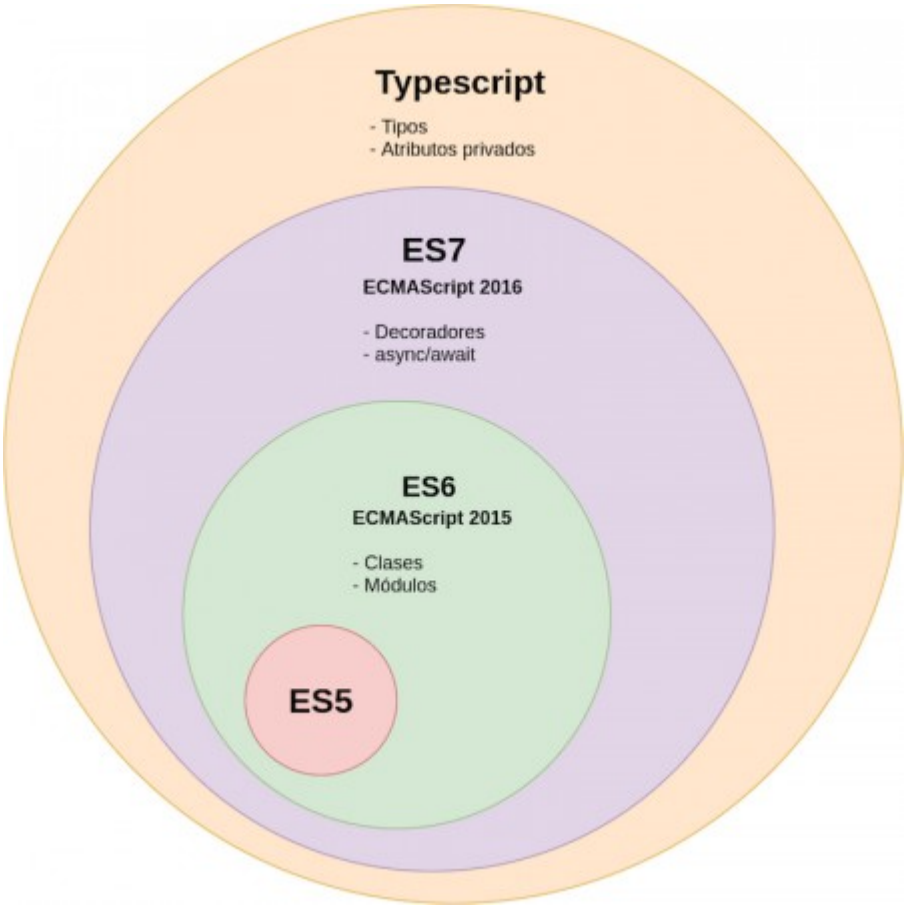
LABORATORIO – TYPESCRIPT

LABORATORIO – TYPESCRIPT



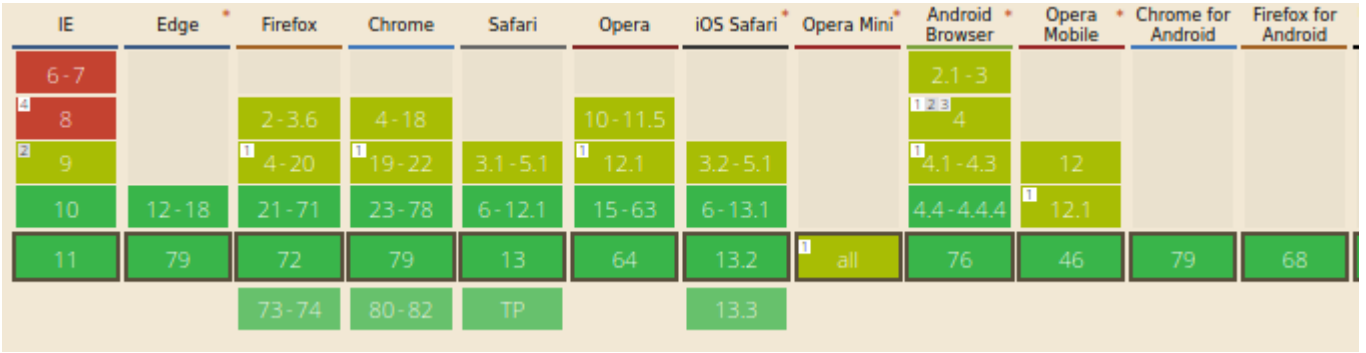
Typescript es un superset de javascript

LABORATORIO – TYPESCRIPT

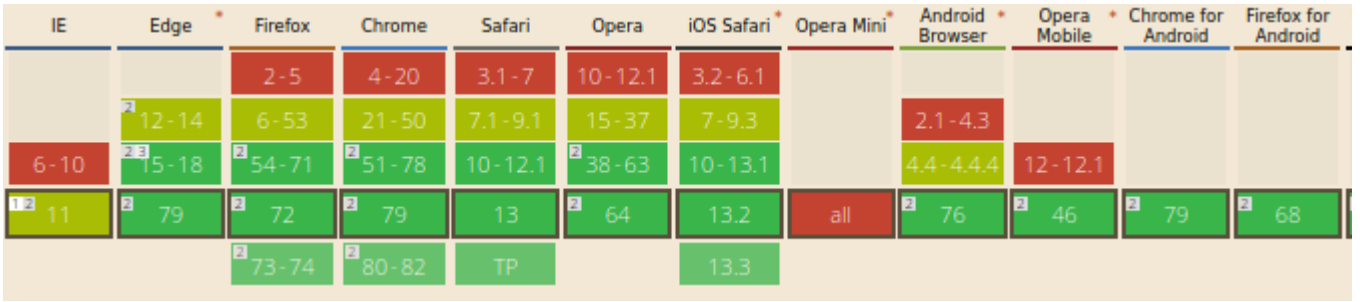
- **JavaScript:**

  - En 1997 se crea el comité TC39, en la ECMA, para estandarizar JavaScript.
  - Los estándares Javascript se rigen por los llamados ECMAscript (ISO/IEC 16262).
  - Actualmente vamos por la versión ES7 (ECMAscript 2017) LTS y ES8 borrador.
