OBJETOS PREDEFINIDOS

Los objetos predefinidos en JavaScript son objetos que ya están disponibles en el lenguaje y no necesitan ser creados por el usuario. Estos objetos proporcionan funcionalidades básicas y comunes que son esenciales para el desarrollo web y otras aplicaciones JavaScript. Algunos de los objetos predefinidos más importantes son:

1. **Objeto Window**
   - Representa la ventana del navegador.
   - Permite acceder a propiedades y métodos como la URL, historial, tamaño de la ventana, etc.
2. **Objeto Document**
   - Representa el documento HTML de la página.
   - Facilita el acceso y manipulación de elementos y atributos del DOM (Document Object Model).
3. **Objeto String**
   - Representa cadenas de texto.
   - Incluye métodos para manipular texto: concatenar, dividir, buscar, reemplazar, y convertir entre mayúsculas/minúsculas.
4. **Objeto Date**
   - Representa fechas y horas.
   - Ofrece métodos para obtener y configurar datos de fecha como año, mes, día, hora, minutos y segundos.
5. **Objeto Math**
   - Proporciona constantes y funciones matemáticas: `PI`, `E`, `sen()`, `cos()`, `sqrt()`, `abs()`, `pow()`, etc.
6. **Objeto Array**
   - Representa listas ordenadas de valores.
   - Métodos para manipular arrays: agregar, eliminar, buscar, insertar, y ordenar elementos.
7. **Objeto Object**
   - Objeto base para todos los demás objetos en JavaScript.
   - Métodos y propiedades para crear y gestionar objetos.
8. **Objeto Error**
   - Representa errores en JavaScript.

- Proporciona propiedades para acceder al mensaje de error, nombre del archivo, y número de línea.

9. **Objeto JSON**
   - Facilita la interpretación y generación de JSON (JavaScript Object Notation), un formato de datos ligero.
10. **Objeto XMLHttpRequest**

- Permite realizar solicitudes HTTP y obtener datos de servidores web.

Window

El objeto window en JavaScript es un objeto global que representa la ventana del navegador y proporciona acceso a sus propiedades y métodos. Es uno de los objetos más importantes en JavaScript y se utiliza para interactuar con la ventana del navegador, obtener información sobre la página web actual, manipular elementos HTML y controlar el comportamiento del navegador.

Propiedades:

```
// 1. location: Proporciona información sobre la URL actual de la página web y permite redirigir a otras páginas.
console.log(window.location.href); // Imprime la URL actual
window.location.href = "https://www.example.com"; // Redirige a otra página

// 2. document: Representa el documento HTML de la página web, proporcionando acceso a sus elementos y métodos (DOM - Document Object Model).
let titulo = document.getElementById("tituloPagina");
titulo.textContent = "Nuevo título de la página"; // Cambia el texto del título

// 3. history: Proporciona acceso al historial del navegador, permitiendo navegar entre las páginas visitadas.
window.history.back(); // Navega a la página anterior
window.history.go(2);  // Navega dos páginas hacia atrás

// 4. screen: Proporciona información sobre las dimensiones de la pantalla, como la resolución.
let anchoPantalla = window.screen.width;
console.log(`Ancho de la pantalla: ${anchoPantalla} pixeles`); // Imprime el ancho de pantalla

// 5. navigator: Proporciona información sobre el navegador utilizado, como el nombre, la versión y el sistema operativo.
let navegador = window.navigator.userAgent;
```

```
console.log(`Navegador: ${navegador}`); // Imprime la información del navegador
```

Métodos:

```
// 1. alert(): Muestra un cuadro de diálogo emergente con un mensaje y un botón de "Aceptar".
window.alert("¡Hola, mundo!"); // Muestra una alerta al usuario

// 2. prompt(): Muestra un cuadro de diálogo emergente para obtener información del usuario.
let nombre = window.prompt("Dime un nombre:");
console.log("Nombre:", nombre); // Imprime el nombre introducido o `null` si se canceló

// 3. confirm(): Muestra un cuadro de diálogo emergente para confirmar una acción, con "Aceptar" y "Cancelar".
let confirmar = window.confirm("¿Desea continuar?");
if (confirmar) {
        console.log("Usuario confirmó la acción");
} else {
        console.log("Usuario canceló la acción");
}

// 4. open(): Abre una nueva ventana del navegador con la URL especificada.
window.open("https://www.example.com", "_blank"); // Abre en una nueva pestaña

// 5. close(): Cierra la ventana actual o una ventana abierta con open.
window.close(); // Cierra la ventana actual (si fue abierta mediante JavaScript)
```

