UNIDAD 3
Utilización de los objetos predefinidos del lenguaje

Profesor: Santiago Fernández

Contenido

INTRODUCCIÓN

¡Bienvenidos al capítulo 3! En este capítulo, profundizaremos en la utilización de los objetos predefinidos de JavaScript. Los objetos son una parte fundamental de JavaScript y entender cómo utilizarlos te permitirá escribir código más eficiente y poderoso. Además, exploraremos cómo interactuar con el navegador, generar texto y elementos HTML dinámicamente, gestionar marcos, manejar la apariencia de la ventana, crear nuevas ventanas y utilizar cookies. Cada sección está diseñada para que puedas seguir y aprender de manera autodidacta, con ejemplos claros y explicaciones detalladas.

OBJETIVOS

- Comprender y utilizar los objetos nativos de JavaScript.
- Manipular el DOM con el objeto document.
- Controlar la apariencia y comportamiento de la ventana del navegador.
- Crear y gestionar marcos (iframes) en aplicaciones web.
- Abrir y comunicarse entre múltiples ventanas del navegador.
- Utilizar cookies para almacenar y gestionar datos del usuario.
- Interactuar con la URL y la navegación del navegador mediante el objeto location.
- Generar y manipular contenido textual y elementos HTML desde código.

3. OBJETOS

En JavaScript, casi todo es un objeto. Desde los simples arrays hasta las funciones y las fechas, todos se manejan mediante estructuras de objetos. Esto hace que entender los objetos sea fundamental para cualquier desarrollador que aspire a manipular y controlar efectivamente el entorno web.

3.1. ¿QUÉ ES UN OBJETO EN JAVASCRIPT?

En JavaScript, un objeto es una colección de propiedades y métodos. Las propiedades son variables que pertenecen al objeto y los métodos son funciones que pertenecen al objeto. Un objeto puede representar entidades del mundo real (como un coche o una persona) o estructuras más abstractas (como configuraciones o datos complejos).

3.2. CLAVE-VALOR

Cada propiedad de un objeto se define como un par de clave-valor. La clave (también llamada "nombre" o "key") es una cadena que identifica la propiedad, y el valor es el dato asociado a esa clave. Los valores pueden ser de cualquier tipo, incluyendo otros objetos.

Creación de Objetos Sintaxis Literal

La forma más común y sencilla de crear un objeto es usando la sintaxis literal:

```
let persona = {
    nombre: "Juan",
    edad: 30,
    saludar: function() {
        console.log("Hola, soy " + this.nombre);
    }
};
```

En este ejemplo:

- **nombre** y **edad** son claves.
- **"Juan"** y **30** son los valores asociados a las claves **nombre** y **edad**, respectivamente.
- **saludar** es un método que imprime un mensaje usando la propiedad **nombre**.

Acceso a Propiedades

Puedes acceder a las propiedades de un objeto utilizando la notación de punto o la notación de corchetes.

Notación de Punto:

```
console.log(persona.nombre);  // "Juan"
console.log(persona.edad);  // 30
persona.saludar();  // "Hola, soy Juan"
```

Notación de Corchetes:

```
console.log(persona["nombre"]);  // "Juan"
console.log(persona["edad"]);  // 30
```

Modificación de Propiedades

Puedes agregar, modificar o eliminar propiedades de un objeto después de su creación.

Agregar o Modificar

```
persona.apellido = "Pérez";
persona.edad = 31;

console.log(persona.apellido);  // "Pérez"
console.log(persona.edad);  // 31
```

Eliminar:

```
delete persona.apellido;
console.log(persona.apellido);  // undefined
```

Iteración sobre Propiedades

Para iterar sobre todas las propiedades de un objeto, puedes usar un bucle for...in:

```
for (let clave in persona) {
    console.log(clave + ": " + persona[clave]);
}
// Salida:
// nombre: Juan
// edad: 31
// saludar: function() { console.log("Hola, soy " + this.nombre); }
```

3.2.1. PRACTICA GUIADA OBJETOS

Vamos a crear un objeto coche y trabajar con sus propiedades y métodos:

```
let coche = {
    marca: "Toyota",
    modelo: "Corolla",
    año: 2020,
    arrancar: function() {
        console.log(this.marca + " " + this.modelo + " ha arrancado.");
    },
    mostrarInfo: function() {
        console.log(`Marca: ${this.marca}, Modelo: ${this.modelo}, Año:
${this.año}`);
    }
};

// Acceder a propiedades
console.log(coche.marca);  // "Toyota"
console.log(coche["modelo"]);  // "Corolla"

// Llamar a métodos
coche.arrancar();  // "Toyota Corolla ha arrancado."
coche.mostrarInfo();  // "Marca: Toyota, Modelo: Corolla, Año: 2020"
```

```
// Modificar propiedades
coche.año = 2021;
coche.color = "Rojo";

// Iterar sobre propiedades
for (let propiedad in coche) {
    console.log(propiedad + ": " + coche[propiedad]);
}
// Salida:
// marca: Toyota
// modelo: Corolla
// año: 2021
// arrancar: function() { console.log(this.marca + " " + this.modelo + "
ha arrancado."); }
// mostrarInfo: function() { console.log(`Marca: ${this.marca}, Modelo:
${this.modelo}, Año: ${this.año}`); }
// color: Rojo
```

3.3. CONSTRUCTORES EN JAVASCRIPT

En JavaScript, los constructores son una pieza fundamental para la creación de objetos y la implementación de la programación orientada a objetos (POO). Un constructor es una función especial que se utiliza para inicializar nuevos objetos. La convención en JavaScript es nombrar las funciones constructoras con la primera letra en mayúscula para diferenciarlas de otras funciones.

Imagina que quieres crear un objeto que represente a un animal. En JavaScript, podrías hacerlo de la siguiente manera:

```
function Animal(nombre, especie) {
    this.nombre = nombre;
    this.especie = especie;
  }

var miMascota = new Animal("Firulais", "Perro");
console.log(miMascota.nombre); // "Firulais"
console.log(miMascota.especie); // "Perro"
```

En este ejemplo, Animal es nuestro constructor. Cada vez que creamos un nuevo Animal, le damos un nombre y una especie, que son propiedades del objeto.

3.3.1. PROTOTIPOS Y CONSTRUCTORES

JavaScript utiliza prototipos para heredar propiedades y métodos a objetos. Cada función constructora tiene una propiedad prototype que es compartida por todas las instancias del objeto. Esto es útil para definir métodos que deben ser compartidos, lo que reduce la redundancia de código.

Ahora, ¿qué pasa si queremos que todos los animales puedan hacer algo, como emitir un sonido? Aquí es donde entra en juego el prototipo. Podemos agregar un método al prototipo de `Animal` para que todos los animales puedan usarlo:

```
Animal.prototype.hacerSonido = function(sonido) {
    console.log(this.nombre + " hace " + sonido + "!");
  };

  miMascota.hacerSonido("guau guau"); // "Firulais hace guau guau!"
```

Con este método hacerSonido, cualquier animal que creemos podrá hacer un sonido específico. Esto es muy útil porque no tenemos que escribir el mismo método una y otra vez para cada animal; simplemente lo agregamos una vez al prototipo, y todos los animales lo heredan.

3.3.2. CONSTRUCTORES PREDEFINIDOS

JavaScript también ofrece constructores predefinidos como Date, String, Number, y otros. Estos constructores permiten crear objetos de tipos de datos específicos con propiedades y métodos ya definidos.

```
var fechaActual = new Date();
console.log(fechaActual); // Muestra la fecha y hora actual
```

3.3.3. ACTIVIDAD: CREANDO UN CATÁLOGO DE LIBROS

Objetivo:

Crear un catálogo de libros utilizando constructores en JavaScript.

Instrucciones:

1. Define un constructor llamado Libro con las siguientes propiedades:
    - título: El título del libro.
    - autor: El autor del libro.
    - género: El género literario al que pertenece el libro.

3.4. ARRAYS EN PROFUNDIDAD

En la unidad anterior, introdujimos el concepto de arrays, una colección ordenada de elementos accesibles mediante un índice. En esta unidad, profundizaremos en su funcionamiento, métodos avanzados y aplicaciones prácticas en el desarrollo web.