  - Más ampliamente soportado: ES5. Y ES6 prácticamente soportado por todos.

Compatibilidad ES5



Compatibilidad ES6



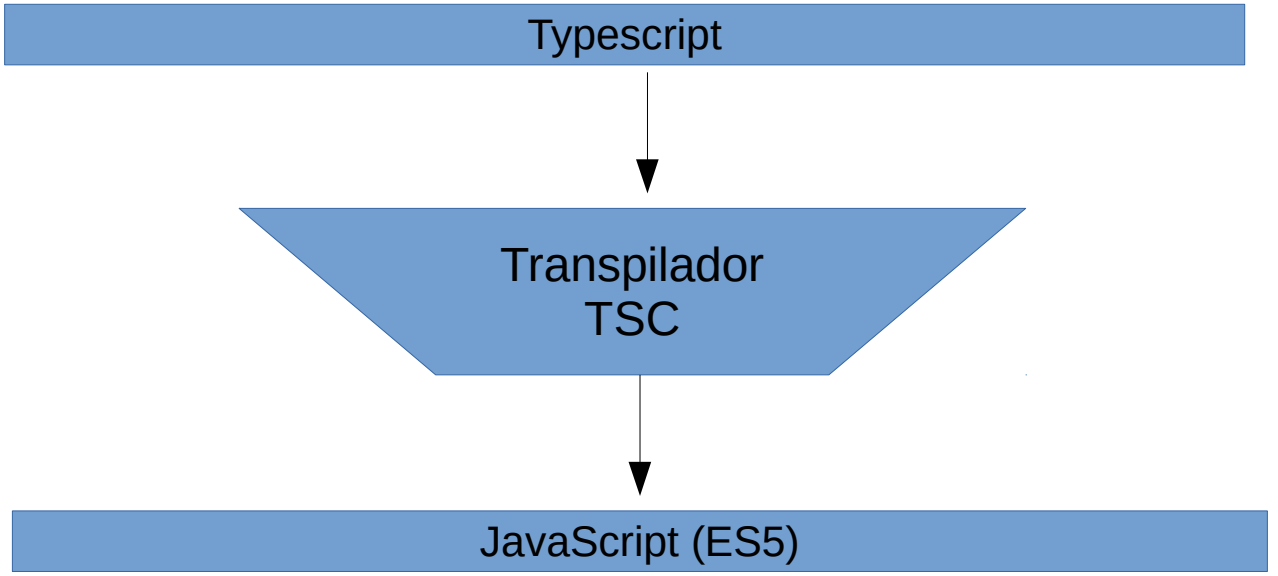
LABORATORIO – TYPESCRIPT

- **TypeScript:**
  - Creado por: Anders Hejlsberg (Turbo Pascal, Delphi, C# y arquitecto principal de .NET)
  - Publicado por Microsoft en 2012.
  - Finalmente adoptado por GOOGLE, principio de 2015, frente a sus propios lenguajes.
  - Codificamos moderno -> ejecutamos compatible (transpilado a JS).
  - Todo código JS es TS válido!

LABORATORIO – TYPESCRIPT

- **Tres aportaciones principales:**
  - Tipado de variables y funciones (parámetros y tipos de retorno):
    - (JS) - let importe=45;
    - (TS) - let importe:number = 123456;
    - Detección temprana de errores en fase de compilación!
  - Decoradores.
    - @input()
  - Interfaces: útil en POO.

LABORATORIO – TYPESCRIPT



LABORATORIO – TYPESCRIPT

Compilación:

Tsc <fichero.ts> → <fichero.js>

LABORATORIO – TYPESCRIPT

Declaración de variables:

```
// var (ámbito función):

function ejemplo(){
        var i=22;
        for(var i=0;i<5;i++){
        ...
        }
        Console.log(i);  --> 5
}

function ejemplo(){
        console.log(i);  //→ OK. undefined
        var i:number=22;
}
```

```
// let (ámbito bloque):

Function ejemplo(){
        let i=22;
        for(let i=0;i<5;i++){
        ...
        }
        Console.log(i);  --> 22
}

function ejemplo(){
        console.log(i);  //→ ERROR
        let i:number=22;
}
```

Recomendable usar LET siempre que sea posible!

LABORATORIO – TYPESCRIPT

Declaración de variables:

```
var x=45;
x=”pepe”; // OK!


var x:number=45;
x=”pepe”; // ERROR!!


var x; // Undefined!
```

LABORATORIO – TYPESCRIPT

TIPOS BÁSICOS: (llamados tipos primitivos. Siempre en minúsculas!)

- **NUMBER:** (Internamente siempre se usa coma flotante, 64 bits con doble precisión)
  - let x:number=3;
  - let x:number=3.141592;
  - let x:number=6.022e-23;
  - Number.MAX_VALUE, Number.MIN_VALUE
- **STRING**:
  - let titulo:string=”El club de la lucha”;
  - let titulo:string='El club de la lucha';

LABORATORIO – TYPESCRIPT

FUNCIONES PARA TIPOS BÁSICOS:

- **NUMBER:**
  - Math.abs()
  - Math.log()
  - Math.max(), Math.min()
  - Math.random()
  - Math.sqrt()
  - x.toPrecision(length)
  - x.toString()
- BOOLEAN:
  - let f:boolean=true;
  - let f:boolean=false;
  - let f:boolean=8>12;

LABORATORIO – TYPESCRIPT

FUNCIONES PARA TIPOS BÁSICOS:

- **STRING**:
  - .indexOf() *// primer carácter tiene índice 0.*
  - .lastIndexOf()
  - .concat(s1,s2,s3) *->s1+s2+s3*
  - .math()
  - .slice(begin,end) *// extrae una subcadena.*
  - .split(delim)

LABORATORIO – TYPESCRIPT

ENUMERACIONES:

- enum Direccion {NORTE, SUR, ESTE, OESTE};
- enum Direccion {NORTE=0.0, SUR=180.0, ESTE=90.0, OESTE=270.0};

- let dir:Direccion=Direccion.NORTE;

- If (dir==Direccion.ESTE) {}

LABORATORIO – TYPESCRIPT

ARRAYs:

- **Declarar:**
- let a:Array<tipo>;
- let a:string[];

- **Inicializar:**
- let colores:string[]=['rojo','verde','azul'];

- **Recorrer (cualquier tipo iterable!!!):**
- for(let i=0;i<colores.length;i++) {document.write(colores[i]);}

- for(let color of colores) {document.write(color);}

- **Funciones de iteracción: (MUY ÚTILES!!!!)**
  var task_names = tasks.map((task) => task.name);
  var difficult_tasks = tasks.filter((task) => task.duration >= 120);

LABORATORIO – TYPESCRIPT

object:

- **object vs Object vs {}**
- **object : cualquier tipo no primitivo.**
  let colores:object=['rojo','verde','azul'];

LABORATORIO – TYPESCRIPT

Object:

- **Object ó {}: objetos de javascript, ± un diccionario vitaminado!**

- let house:Object={habitaciones:5, direccion:'jose abascal, 32'};
  document.write(house[‘habitaciones’]);

  house[‘precio’]=135000; → OK. Resuelto en tiempo de ejecución!

- house.hasOwnProperty(‘precio’) → boolean;
-
- Console.log(house.precio) → ERROR en tiempo de compilación!

LABORATORIO – TYPESCRIPT

Iterar sobre las propiedades del Object:

- let house:Object={habitaciones:5, direccion:'jose abascal, 32'};

  Object.keys(house); → [‘habitaciones’, ‘direccion’]

  Object.values(house) → [5,'jose abascal, 32']

  Object.entries(house) → [['habitaciones', 5], ['direccion', 'josé abascal’]]

```
for (let [key, value] of Object.entries(house)) {
  console.log(key + ':' + value);
}
```

LABORATORIO – TYPESCRIPT

Object – funciones ‘miembro’

```
var person = {
   firstName:"Tom",
   lastName:"Hanks",
   sayHello:function() {  }
}


person[‘sayHelloNew’] = function() {
   console.log("hello ");
}

person[‘sayHello’]()
```

LABORATORIO – TYPESCRIPT

- **Type Checking:**

  If (typeof x == “string”){ //(trabaja solo con primitivas!!)
  …..
  }

- If (v == undefined){
  …
  }

- alert(rabbit instanceof Rabbit); // true (instanceof trabaja con clases!!)

LABORATORIO – TYPESCRIPT

- **Comparaciones: (==) y (===) escricta, (!=) y (!==) estricta**
- == -> primero convierte a tipos similares, si es necesario.

Comparación	Resultado
'hola' == 'hola'	true
'hola' === 'hola'	true
5==”5”	true
5===”5”	false
var a = new String('hola'); a==’hola’ a===’hola’	true false

LABORATORIO – TYPESCRIPT

- **Comparaciones: (==) y (===) escricta, (!=) y (!==) estricta**

  En comparación de objetos se compara la REFERENCIA no el CONTENIDO!

  ```
  var a = new String('foo');
  var b = new String('foo');
  ```

Comparación	Resultado

var a = new String('hola');
var b = new String('hola');
 	
a==b
	false

a===b	false

LABORATORIO – TYPESCRIPT

TUPLAs y tipos combinados:

- **Declarar e inicializar una TUPLA:**
  let t:[boolean,number]=[true,6];
  document.write(t[1]);

- **Combinados:**
  let a:Array<Object>=[{habitaciones:5, direccion:'jose abascal, 32'},
  {habitaciones:2, direccion:'fuencarral, 7'}];

LABORATORIO – TYPESCRIPT

Funciones:

- Function log2(x:number):number{
  return Math.log(x)/Math.log(2);
  }

- **Valor por defecto:**
  Function logb(x:number,base:number=2.0):number{
  return Math.log(x)/Math.log(base);
  }

- **Parámetros opcionales:**
  Function logb(x:number,base?:number):number{
  if (base == undefined) base=2.0;
  return Math.log(x)/Math.log(base);
  }

LABORATORIO – TYPESCRIPT

Funciones anónimas y Funciones Flecha:

- **Funciones anónimas:**
- var f= function(x:number):number{
  return Math.log(x)/Math.log(2);
  }

- Ideal para callbacks:
  Var miobj={
  …
  var func_procesa={}
  }

  Miobj.func_procesa={...}

- **Funciones Flecha:**
- var f=(x)=>{return Math.sqrt(x)/2.0;}

LABORATORIO – TYPESCRIPT

Declaración de Clases:

```
class Persona{
   edad:number;
    nombre:string;
    altura:number;

    comer() {…};
    nacer() {…};
    hablar() {…};

}
```

```
Var alumno=new Persona();
alumno.edad=20;
alumno.comer();
```

console.log(alumno instanceof Persona) → true
console.log(alumno instanceof Object) → true

CLASE = Object
+
Prototipo
+
Constructor
+
declaración de métodos y atributos

LABORATORIO – TYPESCRIPT

Declaración de Clases - Herencia:

```
class Vehiculo{
      numRuedas:number;
      edad:number;
};

class Coche extends Vehiculo{
   modelo:string;
   matricula:string;
}


Var coche1=new Coche();
coche1.numRuedas=4;
coche1.edad=7;
coche1.matricula='1234FFR';
```

LABORATORIO – TYPESCRIPT

Declaración de Clases - Constructores:

```
class Vehiculo{
        numRuedas:number;
        edad:number;
};

class Coche extends Vehiculo{
    modelo:string;
    matricula:string;

    constructor(modelo:string,matricula:string)
    {
        this.modelo=modelo;
        this.matricula=matricula;
    }
}
```

- **Var coche1=new Coche('Audi','1234FFR');**

LABORATORIO – TYPESCRIPT

Declaración de Clases – Public, Private y Static:

```
class Circulo{
    private static pi=3.141592;
    public radio:number;

    getArea():number
    {
       return radio*radio*Circle.pi;
    }
};
```

LABORATORIO – TYPESCRIPT

Declaración de Clases – Interfaces *vs* Herencia:

Coche hereda de Vehiculo

vs

Persona implementa interface IPagos

Compute implementa interface Ipagos

¿CLASE es un CLASEBASE?
Sí: Herencia
No: Interface

```
Interface Ipagos{
        enviarPago():number{}
        getSaldo():number{}
}


Class Persona implements Ipagos, IMovilidad {
        enviarPago(){}
        getSaldo():number{}

        frenar(){};
        acelerar(){};
}
```

LABORATORIO – TYPESCRIPT

Object o JSON → Class

```
class Contacto{
    nombre: string;
    telefono:string;
    email: string;
}
```

```
let contacto:Contacto={
   nombre:'roberto',
   telefono:'1234',
   email:'roberto@noseque.ess'
};
```

LABORATORIO – TYPESCRIPT

Object o JSON → Clases compuestas

```
export class Dire{
    public calle:string;
    public numero:string;
};

export class Contacto{
    public nombre:string;
    public apellidos:string;
    public direccion:Dire;
}
```

```
let contacto:Contacto=
   {
   nombre:'Pepe',
   apellidos:'Castejon',
   direccion:{calle:'José Zorrilla',numero:'32'}
   };
```

LABORATORIO – TYPESCRIPT

Object o JSON → Class (con métodos)

```
export class Contacto{
    public nombre:string;
    public apellidos:string;
    public isHabitual() {};   //Impide transformación de JSON a clase!
}
```

LABORATORIO – TYPESCRIPT

Copiar atributos: **SOLO ATRIBUTOS! (Prototipo, métodos y constructor NO).**

```
let a:Contacto;
let b:Contacto;
```

- a=b; // **COPIA REFERENCIA, NO COPIA ATRIBUTOS!**
- Object.assign(a,b);
- c={… b};
- Console.log(b instanceof Contacto) → true
- Console.log(c instanceof Contacto) → false

- constructor (o:Object)
  {if(o !== undefined) Object.assign(this,o); };
- constructor (o:IContacto)
  {if(o !== undefined) Object.assign(this,o); };
- a:Contacto=new Contacto({});

LABORATORIO – TYPESCRIPT

Object o JSON → Class (con métodos)

MÉTODO 1: constructor desde Interface.

```
export class Contacto{
    public nombre:string;
    public apellidos:string;
    public isHabitual() {};

   constructor (o?:IContacto)
      {if(o !== undefined) Object.assign(this,o);  };
}

let c:Contacto=new Contacto({nombre:'ricardo'...});

let lista:Contacto[]= [{},{},{}].map(ic=>new Contacto(ic));
```

LABORATORIO – TYPESCRIPT

Object o JSON → Class (con métodos)

MÉTODO 2: Patrón Clase POJO + Process Class

```
export class Contacto{   //Aquí solo se definen atributos
    public nombre:string;
    public apellidos:string;
}

export class PContacto{  //Aquí se definen los métodos
   public static isHabitual(c:Contacto) {};


let c:Contacto={nombre:'ricardo'...};
PContacto.isHabitual(c)
```

LABORATORIO – TYPESCRIPT

CONSEJOS:

1. Utilizar siempre que sea posible Classes y/o Interfaces.

2. Utilizar el método 2: Patrón de diseño: POJO + Process Class, para TODA clase susceptible de ser intercambiada hacia y desde Api REST o BBDD...

LABORATORIO – TYPESCRIPT

Micropráctica 02 - ANGULAR

Filtrar por: (marca de coche)

Foto	Marca	Modelo	Año	En venta desde	Precio	PVP	Acciones
	RENAULT	scenic	2007	04-2018	5,000€	6,050€	Rebajar Vendido
	SEAT	ibiza	2003	03-2018	1,200€	1,452€	Rebajar Vendido
	RENAULT	megane	2007	03-2018	3,500€	4,235€	Rebajar Vendido
(sin foto!)	TESLA	model 3	2007	03-2018	4,000€	4,840€	Rebajar Vendido