



PRACTICA 1: INTRODUCCIÓN A JAVASCRIPT

INTRODUCCIÓN AL LENGUAJE JAVASCRIPT

¿QUÉ ES EL LENGUAJE JAVASCRIPT?

JavaScript es un lenguaje de programación, de secuencias de comandos, capaz de aportar soluciones eficaces en la mayoría de los ámbitos de la tecnología. Te permite crear contenido de actualización dinámica, controlar multimedia, animar imágenes, etc.

¿PARA QUÉ SE UTILIZA JAVASCRIPT?

Los usos más importantes de JavaScript son los siguientes:

- Desarrollo de sitios web del lado del cliente (front end, en el navegador).
- Desarrollo de aplicaciones para dispositivos móviles, híbridas o que compilan a nativo.
- Construcción de servidores web y aplicaciones de servidor.
- Desarrollo de aplicaciones de escritorio para sistemas Windows, Linux y Mac.
- Desarrollo de juegos.

JAVASCRIPT. CARACTERÍSTICAS

- **Simplicidad.** Posee una estructura sencilla que lo vuelve más fácil de aprender e implementar.
- **Velocidad.** Se ejecuta más rápido que otros lenguajes y favorece la detección de los errores.
- **Versatilidad.** Es compatible con otros lenguajes, como: PHP, Perl y Java. Además, hace que la ciencia de datos y el aprendizaje automático sean accesibles.
- **Popularidad.** Existen numerosos recursos y foros disponibles para ayudar a los principiantes con habilidades y conocimientos limitados.
- **Carga del servidor.** La validación de datos puede realizarse a través del navegador web y las actualizaciones solo se aplican a ciertas secciones de la página web.
- **Actualizaciones.** Se actualiza de forma continua con nuevos frameworks y librerías, esto le asegura relevancia dentro del sector.

¿CUÁLES SON LAS DESVENTAJAS DE JAVASCRIPT?

- **Compatibilidad con los navegadores.** Los diferentes navegadores web interpretan el código JavaScript de forma distinta. Por lo tanto, necesitarás probarlo en todos los navegadores populares, incluyendo las versiones más antiguas.
- **Depuración.** Aunque algunos editores de HTML admiten la depuración, son menos eficaces que otros editores. Encontrar el problema puede ser un reto, ya que los navegadores no muestran ninguna advertencia sobre los errores.



SINTAXIS DEL LENGUAJE

1. REGLAS BÁSICAS DE LA SINTAXIS DEL LENGUAJE

- Las instrucciones en JavaScript terminan en un punto y coma
- Los números en JavaScript que tengan decimales utilizarán el punto como separador de las unidades de la parte decimal.
- Los literales pueden escribirse con comillas dobles o simples.
- Podemos agrupar varias instrucciones en la misma línea separándolas por puntos y comas.
- Los espacios en blanco entre elementos hacen el código más legible.
- Los bloques de código se limitan con llaves.
- Hay una serie de palabras reservadas que no pueden ser utilizadas como identificadores
- JavaScript permite añadir comentarios al código que no serán ejecutados ni visualizados por el navegador. Estos comentarios sirven para explicar el código del programa; por ejemplo, para aclarar una expresión compleja, para indicar cuál es el contenido de una variable, etcétera. Se visualizarán cuando editemos el archivo HTML.

Los comentarios pueden ser:

- **De una línea.** Se indican mediante dos barras inclinadas a la derecha (//). Todo lo que aparezca detrás de dichas barras hasta el final de la línea será considerado comentario y, por tanto, será ignorado por el navegador.
- **De varias líneas.** En este caso deberemos indicar el comienzo del comentario mediante los caracteres barra inclinada a la derecha y asterisco (*). También indicaremos el final del comentario, en este caso mediante la secuencia de caracteres inversa: asterisco y barra inclinada a la derecha (*/).

2. USO DE VARIABLES

Una variable es un espacio en la memoria del ordenador donde se almacenará un valor que podrá cambiar durante la ejecución de un programa.

Para declarar una variable debemos especificar su nombre y ocasionalmente almacenar su valor, no es necesario inicializar siempre la variable con un valor.

Para usar una variable, primero debes crearla, precisamente, a esto lo llamamos declarar la variable. Para hacerlo, escribimos la palabra clave **var** o **let** seguida del nombre con el que deseas llamar a tu variable

En JavaScript la palabra reservada utilizada tradicionalmente para la definición de variables es **var**. Las variables definidas con **var** tienen un ámbito de función. Por lo tanto, pueden ser utilizadas en cualquier punto de la función, incluso antes de ser definidas

Las palabras clave introducidas en el estándar JavaScript ECMAScript 6 para la definición de variables, **let** y **const**, tienen ámbito de bloque. Por lo tanto, solamente pueden ser utilizadas a partir de su definición en el bloque de código

Diferencias entre let, var y const:

let te permite declarar variables limitando su alcance (scope) al bloque, declaración, o expresión



donde se está usando. a diferencia de la palabra clave **var** la cual define una variable global o local en una función sin importar el ámbito del bloque.

las **variables const** no pueden ser modificadas ni re-declaradas.

Definir múltiples variables con el mismo nombre

El uso de **var** permite redefinir la misma variable en el mismo bloque, algo que no se puede hacer con **let**. Aún así en diferentes bloques si que es posible definir múltiples variables con el mismo nombre con **let**. Siendo cada una de ellas una variable diferente. Algo que se puede ver en el siguiente fichero (*ejemplo1.js*)

3. PALABRAS RESERVADAS DE JAVASCRIPT

Identificadores JavaScript:

Un identificador es solo un nombre que identifica algo. Son nombres asignados a objetos, variables, funciones, etc. Obligatoriamente los identificadores deben comenzar o bien con una letra (a), con una barra baja(_) o con el símbolo dólar (\$). **No podremos** utilizar números como primer carácter del nombre de un identificador u otros símbolos que no sean el del dólar o el guion bajo.

Palabras reservadas en JavaScript

abstract	arguments	boolean	break	byte
case	catch	char	class*	const
continue	debugger	default	delete	do
double	else	enum*	eval	export*
extends*	false	final	finally	float
for	function	goto	if	implements
import*	in	instanceof	int	interface
let	long	native	new	null
package	private	protected	public	return
short	static	super*	switch	synchronized
this	throw	throws	transient	true
try	typeof	var	void	volatile
while	with	yield		

4. OPERADORES EN JAVASCRIPT

Los operadores aritméticos, como su propio nombre indica, permiten realizar operaciones matemáticas aritméticas, es decir, las operaciones básicas: suma, resta, multiplicación y división. Además, en programación trabajamos con un operador muy interesante llamado módulo que nos sirve para obtener el resto de una división entera.

Los operadores aritméticos en Javascript pueden dividirse en dos grupos: binarios (requieren de dos operandos) o unarios (se aplican sobre un solo operando).



Por tanto, los **operadores aritméticos binarios** con los que podemos trabajar en Javascript son:

- Operador suma: +.
- Operador resta: -.
- Operador producto: * (¡ojo! Utilizamos el asterisco y no la x).
- Operador división: /.
- Operador módulo o resto: % (¡cuidado! En programación no utilizamos el % para operaciones con porcentajes).

Y los operadores aritméticos unarios son:

- Operador incremento: ++.
- Operador decremento: --.

5. TIPOS DE DATOS:

En Javascript disponemos de los siguientes tipos de datos principales:

Tipo de dato	Descripción	Ejemplo básico
Number	Valor numérico (enteros, decimales, etc...)	42
BigInt	Valor numérico grande 1	234567890123456789n
String	Valor de texto (cadenas de texto, etc...) ' '	MZ'
Boolean	Valor booleano (valores verdadero o falso)	true
Null	Valor para inicializar variables que representan objetos	
undefined	Valor sin definir (variable sin inicializar)	undefined
Function	Función (función guardada en una variable)	function() {}
Symbol	Símbolo (valor único)	Symbol(1)
Object	Objeto (estructura más compleja)	{}

Para saber que tipo de dato tiene una variable, debemos observar que valor le hemos dado. Si es un valor numérico, será de tipo number. Si es un valor de texto, será de tipo string, si es verdadero o falso, será de tipo booleano. Veamos un ejemplo en el que identificaremos que tipo de dato tiene cada variable:

JavaScript proporciona una función que permite identificar el tipo de dato que representa una variable. Dicha función, llamada **typeof**, se utiliza junto la variable a la cual se desea consultar el tipo de dato.

6. CONVERSIÓN DE DATOS ENTRE TIPOS DE DATOS

Existen algunas funciones propias del lenguaje que nos van a permitir convertir cadenas a enteros o reales, evaluar una expresión, etc..

- **parseFloat(cadena)** devuelve un número en coma flotante a partir de la cadena especificada. La función finalizará la conversión cuando encuentre un carácter que no sea un dígito (0..9), un punto decimal (.) o una "e". Ejemplos:
 - `parseFloat("27.11.2000");` devuelve 27.11
 - `parseFloat("20H30M07S");` devuelve 20



- `parseFloat("345.6e5");` devuelve 34560000
- `parseFloat("VUELO506");` devuelve NaN (Not a Number)
- **`parseInt(cadena)`** devuelve un número entero a partir de la cadena especificada. La función finalizará la conversión cuando encuentre un carácter que no sea un dígito. Ejemplos:
 - `parseInt("27.11.2000");` devuelve 27
 - `parseInt("20H30M07S");` devuelve 20
 - `parseInt("345.6e5");` devuelve 345
 - `parseInt("VUELO506");` devuelve NaN (Not a Number)

En principio se considerará como base la decimal (excepto cuando el número comienza con 0x o con 0); pero este formato se puede añadir un segundo argumento que indicaría la base en la que creemos que está el número que queremos convertir.

El formato ampliado será **`parseInt(cadena, base)`**. Ejemplos:

- `parseInt("27", 8);` devuelve 23 `parseInt("11111111", 2);` devuelve 255
- `parseInt("A2", 16);` devuelve 162
- **`toString(argumento)`** convierte a cadena el dato especificado en argumento. Normalmente se utiliza para convertir números a cadena pero también funciona con otros tipos como Booleano, etcétera. Su utilización es restringida ya que JavaScript aplica una conversión automática de tipos a cadena cuando encuentra expresiones de concatenación en las que hay diversos tipos de datos.
- **`typeof(argumento)`** devuelve una cadena que indica el tipo del argumento (number,string, boolean, undefined).

7. INTRODUCCIÓN A LAS FUNCIONES

En programación, cuando nuestro código se va haciendo cada vez más grande, necesitaremos buscar una forma de organizarlo y prepararnos para reutilizarlo y no repetir innecesariamente las mismas tareas. Para ello, un primer recurso muy útil son las funciones.

Una función nos permite agrupar líneas de código en tareas con un nombre, para que podamos hacer referencia a ese nombre.

Pasos para usar funciones en JavaScript

- **Declarar la función:** Preparar la función, darle un nombre y decirle las tareas que realizará.
- **Ejecutar la función:** «Llamar» a la función para que realice las tareas de su contenido.

Ejemplo:

```
// Declaración de la función "saludar"
function saludar() {
  // Contenido de la función
  console.log("Hola, soy una función");
}
```

8. SENTENCIAS DE CONTROL



Al hacer un programa necesitaremos establecer condiciones o decisiones, donde busquemos que se realice una acción A si se cumple una condición o una acción B si no se cumple. Este es el primer tipo de estructuras de control que encontraremos.

Tenemos varias estructuras de control condicionales:

- **Estructura de control if**

If **Condición simple: Si ocurre algo, haz lo siguiente...**

If/else **Condición con alternativa: Si ocurre algo, haz esto, sino, haz lo otro...**

?: **Operador ternario: Equivalente a If/else, forma abreviada.**

Switch **Estructura para casos específicos: Similar a varios If/else anidados.**

Se puede dar el caso que queramos establecer una alternativa a una condición. Para eso utilizamos el if seguido de un else. Con esto podemos establecer una acción A si se cumple la condición, y una acción B si no se cumple.

Condicional If múltiple: Es posible que necesitemos crear un condicional múltiple con más de 2 condiciones, por ejemplo, para establecer la calificación específica. Para ello, podemos anidar varios if/else uno dentro de otro. Sin embargo, anidar de esta forma varios if suele ser muy poco legible y produce un código repetitivo algo feo. En algunos casos se podría utilizar otra estructura de control llamada **switch**, que veremos a continuación.

- **Switch:** La estructura de control switch permite definir casos específicos a realizar cuando la variable expuesta como condición sea igual a los valores que se especifican a continuación mediante cada case:
 - La sentencia switch establece que vamos a realizar múltiples condiciones analizando la variable nota.
 - Cada condición se establece mediante un case, seguido del valor posible de cada caso.
 - El switch comienza evaluando el primer case, y continua con el resto, hacia abajo.
 - Observa que algunos case tienen un break. Esto hace que deje de evaluar y se salga del switch.
 - Los case que no tienen break, no se interrumpen, sino que se salta al siguiente case.
 - El caso especial default es como un else. Si no entra en ninguno de los anteriores, entra en default.
- **Bucles:** Una de las principales ventajas de la programación es la posibilidad de crear bucles y repeticiones para tareas específicas, y que no tengamos que realizar el mismo código varias veces de forma manual.

Tipo de bucle	Descripción
while	Bucles simples.
for	Bucles clásicos por excelencia.
do..while	Bucles simples que se realizan siempre como mínimo una vez.
for..in	Bucles sobre posiciones de un array. Los veremos más adelante.
for..of	Bucles sobre elementos de un array. Los veremos más adelante.

En los bucles se va a evaluar una condición para saber si se debe seguir repitiendo el bucle o se debe finalizar.

Otro concepto es el de iteración, que se refiere a cada una de las repeticiones de un bucle.

Muchas veces necesitamos incorporar un contador que va guardando el número de repeticiones realizadas y así poder finalizar cuando se llegue a un número concreto.

También debemos tener una parte donde se haga un incremento o un decremento.

- **Bucle while:** El bucle while es uno de los bucles más simples que podemos crear.

```
let i = 0; // Inicialización de la variable contador
```

```
// Condición: Mientras la variable contador sea menor de 5
```

```
while (i < 5) {
```

```
  console.log("Valor de i:", i);
```

```
  i = i + 1; // Incrementamos el valor de i
```

```
}
```

- **Bucle for:** La instrucción for permite repetir una instrucción o una instrucción compuesta un número especificado de veces. El cuerpo de una instrucción for se ejecuta cero o más veces hasta que una condición opcional sea false.

```
// for (inicialización; condición; incremento)
```

```
for (let i = 0; i < 5; i++) {
```

```
  console.log("Valor de i:", i);
```

```
}
```

- **Bucle do while:** Este tipo de bucle siempre se ejecuta una vez, al contrario que el bucle while que en algún caso podría no ejecutarse nunca.

```
let i = 5;
```

```
do {
```

```
  console.log("Hola a todos");
```

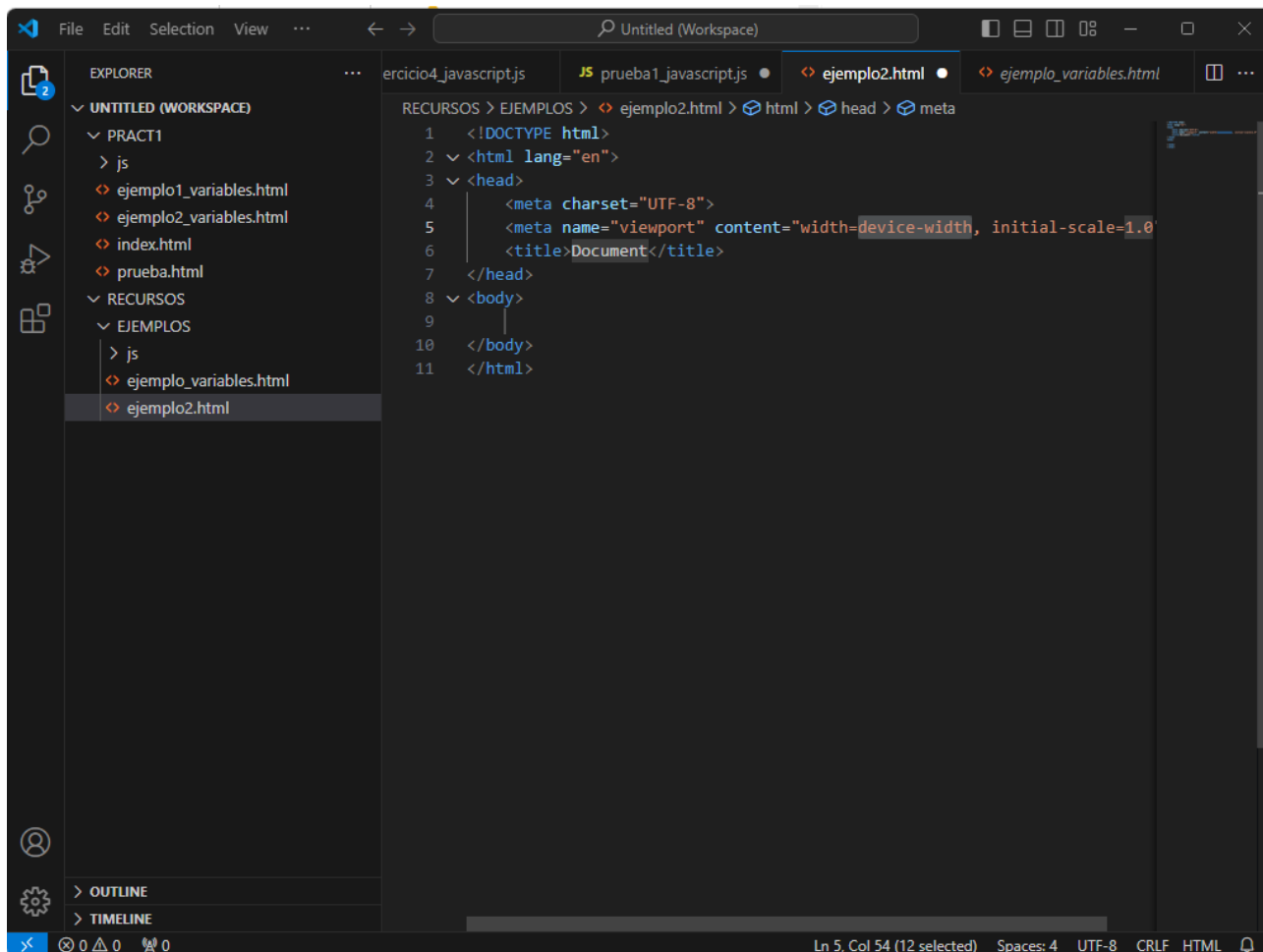
```
  i = i + 1;
```

```
} while (i < 5);
```

```
console.log("Bucle finalizado");
```

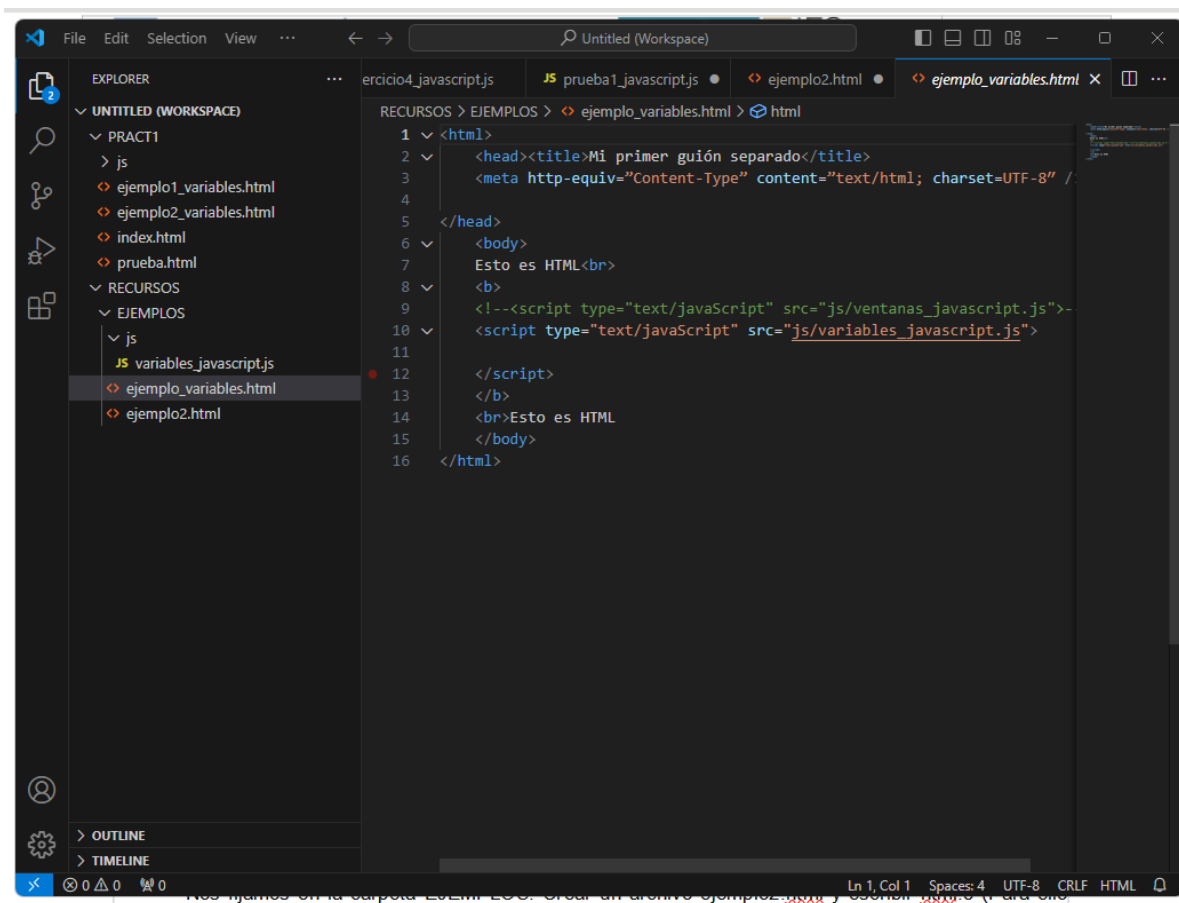
SECUENCIA/DESARROLLO

1. **Ejercicio 1:** Vamos a crear nuestra primera práctica javascript. Para ello creamos una carpeta y ejecutamos visual studio code

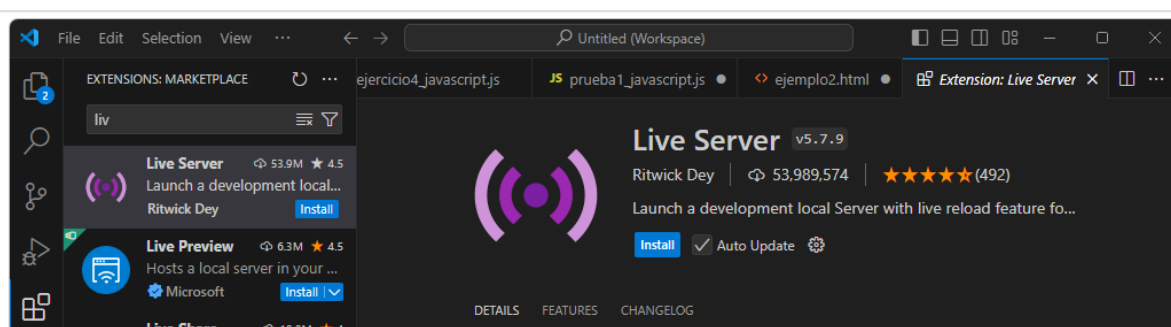


Nos fijamos en la carpeta EJEMPLOS. Crear un archivo ejemplo2.html y escribir html:5 (Para ello debes tener instaladas las extensiones que te indica el archivo de la unidad 01). Al pulsar intro nos aparece la estructura de una página html, como muestra la imagen.

Crear la subcarpeta js y almacenar el archivo javascript el cual se invoca a través de la etiqueta <script> tal y como muestra la siguiente imagen

