

Unidad 2

Introducción a la programación en JavaScript

CONTENIDOS

- 2.1. Conceptos básicos**
- 2.2. Variables**
- 2.3. Entrada y salida en el navegador**
- 2.4. Operadores**
- 2.5. Estructuras de control**

2.1. Conceptos básicos

Características de JavaScript

- Ligero.
- Interpretado.
- Basado en prototipos.
- Case sensitive.
- Débilmente tipado.
- Multiparadigma.
- Monohilo.
- Dinámico.
- Programación orientada a objetos.
- Programación imperativa.
- Programación declarativa.

Dónde ejecutar JavaScript

■ Lado cliente

Puede controlar y responder a los eventos del usuario: formularios, navegación entre páginas o cualquier otro elemento de la interfaz del usuario.

■ Lado servidor

Amplía el núcleo del lenguaje dándole la capacidad para: comunicarse con una base de datos, gestionar ficheros del servidor u ofrecer respuestas a solicitudes de aplicaciones.

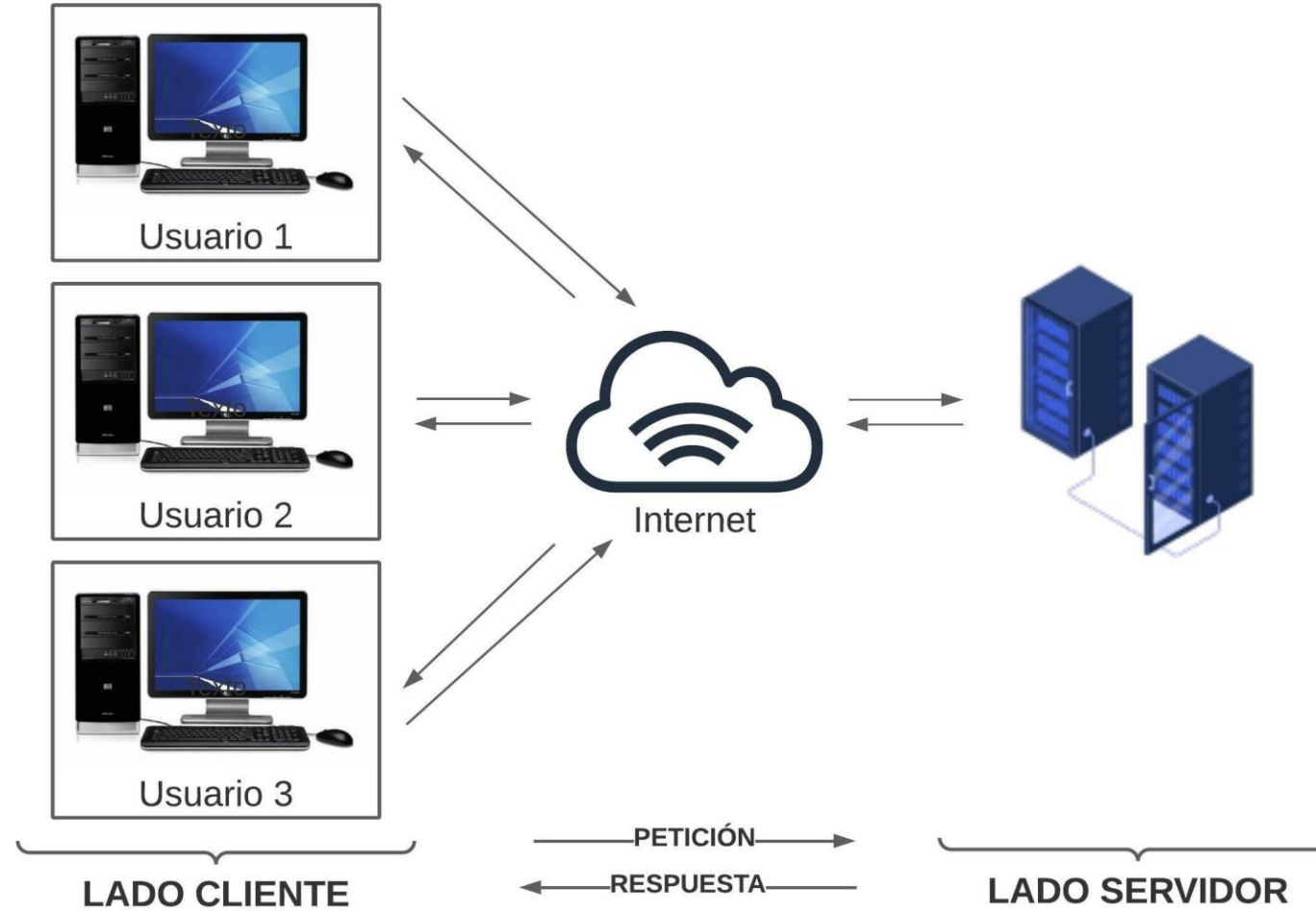


Figura 2.1. Esquema simplificado de una arquitectura cliente-servidor.


