TEMA 2

INICIÁNDONOS EN JAVASCRIPT

INTRODUCCIÓN

Durante el primer curso de este ciclo se aprende a trabajar con HTML y CSS, que permiten indicar al navegador cómo organizar y visualizar un documento y su contenido, pero la interacción de estos lenguajes con el usuario y el sistema se limita a un pequeño grupo de respuestas predefinidas. Podemos crear un formulario con campos de entrada, controles y botones, pero con HTML solo podemos enviar la información introducida por el usuario al servidor o para limpiar el formulario. En CSS podemos construir instrucciones con seudoclases como **: hover** para que suceda algo cuando movemos el ratón sobre un elemento, pero si queremos realizar tareas personalizadas, como modificar estilos de varios elementos al mismo tiempo, deberemos cargar una nueva hoja de estilo con esos cambios. Cuando queremos modificar o interactuar con una página web de forma dinámica, los navegadores incluyen un tercer lenguaje llamado JavaScript.

JavaScript es un lenguaje de programación propiamente dicho y podemos generar instrucciones que se ejecutarán de forma secuencial para indicarle al sistema lo que queremos que haga. Cuando el navegador encuentra este código en nuestro documento, ejecuta las instrucciones al momento y las modificaciones realizadas en el documento se ven en pantalla.

JAVASCRIPT

1. IMPLEMENTANDO JAVASCRIPT

Como CSS, el código JavaScript se puede incorporar mediante tres técnicas diferentes: el código se puede insertar en un elemento por medio de atributos (en línea), incorporado al documento como contenido del elemento <script> o cargándolo de un archivo externo. La técnica “en línea” aprovecha atributos especiales que describen un evento, como un clic del ratón. Para lograr que un elemento responda a un evento usando esta técnica, solo tenemos que agregar el atributo correspondiente con el código que queremos que se ejecute.

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <title>JavaScript</title>
5   </head>
6   <body>
7       <p onclick="alert('Has pulsado')";>Pulsa aquí y verás que pasa</p>
8       <br>
9       <p>Cuando pulsas aquí no sucede nada</p>
10  </body>
11  </html>
```

En este caso **onclick** es un atributo agregado al elemento <p> que implica “cuando alguien hace clic en este elemento, ejecutar el código que hay después del igual”.

Además de onclick, tenemos una larga lista de atributos asociada a JavaScript, pero que se pueden organizar en grupos según sus propósitos. Por ejemplo, los atributos más usados asociados al ratón son:

- **onclick**: Responde al evento click. El evento se ejecuta cuando el usuario hace clic con el botón izquierdo del ratón. Existen dos atributos similares **ondblclick** (asociada a hacer doble

clic con el botón izquierdo del ratón) y **oncontextmenu** (el usuario hace clic con el botón derecho del ratón).

- **onmousedown:** Este atributo responde al evento mousedown, que se desencadena si el usuario pulsa el botón izquierdo o el botón derecho del ratón.
- **onmouseup:** Este atributo responde al evento mouseenter, desencadena cuando el ratón se introduce en el área ocupada por el elemento.
- **onmouseenter**: Es el atributo asociado al evento mouseenter, desencadenado cuando el ratón se introduce en el área ocupada por el elemento.
- **onmouseleave**: Asociado al evento mouseleave, que se desencadena cuando el ratón abandona el área ocupada por el elemento.
- **onmouseover**: Asociado al evento mouseover que se desencadena cuando el ratón se mueve sobre el elemento o cualquiera de sus hijos.
- **onmouseout**: este atributo asociado al evento mouseout, que se desencadena cuando el ratón abandona el área ocupada por el elemento o cualquiera de sus hijos.
- **onmousemove**: Atributo asociado al evento mousemove, desencadenada cuando el ratón está sobre el elemento y se mueve.
- **onwheel**: atributo asociado al evento Wheel, que se desencadena cada vez que se hace girar la rueda del ratón.

Algunos de los atributos asociados a eventos generados por el teclado. Estos atributos solo se aplican a elementos que aceptan una entrada del usuario, como **<input>** y **<textarea>.**

- **onkeypress**: Atributo asociado al evento keypress, que se desencadenada cuando se activa el elemento y se pulsa una tecla.
- **onkeydown**: atributo asociado al evento keydown, que se desencadena cuando se activa el elemento y se pulsa una tecla.
- **onkeyup**: Atributo asociado al evento keyup, que se desencadena cuando se activa el elemento y se libera una tecla.

También existe un atributo importante asociado al documento:

- **onload**: Atributo asociado al evento load. Este evento se desencadena cuando un recurso termina de cargarse.

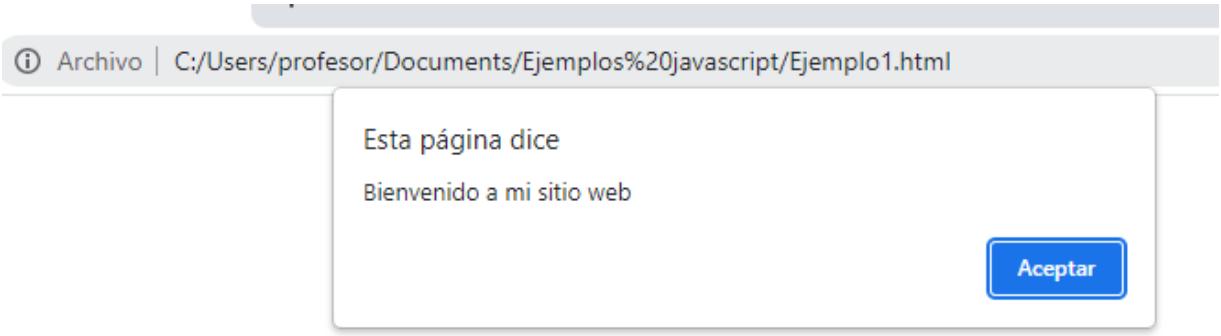
Los atributos de evento se incluyen dependiendo de cuándo queremos que se ejecute el código, podemos incluso añadir más de un atributo a un mismo elemento, veamos un ejemplo modificando el anterior:

```
7     <p onclick="alert('Has pulsado')" onmouseout="alert('No te vayas')">
8         Pulsa aquí y verás que pasa</p>
9     <br>
10    <p>Cuando pulsas aquí no sucede nada</p>
```

Los eventos no solo los produce el usuario, también el navegador. Un evento útil desencadena por el navegador es load, que se desencadena cuando se ha terminado de cargar un recurso, y se usa frecuentemente para ejecutar código JavaScript después de que el navegador ha cargado el documento, para ello incluiremos ese evento en el <body>. Veamos un ejemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>JavaScript</title>
</head>
<body onload="alert('Bienvenido a mi sitio web')">
    <h1>Mi sitio web</h1>
    <br>
    <p>Este es mi sitio web</p>
</body>
</html>
```

Esto es lo primero que nos encontraremos cuando ejecutemos nuestra página web.



El navegador primero carga el contenido del documento y cuando termina, llama a la función alert() que muestra el mensaje en la pantalla.

Los atributos explicados hasta ahora, son útiles si queremos probar código, pero no son apropiados para aplicaciones importantes, en este caso se agrupa el código con el elemento <script>, que actúa igual que <style> en CSS, organizando el código en un solo lugar y afectando al resto de los elementos en el documento usando referencias.

El elemento <script> se puede ubicar en cualquier parte del documento, pero se suele introducir en la cabecera, así cuando el navegador carga el archivo, lee el contenido del elemento y ejecuta el código al instante, y luego continúa procesando el resto del documento. Aunque realmente este elemento puede aparecer cualquier número de veces en el <head> y en el <body>.

Introducir JavaScript en el documento con elementos <script> puede ser práctico si tenemos un grupo pequeño de instrucciones, pero si hay mucho código, o si queremos usar el mismo código en más de un documento, tendremos que mantener diferentes versiones del mismo programa y los navegadores tendrán que descargar el mismo código una y otra vez con cada documento solicitado por el usuario. Lo que podemos hacer es introducir el código JavaScript en un archivo externo y cargarlo desde los documentos que lo requieren. De este modo, solo los documentos que necesitan ese grupo de instrucciones deberán incluir el archivo, y el navegador tendrá que descargar el archivo una sola vez (los navegadores mantienen los archivos en un caché en el ordenador del usuario en caso de que se necesiten más adelante por el mismo sitio web). Para esto, el elemento <script> incluye el atributo src, donde escribiremos la ruta al archivo JavaScript donde hemos escrito todo nuestro código dentro de este archivo.

```
</head>
<body onload="alert('Bienvenido a mi sitio web')">
    <h1>Mi sitio web</h1>
    <script src="archivo.js"></script>
    <p>Este es mi sitio web</p>
</body>
```

El elemento <script> carga el código del archivo JavaScript (con extensión .js). A partir de ahora, podemos insertar este archivo en todos los documentos de nuestro sitio web y reusar el código cada vez que lo necesitemos.

