

2. INTRODUCCIÓN A LA PROGRAMACIÓN EN JAVASCRIPT

Los conceptos de HTML y CSS permiten indicar al navegador cómo se organiza y visualiza 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. Por ejemplo, los enlaces y botones solo permiten ciertas acciones estándar, como la redirección a otra página o la ejecución de estilos visuales estáticos.

Es aquí donde entra en juego JavaScript. Este lenguaje permite dotar de dinamismo a las páginas web, proporcionando una interacción mucho más rica y adaptable con el usuario. A través de JavaScript, no solo podemos responder a eventos como clics o movimientos del ratón, sino que también podemos modificar el DOM en tiempo real, realizar peticiones asíncronas a servidores, validar formularios, crear animaciones, entre muchas otras funcionalidades.

Este tema es fundamental para abordar con éxito el desarrollo de aplicaciones web en el entorno cliente utilizando JavaScript. El objetivo es enseñar a programar específicamente para el cliente web, por lo que se asume que ya se cuenta con conocimientos previos sobre conceptos como la Programación Orientada a Objetos (POO), HTML, HTTP, entre otros. Por ello, nos centraremos exclusivamente en el lenguaje JavaScript y sus particularidades en este contexto.

1. Cómo ejecutar código JavaScript:

Para ejecutar código JavaScript, es necesario utilizar un archivo que pueda ser interpretado por el navegador, lo que implica que lo integraremos en un documento HTML para que se ejecute en el entorno cliente (navegador). Hay tres formas:

- **Integrar JavaScript directamente en HTML:** Se realiza utilizando la etiqueta `<script>`. Es útil para pequeños fragmentos de código.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Mi página web</title>
5  </head>
6  <body>
7    <h1>Bienvenidos a mi página web</h1>
8    <script>
9      console.log('Hola mundo!');
10   </script>
11 </body>
12 </html>
13 |

```

- **Incluir JavaScript a través de un archivo externo:** Se utiliza también mediante la etiqueta `<script>`, pero añadiendo el atributo `src` para especificar la ruta del archivo. Permite una mejor organización del código y reutilización.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Mi página web</title>
5  </head>
6  <body>
7    <h1>Bienvenidos a mi página web</h1>
8    <script src="ruta/del/archivo.js"></script>
9  </body>
10 </html>
11 |

```

- **Usar JavaScript con eventos:** Existen determinados atributos especiales que describen un evento (click del botón) y permite ejecutar código JavaScript en respuesta a las acciones del usuario.

```

<body>
  <h1>Bienvenidos a mi página web</h1>
  <button onclick="alert('¡Has pulsado el botón!')">Haz clic aquí</button>
</body>

```

Algunos de los atributos más conocidos son:

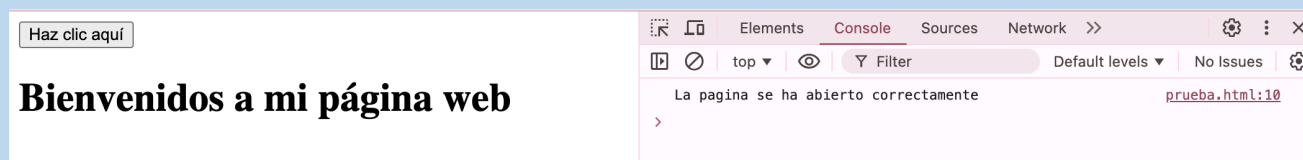
Evento	Descripción
onclick	Se ejecuta cuando el usuario hace clic en el elemento.
ondblclick	Se activa cuando el usuario hace doble clic en el elemento.
onmouseover	Ocurre cuando el puntero de ratón se coloca sobre un elemento.

onmouseout	Se ejecuta cuando el puntero del ratón se coloca sobre un elemento.
onkeydown	Se activa cuando el usuario presiona una tecla
onkeyup	Ocurre cuando el usuario suelta una tecla después de haberla presionado.
onfocus	Se activa cuando un elemento, recibe el foco (cuando el usuario lo selecciona). Por ejemplo en un campo de formulario.
Onblur	Se dispara cuando un elemento pierde el foco, cuando se pasa a otro elemento.
Onchange	Se ejecuta cuando se cambia el valor de un elemento, como un campo de texto o elemento desplegable y el usuario sale del elemento (se pierde el foco).
Onload	Se activa cuando la página web o elemento específico, como una imagen o script, se ha cargado completamente.
Onunload	Se ejecuta cuando el usuario está a punto de abandonar la página web, generalmente al cerrar la pestaña o navegar a otra página.
Ondrop	Se ejecuta cuando el usuario suelta el elemento arrastrado en el área de destino.
Ondrag	Se dispara cuando se arrastra un elemento y generalmente se combina con ondrop

