



## 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

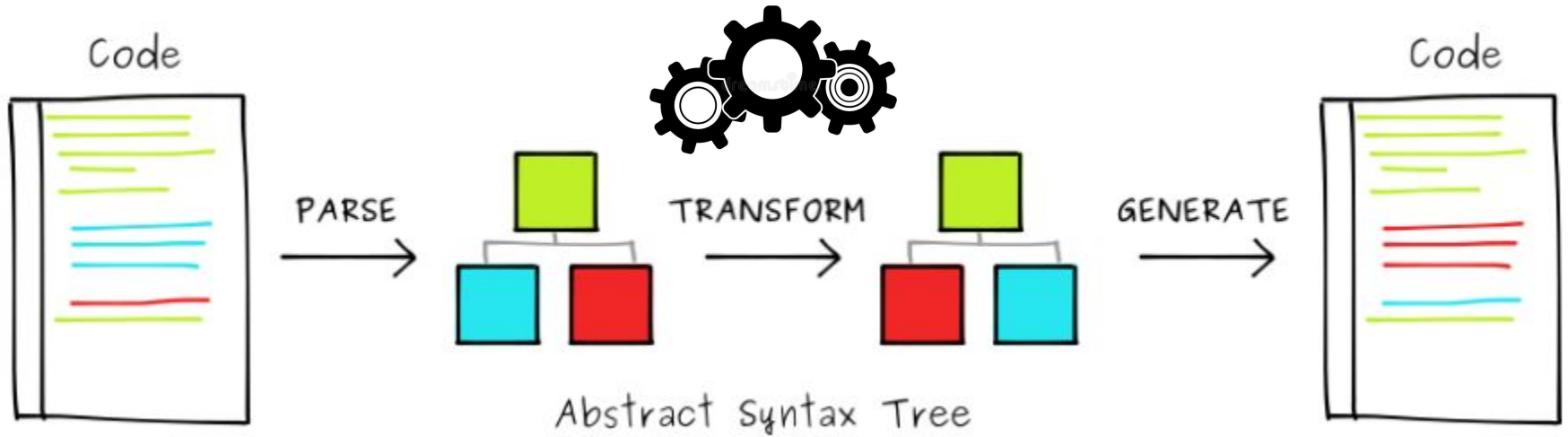


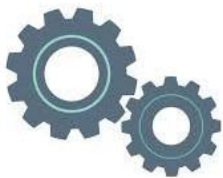
**Webpack: Module  
Bundler**



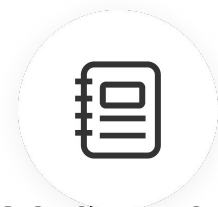
**1º ENTREGA PROYECTO FINAL**

# Transpilador





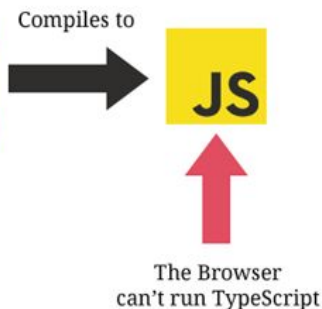
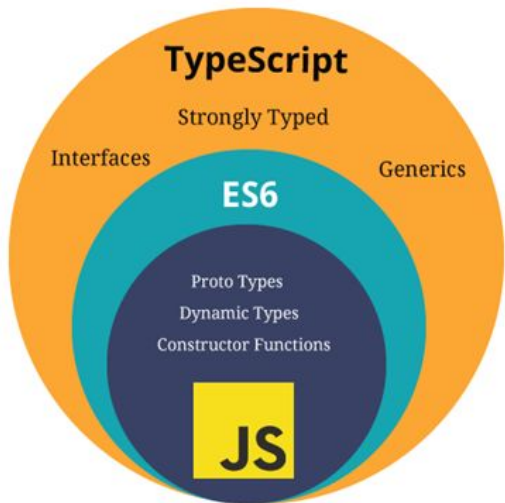
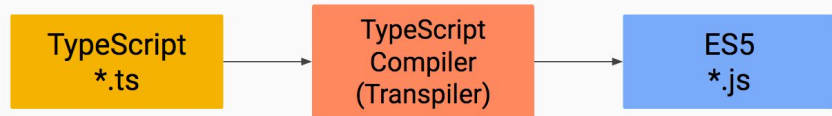
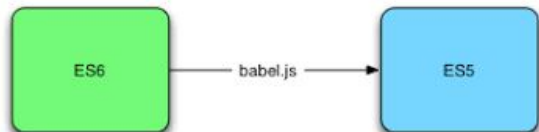
# ¿Qué es un transpilador?



- Un transpilador 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**.



# Ejemplos



Compilación



```
Code:
0:  iconst_2
1:  istore_1
2:  iload_1
3:  sipush 1000
6:  if_icmpge 44
9:  iconst_2
10: istore_2
11: iload_2
12: iload_1
13: if_icmpge 31
16: iload_1
17: iload_2
18: irem      # remainder
19: ifne 25
22: goto 38
```

TypeScript



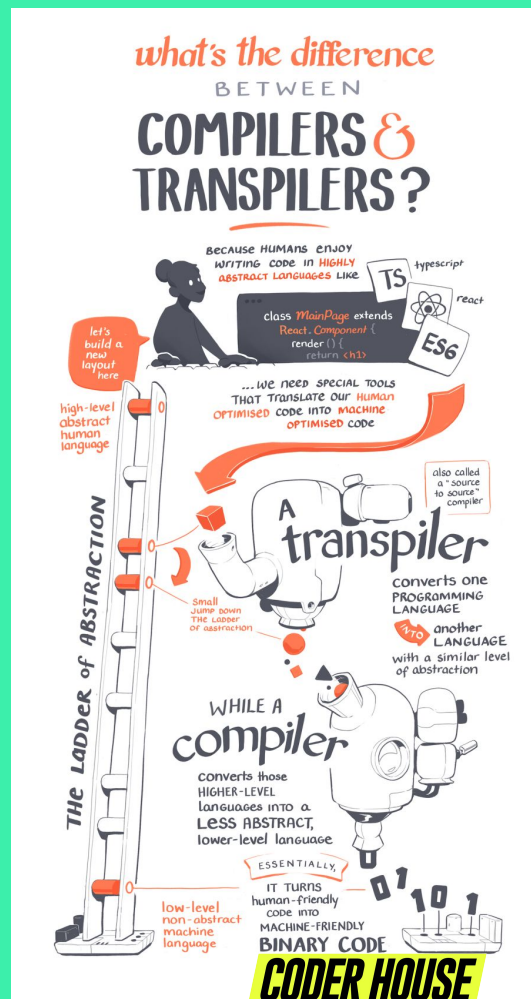
tsc first.ts

JavaScript

**CODER HOUSE**

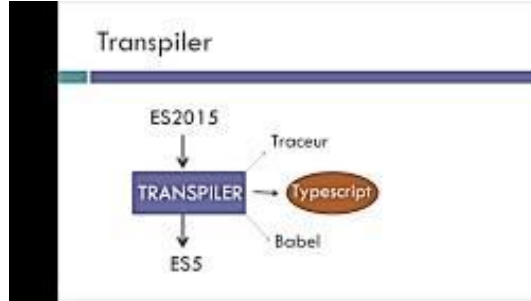
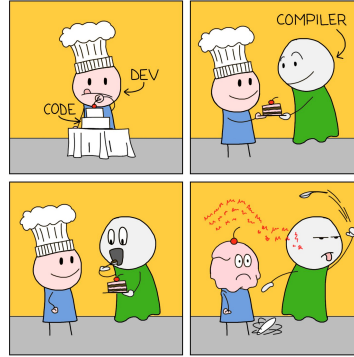
# Diferencias entre transpiladores y compiladores

- 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



# Ejemplos

COMPILER



Compilation

Transpilation

```
C  
  
int doThing(const char* desc) {  
    ...  
}
```

Machine Code

```
MOV AL, 1h  
MOV CL, 2h  
MOV DL, 3h  
MOV EAX, [EBX]  
MOV [ESI+EAX], CL
```

JavaScript Source

```
function doThing(desc) {  
    ...  
}
```

## Compiladores

### Transpiladores

Hi.cpp

```
void main()  
{  
    cout<<"hi";  
}
```

Source file

Compiler

Hi.exe

```
11011001  
01000100  
00010111  
10101011
```

Machine code

ES6  
ES2016  
ES2017

BABEL

ES5  
+  
Polyfills  
[if needed]

**CODER HOUSE**



# 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



# ¿Qué es Babel?



- 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



# ***Babel.js y Node.js***



Existen varias formas de utilizar Babel. Vamos a trabajar con la versión en línea de comandos (CLI) que realiza una compilación directa. Para ello:

1. Creamos un proyecto de Node.js con **npm init -y**
2. Instalamos la librería Babel, el cliente, y el plugin  
**npm install @babel/core @babel/cli @babel/preset-env**
3. El primer módulo es la librería principal, el segundo es el cliente por terminal, y el tercero es el plugin de configuración para que soporte todos los JavaScript de la nueva generación.



# ***Babel.js y Node.js***

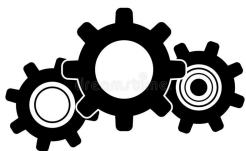


El último paso que nos queda es crear el fichero de configuración de Babel “**.babelrc**” y decirle con que plugin vamos a trabajar

```
{  
  "presets": ["@babel/preset-env"]  
}
```

Luego vamos a crear un archivo **origen.js** con el siguiente código:

```
const lista=[2,3,5,7];  
lista.map(x => x*x).forEach(x => console.log(x));
```



# ***Transpilando de ES6 a JS5***



El código escrito en **origen.js** pertenece a *ES6* ya que usa *const* y las nuevas *arrow functions* y queremos que **Babel** lo convierta a *JS5*.

Para ello, definimos un script en el *package.json*:

**"build": "babel ./origen.js -o ./destino.js -w"**

La opción *-w* nos permite transpilar automáticamente ante los cambios en *origen.js*

- Así obtenemos el archivo **destino.js** que Babel.js nos genera.

```
"use strict";  
  
var lista = [2, 3, 5, 7];  
lista.map(function (x) {  
  return x * x;  
}).forEach(function (x) {  
  return console.log(x);  
});
```

# Proyecto completo

Visual Studio Code interface showing a project named "EJEMPLOBABEL" with files: `node_modules`, `.babelrc`, `destino.js`, `origen.js`, `package-lock.json`, and `package.json`.

The `origen.js` file contains the following code:

```
1 const lista=[2,3,5,7];
2 lista.map(x => x*x)
3 .forEach(x => console.log(x));
```

The `destino.js` file contains the following code:

```
1 "use strict";
2
3 var lista = [2, 3, 5, 7];
4 lista.map(function (x) {
5     return x * x;
6 }).forEach(function (x) {
7     return console.log(x);
8 });
9
```

A diagram illustrates the transformation process: **ES6** (yellow box) is transformed by **BABEL** (handwritten text) into **JS ES5** (yellow box), which is then executed by web browsers (Chrome, Edge, Firefox, Safari icons).

The terminal shows the command to run the project:

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase14\ejemploBabel
> node_modules/.bin/babel .\origen.js -o .\destino.js -w
change origen.js
change origen.js
[]
```

The output of the command is shown in the terminal:

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase14\ejemploBabel
> node .\destino.js
4
9
25
49
PS C:\Cursos\Coderhouse\CursoBackend\Clase14\ejemploBabel
>
```

The status bar at the bottom indicates: **CODER HOUSE**, **Ln 3, Col 1**, **Spaces: 4**, **UTF-8**, **CRLF**, **JavaScript**, **Go Live**, **Prettier**.

# 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**

Docs Setup **Try it out** Videos Blog Search Donate Team GitHub

☐ 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
```

**CODER HOUSE**





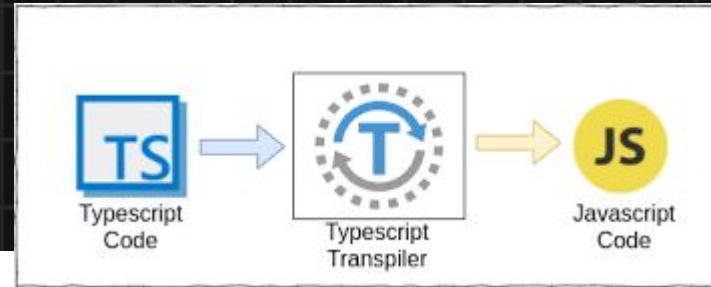
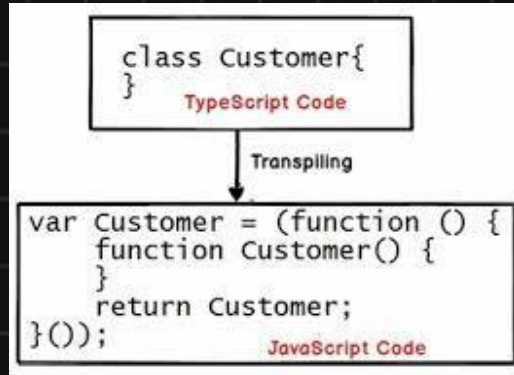
# ***COLOR ALEATORIO CON BABEL***

*Tiempo: 10 minutos*



- Realizar un proyecto ES6 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

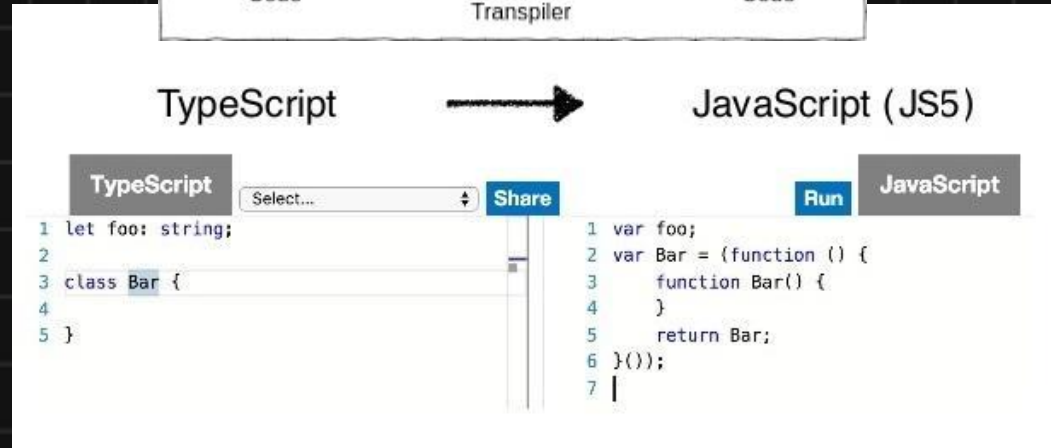
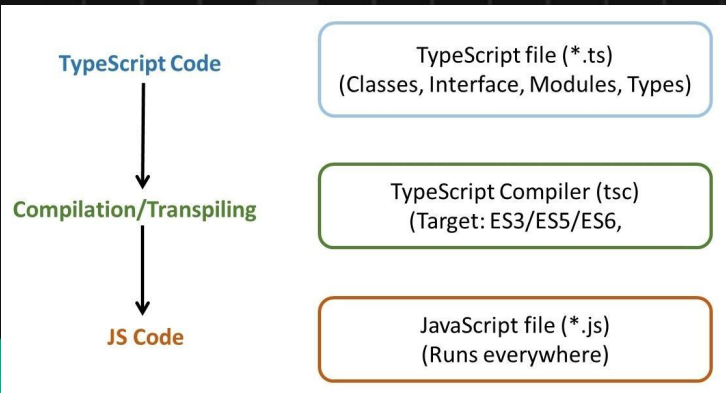
# TSC: Typescript compiler

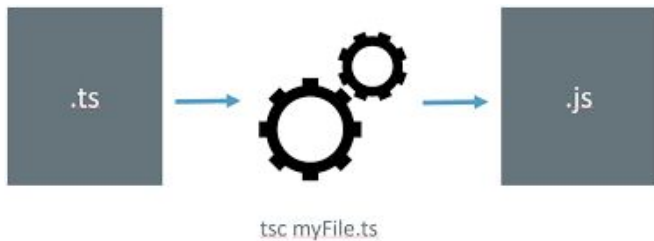


TypeScript



JavaScript (JS5)

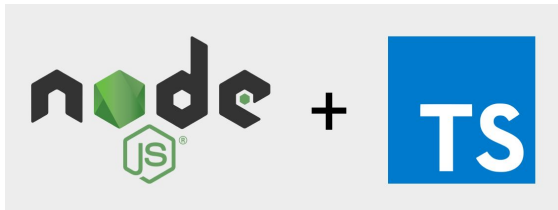




# ***¿Qué es TSC?***



- 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



# ***Typescript y Node.js***



1. Creamos un proyecto de Node.js con **npm init -y**
2. Instalamos el TSC mediante npm: **npm i typescript**
3. Creamos un archivo **index.ts** con contenido en Typescript
4. Transpilamos con el comando: **node\_modules/.bin/tsc ./index.ts -w**
5. Verificamos que en nuestra carpeta de proyecto se encuentre **index.js**

Siguiendo esta serie de pasos logramos convertir un archivo codificado en Typescript en su equivalente Javascript que pueda ser ejecutado en un navegador o en la plataforma Node.js

# Proyecto completo

The image shows a Visual Studio Code editor with two files open: `index.ts` (TypeScript) and `index.js` (JavaScript). The `index.ts` file contains the following code:

```
1  const lista:Array<number>=[2,3,5,7];
2
3  lista.map((x:number) => x*x)
4  .forEach((x:number) => console.log(x));
```

The `index.js` file contains the following code:

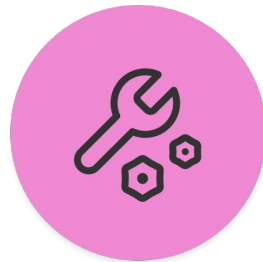
```
1  var lista = [2, 3, 5, 7];
2  lista.map(function (x) { return x * x; })
3  .forEach(function (x) { return console.log(
4
```

A diagram in the center illustrates the compilation process: `TypeScript` is converted to `JavaScript` using the `tsc` command (labeled `tsc first.ts`).

The terminal at the bottom shows the command `node_modules/.bin/tsc .\index.ts -w` being executed, resulting in the output:

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase14\ejemploTSC> node_modules/.bin/tsc .\index.ts -w
[11:52:48] Starting compilation in watch mode...
[11:52:49] Found 0 errors. Watching for file changes
```

The status bar at the bottom indicates the current file is `index.ts` at line 4, column 38, with 4 spaces, UTF-8 encoding, CRLF line endings, and TypeScript language. It also shows the version of Node.js (4.1.2) and the Prettier formatter.



# ***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





***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**

# ***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**.*

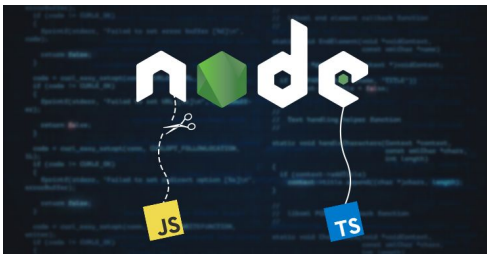
# *Uso avanzado de TSC*

{ TypeScript }



TypeScript Configuration  
file: tsconfig.json

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

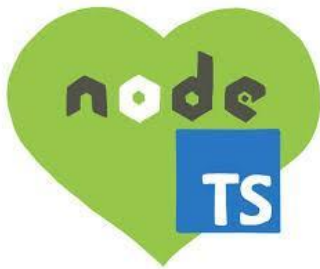


# ***Introducción***



- **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.



# ***Configurar TypeScript***



- TypeScript utiliza un archivo llamado **tsconfig.json** para configurar las opciones del compilador para un proyecto

Para crear el archivo tsconfig.json ejecutamos el siguiente comando: **./node\_modules/.bin/tsc --init**

- Este comando generará un archivo **tsconfig.json** bien redactado.

# Ejemplo tsconfig.json

tsconfig.json - ejemploTSC - Visual Studio Code

EXPLORER

tsconfig.json X

tsconfig.json > ...

```
1  {
2    "compilerOptions": {
3      /* Visit https://aka.ms/tsconfig.json to read more about this file */
4
5      /* Basic Options */
6      // "incremental": true,                    /* Enable incremental compilation */
7      "target": "es5",                          /* Specify ECMAScript target version: 'ES3' (default), 'ES5',
8      "module": "commonjs",                      /* Specify module code generation: 'none', 'commonjs', 'amd',
9
10     "lib": [
11       "DOM",
12       "ES2015",
13       "ES2016",
14       "ES2017",
15       "ES2018",
16       "ES2019",
17       "ES2020",
18       "ESNext"
19     ],
20     /* Specify library files to be included in the compilation. */
21     // "allowJs": true,                       /* Allow javascript files to be compiled. */
22     // "checkJs": true,                       /* Report errors in .js files. */
23     // "jsx": "preserve",                     /* Specify JSX code generation: 'preserve', 'react-native', or
24     // "declaration": true,                   /* Generates corresponding '.d.ts' file. */
25     // "declarationMap": true,                /* Generates a sourcemap for each corresponding '.d.ts' file.
26     // "sourceMap": true,                     /* Generates corresponding '.map' file. */
27     // "outFile": "./",                       /* Concatenate and emit output to single file. */
28     "outDir": "./dist",                      /* Redirect output structure to the directory. */
29     "rootDir": "./src",                      /* Specify the root directory of input files. Use to control
30     // "composite": true,                     /* Enable project compilation */
31     // "tsBuildInfoFile": "./",               /* Specify file to store incremental compilation information
32   },
33 }
```

0 0 0 Live Share

Ln 1, Col 1 Spaces: 2 UTF-8 CRLF JSON with Comments

**CODER HOUSE**

# ***Configuración de tsconfig.json***



*Algunas de las claves más importantes de tsconfig.json*

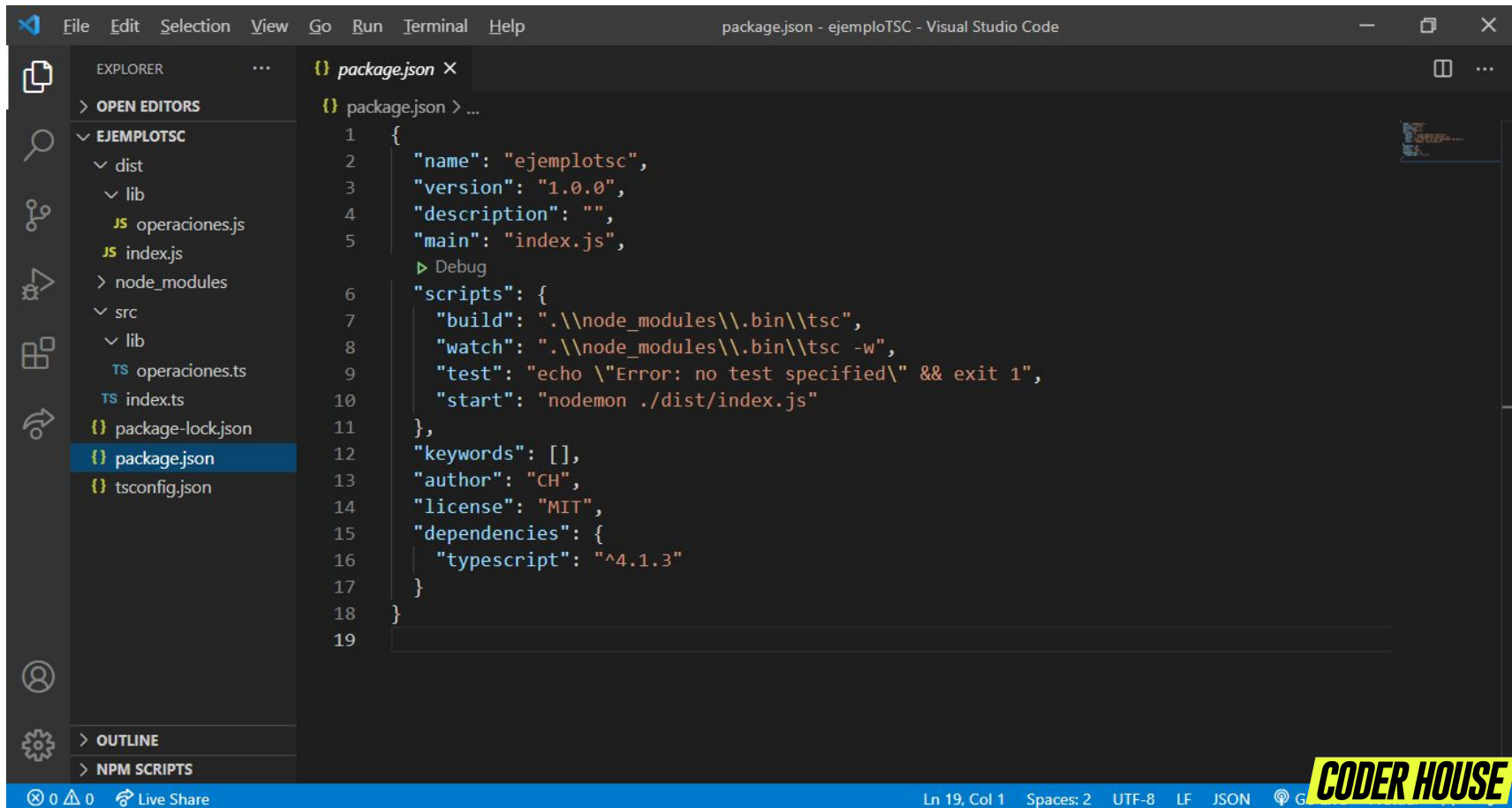
- **module:** Especifica el método de generación de código del módulo.
- **target:** Especifica el nivel de lenguaje de salida.
- **rootDir:** Especifica el directorio raíz de los archivos de entrada. Se usa sólo para controlar la estructura del directorio de salida con *outDir*.
- **outDir:** Esta es la ubicación para los archivos .js tras la *transpilación*.

*En la documentación oficial de TypeScript tenemos más configuraciones:*

<https://www.typescriptlang.org/docs/handbook/compiler-options.html>

# ***Ejemplo de proyecto Typescript con tsconfig.json***

# Proyecto completo: package.json



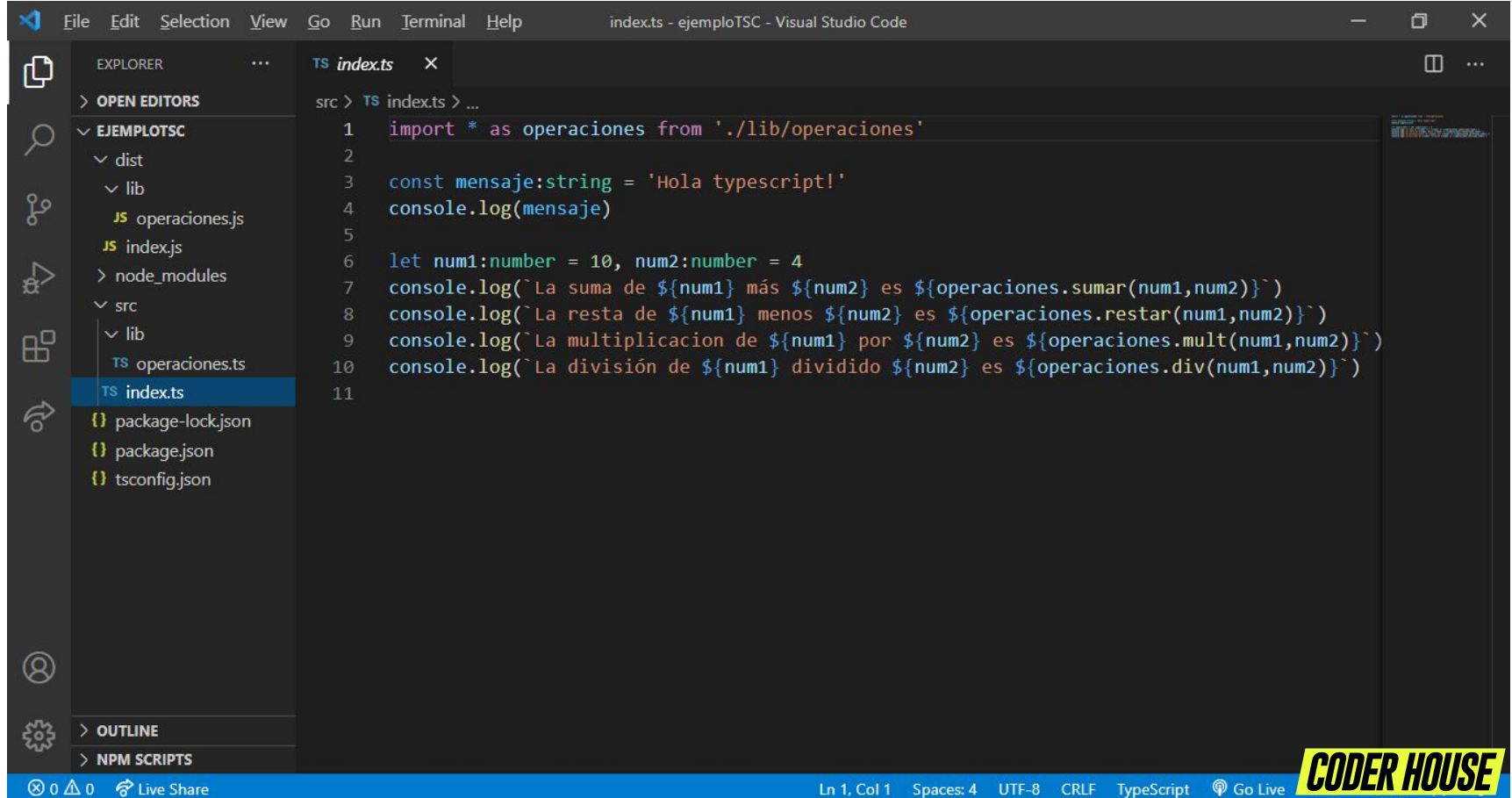
```
1 {
2   "name": "ejemplotsc",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "build": ".\\node_modules\\.bin\\tsc",
8     "watch": ".\\node_modules\\.bin\\tsc -w",
9     "test": "echo \"Error: no test specified\" && exit 1",
10    "start": "nodemon ./dist/index.js"
11  },
12  "keywords": [],
13  "author": "CH",
14  "license": "MIT",
15  "dependencies": {
16    "typescript": "^4.1.3"
17  }
18 }
19
```

0 0 0 Live Share

Ln 19, Col 1 Spaces: 2 UTF-8 LF JSON

**CODER HOUSE**

# Proyecto completo: src (carpeta de entrada)

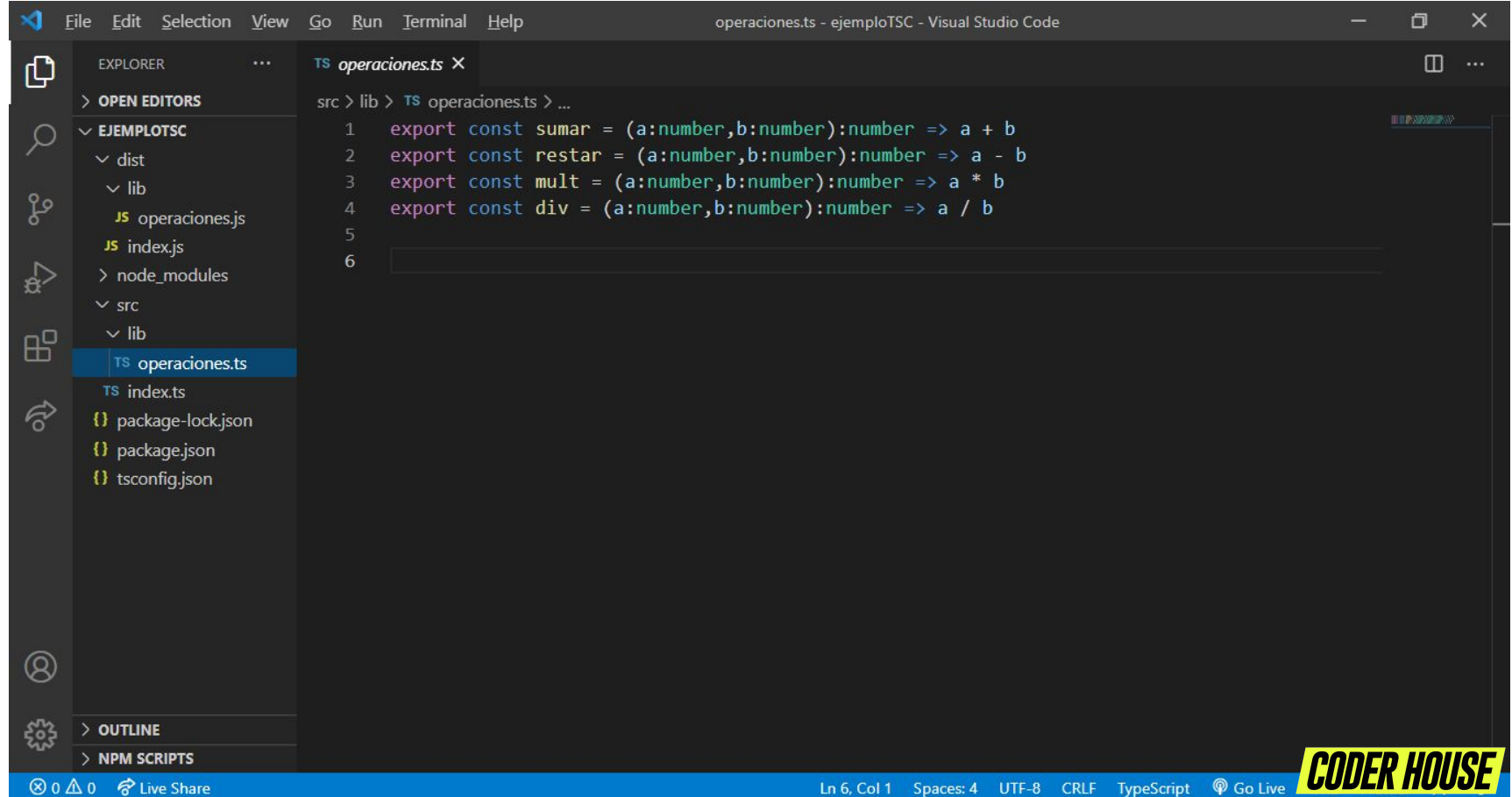


The screenshot shows the Visual Studio Code interface with a project named 'ejemploTSC'. The Explorer sidebar on the left shows the project structure: 'src' > 'lib' > 'operaciones.ts' and 'index.ts'. The 'index.ts' file is open in the editor, showing the following TypeScript code:

```
src > TS index.ts > ...
1  import * as operaciones from './lib/operaciones'
2
3  const mensaje:string = 'Hola typescript!'
4  console.log(mensaje)
5
6  let num1:number = 10, num2:number = 4
7  console.log(`La suma de ${num1} más ${num2} es ${operaciones.sumar(num1,num2)}`)
8  console.log(`La resta de ${num1} menos ${num2} es ${operaciones.restar(num1,num2)}`)
9  console.log(`La multiplicacion de ${num1} por ${num2} es ${operaciones.mult(num1,num2)}`)
10 console.log(`La división de ${num1} dividido ${num2} es ${operaciones.div(num1,num2)}`)
11
```

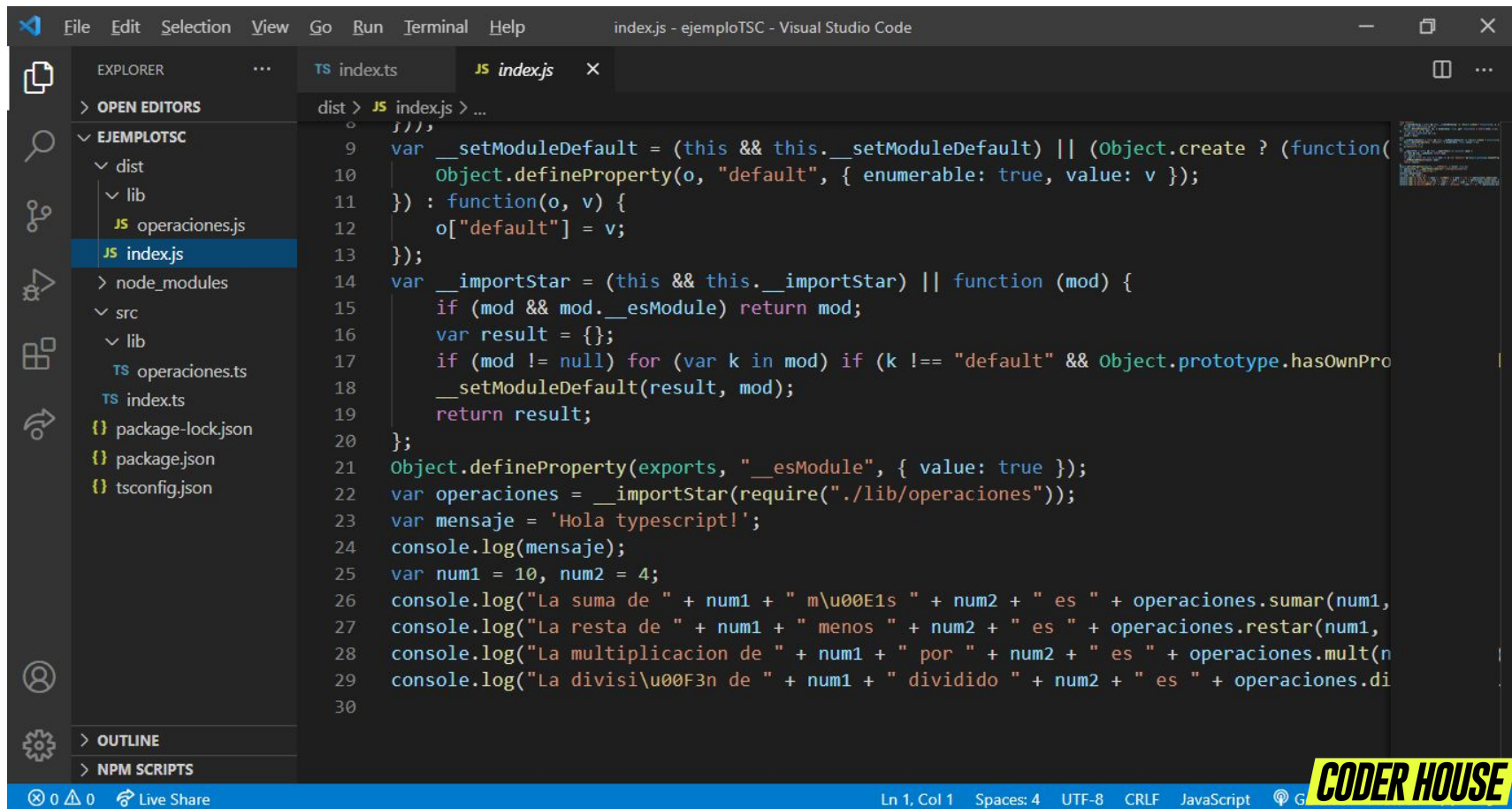
The status bar at the bottom indicates the current position is 'Ln 1, Col 1', with 'Spaces: 4', 'UTF-8', 'CRLF', and 'TypeScript' encoding. A 'Live Share' icon is also present. The 'CODER HOUSE' logo is visible in the bottom right corner.

# Proyecto completo: src (carpeta de entrada)





# Proyecto completo: dist (carpeta de salida)



The screenshot shows the Visual Studio Code interface with a project named 'EJEMPLOTSC'. The Explorer sidebar on the left shows the file structure: 'dist' (containing 'lib' and 'operaciones.js'), 'node\_modules', 'src' (containing 'lib'), and 'ts' (containing 'operaciones.ts', 'index.ts', 'package-lock.json', 'package.json', and 'tsconfig.json'). The 'index.ts' file is selected and open in the editor. The code in 'index.ts' is a TypeScript module that defines a default export, an import function, and performs calculations using the 'operaciones' module. The status bar at the bottom indicates 'Ln 1, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', 'JavaScript', and 'G'. A 'CODER HOUSE' watermark is visible in the bottom right corner.

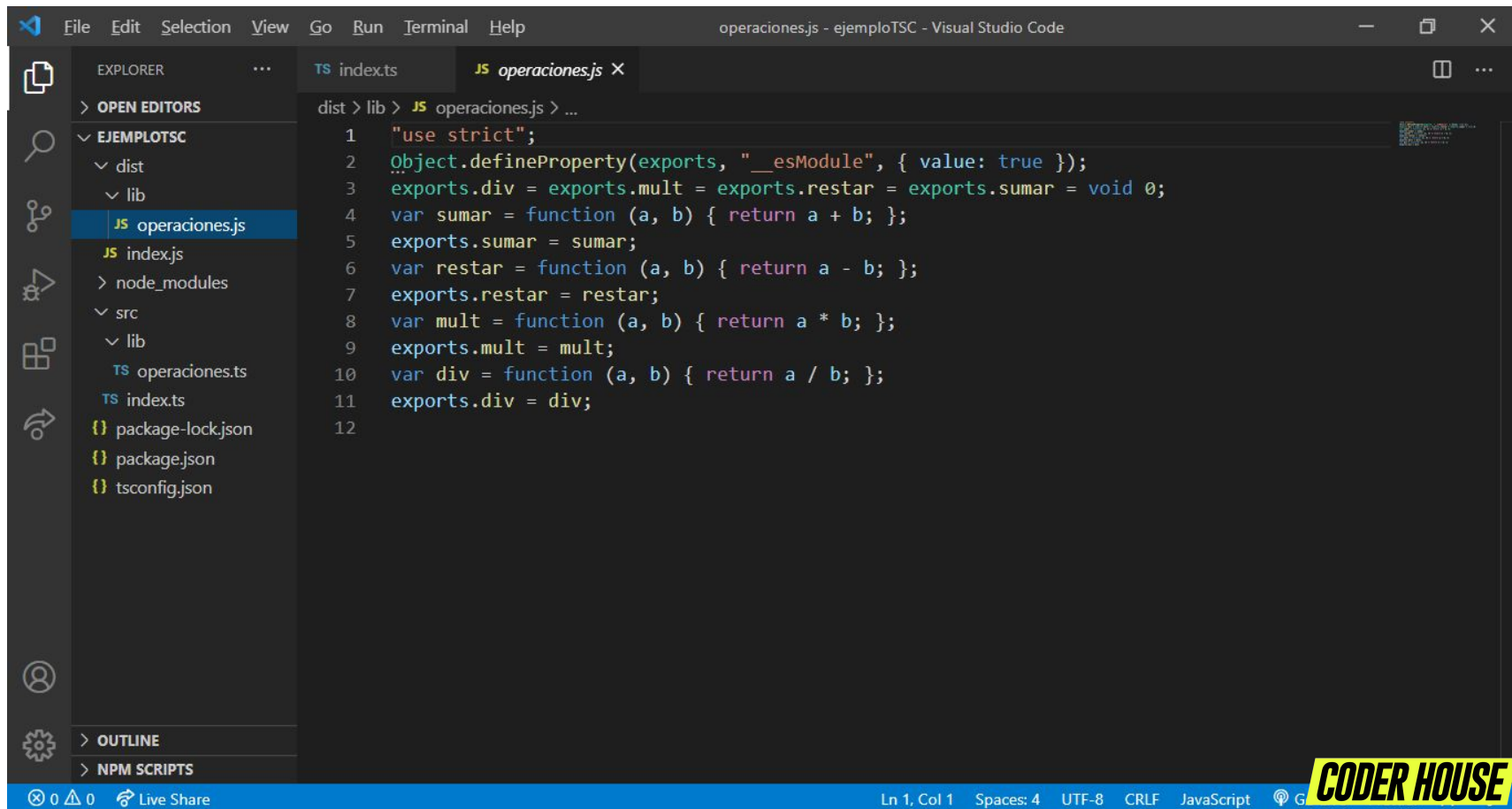
```
index.js - ejemploTSC - Visual Studio Code

EXPLORER
> OPEN EDITORS
EJEMPLOTSC
  dist
    lib
      JS operaciones.js
      JS index.js
  node_modules
  src
    lib
      TS operaciones.ts
      TS index.ts
  package-lock.json
  package.json
  tsconfig.json

> OUTLINE
> NPM SCRIPTS

dist > JS index.js > ...
1 //
2
3
4
5
6
7
8
9 var __setModuleDefault = (this && this.__setModuleDefault) || (Object.create ? (function(o, v) {
10   Object.defineProperty(o, "default", { enumerable: true, value: v });
11 }) : function(o, v) {
12   o["default"] = v;
13 });
14 var __importStar = (this && this.__importStar) || function (mod) {
15   if (mod && mod.__esModule) return mod;
16   var result = {};
17   if (mod != null) for (var k in mod) if (k !== "default" && Object.prototype.hasOwnProperty.call(mod, k)) __setModuleDefault(result, mod);
18   return result;
19 };
20
21 Object.defineProperty(exports, "__esModule", { value: true });
22 var operaciones = __importStar(require("./lib/operaciones"));
23 var mensaje = 'Hola typescript!';
24 console.log(mensaje);
25 var num1 = 10, num2 = 4;
26 console.log("La suma de " + num1 + " m\u00E1s " + num2 + " es " + operaciones.sumar(num1, num2));
27 console.log("La resta de " + num1 + " menos " + num2 + " es " + operaciones.restar(num1, num2));
28 console.log("La multiplicacion de " + num1 + " por " + num2 + " es " + operaciones.mult(num1, num2));
29 console.log("La divisi\u00F3n de " + num1 + " dividido " + num2 + " es " + operaciones.dividir(num1, num2));
30
```

# Proyecto completo: dist (carpeta de salida)



```
1  "use strict";
2  Object.defineProperty(exports, "__esModule", { value: true });
3  exports.div = exports.mult = exports.restar = exports.sumar = void 0;
4  var sumar = function (a, b) { return a + b; };
5  exports.sumar = sumar;
6  var restar = function (a, b) { return a - b; };
7  exports.restar = restar;
8  var mult = function (a, b) { return a * b; };
9  exports.mult = mult;
10 var div = function (a, b) { return a / b; };
11 exports.div = div;
12
```

0 0 0 Live Share Ln 1, Col 1 Spaces: 4 UTF-8 CRLF JavaScript G CODER HOUSE

# ***Operación del proyecto***

Mediante los scripts creados en package.json ponemos en acción los mecanismos de transpilación manual y automática junto con la puesta en marcha del proyecto.

- **"build"**: "tsc" -> transpilación manual.
- **"watch"**: "tsc -w"-> transpilación automática.
- **"start"**: "node ./dist/index.js" -> ejecución de código transpilado.

Con el comando **npm run** se ejecutan los scripts build, watch y start.

# Proyecto completo: resultado final

The image shows a Visual Studio Code editor with two files open: `index.ts` and `index.js`. The `index.ts` file contains TypeScript code for a simple calculator. The `index.js` file is the compiled JavaScript version. Below the editor, the terminal shows the command `node index.js` being executed, and the console output displays the results of the calculations.

```
TS index.ts
src > TS index.ts > num2
1 import * as operaciones from './lib/operaciones'
2
3 const mensaje:string = 'Hola typescript!'
4 console.log(mensaje)
5
6 let num1:number = 10, num2:number = 4
7 console.log(`La suma de ${num1} más ${num2} es $
8 console.log(`La resta de ${num1} menos ${num2} e
9 console.log(`La multiplicacion de ${num1} por $
10 console.log(`La división de ${num1} dividido $r
11

JS index.js
dist > JS index.js > ...
18 __setModuleDefault(result, mod);
19 return result;
20 };
21 Object.defineProperty(exports, "__esModule", { val
22 var operaciones = __importStar(require("./lib/oper
23 var mensaje = 'Hola typescript!';
24 console.log(mensaje);
25 var num1 = 10, num2 = 4;
26 console.log("La suma de " + num1 + " m\u00E1s " +
27 console.log("La resta de " + num1 + " menos " + nu
28 console.log("La multiplicacion de " + num1 + " por
29 console.log("La divisi\u00F3n de " + num1 + " divi
30

TERMINAL
[09:14:14] File change detected. Starting incremental compilation.
..
[09:14:14] Found 0 errors. Watching for file changes.

1: node, node
La multiplicacion de 10 por 6 es 60
La división de 10 dividido 6 es 1.6666666666666667
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node ./dist/index.js`
Hola typescript!
La suma de 10 más 4 es 14
La resta de 10 menos 4 es 6
La multiplicacion de 10 por 4 es 40
La división de 10 dividido 4 es 2.5
[nodemon] clean exit - waiting for changes before restart
```

***¿PREGUNTAS?***



# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Transpiladores JS

- Babel

- TSC



***OPINA Y VALORA ESTA CLASE***

***#DEMOCRATIZANDO LA EDUCACIÓN***

***CODER HOUSE***