# EcmaScript5

Programación OO

`sirve para asignar valores a cadenas conretornosde carro

let/const

Funcion flecha zx=> 2 \* x equivale a funciones lambda de java¿? si la función no tiene argumentos () y si son varios (x,y) si son varias lineas se abre un bloque{ . Se puede guardar la función como parámetro o asignar a una variable.

for.. of iteración mejorada

Sintaxis corta de objeto

Operador ... Rest y spread. Explota el array en los distintos elementos y al revés

Valores por defecto en los parámetros opcionales function ejemplo (nombre="srr")

Destructuring

Clases: herencia, constructores, métodos de instancia, métodos estáticos,....g

get y set

Modulos: export, imports (variables y funciones) pudiendo renombrar

....

# TypeScript

Como javascript pero con tipado

No es ejecutable, hay que compilar con un compilador (tsc) que convierte el typescript a JS

Tipos

let cadena: string

number en general

Arrays

Array<number>

number[]

Hay Interfaces

Tiene visibilidad public y private

Constructores breves, si un argumento de un constructor se le pone public o private crea un atributo de clase y le asigna el valor del argumento

Tipos union y enumerados

# Angular

Instalamos Angular

Luego ejecutamos npm que viene con angular para que instale cli de angular y eso no spermite el comando ng

Visual studio code

Proyecto

carpeta e2e: testeos de selenium

Src: donde se encuentra el proyecto

gitignore: configuración de git para no subir algunas carpetas agit

editorconfig: configuración del editor

angular.json configuración de angular

browserlist para especificar al compilador la versión de navegador que queremos

karma.conf.js: test unitarios

package... versiones que necesita nuestro proyecto (que se descargan a la carpeta node\_modules) lo usa npm para descargarse los ficheros

Readme.md información de ayuda para node

tsconfig.app.json configuración del compilador para typescript

tslint.conf: revisa el estilo de configuración

ng serve dentro de la carpeta arranca el servidor en http://localhost:4200/

recompila y sube automaticamente

para proyectos importados npm install para que se descargue las librerías

Una app Angular:

* Modulos. Toda app necesita un módulo app.
* Componentes

## Modulos

Toda app necesita un módulo app mínimo: app.module.ts y por diseño podemos tener más (facturación...

Un módulo es una clase con una anotación. No tiene código solo es para organizarse una modulo lleva una anotación @ngModule que puede recibir parámetro objeto de javascript para configurar el módulo

## Componentes

Tiene cuatro archivos spec(para testerar) html, css y ts (javascript-typescript). Es una clase on otra anotación @Component. con parámetros como el html del componente y el css del componente (o varios) y con selector se estable una etiqueta (p.e <app-root></app-root>)que es donde se reemplazará el html del componente

Para crear un componente ng generaten enla carpeta del proyecto y esto crea ficheros dentre de la carpeta App. Tambien actualiza el módule.ts

ng generate component layout/cabecera

ng g c layout/cabecera --> es igual

En Index.html solo podemos poner la etiqueta del root.

Un componente puede estar dentro de otro

Cuando compila para producción todos los css los mete en uno solo, las css que modifiquemos a un componenten solo afectan a ese.

Un componente puede repetirse en una página las veces que quiera

Desde el html del componente podemos acceder a cualquier cosa publica del typescript del componente. Con doble llave {{}} se puede acceder al valor de una propiedad. Para atributos se puede usar [] en la propiedad html. Esto provoca que queden enlazados de forma que si la propiedad cambia se cambia automáticamente. esto es 'one way data binding'. Permite expresiones y llamadas a métodos. Se puede hacer dinámico el class del elemento html y utilizar el punto para determinadas clases y lo mismo con el estilo

<input [class.required]="isRequired"

Las expresiones tienen prohibido crear asignaciones a variables ni new ni encadenar expresiones. Que no provoquen efectos colaterales y que sean rápidas.

one way data binding en el otro sentido se hace con eventos quitando el prefijo 'on' usando paréntesis

<button (click)="cambiarNombre('Pepito'); otrometodo()" >Haz click</button>

Todos los eventos tienen información sobre el evento con la variable $event

<input type="text" [value]="apellidos" (keyup)="eventokeyup($event)"/>

doble binding de forma manual

<input type="text" [value]="apellidos" (keyup)="updateApellido($event.target.value)"/>