Cómo ejecutar JavaScript

■ Usando la etiqueta <script>




```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <title></title>
6 </head>
7 <body>
8     <script>
9         document.write("Código JS dentro de HTML");
10    </script>
11 </body>
12 </html>
```

■ Usando los atributos de un elemento



```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4     <meta charset="UTF-8">
5     <title>JS en los atributos</title>
6 </head>
7 <body>
8     <p onclick="alert('Un mensaje lanzado por Javascript')">
9         Esto es un párrafo con un evento asociado.
10    </p>
11 </body>
12 </html>
```

■ Incorporándolo desde un fichero .js externo



```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4     <meta charset="UTF-8">
5     <title>Incluyendo JS externo</title>
6 </head>
7 <body>
8     <p>
9         Un párrafo cualquiera.
10    </p>
11    <script src="micodigo.js"></script>
12 </body>
13 </html>
```


Sentencias

- Cada una de las instrucciones que forman parte de nuestro programa.
- Aunque a veces no es necesario hacerlo, siempre es recomendable terminarlas con un punto y coma (;).



```
1 let colores = {"rojo", "verde", "azul"};  
2 contador++;  
3 var primero = vector[0];
```

Bloques

- Un bloque es una agrupación de sentencias.
- Está delimitado por los símbolos { y }.




```
1 if (!cerrado) {  
2     abierto = 1;  
3     console.log("El candado está abierto");  
4 }
```

Identificadores

- Un identificador es el nombre que le damos a cada uno de los elementos que creamos en nuestro programa.
- Un identificador válido:
 - Debe comenzar obligatoriamente por una letra, guion bajo (_) o el símbolo del dólar (\$).
 - Pueden seguirle más letras, números o guiones bajos.
 - Distingue entre mayúsculas y minúsculas.
 - Puede usar todas las letras que están definidas en UNICODE.

Comentarios

- Son apuntes que dejamos en el código para mejorar su comprensión.
- Pueden ser de una línea (`// comentario`) o de varias líneas (`/* comentario */`).



```
1 // COMENTARIO DE UNA LÍNEA
2 // Controla el primer bucle (número de iteraciones)
3 var contador = 0;
4
5 // COMENTARIO DE VARIAS LÍNEAS
6 /*
7 Almacena la cantidad de elementos actuales de la estructura.
8 Solo se incluyen los positivos.
9 Debe coincidir con los dispositivos válidos.
10 */
11 let numElementos = 0;
```

2.2. 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.

Las hay de varios tipos.

Tipos de datos

- String: una secuencia de caracteres que representan texto.
- Number: cualquier formato de texto con o sin decimales.
- Booleano: dato lógico que solo puede ser true o false.
- Undefined: una variable a la que no se le ha dado valor.
- Null: representa el valor vacío o nulo.
- BigInt: se usa para almacenar números muy grandes.



```
1 // String
2 var cadena1 = 'Una cadena con comillas simples';
3 var cadena2 = "Una cadena con comillas dobles";
4 // Number
5 var alturaCm = 182;
6 var pesoKg = 71.6;
7 var diametroTransistor = 2e-9;
8 // Booleano
9 var encendido = false;
10 // Undefined
11 var sinInicializar;
12 // Null
13 var nulo = null;
14 // BigInt
15 var gigante = 90071992547409910000;
```

Operador typeof

- Lo usamos para conocer cuál es el tipo de dato de una variable en un momento dado.



```
1 console.log("cadena1: "+typeof(cadena1));  
2 console.log("cadena2: "+typeof(cadena2));  
3 console.log("alturaCm: "+typeof(alturaCm));  
4 console.log("pesoKg: "+typeof(pesoKg));  
5 console.log("diametroTransistor: "+typeof(diametroTransistor));  
6 console.log("encendido: "+typeof(encendido));  
7 console.log("sinInicializar: "+typeof(sinInicializar));  
8 console.log("nulo: "+typeof(nulo));  
9 console.log("gigante: "+typeof(gigante));
```


Conversión de tipos

- Algunas operaciones entre tipos de datos distintos son incompatibles por lo que el lenguaje realizará una conversión de tipos de la forma que entienda más lógica.
- Sin embargo podemos forzar a que realice una conversión manual de tipos:



```
1 var numero = 37;  
2 var cadena = "54";  
3 // Así convertimos manualmente la cadena de caracteres "54" en el número 54.  
4 var suma = numero + Number(cadena);
```

Tipos de variables

var

Es la forma tradicional de declarar una variable.

