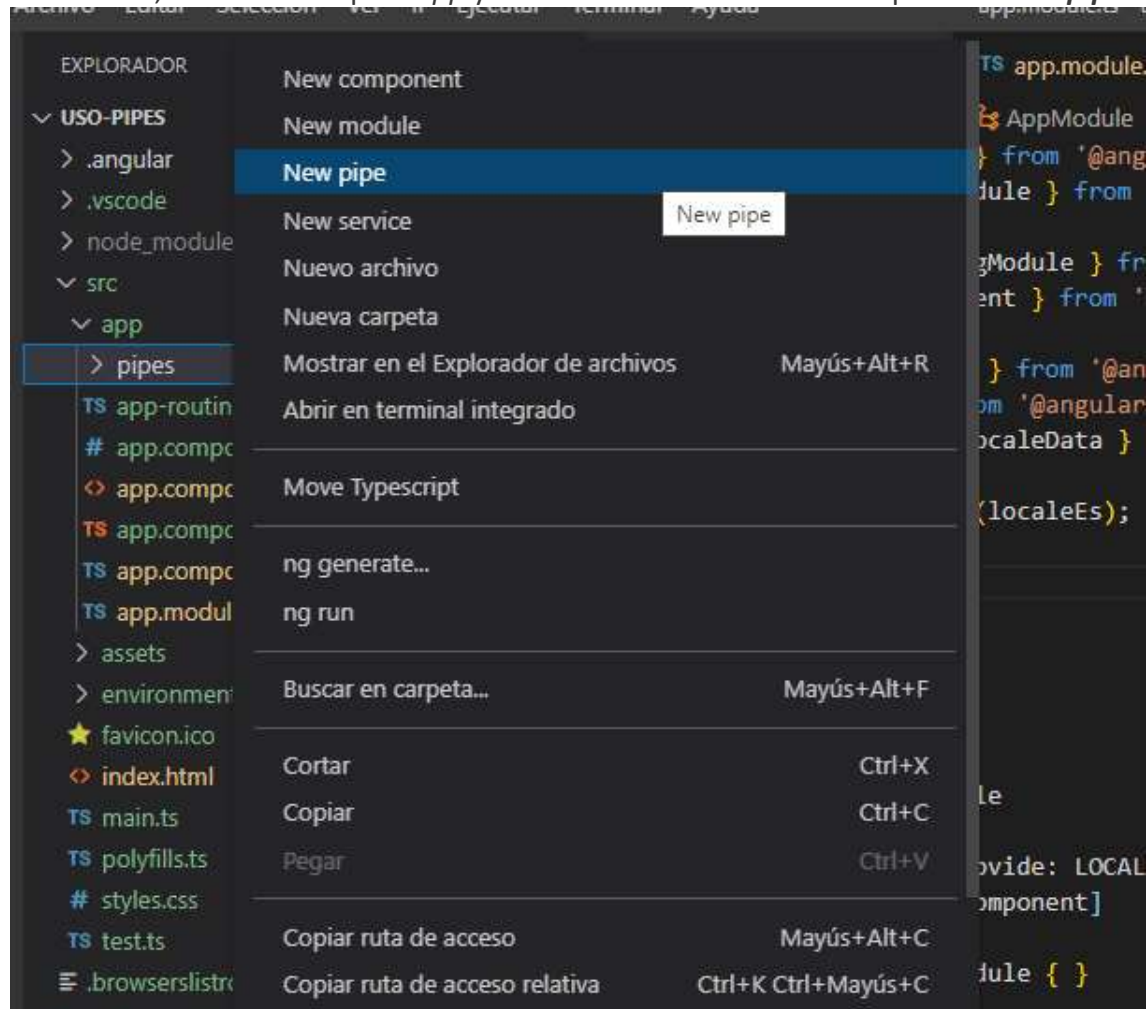
Activar Window

## Pipes personalizados

### Creación manual de un *pipe*

Hasta ahora hemos dado un repaso completo a los *pipes* más utilizados en Angular. Pero también existe la posibilidad de crear nuestros **propios pipes**. [Angular CLI](#) nos permite generar la estructura de un *pipe*, pero ahora se va a realizar manualmente para entender lo que pasa.

