



# JavaScript

## Capítulo 1: Programación Básica

### Variables

#### Declaración de Variables

- `var`: Evitar su uso; tiene alcance global o de función.
- `let`: Para variables que pueden cambiar; alcance de bloque.
- `const`: Para valores constantes; no pueden reasignarse; alcance de bloque.

```
let edad = 25;  
const PI = 3.1416;
```

#### Reglas de Nomenclatura

- Deben comenzar con una letra, `$` o `_`.
- No pueden ser palabras reservadas.
- Sensibles a mayúsculas y minúsculas (`nombre`  $\neq$  `Nombre`).

### Tipos de Datos

#### Primitivos

- **Número**: Enteros y decimales.

```
let entero = 10;  
let decimal = 3.14;
```

- **Cadena de Texto** (`string`): Texto entre comillas simples o dobles.

```
let saludo = "Hola Mundo";
```

- **Caracteres Especiales:**

- `\n`: Nueva línea
- `\t`: Tabulador
- `\'`: Comilla simple
- `\"`: Comilla doble
- `\\`: Barra invertida

- **Booleano:** `true` o `false`.

```
let esVisible = true;
```

- **Undefined:** Variable declarada pero sin asignar.
- **Null:** Intencionalmente sin valor.

## Complejos

- **Array:** Lista ordenada de valores.

```
let frutas = ["Manzana", "Banana", "Cereza"];
```

- **Objeto:** Colección de pares clave-valor.

```
let persona = {  
  nombre: "Juan",  
  edad: 30,  
};
```

## Operadores

### Aritméticos

- **Suma (+)**
- **Resta (-)**
- **Multiplicación (\*)**
- **División (/)**
- **Módulo (%):** Resto de la división.

```
let resto = 17 % 5; // resto = 2
```

## Asignación

- `=`: Asignación simple.
- `+=`, `=`, `=`, `/=`, `%=`: Operadores de asignación compuesta.

```
let x = 10;  
x += 5; // x = x + 5
```

## Comparación

- **Igualdad Débil (`==`)**: Compara valor, realiza coerción de tipo.
- **Igualdad Estricta (`===`)**: Compara valor y tipo.
- **Desigualdad (`!=`, `!==`)**
- **Mayor, Menor, Mayor o Igual, Menor o Igual (`>`, `<`, `>=`, `<=`)**

## Lógicos

- **AND (`&&`)**: Verdadero si ambas expresiones son verdaderas.
- **OR (`||`)**: Verdadero si al menos una expresión es verdadera.
- **NOT (`!`)**: Invierte el valor lógico.

```
let esAdulto = true;  
if (!esAdulto) {  
    // Código si no es adulto  
}
```

## Incremento y Decremento

- **Incremento (`++`)**
- **Decremento (`-`)**

```
let contador = 0;  
contador++; // contador = 1
```

# Estructuras de Control de Flujo

## Condicionales

- `if`

```
if (condicion) {  
    // Código si la condición es verdadera  
}
```

- `if...else`

```
if (condicion) {  
    // Código si verdadera  
} else {  
    // Código si falsa  
}
```

- `if...else if...else`

```
if (condicion1) {  
    // Código  
} else if (condicion2) {  
    // Código  
} else {  
    // Código  
}
```

## Bucles

- `for`

```
for (let i = 0; i < 10; i++) {  
    // Código que se repite 10 veces  
}
```

- `while`

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

- **do...while**

```
do {  
    // Código que se ejecuta al menos una vez  
} while (condicion);
```

- **for...of** (para arrays)

```
for (let fruta of frutas) {  
    console.log(fruta);  
}
```

- **for...in** (para objetos)

```
for (let clave in persona) {  
    console.log(clave, persona[clave]);  
}
```

## Funciones y Propiedades Básicas

### Conversión de Tipos

- **Convertir String a Número**

```
let cadena = "42";  
let numero = Number(cadena); // numero = 42
```

- **Convertir Número a String**

```
let num = 123;  
let texto = String(num); // texto = "123"
```

- **Funciones Alternativas**

- `parseInt(cadena)` : Convierte cadena a número entero.

```
let entero = parseInt("123"); // entero = 123
```

- `parseFloat(cadena)` : Convierte cadena a número decimal.

```
let decimal = parseFloat("3.14"); // decimal = 3.14
```