Su ámbito es global: es accesible desde cualquier parte del programa.

CUIDADO.

let

Su ámbito es más local que var.

No es accesible desde fuera del bloque en el que fue creada.

const

A pesar de ser un tipo de variable, representan a las constantes: una variable que una vez que toma valor, no puede ser modificada.

Ámbito de las variables



```
1 var variableGlobal = "He sido declarada fuera de todo bloque";
2 function unaFuncion() {
3     let variableLocal = "He sido declarada dentro de una función";
4     return 0;
5     // Aquí no tenemos acceso a variableGlobal ni a CONSTANTE
6 }
7 const CONSTANTE = 3.14;
8 variableGlobal = "He sido modificada";
9 // Aquí no tenemos acceso a variableLocal
```

2.3. Entrada y salida en el navegador

Mensajes en la consola

- Permite obtener mucha información de depuración de los programas sin «ensuciar» el aspecto de esos programas.
- Existen 4 tipos de mensajes que se muestran con una estética diferente.
- Se pueden utilizar los filtros que proporciona el navegador para ver sólo aquellos del tipo que se desee.

`console.log()`, `console.info()`, `console.warn()` y `console.error()`

Mensajes de confirmación

- Es un cuadro de diálogo que lanzamos con **confirm()** y un mensaje con dos opciones Aceptar (recibe true) o Cancelar (recibe false).



```
1 let mensaje = "¿Estás seguro de querer eliminar?";  
2 let respuesta = confirm(mensaje);  
3 console.log(`Respuesta del cuadro de diálogo: ${respuesta}.`);
```

Esta página dice

¿Estás seguro de querer eliminar?

Aceptar

Cancelar

Respuesta del cuadro de diálogo: true.



Mensajes de entrada

- Es igual que el anterior pero nos permite introducir un valor que lo retorna si pulsamos en Aceptar o null si pulsamos en Cancelar. Lo lanzamos con **prompt()**.

```
1 let mensaje = "Para eliminar escriba ELIMINAR";  
2 let respuesta = prompt(mensaje);  
3 console.log(`El usuario escribió: ${respuesta}.`);
```

Esta página dice

Para eliminar escriba ELIMINAR

El usuario escribió: ELIMINAR.



Mensajes de alerta

- Se trata simplemente de lanzar una pequeña ventana usando **alert()** donde se mostrará algún mensaje de interés para el usuario. Para cerrarlo pulsamos el botón Aceptar.



```
1 const PI = 3.14159;  
2 alert(`Recuerda usar esta aproximación de  $\pi$  en tus cálculos: ${PI}`);
```

Esta página dice

Recuerda usar esta aproximación de π en tus cálculos: 3.14159

Aceptar

2.4. Operadores

- Son símbolos que permiten manipular los operandos.
- Indica la operación que se va a realizar con los datos.
- La inmensa mayoría se evalúa de izquierda a derecha.
- Siguen un orden de precedencia preestablecido.
- Podemos utilizar los paréntesis para forzar operaciones que deben realizarse antes.
- Tipos de operadores: de asignación, de comparación, aritméticos, bit a bit, lógicos, de cadena, condicionales.

Operadores de asignación

- Asignan al operando izquierdo el valor del operando derecho por medio del símbolo **=**.

Tabla 2.1. Principales operadores de asignación.

Nombre	Aplicación	Simplificación
Asignación	a = b	a = b
Asignación de adición	a = a + b	a += b
Asignación de resta	a = a – b	a -= b
Asignación de multiplicación	a = a * b	a *= b
Asignación de división	a = a / b	a /= b
Asignación de módulo	a = a % b	a %= b
Asignación de exponenciación	a = a ** b	a **= b

Operadores de comparación

- Determinan si una comparación es verdadera (**true**) o falsa (**false**). Por tanto, siempre devuelven un booleano.

