



**Clase 13.** Programación Backend

# ***Node.js como herramienta de desarrollo***





## ***OBJETIVOS DE LA CLASE***

- Comprender el concepto de transpilador.
- Instalación y uso de Babel mediante Node.js.
- Instalación y uso de Typescript en un proyecto Node.js.

# ***CRONOGRAMA DEL CURSO***

Clase 12



**Aplicación chat con  
websocket**

Clase 13



**Node.js como  
herramienta de  
desarrollo**

Clase 14

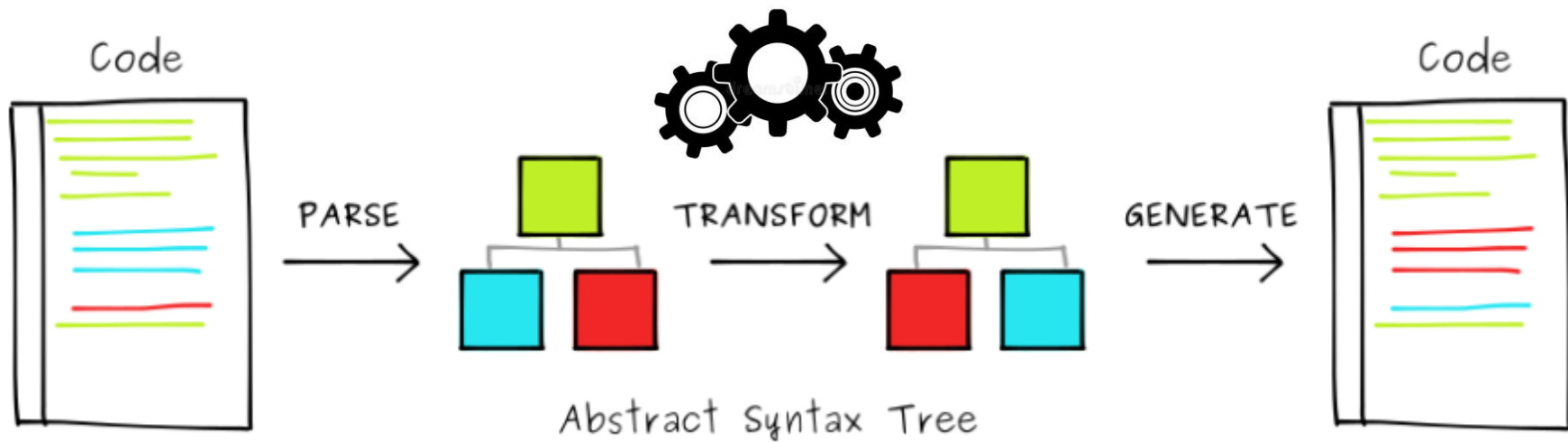


**Webpack: Module  
Bundler**



**1º ENTREGA PROYECTO FINAL**

# *Transpilador*



Es un programa que **traduce código** escrito en **un lenguaje de programación a otro lenguaje**. En este **tipo de traductor** el lenguaje fuente es generalmente un lenguaje de **alto nivel** y el objeto un lenguaje de **bajo nivel**, como assembly o código máquina por ejemplo.

***¿Qué es un  
Compilador?***

- Es un **tipo** especial de **compilador** que **traduce de un lenguaje fuente a otro fuente**, también de un nivel de abstracción parecido.
- Se diferencia de los compiladores tradicionales ya que estos últimos reciben como entrada archivos conteniendo código fuente y generan código máquina del más bajo nivel.
- La **transpilación**, que es la acción que realiza el *transpilador*, es un **caso particular** de la **compilación**.

