4.1 STRINGS

4.1.1 USO DE STRINGS

Ya hemos visto en el tema anterior cómo se deben utilizar los strings. Además, sabemos que son un tipo de datos básico (no son objetos como en otros lenguajes). Repasemos los conceptos ya vistos sobre strings:

- Los valores de tipo string tienen que delimitarse con comillas, bien sean dobles ("), simples (') o invertidas (`).
- Es posible que los strings contengan comillas como carácter literal, siempre que no coincidan con el delimitador del string. Ejemplos:

```
let s1="Miguel dijo 'venid' en voz baja";
let s2='Miguel dijo \'venid\' en voz baja';
```

- En el ejemplo ambos strings tienen el mismo contenido
- Se pueden escapar caracteres para representar caracteres no visibles, que no están en el teclado o que sean problemáticos:

```
let s3="Primera línea\nSegunda línea";
let s4="Mi sonrisa: \u{1F600}"; //s4 vale: Mi sonrisa: ☺
```

- Dentro de valores de string delimitados por comillas invertidas se pueden colocar directamente expresiones del lenguaje siempre que se coloquen dentro de los símbolos ${}. Ejemplo

```
let x=8, y=9;
console.log(`La suma de x e y es: ${x+y}`);
//Escribe: La suma de x e y es: 17
```

4.1.2 COMPARACIÓN DE STRINGS

Los textos se pueden comparar igual que los números.

```
let texto1="ana";
let texto2="Ana";
if(texto1==texto2)
    console.log("son iguales");
else
    console.log("son distintos");
```

Por pantalla aparece el texto ***son distintos*** porque se distingue entre mayúsculas y minúsculas.

Al comparar qué texto es mayor, se usa el orden que tienen los caracteres en la tabla Unicode. Así ***Casa*** es menor que ***Caso***. Pero, hay que tener en cuenta que las minúsculas, en la tabla Unicode, son mayores que las mayúsculas y por eso JavaScript considera que ***casa*** es mayor que ***Caso***. También hay problemas con los caracteres especiales de cada lengua (los que no posee el

inglés) ya que *Ñu* se considera mayor que *Oso*, porque en la tabla Unicode la eñe tiene un código mayor que la letra *O*.

4.1.2.1 MÉTODO LOCALECOMPARE

Es un método que permite comparar sin discriminar entre mayúsculas y minúsculas, y considerando la forma de ordenar de cada lengua.

Este método recibe como parámetro el texto con el que deseamos comparar. Podemos indicar un segundo parámetro que es el código oficial de idioma. Sin ese segundo parámetro se utiliza la configuración local del equipo que ejecuta el código.

Este método devuelve un número negativo si el segundo texto es mayor que el primero, un número positivo si es el primer texto el que es mayor en orden alfabético, y cero si los dos textos son iguales.

```
let texto3="Oso";
let texto4="Ñu";
console.log(texto3.localeCompare(texto4));
```

Este código devuelve el número uno (1), porque considera a la eñe más pequeña en orden alfabético que la letra o. Es lo deseable al ordenar en español. Podemos indicar en el segundo parámetro el código del país cuyas reglas usamos al ordenar (***es*** para español).

```
console.log(texto3.localeCompare(texto4,"es"));
```

4.1.3 MÉTODOS DE LOS STRINGS

JavaScript aporta numerosos métodos y propiedades a los strings. Los métodos y propiedades son parte de la programación orientada a objetos, pero, aun sin entrar en detalle en este tipo de programación (a la que dedicaremos mucho tiempo más adelante), lo único que implica este hecho es que cualquier texto tiene incorporadas funciones que nos ayudan en el uso de estos elementos.

Acceder a los métodos y propiedades es simplemente poner el nombre de la variable o el valor de tipo string, colocar un punto y escribir el nombre del método o función. Los métodos pueden requerir más información para hacer su labor, esos datos extra se conocen como parámetros.

Vamos a detallar los más interesantes métodos y propiedades.

4.1.3.1 TAMAÑO DEL STRING

La propiedad **length** devuelve el tamaño de un texto:

```
let texto="Nabucodonosor";
console.log(texto.length);
```

El resultado es el número 13 (tamaño de la palabra ***Nabucodonosor***).

4.1.3.2 OBTENER UN CARÁCTER CONCRETO

El método **charAt** permite extraer un carácter concreto del texto. Basta con indicar la posición del carácter deseado, teniendo en cuenta que el primer carácter es el número cero, el siguiente el uno y así sucesivamente. Ejemplo:

```
let texto="Nabucodonosor";
console.log(texto.charAt(5));
```

Escribe la letra "**o**", que es el sexto carácter, correspondiente a la posición 5.