String

En JavaScript, los datos de tipo texto se almacenan como cadenas. **No hay un tipo char para un solo carácter**. Los strings se representan mediante texto entrecomillado, existen 3 tipos de entrecomillado:

- **Comillas dobles** o **comillas simples**: Permiten representar únicamente texto y en una sola línea.
- Los **backticks**: Además de representar las cadenas de texto, nos permiten incrustar cualquier expresión en la cadena, envolviéndola en `${…}`. También permiten utilizar un formato multilínea.

```
let nombre = "Juan";
let saludo = `Hola, ${nombre}`;
console.log(saludo); // Imprime "Hola, Juan"
```

Creación de cadenas de texto:

- Literales de cadena:

```
let saludo = "Hola, mundo!";
let nombre = 'Juan';
```

- Constructor String():

```
let nombre = new String(‘Juan’);
```

Propiedades:

```
// 1. length: Devuelve la longitud de la cadena
let frase = "JavaScript es un lenguaje de programación";
console.log(`Longitud de la frase: ${frase.length}`);
```

Métodos:

```
// 1. charAt(índice): Devuelve el carácter en la posición especificada por el
índice.
let texto = "ABCDEFGHIJKLMN";
console.log(texto.charAt(5)); // Imprime "F" (carácter en la posición 5)

// 2. charCodeAt(índice): Devuelve el código Unicode del carácter en la
posición especificada por el índice.
let letra = "ñ";
console.log(letra.charCodeAt(0)); // Imprime "165" (código Unicode de la letra
"ñ")

// 3. indexOf(subcadena, inicio): Busca la primera aparición de una subcadena,
devolviendo la posición de inicio.
let frase = "Esta frase contiene la palabra programación";
let indice = frase.indexOf("programación");
if (indice !== -1) {
  console.log(`programación está en la posición: ${indice}`);
} else {
  console.log("programación no se encuentra en la frase");
```

```
}

// 4. lastIndexOf(subcadena, inicio): Busca la última aparición de una
subcadena, devolviendo la posición de inicio.
let texto2 = "Repetir la palabra palabra";
console.log(texto2.lastIndexOf("palabra")); // Imprime la posición de la
última aparición de "palabra"

// 5. replace(subcadena, reemplazo): Reemplaza todas las apariciones de una
subcadena por una cadena de reemplazo.
let texto3 = "Hola a todos, soy un robot";
let textoReemplazado = texto3.replace("robot", "humano");
console.log(textoReemplazado); // Imprime "Hola a todos, soy un humano"

// 6. toUpperCase() y toLowerCase(): Convierten la cadena a mayúsculas y
minúsculas.
let saludo = "Hola Mundo";
console.log(saludo.toUpperCase()); // Imprime "HOLA MUNDO"
console.log(saludo.toLowerCase()); // Imprime "hola mundo"

// 7. trim(): Elimina los espacios en blanco al inicio y al final de la
cadena.
let texto4 = "   Hola Mundo   ";
console.log(texto4.trim()); // Imprime "Hola Mundo"

// 8. concat(cadena2, ...): Concatena dos o más cadenas.
let nombre = "Juan";
let apellido = "Pérez";
let nombreCompleto = nombre.concat(" ", apellido);
console.log(nombreCompleto); // Imprime "Juan Pérez"

// 9. slice(inicio, fin): Extrae una subcadena desde una posición inicial
hasta una posición final especificada.
let frase2 = "JavaScript es un lenguaje de programación";
let subcadena = frase2.slice(10, 25);
console.log(subcadena); // Imprime "es un lenguaje"

// 10. split(): Divide una cadena de texto en una lista de subcadenas, basado
en un separador especificado.

        // Ejemplo 1: Dividir una cadena por espacios
        let frase3 = "Esta frase se divide por espacios";
        let palabras = frase3.split(" ");
        console.log(palabras); // Imprime ["Esta", "frase", "se", "divide",
"por", "espacios"]
```

```
        // Ejemplo 2: Dividir una cadena por comas
        let lista = "rojo,verde,azul,amarillo";
        let colores = lista.split(",");
        console.log(colores); // Imprime ["rojo", "verde", "azul", "amarillo"]

        // Ejemplo 3: Dividir una cadena con un límite especificado
        let cadenaLarga = "Lorem ipsum dolor sit amet, consectetur adipiscing
elit";
        let partes = cadenaLarga.split(" ", 5); // Límite de 5 subcadenas
        console.log(partes); // Imprime las primeras 5 palabras de la cadena

// Ejemplo 4: Obtener caracteres individuales
        let texto5 = "Hola, mundo!";
        let caracteres = texto5.split("");
        console.log(caracteres); // Imprime un array con cada carácter: ["H",
"o", "l", "a", ",", " ", "m", "u", "n", "d", "o", "!"]
```