***¿Qué es un  
transpilador?***

# BABEL



***Ejemplos de  
transpiladores***

***CODER HOUSE***

- Los transpiladores y los compiladores traducen código desde un origen hacia un destino.
- La diferencia radica en la relación entre los lenguajes origen y destino de la traducción.
- El transpilador traduce código entre dos lenguajes que están al mismo nivel de abstracción, mientras que el compilador lo hace entre lenguajes de diferente nivel de abstracción

## ***Diferencias entre transpiladores y compiladores***





***¿Alguna pregunta hasta  
ahora?***

# Babel

## ES6 JAVASCRIPT

```
const str1 = "Hello";  
const str2 = "World";  
console.log(`${str1} ${str2}`);
```

Not compatible  
to all browsers

## ES5 JAVASCRIPT

```
var str1 = "Hello";  
var str2 = "World";  
console.log(str1 + " " + str2);
```

Compatible to  
all browsers

*BABEL*

COMPILER  
TRANSPILER

- Babel es un transpilador que nos permite transformar nuestro código JS de última generación (o con funcionalidades extras) a JS que cualquier navegador o versión de Node.js entienda.
- Babel funciona mediante plugins con los cuales le indicamos cuál es la transformación que vamos a efectuar.
- Con el plugin babel-plugin-transform-es2015-arrow-functions podemos decirle que transforme las arrow functions de ECMAScript 2015 a funciones normales.

***¿Qué es Babel?***

***Vamos al IDE...*** 



***¿Alguna pregunta hasta  
ahora?***

# Babel: Web oficial <https://babeljs.io/>

BABEL

Docs

Setup

Try it out

Videos

Blog

Search

Donate

Team

GitHub

GET BABEL HOLIDAY APPAREL

## Babel is a JavaScript compiler.

Use next generation JavaScript, today.

Babel 7.12 is released! Please read our [blog post](#) for highlights and [changelog](#) for more details!

Put in next-gen JavaScript

```
var name = "Guy Fieri";  
var place = "Flavortown";  
  
`Hello ${name}, ready for ${place}?`;
```

Get browser-compatible JavaScript out

```
var name = "Guy Fieri";  
var place = "Flavortown";  
"Hello " + name + ", ready for " + place + "?";
```

CODER HOUSE

# Babel: Online Transpiler ES6 -> JS5

**BABEL**

DocsSetupTry it outVideosBlogSearchDonateTeamGitHub

☐ typescript

☐ stage-3

☐ stage-2

☐ stage-1

☐ stage-0

ENV PRESET

☒ Enabled

ELECTRON

1,8

☐

NODE

10,13

☐

BUILT-INS

core-js 3.6

Usage

☐

SPEC

☐

LOOSE

☐

BUG FIXES

☐

SHIPPED PROPOSALS

☐

FORCE ALL TRANSFORMS

☒

> PLUGINS

```
1 let mensaje = 'Hola mundo!'
2 console.log(mensaje)
3
4 const sumar = (a,b) => a + b
5 const dobleDe = a => 2*a
6
7 console.log(`La suma es ${sumar(16,9)}`)
8 console.log(`Doble de 80 es ${dobleDe(80)}`)
9
10 class Persona {
11   constructor(nombre, edad) {
12     this.nombre = nombre
13     this.edad = edad
14   }
15
16   getNombre() {
17     return this.nombre
18   }
19
20   getEdad() {
21     return this.edad
22   }
23 }
24
25 const juan = new Persona('Juan',23)
26 const ana = new Persona('Ana',21)
27
28 console.log(juan.getNombre())
```

```
1 "use strict";
2
3 function _classCallCheck(instance, Constructor) { if (!(in
4
5 function _defineProperties(target, props) { for (var i = 0
6
7 function _createClass(Constructor, protoProps, staticProps
8
9 var mensaje = 'Hola mundo!';
10 console.log(mensaje);
11
12 var sumar = function sumar(a, b) {
13   return a + b;
14 };
15
16 var dobleDe = function dobleDe(a) {
17   return 2 * a;
18 };
19
20 console.log("La suma es ".concat(sumar(16, 9)));
21 console.log("Doble de 80 es ".concat(dobleDe(80)));
22
23 var Persona = /*#__PURE__*/function () {
24   function Persona(nombre, edad) {
25     _classCallCheck(this, Persona);
26
27     this.nombre = nombre;
28
```