Doble binding de forma automática

Se usa con [(ngModel)]

Pero hay que importar en el app.models.ts import { FormsModule } from '@angular/forms';

@NgModule({

  declarations: [

    AppComponent,

    CabeceraComponent,

    PieComponent,

    MenuComponent,

    BindingComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    FormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

Ya podemos usarlo

<input [(ngModel)]="apellido"/>

Cada inclusión de componente tiene su propio ámbito de variables.... Ya veremos como intercambiar información entre componentes

## Directivas

Las directivas de componentes son los componentes

Las directivas estructurales:

En HTML son:

ngif: Elmina o añade del DOM un elemento. Si evalua true el elemento existe y si no no. empiezan con \*

<div \*ngIf="hero" class="name">{{hero.name}}</div>

ngFor: Repite una etiqueta conlo que haya dentro de la etiqueta tantas veces como diga el array

<ul>

<li \*ngFor="let hero of heroes">{{hero.name}}</li>

</ul>

El for tiene variables internas como first, index,last, even, odd para usarlas hay que asignarlas dentro de la declaración del for

Todo esto se puede aplicar a una etiquetas de componentes

ngSwitch

Permite seleccionar entre distintas opciones

<div [ngSwitch]="componente">

<app-texto \*ngSwitchCase="'texto'" ></app-texto>

<app-imagen \*ngSwitchCase="'imagen'"></app-imagen>

<app-tree \*ngSwitchCase="'tree'"></app-tree>

</div>

No se pueden usar dos sentencias estructurales if, for, switch en la misma etiqueta

La etiqueta <ng-container> no mete ningún elemento en el DOM pero permite usar estas sentencias estructurales

Se pueden crear más.

Directivas de atributo

Cambian el aspecto o el comportamiento sobre el que se aplica. Se pueden aplicar varias

* NgClass. Cambia las clases del elemento en función de las condiciones que se establezcan
* NgModel
* NgForm

Podemos hacer las nuestras : Un aclase con una anotación @directive. Podemos usar ng generate directive. Las meto en la carpeta directivas

ng generate directive directives/highlight

Se declara sola en app.module.ts

Ejemplo

@Directive({

  selector: '[appHighlight]'

})

Cuando te encuentres un atributo en un elemento HTML con este nombre aplica la directiva (p.e appHighlight)

importando ElementRef

nativeElement puede conseguir una referencia al objeto

import { Directive, ElementRef } from '@angular/core';

@Directive({

  selector: '[appHighlight]'

})

export class HighlightDirective {

  constructor(elRef:ElementRef) {

    elRef.nativeElement.style.backgroundColor='yellow';

  }

}

Los atributos con el atributo appHighlight se les cambia el color de fondo a amarillo.

Se pueden vincular métodos de la directiva a métodos

import { Directive, ElementRef, HostListener } from '@angular/core';

@Directive({

  selector: '[appHighlight]'

})

export class HighlightDirective {

  constructor(public elRef:ElementRef) {

  }

  highlight(color){

    this.elRef.nativeElement.style.backgroundColor=color;

  }

  @HostListener('mouseenter')

  onmouseenter(){

     this.highlight('yellow');

  }

  @HostListener('mouseleave')

  onmouseleave(){

    this.highlight('');

  }

Se puede @HostListener('mouseenter', ['$event', variables del elemento html donde se encuentra la directiva])

  @HostListener('mouseenter', ['$event'])

  onmouseenter(evento){

    console.log(evento);

     this.highlight('yellow');

  }

Una propiedad anotada con input se convierta en un atributo del html al que se lepuede pasar un string

<input appHighlight color="green" [class.required]="isRequired" value="{{ nombre }}"/>

export class HighlightDirective {

  @Input()

  color: string ;

  constructor(public elRef:ElementRef) {

    this.color='yellow';

  }

Input permite renombrar el atributo @Input('micolor')

Si solo tiene un input se puede hacer appHighlight="green"

Y también hacer esto  @Input('appHighlight')

  color: string ;

  constructor(public elRef:ElementRef) {

    this.color='yellow';