Date

El objeto Date en JavaScript representa un punto específico en el tiempo. Se utiliza para trabajar con fechas y horas, permitiendo obtener la fecha actual, crear fechas futuras o pasadas.

Creación de objetos Date:

- **Sin argumentos**: Crea un objeto Date con la fecha y hora actual según el reloj del sistema.

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

- Con argumentos: Permite crear un objeto Date con una fecha y hora específicas especificando valores numéricos para el año, mes, día, hora, minutos, segundos y milisegundos.

```
let fechaEspecifica = new Date(2024, 5, 12, 10, 30, 15, 500);
console.log(fechaEspecifica);
```

- Con cadena de fecha: Permite crear un objeto Date a partir de una cadena de texto que represente una fecha y hora en un formato específico.

```
let fechaCadena = "2024-06-12T10:30:15.500+02:00";
let fechaObjeto = new Date(fechaCadena);
console.log(fechaObjeto);
```

Métodos get del objeto Date

```
// 1. getFullYear(): Obtiene el año de la fecha como un número entero de cuatro dígitos.
let fecha = new Date();
let año = fecha.getFullYear();
console.log(`Año actual: ${año}`); // Ejemplo: 2024

// 2. getMonth(): Obtiene el mes de la fecha como un número entero de 0 a 11, donde 0 representa enero y 11 diciembre.
let mes = fecha.getMonth();
console.log(`Mes actual: ${mes + 1}`); // Ejemplo: 6 (junio)

// 3. getDate(): Obtiene el día del mes de la fecha como un número entero de 1 a 31.
let dia = fecha.getDate();
console.log(`Día actual: ${dia}`); // Ejemplo: 12

// 4. getHours(): Obtiene la hora de la fecha como un número entero de 0 a 23.
let hora = fecha.getHours();
console.log(`Hora actual: ${hora}`); // Ejemplo: 13

// 5. getMinutes(): Obtiene los minutos de la fecha como un número entero de 0 a 59.
let minutos = fecha.getMinutes();
console.log(`Minutos actuales: ${minutos}`); // Ejemplo: 26

// 6. getSeconds(): Obtiene los segundos de la fecha como un número entero de 0 a 59.
let segundos = fecha.getSeconds();
console.log(`Segundos actuales: ${segundos}`); // Ejemplo: 12

// 7. getMilliseconds(): Obtiene los milisegundos de la fecha como un número entero de 0 a 999.
let milisegundos = fecha.getMilliseconds();
console.log(`Milisegundos actuales: ${milisegundos}`);
```

