

# TypeScript



# Temario

- Introducción ¿Por qué TypeScript?
- Instalación y configuración
- Herramientas IDE, proyectos
- Variables y Type Annotations
- Enums y Arrays
- Arrow Functions
- Definiendo y usando Function Types
- Definiendo Parámetros
- Sobrecarga de funciones
- Definiendo y usando Interfaces
- Interfaces para Function types
- Extendiendo Interfaces
- Implementando interfaces con Clases
- Creando y usando Clases
- Extendiendo Clases, Creando Clases abstractas
- Namespaces, Modules, Decorators,
- Usando Expresiones Clase
- Opciones de Compilación con tsconfig

**¿Por qué TypeScript?**

# ¿Por qué TypeScript?

- Nacimiento en 2012
- Transpilador
- Tipado fuerte
- Código fuente y código compilado
- Errores en tiempo de compilación
- Complementa JS
- Integración con el IDE
- Angular

# Entorno de desarrollo



TypeScript

# Entorno de desarrollo

- IDE: Visual Studio Code
  - EditorConfig for VS Code
  - TSLint
  - Configurar:
    - Format on Paste
    - Format on Save
    - tslint:autoFix on Save
- Git
- NodeJS y npm

# Git



# Comandos básicos

- Clonar un repositorio:

`git clone URL`

- Descargar última versión del repositorio:

`git pull origin master`



# Configuración proxy

```
git config --global http.proxy http://username:password@host:port
```

```
git config --global https.proxy http://username:password@host:port
```

# Node.js y npm



# npm

- Instalar última versión después de instalar Node.js (configurar proxy si es necesario): `npm install -g npm`
- Repositorio de módulos distribuibles
- Módulos globales y módulos locales
- La carpeta `node_modules`
- El archivo `package.json`:
  - Registro de dependencias
  - Dependencias de desarrollo y de producción
  - Versiones (SEMVER)

# Comandos npm

- Instalar un paquete globalmente:  
`npm install -g paquete`
- Instalar un paquete de producción:  
`npm install paquete`
- Instalar un paquete de desarrollo:  
`npm install paquete --save-dev`
- Instalar todas las dependencias:  
`npm install`
- Instalar las dependencias de producción:  
`npm install --production`
- Listar paquetes instalados:  
`npm list --depth=0`      (locales)  
`npm list -g --depth=0` (globales)

# Configuración proxy

```
npm config set proxy http://username:password@host:port
```

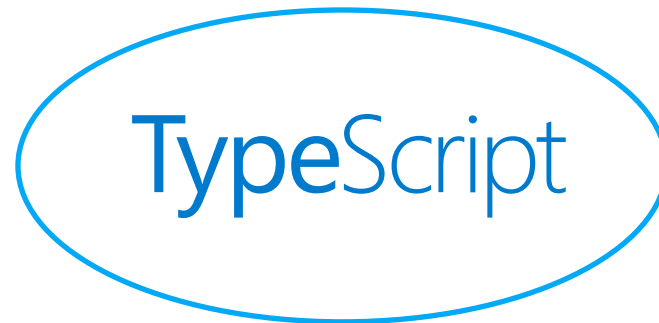
```
npm config set https-proxy http://username:password@host:port
```

# JavaScript



# JavaScript

- Interpretado, compilado y ejecutado en el navegador
- Cada navegador programa su propio motor de JS
- Estandarización: **ECMAScript**
- La versión **ES6** o **ES2015**
- Transpiladores: Babel, TypeScript



# Organización del código JavaScript

- ¿2000 líneas en un solo archivo?

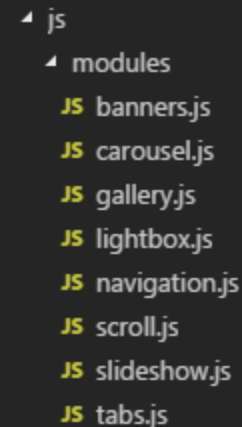
Ventajas	Inconvenientes
<ul style="list-style-type: none"><li>• Una sola petición HTTP</li></ul>	<ul style="list-style-type: none"><li>• Difícil de leer/entender</li><li>• Difícil de mantener</li><li>• Poca reusabilidad</li><li>• Difícil encontrar código no usado</li><li>• Colisiones de nombres</li></ul>