## Funciones para Cadenas de Texto

- `length` : Longitud de la cadena.

```
let longitud = saludo.length;
```

- `toUpperCase()` , `toLowerCase()` : Convertir a mayúsculas o minúsculas.
- `charAt(indice)` : Carácter en la posición indicada.
- `indexOf(cadena)` : Primera posición de la subcadena.
- `lastIndexOf(cadena)` : Última posición de la subcadena.
- `substring(inicio, fin)` : Extrae una subcadena.
- `split(separador)` : Convierte la cadena en un array.

```
let palabras = saludo.split(" ");
```

## Funciones para Arrays

- `length` : Número de elementos.
- `push(elemento)` : Añade al final.
- `pop()` : Elimina el último elemento.
- `shift()` : Elimina el primer elemento.
- `unshift(elemento)` : Añade al inicio.
- `splice(indice, cantidad)` : Elimina o reemplaza elementos.
- `slice(inicio, fin)` : Extrae una sección.
- `join(separador)` : Une los elementos en una cadena.

- `reverse()` : Invierte el orden.
- `sort()` : Ordena los elementos.

## Funciones para Números

- `Math.random()` : Genera un número aleatorio entre 0 (inclusive) y 1 (exclusivo).

```
let aleatorio = Math.random(); // Ejemplo: 0.823432
```

- Para que nos genere mas números y sin comas utilizaremos la función

`Math.floor()`

```
let aleatorio_sin_coma = Math.floor(Math.random() * 10)
//numero del 1 al 10 gracias al +1 sin el seria del 0 a
```

- `toFixed(decimales)` : Formatea con decimales.

```
let num = 3.1416;
console.log(num.toFixed(2)); // "3.14"
```

- **Conversión de Número a String**

```
let numero = 256;
let cadena = numero.toString(); // cadena = "256"
```

## Función `eval()`

- `eval(cadena)` : Evalúa una cadena de texto como código JavaScript.

```
let expresion = "2 + 2";
let resultado = eval(expresion); // resultado = 4
```

Nota: El uso de `eval()` es generalmente desaconsejado por razones de seguridad y rendimiento. Úsalo con precaución.

# Capítulo 2: Programación Avanzada

## Funciones

### Declaración de Funciones

- **Función Declarada**

```
function nombreFuncion(parametros) {  
  // Código  
  return resultado;  
}
```

- **Función Expresada (Anónima)**

```
let sumar = function (a, b) {  
  return a + b;  
};
```

- **Funciones Flecha (ES6+)**

```
let restar = (a, b) => a - b;
```

### Parámetros y Argumentos

- **Parámetros:** Variables en la definición.
- **Argumentos:** Valores al invocar la función.

```
function saludar(nombre) {  
  return "Hola, " + nombre;  
}  
saludar("Ana"); // "Hola, Ana"
```

### Valores de Retorno

- **return**: Finaliza la función y devuelve un valor.

```
function multiplicar(a, b) {  
  return a * b;  
}
```



```
}
```

## Ámbito de Variables (Scope)

- **Local:** Dentro de una función o bloque.
- **Global:** Fuera de funciones y bloques.

```
let global = "Variable global";

function ejemplo() {
  let local = "Variable local";
  console.log(global); // Accede a variable global
}

console.log(local); // Error: local no está definida
```

## Sentencias `break` y `continue`

- `break`: Sale del bucle actual.

```
for (let i = 0; i < 10; i++) {
  if (i === 5) break;
  console.log(i); // 0 a 4
}
```

- `continue`: Salta a la siguiente iteración.

```
for (let i = 0; i < 10; i++) {
  if (i % 2 === 0) continue;
  console.log(i); // Números impares
}
```

## Estructuras de Control Adicionales

### `switch`

- Estructura condicional múltiple.

```
switch (expresion) {
  case valor1:
    // Código
    break;
  case valor2:
    // Código
    break;
  default:
    // Código por defecto
}
```

## Operador Ternario

- Sintaxis concisa para condiciones simples.

```
let acceso = edad >= 18 ? "Permitido" : "Denegado";
```

## Temporizadores

### setTimeout()

- Ejecuta una función después de un retraso específico (en milisegundos).

```
setTimeout(functionLlamada, tiempo);
```

#### ◦ Ejemplo:

```
function mostrarMensaje() {
  alert("¡Hola después de 2 segundos!");
}

setTimeout("mostrarMensaje()", 2000);
```

Nota: Es preferible pasar la función directamente sin comillas:

```
setTimeout(mostrarMensaje, 2000);
```

## Capítulo 3: Document Object Model (DOM)

### Introducción al DOM

- **Definición:** Representación en forma de árbol de todos los elementos de una página web.
- **Nodos:**
  - **Document:** Nodo raíz que representa todo el documento.
  - **Element:** Cada etiqueta HTML.
  - **Text:** El contenido de texto dentro de un elemento.
  - **Attribute:** Atributos de los elementos.
  - **Comment:** Comentarios en el código HTML.

