Curso de TypeScript

Creado por Microsoft sobre el 2012 de código abierto. Superconjunto de JavaScript.

Permite una programación orientada a objetos más fuerte, limpia, potente y avanzada que la que lleva JavaScript.

Dentro del desarrollo de **aplicaciones** web en *tipado fuerte* hay una tendencia importante a las denominadas **SPA (**Single Page Apps), Angular2 (moderna, escalable.

Es un lenguaje que al final "transpila" el código TS y en JS.

Introducción

Que es e instalar TS.

Empezando con TypeScript: Referencias, Documentación, EC6, Variables, tipos de datos, tipado fuerte, diferencias entre let, var, crear funciones, usar los diferentes tipados, etc.

¿Qué es?

Se asemeja mucho a Java por el tipado fuerte. Trabajo con clases y objetos.

Nos permite usar el estándar EC6 por el Angular 2, como las variables let y var, usar imports en vez de includes.

Instalación

Instalar NODEJS https://nodejs.org/es/

Instalar WampServer

https://www.typescriptlang.org/

Utilizaremos el Sublime Text como IDE

Instalar Cygwin para ver como se muestra en Linux una consola.

Instalar TypeScript como pone la documentación. El –g del instalador es para que se instale de forma global en todo el Node. Hacer tsc –v para mostrar la versión.

Primeros pasos con TypeScript.

Crear Hola Mundo

Crear index.html básico: Declarar el javascript en el head para abrir consola y ver error.

```
<!DOCTYPE HTML>
<html>
	<head>
		<meta charset="utf-8"/>
		<title>Curso TypeScript</title>
	</head>
	<body>
		<h1>Curso de Typescript</h1>
		<section id="container">
		</section>
		<script type="text/javascript" src="prueba.js"></script>
	</body>
</html>
```

Crear prueba.ts

```
function holaMundo(nombre){
	return "Hola Mundo! Soy "+nombre;
}

var nombre = "Xavi García";
var resultado = holaMundo(nombre);
var etiqueta =<HTMLElement>document.getElementById("container");
etiqueta.innerHTML = resultado;
```

Pruebas con el compilador tsc prueba.ts

Para tener alerta al compilador y compile los cambios (tws –w *.ts)

Añadir al final del prueba.ts

Console.log(“Esto funciona”);

Variables y tipo de datos

Se puede indicar el tipo de dato de cada variable

Añadir antes del console para ver los tipos:

```
//Variables y tipos

var nombretwo:string="Xavier García";

var edad:number=23;

var programador:boolean = true;

var langs: Array<string>= ["PHP", "JavaScript", "CSS"];

etiqueta.innerHTML = nombretwo+" - "+edad;
```

Para forzar el error, se asigna un número a la variable string “nombre”, el compilador da el aviso del fallo. El sublime también da el aviso en el footer.

Revisar el resto de tipo de variables, añadiendo un string a la numbre o un número al array porque está declarado como array de strings.

El tipo ANY, permite trabajar con cualquier tipo de dato. Pero lo suyo es utilizar el tipado fuerte para asignar cada tipo de variable.

Diferencia entre Let y Var

La diferencia es el alcance de la variable, las let es para alcance corto (nivel de bloque)

Se escribe al final:

```
//Let y var

var a=7;

var b=2;

//El triple igual fuerza a que las variable sean iguales

// (99 == "99") será true

// (99 === "99") será false

if (a===7) {

        let a=4;

        var b=1;

        console.log('Dentro del IF: a='+a+" b= "+b );

}

console.log('Fuera del IF: a='+a+" b= "+b );
```

Funciones y tipado

Al final:

```
//funciones y tipado
function devuelveNumero(num:number){
        return num;
}

console.log(devuelveNumero(45));
```

Se añade :string, así da error porque el tipado fuerte no deja

```
//funciones y tipado
function devuelveNumero(num:number):string{
        return num;
}

console.log(devuelveNumero(45));
```

Si concatenamos un string al num entonces si

```
//funciones y tipado
function devuelveNumero(num:number):string{
        return “el número es el : “+num;
}

console.log(devuelveNumero(45));
```

A la inversa, recibiendo string y devolviendo number

```
function devuelveString(texto:string):number{

        if (texto=="hola"){

                var num = 66;

        }else {

                var num= 99;

        }

        return num;

}

console.log(devuelveString('Javier'));
```

```
console.log(devuelveString('hola'));
```

Como ejercicio: Recibiendo string y devolver booleana

```
//Recibe string y devuelve boolean
function devuelveBoolean(texto:string):boolean{
	if (texto=="hola"){
		var num = true;
	}else {
		var num= false;
	}
	return num;
}

console.log(devuelveBoolean('Javier'));
console.log(devuelveBoolean('hola'));
```

Como ejercicio: Recibe string y devuelve cualquier cosa.

```
//Recibe string y devuelve cualquier cosa
function devuelveCualquierCosa(texto:string):any{
	if (texto=="hola"){
		var num = 50;
	}else {
		 var num= "texto simple";
	}
	return num;
}

console.log(devuelveCualquierCosa('Javier'));
console.log(devuelveCualquierCosa('hola'));
```

Las clases, propiedades y métodos Cómo se utilizan, programación orientada a objetos

Para definir un objeto se utilizan las clases. Una clase es un molde para crear infinidad de objetos con características parecidas. Por ejemplo, se crea una clase coche pero después hay diferentes tipos de coche como color, nombre, caballos, motor, etc. En este caso la clase se llamara “nuevoCoche”.

Cada clase tendrá atributos/propiedades (color del coche, n puertas, tipo rueda, etc.)

Cada clase tendrá métodos (funciones): arrancar, acelerar, frenar, encender luces. Con los métodos se trabajan con los atributos y propiedades. Por ejemplo, el método acelerar cambiará los valores de la propiedad velocidad.

Que es el constructor de una clase, de un objeto,

Clases, Atributos y métodos

Parte 1: Conociendo la programación orientada a objetos (POO)

```
class Coche  {
        //Se definen las propiedades publicas
        public color: string;
        public modelo: string;


        //Funciones
        public getColor(){
                return this.color;
        }

        public setColor(color:string){
                this.color = color;
        }
}

var coche = new Coche(); //declaración de objeto
var cocheDos = new Coche(); //declaración de objeto
var cocheTres = new Coche(); //declaración de objeto
coche.setColor("Rojo");
cocheDos.setColor("Azul");
cocheTres.setColor("Amarillo");
console.log("El color del coche 1 es: " + coche.getColor());
console.log("El color del coche 1 es: " + cocheDos.getColor());
console.log("El color del coche 1 es: " + cocheTres.getColor());
```

Parte 2

Métodos

Son las características (funciones) de cada objeto

```
class coche  {
	//Se definen las propiedades publicas
	public color: string;
	public modelo: string;
	public velocidad: number = 0; //esto se debería hacer con un constructor que ya veremos pero hay que inicializarla para poder acelerar o frenar.

	//Funciones/métodos
	public getColor(){
		return this.color;
	}

	public setColor(color:string){
		this.color = color;
	}

	public acelerar(){
		this.velocidad++;
	}
	public frenar(){
		this.velocidad--;
	}

	public getVelocidad():number{
		return this.velocidad;
	}

}

var turismo = new coche(); //declaración de objeto
var furgoneta = new coche(); //declaración de objeto
var camion = new coche(); //declaración de objeto
turismo.setColor("Rojo");
turismo.acelerar();
turismo.acelerar();
turismo.acelerar();

furgoneta.setColor("Azul");
camion.setColor("Amarillo");
console.log("El color del coche 1 es: " + turismo.getColor());
```

```
console.log("El color del coche 2 es: " + furgoneta.getColor());
console.log("El color del coche 3 es: " + camion.getColor());
console.log("La velocidad del turismo es :"+turismo.getVelocidad());

turismo.frenar();
console.log("La velocidad del turismo es :"+turismo.getVelocidad());
```

El constructor

Es donde se inicializan las variables declaradas en la clase.

Antes de los métodos añadir:

```
//Contructor
        constructor(){
                this.color="Blanco";
                this.velocidad = 0;
                this.modelo = "BMW Generico";
                }
```

Evolución para cuando no llegue ningún modelo se establezca un modelo por defecto.

```
//Contructor
        constructor(modelo = null){
                this.color="Blanco";
                this.velocidad = 0;
                if(modelo==null){
                        this.modelo = "BMW Generico";
                } else {
                        this.modelo =modelo;
                }

        }
```

Se quita uno de los setColor para ver que por defecto coge el blanco.

Ejercicio: Añadir métodos para obtener y establecer modelo.

```
public getModelo(){
                return this.modelo;
        }

        public setModelo(modelo:string){
                this.color = modelo;
        }
```

Cómo funciona la visibilidad de propiedades y métodos

Propiedades públicas, privadas y protegidas.

Public: Llamar al método desde cualquier sitio.

Protected: Solo disponible en la clase y en las herederas “Extends”.

Private: Solo disponible en la clase, nadie más.

Como funcionan y para qué sirven las interfaces.

Declarar la necesidades que debe tener una clase, en este caso, se declaran las funciones que debe tener declaradas obligatoriamente la clase que usa esa interfaz.

```
interface cocheBase{
        getModelo():string;
        getVelocidad():number;
}
class coche implements cocheBase  {
```

Se comenta una de las funciones para ver que el compilador y el sublime avisan del error.

Qué es la herencia de clase y cómo funciona

Son nuevas clases (hijas) que heredan los métodos de la que se extiende (padre).

```
class Programa {
        public nombre:string;
        public version: number;

        getNombre(nombre:string){
                return this.nombre;
        }

        getVersion(){
                return this.version;
        }

        setNombre(nombre:string){
                this.nombre = nombre;
        }

        setVersion(version:number){
                this.version = version;
        }
}
```

```
class EditorVideo extends Programa {
	public timeline:number;

	setTimeLine(timeline:number){
		this.timeline = timeline;
	}

	getTimeLine(){
		return this.timeline;
	}
}

var editor = new EditorVideo();
editor.setNombre("Camtasia Studio");
editor.setVersion(8);
editor.setTimeLine(4000);
```

Se añade método para devolver todas las variables dentro de EditorVideo

```
getAllData(){
		return this.getNombre()+" - "+this.getVersion()+" - "+this.getTimeLine();
	}
```

Ejercicio en coches:

Añadir clase hybrid que extienda la de coche tipo de hybrid (serie, paralelo, combinado)

Añadir clase eléctrico hija de coche que añada autonomía en km. Hacer que se pueda declarar directamente el valor.

Ejemplo con TS.

Añadir programas desde el front con un elemento de formulario y la lista donde se va a mostrar los inputs.

```
<form>
                        <input type="text" name="nombre" id="nombre"
placeholder="Nombre del programa"/>
                        <input type="button" name="guardar" value="guardar"
onclick="guardar">
                </form>


                <h4>Listado de Programas</h4>
                <ul id="listado">
                </ul>
```

En programa.ts

```
//logica del formulario: Guardar elementos dentro de un array.

//Array para guardar los inputs, no se especifica tipado porque almacenará varios tipos
var programas:Array= [];

function guardar(){
        //guardar el valor del input
        var nombre =
(<HTMLInputElement>document.getElementById("nombre")).value.toString();
        var programa = new Programa();
        programa.setNombre(nombre);
        programa.setVersion(1);
        programas.push(programa);

        var list="";
        for (var i =0; i < programas.length; i++){
                list= list+"<li>"+programas[i].getNombre()+"</li>"
        }
        var listado = <HTMLElement>document.getElementById("listado");
        listado.innerHTML = list;
        (<HTMLInputElement>document.getElementById("nombre")).value = "";
}
```