Objetos nativos. String

- C.F.G.S. DAW
- 6 horas semanales
- Mes aprox. de impartición: Nov

Carmelo José Jaén Díaz

- iPasen - cjaedia071@g.educaand.es

Índice

OBJETIVO

- Aprender cómo se puede manejar el tiempo en JavaScript.
- Saber cómo se programa la ejecución de funciones de forma puntual y periódica en el tiempo.
- Registrar en el front-end información de navegación, usuario, etc., mediante cookies o mediante el almacenamiento local que ofrece el nuevo estándar HTML5.
- Profundizar en el concepto de objeto.
- Conocer y manejar funciones relativas al lenguaje sobre arrays, strings, números.

GLOSARIO

Backtracking. Estrategia utilizada en algoritmos que resuelven problemas que tienen ciertas restricciones. Este término fue creado por primera vez por el matemático D. H. Lehmer en la década de los cincuenta.

BOM (Browser Object Model). Convención específica implementada por los navegadores para que JavaScript pudiese hacer uso de sus métodos y propiedades de forma uniforme.

Expresión regular. Secuencia de caracteres que forman un patrón determinado, generalmente un patrón de búsqueda.

NaN. Propiedad que indica Not a Number (valor no numérico).

Objeto window. Aquel que soportan todos los navegadores y que representa la ventana del navegador. Se estudiará en profundidad en capítulos posteriores.

URI (Uniform Resource Identifier). Cadena de caracteres que identifica un recurso en una red de forma unívoca. Una URI puede ser una URL, una URN o ambas.

GLOSARIO

URN. Localizador de recursos en la web que funciona de forma parecida a una URL, pero su principal diferencia es que no indica exactamente dónde se encuentra dicho objeto.

INTRODUCCIÓN

Como seguramente sabréis, un objeto de tipo *String* representa una serie de caracteres dentro de una cadena. O lo que es lo mismo, en una variable de tipo cadena vamos a poder almacenar palabras, frases... cualquier cosa formada por caracteres alfanuméricos y simbólicos.

En esta lección vamos a ver tres operaciones (llamémoslo así) esenciales en una cadena: la instanciación, la concatenación y la propiedad *length*.

INTRODUCCIÓN

Instanciación: o creación de un “ejemplar” de un objeto. Como ya vimos en la lección anterior, podemos hacerlo de dos maneras: o utilizando la palabra *new* o creando datos primitivos.

Concatenación: una operación propia de las cadenas que permite unir dos cadenas de texto en una.

Propiedades: en el caso de *String* podemos hablar de *length*, que nos permite conocer el número de caracteres de la cadena.

¡¡¡ AHORA SÍ QUE SE PERMITE SU USO EN EL MÓDULO !!! Aunque ya hemos visto lo fácil que es su implementación manual gracias a la AE2.1 METODO PUSH MANUAL

INSTANCIACIÓN DE *String*

Para definir una cadena se puede hacer uso de la palabra *new* o definiendo una variable con comillas simples o dobles.

DIFERENTES FORMAS DE INSTANCIACIÓN

```
//Con comillas dobles
	let daw = "Desarrollo de aplicaciones web";
//Con comillas simples Template literals (Template strings)
	let dam = 'Desarrollo de aplicaciones multiplataforma';
//Las comillas dobles permiten el uso de comillas simples en su interior sin romper la declaración
	let asir = "Administración de 'Sistemas Informáticos' en Red";
//Pero no permiten el uso de otras comillas dobles, hay que usar caracteres de escape \"
	let smr = "Sistemas \"Microinformáticos\" y Redes";
//Haciendo uso del constructor de objetos
	let ciclos = new String(""); //Está vacío, dentro de las comillas iría la cadena
```

CONCATENACIÓN DE *String*

Nos permite “sumar” una o varias cadenas, cadenas con otras variables o cadenas con caracteres de escape.