3.4.1. ESTRUCTURA Y FUNCIONAMIENTO DE LOS ARRAYS

Un array en JavaScript es un tipo de objeto especializado para almacenar secuencias de valores. Cada valor es accesible mediante un índice que empieza desde 0. Esta estructura permite manejar colecciones de datos de forma eficiente.

```
let frutas = ["manzana", "banana", "cereza"];
console.log(frutas[0]); // Salida: manzana
console.log(frutas[2]); // Salida: cereza
```

Nota de Interés: Los arrays en JavaScript son dinámicos, lo que significa que su tamaño puede cambiar durante la ejecución del programa. Además, pueden almacenar elementos de diferentes tipos, incluyendo otros arrays, lo que los hace muy flexibles.

3.4.2. MÉTODOS AVANZADOS DE ARRAYS

Los arrays en JavaScript vienen con una amplia variedad de métodos incorporados que permiten manipularlos de múltiples formas. Aquí exploraremos algunos de los métodos más avanzados y sus aplicaciones prácticas.

Método	Explicación	Ejemplo

push()	Añade uno o más elementos al final del array.	let arr = [1, 2, 3];
		arr.push(4);
		console.log(arr); // [1, 2, 3, 4]

pop()	Elimina el último elemento del array y lo devuelve.	let arr = [1, 2, 3];
		let last = arr.pop();
		console.log(last); // 3 console.log(arr); // [1, 2]

shift()	Elimina el primer elemento del array y lo devuelve.	let arr = [1, 2, 3];
		let first = arr.shift();
		console.log(first); // 1 console.log(arr); // [2, 3]

unshift()	Añade uno o más elementos al inicio del array.	let arr = [1, 2, 3];
		arr.unshift(0);
		console.log(arr); // [0, 1, 2, 3]

concat()	Combina dos o más arrays y devuelve uno nuevo.	let arr1 = [1, 2];
		let arr2 = [3, 4];
		let arr3 = arr1.concat(arr2);
		console.log(arr3); // [1, 2, 3, 4]

join()	Une todos los elementos de un array en una cadena.	let arr = [1, 2, 3];
		let str = arr.join('-');
		console.log(str); // "1-2-3"

slice()	Devuelve una copia de una parte del array.	let arr = [1, 2, 3, 4];
		let subArr = arr.slice(1, 3);
		console.log(subArr); // [2, 3]

splice()	Añade/elimina elementos en un array.	let arr = [1, 2, 3, 4];
		arr.splice(1, 2, 'a', 'b');
		console.log(arr); // [1, 'a', 'b', 4]

forEach()	Ejecuta una función para cada elemento del array.	let arr = [1, 2, 3]; arr.forEach(num => console.log(num)); // Imprime: 1, 2, 3

map()	Crea un nuevo array con los resultados de la función.	let arr = [1, 2, 3];
		let squared = arr.map(num => num * num);
		console.log(squared); // [1, 4, 9]

filter()	Crea un nuevo array con los elementos que pasan el test.	let arr = [1, 2, 3, 4];
		let even = arr.filter(num => num % 2 === 0);
		console.log(even); // [2, 4]

reduce()	Aplica una función contra un acumulador y cada elemento.	let arr = [1, 2, 3];
		let sum = arr.reduce((acc, num) => acc + num, 0);
		console.log(sum); // 6

find()	Devuelve el primer elemento que pasa el test.	let arr = [1, 2, 3, 4];
		let found = arr.find(num => num > 2);
		console.log(found); // 3

findIndex()	Devuelve el índice del primer elemento que pasa el test.	let arr = [1, 2, 3, 4];
		let index = arr.findIndex(num => num > 2);
		console.log(index); // 2

includes()	Determina si el array contiene un valor.	let arr = [1, 2, 3];
		let hasTwo = arr.includes(2);
		console.log(hasTwo); // true

La complejidad algorítmica de los métodos de array puede influir significativamente en el rendimiento de tu aplicación. Por ejemplo, **map()**, **filter()** y **reduce()** recorren todo el array, lo que implica una complejidad O(n), donde n es el número de elementos en el array. Es importante considerar esto al trabajar con grandes conjuntos de datos.

3.4.3. RECORRIENDO ARRAYS

Para recorrer un array, puedes utilizar varios enfoques dependiendo de tus necesidades. Los métodos más comunes incluyen bucles **for**, **forEach**, y **for...of**.

Ejemplo con bucle for:

```
let numeros = [1, 2, 3, 4, 5];
for (let i = 0; i < numeros.length; i++) {
  console.log(numeros[i]);
}
```

Ejemplo con forEach:

```
numeros.forEach(function(numero) {
    console.log(numero);
  });
```

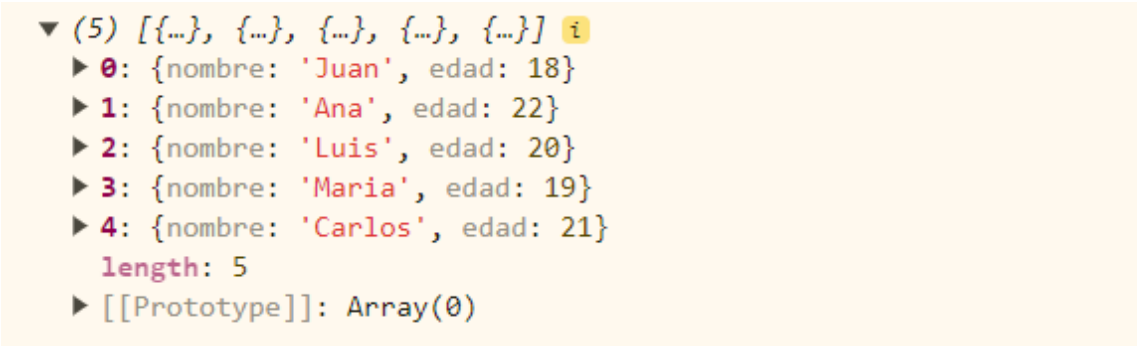
3.4.4. PRÁCTICA GUIADA: MANIPULACIÓN DE ARRAYS EN JAVASCRIPT

En esta práctica guiada, trabajaremos con arrays en JavaScript y utilizaremos varios métodos para manipularlos. Asegúrate de seguir cada paso y ejecutar el código en tu editor o consola para comprender mejor cómo funcionan estos métodos.

Paso 1: Creación de un Array

Primero, vamos a crear un array llamado **estudiantes** que contendrá objetos representando a cada estudiante con su nombre y edad.

```
let estudiantes = [
  { nombre: "Juan", edad: 18 },
  { nombre: "Ana", edad: 22 },
  { nombre: "Luis", edad: 20 },
  { nombre: "Maria", edad: 19 },
  { nombre: "Carlos", edad: 21 }
];
console.log(estudiantes);
```

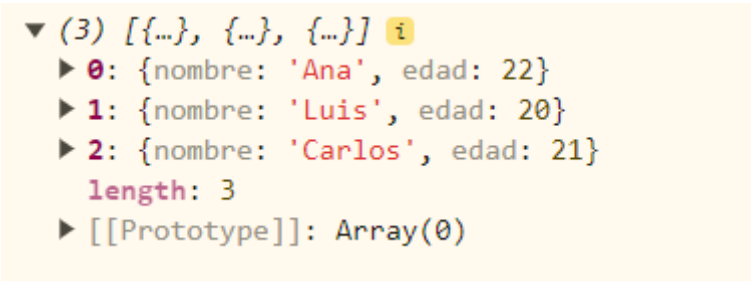


(Vista de el array por consola)

Paso 2: Filtrar Elementos en un Array

Ahora, filtraremos los estudiantes que tienen 20 años o más usando el método **filter()**.

```
  let mayoresDeEdad = estudiantes.filter(estudiante => estudiante.edad >= 20);
console.log(mayoresDeEdad);
```