Métodos set del objeto Date

```
// 1. setFullYear(año): Establece el año de la fecha en el valor entero especificado.
let fecha = new Date();
fecha.setFullYear(2023); // Establece el año a 2023
console.log(fecha.getFullYear()); // Imprime: 2023

// 2. setMonth(mes): Establece el mes de la fecha en el valor especificado (de 0 a 11, donde 0 representa enero y 11 diciembre).
fecha.setMonth(4); // Establece el mes a mayo (mes 4)
console.log(fecha.getMonth() + 1); // Imprime: 5 (mayo)

// 3. setDate(día): Establece el día del mes de la fecha en el valor entero especificado (de 1 a 31).
fecha.setDate(10); // Establece el día a 10
console.log(fecha.getDate()); // Imprime: 10

// 4. setHours(hora): Establece la hora de la fecha en el valor entero especificado (de 0 a 23).
fecha.setHours(18); // Establece la hora a 18
console.log(fecha.getHours()); // Imprime: 18

// 5. setMinutes(minutos): Establece los minutos de la fecha en el valor entero especificado (de 0 a 59).
fecha.setMinutes(30); // Establece los minutos a 30
console.log(fecha.getMinutes()); // Imprime: 30

// 6. setSeconds(segundos): Establece los segundos de la fecha en el valor entero especificado (de 0 a 59).
fecha.setSeconds(15); // Establece los segundos a 15
console.log(fecha.getSeconds()); // Imprime: 15

// 7. setMilliseconds(milisegundos): Establece los milisegundos de la fecha en el valor entero especificado (de 0 a 999).
fecha.setMilliseconds(500); // Establece los milisegundos a 500
console.log(fecha.getMilliseconds()); // Imprime: 500
```

Number y Math

Para **trabajar con números y funciones matemáticas** disponemos de dos objetos: Number y Math.

Number:

El objeto Number en JavaScript representa valores numéricos, tanto enteros como decimales.

```
// Propiedades de Number:
console.log(Number.MAX_VALUE); // Valor numérico más grande posible
console.log(Number.MIN_VALUE); // Valor numérico más pequeño posible
console.log(Number.POSITIVE_INFINITY); // Infinito positivo
console.log(Number.NEGATIVE_INFINITY); // Infinito negativo
console.log(Number.NaN); // Valor no numérico ("Not a Number")

// Métodos principales:
const numero = 123.4567;
console.log(numero.toString()); // Convierte a cadena de texto
console.log(numero.toFixed(2)); // "123.46" - Convierte a texto con 2
decimales
console.log(numero.toExponential(2)); // Notación exponencial: "1.23e+2"
console.log(numero.valueOf()); // Valor primitivo del número

// Métodos de comprobación:
console.log(Number.isFinite(numero)); // true - Comprueba si es finito
console.log(Number.isNaN('Hola')); // true - Comprueba si es NaN
console.log(Number.isInteger(numero)); // false - Comprueba si es un entero

// Convertir cadena a número:
const cadena = '100';
const numeroConvertido = Number(cadena);
console.log(numeroConvertido); // 100

// Crear un objeto Number con el constructor
const nuevoNumero = new Number(200);
console.log(nuevoNumero); // Imprime: Number {valueOf: 200}
```

Math:

El objeto Math en JavaScript es un objeto global que proporciona constantes y funciones matemáticas. Es una herramienta esencial para trabajar con números en aplicaciones web.

```
// Constantes de Math:
console.log(Math.PI); // Valor de pi (aproximadamente 3.14159)
console.log(Math.E); // Número de Euler (aproximadamente 2.71828)
console.log(Math.LN2); // Logaritmo natural de 2
console.log(Math.LN10); // Logaritmo natural de 10
console.log(Math.SQRT2); // Raíz cuadrada de 2
console.log(Math.SQRT1_2); // Raíz cuadrada de 1/2

// Funciones aritméticas:
console.log(Math.abs(-5)); // Valor absoluto: 5
console.log(Math.pow(2, 3)); // Potencia: 2^3 = 8
```

```
console.log(Math.sqrt(16)); // Raíz cuadrada: 4
console.log(Math.floor(4.7)); // Redondea hacia abajo: 4
console.log(Math.ceil(4.3)); // Redondea hacia arriba: 5
console.log(Math.round(4.5)); // Redondea al más cercano: 5
console.log(Math.random()); // Genera un número aleatorio entre 0 y 1

// Funciones trigonométricas:
console.log(Math.sin(Math.PI / 2)); // Seno de π/2: 1
console.log(Math.cos(Math.PI)); // Coseno de π: -1
console.log(Math.tan(0)); // Tangente de 0: 0
console.log(Math.asin(1)); // Arcoseno de 1: π/2
console.log(Math.acos(0)); // Arcocoseno de 0: π/2
console.log(Math.atan(1)); // Arcotangente de 1: π/4
console.log(Math.atan2(1, 1)); // Arcotangente de 1/1: π/4

// Funciones de logaritmos:
console.log(Math.log(10)); // Logaritmo natural de 10
console.log(Math.log10(100)); // Logaritmo base 10 de 100: 2
console.log(Math.log2(8)); // Logaritmo base 2 de 8: 3
console.log(Math.exp(1)); // Exponencial de 1 (e^1): Math.E

// Funciones de redondeo:
console.log(Math.round(4.5)); // Redondea al más cercano: 5
console.log(Math.floor(4.7)); // Redondea hacia abajo: 4
console.log(Math.ceil(4.3)); // Redondea hacia arriba: 5

// Funciones de mínimo y máximo:
console.log(Math.max(1, 3, 2)); // Máximo: 3
console.log(Math.min(1, 3, 2)); // Mínimo: 1
```