# Organización del código JavaScript

- Optimización: dividir el código en varios archivos/módulos

```
<head>
  <meta charset="UTF-8">
  <title>Mi web</title>
  <script src="vendor/jquery/jquery.min.js"></script>
  <script src="js/modules/tabs.js"></script>
  <script src="js/modules/banners.js"></script>
  <script src="js/modules/lightbox.js"></script>
  <script src="js/modules/scroll.js"></script>
  <script src="js/modules/carousel.js"></script>
  <script src="js/modules/slideshow.js"></script>
  <script src="js/modules/gallery.js"></script>
  <script src="js/modules/navigation.js"></script>
</head>
```



```
js
├── modules
│   ├── banners.js
│   ├── carousel.js
│   ├── gallery.js
│   ├── lightbox.js
│   ├── navigation.js
│   ├── scroll.js
│   ├── slideshow.js
│   └── tabs.js
```

```
<head>
  <meta charset="UTF-8">
  <title>Mi web</title>
  <script src="vendor/jquery/jquery.min.js"></script>
  <script src="js/modules/tabs.js"></script>
  <script src="js/modules/banners.js"></script>
  <script src="js/modules/lightbox.js"></script>
  <script src="js/modules/scroll.js"></script>
  <script src="js/modules/carousel.js"></script>
  <script src="js/modules/slideshow.js"></script>
  <script src="js/modules/gallery.js"></script>
  <script src="js/modules/navigation.js"></script>
</head>
```

```
└─ js
   └─ modules
      JS banners.js
      JS carousel.js
      JS gallery.js
      JS lightbox.js
      JS navigation.js
      JS scroll.js
      JS slideshow.js
      JS tabs.js
```

## Ventajas

- Legible e inteligible
- Fácil de mantener
- Reutilizable
- Cargamos sólo lo que necesitamos

## Inconvenientes

- Difícil encontrar código no usado (menos difícil que antes)
- Colisiones de nombres
- Muchas peticiones HTTP
- El orden importa: dependencias

# Organización del código JavaScript

- Dependencias: es difícil asegurar el orden, y no es posible tener dependencias circulares

```

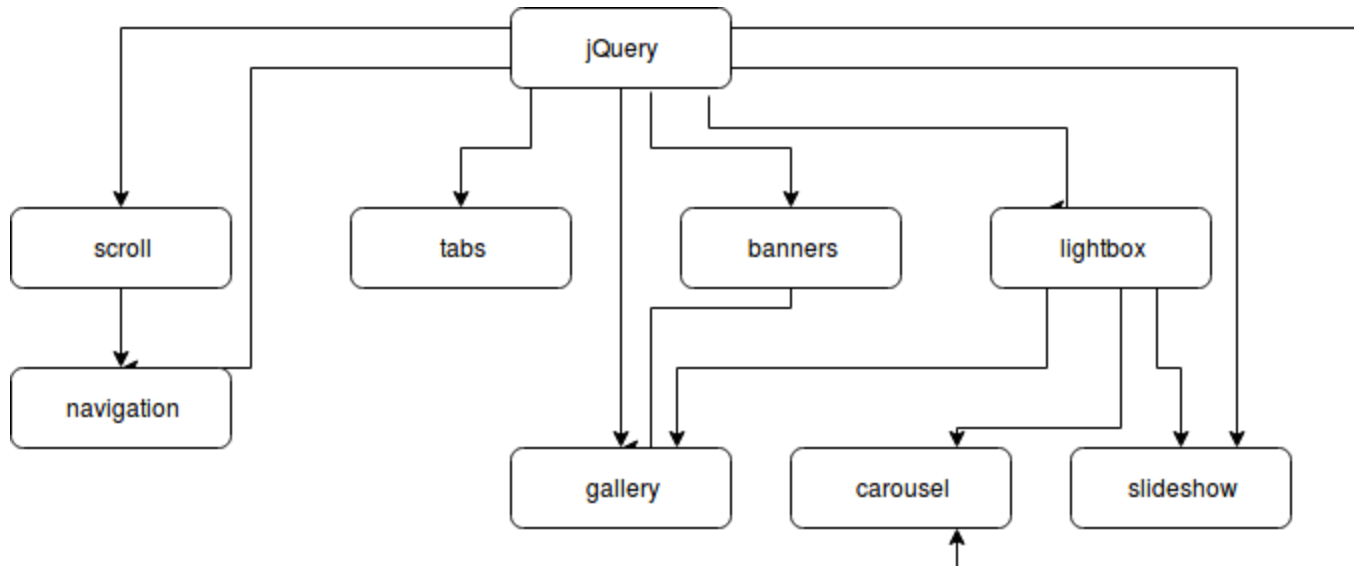
<head>
  <meta charset="UTF-8">
  <title>Mi web</title>
  <script src="vendor/jquery/jquery.min.js"></script>
  <script src="js/modules/tabs.js"></script>
  <script src="js/modules/banners.js"></script>
  <script src="js/modules/lightbox.js"></script>
  <script src="js/modules/scroll.js"></script>
  <script src="js/modules/carousel.js"></script>
  <script src="js/modules/slideshow.js"></script>
  <script src="js/modules/gallery.js"></script>
  <script src="js/modules/navigation.js"></script>
</head>

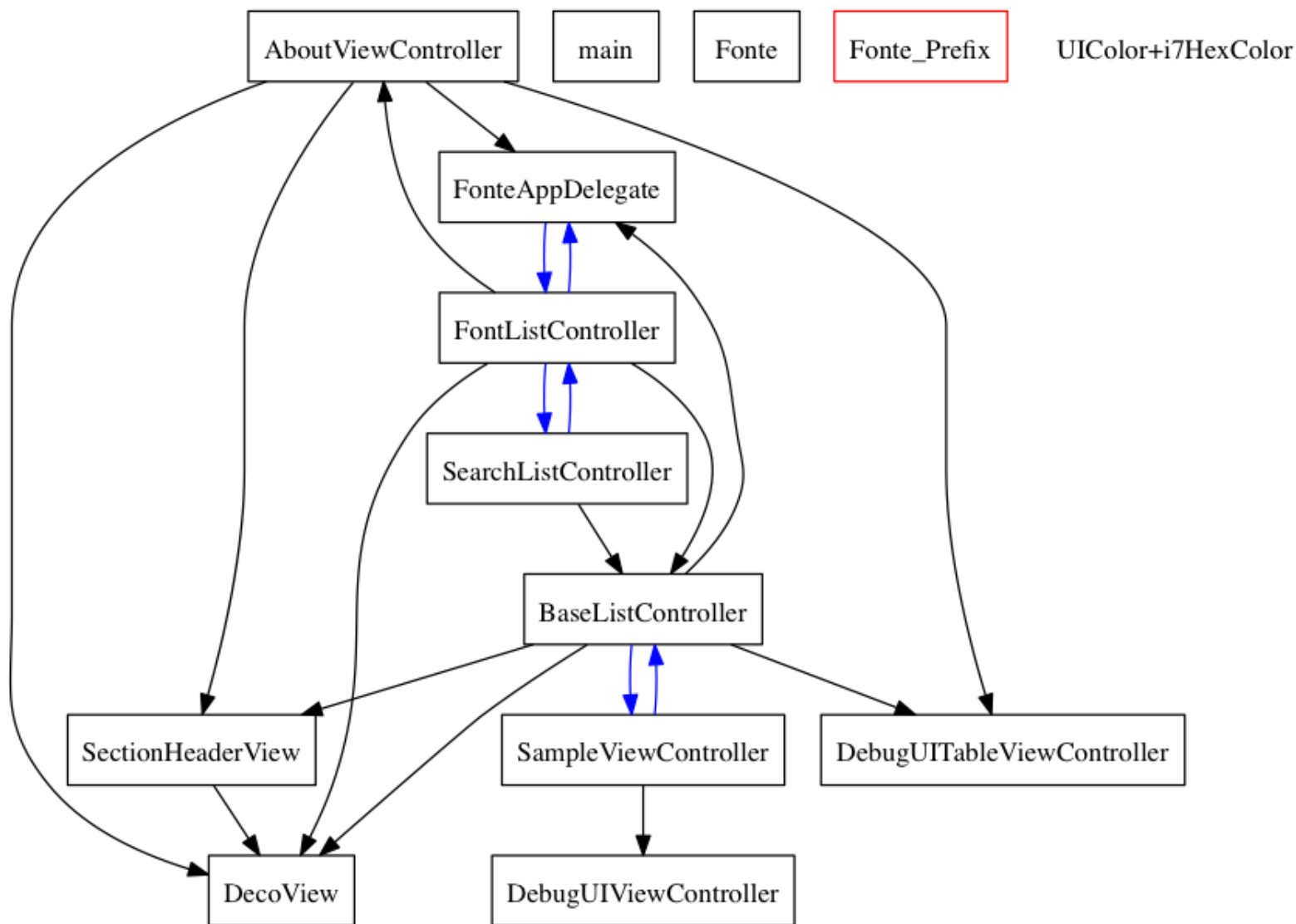
```

```

└─ js
   └─ modules
      JS banners.js
      JS carousel.js
      JS gallery.js
      JS lightbox.js
      JS navigation.js
      JS scroll.js
      JS slideshow.js
      JS tabs.js

```





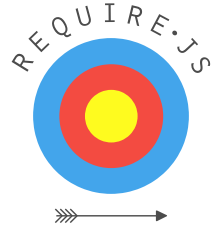
# Organización del código JavaScript: módulos

- **Module loaders:** ellos gestionan las dependencias y cargan los módulos (RequireJS, SystemJS)



**SystemJS**

# Organización del código JavaScript: módulos



## SystemJS

### Ventajas

- Legible e inteligible
- Fácil de mantener
- Reutilizable
- Cargamos sólo lo que necesitamos
- Gestión automática de dependencias
- Encapsulación

### Inconvenientes

- Difícil encontrar código no usado (menos difícil que antes)
- Muchas peticiones HTTP

# Organización del código JavaScript: módulos

- **Module bundlers:** además de lo anterior, generan un solo código encadenado y minificado (browserify, webpack)



**webpack**



# Organización del código JavaScript: módulos



## Ventajas

---

- Legible e inteligible
- Fácil de mantener
- Reutilizable
- Cargamos sólo lo que necesitamos
- Gestión automática de dependencias
- Encapsulación
- Una o muy pocas conexiones HTTP
- Eliminación de código no usado (*tree shaking*)

# Organización del código JavaScript: módulos

- ¿Puedo escribir mis módulos como yo quiera? ¿hay un estándar?
- AMD: Asynchronous Module Definition
- CommonJS
- UMD: Universal Module Definition
- ES6 Modules

```
import { method1 } from './moduleA.js';  
  
method1("hello");  
  
export let method2 = function() {  
    console.log("Method 2");  
}
```

# Organización del código JavaScript: módulos

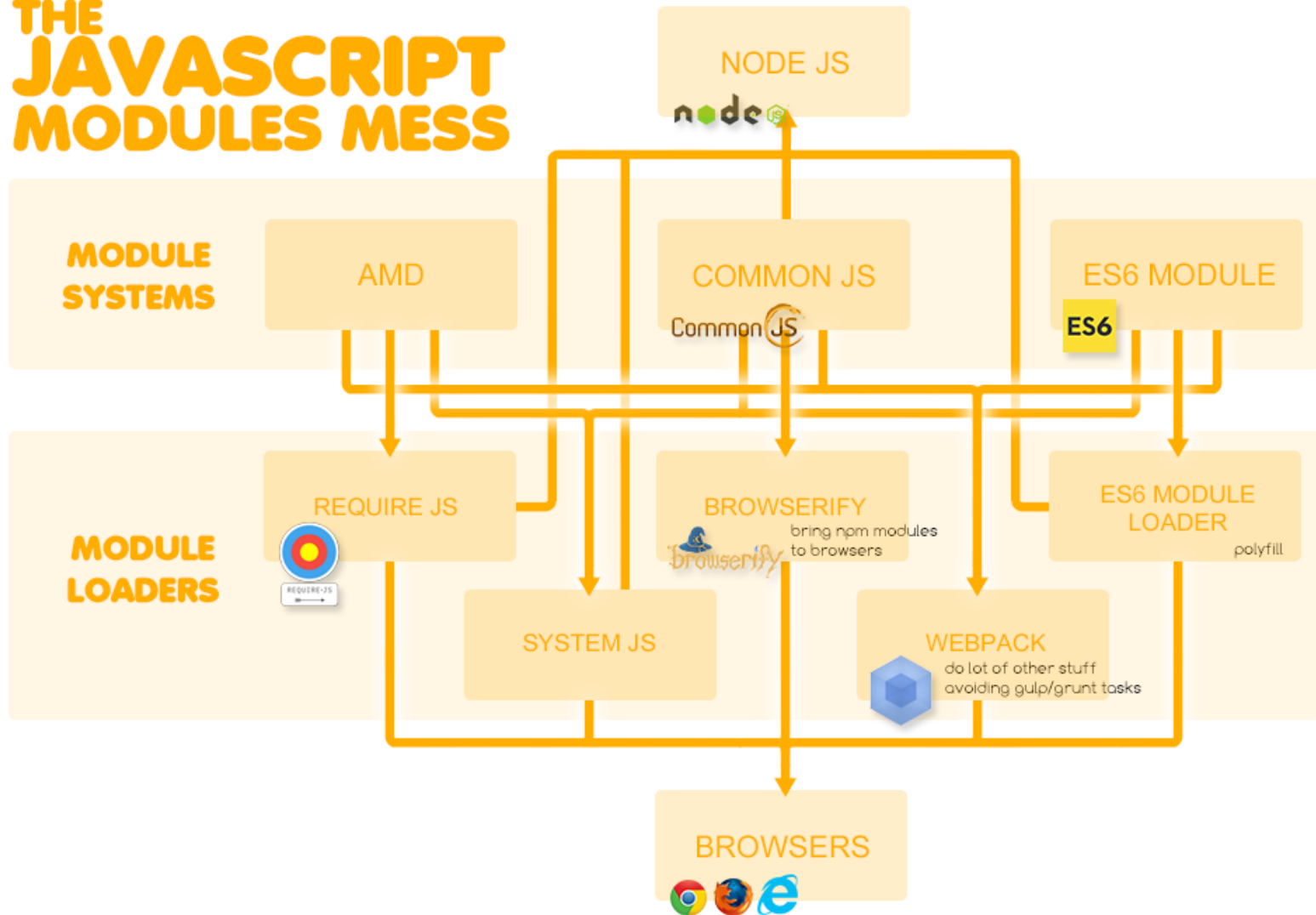
- ¿AMD, CommonJS, UMD, ES6?
- Compatibilidad de los módulos ES6 en navegadores
- ¡Webpack!
- TypeScript usa la sintaxis ES6



**webpack**



# THE JAVASCRIPT MODULES MESS



# ES6

- let y const

# ES6

- let y const

```
let a = 3;

let a = 10; // Error
var a = 12; // Error

const b = 10;

b = 3; // Error

const obj = {
  x: 10,
  y: 12
}

obj.x = 15; // OK

obj = { // Error
  x: 15,
  y: 12
}
```

# ES6

- `let` `const`
- Template literals

# ES6

- let y const
- Template literals

```
let nombre = "Antonio";

let cuadrado = function(x) {
  return x * x;
}

let n = Math.floor(Math.random() * 10);

let saludo1 = "Hola, " + nombre + ". El cuadrado de " + n + " es " + cuadrado(n) + ".";
let saludo2 = `Hola, ${nombre}. El cuadrado de ${n} es ${cuadrado(n)}.`;
```



# ES6

- `let` `const`
- Template literals
- `for ... of`

# ES6

```
let nombres = ["Patricia", "Zacarías", "Miguel", "Maite"]

for (let i in nombres) {
  console.log(nombres[i]);
}

for (let nombre of nombres) {
  console.log(nombre);
}

let obj = {
  x: 3,
  y: 4
}

for (let i in obj) {
  console.log(obj[i]);
}

let nombre = "Antonio Jesús";

for (let c of nombre) {
  console.log(c);
}
```

# ES6

- let y const
- Template literals
- for ... of
- Funciones
  - Parámetros por defecto

# ES6

```
function potencia(x, y = 2) {  
    return Math.pow(x, y);  
}  
  
console.log(`10 elevado a 8 es ${potencia(10, 8)}`)  
console.log(`El cuadrado de 5 es ${potencia(5)}`);
```

# ES6

- let y const
- Template literals
- for ... of
- Funciones
  - Parámetros por defecto
  - Función arrow:  
(parámetros) => expresión\_devuelta;

# ES6

```
const potencia = function (x, y = 2) {  
    return Math.pow(x, y);  
}  
  
const potencia = (x, y = 2) => Math.pow(x, y);  
  
setTimeout(() => console.log("pausa"), 2000);
```

# ES6

- Operador spread
  - Parámetros en funciones
  - Enviar varios parámetros a partir de un array
  - push y unshift
  - Intercalar un array dentro de otro
  - Copiar un array en otro
  - Copiar un objeto en otro

# ES6

```
// function(a, b, c)
let nums = [1, 3, 6];
function sumar(a, b, c) {
  console.log(a + b + c);
}
sumar(...nums);

// function(n parámetros)
let a = 3;
let b = 7;
let c = 8;

function sumar(...nums) {
  let suma = 0;
  for (n of nums) {
    suma += n;
  }
  console.log("La suma es " + suma);
}
sumar(a, b, c);
```