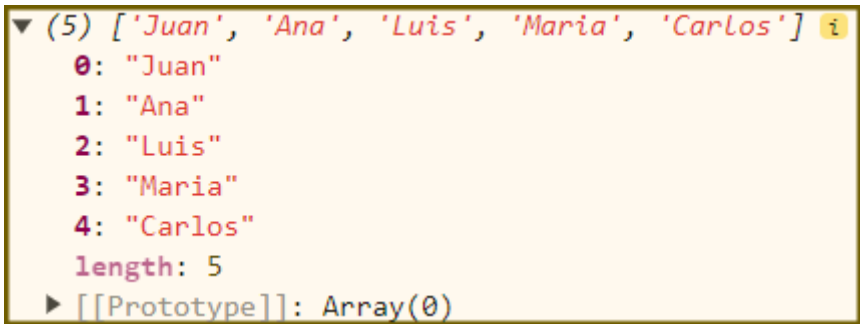


(Resultado esperado paso 2)

Paso 3: Mapear Elementos en un Array

Utilizaremos el método **map()** para crear un nuevo array que contenga solo los nombres de los estudiantes.

```
let nombresEstudiantes = estudiantes.map(estudiante => estudiante.nombre);
console.log(nombresEstudiantes);
```

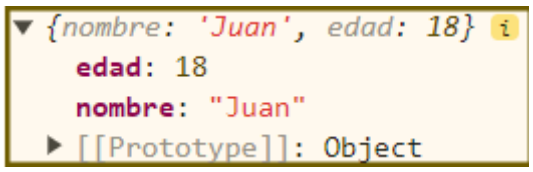


(Resultado esperado paso 3)

Paso 4: Encontrar un Elemento en un Array

Vamos a encontrar el primer estudiante que tenga 18 años utilizando el método **find()**.

```
  let estudiante18 = estudiantes.find(estudiante => estudiante.edad === 
18);
  console.log(estudiante18);
```



(Resultado esperado paso 4)

Paso 5: Reducir un Array a un Valor Único

Utilizaremos el método **reduce()** para calcular la edad promedio de los estudiantes.

```
  let totalEdad = estudiantes.reduce((acc, estudiante) => acc + 
estudiante.edad, 0);
let edadPromedio = totalEdad / estudiantes.length;
console.log(`Edad Promedio: ${edadPromedio}`);
```

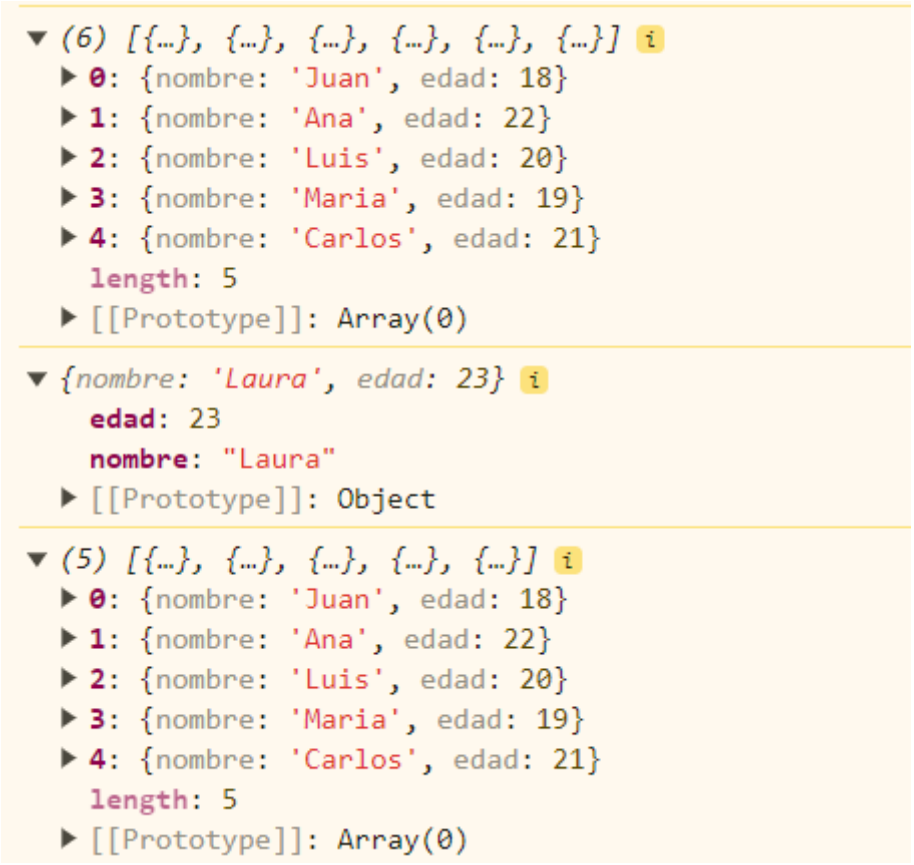
```
Edad Promedio: 20
```

(Resultado esperado paso 5)

Paso 6: Añadir y Eliminar Elementos

Vamos a añadir un nuevo estudiante al array utilizando el método **push()** y luego eliminar el último estudiante añadido utilizando el método **pop()**.

```
estudiantes.push({ nombre: "Laura", edad: 23 });
console.log(estudiantes);

let ultimoEstudiante = estudiantes.pop();
console.log(ultimoEstudiante);
console.log(estudiantes);
```



(Resultado esperado paso 6)

Paso 7: Recorrer un Array

Finalmente, recorreremos el array **estudiantes** y mostraremos el nombre de cada estudiante utilizando **forEach()**.

```
estudiantes.forEach(estudiante => {
    console.log(estudiante.nombre);
  });
```

```
Juan
Ana
Luis
Maria
Carlos
```

(Resultado esperado paso 7)

Resumen de Métodos Utilizados

- **filter()**: Crea un nuevo array con todos los elementos que pasan una prueba.
- **map()**: Crea un nuevo array con los resultados de aplicar una función a cada elemento.
- **find()**: Devuelve el primer elemento que cumple con la función de prueba proporcionada.
- **reduce()**: Aplica una función a un acumulador y cada valor del array (de izquierda a derecha) para reducirlo a un único valor.
- **push()**: Añade uno o más elementos al final del array.
- **pop()**: Elimina el último elemento de un array y lo devuelve.
- **forEach()**: Ejecuta la función proporcionada una vez por cada elemento del array.

3.4.5. EJERCICIO PRÁCTICO DE ARRAY

Para consolidar lo aprendido, intenta realizar lo siguiente:

1. Crea un array de números del 1 al 10.
2. Filtra los números pares.
3. Usa **map()** para crear un nuevo array que contenga los cuadrados de los números pares.
4. Utiliza **reduce()** para encontrar la suma de los cuadrados.
5. Añade un nuevo número al array original y luego elimina el primer número.

3.5. EL OBJETO DATE EN JAVASCRIPT

3.5.1. INTRODUCCIÓN AL OBJETO DATE

El objeto Date en JavaScript es una herramienta fundamental para manejar fechas y horas en el desarrollo web. Fue introducido en la primera versión de JavaScript (ECMAScript 1) y ha evolucionado a lo largo de los años para ofrecer una amplia gama de funcionalidades. Este objeto es esencial para cualquier aplicación que necesite trabajar con fechas, ya sea para mostrar la fecha y hora actuales, programar eventos futuros o calcular diferencias entre fechas.

3.5.2. ¿POR QUÉ SE CREÓ EL OBJETO DATE?

El objeto Date se creó para simplificar el manejo de fechas y tiempos en las aplicaciones web. Antes de su introducción, tratar con fechas y horas era un proceso tedioso y propenso a errores. Con Date, los desarrolladores pueden:

- Obtener la fecha y hora actuales.
- Crear fechas específicas.
- Calcular diferencias entre fechas.
- Formatear fechas para su presentación.
- Manipular componentes individuales de una fecha (como año, mes, día, etc.).

3.5.3. ¿CÓMO FUNCIONA EL OBJETO DATE?

El objeto Date en JavaScript se basa en el tiempo universal coordinado (UTC) y la época Unix, que es el tiempo transcurrido en milisegundos desde el 1 de enero de 1970 a las 00:00:00 UTC. Esto proporciona una referencia común para todas las fechas y horas, permitiendo una manipulación precisa y consistente.