  }

Ejercicios https://carherco.es/curso-angular

Ejercicio dirctiva rotate... con lo que diga el atributo step y en rotate rotacion inicial

css transform= 'rotate(40deg)'

El ngOnInit(/) , lo normal es que si queremos implementarlo implementemos el interfaz OnInit. Se ejecuta una vez al inicio después del constructor, luego se asignan los valores del @Input (por eso no se pueden usar los valores procedentes del Input en el constructor)y luego se ejecuta el ngOnInit que es como el onReady en javascript

# Pipes

Son funciones de transformación de datos. Toman un dato y lo devuelven transformado. Operador |

Ejemplo de fechas. Angular te da algunas pipes hechas. Los argumentos se ponen con :

{{ dateObj | date }} // output is 'Nov 14, 2017'

{{ dateObj | date:'medium' }} // output is 'Nov 14, 2017, 10:43:11 AM'

{{ dateObj | date:'shortTime' }} // output is '10:43 AM'

{{ dateObj | date:'mmss' }} // output is '43:11'

Se pueden encadenar pipes

{{ birthday | date:'fullDate' | uppercase}}

Se pueden generar pipes con ng generate pipe. Tambien hay que declararlo en los componentes.

El nombre se puede acortar en el código. La clase pipe tiene que implementar un interfaz PipeTransform que tiene un método transform

En typescript las funciones se tipan por detrás

Pipes puras son las que cuando se le pasan arrays y no cambian no se cambia la referencia. Si se quiere otra cosa hay que decirselo con el atributo pure:false de la anotación @Pipe

Aquí falté durante unas horas.

La anotación @input puede utilizarse también en módulos de forma que se convierten en atributos de la etiqueta del módulo y en typescript se recoge en una variable pública.

Se pueden utilizar {...objeto} para clonar un objeto lo que pasa es que explota el objeto y con las {} se fuerza a que se cree uno nuevo

La anotación @output que es de tipo emisor de eventos, se convierte en un evento de la etiqueta del módulo

@output()

eliminarEvent: EventEmitter<User>;

En el código hay que llamar a eliminarEvet.emit(usuario) para lanzar el evento

El que usa el componente puede suscribirse al evento

(eliminarEvent)="eliminar($event)"

Existen librerías reutilizables de componentes angular.io resources

## Servicios

Se pueden crear servicios ng create service

No se declaran. Está declarada con @Injectable y se puede decir en la declaración en qué modulo está disponible el servicio

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

import { UserModule } from './user.module';

@Injectable({

providedIn: root,

})

export class UserService {

}

Si se pone root están disponibles en todos los sitios

En Angular se puden usar inyección de dependencias en los constructores de los módulos, tipando los parámetros con tipos de servicios. Y de forma pública o privada

Hay que guardarlo en el constructor

Hay un servicio router de la libería router que se llama así

router.navigate ('/galeria');

## Routing

Se crea el proyecto con routing a true

en app.routing.module.ts

const routes: Routes = [

  { path: 'galeria', component: GaleriaComponent },

  { path: 'binding', component: BindingComponent }

];

En la página html del menú

<ul>

    <li><a routerLink="/galeria">Galería</a></li>

    <li><a routerLink="/binding">Binding</a></li>

</ul>

En la página principal app.component.html

<a class="menu" [routerLink]="['/login']" routerLinkActive="selected">Login</a>

Para usar el servicio router en el codigo de un componente

this. router.navigate(['/galeria']); y esto cambia la url que aparece en el navegor y fuerza a que se revise otra vez el contenido de toda la página.

Para cargar los ejemplos del profesor

npm install desde la carpeta--> con esto se trae las dependencias

ng serve ejercicio01 --port 4002

el servicio debe ser utilizado desde los componentes

## Template Reference variables

Hay variables de plantilla que se pueden declarar en la plantilla. Se declaran con #

<input #valor2>

Y ya se puede usar en el mismo html

<p>{{valor2.value]]</p>

será del mismo tipo del elemento donde se ha declarado

Pero para Angular refresque la pantalla debe pasar alguna de estas cosas:

1) setTimeout()

2) setInterval()

3)Promise que es una forma gestionar asincronia .then()

p.e. ajax.get('url').then(respuesta

4)Observable otra cosa asincrona que se ejecutará cuando .suscribe();