v7.12.14

CODER HOUSE



# ***COLOR ALEATORIO CON BABEL***

*Tiempo: 10 minutos*



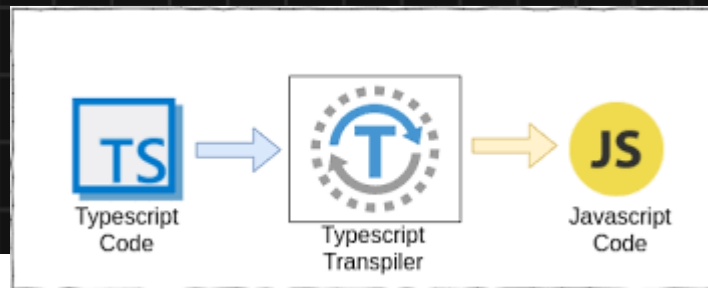
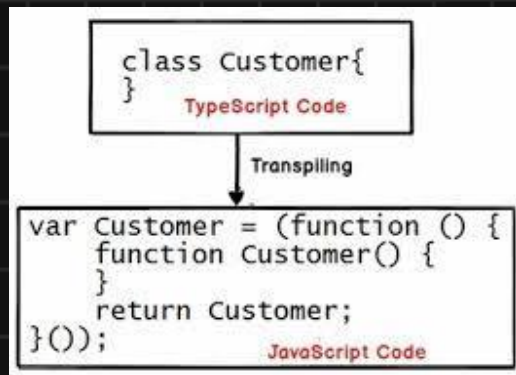


- Realizar un programa que genere un color aleatorio en formato RGB (canal rojo, verde y azul entre 0 y 255) y lo muestre por consola. Este estará implementado en un archivo llamado color.js
- La funcionalidad debe estar implementada dentro de una clase y deberá utilizar sintaxis ES6 (const, let, arrow function y template string).
- Convertir este código ES6 a JS5 con Babel online. Realizar esta conversión en forma automática dentro de un proyecto node.js que utilice Babel CLI



***¿Alguna pregunta hasta  
ahora?***

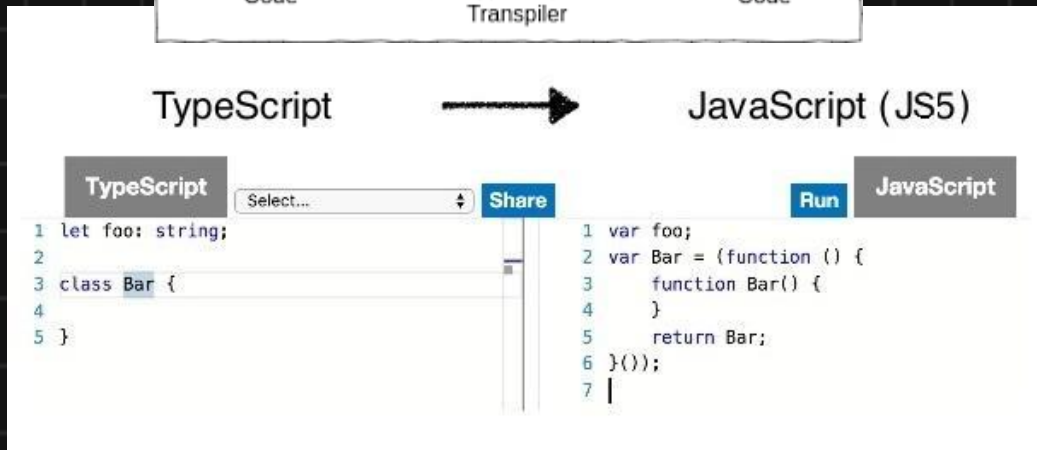
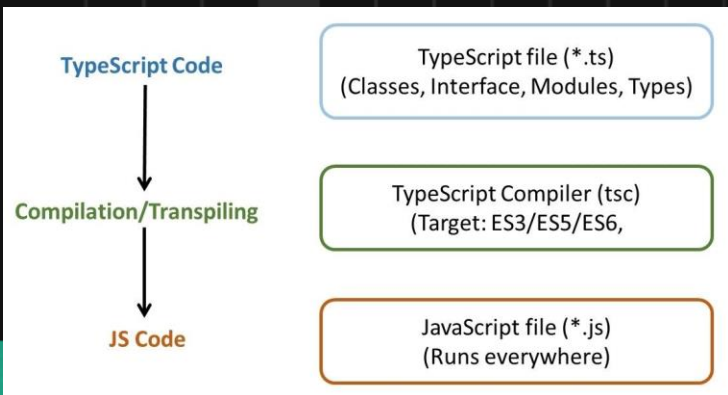
# TSC: Typescript compiler

