



**¡Les damos la
bienvenida!**

¿Comenzamos?

Unidad 01. Programación Backend

Principios Básicos de JavaScript y Backend

Objetivos de la clase

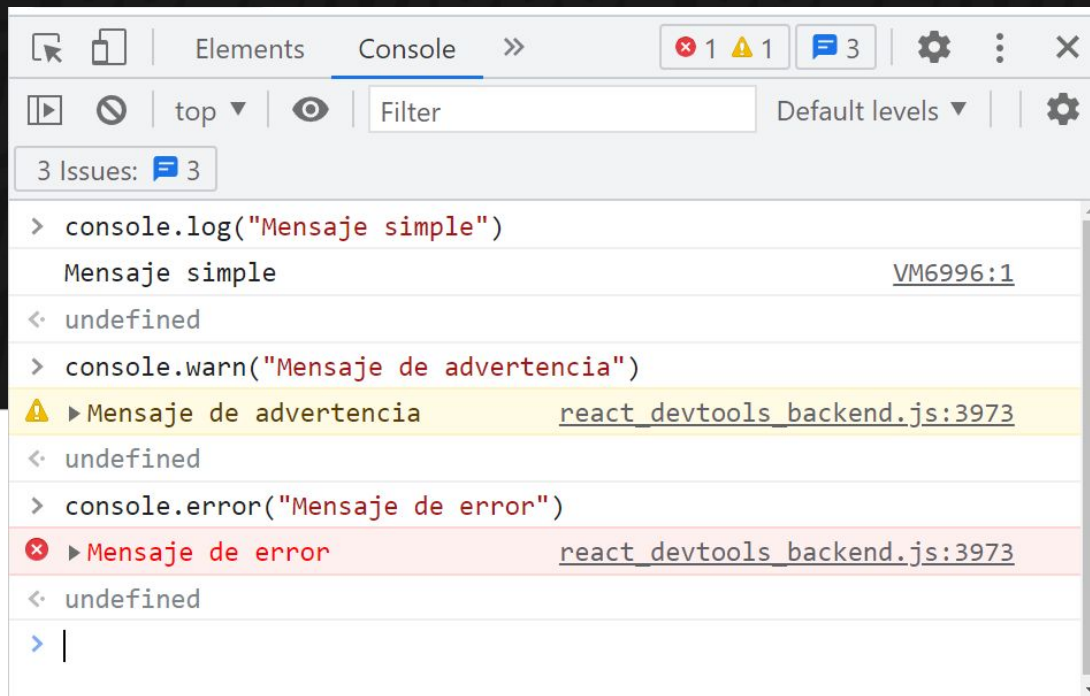
- **Conocer** las diferencias entre programación Front end y Back end
- **Familiarizarse** con las nociones básicas para programar utilizando Javascript y el MERN Stack
- Familiarizarse con las estructuras y conceptos fundamentales al programar utilizando Javascript
- Conocer los nuevos elementos de lenguaje aportados por ES6



Ejemplo en vivo

Consola web: object (.log, .error, .warn y .clear)

Consola de cliente web



Node js

Cuando trabajamos en backend, no tenemos un navegador, es por ello que se precisa de alguna tecnología que nos permita correr código de Javascript, sin necesidad de abrirlo en el navegador. Ahí es cuando utilizamos Node js, para ello, toca considerar:

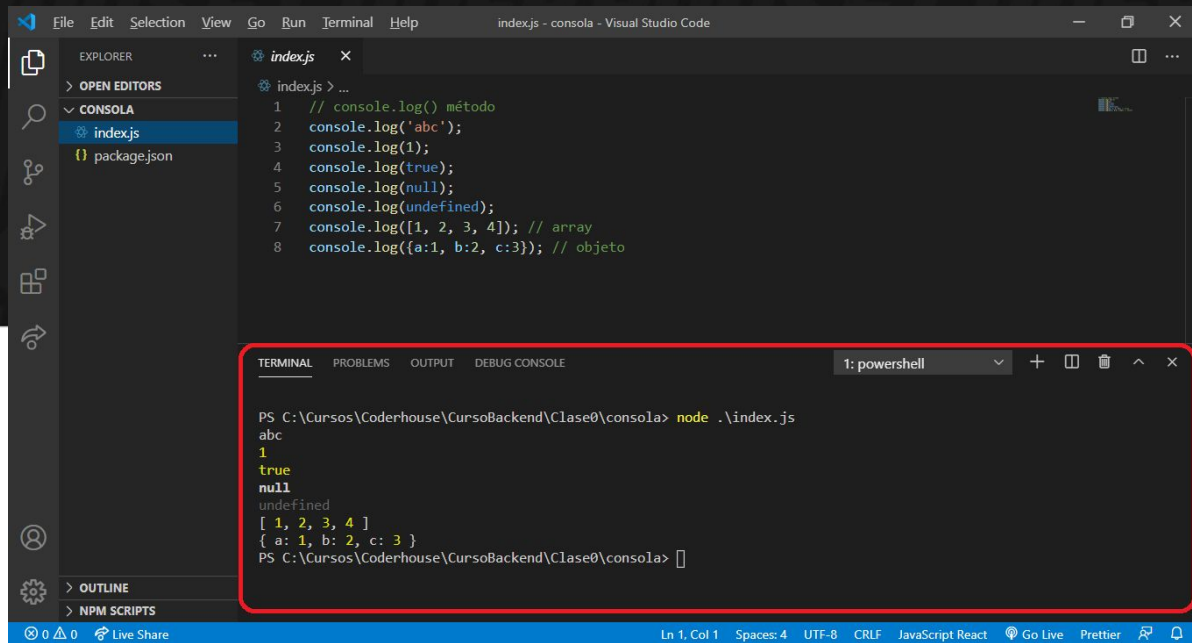
- ✓ Node js levantará un entorno completo para probar nuestras funciones.
- ✓ Debe correrse desde un CLI para poder visualizar los avances del código.
- ✓ Usualmente usaremos el CLI de Visual Studio Code.



Ejemplo en vivo

Consola CLI en Visual Studio Code

Código de Javascript corriendo desde Node js



The image shows a screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'indexjs' with a file 'package.json'. The main editor area displays the contents of 'indexjs', which is a JavaScript file with the following code:

```
1 // console.log() método
2 console.log('abc');
3 console.log(1);
4 console.log(true);
5 console.log(null);
6 console.log(undefined);
7 console.log([1, 2, 3, 4]); // array
8 console.log({a:1, b:2, c:3}); // objeto
```

Below the editor, the TERMINAL panel is open, showing the command 'node .\index.js' executed in a PowerShell window. The output of the command is displayed as follows:

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase0\consola> node .\index.js
abc
1
true
null
undefined
[ 1, 2, 3, 4 ]
{ a: 1, b: 2, c: 3 }
PS C:\Cursos\Coderhouse\CursoBackend\Clase0\consola>
```




Para pensar

¿Cuándo utilizarías un cliente web, y cuándo utilizarías Node js al utilizar Javascript?

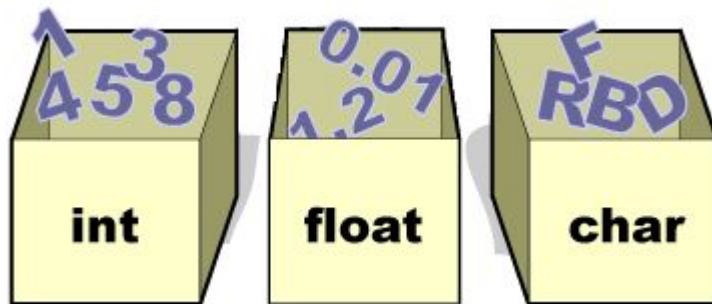
¿Una es reemplazo de la otra, o podríamos considerarlas complementarias?



ACTIVIDAD EN CLASE

Datos y variables

Definir variables que almacenen datos (nombre, edad, precio, nombres de series y películas), mostrar esos valores por consola, incrementar la edad en 1, una serie a la lista y volver a mostrarlas. Compartir la definición en el Visual Studio Code.





Break

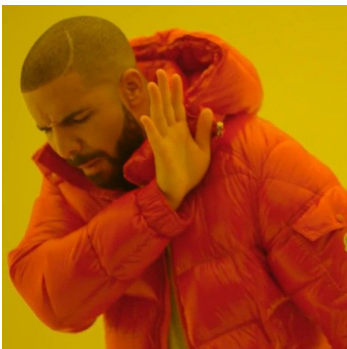
¡10 minutos y volvemos!



Ejemplo en vivo

- Se declara una constante de tipo string. Intentamos cambiar el valor de dicha string. **¿Es posible?**
- Se declara una constante de tipo Array<number>. Intentamos cambiar el último y el primer elemento. **¿Es posible?**
¿Por qué?

Ejemplo de mutabilidad



```
const user = 'Juan';  
//TypeError: Assignment to constant  
user = 'Manolo';
```



```
const user = { name: 'Juan' };  
user.name = 'Manolo';  
console.log(user.name); // Manolo
```



Ejemplo en vivo

- Realizar una función con estructura básica. Se destacarán los elementos y un caso de uso.
- Realizar la misma función con estructura flecha. Se destacarán las diferencias.

Ejemplos de los diferentes tipos de funciones

```
JS 1.js > ☒ sumarDosNumeros
1 function nombreDeLaFuncion(parametros){
2     /*Cuerpo de la función, todas las instrucciones
3     * internas que necesitamos que la función
4     * realice.
5     */
6     let variableParaMiFuncion=2;
7     return variableParaMiFuncion;
8     /* Con la palabra "return" podemos MANDAR FUERA
9     * DEL SCOPE alguna variable que necesite en
10    * otra parte del código.
11    */
12 }
13 /*EJEMPLO COMPLETO*/
14 function sumarDosNumeros(numero1,numero2){
15     //resultado solo existe dentro de la función
16     let resultado;
17     resultado = numero1+numero2;
18     /*Cuando la función acabe, "resultado" muere
19     * así que hay que mandarla afuera
20     */
21     return resultado;
22 }
23 //Mandamos llamar a la función con valores reales
24 let total = sumarDosNumeros(2,3);
25 console.log(total) //5
```

```
JS 2.js > ☒ identificadorDeFuncion
1 /*Una función flecha es ANÓNIMA, quiere decir que
2 * No tiene nombre, pero puede asignarse a una
3 * variable para poder identificarse.
4 */
5 const identificadorDeFuncion = (parametros) =>{
6     /*Cuerpo de la función, todas las instrucciones
7     * internas que necesitamos que la función
8     * realice.
9     */
10    let variableParaMiFuncion=2;
11    return variableParaMiFuncion;
12    /* Con la palabra "return" podemos MANDAR FUERA
13    * DEL SCOPE alguna variable que necesite en
14    * otra parte del código.
15    * La función flecha cuenta con un return
16    * IMPLÍCITO
17    */
18 }
19 /*EJEMPLO COMPLETO */
20 const sumarDosNumeros = (numero1,numero2) =>{
21     let resultado;
22     resultado = numero1+numero2;
23     return resultado;
24 }
25 const sumarReturnImplicito = (num1,num2)=>num1+num2;
```



Hands on lab

En esta instancia de la clase **realizaremos una función que corrobore elementos en una lista.**

¿De qué manera?

El profesor demostrará cómo hacerlo y tú lo puedes ir replicando en tu computadora. Si surgen dudas las puedes compartir para resolverlas en conjunto de la mano de los tutores.

Funciones

¿**Cómo lo hacemos?** Definiremos la función “mostrarLista”, la cual recibirá un arreglo con elementos como parámetro.

- ✓ Si la lista está vacía, devolver un mensaje indicando “Lista vacía”.
- ✓ Si la lista cuenta con elementos, mostrarlos 1 por 1 en consola. Finalizar el proceso devolviendo la longitud de la lista (Utilizar template strings)
- ✓ Invocar la función con los casos de prueba.

Se espera una duración de 10 minutos.



Ejemplo en vivo

- ✓ Se declarará una clase **Persona**, la cual debe crearse con un nombre que identifique la instancia.
- ✓ Además, habrá una variable estática utilizable para todos.
- ✓ Se comprobará la individualidad entre las dos instancias.

JS clases.js x



JS clases.js > Persona > getEspecie

```
48
49 class Persona{
50     constructor(nombre){
51         this.nombre = nombre;
52     }
53     static especie = "humano";
54     saludar = () =>{
55         console.log(`¡Hola, soy ${this.nombre}, mucho gusto!`)
56     }
57     getEspecie = () =>{
58         console.log(`Aunque no lo creas, soy un ${Persona.especie}`)
59     }
60 }
61 let persona1 = new Persona("Jorge");
62 let persona2 = new Persona("Catalina");
63 persona1.saludar();
64 persona2.saludar();
65 persona1.getEspecie();
66 persona2.getEspecie();
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

powerShell + - [] [X] ^ X

```
¡Hola, soy Jorge, mucho gusto!
¡Hola, soy Catalina, mucho gusto!
Aunque no lo creas, soy un humano
Aunque no lo creas, soy un humano
```



Hands on lab

En esta instancia de la clase **repasaremos cómo se crean las clases.**

¿De qué manera?

El profesor demostrará cómo hacerlo y tú lo puedes ir replicando en tu computadora. Si surgen dudas las puedes compartir para resolverlas en conjunto de la mano de los tutores.

Creación de una clase contador

¿Cómo lo hacemos? **Se creará una clase que permitirá llevar cuentas individuales según cada responsable.**

- ✓ Definir clase Contador
- ✓ La clase se creará con un nombre, representando al responsable del contador.
- ✓ El contador debe inicializarse en 0
- ✓ Debe existir una variable estática que funcione como **contador global** de todas las instancias de contador creadas.

Se espera una duración de 10 minutos.

Creación de una clase contador

- ✓ Definir el método `getResponsable`, el cual debe devolver el responsable de dicho contador.
- ✓ Definir el método `contar`, el cual debe incrementar, tanto su cuenta individual, como la cuenta global.
- ✓ Definir el método `getCuentaIndividual`, el cual debe devolver sólo la cuenta individual del contador
- ✓ Definir el método `getCuentaGlobal`, el cual debe devolver la variable estática con el conteo global.
- ✓ Realizar prueba de individualidad entre las instancias.

Muchas gracias.

#DemocratizandoLaEducación