3.5.4. DATOS DE INTERÉS

- **Época Unix**: La referencia de tiempo utilizada por el objeto Date es el 1 de enero de 1970.

- **Zonas Horarias**: El objeto Date maneja automáticamente las diferencias de zonas horarias, lo cual es crucial para aplicaciones globales.
- **Milisegundos**: El objeto Date mide el tiempo en milisegundos, lo que permite una alta precisión en las operaciones de tiempo.

3.5.5. CREACIÓN DE UN OBJETO DATE

El objeto Date nos permite trabajar con fechas y horas. Hay varias formas de crear un objeto Date en JavaScript:

Sin parámetros

Si creas un objeto Date sin parámetros, obtendrás la fecha y hora actuales:

```
let fechaActual = new Date();
console.log(fechaActual);
```

La salida será algo como:

```
Tue May 21 2024 19:19:02 GMT+0200 (hora de verano de Europa central)
```

Con una Cadena de Texto

Puedes crear un objeto Date a partir de una cadena de texto que represente una fecha:

```
let fechaEspecifica = new Date('2024-05-15T14:22:05');
console.log(fechaEspecifica);
```

Con Valores Numéricos

Otra forma es especificando los componentes individuales de la fecha:

```
let fechaConNumeros = new Date(2024, 4, 15, 14, 22, 5); // Año, Mes (0-11), Día, Hora, Minuto, Segundo
console.log(fechaConNumeros);
```

Nota: Los meses en JavaScript están indexados desde 0 (enero) hasta 11 (diciembre).

Por lo general queremos implementar un formato de fecha más amigable para el usuario donde nos muestre 2 digitos para la hora 2 digitos para los minutos y para la fecha o algo parecido con un formato como se ve en la siguiente captura

3.5.6. PRACTICA GUIADA OBJETO DATE

En esta actividad, crearemos una página web sencilla que mostrará la fecha actual de manera formateada. Usaremos el objeto fecha para obtener la fecha y formato adecuado según las convenciones locales (en este caso, el formato español). También añadiremos un botón que, al ser presionado, borrara la fecha.

Para lograrlo utilizaremos:

1. **HTML**: Estructuraremos la página con los elementos necesarios para mostrar la fecha y el botón de restablecimiento.
2. **CSS**: Aplicaremos estilos básicos para mejorar la apariencia de nuestra página.
3. **JavaScript**: Escribiremos el código necesario para:
    - Obtener la fecha y hora actuales.
    - Formatear la fecha utilizando el método toLocaleDateString.
    - Actualizar el contenido de la página con la fecha formateada.
    - Añadir un evento al botón para restablecer la fecha.

- **HTML**: Se añade un botón con id="resetButton" y la clase boton-reset dentro del contenedor de la fecha.

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Mostrar Fecha Actual</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="contenedor-fecha">
        <h1>Fecha Actual</h1>
        <p id="fecha"></p>
        <button id="resetButton" class="boton-reset">Restablecer
Fecha</button>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

- **CSS**: Se estiliza el botón para que se vea atractivo y con un efecto de hover.

```css
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.contenedor-fecha {
    text-align: center;
    background-color: #ffffff;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
```

```css
h1 {
    color: #333;
}

#fecha {
    font-size: 1.5em;
    color: #555;
}

.boton-reset {
    margin-top: 20px;
    padding: 10px 20px;
    font-size: 1em;
    color: #ffffff;
    background-color: #00A4E2;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
}

.boton-reset:hover {
    background-color: #00A4E2;
}
```

```javascript
document.addEventListener('DOMContentLoaded', () => {
    const fechaElemento = document.getElementById('fecha');
    const resetButton = document.getElementById('resetButton');

    function mostrarFechaActual() {
        const fechaActual = new Date();
        const opciones = {
            weekday: 'long',
            year: 'numeric',
            month: 'long',
            day: 'numeric'
        };
        const fechaFormateada = fechaActual.toLocaleDateString('es-ES',
opciones);
        fechaElemento.textContent = fechaFormateada;
    }

    resetButton.addEventListener('click', () => {
```

```
        fechaElemento.textContent = '';
    });

    mostrarFechaActual();
});
```

- **JavaScript**:
    - Se define la función mostrarFechaActual() para actualizar el contenido del elemento #fecha con la fecha actual formateada.
    - Se añade un evento al botón de restablecimiento que limpia el contenido del elemento #fecha y luego llama a mostrarFechaActual() para volver a mostrar la fecha actual.
    - Se llama a mostrarFechaActual() cuando la página se carga por primera vez para mostrar la fecha inmediatamente.

Formatear la Fecha

Para formatear la fecha en un formato legible y localizado, usamos el método toLocaleDateString. Este método convierte un objeto Date en una cadena de texto según las convenciones locales.

- **locales** (opcional): Una cadena de texto con un identificador de idioma (por ejemplo, 'es-ES' para español de España).
- **options** (opcional): Un objeto que configura opciones para la representación de la fecha.

3.5.7. MÉTODOS DEL OBJETO DATE

El objeto Date viene con una variedad de métodos que te permiten obtener y manipular diferentes partes de la fecha y hora. En esta tabla podrás ver varios de ellos.

Categoría	Método	Descripción	Ejemplo
Obtener información	**getFullYear()**	Devuelve el año de la fecha especificada según la hora local.	fecha.getFullYear()
	getMonth()	Devuelve el mes de la fecha especificada según la hora local (0-11).	fecha.getMonth()

	getDate()	Devuelve el día del mes de la fecha especificada según la hora local (1-31).	fecha.getDate()
	getDay()	Devuelve el día de la semana de la fecha especificada según la hora local (0-6).	fecha.getDay()
	getHours()	Devuelve la hora de la fecha especificada según la hora local (0-23).	fecha.getHours()
	getMinutes()	Devuelve los minutos de la fecha especificada según la hora local (0-59).	fecha.getMinutes()
	getSeconds()	Devuelve los segundos de la fecha especificada según la hora local (0-59).	fecha.getSeconds()
Establecer información	**setFullYear(año, mes, día)**	Establece el año de la fecha.	fecha.setFullYear(2025)
	setMonth(mes)	Establece el mes de la fecha (0-11).	fecha.setMonth(6)
	setDate(día)	Establece el día del mes de la fecha (1-31).	fecha.setDate(15)
	setHours(hora)	Establece la hora de la fecha (0-23).	fecha.setHours(10)
	setMinutes(minutos)	Establece los minutos de la fecha (0-59).	fecha.setMinutes(45)
	setSeconds(segundos)	Establece los segundos de la fecha (0-59).	fecha.setSeconds(30)
	setMilliseconds(milisegundos)	Establece los milisegundos de la fecha (0-999).	fecha.setMilliseconds(500)
Formatear fechas	**toISOString()**	Devuelve una cadena en formato ISO (ISO 8601) de la fecha.	fecha.toISOString()
	toDateString()	Devuelve una cadena con la parte de la fecha de la fecha especificada.	fecha.toDateString()
	toTimeString()	Devuelve una cadena con la parte de la hora de la fecha especificada.	fecha.toTimeString()

Métodos estáticos	**Date.now()**	Devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970 00:00:00 UTC hasta ahora.	Date.now()
	Date.parse(cadena)	Parsea una cadena de texto y devuelve el número de milisegundos desde el 1 de enero de 1970 00:00:00 UTC.	Date.parse('2024-05-21T15:30:00Z')
	Date.UTC(año, mes, día, hora, minutos, segundos, ms)	Acepta los mismos parámetros que el constructor de Date, pero devuelve el valor en milisegundos desde el 1 de enero de 1970 00:00:00 UTC.	Date.UTC(2024, 4, 21, 15, 30, 0)

3.6. INTRODUCCIÓN A LOS MARCOS

Los marcos (<iframe>) permiten incrustar un documento HTML dentro de otro documento HTML. Esto es útil para cargar contenido externo, como anuncios, videos, o incluso aplicaciones completas, sin redirigir al usuario a otra página. Los marcos son esenciales para crear interfaces de usuario modulares y para integrar contenido de múltiples fuentes de manera segura y eficiente.