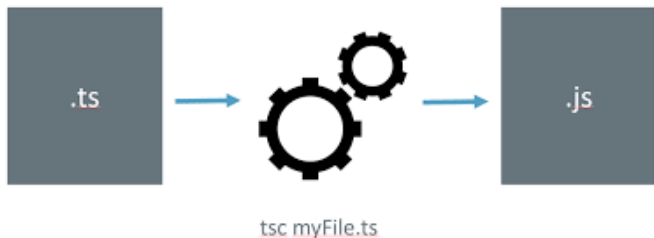


TypeScript



JavaScript (JS5)



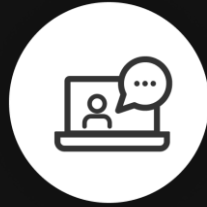


## ***¿Qué es TSC?***



- Los archivos de TypeScript se compilan en JavaScript mediante TSC: el **compilador de TypeScript**
- TSC se puede instalar como paquete TypeScript a través de npm
- Para **transpilar** los archivos **Typescript a Javascript** lo hacemos a través de un proyecto en Node.js configurado como se muestra a continuación

***Vamos al IDE...*** 

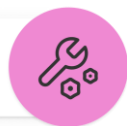


***¿Alguna pregunta hasta  
ahora?***



# ***COLOR ALEATORIO CON TSC***

*Tiempo: 10 minutos*



- Realizar un proyecto TypeScript node.js que genere un **color aleatorio** en formato RGB (canal rojo, verde y azul entre 0 y 255) y lo muestre por consola.
- La funcionalidad debe estar implementada dentro de una clase en un archivo color.ts y deberá utilizar sintaxis Typescript tipada.
- El proyecto deberá convertir este código TS a JS5 en forma automática con TSC CLI



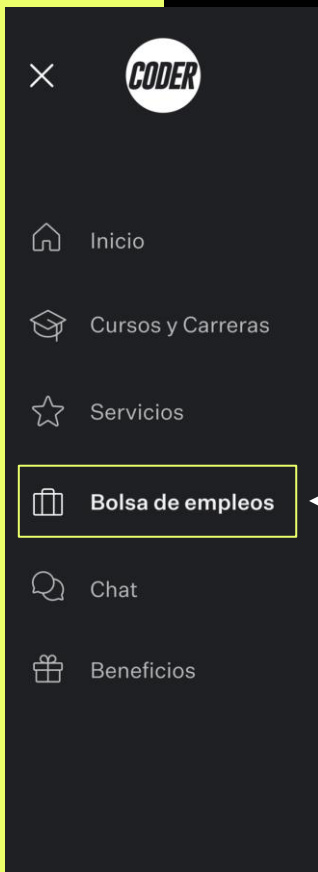


***¿Alguna pregunta hasta  
ahora?***



***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**



Nuevo

# ¡Lanzamos la Bolsa de Empleos!

Un espacio para seguir **potenciando tu carrera** y que tengas más **oportunidades de inserción laboral**.