DIFERENTES FORMAS DE CONCATENACIÓN

```
ciclos += "Hay 3 ciclos de Grado Superior: \n";
ciclos += daw + ", " + dam + " y " + asir + "\n";
ciclos += " y 1 ciclo de Grado Medio: \n";
ciclos += smr;

alert (ciclos);
```

PROPIEDADES DE *String*

Podemos encontrar todos los métodos de *String* en W3Schools.com>JavaScript>JS References.

En el caso de *length*, al ser una propiedad no hace falta poner los paréntesis al final.

length nos devuelve la longitud de una cadena.

EJEMPLO DE USO

```
//Su nomenclatura es variable.propiedadOmetodo
alert ("La longitud de la cadena ciclos es: "+ciclos.length);
```

MÉTODOS DE *String*

A continuación, veremos diferentes métodos de búsqueda en cadenas, como son:

BÚSQUEDA

- *charAt*
- *charCodeAt*
- *fromCharCode*
- *indexOf*
- *lastIndexOf*
- *search*
- *match*

COMPARACIÓN

- *localeCompare*

OTROS

- Métodos de *incluye*, *empieza* o *termina*.
- Métodos para *concatenar* o *repetir*.
- Métodos de *extracción* de subcadenas.
- Métodos para cambiar a mayúsculas o minúsculas.
- Métodos especiales

MÉTODOS DE *String*.
Búsqueda: *chartAt*

Método: charAt(<num>)

Finalidad: devuelve el carácter de una posición especificada como argumento.

Ejemplo de uso:

```
alert ("El carácter 16 de la cadena ciclos es: "+ciclos.charAt(16));
```

MÉTODOS DE *String*.
Búsqueda: *indexOf* y *lastIndexOf*

Método: indexOf(<caracter>) y lastIndexOf(<caracter>)

Finalidad: devuelve la primera (*indexOf*) o la última posición (*lastIndexOf*) de un carácter en una cadena.

Ejemplo de uso:

```
alert ("La primera aparición de la letra a en ciclos es: "+ciclos.indexOf("a"));
alert ("La última aparición de la letra a en ciclos es: "+ciclos.lastIndexOf("a"));
```

MÉTODOS DE *String*.
Búsqueda: *search*

Método: search(<cadena|expresiónRegular>)

Finalidad: buscar una cadena o expresión regular en otra cadena.

Ejemplo de uso:

```
alert ("Busca la cadena 'web' en la variable daw: "+daw.search("web"));
```

MÉTODOS DE *String*.
Búsqueda: *match*

Método: match(<expresión regular>)

Finalidad: busca una expresión regular en otra cadena.

Ejemplo de uso:

```
//Se estudiará en el capítulo de expresiones regulares en el tema de validación de formularios
```

MÉTODOS DE *String*. Comparación: *localeCompare*

Método: localeCompare(<cadena>)

Finalidad: devuelve -1 (antes), 0 (igual) o 1 (después) dependiendo de la posición de una cadena respecto a otra morfológicamente hablando.

Ejemplo de uso:

```
alert ("¿Es daw menor que dam? "+ daw.localeCompare(dam));
//Devuelve 1, ya que daw va después en el alfabeto que dam
//la w de web es posterior a la m de multiplataforma
```

MÉTODOS DE *String*.
Incluye, empieza o termina: *includes*

Método: includes(<cadena>)

Finalidad: devuelve true si la cadena incluye el parámetro.

Ejemplo de uso:

```
alert ("¿Incluye 'aplicaciones' daw? "+daw.includes("aplicaciones"));
```

MÉTODOS DE *String*.
Incluye, empieza o termina: *startsWith*

Método: startsWith(<cadena>)

Finalidad: devuelve true si la cadena empieza con el parámetro.

Ejemplo de uso:

```
alert ("Empieza daw con la palabra 'aplicaciones'? "+daw.startsWith("aplicaciones"));
```

MÉTODOS DE *String*.
Incluye, empieza o termina: *endsWith*

Método: endsWith(<cadena>)

Finalidad: devuelve true si la cadena finaliza con el parámetro.

