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` ≠ `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

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

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

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

- `parseInt(cadena)`, `parseFloat(cadena)`: Convierte cadena a número entero o decimal.

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";
```

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.getElementsByClassNam
e("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 .ite
m");
let copiaItem = itemOriginal.cloneNode(true);

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

- **Explicación**: Se clona un elemento de lista existente y se añade la copia al final de la misma lista.

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.

Capítulo 4: Eventos

Conceptos Básicos de Eventos

- **Eventos**: Acciones que ocurren en la página (clics, teclas presionadas, carga de la página, etc.).
- **Manejadores de Eventos**: Funciones que se ejecutan en respuesta a un evento.

Manejadores de Eventos

Asignación Mediante HTML (No Recomendado)

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

Asignación Mediante Propiedades DOM

```
let boton = document.getElementById("miBoton");
boton.onclick = function () {
```

```
  alert("¡Botón presionado!");
};
```

Uso de `addEventListener` (Recomendado)

- **Sintaxis**

```
elemento.addEventListener("evento", funcion, usoCaptur
a);
```

  - `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", function () {
  alert("Evento escuchado");
});
```

El Objeto `this` en Eventos

- **Referencia al elemento que disparó el evento**

```
boton.addEventListener("click", function () {
  this.style.backgroundColor = "green";
});
```

Objeto `event`

- **Acceso al Objeto `event`**

```
boton.addEventListener("click", function (event) {
  console.log(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.

Eventos de Teclado

- `event.key`: Valor de la tecla.
- `event.code`: Código físico de la tecla.
- **Ejemplo: Detectar la tecla Enter**

```
document.addEventListener("keydown", function (event) {
  if (event.key === "Enter") {
    // Acción al presionar Enter
  }
});
```

Eventos de Ratón

- `event.clientX`, `event.clientY`: Posición del ratón en la ventana.
- `event.pageX`, `event.pageY`: Posición del ratón en la página.

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
});
```