

Arrays y elementos predefinidos del lenguaje

Contenido

Arrays	2
Recorrer arrays.....	3
Métodos en arrays	3
Búsquedas	3
Ordenar	4
Otros métodos	5
Objetos predefinidos en JavaScript.....	7
Propiedades predefinidas	7
Funciones predefinidas	7
Funciones Temporizadores	9
Objetos predefinidos.....	9
String	9
Math	11
Date	12
JSON	14
Colecciones.....	15
Otros predefinidos	16
Bibliografía	17

Arrays

Un array en JavaScript es una estructura de datos que permite almacenar una colección de elementos, los cuales pueden ser de cualquier tipo (números, cadenas, objetos, etc.). Los arrays son dinámicos, lo que significa que su tamaño y contenido pueden cambiar durante la ejecución del programa.

```
let numeros = [];  
numeros = [1, 2, 3, 4, 5];  
numeros[0]; // 1  
numeros[2] = 33; // [1, 2, 33, 4, 5]  
numeros[numeros.length - 1]; // 5  
numeros.at(-1); // 5, sólo lectura  
  
numeros = new Array(4); // Instancio un array de 4 posiciones  
console.log(numeros); // [undefined, undefined, undefined, undefined]  
console.log(numeros.length); // 4  
  
numeros = new Array(1, 2);  
console.log(numeros); // [1, 2]
```

Los arrays únicamente disponen de una propiedad “`.length`” que devuelve el tamaño del array.

RECUERDA: “`new Array(tamaño)`” crea un array del tamaño indicado.

RECUERDA: “`.at(idx)`” no permite modificar el elemento.

Arrays multidimensionales

Para crear **arrays multidimensionales** debemos emplear Arrays de Arrays, es decir, el elemento de un array es otro array, con esto simulamos una nueva dimensión. Por ejemplo, un array de 3 filas y 3 columnas.

```
//multidimensional  
let matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];  
console.log(matriz[1][1]); // 5
```

Copia de arrays

El operador “`...`”, también conocido como operador “**spread**” permite crear una copia rápida de un array, ahora si modificamos la copia los originales permanecen intactos. Por ejemplo.

```
let numeros2 = [...numeros];  
numeros2[0] = 11;  
console.log(numeros); // [1, 2]  
console.log(numeros2); // [11, 2]
```

Recorrer arrays

Disponemos de varias maneras de recorrer un array, mediante operadores y mediante métodos.

Operadores

- **for**: bucle que permite iterar mediante un índice.
- **for..of**: bucle que devuelve cada valor de la colección.
- **for..in**: pensado para recorrer propiedades, aplicado a un array devuelve el índice.

Métodos

- **.forEach()**: similar al bucle for..of
- **.map()**: devuelve una copia del array con las operaciones realizadas a cada elemento.

IMPORTANTE: los métodos no son operadores, no están pensado para usarse con “break” o “continue”. Su empleo produce una excepción “*illegal break statement*”, dicho esto se podría emplear “return” como alternativa a “continue”.

Por ejemplo.

```
numeros = [1, 2];

for (let i = 0; i < numeros.length; i++) {
  console.log(numeros[i]);
}

for (let numero of numeros) {
  console.log(numero);
}

numeros.forEach(numero => console.log(numero));

let resultado = numeros.map(numero => numero * 2);
console.log(resultado); // [2, 4]
console.log(numeros); // [1, 2]
```

Métodos en arrays

IMPORTANTE, tienes que tener en cuenta que en los arrays hay métodos cuyo uso modifican los datos del array (**métodos mutables**) y otros métodos cuyo uso devuelven una copia de los datos (**métodos no mutables**).