# ES6

- Clases
  - Propiedades y métodos

# ES6

```
class A {  
    constructor(z) {  
        this.x = 3;  
        this.y = 10;  
        this.z = z;  
    }  
  
    suma() {  
        return this.x + this.y + this.z;  
    }  
}  
  
let a = new A(20);  
  
console.log(a.suma());
```

# ES6

- Clases
  - Propiedades y métodos
  - Getters y setters

# ES6

```
class A {  
  
    constructor(z) {  
        this.x = 3;  
        this.y = 10;  
        this.z = z;  
    }  
  
    suma() {  
        return this.x + this.y + this.z;  
    }  
  
    set zeta(z) {  
        this.z = z * 2;  
    }  
  
    get zeta() {  
        return this.z / 2;  
    }  
}  
  
let a = new A(20);  
  
a.zeta = 15;  
  
console.log(a.zeta);
```

# ES6

- Clases
  - Propiedades y métodos
  - Getters y setters
  - Métodos estáticos

# ES6

```
class A {  
    constructor(z) {  
        this.x = 3;  
        this.y = 10;  
        this.z = z;  
    }  
  
    static getPI() {  
        return 3.14159;  
    }  
  
    suma() {  
        return this.x + this.y + this.z;  
    }  
  
    set zeta(z) {  
        this.z = z * 2;  
    }  
  
    get zeta() {  
        return this.z / 2;  
    }  
}  
  
let a = new A(20);  
  
a.zeta = 15;  
  
console.log(a.zeta);  
  
console.log(A.getPI());
```

# ES6

- Clases
  - Propiedades y métodos
  - Getters y setters
  - Métodos estáticos
  - Herencia con extends y super()

# ES6

```
class A {  
  
    constructor(z) {  
        this.x = 3;  
        this.y = 10;  
        this.z = z;  
    }  
  
    static getPI() {  
        return 3.14159;  
    }  
  
    suma() {  
        return this.x + this.y + this.z;  
    }  
  
    set zeta(z) {  
        this.z = z * 2;  
    }  
  
    get zeta() {  
        return this.z / 2;  
    }  
}  
  
class B extends A {  
    constructor() {  
        super(100);  
        this.x = 20;  
    }  
  
    suma() {  
        return this.x + this.z;  
    }  
}
```



# ES6

- Módulos

- import

```
import { literal } from 'ruta_modulo';  
import literal from 'ruta_modulo';  
import * as literal from 'ruta_modulo';  
import 'ruta_modulo';
```

- export

```
export let a = 3;  
export let class Clase {  
    ...  
}  
export default {  
    key: value  
}
```

# Programación funcional con arrays

- Métodos:
  - map

# Programación funcional con arrays

```
let nombres = ["juan", "luisa", "amparo", "arturo"];  
nombres = nombres.map(nombre => nombre.toUpperCase());  
console.log(nombres);
```

# Programación funcional con arrays

- Métodos:
  - map
  - filter

# Programación funcional con arrays

```
let personas = [  
  {  
    nombre: "juan",  
    edad: 15  
  },  
  {  
    nombre: "luisa",  
    edad: 35  
  },  
  {  
    nombre: "amparo",  
    edad: 17  
  },  
  {  
    nombre: "arturo",  
    edad: 32  
  }  
];  
  
let mayoresEdad = personas.filter(persona => persona.edad >= 18);  
  
console.log(mayoresEdad);
```

# Programación funcional con arrays

- Métodos:
  - map
  - filter
  - reduce

# Programación funcional con arrays

```
let nums = [2, 4, 10, 15, 12];

let suma = nums.reduce((x, y) => x + y);

let objs = [
  {
    x: 3,
    y: 2
  },
  {
    x: 8,
    y: 10
  },
  {
    x: 10,
    y: 15
  }
]

let sumaX = objs.reduce((x, o2) => x + o2.x, 0);           // Método 1
let sumaX = objs.map(o => o.x).reduce((x, y) => x + y);    // Método 2
```

# Programación funcional con arrays

- Métodos:
  - map
  - filter
  - reduce
  - find
- Encadenamiento



# Programación funcional con arrays

```
let notas = [  
  {  
    nombre: "juan",  
    nota: 6  
  },  
  {  
    nombre: "luisa",  
    nota: 8  
  },  
  {  
    nombre: "amparo",  
    nota: 4  
  },  
  {  
    nombre: "arturo",  
    nota: 3  
  }  
];  
  
let notasAprobados = notas.filter(n => n.nota >= 5).map(n => n.nota)  
console.log(notasAprobados);
```

