Otras estructuras en JavaScript

En JavaScript, además de los tipos de datos primitivos como números, cadenas de texto y booleanos, también existen tipos de datos no primitivos. Estos tipos de datos no primitivos son objetos predefinidos en el lenguaje y proporcionan funcionalidades adicionales. En esta lección, exploraremos algunos de los tipos de datos no primitivos más utilizados en JavaScript.

Fechas

JavaScript proporciona el objeto `Date` para trabajar con fechas y horas. Puedes crear objetos `Date` para representar fechas específicas y utilizar métodos para manipular y formatear las fechas.

```
const fechaActual = new Date(); // Crear una fecha actual
const fechaEspecifica = new Date(2023, 4, 22); // Crear una fecha específica

fechaActual.getFullYear(); // Obtener el año actual
fechaActual.getMonth(); // Obtener el mes actual (0-11)
fechaActual.getDate(); // Obtener el día del mes (1-31)
fechaActual.getHours(); // Obtener las horas (0-23)
fechaActual.getMinutes(); // Obtener los minutos (0-59)
fechaActual.getSeconds(); // Obtener los segundos (0-59)
fechaActual.getDay(); // Obtener el día de la semana (0-6, donde 0 es domingo)

fechaActual.toString(); // Obtener una representación de cadena de la fecha
fechaActual.toLocaleDateString(); // Obtener la fecha en formato localizado
fechaActual.toLocaleTimeString(); // Obtener la hora en formato localizado
```

Math

El objeto `Math` proporciona funciones matemáticas y constantes útiles en JavaScript. Puedes utilizar estas funciones para realizar cálculos matemáticos en tus programas.

```
Math.PI; // Valor de PI
Math.abs(-5); // Valor absoluto (5)
Math.sqrt(25); // Raíz cuadrada (5)
Math.pow(2, 3); // Potencia (2 elevado a la 3, resultando en 8)
Math.random(); // Número aleatorio entre 0 y 1
Math.floor(3.7); // Redondear hacia abajo (3)
Math.ceil(3.2); // Redondear hacia arriba (4)
Math.round(3.5); // Redondear al entero más cercano (4)
```

JSON

JSON (JavaScript Object Notation) es un formato de intercambio de datos ampliamente utilizado en JavaScript. El objeto `JSON` proporciona métodos para trabajar con cadenas JSON y convertir objetos JavaScript en cadenas JSON y viceversa.

```
const persona = {
  nombre: "Juan",
  edad: 30,
  profesion: "Desarrollador"
};

const json = JSON.stringify(persona); // Convertir objeto a cadena JSON
const objeto = JSON.parse(json); // Convertir cadena JSON a objeto JavaScript

console.log(objeto.nombre); // "Juan"
console.log(objeto.edad); // 30
console.log(objeto.profesion); // "Desarrollador"
```

RegExp (Expresiones regulares)

Las expresiones regulares son patrones utilizados para buscar y manipular texto. JavaScript proporciona el objeto `RegExp` para trabajar con expresiones regulares.

```
const texto = "¡Hola, mundo!";
const patron = /mundo/; // Expresión regular para buscar la palabra "mundo"

console.log(patron.test(texto)); // Verificar si el patrón se encuentra en el
console.log(texto.match(patron)); // Obtener la coincidencia encontrada (["mur
console.log(texto.replace(patron, "amigo")); // Reemplazar la coincidencia por
```

File

El objeto `File` representa un archivo seleccionado por el usuario a través de un campo de entrada de archivo (`<input type="file">`). Es utilizado comúnmente al trabajar con la carga y manipulación de archivos en aplicaciones web.

```
const fileInput = document.getElementById("miArchivo"); // Supongamos que tene

const archivo = fileInput.files[0]; // Obtener el archivo seleccionado

console.log(archivo.name); // Nombre del archivo
console.log(archivo.size); // Tamaño del archivo en bytes
console.log(archivo.type); // Tipo MIME del archivo
console.log(archivo.lastModified); // Fecha de la última modificación del arch
```

URL

El objeto `URL` proporciona métodos para trabajar con URLs (Uniform Resource Locators) en JavaScript. Puedes utilizar estos métodos para extraer información específica de una URL y manipularla.

```
const url = new URL("https://www.ejemplo.com/ruta?parametro=valor");

console.log(url.href); // URL completa ("https://www.ejemplo.com/ruta?parametr
console.log(url.host); // Nombre de host ("www.ejemplo.com")
console.log(url.pathname); // Ruta de la URL ("/ruta")
console.log(url.search); // Cadena de búsqueda ("?parametro=valor")
console.log(url.searchParams.get("parametro")); // Obtener el valor del paráme
```

Blob

El objeto `Blob` representa datos binarios brutos y se utiliza para trabajar con contenido binario, como archivos. Puedes crear objetos `Blob` a partir de datos existentes o construirlos desde cero utilizando un `ArrayBuffer` .

```
const arrayBuffer = new ArrayBuffer(4);
const uint8Array = new Uint8Array(arrayBuffer);

uint8Array[0] = 72;
uint8Array[1] = 101;
uint8Array[2] = 108;
uint8Array[3] = 108;

const blob = new Blob([uint8Array], { type: "application/octet-stream" });

console.log(blob.size); // Tamaño del blob en bytes (4)
console.log(blob.type); // Tipo MIME del blob ("application/octet-stream")
```

Set

El objeto `Set` es una colección de valores únicos. Puedes utilizar métodos para agregar, eliminar y verificar la existencia de elementos en un conjunto.

```
const set = new Set();

set.add(1);
set.add(2);
set.add(3);
set.add(2); // No se agregarán duplicados

console.log(set.size); // 3
console.log(set.has(2)); // true

set.delete(2);
console.log(set.has(2)); // false
```

Map

El objeto `Map` permite almacenar pares clave-valor, donde cada clave es única. Puedes utilizar métodos para agregar, obtener y eliminar elementos en un mapa.

```
const map = new Map();

map.set("nombre", "Juan");
map.set("edad", 30);
map.set("profesion", "Desarrollador");

console.log(map.get("nombre")); // "Juan"
console.log(map.has("edad")); // true

map.delete("edad");
console.log(map.has("edad")); // false
```

Promise

Las promesas (`Promise`) se utilizan para manejar operaciones asíncronas en JavaScript. Puedes encadenar métodos `then` y `catch` para trabajar con los resultados o errores de una promesa.

```
const fetchData = new Promise((resolve, reject) => {
  // Lógica para obtener datos asincrónicamente
  if (datosObtenidos) {
    resolve(datosObtenidos);
  } else {
    reject("Error al obtener los datos");
  }
});

fetchData
  .then((datos) => {
    // Trabajar con los datos obtenidos
    console.log(datos);
  })
  .catch((error) => {
    // Manejar errores
    console.error(error);
  });
```

Estos son algunos de los tipos de datos no primitivos más utilizados en JavaScript. Familiarizarse con ellos ampliará tus habilidades de programación y te permitirá aprovechar al máximo las capacidades del lenguaje en el desarrollo web.