Hay muchísimos métodos disponibles de ambos tipos, a continuación, se van a resumir los que considero de mayor utilidad, dicho lo cual, te invito a que revises la lista.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#instance_methods

Búsquedas

Los principales métodos para buscar son:

- **.indexOf(valor, inicio?):** Devuelve el índice del elemento, -1 si no lo encuentra. Admite un segundo parámetro para indicar la posición de comienzo de la búsqueda.
- **.filter(función):** Devuelve un nuevo array con los elementos que cumplen la condición.

Por ejemplo.

```
let cosas = ['uno', 2, 'tres', 4, 'cinco'];

let busqueda = cosas.indexOf('tres');
console.log(busqueda); //devuelve el indice 2
let numeros = cosas.filter(ele => typeof ele === 'number');
console.log(numeros); //devuelve [2, 4]

// os gustan pero no son esenciales
let indiceNumero = cosas.findIndex(ele => typeof ele === 'number');
console.log(indiceNumero); // devuelve el indice 1
let primerNumero = cosas.find(ele => typeof ele === 'number');
console.log(primerNumero); // devuelve 2
```

Ordenar

Disponemos de dos métodos para ordenar un array:

- **.sort():** Mutable, ordena sobre el propio array modificándolo.
- **.toSorted():** No mutable, devuelve una copia ordenada del array.

Por ejemplo.

```
//Ordenación
let original = [4, 55, 5, 1, 11, 'tres', 'dos', 7];
let ordenados = original.toSorted();
//Devuelve [1, 11, 4, 5, 55, 7, "dos", "tres"]

let copia = [...original];
copia.sort();
//Devuelve [1, 11, 4, 5, 55, 7, "dos", "tres"]

console.log(original); //Devuelve [4, 55, 5, 1, 11, 'tres', 'dos', 7]
console.log(ordenados); //Devuelve [1, 11, 4, 5, 55, 7, "dos", "tres"]
console.log(copia); //Devuelve [1, 11, 4, 5, 55, 7, "dos", "tres"]
```

Criterios de ordenación personalizados

Es posible que necesitemos indicar el criterio de ordenación, supón por ejemplo que trabajamos con objetos de tipo “Persona” y queremos ordenar por edad y luego por apellido. En este caso las funciones “.sort()” y “.toSorted()” no van a ser capaces de resolver el problema.

Las funciones de ordenación admiten como parámetro una función que resuelva que elemento es mayor que otro. Esta función se parece a “(a, b) => return a-b;”, donde a es el primer

parámetro y b el segundo parámetro, si a va antes que b devolverá un valor negativo, en caso contrario devolverá un valor positivo. En el caso de que a y b sean iguales devolverá 0.

Por ejemplo, devolver las cadenas antes que los números.

```
//Ordenar con criterio, cadenas antes de números
ordenados = original.toSorted((a, b) => {
  if (typeof a === 'string' && typeof b === 'number') {
    return -1;
  }
  if (typeof a === 'number' && typeof b === 'number') {
    return a - b;
  } else {
    //return a.toString().localeCompare(b.toString());
    return a.toString() === b.toString();
  }
});

console.log(ordenados); //Devuelve ["dos", "tres", 1, 4, 5, 7, 11, 55]
```

Otros métodos

La lista de métodos disponibles es muy extensa,

- **.push(...elementos)**: Añade uno o más elementos al final del array.
- **.pop()**: Elimina el último elemento del array y lo devuelve.
- **.shift()**: Elimina el primer elemento del array y lo devuelve.
- **.unshift(...elementos)**: Añade uno o más elementos al principio del array.
- **.join(separador)**: Devuelve una cadena con todos los elementos del array unidos mediante el valor del separador.
- **.split(separador)**: Divide una cadena según el separador, devuelve un array con las subcadenas.
- **.concat(...arrays)**: Combina dos o más arrays en un nuevo array.
- **.flat(profundidad=1)**: Aplana arrays anidados en un nuevo array hasta la profundidad indicada.
- **.fill(valor, inicio?, fin?)**: Llena el array con el valor indicado desde los índices de inicio hasta fin.
- **.reverse()**: Invierte el array.
- **.toReversed()**: Devuelve una copia del array invertido.
- **.slice(inicio, fin?)**: Devuelve una copia de una porción del array con los valores indicados en el rango.
- **.splice(inicio, númeroElementosEliminar)**: Elimina a partir de la posición de inicio el número de elementos indicados.
- **.splice(inicio, númeroElementosEliminar, elementoInsertar1, ...elementoInsertarN)**: Intercambia a partir de la posición de inicio los elementos a insertar indicados, si el número de elementos a eliminar es 0 se insertan los elementos en la posición indicada de inicio.