# TypeScript



# TypeScript

- Superconjunto de JavaScript
- Transpila a ES5 (o a otra versión)
- Tipado
- Errores en tiempo de compilación
- tsc
- tsconfig.json

# TypeScript

```
{  
  "compilerOptions": {  
    "target": "es5",  
    "module": "es2015",  
    "moduleResolution": "node",  
    "sourceMap": true,  
    "outDir": "./public/js/",  
  }  
}
```

tsconfig.json

# Tipos

- Tipos básicos:
  - number
  - string
  - boolean
  - Array
  - any
  - void

# Tipos

```
let peso: number;
peso = 89.5;

let saludo: string;
saludo = 'Vais a petarlo con TypeScript';

let esVerano: boolean;
esVerano = false;

let nums: Array<number>;
nums = [10, 55, -3, 4.14];

let nombres: string[];
nombres = ['Juan', 'Paqui', 'Lorenzo', 'Alicia'];

let cosas: any[];
cosas = [10, 'Teruel', -5, true, [0, -10, 15], false];

function imprimeSaludo(s: string): void {
    console.log(s);
}
imprimeSaludo('Buenas tardes');
```

# Tipos

- Tipos básicos:
  - number
  - string
  - boolean
  - Array
  - any
  - void
- Enum

# Tipos

```
enum FormasPago {
    TPV,
    PayPal,
    transferencia
}
let pago: FormasPago;

pago = FormasPago.PayPal;
procesarPago(pago);

function procesarPago(formaPago: FormasPago): void {
    switch (formaPago) {
        case FormasPago.TPV:
            // ...
            break;
        case FormasPago.PayPal:
            // ...
            break;
        case FormasPago.transferencia:
            // ...
            break;
    }
}
```



# Tipos

- Tipos básicos:
  - number
  - string
  - boolean
  - Array
  - any
  - void
- Enum
- Union types

# Tipos

```
let numeros: Array<number | string>;
numeros = ['3', 6, '15.8', 0];

function procesar(a: string | number): void {
  if (typeof a === 'string') {
    console.log(a.toUpperCase());
  } else {
    console.log(a.toFixed(2));
  }
}
```

# Tipos

- Tipos básicos:
  - number
  - string
  - boolean
  - Array
  - any
  - void
- Enum
- Union types
- Genéricos

# Tipos

```
function verDoble<T>(elem: T): T[] {  
  let elemDoble: T[] = [elem, elem];  
  return elemDoble;  
}
```

# Tipos

- Tipos básicos:
  - number
  - string
  - boolean
  - Array
  - any
  - void
- Enum
- Union types
- Genéricos
- Type assertion

# Tipos

```
const inputText = <HTMLInputElement>document.getElementById("nombr  
inputText.select();
```

# Funciones

- Sin flexibilidad en el número de parámetros

# Funciones

```
function sumar(a: number, b: number): number
  return a + b;
}

sumar();           // Error
sumar(3);          // Error
sumar(10, 2);      // OK
sumar(4, -3, 10, 8) // Error
```



# Funciones

- Sin flexibilidad en el número de parámetros
- Parámetros opcionales

# Funciones

```
function sumar(a: number, b: number, c?: number): number {
  if (c) {
    return a + b + c;
  } else {
    return a + b;
  }
}

sumar(10, 2);
sumar(10, 2, 15);
```

# Funciones

- Sin flexibilidad en el número de parámetros
- Parámetros opcionales
- Sobrecarga

# Funciones

```
function nChars(a: number): string;
function nChars(a: string): number;
function nChars(a: string | number): number | string {
  if (typeof a === 'number') {
    return '¡Es un número!';
  } else if (typeof a === 'string') {
    return a.length;
  }
}
```

# Funciones

- Sin flexibilidad en el número de parámetros
- Parámetros opcionales
- Sobrecarga
- Function types

# Funciones

```
function transformaNumero(x: number, callback: (n: number) => void) {  
    callback(x);  
}  
  
let a = 10;  
  
transformaNumero(a, m => console.log(m * 2));
```

# Módulos

- Sintaxis ES6
- Se omite la extensión .ts
- moduleResolution = node
- Importar de paquetes npm: nombre del paquete
- Importar de nuestros módulos: rutas relativas