### Selección de Elementos

#### Métodos Clásicos

- `document.getElementById(id)`

```
let titulo = document.getElementById("tituloPrincipal");
```

- `document.getElementsByTagName(etiqueta)`

```
let parrafos = document.getElementsByTagName("p");
```

- `document.getElementsByClassName(clase)`

```
let elementosDestacados = document.getElementsByClassName("destacado");
```

- `document.getElementsByName(name)`

```
let camposFormulario = document.getElementsByName("usuar
```

```
io");
```

## Métodos Modernos

- `document.querySelector(selector)`

- Selecciona el primer elemento que coincide con el selector CSS.

```
let primerElemento = document.querySelector(".clase .  
subclase");
```

- `document.querySelectorAll(selector)`

- Selecciona todos los elementos que coinciden.

```
let todosLosEnlaces = document.querySelectorAll("a");
```

## Manipulación del DOM

### Creación de Elementos

- **Crear un nuevo elemento**

```
let nuevoDiv = document.createElement("div");
```

- **Crear un nodo de texto**

```
let texto = document.createTextNode("Hola Mundo");
```

- **Añadir contenido al elemento**

```
nuevoDiv.appendChild(texto);
```

### Inserción en el DOM

- **Añadir al final del cuerpo**

```
document.body.appendChild(nuevoDiv);
```

- **Insertar antes de un elemento existente usando `insertBefore()`**

```
let referencia = document.getElementById("referencia");
let padre = referencia.parentNode;
padre.insertBefore(nuevoDiv, referencia);
```

- **Explicación:** `insertBefore()` inserta un nodo antes de otro nodo hijo específico. Se aplica al nodo padre.

```
padre.insertBefore(nuevoNodo, nodoReferencia);
```

- **Insertar en una posición específica**

```
let lista = document.getElementById("miLista");
let nuevoElemento = document.createElement("li");
nuevoElemento.textContent = "Nuevo ítem";

let primerItem = lista.firstChild;
lista.insertBefore(nuevoElemento, primerItem); // Insert
a al inicio de la lista
```

## Clonación de Elementos usando `cloneNode()`

- **Clonar un nodo existente**

```
let elementoOriginal = document.getElementById("elemento
Original");
let copiaElemento = elementoOriginal.cloneNode(true);
```

- **Parámetro `true`:** Clona el nodo y todos sus descendientes (clonación profunda).
- **Parámetro `false`:** Clona solo el nodo sin sus hijos (clonación superficial).

- **Insertar el nodo clonado en el DOM**

```
document.body.appendChild(copiaElemento);
```

- **Ejemplo Práctico**

```
// Clonar un elemento de lista y añadirlo al final
let itemOriginal = document.querySelector(".lista .item");
let copiaItem = itemOriginal.cloneNode(true);

let lista = document.querySelector(".lista");
lista.appendChild(copiaItem);
```

## Reemplazo de Elementos usando `replaceChild()`

- Reemplazar un elemento por otro

```
let nuevoElemento = document.createElement("section");
nuevoElemento.textContent = "Contenido nuevo";

let elementoViejo = document.getElementById("viejo");
let padre = elementoViejo.parentNode;
padre.replaceChild(nuevoElemento, elementoViejo);
```

- **Explicación:** `replaceChild()` reemplaza un nodo hijo por otro. Se aplica al nodo padre.

```
padre.replaceChild(nuevoNodo, nodoExistente);
```

## Eliminación de Elementos

- Eliminar un elemento existente

```
let elementoAEliminar = document.getElementById("elemento");
elementoAEliminar.parentNode.removeChild(elementoAEliminar);
```

## Atributos y Propiedades

### Manipulación de Atributos

- Obtener un atributo

```
let enlace = document.getElementById("enlace");  
let url = enlace.getAttribute("href");
```

- **Establecer un atributo**

```
enlace.setAttribute("href", "https://nueva-url.com");
```

- **Eliminar un atributo**

```
enlace.removeAttribute("target");
```

## Manipulación de Clases

- **Añadir una clase**

```
elemento.classList.add("nueva-clase");
```

- **Eliminar una clase**

```
elemento.classList.remove("clase-antigua");
```

- **Comprobar si tiene una clase**

```
elemento.classList.contains("activo");
```

- **Alternar una clase**

```
elemento.classList.toggle("visible");
```

## Estilos en Línea

- **Modificar estilos directamente**

```
elemento.style.color = "red";  
elemento.style.backgroundColor = "#f0f0f0";
```

Nota: Las propiedades CSS con guiones se convierten a camelCase en JavaScript.