1. Para ello, vamos a la carpeta *app* y dentro de ella creamos una carpeta llamada *pipes*.

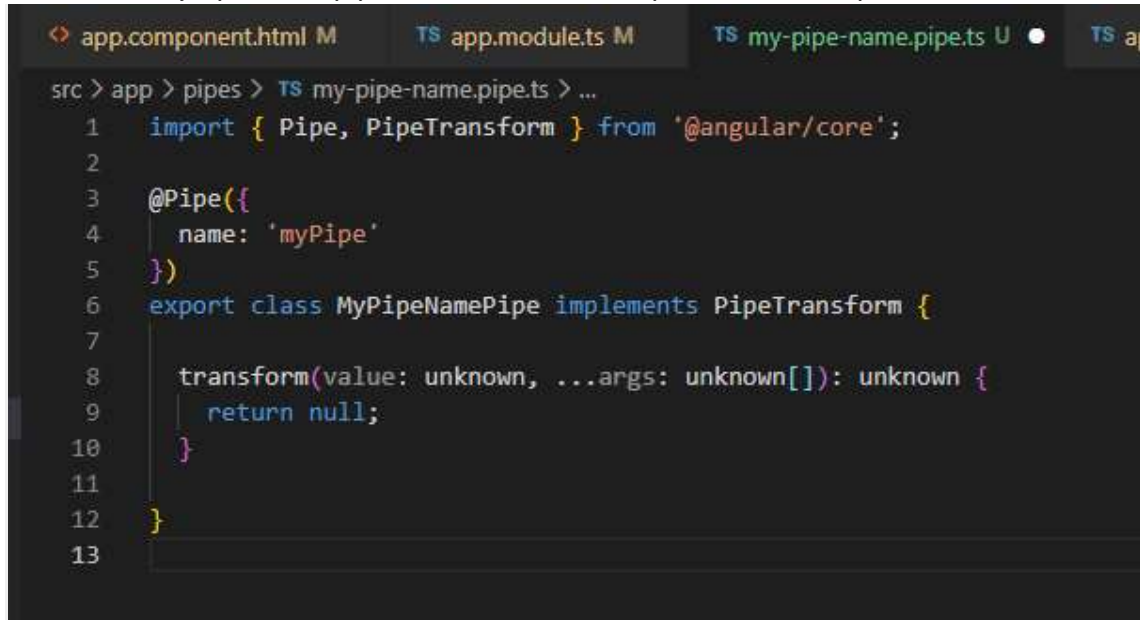


Dentro de dicha carpeta es donde se creara este *pipe*, que debe tener este formato: `myPipeName.pipe.ts`

El *.pipe* no es obligatorio, pero es una **convención**. Si utilizas vsCode tendrás la ventaja de que el nombre de tu carpeta *pipes* es reconocido por vsCode y le asigna un icono

único. Lo mismo ocurre con el archivo *myPipeName.pipe.ts*.

Dentro del *myPipeName.pipe.ts* escribimos su esqueleto básico, que sería:



```
src > app > pipes > TS my-pipe-name.pipe.ts > ...
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'myPipe'
5  })
6  export class MyPipeNamePipe implements PipeTransform {
7
8    transform(value: unknown, ...args: unknown[]): unknown {
9      return null;
10   }
11
12 }
13
```

Puede generarse este esqueleto básico con alguno de los *snippets* de Angular. Visual Studio Code tiene extensiones al respecto.

*Podemos dejar nuestro pipe tal y como viene, sólo con su contenido (value) o añadirle un parámetro según nuestras necesidades.*

Para este ejemplo, nuestro *pipe* se llama simplemente *myPipe*, así que se lo cambiamos. También le añadimos un *return* simple para que no nos de error:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'myPipe'
})
export class MyPipeNamePipe implements PipeTransform {

  transform(value: string): string {

    return 'Trabajando con Angular 12';
  }

}
```

2. Informamos a Angular de que hemos creado este pipe. Para ello, vamos al archivo *app.module.ts* e importamos ahí este pipe. Se añade también en el array *declarations*. El archivo *app.module.ts* nos queda así:

```

src > app > TS app.module.ts > AppModule
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6
7  import { LOCALE_ID } from '@angular/core';
8  import localeEs from '@angular/common/locales/es';
9  import { registerLocaleData } from '@angular/common';
10 import { MyPipeNamePipe } from './pipes/my-pipe-name.pipe';
11
12 registerLocaleData(localeEs);
13
14 @NgModule({
15   declarations: [
16     AppComponent,
17     MyPipeNamePipe
18   ],
19   imports: [
20     BrowserModule,
21     AppRoutingModule
22   ],
23   providers: [{ provide: LOCALE_ID, useValue: 'es' }],
24   bootstrap: [AppComponent]
25 })
26 export class AppModule { }
27

```

Y con estos cambios, ya podemos aplicar la **lógica** de programación que necesitemos en el archivo *myPipeName.pipe.ts*, para posteriormente aplicar nuestro *pipe* personalizado a cualquier variable, igual que aplicamos un *pipe* de los que vienen con Angular por defecto.

## Creación automática de un pipe personalizado mediante Angular CLI

Ahora que ya hemos visto cómo crear un pipe de manera manual, vamos a crearnos uno usando Angular CLI. El objetivo de este pipe será poder incrustar vídeos de forma segura sin que Angular nos los bloquee. El problema, es que por defecto, Angular bloquea los vídeos por motivos de seguridad. Se hará una prueba:

1. Accesar a youtube y copiar el **iframe** del vídeo que se desee, en este caso:

```

<iframe width="560" height="315" src="
https://www.youtube.com/embed/lbE7WsuK9rI" frameborder="0"
allow="accelerometer; autoplay; encrypted-media; gyroscope;
picture-in-picture" allowfullscreen></iframe>

```

Se agrega en el archivo *app.component.html*. Aquí el bloque de código que se ha añadido:

```

<tr>
  <td>YouTube video</td>

```

```

<td></td>
<td><iframe width="560" height="315" src="
https://www.youtube.com/embed/lbE7WsuK9rI" frameborder="0"
allow="accelerometer; autoplay; encrypted-media; gyroscope;
picture-in-picture" allowfullscreen></iframe>
</td>
</tr>

```

Se observa en el navegador que el vídeo se muestra sin problemas. Esto funciona bien porque el código es **estático**. El vídeo siempre será el mismo porque así lo hemos escrito en *hardcoded*. El bloqueo por parte de Angular vendrá cuando queramos mostrar un vídeo de manera **dinámica**, es decir, que el vídeo a mostrar vendrá de un código configurado en el archivo de typescript, no en el HTML.

2. Para hacer la prueba, cortamos el código del vídeo, la parte subrayada:

*https://www.youtube.com/embed/LbE7WsuK9rI*

y lo declaramos como el **valor** de una variable en nuestro *app.component.ts*. A esa variable la llamamos *video*, por ejemplo. Aquí el bloque de código añadido:

```
video = 'lbE7WsuK9rI';
```

3. Volvemos al *app.component.html* y le pasamos la variable *video* mediante *string interpolation*. Aquí el bloque de código que hemos editado:

```

<tr>
  <td>YouTube video</td>
  <td></td>
  <td><iframe width="560" height="315"
src="https://www.youtube.com/embed/{{ video }}" frameborder="0"
allow="accelerometer; autoplay; encrypted-media; gyroscope;
picture-in-picture" allowfullscreen></iframe>
  </td>
</tr>

```

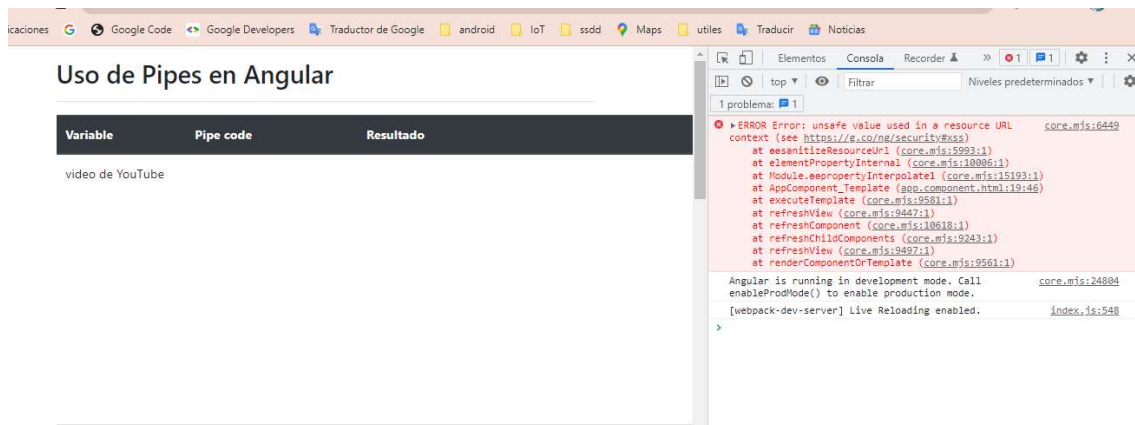
Se observa que el vídeo ha dejado de verse en el navegador, y además, la consola nos devuelve este **error**:

```

ERROR Error: unsafe value used in a resource URL context (see
http://g.co/ng/security#xss)
at DomSanitizerImpl.sanitize (platform-browser.js:3003)

```

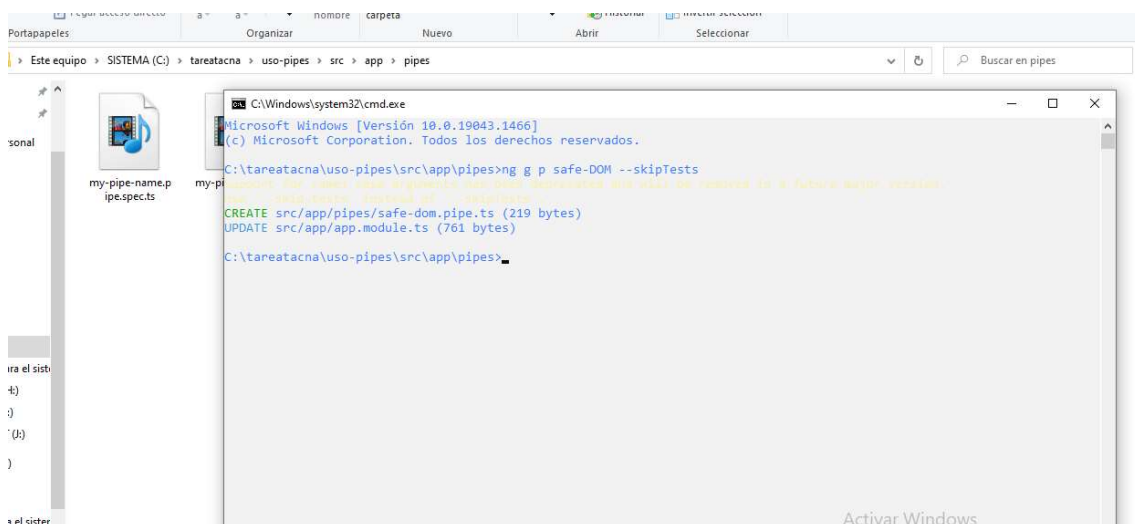




Para solucionar esto, vamos a crearnos nuestro propio *pipe*, al que llamaremos **safeDOM**.

1. En la carpeta *pipes*, nos creamos la **base** de un *pipe* mediante el Angular CLI, usando el comando:

`ng generate pipe safe-DOM --skipTests`, o su abreviatura `ng g p safe-DOM --skipTests`.



2. En el archivo recién creado *safe-dom.pipe.ts* importamos un paquete llamado *DomSanitizer* y lo declaramos en el **constructor** para poder usarlo.

3. Observa que la función *transform* viene con un contenido (*value*) y un parámetro (*...args*):

`transform(value: any, ...args: any[])`

En realidad, "*...args*" significa que acepta todos los **parámetros** que quieras añadirle. Nosotros sólo necesitamos uno: la *url*, que en nuestro caso será la dirección de youtube. Así que sustituimos "*...args*" por "*url*", que será de tipo *string*.

El *value* será el código identificador del vídeo.

4. Hacemos un *return* del objeto *domSanitizer* y le aplicamos uno de sus **métodos**. Dentro del método es donde verificamos que nuestro *value* y la *url* sean seguros y válidos.

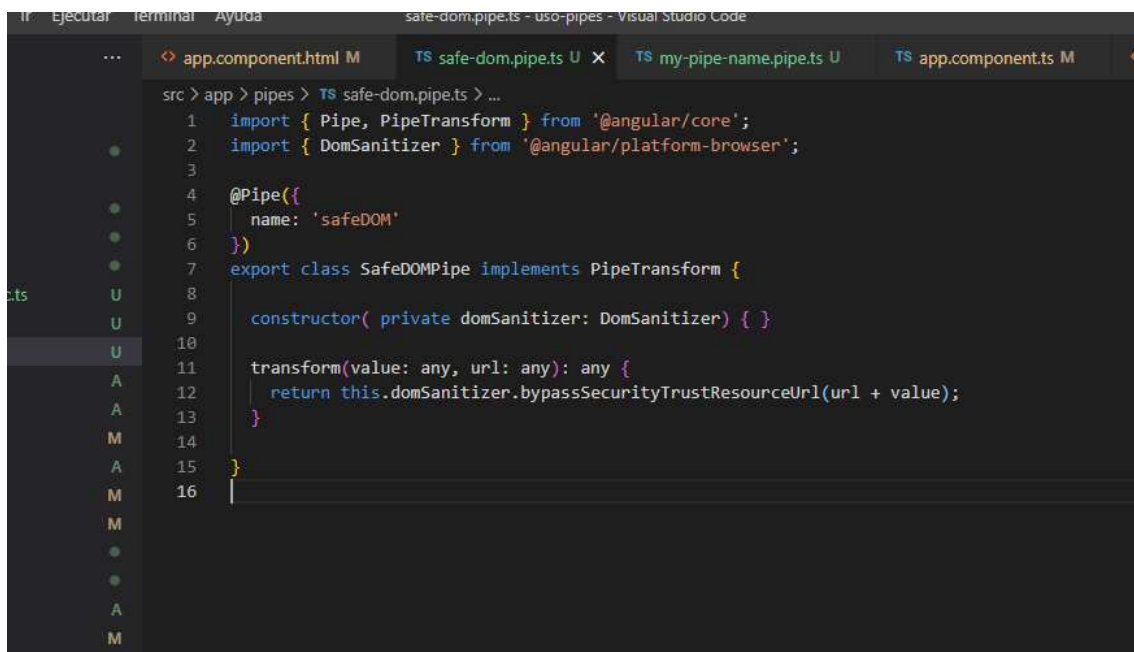
Aquí el archivo *safe-dom.pipe.ts* con los cambios aplicados:

```
import { Pipe, PipeTransform } from '@angular/core';
import { DomSanitizer } from '@angular/platform-browser';

@Pipe({
  name: 'safeDOM'
})
export class SafeDOMPipe implements PipeTransform {

  constructor( private domSanitizer: DomSanitizer) { }

  transform(value: any, url: any): any {
    return this.domSanitizer.bypassSecurityTrustResourceUrl(url
+ value);
  }
}
```



5. En el archivo *app.component.html*, le aplicamos nuestro *pipe* *safeDOM* a la variable *video*. Lo hacemos usando *property binding* sobre el atributo *src* del *iframe*. Le pasamos un parámetro a nuestro *pipe* (el *pipe* espera una *url* como parámetro). En nuestro caso, es la **dirección** de youtube. Aquí el bloque de código que hemos editado:

```

<tr>
  <td>YouTube video</td>
  <td></td>
  <td><iframe width="560" height="315" [src]="video |
safeDOM: 'https://www.youtube.com/embed/'" frameborder="0"
allow="accelerometer; autoplay; encrypted-media; gyroscope;
picture-in-picture" allowfullscreen></iframe>
  </td>
</tr>

```

De esta manera, el vídeo vuelve a estar **visible**, y ahora se está cargando dinámicamente.

The screenshot shows a web browser window with the title "Uso de Pipes en Angular". Below the title is a table with three columns: "Variable", "Pipe code", and "Resultado". The table contains three rows of data. To the right of the table is a video player showing a video titled "Introducción al Internet de las Cosas...". The video player has a progress bar and a play button.

Variable	Pipe code	Resultado
Video de Youtube		
Jorge Guerra	uppercase	JORGE GUERRA
Jorge Guerra	lowercase	jorge guerra
Jorge Guerra	slice:0:4	Jorg

## Un último *pipe* personalizado

Una vez mas hagamos otro *pipe* personalizado. Esta vez, uno que muestre y **oculte** una contraseña al hacer el click en un botón. Es decir, que sustituya esto:

contraseña: *mySuperSecretPassword123*  
por esto:

contraseña: \*\*\*\*\*

## Instrucciones:

- el *pipe* debe recibir un parámetro de tipo *boolean*, que por defecto será *true*, es decir, por defecto la contraseña estará oculta.
  - pista: usar el método *replace*.
- Comenzaremos a codificar el ejemplo:

1. Nos creamos la variable *password* en el archivo **app.component.ts** y el *boolean* que recibirá como parámetro nuestro *pipe*. El *boolean* nos sirve para indicar si el *pipe* está **activado** o desactivado. Por defecto estará activado, así que el *boolean* será *true*. Esto sigue la simple lógica de que una contraseña siempre estará encriptada por defecto (con los típicos asteriscos), y sólo dándole a un botón que diga

"mostrar" podremos ver su contenido.

Aquí el bloque de código que hemos añadido:

```
password = 'mySuperSecretPassword123';  
enable = true;
```

2. Nos creamos el *pipe* con Angular CLI, al que llamaremos **encrypt**, por ejemplo. Lo creamos dentro de la carpeta *pipes*.

*ng g p encrypt --skipTests*

```
C:\tareatacna\uso-pipes\src\app\pipes>ng g p encrypt --skipTests  
Support for camel case arguments has been deprecated and will be removed  
Use '--skip-tests' instead of '--skipTests'.  
CREATE src/app/pipes/encrypt.pipe.ts (219 bytes)  
UPDATE src/app/app.module.ts (830 bytes)  
  
C:\tareatacna\uso-pipes\src\app\pipes>_
```

Este comando nos debería haber generado un esqueleto básico:

```
import { Pipe, PipeTransform } from '@angular/core';  
  
@Pipe({  
  name: 'encrypt'  
})  
export class EncryptPipe implements PipeTransform {  
  
  transform(value: any, ...args: any[]): any {  
    return null;  
  }  
}
```

3. Vamos al *app.component.html* y nos creamos el bloque de HTML donde vamos a usar el *pipe*. Como hemos mencionado, nuestro *pipe* va a recibir un **parámetro**, así que lo añadimos. Añadimos también un **botón** con un *click event* que hará visible o invisible la contraseña. Por ahora, simplemente mostrará el estado del *boolean* (*true* o *false*).

Aquí el bloque de código que hemos añadido:

```
<tr>
  <td> Mi password: </td>
  <td>
    encrypt:{{ enable }}
    <button class="btn btn-info" (click)="enable
= !enable">Mostrar/Ocultar Clave</button>
  </td>
  <td>{{ password | encrypt }} </td>
</tr>
```

Así es como se ve nuestro código en el navegador. Si haces click en el botón, debería alternar el estado del *boolean* a cada click. Pero nuestra contraseña no se muestra en ningún sitio, porque aún no hemos **configurado** el *pipe*.



4. Vamos a configurar el *pipe* en su archivo, el *encrypt.pipe.ts*. Lo que se quiere hacer con el valor (*value*) sobre el que aplicamos el *pipe* es convertir cada caracter en un asterisco, así que usamos el método **replace** y una sencilla *regular expression* para que JavaScript recorra todos los caracteres de nuestro valor y busque si hay letras o números en él.

De haberlos, le indicamos que lo **sustituya** por el símbolo del asterisco. Hemos dicho que queremos que nuestro *pipe* reciba un parámetro. Este parámetro le servirá al *pipe* para saber si debe aplicarse o no. Lo que queremos es que el *pipe* oculte la contraseña y la convierta en asteriscos sólo si está visible, y a la inversa: que la muestre sólo si está encriptada (con asteriscos). Esto tiene pinta de que vamos a tener que usar

un ***if statement***.

5. Como decíamos, le añadimos un parámetro llamado ***hidden***, que es *true* por defecto, porque ese es el estado inicial en el que queremos que se muestre nuestra contraseña (*hidden* = *oculto*). Así que englobamos lo que le ocurrirá al valor si el *pipe* si su parámetro *hidden* es *true*, cosa que ocurre por defecto.

6. Por último, hacemos un *return* de nuestro *value* para que la función sea válida. Aquí cómo ha quedado el código del *encrypt.pipe.ts*:

```
import { Pipe, PipeTransform } from '@angular/core';
```

```

@Pipe({
  name: 'encrypt'
})
export class EncryptPipe implements PipeTransform {

  transform(value: string, hidden: boolean = true): any {

    if (hidden) {
      value = value.replace(/[a-zA-Z1-9]/gi, '*');
    }

    return value;
  }
}

```

la letra g coincide con letras mayúsculas y minúsculas, pero se asegura de que sean mayúsculas o minúsculas.

la letra i al final hace que la regex no distinga entre mayúsculas y minúsculas, lo que significa que no importa si la letra que encuentra es mayúscula o minúscula.

7. Volvemos al *app.component.html* y le añadimos el parámetro a nuestro *pipe encrypt*. Recuerda que el *pipe* espera un parámetro de tipo **boolean**, y para eso nos hemos creado la propiedad *enable*. Ya que *enable* es una propiedad que cambia su valor (*true* o *false*) dinámicamente al clicar el botón, también cambiará de valor de forma dinámica como parámetro del *pipe*, **alternando** entonces su activación.

Configuramos también nuestro **botón** para que muestre una frase u otra según la contraseña esté visible o encriptada. Aquí el bloque de código que hemos editado:

```

<tr>
  <td> Mi password: </td>
  <td>
    encrypt:{{ enable }}
    <button class="btn btn-info" (click)="enable = !enable">
      <p *ngIf="enable" class="mb-0">Mostrar Clave</p>
      <p *ngIf="!enable" class="mb-0">Ocultar Clave</p>
    </button>
  </td>
</tr>

```

```

</td>
<td>{{ password | encrypt:enable }} </td>
</tr>

```

Y produce el resultado final.

The screenshot shows a web interface with two sections. The top section has a label 'Mi password:', a text input, a button labeled 'Mostrar Clave' (Show Key), and a masked password '\*\*\*\*\*'. The bottom section has a label 'Mi password:', a text input containing 'mySuperSecretPassword123', a button labeled 'Ocultar Clave' (Hide Key), and a masked password '\*\*\*\*\*'. There are also some navigation links and a 'Ver en YouTube' button at the top.

Al terminar este laboratorio, se muestran los archivos generados y modificados:

#### a. app.module.ts

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

import { LOCALE_ID } from '@angular/core';
import localeEs from '@angular/common/locales/es';
import { registerLocaleData } from '@angular/common';
import { MyPipeNamePipe } from './pipes/my-pipe-name.pipe';
import { SafeDOMPipe } from './pipes/safe-dom.pipe';
import { EncryptPipe } from './pipes/encrypt.pipe';

registerLocaleData(localeEs);

@NgModule({
  declarations: [
    AppComponent,
    MyPipeNamePipe,
    SafeDOMPipe,
    EncryptPipe
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],

```

```

    providers: [{ provide: LOCALE_ID, useValue: 'es' }],
    bootstrap: [AppComponent]
  })
  export class AppModule { }

```

## b. app.component.html

```

<div class="container my-3">
  <h2>Uso de Pipes en Angular</h2>
  <hr>

  <table class="table table-hover">
    <thead class="thead-dark">
      <tr>
        <th scope="col">Variable</th>
        <th scope="col">Pipe code</th>
        <th scope="col">Resultado</th>
      </tr>
    </thead>
    <tbody>

      <tr>
        <td>Video de Youtube</td>
        <td></td>
        <td><iframe width="560" height="315" [src]="video |
safeDOM:'https://www.youtube.com/embed/'" frameborder="0"
allow="accelerometer; autoplay; encrypted-media; gyroscope;
picture-in-picture" allowfullscreen></iframe>
        </td>
      </tr>

      <tr>
        <td>Mi password: </td>
        <td>
          encrypt:{{ enable }}
          <button class="btn btn-info" (click)="enable = !enable">
            <p *ngIf="enable" class="mb-0">Mostrar Clave</p>
            <p *ngIf="!enable" class="mb-0">Ocultar Clave</p>
          </button>
        </td>
        <td>{{ password | encrypt:enable }} </td>
      </tr>
    </tbody>
  </table>