Por ejemplo.

```

let array = [1, 2, 3];
let resultado;

// 1. push: Añadir elementos al final
resultado = array.push(4, 5);
console.log(array); // [1, 2, 3, 4, 5]
console.log(resultado); // 5 (nueva longitud)

// 2. pop: Eliminar el último elemento
resultado = array.pop();
console.log(array); // [1, 2, 3, 4]
console.log(resultado); // 5 (elemento eliminado)

// 3. shift: Eliminar el primer elemento
resultado = array.shift();
console.log(array); // [2, 3, 4]
console.log(resultado); // 1 (elemento eliminado)

// 4. unshift: Añadir elementos al principio
resultado = array.unshift(0);
console.log(array); // [0, 2, 3, 4]
console.log(resultado); // 4 (nueva longitud)

// 5. join: Convertir el array en una cadena
let cadena = array.join('-');
console.log(cadena); // "0-2-3-4"

// 6. split: Convertir una cadena en un array
let nuevoArray = cadena.split('-');
console.log(nuevoArray); // ["0", "2", "3", "4"]

// 7. concat: Combinar arrays
let array2 = [5, 6];
let combinado = array.concat(array2);
console.log(combinado); // [0, 2, 3, 4, 5, 6]

// 8. flat: Aplanar un array anidado
let arrayAnidado = [1, [2, [3, 4]]];
let aplanado = arrayAnidado.flat(2);
console.log(aplanado); // [1, 2, 3, 4]

// 9. fill: Llenar un array con un valor
array.fill(9, 1, 3);
console.log(array); // [0, 9, 9, 4]

// 10. reverse: Invertir el array
array.reverse();
console.log(array); // [4, 9, 9, 0]

```

```
// 11. toReversed: Crear una copia invertida del array (sin modificar el original)
let copiaInvertida = array.toReversed();
console.log(copiaInvertida); // [0, 9, 9, 4]
console.log(array); // [4, 9, 9, 0] (el array original no cambia)

// 12. slice: Extraer una porción del array
let subArray = array.slice(1, 3);
console.log(subArray); // [9, 9]

// 13. splice: Eliminar elementos con splice
array = [1, 2, 3, 4, 5];
let eliminados = array.splice(2, 2); // Elimina 2 elementos a partir del índice 2
console.log(array); // [1, 2, 5]
console.log(eliminados); // [3, 4]

// 14. Insertar elementos con splice
array = [1, 2, 5]; // Reiniciamos el array
array.splice(2, 0, 3, 4); // Inserta 3 y 4 en la posición 2 sin eliminar elementos
console.log(array); // [1, 2, 3, 4, 5]
```

Objetos predefinidos en JavaScript

El lenguaje JavaScript tiene implementado una colección de elementos (variables, funciones, objetos, colecciones, etc...) que nos facilitan las labores de codificación. Adicionalmente esta colección es ampliada por los navegadores o por los entornos de ejecución.

En los siguientes apartados vamos a presentar los elementos más interesantes, como siempre, recuerda revisar la documentación para ver todas las opciones disponibles

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

Propiedades predefinidas

Representan valores especiales.

- **globalThis**: Versión más potente de "this".
- Infinity
- NaN
- undefined

Funciones predefinidas

Estas funciones se llaman a nivel global y devuelven directamente el resultado de su ejecución.

