Conceptos básicos de String y Array

Primero recordemos: ¿Qué es una String?

Una String es una cadena de texto o un conjunto de caracteres. Y existen distintos métodos que nos proporciona JavaScript de forma nativa para facilitarnos la manipulación de éstas.

A continuación, un listado de los métodos más utilizados comúnmente.

Método	Descripción
charAt()	Devuelve una String según el índice que le indiquemos.
concat()	Une varias Strings y devuelve una sola cadena de texto.
endsWith()	Devuelve un boolean si la String termina con el valor que le indiquemos.
trim()	Elimina los espacios sobrantes en una String.
includes()	Devuelve un boolean si la String contiene ese valor.
indexOf()	Devuelve el primer índice donde encuentra el valor dentro de la String.
toString()	Convierte un valor (Por ejemplo, un número) a String.
lastIndexOf()	Devuelve el último índice donde encuentra el valor dentro de la String.
repeat()	Repite la String la cantidad de veces que le indiquemos y la devuelve concatenada.
replace()	Reemplaza un valor dentro de la String por el segundo valor que le indiquemos.
search()	Busca un valor dentro de la String y devuelve el índice donde se encuentra.
slice()	Nos permite recortar nuestra String usando 2 parámetros, el inicio y el final.
split()	Divide la String en un Array, separando según el parámetro que le indiquemos.
startsWith()	Devuelve un boolean si la String empieza con el valor que le indiquemos.
toLowerCase()	Convierte toda la String a minúscula.
toUpperCase()	Convierte toda la String a mayúscula.

¿Cómo podemos utilizar un método un método sobre una String?

Primero creamos una variable que contenga nuestra cadena de texto.

```
var nombre = "Academia Numen"
```

Ahora para utilizar un método tenemos que utilizar esa variable seguido del nombre del método y sus parámetros.

Ejemplos prácticos:

```
nombre.toUpperCase()
// ACADEMIA NUMEN
nombre.toLowerCase()
// academia numen
nombre.repeat(2)
// Academia NumenAcademia Numen
nombre.replace("Academia", "Full Stack")
// Academia Full Stack
```

Métodos de Arrays

Los arreglos (o arrays) son utilizados para guardar múltiples valores en una sola variable, se identifican por índices.

Ejemplo:

```
var array = ["Academia", "Numen", "Full Stack"]
```

La primera posición de nuestro array comienza desde el número 0

Si quisieramos imprimirlo en la consola deberíamos escribir:

```
array[0] // Academia

array[1] // Numen

array[2] // Full Stack
```

Primero declaramos el arreglo que vamos a utilizar para los próximos ejemplos:

```
var users = [

{ nombre: "Bart", apellido: "Simpson", edad: 10 },

{ nombre: "Lisa", apellido: "Simpson", edad: 8 },

{ nombre: "Maggie", apellido: "Simpson", edad: 1 },

]
```

Find

Busca y devuelve aquella posición que cumpla con el parámetro:

```
var resultado = users.find(user => user.edad === 8)
```

En este caso queremos encontrar la posición que contenga la edad de 8:

```
// { nombre: "Lisa", apellido: "Simpson", edad: 8 }
```

Includes

Devuelve true o false si en el array existe el valor indicado

```
resultado = users.some(user => user.apellido === "Flanders")

// false
```

Map

Recorre todo nuestro arreglo y devuelve otro arreglo con lo indicado en el método.

```
resultado = users.map(user => user.nombre.toUpperCase())
```

Resultado:

```
resultado = [

{ nombre: "BART", apellido: "Simpson", edad: 10 },
```

```
{ nombre: “LISA”, apellido: “Simpson”, edad: 8 },
{ nombre: “MAGGIE”, apellido: “Simpson”, edad: 1 },
]
```

Filter

Recorre todo nuestro arreglo y devuelve otro arreglo que cumpla la condición.

```
resultado = users.filter(user => user.edad !== 1)
```

Resultado:

```
resultado = [
{ nombre: “BART”, apellido: “Simpson”, edad: 10 },
{ nombre: “LISA”, apellido: “Simpson”, edad: 8 },
]
```

Función typeof

Typeof es un operador JavaScript que, al ser llamado sobre una variable, devuelve el tipo de dato que dicha variable contiene. Entre otras cosas, podemos utilizarlo para validar de parámetros de una función o para comprobar si una variable ha sido definida.

El operador typeof es muy útil porque nos permite consultar fácilmente el tipo de datos que una variable contiene. Esto es importante en JavaScript porque se trata de un lenguaje escrito dinámicamente. Esto significa que no necesitamos asignar un tipo de datos fijo a una variable cuando la creamos. Al no restringir las variables de esta manera, el tipo de datos que contienen puede cambiar durante la ejecución de un programa.

Por ejemplo:

```
var x = 12345; // number
x = ‘string’; // string
```

```
x = { key: 'value' }; // object
```

Como podemos ver en el ejemplo anterior, una variable puede cambiar de tipo durante la ejecución de un programa. En nuestro trabajo diario puede ser complejo llevar un registro de estos cambios y es en esos casos donde el operador typeof será útil.

El operador typeof devuelve una cadena de texto que representa el tipo de dato actualmente contenido en una variable. Para usarlo podemos

escribir typeof(variable) o typeof variable. Volviendo al ejemplo anterior, podemos utilizarlo para comprobar el tipo de dato de la variable x en cada paso:

```
var x = 12345;
console.log(typeof x) // number
x = 'string';
console.log(typeof x) // string
x = { key: 'value' };
console.log(typeof x) // object
```

De esta manera, por ejemplo, podemos comprobar el tipo de dato de una variable dentro de una función y continuar en caso de obtener el resultado esperado.

Aquí tenemos una función de ejemplo que puede tomar una variable y anunciar si es una cadena de texto o un número:

```
function verificar(x) {
 if(typeof(x) === 'string') {
        alert('x es una string')
 } else if(typeof(x) === 'number') {
        alert('x es un number)
 }
```

```
}
```

También podemos utilizar el operador typeof para asegurarnos de que una variable ha sido definida antes de utilizarla en nuestro código. Esto puede ser útil para prevenir un error muy común, intentar acceder a una variable cuyo valor aún no ha sido definido.

```
function(x){
if (typeof(x) === 'undefined') {
        console.log('la variable x no está definida');
        return;
        }
        // aquí continuamos con la función…
}
```

Cuando trabajamos con números, el valor que devuelve el operador typeof puede no ser lo que estamos esperando.

Los números pueden convertirse en NAN (No es un número) por distintos motivos.
console.log(typeof NaN); // "number"

Quizás estamos intentando multiplicar un número con un objeto, porque olvidamos acceder al número que ese objeto contiene:

```
var x = 1;
var y = { number: 2 };
console.log(x * y); // NaN
console.log(typeof (x * y)); // number
```

En este caso, no será suficiente utilizar typeof para comprobar que el resultado es un número, ya que NaN también cumple con esta condición.

En la siguiente función, vamos a comprobar que el valor es un número, pero excluyendo a NaN de la condición:

```
function esNumero(data) {
        return (typeof data === 'number' && !isNan(data));
}
```

Si bien este es un método muy útil para validar tipos de datos, también debemos tener cuidado porque JavaScript tiene algunas rarezas y entre ellas encontramos el resultado de typeof en algunas situaciones particulares. En JavaScript muchas cosas son inesperadamente objetos, como por ejemplo los arreglos (arrays) y los valores null.

```
var x = [1,2,3,4];
console.log(typeof x) // object
console.log(typeof null) // object
```