```



```

<tr>
  <td>{{ name }} </td>
  <td>uppercase </td>
  <td>{{ name | uppercase }} </td>
</tr>

<tr>
  <td>{{ name }} </td>
  <td>lowercase </td>
  <td>{{ name | lowercase }} </td>
</tr>

<tr>
  <td>{{ name }} </td>
  <td>slice:0:4 </td>
  <td>{{ name | slice:0:4 }} </td>
</tr>

<tr>
  <td>{{ myArray }} </td>
  <td>slice:0:4 </td>
  <td>{{ myArray | slice:0:4 }} </td>
</tr>

<tr>
  <td>{{ myArray }} </td>
  <td>*ngFor="let number of myArray | slice:0:3" </td>
  <td>
    <ul>
      <li *ngFor="let number of myArray | slice:0:3">{{ number }}
</li>
    </ul>
  </td>
</tr>

<tr>
  <td>{{ PI }} </td>
  <td>number</td>
  <td>{{ PI | number }} </td>
</tr>

<tr>
  <td>{{ PI }} </td>
  <td>number:'1.0-2'</td>
  <td>{{ PI | number:'1.0-2' }} </td>
</tr>

<tr>
  <td>{{ myVariable }} </td>
  <td>number:'1.0-2'</td>
  <td>{{ myVariable | number:'.0-0' }} </td>
</tr>

<tr>
  <td>{{ myNum }} </td>
  <td>percent</td>

```

```

        <td>{{ myNum | percent }} </td>
    </tr>
    <tr>
        <td>{{ myNum }} </td>
        <td>percent:'2.2-2'</td>
        <td>{{ myNum | percent:'2.2-2' }} </td>
    </tr>
    <tr>
        <td>{{ salario }} </td>
        <td>currency</td>
        <td>{{ salario | currency }} </td>
    </tr>
    <tr>
        <td>{{ salario }} </td>
        <td>currency:'PEN'</td>
        <td>{{ salario | currency:'PEN' }} </td>
    </tr>

    <tr>
        <td>{{ salario }} </td>
        <td>currency:'EUR':'code'</td>
        <td>{{ salario | currency:'EUR':'code' }} </td>
    </tr>
    <tr>
        <td>{{ salario }} </td>
        <td>currency:'EUR':'symbol':'4.0-0'</td>
        <td>{{ salario | currency:'EUR':'symbol':'4.0-0' }} </td>
    </tr>

    <tr>
        <td> {{ novato }} </td>
        <td>json</td>
        <td><pre>{{ novato | json }} </pre> </td>
    </tr>

    <tr>
        <td> {{ myDate }} </td>
        <td>date</td>
        <td>
            <pre>{{ myDate | date }} </pre>
        </td>
    </tr>
    <tr>
        <td> {{ myDate }} </td>
        <td>date:'medium'</td>
        <td>
            <pre>{{ myDate | date:'medium' }} </pre>
        </td>
    </tr>

```

```

    <tr>
      <td> {{ myDate }} </td>
      <td>date:'EEE, d of LLL of yyyy'</td>
      <td>
        <pre>{{ myDate | date:'EEE, d of LLL of yyyy' }} </pre>
      </td>
    </tr>

  </tbody>
</table>

</div>

```

### c. app.component.ts

```

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Uso de Pipes';
  name = 'Jorge Guerra';
  myArray = [1, 2, 3, 4, 5, 6, 7, 8];

  PI = Math.PI;

  myVariable = 123456789

  myNum = 0.589;

  salario = 3500.5;

  novato = {
    name: 'Elena Soto',
    alias: 'prima',
    song: 'Hurt So Good',
    skills: ['divertida y graciosa', 'Come mucha pizza'],
    youtubeChannel: 'Riete con Elena',
    address: {
      street: 'Av Universitaria',
      number: 3567,
      city: 'Lima'
    }
  }
}

```

```
    }  
  };  
  
  promiseValue = new Promise((resolve, reject) => {  
    setTimeout(() => {  
      resolve('dato X se muestra!');  
    }, 3000);  
  });  
  myDate = new Date();  
  
  video = 'lbE7WsuK9rI';  
  
  password = 'mySuperSecretPassword123';  
  enable = true;  
  
}
```