- **eval(cadena)**: Evalúa una cadena como código JavaScript. **NO SE RECOMIENDA SU USO POR MOTIVOS DE SEGURIDAD**, dicho lo cual es una maravilla.

- **isFinite(valor)**: Devuelve true si el valor es finito, false en caso contrario o NaN.
- **isNaN(valor)**: Devuelve true si el valor es NaN (Not a Number). Muy útil para comprobar números después de parsear una entrada.
- **parseFloat(cadena)**: Convierte una cadena en un número flotante.
- **parseInt(cadena, base=10)**: Convierte una cadena en un número entero. Opcionalmente se puede indicar la base para la cadena de entrada, por defecto 10.
- **decodeURI(uri)**: Decodifica una URI codificada previamente con “encodeURIComponent”, convirtiendo las secuencias de escape “%xx” a caracteres especiales.
- **encodeURIComponent(uri)**: Codifica una URI, reemplazando ciertos caracteres por sus equivalentes de escape “%xx”.

Por ejemplo.

```
//1. eval
let expresion = '2 + 2';
console.log(eval(expresion)); // 4

//2. isFinite
console.log(isFinite(10)); // true
console.log(isFinite(Infinity)); // false
console.log(isFinite('100')); // true (cadena convertida a número)

//3. isNaN
console.log(isNaN(10)); // false (10 es un número)
console.log(isNaN(NaN)); // true (NaN)
console.log(isNaN('100')); // false (cadena convertida a número 100)
console.log(isNaN('abc')); // true (cadena no numérica -> NaN)
console.log(isNaN('')); // false (cadena vacía -> 0)
console.log(isNaN(' ')); // false (solo espacios -> 0)
console.log(isNaN(undefined)); // true (undefined -> NaN)

//4. parseFloat
console.log(parseFloat('3.14')); // 3.14
console.log(parseFloat('10abc')); // 10
console.log(parseFloat('abc10')); // NaN

//5. parseInt
console.log(parseInt('42')); // 42
console.log(parseInt('101', 2)); // 5 (binario a decimal)
console.log(parseInt('abc')); // NaN

//6. decodeURI
let uri = 'https://example.com?name=John%20Doe';
console.log(decodeURI(uri)); // 'https://example.com?name=John Doe'

//7. encodeURIComponent
uri = 'https://example.com?name=John Doe';
console.log(encodeURIComponent(uri)); // 'https://example.com?name=John%20Doe'
```


Funciones Temporizadores

Estas funciones no son propias del lenguaje JavaScript ya que las incluyen los navegadores, debido a su importancia las vamos a incluir aquí.

- **setTimeout(función, tiempo)**: Ejecuta la función indicada después de un tiempo específico. El tiempo se indica en milisegundos. Devuelve el ID del temporizador creado.
- **clearTimeout(id)**: Cancela un temporizador creado mediante “setTimeout”.
- **setInterval(función, tiempo)**: Ejecuta la función indicada de forma repetitiva según el tiempo indicado. El tiempo se indica en milisegundos. Devuelve el ID del temporizador creado.
- **clearInterval(id)**: Detiene la ejecución de una función repetitiva establecida mediante “setInterval”.

Por ejemplo.

```
//Temporizadores
//1. setTimeout
let timer = setTimeout(() => console.log('Hola después de 2 segundos'),
2000);

//2. clearTimeout
clearTimeout(timer);

//3. setInterval
let intervalo = setInterval(() => console.log('Hola cada 3 segundos'),
3000);

//4. clearInterval
clearInterval(intervalo);
```

Objetos predefinidos

A continuación, vamos a presentar los objetos de uso común, recuerda que la lista es amplia.

NOTA: A continuación voy a presentar métodos de los tipos primitivos y de las clases envoltorio indistintamente, con lo que nos salimos de los elementos predeterminados.

String

Los siguientes métodos son no mutables, devuelven un nuevo dato.