Podrás encontrar la **Bolsa de Empleos** en el menú izquierdo de la plataforma.

Te invitamos a conocerla y ¡postularte a tu futuro trabajo!

Conócela

# ***Módulos en ES6***

# ***Módulos en ES6: Introducción***



A partir de ES6 de Node.js admite definir archivos y proyectos como *módulos*. A diferencia de los archivos y proyectos comunes en JavaScript (“commonJs”), los módulos permiten ser importados en forma asincrónica en lugar de sincrónica, lo cual libera el hilo principal y mejora la performance de los programas (entre otras ventajas). Cuando se trata de proyectos, este cambio se puede realizar fácilmente desde el archivo *package.json*, agregando el siguiente par clave-valor: `"type": "module"`.

# Módulos en ES6: Sintaxis



Una vez definido el proyecto como módulo, ya no podremos utilizar la función *require* para importar otros archivos, ni *module.exports* para exportar objetos hacia otros archivos. Para esto se utiliza la nueva sintaxis, según las siguientes equivalencias:

```
// MiClase.js
class MiClase { }
module.exports = MiClase
```

```
// libreria.js
function f() { }
module.exports = { f }
```

```
// main.js
const Clase = require('./MiClase.js')
const { f } = require('./libreria.js')
```



```
// MiClase.js
class MiClase { }
export default MiClase
```

```
// libreria.js
function f() { }
export { f }
```

```
// main.js
import Clase from './MiClase.js'
import { f } from './libreria.js'
```

# ***Módulos en ES6: Sintaxis***



En caso de querer realizar una importación condicional, se puede import como función:

```
if (condicion) {  
  const { default: Clase } = await import('./MiClase.js')  
  const { f } = await import('./libreria.js')  
}
```

Notese que al ser asincrónica, devuelve una promesa, y admite el uso de *async/await*.

*Dentro de los módulos es posible escribir await aún estando fuera de una función async (uso a nivel archivo), causando la espera de la resolución de la promesa como es de esperarse. A esta funcionalidad se la conoce como: **Top-level Await**.*



***¿Alguna pregunta hasta  
ahora?***



# *Uso avanzado de TSC*

{ TypeScript }



TypeScript Configuration  
file: tsconfig.json

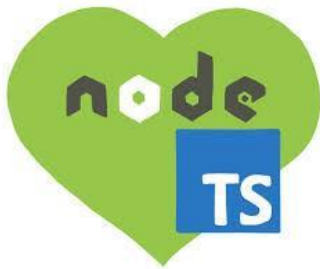
# ***Creando un proyecto Typescript en node.js***



# ***Introducción***



- **Node.js** es un entorno de tiempo de ejecución que hace que sea posible **escribir JavaScript en el lado del servidor**. Esto puede ser difícil a medida que la base de código crece debido a la naturaleza del lenguaje JavaScript: dinámico y con escritura débil.
- Los desarrolladores que llegan a JavaScript desde otros lenguajes a menudo se quejan sobre su falta de escritura estática fuerte, pero aquí es donde entra **TypeScript**, para cerrar esta brecha.



# ***Introducción***



- TypeScript puede **ayudar** a la hora de crear y gestionar **proyectos** JavaScript a **gran escala**. Puede verse como JavaScript con funciones adicionales como escritura estática fuerte, compilación y programación orientada a objetos.
- TypeScript es técnicamente un **superconjunto de JavaScript**, lo que significa que todo el código JavaScript es código TypeScript válido.

***Vamos al IDE...*** 

***¿PREGUNTAS?***



# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Transpiladores JS
- Babel
- TSC



***OPINA Y VALORA ESTA CLASE***



***#DEMOCRATIZANDO LA EDUCACIÓN***