3.6.1. ESTRUCTURA BÁSICA DE UN MARCO

La etiqueta <iframe> se utiliza para crear un marco en HTML. Aquí tienes un ejemplo básico:

```
<iframe src="https://www.example.com" width="600" height="400"></iframe>
```

src: La URL del documento que se va a cargar en el marco.

width y height: Las dimensiones del marco en píxeles.

Puedes agregar parámetros adicionales para mejorar la funcionalidad y la apariencia del marco.

3.6.2. PARÁMETROS COMUNES DE <IFRAME>

- **frameborder**: Establece el borde del marco (valores: 0 para sin borde, 1 para borde).
- **scrolling**: Controla la aparición de barras de desplazamiento (yes, no, auto).
- **name**: Asigna un nombre al marco, útil para la navegación dirigida.

```
<iframe src="https://www.example.com" width="600" height="400"
frameborder="0" scrolling="no" name="miMarco"></iframe>
```

3.6.3. GESTIÓN DINÁMICA DE MARCOS CON JAVASCRIPT

JavaScript permite manipular marcos de manera dinámica, ofreciendo una mayor flexibilidad y control sobre el contenido y comportamiento del mismo. Puedes cambiar la

URL del marco utilizando JavaScript. Esto es útil cuando necesitas actualizar el contenido del marco sin recargar toda la página.

Por ejemplo, imagina que tienes un <iframe> en tu HTML:

3.7. GESTIÓN DE LA APARIENCIA DE LA VENTANA CON EL OBJETO WINDOW

El objeto window es un objeto global que representa la ventana del navegador y proporciona métodos y propiedades para controlar su apariencia y comportamiento. A continuación, exploraremos cómo utilizar el objeto window para gestionar el tamaño, la posición y otras características de la ventana.

```
<iframe id="miMarco" src="https://www.example.com" width="600"
height="400"></iframe>
```

Para cambiar la URL del marco, puedes usar el siguiente código JavaScript:

```
let marco = document.getElementById('miMarco');
marco.src = "https://www.otro-sitio.com";
```

Además, es posible interactuar con el contenido del marco mediante JavaScript. Esto te permite acceder y manipular el documento cargado dentro del marco. Supongamos que quieres cambiar el título del documento dentro del marco. Puedes hacerlo de la siguiente manera:

```
let marco = document.getElementById('miMarco');
marco.onload = function() {
  let doc = marco.contentDocument || marco.contentWindow.document;
  doc.title = "Nuevo Título";
};
```

Este enfoque no solo te permite cambiar la URL del marco, sino también interactuar con su contenido, ajustando dinámicamente lo que se muestra al usuario en función de sus acciones o del estado actual de la aplicación.

3.7.1. INTEGRACIÓN DE CONTENIDO EXTERNO

Incrustar contenido de otros sitios web sin redirigir al usuario puede mejorar la experiencia del usuario y mantener la coherencia del sitio. Un ejemplo común es la inclusión de videos de

YouTube dentro de un marco. Esto se puede hacer fácilmente utilizando la etiqueta <iframe> con la URL del video de YouTube.

Ejemplo: Incluir un video de YouTube:

```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/VIDEO_ID" frameborder="0"
allowfullscreen></iframe>
```

3.7.2. INTERFACES DE USUARIO MODULARES

Dividir una aplicación en varios marcos puede simplificar la gestión y actualización del contenido. Por ejemplo, una aplicación de correo electrónico puede tener un marco para la lista de correos y otro para el contenido del correo seleccionado. Esto permite actualizar cada sección de la aplicación de manera independiente, mejorando la modularidad y la mantenibilidad del código.

Ejemplo: Aplicación de correo electrónico:

En este ejemplo, el primer marco (lista-correos.html) podría contener una lista de correos electrónicos, mientras que el segundo marco (contenido-correo.html) mostraría el contenido del correo seleccionado. Esto facilita la navegación y mejora la experiencia del usuario al permitir que diferentes partes de la interfaz se actualicen de manera independiente.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Aplicación de Correo</title>
  <style>
    body { display: flex; }
    iframe { border: none; }
  </style>
</head>
<body>
  <iframe src="lista-correos.html" width="30%" height="100%"></iframe>
  <iframe src="contenido-correo.html" width="70%" height="100%"></iframe>
</body>
</html>
```

3.8. GESTIÓN DE LA APARIENCIA DE LA VENTANA CON JAVASCRIPT

La gestión de la apariencia de la ventana mediante JavaScript es una práctica esencial en el desarrollo web moderno. Te permitirá crear interfaces dinámicas y responsivas que mejoran significativamente la experiencia del usuario. A continuación, se detallan varias técnicas y ejemplos sobre cómo puedes utilizar JavaScript para controlar y mejorar la apariencia de tus ventanas web.

3.8.1. 1. CAMBIAR ESTILOS DINÁMICAMENTE

JavaScript proporciona métodos para cambiar los estilos de los elementos en tiempo real. Puedes modificar propiedades CSS directamente o agregar y eliminar clases CSS para aplicar estilos predefinidos.

3.8.2. EJEMPLO: CAMBIAR EL COLOR DE FONDO DE UNA VENTANA

En este ejemplo, utilizas JavaScript para alternar una clase que cambia el color de fondo de la ventana cuando haces clic en ella.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Gestión de Apariencia con JavaScript</title>
  <style>
    .ventana {
      width: 500px;
      height: 300px;
      padding: 20px;
      background-color: white;
      border: 1px solid #ccc;
      transition: background-color 0.3s;
    }
    .ventana.activa {
      background-color: #f0f8ff;
    }
  </style>
</head>
<body>
  <div id="ventana" class="ventana">Haz clic para cambiar el color de
fondo.</div>

  <script>
    const ventana = document.getElementById('ventana');
```

```
    ventana.addEventListener('click', () => {
      ventana.classList.toggle('activa');
    });
  </script>
</body>
</html>
```

3.8.3. 2. MODIFICAR TAMAÑO Y POSICIÓN

El código también te permite controlar el tamaño y la posición de las ventanas y elementos dentro de ellas. Esto es especialmente útil para crear interfaces de usuario adaptativas.

Ejemplo: Redimensionar una Ventana

En este ejemplo, la ventana puede ser redimensionada por ti, y el script se usa para cambiar el color del borde durante la interacción.

3.8.4. CREACIÓN DE VENTANAS MODALES

Las ventanas modales son un componente común en las interfaces web. Los scripts facilitan la creación y gestión de estas ventanas, permitiendo mostrar contenido adicional sin abandonar la página actual.

3.9. CREACIÓN DE NUEVAS VENTANAS Y COMUNICACIÓN ENTRE VENTANAS

El objeto window en JavaScript es fundamental para la manipulación y gestión de ventanas en el navegador. Este objeto representa una ventana que contiene un DOM document y proporciona una API rica para controlar su comportamiento. A continuación, se explorarán las capacidades de window, cómo abrir nuevas ventanas, y cómo lograr la comunicación entre ellas.

3.9.1. CREACIÓN DE NUEVAS VENTANAS

Puedes abrir nuevas ventanas o pestañas utilizando el método window.open(). Este método permite especificar la URL, el nombre de la ventana, y varias características de la ventana (como su tamaño, si es redimensionable, etc.).

window.open(url, windowName, [windowFeatures]);

- url: La URL que se abrirá en la nueva ventana. Puede ser una URL completa o una ruta relativa.
- windowName: Un nombre para la ventana. Si se abre una ventana con el mismo nombre, se reutilizará la existente.
- windowFeatures: Una cadena de características de la ventana, como tamaño, posición, y si tiene barras de herramientas.

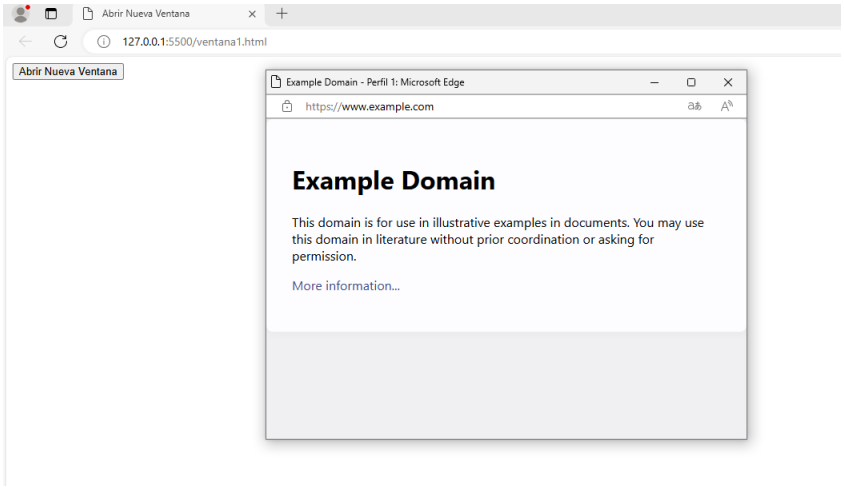
3.9.2. PRÁCTICA: ABRIR UNA NUEVA VENTANA

En este ejemplo, hacer clic en el botón abrirá una nueva ventana con la URL especificada, un tamaño de 600x400 píxeles y la capacidad de ser redimensionada.

1. **Estructura HTML**:
    - La estructura básica de un documento HTML con un botón que, cuando se presiona, ejecutará un script.
    - El botón tiene un id de abrirVentana.
2. **JavaScript**:
    - Se agrega un eventListener al botón que escucha el evento click.
    - Cuando se hace clic en el botón, la función anónima dentro de addEventListener se ejecuta.
    - Dentro de esta función, se llama a window.open() con tres argumentos:
        - 'https://www.example.com': La URL que se abrirá en la nueva ventana.
        - '_blank': Indica que la URL debe abrirse en una nueva ventana o pestaña.
        - 'width=600,height=400,resizable=yes': Especifica las características de la ventana (ancho, alto, y si es redimensionable).

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Abrir Nueva Ventana</title>
```

```
</head>
<body>
  <button id="abrirVentana">Abrir Nueva Ventana</button>

  <script>
    document.getElementById('abrirVentana').addEventListener('click', ()
=> {
      window.open('https://www.example.com', '_blank',
'width=600,height=400,resizable=yes');
    });
  </script>
</body>
</html>
```



(Resultado esperado)

3.9.3. MÉTODOS DEL OBJETO WINDOW

El objeto window proporciona una amplia variedad de métodos para interactuar con la ventana del navegador:

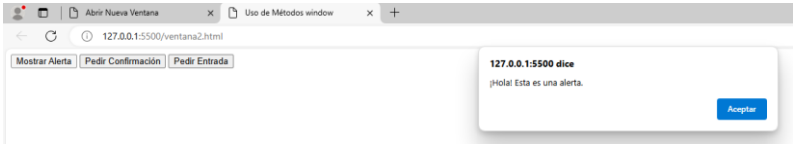
- **window.alert(message)**: Muestra un cuadro de alerta con un mensaje y un botón "Aceptar".
- **window.confirm(message)**: Muestra un cuadro de diálogo con un mensaje y botones "Aceptar" y "Cancelar". Retorna true si se hace clic en "Aceptar".
- **window.prompt(message, defaultValue)**: Muestra un cuadro de diálogo con un mensaje, un campo de entrada de texto, y botones "Aceptar" y "Cancelar". Retorna el texto ingresado si se hace clic en "Aceptar".
- **window.close()**: Cierra la ventana actual.
- **window.focus()**: Lleva la ventana actual a primer plano.
- **window.blur()**: Saca el foco de la ventana actual.

Función	Descripción	Ejemplo
window.alert(message)	Muestra un cuadro de alerta con un mensaje y un botón "Aceptar".	window.alert("Este es un mensaje de alerta.");
window.confirm(message)	Muestra un cuadro de diálogo con un mensaje y botones "Aceptar" y "Cancelar". Retorna true si se hace clic en "Aceptar".	if (window.confirm("¿Está seguro de que desea continuar?")) { console.log("El usuario hizo clic en Aceptar."); } else { console.log("El usuario hizo clic en Cancelar."); }
window.prompt(message, defaultValue)	Muestra un cuadro de diálogo con un mensaje, un campo de entrada de texto, y botones "Aceptar" y "Cancelar". Retorna el texto ingresado si se hace clic en "Aceptar".	let nombre = window.prompt("Por favor, ingrese su nombre:", "Nombre"); if (nombre !== null) { console.log(\Hola, ${nombre}!`); } else { console.log("El usuario canceló la entrada."); }`
window.close()	Cierra la ventana actual.	window.close();
window.focus()	Lleva la ventana actual a primer plano.	window.focus();
window.blur()	Saca el foco de la ventana actual.	window.blur();

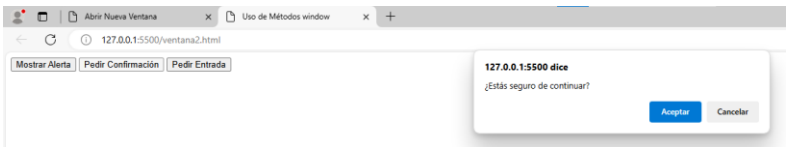
Práctica: Uso de Métodos del Objeto window

Esta práctica vemos cómo usar métodos del objeto window para interactuar con el usuario mediante alertas, confirmaciones y entradas de texto.

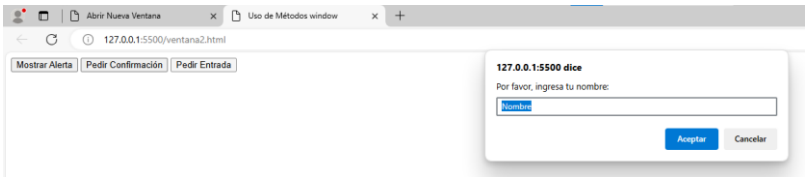
1. **Estructura HTML**:
    - Tres botones con diferentes id para mostrar alertas, pedir confirmaciones y pedir entradas de texto.
2. **JavaScript**:
    - **Mostrar Alerta**:
        - Se agrega un eventListener al botón con id="mostrarAlerta".
        - Cuando se hace clic, se llama a window.alert(), que muestra una alerta con el mensaje provisto.



- **Pedir Confirmación**:
    - Similarmente, se agrega un eventListener al botón con id="pedirConfirmacion".
    - Al hacer clic, se llama a window.confirm(), que muestra un cuadro de diálogo con un mensaje y botones "Aceptar" y "Cancelar".
    - El resultado (true o false) se registra en la consola.



- **Pedir Entrada**:
    - Se agrega un eventListener al botón con id="pedirEntrada".
    - Al hacer clic, se llama a window.prompt(), que muestra un cuadro de diálogo con un mensaje y un campo de entrada de texto.
    - El valor ingresado se registra en la consola.




```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Uso de Métodos window</title>
</head>
<body>
  <button id="mostrarAlerta">Mostrar Alerta</button>
  <button id="pedirConfirmacion">Pedir Confirmación</button>
  <button id="pedirEntrada">Pedir Entrada</button>