1. **.charAt(posición)**: Devuelve el carácter en una posición específica dentro de la cadena.
2. **.concat(...cadenas)**: Combina una o más cadenas y las une a la cadena original.
3. **.indexOf(subcadena, posiciónInicio=0)**: Devuelve el índice de la primera aparición de una subcadena, -1 si no se encuentra.
4. **.includes(subcadena, posiciónInicio = 0)**: Devuelve true si la subcadena se encuentra incluida en la cadena, falso en caso contrario.
5. **.slice(inicio, fin)**: Extrae una porción de la cadena entre los índices inicio y fin.

6. **.replace(subcadena, nuevaSubcadena)**: Reemplaza la primera aparición de una subcadena con otra.
7. **.split(separador)**: Divide una cadena en un array de subcadenas. Devuelve un array de subcadenas.
8. **.toLowerCase()**: Convierte la cadena a minúsculas.
9. **.toUpperCase()**: Convierte la cadena a mayúsculas.
10. **.trim()**: Elimina los espacios en blanco al inicio y al final de la cadena.
11. **.localeCompare(cadena, locales?)**: Compara dos cadenas según las convenciones del idioma especificado. Los locales son opcionales, por defecto recupera los de la página. Devuelve -1 si la cadena original es menor, 0 si son iguales y 1 si la cadena original es mayor.

Por ejemplo

```
//String
let texto = ' Bienvenido a JavaScript ';
```



```
// 9. Eliminar espacios al inicio y al final
let textoLimpio = texto.trim(); // 'Bienvenido a JavaScript'
```



```
// 8. Convertir a minúsculas
let textoMinis = textoLimpio.toLowerCase(); // 'bienvenido a javascript'
```



```
// 9. Convertir a mayúsculas
let textoMayus = textoLimpio.toUpperCase(); // 'BIENVENIDO A JAVASCRIPT'
```



```
// 7. Dividir en palabras
let palabras = textoMayus.split(' '); // ['BIENVENIDO', 'A',
'JAVASCRIPT']
```



```
// 6. Reemplazar 'JavaScript' por 'el mundo de la programación'
let textoFinal = textoLimpio.replace('JavaScript', 'el mundo de la
programación');
console.log(textoFinal); // 'Bienvenido a el mundo de la programación'
```



```
// 1. Obtener el carácter en la posición 0
console.log(textoLimpio.charAt(0)); // 'B'
```



```
// 2. Combinar cadenas
let nuevaCadena = textoLimpio.concat(' ', 'es increíble');
console.log(nuevaCadena); // 'Bienvenido a JavaScript es increíble'
```



```
// 3. Obtener índice de 'JavaScript'
console.log(textoLimpio.indexOf('JavaScript')); // 13
```



```
// 4. Comprobar si contiene una subcadena
console.log(textoLimpio.includes('JavaScript')); // true
console.log(textoLimpio.includes('Python')); // false
```

```
// 5. Extraer una parte de la cadena
let parte = textoLimpio.slice(0, 10);
console.log(parte); // 'Bienvenido'

// 11. Comparar cadenas usando localeCompare
let cadena1 = 'mañana';
let cadena2 = 'manana';
console.log(cadena1.localeCompare(cadena2, 'es')); // Comparación según
el idioma español
```

Math

El objeto Math incluye propiedades y métodos.

Propiedades.

- **Math.PI**: Constante PI
- **Math.E**: Constante de Euler
- **Math.LN10**: Leperiano en base 10
- **Math.LN2**: Leperiano en base 2
- **Math.LOG10E**: Logaritmo en base 10 de E.
- Etc...

Métodos de redondeo.

- **Math.ceil(x)**: Redondea un número hacia arriba al entero más cercano.
- **Math.floor(x)**: Redondea un número hacia abajo al entero más cercano.
- **Math.round(x)**: Redondea un número al entero más cercano.