Hay otro método llamado **charCodeAt** que funciona igual, pero devuelve el código del carácter de esa posición:

```
let texto="Nabucodonosor";
console.log(texto.charCodeAt(5));
```

Escribe el valor ***99*** que es el código Unicode de la letra ***o***.

4.1.3.3 CONVERTIR A MAYÚSCULAS Y MINÚSCULAS

Lo realizan los métodos (sin argumentos) **toUpperCase**, para pasar un texto a mayúsculas, y **toLowerCase**, para minúsculas.

```
let texto="En el año 2019 saqué el carnet de Camión";
console.log(texto.toUpperCase());
```

El resultado es:

```
EN EL AÑO 2019 SAQUÉ EL CARNET DE CAMIÓN.
```

Para evitar problemas debido a la forma específica de pasar a mayúsculas de algunas lenguas, disponemos de las funciones **toLocaleUpperCase** (pasar a mayúsculas) y **toLocaleLowerCase** (pasar a minúsculas). Se puede indicar un segundo parámetro con el código específico de país (***es, fr, en***, etc.) pero, hoy en día, las funciones habituales (toUpperCase y toLowerCase) funcionan correctamente en casi cualquier lengua.

4.1.3.4 MÉTODOS DE BÚSQUEDA DE CARACTERES

indexOf

Sintaxis:

```
texto.indexOf(texto [,inicio])
```

Devuelve la posición del texto indicado en la variable, empezando a buscar desde el lado derecho. Si aparece varias veces, se devuelve la primera posición en la que aparece. El segundo parámetro (***inicio***) es opcional y nos permite empezar a buscar desde una posición concreta.

Ejemplo:

```
var var1="Dónde esta la x, busca, busca";
```

```
document.write(var1.indexOf("x"));
//Escribe 14, posición del carácter "x"
```

En el caso de que el texto no se encuentre, devuelve **-1**

lastIndexOf

Igual que el método anterior, pero, en este caso, se devuelve la última posición en la que aparece el texto (o menos uno, si no se encuentra).

```
texto.lastIndexOf(texto [,inicio])
```

endsWith

```
texto.endsWith(textoABuscar[, tamaño])
```

Este método devuelve verdadero si el texto finaliza con el texto que indiquemos en el parámetro **textoABuscar**.

```
var texto="Esto es una estructura estática";
console.log(texto.endsWith("estática"));
```

Escribe **true**, porque efectivamente el texto termina con la palabra ***estática***.

El parámetro ***tamaño*** solo mira si termina el string con el texto indicado considerando que el tamaño del texto es el indicado:

```
var texto="Esto es una estructura estática";
console.log(texto.endsWith("estructura",22));
```

El nuevo código vuelve a escribir **true**, porque los primeros 22 caracteres del texto terminan con el texto ***estructura***.

startsWith

```
texto.startsWith(textoABuscar[, posición])
```

Es muy parecido al anterior, solo que este método busca si el string comienza con el texto indicado. Si es así, devuelve true. El parámetro ***posición*** mira si el texto comienza por el indicado, pero empezando a mirar desde la posición indicada:

```
var texto="Esto es una estructura estática";
console.log(texto.startsWith("Esto")); //Escribe true
console.log(texto.startsWith("es",5)); //También escribe true
```

4.1.3.5 EXTRAER Y MODIFICAR SUBCADENAS

replace

```
texto.replace(textoBuscado,textoReemplazo)
```

Busca en el string el texto indicado en el primer parámetro (***textoBuscado***) y lo cambia por el segundo texto (***textoReemplazo***). Ejemplo:

```
var texto="Esto es una estructura estática";
console.log(texto.replace("st","xxtt"));
```

Escribe:

```
Exxtto es una estructura estática
```

trim

Es un método que no necesita argumentos. Se encarga de quitar los espacios en blanco a derecha y a izquierda del texto. Ejemplo:

```
let texto="      texto con muchos espacios    ";
console.log("-"+texto+"-");
console.log("-"+texto.trim()+"-");
```

El resultado es:

```
-      texto con muchos espacios    -
-texto con muchos espacios-
```

El primer texto que se muestra es el original. Hemos colocado guiones a izquierda y derecha del mismo para que se aprecie realmente su contenido. El segundo texto deja pegados los guiones porque los espacios se han eliminado.

Es un método muy útil para ser usado cuando se realizan lecturas de datos sobre los usuarios. Es habitual necesitar que se eliminen los espacios a izquierda y derecha que, inadvertidamente, escriben los usuarios.

slice

```
texto.slice(inicio [,fin])
```

Extrae del texto los caracteres desde la posición indicada por ***inicio***, hasta la posición ***fin*** (sin incluir esta posición). Si no se indica fin, se toma desde el inicio hasta el final.