  <script>
    document.getElementById('mostrarAlerta').addEventListener('click', ()
=> {
      window.alert('¡Hola! Esta es una alerta.');
    });

    document.getElementById('pedirConfirmacion').addEventListener('click'
, () => {
      const resultado = window.confirm('¿Estás seguro de continuar?');
      console.log(`Confirmación: ${resultado}`);
    });

    document.getElementById('pedirEntrada').addEventListener('click', ()
=> {
      const nombre = window.prompt('Por favor, ingresa tu nombre:',
'Nombre');
      console.log(`Nombre ingresado: ${nombre}`);
    });
  </script>
</body>
</html>
```

3.9.4. PRÁCTICA GUIADA: MANIPULACIÓN DE VENTANAS CON EL OBJETO WINDOW EN JAVASCRIPT

Descripción

En esta práctica, aprenderás a utilizar varios métodos del objeto window en JavaScript para manipular ventanas y gestionar la navegación del navegador. Implementarás una página web que abrirá y cerrará ventanas, cambiará la ubicación y utilizará métodos como alert, confirm, prompt, close, focus y blur.

Objetivos

- Mostrar mensajes y cuadros de diálogo utilizando window.alert, window.confirm y window.prompt.
- Manipular ventanas con window.open, window.close, window.focus y window.blur.
- Probar los métodos en un entorno local.

Instrucciones de la Práctica

1. Preparación del Entorno

- **Instala XAMPP (si no lo tienes instalado):**
  - Descárgalo desde Apache Friends e instálalo en tu computadora.
  - Inicia el servidor Apache desde el Panel de Control de XAMPP.
- **Crear la Estructura de Archivos:**
  - Crea una carpeta llamada manipulacion_ventanas dentro del directorio htdocs de XAMPP (C:\xampp\htdocs\manipulacion_ventanas).

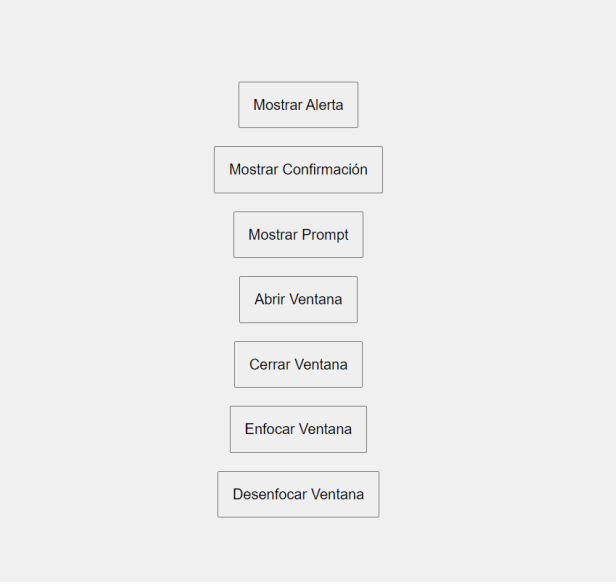
2. Crear la Página Principal

- Crea el archivo index.html en la carpeta manipulacion_ventanas.
- Copia el siguiente código en index.html:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Manipulación de Ventanas</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
```

```
      height: 100vh;
      background-color: #f0f0f0;
    }
    button {
      margin: 10px;
      padding: 15px;
      font-size: 16px;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <button onclick="mostrarAlerta()">Mostrar Alerta</button>
  <button onclick="mostrarConfirmacion()">Mostrar Confirmación</button>
  <button onclick="mostrarPrompt()">Mostrar Prompt</button>
  <button onclick="abrirVentana()">Abrir Ventana</button>
  <button onclick="cerrarVentana()">Cerrar Ventana</button>
  <button onclick="enfocarVentana()">Enfocar Ventana</button>
  <button onclick="desenfocarVentana()">Desenfocar Ventana</button>

  <script>
    function mostrarAlerta() {
      window.alert('¡Hola! Esto es una alerta.');
    }

    function mostrarConfirmacion() {
      const resultado = window.confirm('¿Estás seguro de que deseas
continuar?');
      if (resultado) {
        window.alert('Has confirmado.');
      } else {
        window.alert('Has cancelado.');
      }
    }

    function mostrarPrompt() {
      const nombre = window.prompt('¿Cómo te llamas?', 'Nombre');
      if (nombre) {
        window.alert(`Hola, ${nombre}`);
      } else {
        window.alert('No ingresaste un nombre.');
      }
    }

    let nuevaVentana;

    function abrirVentana() {
      nuevaVentana = window.open('', 'nuevaVentana',
'width=400,height=400');
```

```
      nuevaVentana.document.write('<p>Esta es una nueva ventana.</p>');
    }

    function cerrarVentana() {
      if (nuevaVentana) {
        nuevaVentana.close();
      } else {
        window.alert('No hay ninguna ventana abierta.');
      }
    }

    function enfocarVentana() {
      if (nuevaVentana) {
        nuevaVentana.focus();
      } else {
        window.alert('No hay ninguna ventana abierta.');
      }
    }

    function desenfocarVentana() {
      if (nuevaVentana) {
        nuevaVentana.blur();
      } else {
        window.alert('No hay ninguna ventana abierta.');
      }
    }
  </script>
</body>
</html>
```



(resultado esperado)

3. Probar la Página

- Inicia XAMPP y asegúrate de que el servidor Apache esté corriendo.
- Abre tu navegador y navega a http://localhost/manipulacion_ventanas/index.html.
- Interactúa con los botones para probar las funciones de window.alert, window.confirm, window.prompt, window.open, window.close, window.focus y window.blur.

3.9.5. COMUNICACIÓN ENTRE VENTANAS

Una vez que tienes múltiples ventanas abiertas, es posible que necesites que se comuniquen entre sí. Puedes lograr esto utilizando la referencia a la ventana que retorna window.open() y métodos como postMessage.

La URL que debes usar en el método postMessage debe coincidir con el origen de la ventana principal desde la cual abriste la ventana secundaria. Si estás accediendo a tus archivos a través de http://localhost:80/ejercicio/index.html, entonces la URL en postMessage debería ser http://localhost:80.

Problemas Comunes

Es importante asegurarse de que la comunicación entre ventanas funcione correctamente, y esto incluye tener en cuenta varios factores como el mismo origen (misma URL y puerto) y el uso adecuado de postMessage. Aquí tienes una explicación detallada y soluciones para posibles problemas.

1. **Origen Incorrecto**: El origen (origin) debe ser el mismo en ambas ventanas. Esto incluye el protocolo (http:// o https://), el dominio (localhost o un dominio específico) y el puerto (si lo hay).
2. **Cargar Correctamente los Archivos**: Asegúrate de que ambos archivos HTML estén correctamente cargados y accesibles.

3.10. PRÁCTICA GUIADA: COMUNICACIÓN ENTRE DOS VENTANAS

En esta práctica, aprenderás a establecer comunicación entre dos ventanas utilizando la API de postMessage en JavaScript. Implementarás una ventana principal que abrirá una ventana secundaria y enviará mensajes entre ellas. Además, se proporcionarán instrucciones para configurar y probar esta comunicación en un entorno local utilizando XAMPP.

Objetivos

1. Crear una ventana principal que pueda abrir una ventana secundaria.
2. Enviar mensajes desde la ventana secundaria a la ventana principal utilizando postMessage.
3. Recibir y mostrar mensajes en la ventana principal.
4. Configurar un entorno local con XAMPP para probar la comunicación entre las ventanas.

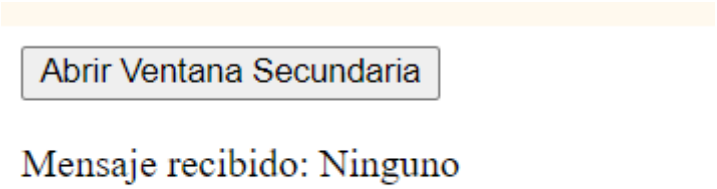
1. Crear la Estructura de Archivos

- Crea una carpeta para tu proyecto: Por ejemplo, comunicacion_ventanas.
- Dentro de esta carpeta, crea dos archivos HTML:
- index.html (para la ventana principal)
- ventana_secundaria.html (para la ventana secundaria)

2. Código para index.html (Ventana Principal)

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ventana Principal</title>
</head>
<body>
  <button id="abrirVentanaSecundaria">Abrir Ventana Secundaria</button>
  <p id="mensajeRecibido">Mensaje recibido: Ninguno</p>

  <script>
    let ventanaSecundaria;

    document.getElementById('abrirVentanaSecundaria').addEventListener('c
lick', () => {
      ventanaSecundaria = window.open('ventana_secundaria.html',
'_blank', 'width=600,height=400');
    });

    window.addEventListener('message', (event) => {
      if (event.origin === 'http://localhost') {
        document.getElementById('mensajeRecibido').innerHTML = `Mensaje
recibido: ${event.data}`;
      }
      else{
        console.log(event.origin);
        console.log(event.data);
      }
    });