Por ejemplo.

```
console.log(Math.ceil(4.2)); // 5
console.log(Math.floor(4.8)); // 4
console.log(Math.round(4.5)); // 5
```

Otros métodos.

- **Math.abs(x)**: Devuelve el valor absoluto de un número.
- **Math.max(...valores)**: Devuelve el mayor número de una lista de valores.
- **Math.min(...valores)**: Devuelve el menor número de una lista de valores.
- **Math.pow(base, exponente)**: Eleva un número a una potencia.
- **Math.sqrt(x)**: Devuelve la raíz cuadrada de un número.
- **Math.trunc(x)**: Elimina los decimales de un número, devolviendo solo la parte entera.
- **Math.random()**: Devuelve un número pseudoaleatorio entre 0 (inclusive) y 1 (exclusivo).

Por ejemplo.

```
console.log(Math.abs(-5)); // 5
console.log(Math.max(1, 5, 10)); // 10
```

```
console.log(Math.min(1, 5, 10)); // 1
console.log(Math.pow(2, 3)); // 8
console.log(Math.sqrt(16)); // 4
console.log(Math.trunc(4.9)); // 4
console.log(Math.random()); // Ejemplo: 0.726
```

Date

Los objetos Date representan un momento fijo de tiempo. Internamente almacenan los milisegundos transcurridos desde el 1 de enero de 1970 UTC (Universal Time Coordinated), ten en cuenta que el uso horario de España es UTC+01:00.

IMPORTANTE: no debes preocuparte por el uso horario ya que los métodos básicos están preparados para trabajar con la zona horaria local.

Crear una fecha.

- **new Date():** Crea un nuevo objeto Date con la fecha y hora actuales.
- **new Date(valor):** Crea un objeto Date usando un valor de milisegundos desde el 1 de enero de 1970 (Epoch Unix).
- **new Date(cadenaFecha):** Crea un objeto Date usando una cadena de texto que representa una fecha (formato "YYYY-MM-DD HH:mm:ss.sss" o compatible, lo mínimo YYYY).
- **new Date(año, mes=1, día=1, horas=0, minutos=0, segundos=0, milisegundos=0):** Crea un objeto Date especificando cada parte de la fecha. **Los meses se cuentan desde 0** (enero = 0, diciembre = 11).

Por ejemplo.

```
let ahora = new Date();
console.log(ahora); // Fecha y hora actual
```

```
let fechaEpoch = new Date(0);
console.log(fechaEpoch); // Thu Jan 01 1970 01:00:00 GMT+0100
```

```
let fechaTexto = new Date('2023-09-30');
console.log(fechaTexto); // Sat Sep 30 2023 00:00:00 GMT+0200
```

```
let fechaEspecifica = new Date(2023, 8, 30, 14, 30); // Septiembre = 8
console.log(fechaEspecifica); // Sat Sep 30 2023 14:30:00 GMT+0200
```

Métodos para obtener información de una fecha.

- **.getFullYear():** Devuelve el año completo.
- **.getMonth():** Devuelve el mes (0 = enero, 11 = diciembre).
- **.getDate():** Devuelve el día del mes (1-31).
- **.getDay():** Devuelve el día de la semana (0 = domingo, 6 = sábado).
- **.getHours():** Devuelve la hora (0-23).
- **.getMinutes():** Devuelve los minutos (0-59).

- **.getSeconds()**: Devuelve los segundos (0-59).
- **.getTime()**: Devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970.

Por ejemplo.

```
//1. getFullYear
let año = ahora.getFullYear();
console.log(año); // 2024
//2. getMonth
let mes = ahora.getMonth();
console.log(mes); // 8 (Septiembre)
//3. getDate
let dia = ahora.getDate();
console.log(dia); // 30
//4. getDay
let diaSemana = ahora.getDay();
console.log(diaSemana); // 6 (Sábado)
//5. getHours
let hora = ahora.getHours();
console.log(hora); // Ejemplo: 16
//6. getMinutes
let minutos = ahora.getMinutes();
console.log(minutos); // Ejemplo: 38
//7. getSeconds
let segundos = ahora.getSeconds();
console.log(segundos); // Ejemplo: 59
//8. getMilliseconds
let tiempo = ahora.getTime();
console.log(tiempo); // Ejemplo: 1696079939000
```

Métodos para modificar una fecha.