## Trabajando con Contenido de Texto

- **Modificar el contenido de texto de un elemento**

```
elemento.textContent = "Nuevo contenido de texto";
```

- **Obtener el contenido de texto**

```
let texto = elemento.textContent;
```

## Trabajando con Formularios

- **Obtener el valor de un input**

```
let input = document.getElementById("miInput");  
let valor = input.value;
```

- **Establecer el valor de un input**

```
input.value = "Nuevo valor";
```

# Capítulo 4: Eventos

## Conceptos Básicos de Eventos

- **Eventos:** Acciones que ocurren en la página (clics, teclas presionadas, carga de la página, etc.).
- **Programación Basada en Eventos:** El código se ejecuta en respuesta a acciones del usuario o del navegador.
- **Nomenclatura de Eventos:** Los nombres de los eventos comienzan con el prefijo `on` seguido del nombre en inglés de la acción asociada. Por ejemplo, `onclick`, `onmouseover`.

## Tipos de Eventos



En JavaScript, cada elemento HTML puede tener asociados diferentes tipos de eventos. Un mismo evento puede aplicarse a múltiples elementos, y un elemento puede tener múltiples eventos.

## Eventos Comunes

Evento	Descripción	Elementos Aplicables	Ejemplo
<code>onblur</code>	Cuando un elemento pierde el foco	<code>&lt;input&gt;</code> , <code>&lt;select&gt;</code> , <code>&lt;textarea&gt;</code> , <code>&lt;button&gt;</code>	<code>elemento.onblur = funcion;</code>
<code>onchange</code>	Cuando el valor de un elemento cambia	<code>&lt;input&gt;</code> , <code>&lt;select&gt;</code> , <code>&lt;textarea&gt;</code>	<code>elemento.onchange = funcion;</code>
<code>onclick</code>	Cuando se hace clic en un elemento	Todos los elementos	<code>elemento.onclick = funcion;</code>
<code>ondblclick</code>	Doble clic sobre un elemento	Todos los elementos	<code>elemento.ondblclick = funcion;</code>
<code>onfocus</code>	Cuando un elemento recibe el foco	<code>&lt;input&gt;</code> , <code>&lt;select&gt;</code> , <code>&lt;textarea&gt;</code> , <code>&lt;button&gt;</code>	<code>elemento.onfocus = funcion;</code>
<code>onkeydown</code>	Al presionar una tecla (sin soltar)	Elementos de formulario y <code>&lt;body&gt;</code>	<code>elemento.onkeydown = funcion;</code>
<code>onkeypress</code>	Al presionar una tecla	Elementos de formulario y <code>&lt;body&gt;</code>	<code>elemento.onkeypress = funcion;</code>
<code>onkeyup</code>	Al soltar una tecla	Elementos de formulario y <code>&lt;body&gt;</code>	<code>elemento.onkeyup = funcion;</code>
<code>onload</code>	Cuando la página ha cargado completamente	<code>&lt;body&gt;</code>	<code>window.onload = funcion;</code>
<code>onmousedown</code>	Al presionar un botón del ratón	Todos los elementos	<code>elemento.onmousedown = funcion;</code>
<code>onmousemove</code>	Al mover el ratón sobre un elemento	Todos los elementos	<code>elemento.onmousemove = funcion;</code>

<code>onmouseout</code>	Cuando el ratón sale de un elemento	Todos los elementos	<code>elemento.onmouseout = funcion;</code>
<code>onmouseover</code>	Cuando el ratón entra en un elemento	Todos los elementos	<code>elemento.onmouseover = funcion;</code>
<code>onmouseup</code>	Al soltar un botón del ratón	Todos los elementos	<code>elemento.onmouseup = funcion;</code>
<code>onreset</code>	Al reiniciar un formulario	<code>&lt;form&gt;</code>	<code>formulario.onreset = funcion;</code>
<code>onresize</code>	Al cambiar el tamaño de la ventana	<code>&lt;body&gt;</code>	<code>window.onresize = funcion;</code>
<code>onselect</code>	Cuando se selecciona un texto	<code>&lt;input&gt;</code> , <code>&lt;textarea&gt;</code>	<code>elemento.onselect = funcion;</code>
<code>onsubmit</code>	Al enviar un formulario	<code>&lt;form&gt;</code>	<code>&lt;form onsubmit="return funcion();"&gt;</code>
<code>onunload</code>	Al abandonar la página	<code>&lt;body&gt;</code>	<code>window.onunload = funcion;</code>