5)Evento del domo al que estemos suscritos

Para pasar las variables de plantilla al código del componente podemos usar

 @ViewChild('valor2', {static: false})

  inputRef: ElementRef

  constructor() { }

  ngOnInit() {

    let objetoInput = this.inputRef.nativeElement;

  }

Los formularios suelen usar variables

## Formularios template-driven

La pipe json transforma un objeto en json

Para trabajar con formularios hay que hacer un import de Forms.module en el app.module.ts

El ngmodel es propio de este modulo

En javascript 6 si a un fichero se le pone expor en otro fichero se puede hacer un imports por ejemplo una constante de un array

A los elementos de formulario la directiva ngModel pone clases al elemento

El atributo class de un input se llama className en dom javascript

En el css del componente se puede escribir el código para esas clases ng-invalid,...

directivas de formularios: required, email,...

En un campo de formulario se puede declarar una variable y decir que extienda #name="ngModel"

Con eso conseguimos todo lo que tiene el imput mas lo que le da ngModel

Por ejemplo podemos leer el objeto errors que es un array de clave valor con claves como email requiered, etc

Y llamar a métodos

<p \*ngIf="name.invalid">el nombre es obligatorio</p>

y usar directivas de validación

<input type="text" id="name" name="name" email [(ngModel)]="usuario.name" #minombre="ngModel">

        <p>{{minombre.errors|json}}</p>

        <div \*ngIf="minombre.invalid">

          <div \*ngIf="minombre.errors.email">Mail incorrecto</div>

          <div \*ngIf="minombre.errors.required">Mínimo 8 caracteres</div>

        </div>

[(ngModel )] actualiza la pantall con el evento change

minombre.errors?.email Solo lo lee cuando no sea nulo (no sabemo si javascritp o typescript)

Una variable de un elemento formulario podemos hacer que herede de ngForm y de esta forma tenemos todo lo del formulario mas las validaciones a nivel de formulario

<form #formulario="ngForm">

<button [disabled]="formulario.invalid" type="submit">Submit</button>

El formulario tiene un evento ngSubmit que se dispara cuando hacemos click en el formulario que se lanza cuando se hace un submit del formulario que podemos programar en el componente

    <form #formulario="ngForm" (ngSubmit)="elsubmit()">

En el código

elsubmit(){

    alert('submit');

  }

## Observables

en la libreria rxjs es la que vamos aa usar (mirar package.json que son las librerías que se trae ngpm) es para observables. Se puede usar sin angular, es javascript

Sirven para manejar sincronia, se trata de tratar la información como un flujo de datos. Es programación reactiva cada vez que llega un mensaje se lo pasa a nuestro código (no pregunta, escuchar).

Se les suele poner un $ en el nombre

Nos da metodos para convertir objetos en flujos.

from genera un flujo a partir de un objeto p.e. un array

let observable$ = from ([0,1,2,3,4,5,6,-1]);

Necesitamos un observador que hay que indicar en tres funciones (al crearlo) (ninguna obligatoria)

1. La función que recibe el dato y hace algo con el dato

2.La función que dice qué hacer si ocurre un error en el flujo, recibe el error que ha llegado del flujo

3. Qué hacer en el caso en el que flujo termine. No tiene argumento de entrada

let observador = observer.create( // No es observer.create pero es algo parecido

          (dato) => {console.log(dato);

          (error) => {console.log(error);

          ()=>{console.log('El observable no va a publicar nada más');}

          }

        )

Ahora hay que decirle al observador que observe este flujo

observable$.subscribe(observador);

Tambien se puede hacer todo en uno

observable$.subscribe(

          (dato) => {console.log(dato);},

          (error) => {console.log(error);},

          ()=>{console.log('El observable no va a publicar nada más');}

        )

El códiog de las funciones se ejecutan en otro hilo

En http://reactivex.io además de observable , operator y subjects. Otra sección Languages se puede usar en otros lenguajes como por ejemplo Java

el método pipe de un observable permite utilizar un operador por ejemplo para filtrar datos (operador filter si cumple la condición es decir si devuelve true permite que pase si no filtra)