  </script>
```

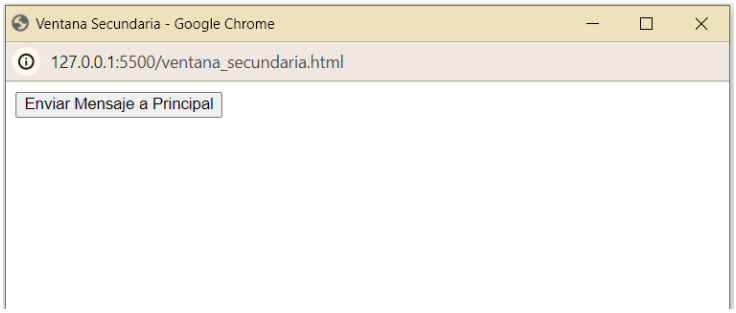
```
</body>
</html>
```



(Captura de ventana principal)

2. Código para ventana_secundaria.html (Ventana Secundaria)

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ventana Secundaria</title>
</head>
<body>
  <button id="enviarMensaje">Enviar Mensaje a Principal</button>

  <script>
    document.getElementById('enviarMensaje').addEventListener('click', ()
=> {
      window.opener.postMessage('Hola desde la ventana secundaria',
'http://localhost:80');

    });
  </script>
</body>
</html>
```

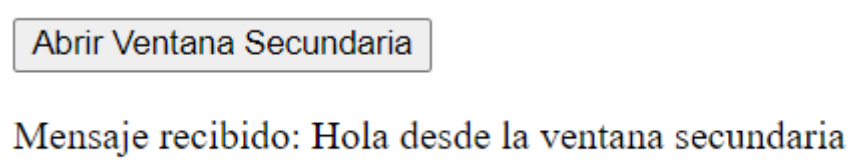


(captura de ventana secundaria)

3. Configurar XAMPP para Ejecutar el Proyecto

1. **Instalar XAMPP:**
    - Descarga e instala XAMPP desde Apache Friends.
2. **Iniciar el Servidor Apache:**
    - Abre el Panel de Control de XAMPP.
    - Inicia el servidor Apache haciendo clic en el botón "Start" junto a "Apache".
3. **Colocar los Archivos en el Directorio de XAMPP:**
    - Copia la carpeta comunicacion_ventanas a htdocs dentro del directorio de instalación de XAMPP. Por ejemplo, si instalaste XAMPP en C:\xampp, la ruta completa sería C:\xampp\htdocs\comunicacion_ventanas.
4. **Acceder a la Ventana Principal:**
    - Abre tu navegador y navega a http://localhost/comunicacion_ventanas/index.html.
5. **Probar la Comunicación:**
    - En la ventana principal, haz clic en el botón "Abrir Ventana Secundaria". Esto abrirá ventana_secundaria.html en una nueva ventana.
    - En la ventana secundaria, haz clic en el botón "Enviar Mensaje a Principal". Deberías ver Mensaje recibido: Hola desde la ventana secundaria en la ventana principal y http://localhost:80 en la consola del navegador (ajusta el puerto si es necesario).

Resultado esperado en la ventana Principal:



3.11. EJERCICIOS

Objetivo: El objetivo de estos ejercicios es que afiances tus conocimientos en la manipulación de objetos, arrays, y fechas en JavaScript, así como en la creación y uso de constructores y métodos avanzados de manipulación de arrays. Además, te familiarizarás con la dinámica de estilos en una página web y la creación de ventanas del navegador, lo cual es crucial para el desarrollo web interactivo. A través de estos ejercicios, aprenderás a crear y modificar objetos, iterar sobre sus propiedades, y utilizar funciones constructoras. También desarrollarás habilidades para trabajar con arrays y fechas, y para manipular el DOM de manera efectiva.

Duración: 3.5hs.

Entrega: Para la entrega de los ejercicios de la Unidad 3, debes crear una carpeta llamada Unidad3_Ejercicios que contenga un archivo HTML (index.html) y un archivo JavaScript (scripts.js) en los cuales hayas implementado y comentado todas las soluciones de los ejercicios propuestos. Asegúrate de que el archivo HTML esté correctamente enlazado al archivo JavaScript y que todo el código esté claramente organizado y documentado. Una vez que hayas terminado, comprime la carpeta Unidad3_Ejercicios en un archivo ZIP y súbelo a la plataforma.

EJERCICIO 1: CREACIÓN DE OBJETOS

- Crea un objeto estudiante que contenga las propiedades nombre, edad, curso, y un método mostrarInfo que imprima en la consola la información del estudiante.

EJERCICIO 2: MODIFICACIÓN DE PROPIEDADES

- A partir del objeto estudiante creado en el ejercicio anterior, añade una propiedad promedio e inicialízala con un valor numérico. Luego, modifica la propiedad curso y cámbiala por otro curso diferente.

EJERCICIO 3: ITERACIÓN SOBRE PROPIEDADES

- Crea una función que reciba un objeto como parámetro e imprima en la consola todas las claves y valores de dicho objeto utilizando un bucle for...in.

EJERCICIO 4: CREACIÓN DE CONSTRUCTOR

- Define una función constructora Vehiculo que tome marca, modelo, y año como parámetros. Define un método mostrarDetalles que imprima estos detalles en la consola. Crea dos instancias de Vehiculo y llama al método mostrarDetalles para cada una.

EJERCICIO 5: MANIPULACIÓN DE ARRAYS

- Crea un array de cadenas que contenga nombres de frutas. Ordena el array alfabéticamente y luego invierte el orden. Muestra el array resultante en la consola.

EJERCICIO 6: MÉTODOS AVANZADOS DE ARRAYS

- Utiliza el método map para crear un nuevo array que contenga la longitud de cada cadena de un array dado de nombres. Imprime el nuevo array en la consola.

EJERCICIO 7: TRABAJANDO CON FECHAS

- Crea un objeto Date con la fecha actual y muestra en la consola el día de la semana, el día del mes, el mes y el año por separado.

EJERCICIO 8: CAMBIO DINÁMICO DE ESTILOS

- Escribe una función que cambie el color de texto de todos los párrafos en una página web a un color proporcionado como parámetro.

EJERCICIO 9: CREACIÓN DE VENTANAS Y COMUNICACIÓN

- Crea una nueva ventana del navegador que contenga un mensaje de bienvenida. Implementa un botón en la ventana principal que, al ser clicado, cierre la nueva ventana.

3.12. RESUMEN

1. **¿Qué es un Objeto en JavaScript?**
    - Definición y estructura de un objeto en JavaScript.
    - Clave-Valor: Cada propiedad de un objeto se define como un par clave-valor.
2. **Creación y Modificación de Objetos**
    - Sintaxis literal para crear objetos.
    - Acceso y modificación de propiedades usando notación de punto y corchetes.
    - Iteración sobre propiedades con bucles for...in.
3. **Constructores en JavaScript**
    - Uso de constructores para crear nuevos objetos.
    - Concepto de prototipos y cómo se utilizan para heredar propiedades y métodos.
    - Ejemplos prácticos de creación y uso de constructores y prototipos.
4. **Arrays en Profundidad**
    - Estructura y funcionamiento de arrays.
    - Métodos avanzados para manipular arrays.
    - Recorrer arrays y ejemplos prácticos de manipulación.
5. **El Objeto Date en JavaScript**
    - Introducción y creación de objetos Date.
    - Importancia y métodos del objeto Date.
    - Prácticas guiadas para trabajar con fechas.
6. **Gestión de Marcos (<iframe>)**
    - Estructura básica y parámetros comunes.
    - Gestión dinámica de marcos con JavaScript.
7. **Gestión de la Apariencia de la Ventana**
    - Uso del objeto window para controlar la apariencia de la ventana.
    - Ejemplos de cómo cambiar estilos dinámicamente y crear ventanas modales.
8. **Creación de Nuevas Ventanas y Comunicación entre Ventanas**
    - Métodos para crear nuevas ventanas y comunicarse entre ellas.
    - Prácticas guiadas para manejar ventanas con el objeto window.

3.13. BIBLIOGRAFÍA

- Mozilla Developer Network (MDN) URL: https://developer.mozilla.org/
- W3Schools URL: https://www.w3schools.com/
- DesarrolloWeb.com URL: https://desarrolloweb.com/
- AMX Web URL: https://amxweb.com/