Ejemplo de uso:

```
alert ("Acaba daw con la palabra 'aplicaciones'? "+daw.endsWith("aplicaciones"));
```

MÉTODOS DE *String*.
Concatenación y repetición: *concat*

Método: concat(<cadena>)

Finalidad: concatena a la cadena el parámetro

Ejemplo de uso:

```
alert ("Concatena daw con dam \n"+daw.concat(dam));
```

MÉTODOS DE *String*.
Concatenación y repetición: *repeat*

Método: repeat(<número>)

Finalidad: repetir la cadena el número de veces que aparece como parámetro

Ejemplo de uso:

```
alert ("Repetir daw \n"+daw.repeat(3));
```

MÉTODOS DE *String*.
Extracción: *slice*

Método: slice(<posicion inicial>,<posicion final>)

Finalidad: devuelve la cadena comprendida entre ambas posiciones

Ejemplo de uso:

```
alert ("La cadena que hay entre el 4 y el 6 en daw es: "+daw.slice(4,6));
```

MÉTODOS DE *String*. Extracción: *slice*

Método: slice(<posicion inicial>,<posicion final>)

Finalidad: devuelve la cadena comprendida entre ambas posiciones

Ejemplo de uso:

```
alert ("La cadena que hay del 4 en adelante en daw es: "+daw.slice(4));

//Si queremos que nos muestre la cadena entre un parámetro y el final
independientemente de cuál sea este, pasaremos solo ese parámetro como
argumento.
```

MÉTODOS DE *String*.
Extracción: *slice*

Método: slice(<posicion inicial>,<posicion final>)

Finalidad: devuelve la cadena comprendida entre ambas posiciones

Ejemplo de uso:

```
alert ("El primer caracter en daw es: "+daw.slice(0,1));

//Si queremos obtener el primer carácter, pasamos como argumentos (0,1)
```

MÉTODOS DE *String*.
Extracción: *slice*

Método: slice(<posicion inicial>,<posicion final>)

Finalidad: devuelve la cadena comprendida entre ambas posiciones

Ejemplo de uso:

```
alert ("El ultimo caracter en daw es: "+daw.slice(-1));

//Si queremos obtener el último carácter, pasamos como argumentos (-1)
```

MÉTODOS DE *String*. Extracción: *substring*

Método: substring(<posicion inicial>,<posicion final>).

Funciona de manera similar al método *slice*.

Finalidad: devuelve la cadena comprendida entre ambas posiciones

Ejemplo de uso:

```
alert ("La cadena que hay del 4 en adelante en daw es: "+daw.substring(4));

//Si queremos que nos muestre la cadena entre un parámetro y el final
independientemente de cuál sea este, pasaremos solo ese parámetro como
argumento.
```

MÉTODOS DE *String*. Extracción: *substring*

Método: substring(<posicion inicial>,<posicion final>).

Funciona de manera similar al método *slice*.

Finalidad: devuelve la cadena comprendida entre ambas posiciones

Ejemplo de uso:

```
alert ("El primer caracter en daw es: "+daw.substring(0,1));

//Si queremos obtener el primer carácter, pasamos como argumentos (0,1)
```

MÉTODOS DE *String*. Extracción: *substring*

Método: substring(<posicion inicial>,<posicion final>).

Funciona de manera similar al método *slice*.

Finalidad: devuelve la cadena comprendida entre ambas posiciones

Ejemplo de uso:

```
alert ("El ultimo caracter en daw es: "+daw.substring(daw.length-1, daw.length));

        //Si queremos obtener el último carácter, pasamos como argumentos la
        penúltima posición y la última posición
```

MÉTODOS DE *String*. Extracción: *substring*

Método: substring(<posicion inicial>,<posicion final>).

Funciona de manera similar al método *slice*.

Finalidad: devuelve la cadena comprendida entre ambas posiciones

Ejemplo de uso:

```
alert ("La cadena que hay entre el 4 y el 6 en daw es: "+daw.subsrting(4,6)); ==
alert ("La cadena que hay entre el 4 y el 6 en daw es: "+daw.subsrting(6,4));

        //Si nos equivocamos e invertimos los argumentos, ponemos (6,4) en vez de
        (4,6) substring interpreta que nos hemos equivocado y nos dará el mismo
        resultado
```

MÉTODOS DE *String*. Extracción: *substr*

Método: substr(<posicion inicial>,<número de caracteres>)

Funciona de manera similar al método *slice* y *substring*.