# Clases

- Propiedades fuera del constructor



# Classes

```
class Factura {  
  numero: string;  
  base: number;  
  tipoIva: number;  
  
  constructor(numero: string, base: number, tipoIva: number = 21)  
  {  
    this.numero = numero;  
    this.base = base;  
    this.tipoIva = tipoIva;  
  }  
}
```

# Clases

- Propiedades fuera del constructor
- Visibilidad de los miembros
- Getters y setters
- Modificador readonly
- Propiedades estáticas

# Clases

```
class Factura {
  private static caracteresSerie = 2;
  public num: string;
  public serie: string;
  public base: number;
  private readonly intTipoIva: number;

  constructor(base: number, tipoIva: number = 21) {
    this.base = base;
    this.intTipoIva = tipoIva;
  }

  get numero(): string {
    return this.serie + this.num;
  }

  set numero(n: string) {
    this.serie = n.slice(0, Factura.caracteresSerie - 1);
    this.num = n.slice(Factura.caracteresSerie);
  }
}

let f = new Factura(100);

f.numero = 'AB600';
console.log(f.numero);
```

# Clases

- Propiedades fuera del constructor
- Visibilidad de los miembros
- Getters y setters
- Modificador readonly
- Propiedades estáticas
- Métodos abstractos

# Clases

```
abstract class Vehiculo {  
    public manual: boolean;  
  
    constructor(public ruedas: number, public motor: Motor)  
        this.manual = this.motor === Motor.ninguno;  
    }  
  
    public abstract arrancar(): void;  
}  
  
class Bici extends Vehiculo {  
    public arrancar(): void {  
        console.log('Me pongo de pie y pedaleo');  
    }  
}
```

# Clases

- Propiedades fuera del constructor
- Visibilidad de los miembros
- Getters y setters
- Modificador readonly
- Propiedades estáticas
- Métodos abstractos
- Interfaces

# Clases

```
interface Arrancable {
    arrancar(): void;
    apagar(): void;
}

abstract class Vehiculo {

    public manual: boolean;

    constructor(public ruedas: number, public motor: Motor) {
        this.manual = this.motor === Motor.ninguno;
    }

}

class Bici extends Vehiculo implements Arrancable {
    public arrancar(): void {
        console.log('Me pongo de pie y pedaleo');
    }
    public apagar(): void {
        console.log('Me bajo de la bici');
    }
}
```

# Clases

```
interface Cliente {  
  id: number;  
  login: string;  
  nombre: string;  
  tipo: TiposCliente,  
  fechaAlta: Date;  
}  
  
function getClientes(): Cliente[] {  
  let clientes: Cliente[] = conectaBD('clientes');  
  return clientes;  
}
```



# Decoradores

- @
- Asignar metadatos
- Muy utilizados en frameworks como Angular
- Tipos:
  - de clase
  - de propiedad
  - de método
  - de parámetro

# Decoradores

```
function Electrico(datos: { autonomia: number, bateriaExtraible: boolean }) {  
  return function (clase: Function): void {  
    clase.prototype.describir = function (): void {  
      console.log(`Soy un vehículo eléctrico, con una autonomía de ${datos.autonomia} h  
        y mi batería ${datos.bateriaExtraible ? ' ' : 'no '}es extraíble.`);  
    }  
  }  
}  
  
@Electrico({  
  autonomia: 24,  
  bateriaExtraible: true  
})  
class Coche {  
  
}  
  
@Electrico({  
  autonomia: 6,  
  bateriaExtraible: false  
})  
class Bici {  
  
}  
  
let coche = new Coche();  
coche.describir();  
let bici = new Bici();  
bici.describir();
```

# Decoradores

```
function maxRuedas(target: Object, propertyKey: string) {
  let valor = this[propertyKey];

  function get() { return valor; }
  function set(nuevoValor) {
    if (nuevoValor > 8) { throw new Error('Máximo 8 puertas.')}
    valor = nuevoValor;
  }

  Object.defineProperty(target, propertyKey, {get, set});
}

class Coche {
  @maxRuedas
  public nRuedas: number;
  constructor() {}
}

let c = new Coche();
c.nRuedas = 10; // Error
```

# Links

- [Generador de webpack.config.js](#)
- [Playground para TypeScript](#)
- [Configuración del compilador TypeScript](#)
- [Documentación sobre todas las API de JavaScript](#)
- [JSON Server API](#)
- [Tablas de compatibilidad en navegadores](#)
- [Starter para TypeScript](#)
- [Starter para TypeScript + Webpack](#)

mario@mariogl.com  
@marioglweb