Enviroments.ts permite tener varios uno poara producción y otro para desarrollo por ejemplo y así especificar todo lo que dependa de tu entorno. Todas las propiedades que se pongan se pueden luego importar en otro sitio. Al hacer el import solo enviroment sin el sufijo de producción

Para llamar al servicio hay que importar HttpClientModule en el app.module y en el modulo que lo va a usar

HttpClient es un servicio de esa librería y se puede usar para hacer llamadas con .get<Objeto> y con parametros sirve para hacer una llamada ajax. El que llame a esta debería ser una función que devolviera un observable p.e.

getAll(): Observable<User[]> {

    return this.http.get<User[]>(this.url\_api);

  }

Recordar que los dos puntos es para decir el tipo de retorno

Si el servidor es reactivo podrías seguir recibiendo información (por ejemplo BD amazon)

La funcion of de rxjs permite programar un servicio que debe devolver unobservable pero todavía no está hecho

return of (USERS). pipe(delay(500))

además añado un retraso para simular que es un servidor

of

## Build

ng build sin nada hace un build con el entorno de desarrollo. Para otra cosa ng build --prod que optimiza el código (todo en una linea y renombrando funciones y variables)

Genera una carpeta dist con el nombre del poryecto y lcon los archivos:

\* un htML y una css y varios js

En assets se meten los recuros estaticos como los iconos .

En ts.config se le puede parametrizar la version de javascript per o cuando genera genera par a esa versión y otro js para una versión más antigua posible

En la configuracion de angular vienen todas las propiedades se podrían modificar al compilar.

Los nombres de los ficheros cambian tienen un hash que si cambia es diferente y ya no se usa el cacheado.

Para que funcionen ls rutas en la página html hay una etiqueta base con atributo href=" La ruta prefijo a utilizar"-->pegenet/pef.. Para que esto se automatico el build tiene un --base--href para establecer el valor.

En la documentación de angular hay una sección de deployment que cuenta todo esto

## Routing

Se puede generar aunque al crear el proyecto no se haya dicho.

El orden de las rutas importa

En routing.module se puede redirigir por ejmplo para la raiz '', redirectTo: 'home', pathMatch: 'igae' y le decimos que todo lo que empiece por igae.

El comodin '\*\*' que se suele poner al final para si ninguna ruta coincide se le manda a una página de error

Se pueden usar parámetros en la url 'usuario/edit/:id'. Hay mas parámetros de ruta

enabletrace = true hace que en la consola del navegador salga todas las rutas por donde ha pasado.

La librería de routing tiene un libro...

Se puede tenermas de un router outlet especificando

routerlink sirve para hacer links en los elementos a en lugar de href

routerlinactive es una directiva que permite poner un class si la ruta actual es la seleccionado

El servicio router que ya utilizamos para navegar(navigate)

El servicio Activated route permite acceder a la información de la ruta por ejemplo parámetros en la url

 private ruta:ActivatedRoute

Los Guard permiten controlar el acceso a rutas o controlar que un usuario salga sin guardar cambios. Son métodos que devuelven true si se quiere dejar hacer algo o false si no se quiere.

CanActivate decide si una ruta puede ser activada y CanDeActivate si no puede

Un guard es como un servicio (@Injectable) que tiene que implementar un interfaz p.e. CanActivate o los que quieras. Luego hay que configrarlo en las rutas diciendo cual o cuales tienen que cumplirse:

{path: "auth", component: AuthComponent, canActivate: [AuthenticatedGuard, AuthComponentGuard]

Para dejar que se vaya de un path

{path: "projects", component: ProjectsComponent, canActivate: [AdminGuard], **canDeactivate: [ConfirmDeactivateGuard]}**

Para creaar guard

ng g guard el path

Se puede elegir de las que salen pero en versiones antiguas no sale deactivate pero se puede hacer a mano en la clase que se crea

Con deactivate disponemos de la instancia del componente. Normalmente en el componente se hace un método público de forma que guard puede preguntar si es seguro salir o no.