Finalidad: devuelve la cadena comprendida entre la posición inicial y el número de caracteres

Ejemplo de uso:

```
alert ("La cadena de 7 caracteres que hay a partir del caracter 5 es:
"+daw.substr(5,7));
```

IMPORTANTE: ESTOS TRES MÉTODOS DE EXTRACCIÓN PUEDEN TENER COMPORTAMIENTOS DIFERENTES SEGÚN EL NAVEGADOR WEB EN EL QUE SE EJECUTEN.

MÉTODOS DE *String*. Extracción: *split*

Método: split(<caracterDeSeparacion>,[<número de veces])

Finalidad: divide la cadena en un array de subcadenas separadas por el carácter del primer parámetro.

Ejemplo de uso:

```
alert("La cadena daw separada por espacios es: "+daw.split(" "));

//Devuelve Desarrollo,de,aplicaciones,web
```

MÉTODOS DE *String*. Extracción: *split*

Método: split(<caracterDeSeparacion>,[<número de veces])

Finalidad: divide la cadena en un array de subcadenas separadas por el carácter del primer parámetro.

Ejemplo de uso:

```
alert("La cadena daw separada por espacios es: "+daw.split(" ",2));

//Devuelve Desarrollo,de
```

MÉTODOS DE *String*. Extracción: *split*

Método: split(<caracterDeSeparacion>,[<número de veces])

Finalidad: divide la cadena en un array de subcadenas separadas por el carácter del primer parámetro.

Ejemplo de uso:

```
alert("La cadena daw separada por espacios es: "+daw.split("",2));

//Si no indicamos el carácter de separación devuelve D,e
//Si no indicamos el numero de veces, obtenemos el string separado
por caracteres en un array
```

MÉTODOS DE *String*. Extracción: *trim*

Método: trim(<cadena>)

Finalidad: extrae los caracteres de la cadena eliminando los espacios del principio y del final.

Ejemplo de uso:

```
alert ("La cadena sin espacios quedaría: \n"+"          Hola, caracol    a".trim());
```

MÉTODOS DE *String*.
Minúsculas: *toLowerCase*

Método: toLowerCase()

Finalidad: devuelve la cadena en minúsculas.

Ejemplo de uso:

```
alert (daw.toLowerCase());
```

MÉTODOS DE *String*.
Mayúsculas: *toUpperCase*

Método: toUpperCase()

Finalidad: devuelve la cadena en mayúsculas.

Ejemplo de uso:

```
alert (daw.toUpperCase());
```

Variante: toLocaleUpperCase()

Finalidad: devuelve la cadena en mayúsculas acorde a un idioma específico.

Ejemplo de uso:

```
const city = 'istanbul';
console.log(city.toLocaleUpperCase('en-US'));
// Expected output: "ISTANBUL"
console.log(city.toLocaleUpperCase('TR'));
// Expected output: "İSTANBUL"
```

MÉTODOS DE *String*.
toString

Método: toString()

Finalidad: devuelve el valor del objeto String.

Ejemplo de uso:

```
const stringObj = new String('foo');

console.log(stringObj);
// Expected output: String { "foo" }

console.log(stringObj.toString());
// Expected output: "foo"
```

```
const numero = 34.67;

console.log(numero.toString());
// Expected output: "34.67"
```

MÉTODOS DE *String*.
valueOf

Método: valueOf()

Finalidad: devuelve el valor primitivo de un objeto String como un tipo de dato cadena. Este dato es equivalente al obtenido por el método *toString()*.

Este método es usado normalmente de manera interna por JavaScript y no de manera explícita en el código.

Ejemplo de uso:

```
const stringObj = new String('foo');
console.log(stringObj);
// Expected output: String { "foo" }
console.log(stringObj.valueOf());
// Expected output: "foo"
```