- **.setFullYear(año)**: Establece el año de la fecha.
- **.setMonth(mes)**: Establece el mes (0-11).
- **.setDate(día)**: Establece el día del mes (1-31).
- **.setHours(horas)**: Establece la hora (0-23).
- **.setMinutes(minutos)**: Establece los minutos (0-59).

Por ejemplo.

```
//1. setFullYear
ahora.setFullYear(2024);
console.log(ahora.getFullYear()); // 2024
//2. setMonth
ahora.setMonth(11); // Diciembre
console.log(ahora.getMonth()); // 11
//3. setDate
```

```

ahora.setDate(15);
console.log(ahora.getDate()); // 15
//4. setHours
ahora.setHours(10);
console.log(ahora.getHours()); // 10
//5. setMinutes
ahora.setMinutes(45);
console.log(ahora.getMinutes()); // 45

```

Cálculos con fechas.

Podemos realizar operaciones aritméticas con fechas, ya que por detrás trabajamos con los milisegundos transcurridos desde la fecha de referencia. Una vez obtenidos los milisegundos de diferencia es labor nuestra convertirlos a la unidad deseada.

Por ejemplo.

```

let fecha1 = new Date('2023-09-01');
let fecha2 = new Date('2023-09-30');
let diferencia = fecha2 - fecha1; // Diferencia en milisegundos
let diasDiferencia = diferencia / (1000 * 60 * 60 * 24); // Convertir a
días
console.log(diasDiferencia); // 29

```

Métodos estáticos de Date.

El objeto Date cuenta con varios métodos estáticos para trabajar principalmente con los milisegundos desde la fecha de referencia.

- **Date.now():** Devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970 (Epoch Unix).
- **Date.parse(cadenaFecha):** Analiza una cadena de texto que representa una fecha y devuelve el número de milisegundos desde el 1 de enero de 1970 hasta la fecha especificada.
- **Date.UTC(año, mes, día, horas, minutos, segundos, milisegundos):** Devuelve el número de milisegundos desde el 1 de enero de 1970 para una fecha y hora especificada en el tiempo universal coordinado (UTC). Los meses empiezan desde 0 (enero).

JSON

JSON (*JavaScript Object Notation*) es un formato de texto ligero utilizado para el intercambio de datos. Es ampliamente utilizado en aplicaciones web para enviar y recibir datos entre un servidor y un cliente. Aunque JSON es independiente del lenguaje de programación, se basa en la sintaxis de objetos y arrays de JavaScript, lo que lo hace fácilmente integrable en este lenguaje.

En JavaScript, el objeto global JSON proporciona métodos para convertir datos entre objetos de JavaScript y cadenas de texto en formato JSON. Ten en cuenta que **JSON es el formato más**

comúnmente utilizado para transmitir datos entre el frontend y el backend de las aplicaciones web.

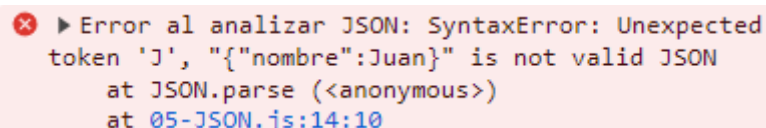
- **JSON.stringify(objeto):** Para convertir objetos de JavaScript en cadenas JSON.
- **JSON.parse(cadena):** Para convertir cadenas JSON en objetos de JavaScript.
 - **JSON.parse(cadena, fn(propiedad, valor)):** permite indicar una función encargada de la transformar la cadena de entrada de cada propiedad, útil por ejemplo con Date, Map, Set...