Tabla 2.2. Operadores de comparación

Operador	Significado	Resultado true
Igualdad (==)	Devuelve true si los operandos son iguales.	5 == 5
Distinto (!=)	Devuelve true si los operandos no son iguales.	5 != 9
Igualdad estricta (===)	true si son iguales y del mismo tipo.	5 === 5
Desigualdad estricta (!==)	true si son del mismo tipo pero no iguales, o son de diferente tipo.	5 !== "5"
Mayor que (>)	true si el izquierdo es mayor que el derecho.	9 > 5
Mayor o igual que (>=)	true si el izquierdo es mayor o igual que el derecho.	9 >= 9
Menor que (<)	true si el izquierdo es menor que el derecho.	3 < 9
Menor o igual que (<=)	true si el izquierdo es menor o igual que el derecho.	9 <= 9

Operadores aritméticos

- Toma los valores numéricos de los operandos y les aplica la operación aritmética que indique el operador.

Tabla 2.3. Operadores aritméticos

Operador	Significado	Ejemplo
Módulo (%)	Devuelve el resto de la división entera.	9 % 5
Incremento (++)	Operador unario que suma uno a su operando. Preincremento (++a): suma 1 a la a y devuelve el valor Posincremento (a++): devuelve el valor y suma 1 a la a.	b = ++a; b = a++;
Decremento (--)	Operador unario que resta uno a su operando. Predecremento (--a): resta 1 de a y devuelve su valor. Posdecremento (a--): devuelve su valor y resta 1 de a	b = --a; b = a--;
Potencia (**)	Elevando el operando izquierdo al operando derecho.	2 ** 3
Negativo unario (-)	Devuelve la negación de su operando.	Si a es 5, -5
Positivo unario (+)	Intenta convertir el operando en un número, si no lo es.	+"3" devuelve 3

Operadores bit a bit

- Tratan a sus operandos como una ristra de 32 bits.
- Operan en binario bit a bit, en lugar de hacerlo en decimal, octal o hexadecimal.
- Una vez terminada la operación, el intérprete devolverá el resultado nuevamente convertido a decimal.
- Son unos operadores que se utilizan en muy contadas ocasiones.
- Son: AND, OR, XOR, NOT y desplazamientos.

Operadores lógicos

- Evalúan condiciones para tomar decisiones a lo largo del flujo de ejecución del programa.

Tabla 2.4. Operadores lógicos

Operador	Uso	Descripción
AND lógico (&&)	expr1 && expr2	Devuelve expr1 si se puede convertir a false , de lo contrario devuelve expr2 . Por tanto, cuando se usa con valores booleanos, devuelve true si ambos operandos son true y false en caso contrario.
OR lógico ()	expr1 expr2	Devuelve expr1 si se puede convertir a true , de lo contrario devuelve expr2 . Por tanto, cuando se usa con valores booleanos, devuelve true si alguno de los operandos es true o false si ambos son false .
NOT lógico (!)	!expr	Devuelve false si su único operando se puede convertir a true ; en caso contrario devuelve true .

Operadores lógicos

- Para tener una comprensión completa de qué valor se puede esperar al evaluar expresiones donde intervienen los operadores lógicos veamos sus tablas de verdad donde un 1 representa **true** y un 0 representa **false**:

Tabla 2.5. Tablas de verdad de los operadores lógicos

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a b
0	0	0
0	1	1
1	0	1
1	1	1

a	!a
0	1
1	0

Operadores de cadena

- Podemos utilizar el símbolo `+` para concatenar cadenas:



```
1 console.log("Así" + " concatenamos " + 4 + " cadenas");  
2 // Salida: "Así concatenamos 4 cadenas"  
3  
4 let cadena = "nombre";  
5 cadena += " y apellidos";  
6 console.log(cadena);  
7 // Salida: "nombre y apellidos"
```

Operador condicional

- Se le suele llamar operador ternario, puesto que es el único en el que intervienen tres operandos.
- La sintaxis es: **condición ? valor1 : valor2**.
- Si la condición se evalúa a true, la operación devuelve valor1; en caso contrario devuelve valor2.



```
1 let edad = 21;  
2 let mensaje = (edad >= 18) ? "mayor" : "menor";  
3 console.log(`El usuario es ${mensaje} de edad.`);  
4 // Salida: "El usuario es mayor de edad."
```


2.5. Estructuras de control

- Son instrucciones que nos permiten alterar el flujo de ejecución secuencial de nuestros programas.
- Podemos bifurcar para tomar caminos distintos en función de la evaluación de condiciones de nuestro programa, y también realizar tareas repetitivas o bucles mientras se cumpla una determinada condición.
- Las más importantes son:
 - **Selectivas:** *if, switch.*
 - **Repetitivas:** *while, do-while, for.*
 - **De salto:** *break, continue, labeled.*

Estructura selectiva if

- Si se cumple la condición se ejecutan las instrucciones que contiene, y si no, se ejecutan las del bloque **else**.
- Podemos encadenar tantas estructuras **if-else if-else** que consideremos.
- Cada uno de sus bloques deben ser mutuamente excluyentes.



```
1 console.log("Inicio");
2 let local = 2;
3 let visitante = 1;
4 if (local > visitante) {
5     console.log("Local gana.");
6 }
7 else if (local < visitante) {
8     console.log("Visitante gana.");
9 }
10 else {
11     console.log("¡Hay empate!");
12 }
13 console.log("Fin");
```

Estructura selectiva switch

- Cobra sentido cuando tras evaluar una expresión cuyo resultado puede ser variado, es preciso abrir múltiples vías distintas de ejecución.
- Tras evaluar la condición se ejecutan las instrucciones del bloque **case** cuyo valor coincida.
- Si no coincide ninguna, se ejecuta la sección **default**.
- Si usamos **break**, ahí terminará la ejecución de la estructura. Si no, se ejecuta todo lo que hay desde el primer case coincidente hasta el final.

```
1 console.log("Menú abierto");
2 let letra_pulsada = 'c';
3 switch (letra_pulsada) {
4     case 'a':
5     case 'A':
6         console.log("Abrir archivo");
7         break;
8     case 'c':
9     case 'C':
10        console.log("Copiar");
11        break;
12    case 'p':
13    case 'P':
14        console.log("Pegar");
15        break;
16    default:
17        console.log("Opción incorrecta");
18 }
19 console.log("Menú cerrado");
```

Estructura repetitiva while

- Ejecuta las instrucciones que formen parte de su interior mientras que su condición se evalúe a **true**.
- Solo terminará cuando la condición resulte **false**.
- El bloque se ejecutará 0, 1 o más de una vez.
- Cuidado con los bucles infinitos, ya que provocará un consumo de recursos desproporcionado, puede colgar el navegador e incluso la propia máquina en la que se ejecuta.



```
1 let pases = 0;
2 while (pases < 10) {
3     console.log(`Pase número ${pases+1}`);
4     pases++;
5 }
```

Estructura repetitiva for

- Se ejecuta una y otra vez hasta que la condición (2.º parámetro) se evalúe a **false**.
- Dispone de una variable de control que definimos en el 1.º parámetro y que avanza según la pauta que indiquemos en el 2.º parámetro.



```
1 // Tabla de multiplicar
2 const TABLA = 9;
3 for (let contador=1; contador<=10; contador++) {
4     console.log(`${TABLA} x ${contador} = ${TABLA*contador}`);
5 }
```

Estructura de salto break

- Se utiliza para finalizar un bucle y transferir el control a la siguiente instrucción.



```
1 // Tabla de multiplicar hasta la mitad
2 const TABLA = 9;
3 for (let contador=1; contador<=10; contador++) {
4     console.log(`${TABLA} x ${contador} = ${TABLA*contador}`);
5     if (contador == 5)
6         break;
7 }
```


Instrucción de salto continue

- Fuerza a la finalización de la iteración actual de un bucle, y continúa la ejecución del bucle con la siguiente iteración.
- En el caso del for, se ejecuta también el tercer parámetro (la actualización de la variable de control).

```
1 console.log("Primeros 10 impares NO múltiplo de 3");
2 let contador = 0;
3 let numero = 1;
4 while (contador < 10) {
5     if (numero%3 == 0) {
6         numero++;
7         continue;
8     }
9     if (numero%2 != 0) {
10        console.log(`IMPAR: ${numero}`);
11        contador++;
12    }
13    numero++;
14 }
```

Instrucción de salto labeled

- Complementa al **break** y **continue**.
- Establece puntos en el programa, a los que se le asigna un nombre (etiqueta) al que hacer referencia cuando se desea efectuar un salto.
- Debe utilizarse en casos muy excepcionales, puesto que abusar de ellos va en contra de las normas de estilo.



```
1 let primero = segundo = 1;
2 buclePrincipal: while (true) {
3     console.log(`(Bucle 1) Iteración ${primero}`);
4     primero++;
5     while (true) {
6         console.log(`(Bucle 2) Iteración ${segundo}`);
7         segundo++;
8         if (segundo == 5)
9             break;
10        else if (primero == 3)
11            break buclePrincipal;
12    }
13 }
```