Almacenamiento local

El almacenamiento de datos en JavaScript se maneja principalmente a través de **cookies**, **Web Storage API** (localStorage y sessionStorage) e **IndexedDB**.

1. **Cookies**: Son pequeñas piezas de datos que un servidor envía al navegador. Se almacenan como pares clave-valor y se usan para gestionar sesiones, personalizar contenido y rastrear actividad del usuario. Las cookies se configuran usando `document.cookie` y pueden tener un tiempo de expiración (con `expires` o `max-age`). Las cookies se envían con cada solicitud al servidor, lo que puede impactar el rendimiento, por lo que se recomiendan otras alternativas para almacenamiento.

```
// Creación y manejo de cookies en JavaScript
```

```
// Crear cookies
document.cookie = "nombre=Pepe";
document.cookie = "edad=30";
document.cookie = "ciudad=Valencia";
alert("Cookies guardadas correctamente.");
console.log(document.cookie); // Mostrar todas las cookies

// Configuración de propiedades importantes
// 1. path: Accesibilidad de la cookie en todas las páginas
document.cookie = "ciudad=Valencia; path=/";

// 2. expires y max-age: Controlan la duración de la cookie
// Expira cuando el navegador se cierra si no se establece
// Explicación de expires
let date = new Date(Date.now() + 86400e3); // +1 día
document.cookie = "ciudad=Valencia; expires=" + date.toUTCString();

// Eliminar una cookie estableciendo expires en el pasado
document.cookie = "ciudad=Valencia; expires=Thu, 01 Jan 1970 00:00:00 GMT";

// Explicación de max-age
document.cookie = "nombre=Pepe; max-age=3600"; // Expira en 1 hora
document.cookie = "nombre=Pepe; max-age=0"; // Eliminar cookie inmediatamente
```

3. **Web Storage API**:
    - **sessionStorage**: Almacena datos durante la sesión del navegador; los datos se eliminan al cerrar el navegador.
    - **localStorage**: Almacena datos de forma persistente en el navegador, incluso después de cerrarlo. Usa pares clave-valor, y los métodos más usados son `setItem`, `getItem`, `removeItem` y `clear`.

```
// Uso de LocalStorage en JavaScript

// Almacenamiento local: persistente y basado en pares clave-valor

// 1. Almacenar datos
localStorage.setItem("nombre", "Juan Pérez");
localStorage.setItem("edad", 30);

// 2. Recuperar datos
let nombre = localStorage.getItem("nombre");
console.log(nombre); // Imprime: "Juan Pérez"
let edad = localStorage.getItem("edad");
console.log(edad); // Imprime: 30
```

```
// 3. Eliminar datos
localStorage.removeItem("nombre");
console.log(localStorage.getItem("nombre")); // Imprime: null
localStorage.removeItem("edad");
console.log(localStorage.getItem("edad")); // Imprime: null

// 4. Limpiar todo el almacenamiento local
localStorage.clear();
console.log(localStorage.getItem("nombre")); // Imprime: null
console.log(localStorage.getItem("edad")); // Imprime: null
```

4. **IndexedDB**: Es una API avanzada para almacenar grandes cantidades de datos estructurados en el cliente, como archivos y blobs. Es útil para aplicaciones complejas y permite realizar búsquedas de alto rendimiento usando índices.