Ejemplo:

```
let texto="Esto es una estructura estática";
console.log(texto.slice(3,10));
```

El resultado obtiene desde el carácter de la posición número 3 (en realidad el cuarto) hasta el índice anterior al 10 (coge hasta el décimo carácter). En definitiva, escribe:

```
o es un
```

El parámetro ***fin*** puede usar números negativos, en ese caso, los números indican la posición contando desde el final del texto. Ejemplo:

```
let texto="Esto es una estructura estática";
```

```
console.log(texto.slice(3,-5));
```

Escribe por consola:

```
o es una estructura est
```

substring

```
texto.substring(inicio [,fin])
```

Funciona exactamente igual que slice anterior, pero no admite usar números negativos.

substr

```
texto.substr(inicio [,tamaño])
```

Extrae del texto el número de caracteres indicados por el parámetro ***tamaño***, desde la posición indicada por el número asociado al carácter ***inicio***. Si no se indica tamaño alguno, se extraen los caracteres desde la posición indicada por ***inicio*** hasta el final.
Ejemplo:

```
let texto="Esto es una estructura estática";
console.log(texto.substr(3,10));
```

Escribe por consola:

```
o es una e
```

split

```
texto.split([textoDelim [,límite])
```

Es un potente método capaz de dividir el texto en un array de textos (los arrays se explican en el apartado siguiente: 4.2 *"Arrays"*, en la página 126).

Sin indicar parámetro alguno, se genera un array donde cada elemento del array será cada carácter del string. Si se indica el parámetro ***textoDelim***, este se usa como texto delimitador; es decir, se divide el texto en trozos separados por ese delimitador. Ejemplo:

```
let texto="Esto es una estructura estática";
console.log(texto.split(" "));
```

Escribe:

```
[ 'Esto', 'es', 'una', 'estructura', 'estática' ]
```

El segundo parámetro pone un límite tope de divisiones. Por ejemplo:

```
var texto="Esto es una estructura estática";
console.log(texto.split(" ",3));
```

Escribe:

```
[ 'Esto', 'es', 'una']
```

El texto delimitador puede ser una expresión regular en lugar de un texto, eso permite dividir el texto en base a criterios más complejos

4.1.3.6 CONVERTIR CÓDIGO A TEXTO

El método **fromCharCode** permite que indiquemos una serie de números de código de la tabla Unicode y nos devolverá un string formado por los caracteres correspondientes a estos códigos.

Este método es estático, lo que significa que se usa indicando la clase String, es decir: `String.fromCharCode`. Lo cual es lógico porque este método crea un string. Ejemplo:

```
console.log(String.fromCharCode(65,66,67));
```

El código anterior escribe el texto *ABC*, porque 65 es el código de la letra *A*, 66 el de la *B* y 67 el de la *C*.

4.2 ARRAYS

4.2.1 ¿QUÉ ES UN ARRAY?

Todos los lenguajes de programación disponen de un tipo de variable que es capaz de manejar conjuntos de datos. A este tipo de estructuras se las llama **arrays** de datos. También se las llama **listas**, **vectores** o **arreglos**, pero todos estos nombres tienen connotaciones que hacen que se puedan confundir con otras estructuras de datos. Por ello, es más popular el nombre sin traducir al castellano: array.

Los arrays aparecieron en la ciencia de la programación de aplicaciones para solucionar un problema habitual al programar. Para entender este problema, supongamos que deseamos almacenar 25 notas de alumnos para ser luego manipuladas dentro del código JavaScript. Sin arrays necesitamos 25 variables distintas. Manejar esos datos significaría estar continuamente indicando, de forma individual, el nombre de esas 25 variables. Los arrays permiten manejar las 25 notas bajo un mismo nombre.

En definitiva, los arrays son variables que permiten almacenar, usando una misma estructura, una serie de valores. Para acceder a un dato individual dentro del array, hay que indicar su posición. La posición es un número entero conocido como **índice**. Así, por ejemplo ***nota[4]*** es el nombre que recibe el quinto elemento de la sucesión de notas. La razón por la que ***nota[4]*** se refiere al quinto elemento y no al cuarto es porque el primer elemento tiene índice cero.

Esto, con algunos matices, funciona igual en casi cualquier lenguaje. Sin embargo, en JavaScript los arrays son **objetos**. Es decir, no hay un tipo de datos array, si utilizamos **typeof** para averiguar el tipo de datos de un array, el resultado será la palabra **object**.

En algunos lenguajes como C, el tamaño del array se debe anticipar en la creación del array y no se puede cambiar más adelante. Este tipo de arrays son estáticos. Los arrays de JavaScript son totalmente dinámicos, su tamaño se puede modificar después de la declaración a voluntad.