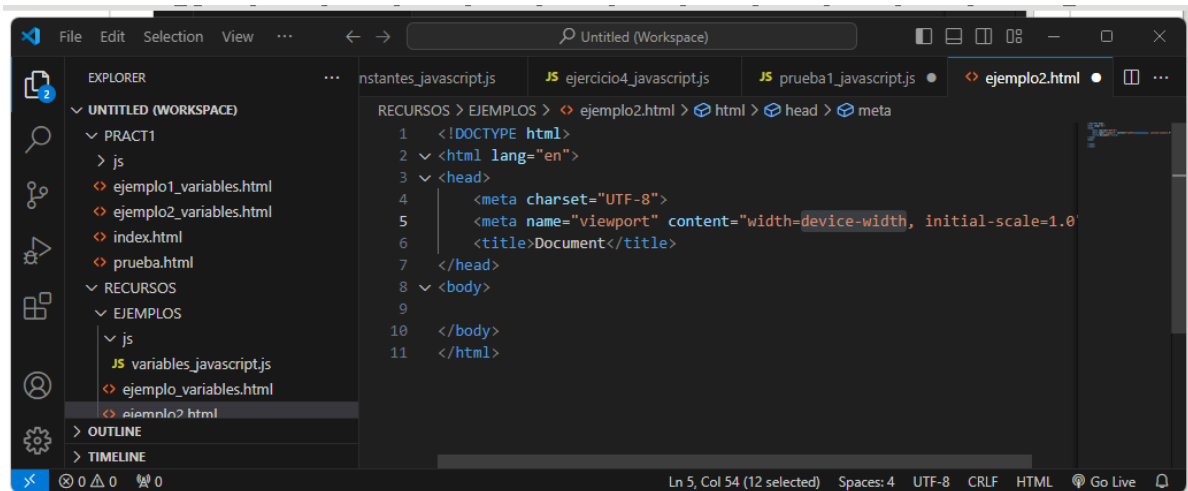


Instalar la extensión liveServer

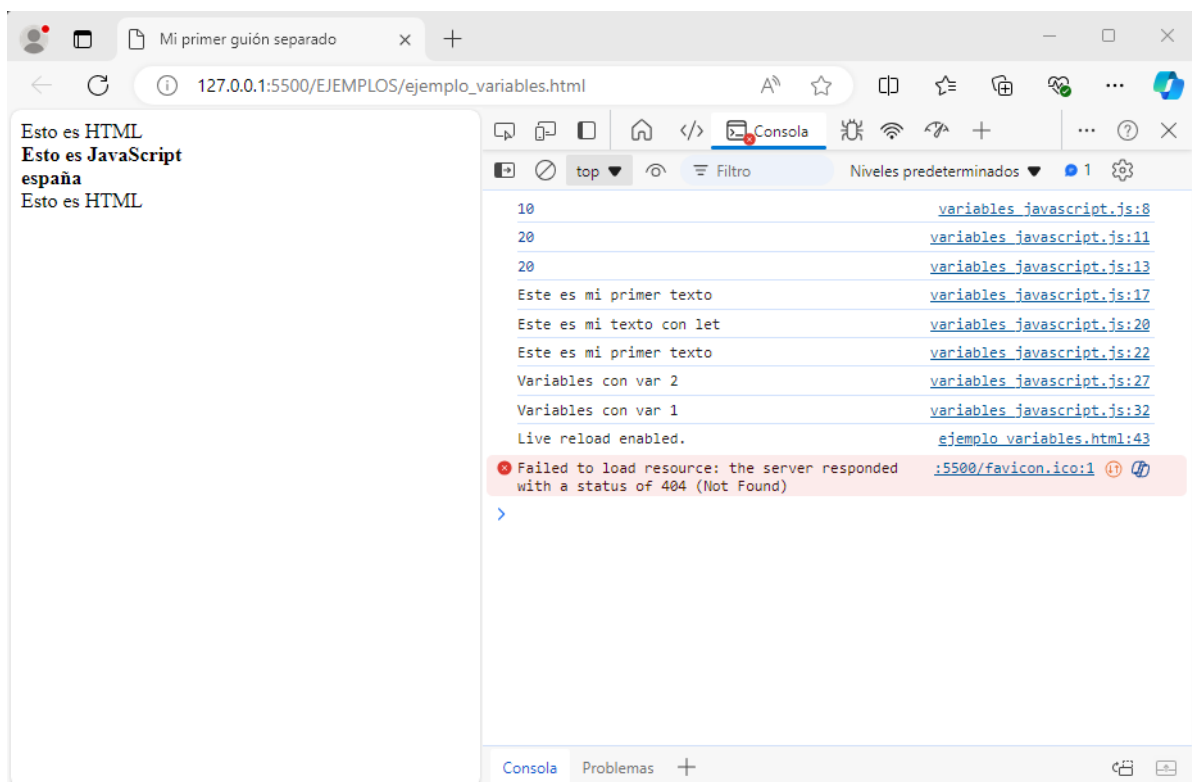


Y a continuación ejecutar el ejemplo. De esta forma es mucho más fácil hacer cambios en el código y visualizarlos en el navegador directamente.

Como puede ver en la siguiente imagen, en la esquina inferior derecha aparece **Go live**, pulsar y a continuación nos muestra la página en el navegador.



Mostramos a continuación el resultado de ejecutar **ejemplo_variable.html**. Como queda reflejado en la siguiente imagen en el navegador debemos mostrar la ventana de inspector de elementos para mostrar los datos con **console.log**



Este método imprime datos sobre la consola de desarrollo. Y es muy útil para tomar notas, verificar el estado de nuestras variables, hacer debugging, y probar nuestro proyecto.

Si ejecutas **console.log(console)**, verás que hay mucho más en el objeto console además del método log que todos conocemos.



2. **Ejercicio 2:** Realiza un script que dados dos números obtenga como resultado la suma, resta, multiplicación y división. Debes declarar todas las variables con las palabras reservadas let, var o const en función de tus necesidades.
3. **Ejercicio 3:** Escribir un programa que lea un número positivo y escriba la tabla de multiplicar de ese número. Para ello debes utilizar la ventana de ingreso de datos (prompt). Debes tener en cuenta que la ventana prompt devuelve un dato no numérico, cuestión a tener en cuenta antes de realizar las operaciones matemáticas. Para comprobar el funcionamiento de prompt tenemos el **ejemploprompt**