Interesa crear un interfa con un metodo hasUnsavedChanged y se exporta y todos los que se deban controlar deben implementarlo

export interface HasUnsavedChanges{

  hasUnsavedChanges(): boolean;

}

export class ConfirmGuard implements CanDeactivate<any> {

  canDeactivate(component: any, currentRoute: ActivatedRouteSnapshot, currentState: RouterStateSnapshot, nextState?: RouterStateSnapshot): boolean | UrlTree | Observable<boolean | UrlTree> | Promise<boolean | UrlTree> {

    throw new Error("Method not implemented.");

    if (!component.hasUnsavedChanges()){

      return  window.confirm('Tienes cambios sin guardar seguro que quieres salir (S/N');

    }

    return true;

  }

En javascript se pemite decir que el tipo de un parámetro o retorno de una funcion sea de varios tipos

## Ciclo de vida de un componente

* **ngOnChanges():** Es llamado cuando Angular establece datos asociados a propiedades asociadas a @Input. El método recibe un objeto SimpleChanges con el valor actual y el valor anterior. Es llamado antes de ngOnInit() y cuando una o más variables @Input cambie.
* **ngOnInit():** Es llamado una única vez, después del primer ngOnChanges() cuando Angular ha renderizado las propiedades bindeadas y ha seteado las propiedades decoradas con @Input. En este momento es cuando se puede dar por **inicializado** el componente/directiva.
* **ngDoCheck():** Cada vez que refresca la pantalla. Es llamado durante cada ciclo de detección de cambios, después de cada ngOnChanges() y después de ngOnInit(). Puede servir para gestionar cambios que Angular no sea capaz de tratar por sí mismo.
* **ngAfterContentInit():** Es llamado después de que Angular proyecte el contenido externo dentro de la vista del componente. Es llamado después del primer ngDoCheck(). Sólo disponible para componentes.
* **ngAfterContentChecked():** Responde después de que Angular compruebe el contenido proyectado en el componente. Es llamado después de cada ngAfterContentInit() y de cada ngDoCheck(). Sólo disponible para componentes.
* **ngAfterViewInit():** Responde después de que Angular inicialice la vista del componente y las vistas hijas. Es llamado una sola vez después del primer ngAfterContentChecked(). Sólo disponible para componentes. Es el lugar seguro para trabajar con variables obtenidas con @ViewChild.
* **ngAfterViewChecked():** Responde después de que Angular compruebe la vista del componente y las vistas hijas. Es llamado después de ngAfterViewInit y de cada ngAfterContentChecked(). Sólo disponible para componentes.
* **ngOnDestroy():** Es llamado usto antes de que Angular destruya el componente. Se puede utilizar por ejemplo para desuscribirse de los observables.

Formularios reactivos: porque tienen observables esperando cosas.

Para trabajar con formularios reactivos hay que importar reactiveformsmodule. Se trata de trabajar con controles, arrays de controles y grupos de controles que luego vamos a poder leer en nuestro código. Con esto vamos a consiguir enlazar desde codigo objetos a controles del formulario (en lugar de hacerlo desde el HTML)

Para usarlo se utiliza el servicio FormBuilder. Todo formulario tiene que empezar siendo un grupo

export class ReactiveFormComponent implements OnInit {

  usuarioForm: FormGroup;

  constructor(private fb: FormBuilder) {

    this.createForm();

  }

  ngOnInit() {

  }

  createForm(){

    this.usuarioForm=this.fb.group(

      name: this.fb.control(this.usuario.name), // valor inicial

      address: this.fb.group({

        street: this.fb.control(this.usuario.street),// se puede hacer mas corto con this.usuario.street crea un control

        city: this.fb.control(this.usuarioForm.city)

      })

    )

  }

En el HTML está la directiva [formGroup]

<form [formGroup]="usuarioForm">

      <div>

        <label for="name">Name: </label>

        <input type="text" id="name" name="name" required formFontrolName="name">

      </div>

Angular asocia al input el objeto con la clave name del formulario usuarioForm

Para el control street en alguna etiqueta intermedia decir que está dentro del grupo address

<div formGroupName="address">

        <div>

          <label for="street">Street:

            <input type="text" id="street" name="street" formFontrolName="street">

          </label>

Siempre se puede utilizar en el html la variable usuarioForm que siempre estará actualizada.

Se puede poner un validador al crear un control

name: this.fb.control(this.usuario.name, [Validators.required,Validators.minLength(0)]), // valor inicial