## Manejadores de Eventos

### Asignación Mediante HTML (No Recomendado)

```
<button onclick="alert('Hola')">Haz clic</button>
```

### Asignación Mediante Propiedades DOM

```
elemento.onclick = funcion;
```

- **Ejemplo:**

```
let boton = document.getElementById("miBoton");
boton.onclick = function () {
    alert("¡Botón presionado!");
};
```

### Uso de `addEventListener` (Recomendado)

- **Sintaxis**

```
elemento.addEventListener("evento", funcion, usoCaptura);
```

- **evento**: Tipo de evento (e.g., "click", "mouseover").
- **funcion**: Función que maneja el evento.
- **usoCaptura**: Booleano que indica la fase de propagación (por defecto, **false** para burbuja).

- **Ejemplo**

```
boton.addEventListener("click", funcion);
```

## Obteniendo Información del Evento

Para acceder al objeto **event** y obtener información sobre el evento ocurrido, se puede definir una función manejadora que reciba el evento como parámetro.

```
function manejadorEventos(elEvento) {  
    var evento = elEvento || window.event;  
    // Código que utiliza el objeto evento  
}
```

## Ejemplo Práctico

```
function resalta(elEvento) {  
    var evento = elEvento || window.event;  
    switch (evento.type) {  
        case "mouseover":  
            this.style.borderColor = "black";  
            break;  
        case "mouseout":  
            this.style.borderColor = "silver";  
            break;  
    }  
}
```

```

window.onload = function () {
    var seccion = document.getElementById("seccion");
    seccion.onmouseover = resalta;
    seccion.onmouseout = resalta;
};

```

## HTML Asociado:

```

<div
    id="seccion"
    style="width:150px; height:60px; border:thin solid silver"
>
    Sección de contenidos...
</div>

```

- **Explicación:** La función `resalta` cambia el color del borde del elemento según el tipo de evento (`mouseover` o `mouseout`). Utiliza `this` para referirse al elemento que generó el evento.

## Objeto `event`

- **Acceso al Objeto `event`**

```

elemento.onclick = function (event) {
    // Código que utiliza el objeto event
};

```

## Propiedades Comunes

- `event.type`: Tipo de evento.
- `event.target`: Elemento que originó el evento.
- `event.currentTarget`: Elemento al que se le asignó el manejador.
- `event.preventDefault()`: Evita la acción por defecto del evento.
- `event.stopPropagation()`: Detiene la propagación del evento.

## Propagación y Delegación de Eventos

## Fases de Propagación

1. **Captura:** El evento se propaga desde el elemento más externo hasta el objetivo.
2. **Objetivo:** El evento alcanza el elemento objetivo.
3. **Burbuja:** El evento se propaga desde el objetivo hacia los elementos padres.

## Detener la Propagación

- `event.stopPropagation()`

```
elemento.addEventListener("click", function (event) {  
    event.stopPropagation();  
    // Código específico  
});
```

## Delegación de Eventos

- Asignar un evento a un elemento padre para manejar eventos en elementos hijos.

```
contenedor.addEventListener("click", function (event) {  
    if (event.target.matches(".boton")) {  
        // Acción para elementos con clase 'boton'  
    }  
});
```

## Prevención de Comportamientos por Defecto

- `event.preventDefault()`
  - Evita acciones como seguir un enlace o enviar un formulario.

```
enlace.addEventListener("click", function (event) {  
    event.preventDefault();  
    // Código alternativo  
});
```

# Notas Adicionales y Consejos

## Carga del Documento

### **window.onload**

- Se ejecuta cuando todos los recursos (imágenes, scripts, etc.) han cargado.

```
window.onload = function () {  
    // Código a ejecutar  
};
```

### **DOMContentLoaded**

- Se ejecuta cuando el DOM ha sido completamente cargado, sin esperar a recursos externos.

```
document.addEventListener("DOMContentLoaded", function  
( ) {  
    // Código a ejecutar  
});
```