En JavaScript se recomienda finalizar cada instrucción con un punto y coma para asegurarnos que el navegador no tenga ninguna dificultad para identificar el final de cada instrucción, aunque no es obligatorio ponerlo, nos puede ayudar a evitar errores cuando el código está compuesto por múltiples instrucciones.

2. VARIABLES EN JAVASCRIPT

Como ya sabemos, una variable es un contenedor para un valor, como un número que podríamos utilizar para hacer una suma, o una cadena que formaría parte de una frase. La característica más importante de las variables es que pueden cambiar su valor. Además, JavaScript no es un lenguaje de programación tipado, es decir, cuando definimos una variable no debemos especificar qué tipo de valor va a contener, puede empezar conteniendo una cadena de caracteres como un nombre, y después contener en número. Es más, las variables JavaScript pueden contener casi cualquier cosa, desde variables booleanas (true/false) hasta objetos.

2.1 DECLARAR UNA VARIABLE

Para usar una variable, primero debemos crearla, y para ello debemos declararla. Para hacerlo utilizaremos las palabras clave var o let seguida del nombre con el que queremos llamar a nuestra variable:

let nombre;

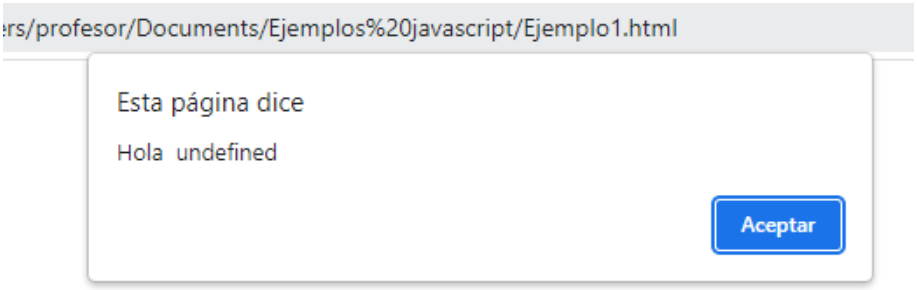
let edad;

Aunque también podemos utilizar la palabra clave var para declarar una variable, en la actualidad, es más común utilizar let. Con las instrucciones anteriores, tenemos dos contenedores vacíos, ya que no hemos asignado ningún valor a nuestras variables. Si intentamos recuperar el contenido de estas variables el resultado será undefined. Si intentamos utilizar una variable que no está declarada lo que recibiremos será un error.

Veamos un ejemplo:

```
<body >
    <h1>Mi sitio web</h1>
    <script >
    let nombre;
    alert("Hola  "+nombre);
    </script>
    <p>Este es mi sitio web</p>
</body>
```

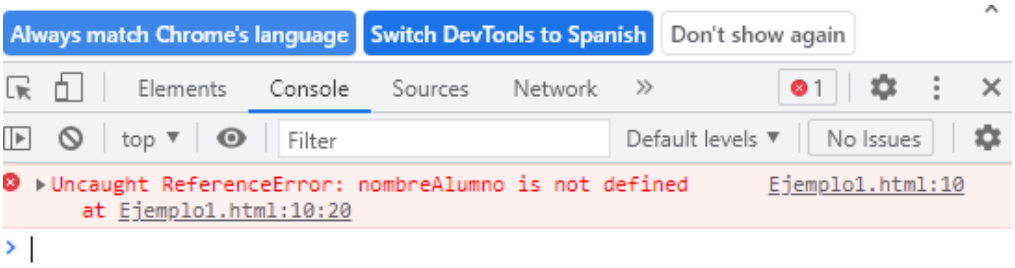
Si ejecutamos este archivo el resultado será una ventana emergente donde aparecerá Hola undefined, ya que la variable existe pero no está inicializada. En otros lenguajes de programación como Java si una variable no está inicializada, recordar que teníamos un problema.



Si el código que incluimos es el siguiente:

```
8    <script >
9    let nombre;
10   alert("Hola  "+nombreAlumno);
11   </script>
```

En este caso la ventana emergente no se ejecuta porque se genera un error, sin embargo, la página web si que se carga sin problema. Para poder ver los errores que genera nuestro código debemos ir a la consola de nuestra página web. Hay muchas formas de ir a esa consola, la más sencilla es pulsar botón derecho del ratón en cualquier parte de la página y seleccionar la opción de inspeccionar. En el caso anterior, esto es lo que nos encontramos en la consola.



Es más, como podemos ver el error que nos da es que **nombreAlumno** no está definida, y nos especifica la línea en la que tenemos el error. Esto en ocasiones puede ser confuso, ya que el error puede estar en otra línea distinta de la marcada, por ejemplo, porque no hemos cerrado una llave en un bucle, o en una función, …

2.2 INICIALIZAR UNA VARIABLE

Una vez que hemos declarado una variable, la podemos inicializar asignando un valor. Para ello, escribiremos el nombre de la variable, seguido de un signo igual (=), seguido del valor que le queremos dar. Cuando el valor sea de tipo carácter, el valor irá encerrado entre comillas dobles o comillas simples; si vamos a insertar un número bastará con escribir el valor numérico correspondiente. A lo largo del curso iremos viendo como insertar otro tipo de datos. Por ejemplo:

nombre =”Maite”;

edad = 18;

Aunque lo normal es declarar e inicializar la variable en la misma instrucción de la siguiente forma:

let nombre = “Maite”;

2.3 DIFERENCIA ENTRE VAR Y LET

Aunque en la definición de variable hemos dado dos palabras clave para definirlas let y var, como podéis ver en los ejemplos he utilizado let. Pero entonces ¿por qué var y let?

Cuando se creó JavaScript por primera vez, solo existía var, y aunque funciona bien en la mayoría de los casos, su diseño a veces puede ser confuso o molesto; por eso se creó en versiones modernas de JavaScript la palabra let que solucionaba estos problemas.

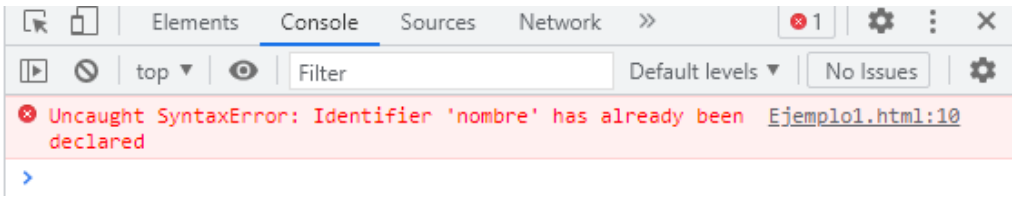
Veamos algunas diferencias simples entre usar var y let. Para ello escribiremos un programa JavaScript de varias líneas que declara e inicia una variable. Si definimos dos veces una variable con la palabra var no nos da ningún error, simplemente incluye en la variable el segundo valor, en el siguiente caso.

```
8   <script >
9   var nombre="Maite";
10  var nombre="Juan";
11  alert("Hola  "+nombre);
12  </script>
```

En este caso nuestro código es muy corto y sabemos perfectamente como funciona esta asignación, pero cuando tenemos muchas líneas de código, var nos permite volver a definir una variable con un nombre que ya tiene asignado, esto puede conllevar muchos problemas. Sin embargo, si intentamos hacer lo mismo con let, nos avisa que esa variable ya está definida, como podemos ver en el siguiente ejemplo:

```
8   <script >
9   let nombre="Maite";
10  let nombre="Juan";
11  alert("Hola  "+nombre);
12  </script>
```

Y en la consola nos aparece que la variable ya había sido definida, como vemos en la siguiente imagen:



Otra cosa que nos puede generar confusión es que si utilizamos var, podemos utilizar la variable y definirla después, pero nos dará error si intentamos hacerlo con let.

```
8   <script >
9   nombre="Maite";
10  var nombre;
11  alert("Hola  "+nombre);
12  </script>
```

Esto no nos dará error y asignará a nombre el valor Maite, pero si lo hacemos con let, nos dirá que la variable no está definida todavía, y nos generará un error cuando ejecutemos la página donde la estamos utilizando. Veamos que es la forma en la que trabajan la mayoría de los lenguajes de programación, luego es mejor intentar trabajar todos de la misma manera.

```
8    <script >
9    nombre="Maite";
10   let nombre;
11   alert("Hola  "+nombre);
12   </script>
```

2.4 NOMBRES DE VARIABLES

Podemos llamar a una variable prácticamente como queramos, aunque existen limitaciones y convenios que se suelen seguir.

1. En general debemos limitarnos a usar caracteres latinos (0-9, a-z, A-Z) y el carácter subrayado.
2. No se deben usar guiones bajos al comienzo de los nombres de las variables, ya que ciertas instrucciones en JavaScript tienen un significado específico, lo que puede hacer que el código sea confuso.
3. La variable no puede comenzar por un número, generará un error.
4. Como en otros lenguajes se suele utilizar la nomenclatura camel, es decir, si está formada por varias palabras, la primera se escribirá en minúsculas, y cada nueva palabra se escribirá comenzando por mayúsculas.
5. Debemos intentar usar nombre intuitivos, que describan los datos que contienen.
6. Las variables distinguen entre mayúsculas y minúsculas (SensitiveCase).
7. Tampoco se pueden utilizar palabras clave reservadas de JavaScript.

2.5 TIPOS DE VARIABLES

Hay algunos tipos de datos que podemos almacenar en las variables JavaScript. Veamos los más específicos, aunque veremos que existen otras más adelante.

➢ NUMEROS

Podemos almacenar números en variables, tanto enteros (a ellos nos solemos referir como integer) o números decimales (denominados number). En JavaScript no necesitamos declarar el tipo de variables que vamos a utilizar, al contrario de lo que sucede en otros lenguajes de programación como java. Cuando queremos guardar en una variable un valor numéricos, no incluiremos las comillas:

let edad = 18;

➢ STRINGS (CADENAS DE CARACTERES)

Los strings son trozos de texto. Cuando queremos que en una variable se guarde un valor de cadena, debemos encerrarlo entre comillas dobles o simples.

let saludo = “Hola, ¿qué tal?”;

let saludo = ‘Hola, buenos días’;

➢ BOOLEANOS

Los booleanos son valores verdadero/falso (true/false). Estos se suelen utilizar para probar una condición, por ejemplo:

let cierto = true;

let test = 6 < 3;

en este caso se usa el operador “menor que “ (<) para probar una condición, es el ejemplo anterior lo que se devuelve es false, ya que no se cumple esa condición.

- **ARREGLOS O ARRAYS**

Un arreglo es un objeto único que contiene múltiples valores encerrados entre corchetes y separados por comas. Veamos un ejemplo:

```
let nombres = [‘Maite’, ‘Jaime’, ‘Lorenzo’];

let edades = [10, 25, 46];
```

Para acceder a sus elementos, usaremos un índice, que nos indica en qué posición está el valor que queremos. En ejemplo anterior **nombres[0]** nos devolverá el valor ‘Maite’ y **edades[2]** nos devolverá 46. Como podemos ver, los elementos del array empieza a contar en el cero, como en otros lenguajes de programación.

- **OBJETOS**

En programación, un objeto es una estructura de código que modela un objeto de la vida real. Podemos tener un objeto sencillo que represente una caja y contenga información sobre su ancho, largo y alto; o podemos tener un objeto que represente una persona y contenga datos como nombre, estatura, peso, idioma, cómo saludarlo, etc.

Para delimitar un objeto usaremos las llaves para delimitar sus características y los dos puntos para separar el nombre del atributo y su valor.

```
let perro = { raza: “Caniche”, color: “blanco”};
```

Para recuperar la información guardada en el objeto utilizaremos el punto y el atributo que queremos recuperar:

```
perro.nombre;
```

2.6 TIPADO DINÁMICO

JavaScript es un lenguaje “tipado dinámicamente”, es decir, no es necesario especificar qué tipo de datos va a contener una variable. Si declaramos una variable y le damos un valor entre comillas, el navegador entenderá que es un string.

```
let number = ‘500’;
```

Aunque el valor que guarda es un número, al estar entre comillas, se entiende de tipo string. Para saber el tipo de valor que tenemos en una variable podemos utilizar el operador **typeof**. En el caso anterior devolverá string.

2.7 CONSTANTES EN JAVASCRIPT

Muchos lenguajes de programación entienden una constante como una variable a la que asignamos un valor, que no se puede cambiar. Esto se utiliza para evitar que un script de terceros puede cambiar el valor.

Al principio en JavaScript no existían las constantes, actualmente usaremos la palabra clave **const** para almacenar valores que nunca se pueden cambiar.

```
const numeroMeses = 12;
```

const funciona igual que let, solo que en este caso a numeroMeses no se le puede dar un nuevo valor, y tiene sentido, ya que el número de meses es 12.

OPERACIONES EN JAVASCRIPT

1. OPERADORES ARITMÉTICOS

Como ya hemos dicho JavaScript no distingue entre los diferentes tipos de números que podemos utilizar, todos los datos de tipo número se denomina Number. Es decir, cualquiera que sea el tipo de números con los que se trata, JavaScript los maneja de la misma manera.

Los operadores aritméticos son los operadores básicos que usamos para hacer operaciones en JavaScript.

Operador	Nombre	Objetivo	Ejemplo
+	Suma	Suma dos números.	6 + 9
-	Resta	Resta el número de la derecha del de la izquierda.	15 - 9
*	Multiplicación	Multiplica dos números juntos	3 * 7
/	División	Divide el número de la izquierda por el de la derecha	15 / 5
**	Potencia	Eleva un número al exponente que le indicamos como segundo valor	3 ** 5(es lo mismo que 3 * 3 * 3 * 3 * 3)
%	Resto de la división entera	Devuelve el resto que queda tras dividir el número de la izquierda por el de la derecha sin sacar decimales	8 % 3 (devolvería 2)
++	Incremento 1	Incrementa en 1 la variable a la que esté acompañando	X++ equivale a X=X+1
--	Decremento 1	Decrementa en 1 la variable a la que acompaña	x—equivale a x=x-1;

2. PRECEDENCIA DEL OPERADOR

Al igual que sucede en matemáticas, algunos operadores se aplican antes que otros cuando se calcula el resultado al aplicar operadores. En general, la precedencia es la misma que se enseña en las clases de matemáticas en la escuela: primero se multiplican y dividen, luego se suman y restan; y el cálculo siempre se evalúa de izquierda a derecha.

Para anular la precedencia del operador, puede poner paréntesis alrededor de las partes que desea que se traten explícitamente primero.

3. OPERADORES DE INCREMENTO Y DECREMENTO

A veces queremos sumar o restar repetidamente uno al valor de una variable numérica. Para ello podemos utilizar el operador **++** para incrementar, y **--** para decrementar.

```
let number = 5;

number++; // En este caso la variable tendrá el valor 6
```

aunque finalmente lo que vale la variable es 6, si utilizamos number en alguna función, primero usa su valor 4 y luego incrementa. Si queremos que primero se incremente y luego se use utilizaremos el ++ antes, de la siguiente forma:

```
++number; // En este caso la variable tendrá el valor 6 y luego ser
```

4. OPERADORES DE ASIGNACIÓN

Los operadores de asignación asignan un valor a una variable. El más básico "**=**", asigna a la variable de la izquierda el valor indicado a la derecha.

```
let x = 3;

let y = 4;

x = y // se asigna a x el valor 4
```

Hay algunos tipos más complejos, que proporcionan atajos útiles para mantener su código más ordenado y eficiente. Los más comunes son:

Operador	Nombre	Objetivo	Ejemplo
+=	Asignación suma	Suma el valor de la derecha al valor de la variable de la izquierda.	x += 4 (x = x+4)
-=	Asignación resta	Resta el valor de la derecha del valor de la variable de la izquierda	x -= 3 (x = x-3)
***=**	Asignación de multiplicación	Multiplica el valor de la derecha del valor de la variable de la izquierda	x *= 3 (x = x*3)
/=	Asignación de división	Divide el valor de la derecha del valor de la variable de la izquierda	x / = 5; (x = x/5)

5. OPERADORES DE COMPARACIÓN

Utilizamos los operadores de comparación para ejecutar pruebas de verdadero/falso. En la siguiente tabla podemos ver algunos de los operadores de comparación que podemos utilizar.

Operador	Nombre	Objetivo	Ejemplo
===	Igual estricto	Comprueba si los valores de la izquierda y la derecha son idénticos entre sí, incluyendo si son del mismo tipo	5 === 2 + 4
!==	Igual no estricto	Comprueba si los valores izquierdo y derecho **no** son idénticos entre sí	5!== 2 + 4

<	Menor que	Comprueba si el valor izquierdo es menor que el derecho.	6 < 10
>	Mayor que	Comprueba si el valor izquierdo es mayor que el derecho.	6 > 10
<=	Menor o igual a	Comprueba si el valor izquierdo es menor o igual que el derecho.	6 <= 10
>=	Mayor o igual a	Comprueba si el valor izquierdo es mayor o igual que el derecho..	6 >= 10

Es posible que en algunos códigos se utilice == y ¡= en sus pruebas de igualdad y no igualdad. Estos operadores son válidos en JavaScript, pero son distintos de === y ¡==. En el primer caso se comprueba si los valores son iguales, pero el tipo de datos puede ser diferente, mientras que en el segundo caso se comprueba que además del mismo valor, el tipo de datos son los mismos. Cuando se utiliza la versión estricta se reducen el número de errores que no se detectan, por lo que se recomienda utilizar esta opción.