Como se puede observar, se ha colocado el código dentro de la sección body de la web, pero no tiene por qué ser así. Técnicamente podría colocarse en cualquier parte del documento, pero hay unas “normas de estilo”:

- Se coloca en <body> si se va a interactuar con los elementos del DOM.
- Se coloca <head> si se van a incluir librerías o declarar estructuras de datos globales.
- Se coloca dentro de la etiqueta HTML si se quiere modificar el comportamiento de ese único elemento.

La forma de comprobar que todo funciona correctamente es con la consola de depuración avanzada que se incluye en los navegadores modernos que permite interactuar con el CSS, el JS.. Se puede abrir con F12 o con el botón derecho -> inspeccionar. Esta consola la mantendremos abierta casi todo el tiempo.



2. Comentarios:

A lo largo del proceso de desarrollo de aplicaciones en entorno cliente va a surgir la necesidad de dejar comentarios en el código. Se trata de anotaciones, indicadores, recordatorios o apuntes que se dejan en el código para aumentar la información sobre su significado. Los comentarios no alteran el flujo de ejecución del programa ni intervienen en la lógica de la aplicación

```

13      // Comentarios de una línea
14
15      /*
16      Comentarios de varias líneas
17      *
18      */

```

3. Variables:

Una variable es una zona de memoria a la que se asigna un nombre y en la que se guarda un valor. De esta forma, puede hacerse referencia a esa zona de memoria mediante un identificador que el programador ha definido, recuperar su valor o modificarlo.

```

// Única manera antes de ES6. Ya no hay motivo para utilizarlo. (Prohibido en clase)
var a = 1;

// Soluciona problemas de var.
// No se puede declarar dos veces.
let a = 1;

```

```
// const a = 1;
// No se puede reasignar el valor.
const a = 1;

// Declaración con valor 'undefined'.
// Asignación del valor.
let a;
a = 1;

//Declaración y asignación de valor
let a = 1;
```

Como ya sabemos JavaScript es un lenguaje **no tipado**, eso significa que no es necesario declarar el tipo de datos de una variable al definirla. Pero, la correcta definición sería decir que JavaScript es un lenguaje **dinámicamente tipado**, ya que aunque no requiere que se declara el tipo de datos, una vez que la variable es asignada, adquiere ese tipo y puede cambiar de tipo a lo largo de la ejecución del programa.

Los **tipos de datos que soporta** son:

- **String:** Es una secuencia de caracteres que se utiliza para representar texto.

```
//Si se elige un formato se debe seguir ese patrón para todos los String
let miString_1 = "Con comillas dobles, ";
let miString_2 = 'Con comillas simples. ';
let miString_3 = `Con comillas invertidas. `;

//para las comillas dobles y comillas simples se puede concatenar o incluir expresiones con (+):
let cadenaFinal = "Cadena: " + miString_1 + miString_2;
console.log(cadenaFinal);

//para las comillas invertidas se puede incluir expresiones incluyendolo entre ${}:
let cadenaFinal2 = `Cadena: ${miString_1} y también ${miString_3}`;
console.log(cadenaFinal2);
```

Cadena: Con comillas dobles, Con comillas simples.

[prueba.html:51](#)

Cadena: Con comillas dobles, y también Con comillas invertidas.

[prueba.html:56](#)

Para finalizar los String es importante señalar que existe un conjunto de caracteres que no se pueden escribir, pero que son fundamentales en algunas situaciones. Son lo que se conoce como secuencias de escape:

Secuencia	Uso
\'	Comillas simples
\"	Comillas dobles

\\	Barra invertida
\f	Salto de página
\n	Salto de línea
\t	Tabulador horizontal
\v	Tabulador vertical

Los ejemplos más comunes son:

```
// Comillas simples
console.log('Este es un ejemplo de comillas simples: \'Hola, Mundo!\'');

// Comillas dobles
console.log("Este es un ejemplo de comillas dobles: \"Hola, Mundo!\"");

// Tabulador
console.log("Este es un ejemplo con tabulador:\tHola, Mundo!");

// Barra invertida
console.log("Esta es una barra invertida: C:\\Archivos\\mi_archivo.txt");

// Salto de línea
console.log("Este es un ejemplo de salto de línea:\nHola,\nMundo!");
```

Y así es como se observa la salida por consola de las cadenas de caracteres en las que se han incluido la sentencia de escape:

Este es un ejemplo de comillas simples: 'Hola, Mundo!'	prueba.html:60
Este es un ejemplo de comillas dobles: "Hola, Mundo!"	prueba.html:63
Este es un ejemplo con tabulador: Hola, Mundo!	prueba.html:66
Esta es una barra invertida: C:\Archivos\mi_archivo.txt	prueba.html:69
Este es un ejemplo de salto de línea: Hola, Mundo!	prueba.html:72

- **Number**: Hace referencia a un único tipo de dato numérico que recoge cualquier formato de número con o sin decimales, aunque los recoge con formato de doble precisión de 64 bits.

```

79  /*
80  Si no doy por hecho que siempre se usará el sistema de representación decimal
81  en los programas, tengo que indicar delante del 0 si es binario (b), octal (o)
82  o se trata de hexadecimal (x).
83
84  Por ejemplo, el 32 representado en todos los sistemas de numeración:
85
86  Decimal:   32
87  Binario:   0b100000
88  Octal:     0o40
89  Hexadecimal: 0x20
90  */
91
92  let decimal = 32;           // Decimal
93  let binario = 0b100000;     // Binario
94  let octal = 0o40;           // Octal
95  let hexadecimal = 0x20;     // Hexadecimal

```

A la hora de mostrar la salida por pantalla de los distintos sistemas de numeración:

Decimal: 32	prueba.html:99
Binario: 32	prueba.html:100
Octal: 32	prueba.html:101
Hexadecimal: 32	prueba.html:102

¿Qué ha ocurrido? Lo que realmente ocurre es que la consola está configurada para trabajar por **defecto con el sistema de numeración decimal**.

Otra cosa interesante es cómo se manejan las **divisiones por cero**. En la mayoría de los lenguajes de programación, esto devolvería un error. Sin embargo, veamos qué ocurre con JavaScript:

```

let divisionPorCero = 23 / 0; // División por cero (positivo)
let divisionPorCeroNegativo = -23 / 0; // División por cero (negativo)
let divisionIndefinida = 0 / 0; // 0 dividido por 0 (indefinido)
let otraOperacion = divisionPorCero + 12; // Sumar 12 a Infinity

console.log("División por cero: " + divisionPorCero);
console.log("División por cero negativo: " + divisionPorCeroNegativo);
console.log("División indefinida (0/0): " + divisionIndefinida);
console.log("Otra operación: " + otraOperacion);

```

En JavaScript, al dividir un número por cero, se obtiene **Infinity**. Esto significa que el lenguaje es capaz de entender y operar con el concepto de infinito.

División por cero: Infinity	prueba.html:109
División por cero negativo: -Infinity	prueba.html:110
División indefinida (0/0): NaN	prueba.html:111
Otra operación: Infinity	prueba.html:112

Para finalizar, en JavaScript si intentas realizar operaciones matemáticas con distintos tipos de datos :

```
let numero = 54;
let numero2 = 1;
let cadena = "cadena de texto";
let cadena2 = "2"

let resultado = numero * cadena; //string (texto) * número
let resultado1 = numero + cadena2; //string (número) * cadena
let resultado2 = numero + numero; // número * número
let resultado3 = cadena + cadena2; //cadena * cadena

console.log(resultado);
console.log(resultado1);
console.log(resultado2);
console.log(resultado3);
```

NaN	prueba.html:126
542	prueba.html:127
108	prueba.html:128
cadena de texto2	prueba.html:129

EL resultado que nos ha arrojado en la primera operación es **NaN** eso significa **Not-a-Number** y representa un valor que no es un número válido. Es un tipo especial de valor que indica que una operación aritmética o de conversión ha fallado o ha dado un resultado indefinido.

- **Undefined:** En JavaScript se obtiene el valor undefined cuando todavía no se ha asignado un valor a una variable.
- **Null:** Es el valor vacío o nulo.

La diferencia entre estos dos tipos de datos, null y undefined, es que **undefined** no hay valor por que aún no se ha definido; en cambio, **null** significa que no hay valor porque así lo ha indicado expresamente el programador.

- **Booleano:** Es un tipo de dato lógico que solo puede tomar dos valores, true o false. Ambas son palabras reservadas que no se pueden asignar a otro campo. Sin embargo, a la hora de evaluar expresiones lógicas se considera que todo lo que sea igual a cero es falso y todo lo que sea distinto de cero es verdadero.

```
// Boolean true y false
console.log("¿true?:", Boolean(true)); // true
console.log("¿false?:", Boolean(false)); // false

// Boolean con números
console.log("Boolean(1):", Boolean(1)); // true
console.log("Boolean(0):", Boolean(0)); // false

// Boolean con cadenas de texto
console.log('Boolean("texto"):', Boolean("texto")); // true
console.log('Boolean(""):', Boolean("")); // false

// Boolean con Infinity
console.log("Boolean(Infinity):", Boolean(Infinity)); // true

// Boolean con NaN
console.log("Boolean(NaN):", Boolean(NaN)); // false
```

¿true?: true	prueba.html:133
¿false?: false	prueba.html:134
Boolean(1): true	prueba.html:137
Boolean(0): false	prueba.html:138
Boolean("texto"): true	prueba.html:141
Boolean(""): false	prueba.html:142
Boolean(Infinity): true	prueba.html:145
Boolean(NaN): false	prueba.html:148

- **Object:** 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;**

- **Function:** Una función es referenciable como una variable, pero se verá más adelante en los siguientes temas.

Si se desea obtener que tipo de dato es una variable concreta, se puede utilizar la función **typeof()**:

```
let array_mix = [
  "abcdef", 2, 2.1, 2.9e3, 2e-3,
  0o234, 0x23AF, true, [1,2,3], {'a': 1, 'b': 2}
];
for (let i=0; i<array_mix.length; i++) {
  console.log(typeof(array_mix[i]));
}
```

6	number	prueba.html:156
	boolean	prueba.html:156
2	object	prueba.html:156

Como ya hemos visto anteriormente, JavaScript es un lenguaje débilmente tipado. Esto significa que el lenguaje es capaz de lidiar con operaciones donde se mezclan tipos de datos distintos sin lanzar errores constantemente, como ocurre en otros lenguajes de programación. Esto lo logra gracias a su potente capacidad de convertir unos tipos de datos en otros de forma dinámica. Existen dos formas de realizar este tipo de conversiones: sin intervención del programador, o con la indicación explícita por parte del programador:

- **Conversión automática de tipos:**

La regla de oro que aplica este lenguaje para realizar su conversión desasistida de tipos es actuar de la forma más razonable, pero hay ocasiones en las que las conversiones sencillamente no son posibles:

```
console.log(`1. true*7= ${true*7}`);
console.log(`2. 9-false= ${9-false}`);
console.log(`3. 12*"5"= ${12+"5"}`);
console.log(`4. "67"+11= ${"67"+11}`);
console.log(`5. 12*"5"= ${12*"5"}`);
console.log(`6. "67"*11= ${"67"*11}`);
console.log(`7. "texto"*8= ${"texto"*8}`);
console.log(`8. undefined/8= ${undefined/8}`);
console.log(`9. null*6= ${null*6}`);
console.log(`10. Infinity - 6= ${Infinity - 6}`);
console.log(`11. NaN + 4= ${NaN + 4}`);
```

1. true*7= 7	prueba.html:160
2. 9–false= 9	prueba.html:161
3. 12*"5"= 125	prueba.html:162
4. "67"+11= 6711	prueba.html:163
5. 12*"5"= 60	prueba.html:164
6. "67"*11= 737	prueba.html:165
7. "texto"*8= NaN	prueba.html:166
8. undefined/8= NaN	prueba.html:167
9. null*6= 0	prueba.html:168
10. Infinity – 6= Infinity	prueba.html:169
11. NaN + 4= NaN	prueba.html:170

- **Conversión manual de tipos:**

Conversión	Método/Función	Descripción
Número	Number()	Convierte un valor a un número. Si el valor no es convertible, devuelve NaN (Not-a-Number). Funciona para cadenas numéricas, booleanos (true → 1, false → 0), etc.
	parseInt()	Convierte una cadena a un número entero. Si la cadena empieza con un número, convierte hasta que encuentra un carácter no numérico. Si no puede convertir, devuelve NaN.
	parseFloat()	Convierte una cadena a un número de punto flotante (decimal). Similar a parseInt(), convierte hasta que encuentra un carácter no numérico.
A Cadena	String()	Convierte cualquier valor a una cadena. Funciona con números, booleanos, null, undefined, objetos, etc.
	toString()	Convierte un valor a cadena. Similar a String(), pero no funciona con null o undefined.
A booleano	Boolean()	Convierte un valor a un booleano (true o false). Los valores como 0, NaN, null, undefined, y "" (cadena vacía) son false, mientras que cualquier otro valor es true.

4. Entrada y salida navegador:

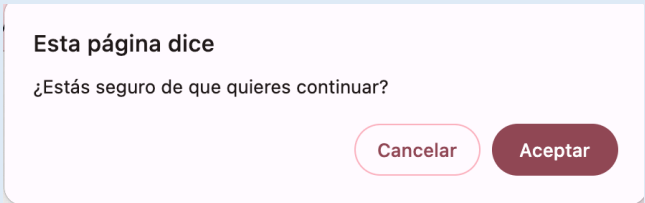
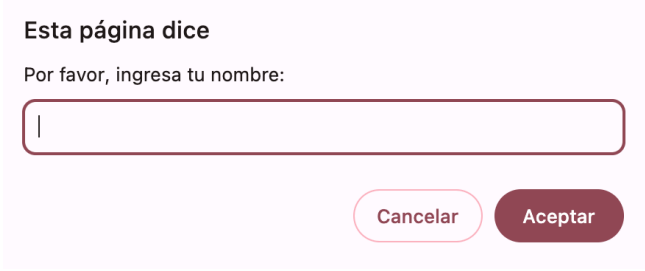
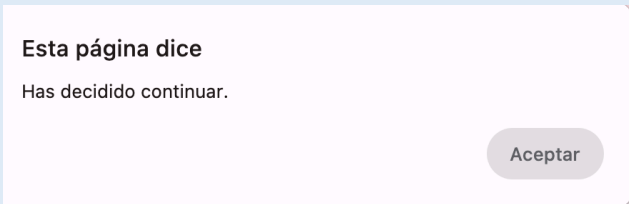
Para mejorar la comprensión de los conceptos visto hasta ahora, se venían utilizando algunas de las funciones que ofrecen los navegadores para sacar datos en pantalla. Así que como habéis visto un recurso muy habitual que utilizan los programadores durante el desarrollo de sus aplicaciones enviar texto a la consola. Es bastante útil porque permite obtener mucha información de depuración de los programas sin “ensuciar” el aspecto de estos programas y, al mismo tiempo, son fácilmente localizables para retirarlos antes de poner las aplicaciones en entornos de producción.

Si trabajamos con la consola podemos distinguir:

Método	Descripción
Console.log()	Muestra información o mensajes más detallados en la consola, típicamente para notificaciones informativas.
Console.info()	Muestra información o mensajes más detallados en la consola, típicamente para notificaciones informativas.
Console.warn()	Muestra advertencias en la consola, generalmente en un color diferente (amarillo o naranja). Se usa para advertir sobre posibles problemas en el código.
Console.error()	Muestra mensajes de error en la consola. Suele usarse cuando ocurre un fallo importante en el código y aparece en rojo para mayor visibilidad.

Si hablamos de alertas o mensajes para el usuario podemos distinguir entre:

Método	Descripción
Confirm(“Mensaje”)	Muestra un cuadro de diálogo con un mensaje y dos botones: "Aceptar" y "Cancelar". Devuelve true si el usuario selecciona "Aceptar", y false si

	<p>selecciona "Cancelar".</p> 
Prompt("Mensaje")	<p>Muestra un cuadro de diálogo con un campo de entrada, donde el usuario puede introducir un valor. Devuelve el texto introducido por el usuario o null si se cancela.</p> 
Alert ("Mensaje")	<p>Muestra un cuadro de diálogo con un mensaje y un botón "Aceptar". Se usa para mostrar información al usuario.</p> 

5. Operadores:

- **Operadores aritméticos:**

Operador	Descripción	Ejemplo
+	Suma	5 + 2 // 7
-	Resta	5 - 2 // 3
*	Multiplicación	5 * 2 // 10
/	División	5 / 2 // 2.5
%	Módulo	5 % 2 // 1

**	Potencia	5 ** 2 // 25
----	----------	--------------

- **Operadores de asignación:**

Operador	Descripción	Ejemplo
=	Asignación básica	X = 5
+=	Asignación con suma	X+5 (x=x+5)
-=	Asignación con resta	x -= 5 (x = x - 5)
*=	Asignación con multiplicación	X *= 5 (x = x * 5)
/=	Asignación con división	X /= (x = x/5)
%=	Asignación con módulo	X %= 5 (x = x%5)

- **Operadores de comparación:**

Operador	Descripción	Ejemplo
==	Igualdad	5 == "5" //true
===	Igualdad estricta	5 === "5" //false
!=	Desigualdad	5 != "5" //false
!==	Desigualdad estricta	5 !== "5" //true
>	Mayor que	5 > 2 //true
<	Menor que	5 < 2 //false
>=	Mayor o igual que	5 >= 5 //true
<=	Menor o igual que	5 <= 5 //true

- **Operadores de incremento y decremento:**

Operador	Descripción	Ejemplo
++	Incremento a 1	++x o x++
--	Decremento a 1	--x o x--

```
// Pre-incremento
let a = 5;
let b = ++a; // Primero incrementa 'a' a 6, luego asigna a 'b'
console.log("Pre-incremento: a =", a, "b =", b); // a = 6, b = 6

// Post-incremento
let c = 5;
let d = c++; // Primero asigna el valor de 'c' (5) a 'd', luego incrementa 'c' a 6
console.log("Post-incremento: c =", c, "d =", d); // c = 6, d = 5

// Pre-decremento
let e = 5;
let f = --e; // Primero decrementa 'e' a 4, luego asigna a 'f'
console.log("Pre-decremento: e =", e, "f =", f); // e = 4, f = 4

// Post-decremento
let g = 5;
let h = g--; // Primero asigna el valor de 'g' (5) a 'h', luego decrementa 'g' a 4
console.log("Post-decremento: g =", g, "h =", h); // g = 4, h = 5
```

6. Estructuras de control:

- Estructuras condicionales (if, else if, else):

```
if (condición) {
    // Código a ejecutar si la condición es verdadera
} else if (otraCondición) {
    // Código a ejecutar si la primera condición es falsa y la segunda es verdadera
} else {
    // Código a ejecutar si todas las condiciones anteriores son falsas
}
```

- Switch:

```
switch (expresión) {
    case valor1:
        // Código a ejecutar si expresión === valor1
        break;
    case valor2:
        // Código a ejecutar si expresión === valor2
        break;
    default:
        // Código a ejecutar si no coincide ningún valor
}
```

- Bucle for:

```
for (inicialización; condición; incremento) {
    // Código a ejecutar en cada iteración
}
```

- Bucle While:

```
while (condición) {
    // Código a ejecutar mientras la condición sea verdadera
}
```

- **Bucle do..while:**

```
do {  
    // Código a ejecutar  
} while (condición);
```

- **Break y continue:**

- **Break:** Termina el bucle inmediatamente.
- **Continue:** Salta a la siguiente interacción del bucle

```
for (let i = 0; i < 10; i++) {  
    if (i === 5) {  
        break; // Detiene el bucle cuando i es 5  
    }  
    if (i === 3) {  
        continue; // Salta la iteración cuando i es 3  
    }  
    console.log(i);  
}
```

- **Ternario:** (condición ? valorSiVerdadero : valorSiFalso)

```
let edad = 20;  
let mensaje = (edad >= 18) ? "Eres mayor de edad" : "Eres menor de edad";  
console.log(mensaje);
```