Por ejemplo.

```
//JSON
// 1. JSON.stringify(objeto) : Convierte un objeto en una cadena JSON.
let objeto = { nombre: "Juan", edad: 30, activo: true };
let cadenaJSON = JSON.stringify(objeto);
console.log(cadenaJSON); // '{"nombre":"Juan","edad":30,"activo":true}'

//2. JSON.parse(cadenaJSON) : Convierte una cadena JSON en un objeto.
cadenaJSON = '{"nombre":"Juan","edad":30,"activo":true}';
objeto = JSON.parse(cadenaJSON);
console.log(objeto.nombre); // "Juan"

//Si la cadena JSON no es válida, se lanza un error.
try {
  JSON.parse('{"nombre":Juan}'); // Error: falta comillas en "Juan"
} catch (error) {
  console.error('Error al analizar JSON: ', error);
}
```



```
✖ ▶ Error al analizar JSON: SyntaxError: Unexpected
  token 'J', '{"nombre":Juan}' is not valid JSON
    at JSON.parse (<anonymous>)
    at 05-JSON.js:14:10
```

Colecciones

JavaScript también cuenta con colecciones genéricas.

- **Array:** Lo hemos desarrollado al principio debida su importancia.
- **Map:** Colección de pares clave-valor. Se suele usar JSON por aportar ventajas en la comunicación y tratamiento.
- **Set:** Colección que permite definir un conjunto.

Map

Un **Map** es una colección de pares clave-valor, donde cualquier tipo de dato (objetos, funciones, números, etc.) puede ser utilizado como clave.

- **new Map():** Crea un nuevo Map.
- **.set(clave, valor):** Añade o actualiza un valor en el mapa asociado a la clave especificada.

- **.get(clave):** Devuelve el valor asociado con la clave dada.
- **.keys():** Devuelve un iterador con todas las claves del Map.
- **.values():** Devuelve un iterador con todos los valores del Map.
- **.entries():** Devuelve un iterador con los pares [clave, valor] del Map. Es el método que utiliza for...of por defecto.
- **.has(clave):** Verifica si existe una clave en el mapa.
- **.delete(clave):** Elimina un elemento del mapa por su clave.
- **.clear():** Elimina todos los elementos del mapa.
- **.size:** Devuelve el número de elementos en el mapa.

NOTA: “Map” no se puede iterar con for..in. No tiene propiedades como los JSON.

Por ejemplo.

```
let mapa = new Map();
mapa.set('nombre', 'Juan');
mapa.set('edad', 30);

for (let clave of mapa.keys()) {
  console.log(clave); // "nombre", "edad"
}
```

Set

Un **Set** es una colección de valores únicos. Los elementos dentro de un Set no se repiten, y el orden de los elementos es el mismo en el que fueron insertados.

- **.add(valor):** Añade un nuevo valor al conjunto. Si el valor ya existe, no se añade.
- **.values():** Devuelve un iterador que contiene todos los valores del conjunto.
- **.has(valor):** Verifica si un valor está presente en el conjunto.
- **.delete(valor):** Elimina un valor del conjunto.
- **.clear():** Elimina todos los valores del conjunto.
- **.size:** Devuelve el número de elementos en el conjunto.

Por ejemplo.

```
let conjunto = new Set();
conjunto.add(1);
conjunto.add(2);
conjunto.add(3);

console.log(conjunto.has(2)); // true
conjunto.delete(2);
console.log(conjunto.size); // 2
```

Otros predefinidos

Hay otros elementos predefinidos importantes como “**Error**” y “**RegExp**” que desarrollaremos en detalle cuando trabajemos con programación orientada a objetos y las validaciones.

Otros objetos como “**Number**” no los vamos a desarrollar aunque se invita a revisar su documentación.

Bibliografía

Referencia MDN a objetos globales JavaScript

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

Documentación MDN JSON

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON