TEMA 3

METODOS STRINGS

Y MATH

MANEJAR TEXTO

A continuación, centraremos nuestra atención en las cadenas de caracteres, estos son los fragmentos de texto en programación. Veamos las cosas que debemos saber sobre cadenas de caracteres al aprender JavaScript, como crear cadenas, comillas en cadenas y unir cadenas.

A primera vista, las cadenas se tratan de forma similar a los números, pero al profundizar vemos diferencias notables. Como ya hemos dicho para indicar que una variable va a contener texto, bastará con incluir los caracteres entre comillas simples o comillas dobles.

En JavaScript, puedes escoger entre comillas simples y dobles para envolver tus cadenas, de las dos formas funcionará correctamente. No hay diferencia entre las dos, y la que usemos depende de nosotros, sin embargo se aconseja elegir una y mantenerla, ya que usar diferentes tipos de comillas en el código podría llegar a ser confuso, especialmente si usamos diferentes comillas en la misma cadena.

Si queremos que en el texto se añada el carácter comillas y evitar problemas con que JavaScript lo entienda correctamente lo que podemos utilizar es el carácter \ (contrabarra) que hará que el carácter que venga después no sea tenido en cuenta por el intérprete. Existen otras opciones que podemos utilizar con el carácter contrabarra y que tienen un significado especial, entre ellos: **\n** genera una nueva línea y **\r** que devuelve el cursor al inicio de línea.

1. CONCATENAR CADENAS

Concatenar significa “unir”, y para unir cadenas en JavaScript el símbolo de más (+), el mismo operador que usamos para sumar números, pero en este contexto hace algo diferente. Veamos un ejemplo:

let primero = ‘Hola, ‘

let segundo = ‘¿cómo estás?’

let saludo = primero + segundo

Ahora en la variable saludo tenemos guardado lo siguiente “Hola, ¿cómo estás?”, aunque sin las comillas claro.

Si en lugar de texto intentamos concatenar una cadena de caracteres con un número, el número se convierte en una cadena de caracteres y se agrega al valor actual. El siguiente código produce la cadena de caracteres “El número es 5”.

```
9   <body>
10      <script>
11          let texto="El número es "
12          let numero=5
13          alert(texto+numero)
14      </script>
15  </body>
```

Este procedimiento es importante cuando tenemos una cadena de caracteres que contiene un número y queremos agregarle otro número. Como JavaScript considera el valor actual como una cadena de caracteres, el número también se convierte en una cadena de caracteres y los valores no se suman, se concatenan. En el ejemplo siguiente el resultado que obtendremos será 203.

```
11      let valor = "20"+3
12      alert(valor)
```

En el siguiente caso el resultado sí que será la suma de los dos números y el resultado será 23.

```
10  <script>
11      let valor = 20+3
12      alert(valor)
13  </script>
```

Qué podemos hacer para que se produzca una suma en el primer caso, cuando tenemos un string pero que realmente representa un valor numérico. Para hacer conversiones entre números y string o cadenas de caracteres usaremos los siguientes métodos o funciones:

- **Number**: con esta función convertimos cualquier cosa que se le pase en un número, si puede claro. Si intentamos convertir en un número la cadena “Maite” obtendremos un error. Veamos cómo solucionar el ejemplo anterior para que en lugar de concatenar el resultado sea la suma. En este caso el resultado que encontraremos en valor será 23.

```
11      let valor = Number("20")+3
12      alert(valor)
```

- **toString**: este método convierte un número en el equivalente en una cadena de caracteres.

```
11      let valor = 58
12      valor = valor.toString()
13      alert(typeof valor)
```

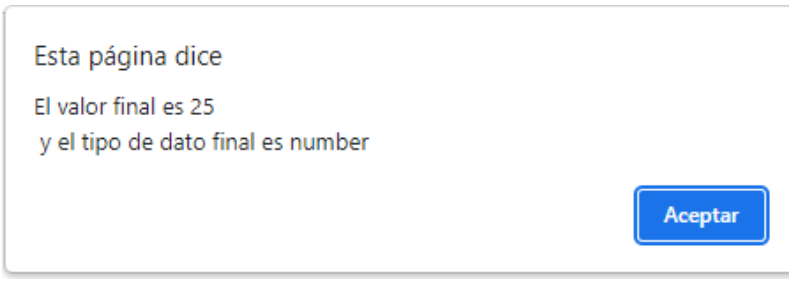
  En este caso el resultado que obtendremos será String, ya que aunque inicialmente el contenido de la variable valor es numérico, posteriormente se convierte en una cadena de caracteres, es decir, el resultado será “58”.

Estas instrucciones pueden ser muy útiles si un usuario introduce un número en un campo de texto de un formulario, al recoger este valor el tipo de dato será string. Si queremos operar con ese valor en alguna instrucción posterior, deberemos transformar ese dato en un número.

En las últimas versiones si utilizamos un operador numérico entre dos strings, el intérprete intenta reconvertir en número las cadenas de caracteres y se realiza la operación dando un valor final numérico.

```
let valor = "58"
let valor2 = "33"
let resultado = valor - valor2
alert("El valor final es "+resultado+"\n y el tipo de dato final es "+ typeof resultado)
```

En este ejemplo, el operador resta utilizado, solo es posible entre números, así que las cadenas de caracteres se convierten en números y se realiza la operación. Como además, el valor de una resta da como resultado un número el tipo de dato generado en la resta será un número.



2. LONGITUD DE UNA CADENA

Para conseguir la longitud que tiene una cadena basta con usar la propiedad length. Ten en cuenta que es una propiedad, luego no necesitamos utilizar paréntesis al utilizarlo. Veamos un ejemplo:

```
10      <script>
11          let saludo = "Hola, buenos días"
12          alert("La longitud del saludo es " + saludo.length)
13      </script>
```

El resultado será una ventana de alerta que nos indicará que la longitud del saludo es 17.

3. EXTRAER UN CARÁCTER DE UNA CADENA

Para devolver cualquier carácter de una cadena usaremos la notación de corchetes, es decir, incluiremos estos corchetes al final del nombre de nuestra variable. Dentro de los corchetes incluiremos el número del carácter que queremos obtener. Debemos tener en cuenta que los ordenadores cuentan desde 0, por eso para extraer el último carácter de cualquier cadena podemos utilizar la siguiente línea, combinando esta técnica con la propiedad length que ya vimos:

saludo [saludo.length-1];

En este caso lo que obtendremos será el carácter ‘s’. Podemos utilizar esto para encontrar la primera letra de una serie de cadenas y ordenarlas alfabéticamente.

4. ENCONTRAR UNA SUBCADENA DENTRO DE UNA CADENA

Algunas veces queremos encontrar si hay una cadena más pequeña dentro de una más grande, para saberlo podemos utilizar el método **indexOf()**, que toma un único parámetro, la subcadena que queremos buscar. El siguiente ejemplo nos devolverá la posición 13 que el primer carácter donde se encuentra la subcadena día que es la que buscamos.

```
11          let saludo = "Hola, buenos días"
12          alert(saludo.indexOf("día"))
```

Cuando la cadena no se encuentra el resultado devuelto es -1.

5. EXTRAER UNA SUBCADENA DENTRO DE UNA CADENA

Cuando lo que queremos es extraer una subcadena de una cadena utilizaremos el método **slice().** La sintaxis asociada a slice es la siguiente, le pasaremos dos parámetros, el primero será el primer carácter desde donde queremos empezar a extraer y el segundo es la posición del carácter posterior al último a ser extraído. Es decir, el corte ocurre desde la primera posición en adelante, pero excluyendo la última posición.

Si lo que queremos es extraer todos los caracteres restantes de una cadena después de cierto carácter, no hace falta incluir el segundo parámetro, solo necesitamos incluir la posición del carácter desde donde queremos extraer los caracteres restantes en la cadena.

```
<script>
    let saludo = "Hola, buenos días"
    alert(saludo.slice(0,4))
</script>
```

En este primer ejemplo, el resultado obtenido será la cadena “Hola”. En el siguiente ejemplo se extraerá la cadena “buenos días”.

```
<script>
    let saludo = "Hola, buenos días"
    alert(saludo.slice(6))
</script>
```

Si el primer argumento es negativo, como -4 nos devolverá los cuatro últimos caracteres de la cadena. Veamos un ejemplo:

```
let saludo = "Hola, buenos días"
alert(saludo.slice(-4))
```

Si pasamos dos argumentos y el segundo es negativo, lo que devolverá será todos los caracteres desde donde le indicamos excepto los últimos que indicamos en el segundo argumento.

```
let saludo = "Hola, buenos días"
alert(saludo.slice(0,-4))
```

En este caso la cadena estará formada por todos los caracteres excepto los cuatro últimos, en este caso el resultado será “Hola, buenos “.

Otro método que podemos utilizar para extraer una subcadena de una cadena es **substring()**. El método substring lo que hace es recuperar de una cadena de caracteres una subcadena. Este método se puede utilizar de dos formas, con dos parámetros en cuyo caso, el primer argumento indica la posición desde donde queremos empezar a subextraer, y el segundo argumento que indica la posición donde queremos terminar. En el caso de que a substring solo le pasemos un argumento, le indicamos desde que posición queremos extraer y se terminaría al final de la cadena, como vemos es similar al slice, aunque menos completo, por eso se suele utilizar slice.

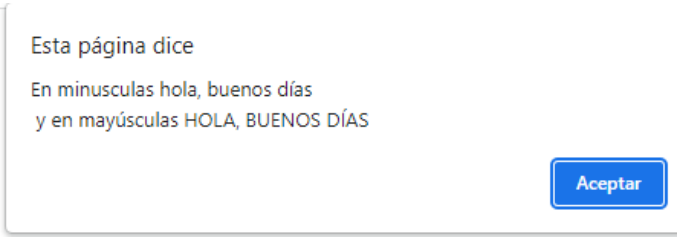
6. CAMBIAR TODO A MAYÚSCULAS O MINÚSCULAS

Igual que sucede con Java, los métodos que permiten cambiar una cadena de caracteres a mayúsculas es **toUpperCase()** y cambiar una cadena de caracteres a minúsculas es **toLowerCase().** En el siguiente ejemplo vemos cómo utilizar ambos métodos.

```
<script>
    let saludo = "Hola, buenos días"
    alert("En minusculas "+saludo.toLowerCase()+"\n y en mayúsculas "+saludo.toUpperCase())

</script>
```

Y este sería el resultado.



7. ACTUALIZANDO PARTES DE UNA CADENA

En una cadena podemos reemplazar una subcadena por otra usando el método **replace().** Este método tiene dos parámetros, la cadena que queremos reemplazar y la cadena con la que deseamos reemplazarla. Veamos un ejemplo, si queremos que la variable tenga el nuevo valor lo que hacemos es asignarle a la variable el resultado aplicando el método.

```
let saludo = "Hola, buenos días"
saludo = saludo.replace("buenos días","buendas tardes")
alert(saludo)
```

El resultado nos dará una ventana de alerta donde nos aparecerá “Hola, buenas tardes”.

8. ELIMINAR ESPACIOS EN BLANCO

El método **trim** lo que hace es eliminar los espacios en blanco al principio y al final de una cadena. Veamos un ejemplo:

```
let saludo = "              Hola, buenos días        "
alert(saludo.trim())
```

Al escribir estos espacios en blanco al principio y al final, cuando se escriba la variable saludo en el mensaje de alert nos escribirá estos espacios en blanco, algo que no queda bien en la escritura. Añadiendo el método trim estos espacios en blanco no se tienen en cuenta. Esto se utiliza en formularios porque al incluir información el usuario en ellos, puede ser que añada espacios en blanco al principio o al final de la cadena. Al comparar el valor con alguna cadena, como ese espacio en blanco cuenta, puede ser que no nos de igualdad, cuando en realidad los caracteres incluidos si son iguales.

9. METODOS STARTWITH Y ENDWITH

El método **startswith** lo que hace es comprobar si una cadena de caracteres empieza por un determinado carácter. Esta función devuelve true si la cadena comienza por ese carácter y false si no comienza por ese carácter. Veamos un ejemplo:

```
let saludo = "Hola, buenos días"
alert(saludo.startsWith("H"))
```

la sentencia anterior nos devolverá true, ya que la cadena saludo comienza por “H”. La sentencia siguiente nos devolverá false.

```
alert(saludo.startsWith("h"))
```

A este método se le puede añadir un argumento para indicar desde donde queremos empezar a contar. Por ejemplo, la siguiente sentencia nos devolverá true,

```
<script>
    let saludo = "Hola mundo"
    alert(saludo.startsWith("m",5))

</script>
```

ya que en la posición 5 nos encontramos la cadena 'mundo'.

El método **endswith** lo que hace es comprobar si una cadena de caracteres termina por un determinado carácter. Esta función devuelve true si la cadena comienza por ese carácter y false si no termina por ese carácter, la forma de uso es muy similar a la anterior.

```
let saludo = "Hola mundo"
alert(saludo.endsWith("o"))
```

La sentencia anterior nos devolverá true, ya que saludo termina por el carácter 'o'. El parámetro opcional que recibe es longitud, no inicio, así la sentencia siguiente nos devuelve true, ya que en este caso el carácter con el que termina es 'a', pero le decimos que solo busque en los cuatro primeros caracteres, veamos cómo escribirlo:

```
let saludo = "Hola mundo"
alert(saludo.endsWith("a",4))
```

Podríamos buscar palabras enteras, así la siguiente instrucción nos devolverá true:

```
<script>
    let saludo = "Hola mundo"
    alert(saludo.endsWith("mundo"))
</script>
```

10. METODO INCLUDES

El método includes lo que hace es comprobar si la cadena que le indicamos contiene el carácter que le pasamos como parámetro. También nos devolverá true o false, según la cadena contenga o no el carácter respectivamente. Veamos un ejemplo:

```
<script>
    let saludo = "Hola mundo"
    alert(saludo.includes("m"))
</script>
```

la sentencia anterior nos devolverá true. También le podemos pasar un argumento si queremos indicarle desde donde queremos empezar a contar.

```
let saludo = "Hola mundo"
alert(saludo.includes("a",5))
```

La sentencia anterior nos devolverá false, ya que aunque saludo contiene el carácter 'a', empezaríamos a contar en la posición 5, es decir en 'mundo', no existe ese carácter.

11. METODO REPEAT

El método repeat escribe la cadena que le indicamos, el número de veces que le indicamos. Veamos un ejemplo:

```
<script>
    let saludo = "Hola mundo"
    alert(saludo.repeat(2))
</script>
```

la sentencia anterior lo que hace es escribir en la consola dos veces la cadena 'Hola mundo'. Lo podemos hacer con una variable, o con la cadena directamente, de la siguiente forma:

console.log('Hola mundo'.repeat(2));

esta última sentencia y la anterior harían lo mismo.

12. MANEJO DE TEMPLATE STRINGS (PLANTILLAS LITERALES)

Las plantillas literales son cadenas de caracteres que habilitan el uso de expresiones incrustadas. Con ellas, es posible usar cadenas de caracteres de más de una línea, y funcionalidades de interpolación de cadenas de caracteres.

Las plantillas literales se delimitan con el carácter de comillas o tildes invertidas (` `), en lugar de las comillas sencillas o dobles. Las plantillas de cadena de caracteres pueden contener marcadores, identificados por el signo de dólar y envueltos entre llaves ${expresión}. Las expresiones contenidas en los marcadores, son enviados como argumentos a una función. La función por defecto concatena las partes para formar una única cadena de caracteres. Veamos un ejemplo de uso, en esta primera parte utilizaremos el símbolo "+" para concatenar cadenas de caracteres con contenido de variables.

```
<script>
    let nombre = "Maite"
    let apellido = "García"
    let edad = 20
    alert("Me llamo "+nombre+ " "+apellido+ " y tengo "+edad+ " años")
</script>
```

Como vemos se utilizan demasiados símbolos de suma, y esto puede generar errores, otra forma de escribirlo es utilizando el template strings de la siguiente forma:

```
<script>
    let nombre = "Maite"
    let apellido = "García"
    let edad = 20
    alert(`Me llamo ${nombre} ${apellido} y tengo ${edad} años`)
</script>
```

Como podemos ver, queda bastante más claro el contenido utilizando esta utilidad.

OBJETO MATH

Math es un objeto incorporado que tiene propiedades y métodos para constantes y funciones matemáticas. Math funciona con variables de tipo Number, es decir, variables de tipo numérico.

A diferencia de los demás objetos globales, el objeto Math no se puede editar, todas las propiedades y métodos de Math son estáticos, es decir, no se aplican sobre objetos de una clase, se ejecutan siempre asociados a la clase.

Dentro de Math nos encontramos dos variables Math.PI que guarda el valor de pi utilizado para el cálculo de áreas y volúmenes en círculos y esferas, entre otras cosas. También está la constante Math.E que es la que está asociado a logaritmos neperianos que se utiliza menos. Aunque existen muchas más propiedades que podemos utilizar, estos son los más importantes. A continuación, veremos alguno de los métodos que nos encontramos predefinidos dentro de Math y que podemos utilizar.

1. METODO abs

El método abs devuelve el valor absoluto del valor numérico que le pasamos como argumento, es decir, es el valor del número, pero sin el signo.

Math.abs(5); //devuelve un 5

Math.abs(-5); //devuelve el valor 5

2. METODO ceil

El método abs devuelve el valor absoluto del valor numérico que le pasamos como argumento, es decir, es el valor del número, pero sin el signo.

```
<script>

    alert(Math.ceil(4.5))
</script>
```

La sentencia anterior devuelve 5 que es el siguiente entero por encima del número pasado como argumento.

3. METODO floor

El método floor devuelve el entero más grande menor o igual que un número.

```
<script>
    alert(Math.floor(4.5))
</script>
```

La sentencia anterior devuelve 4 que es el anterior entero por debajo del número pasado como argumento.

4. METODO pow

El método pow sirve para aplicar potencias. Tenemos que pasar dos argumentos, el primero será la base de la potencia, y el segundo será el exponente.

```
<script>
    alert(Math.pow(4,5))
</script>
```

La sentencia anterior calcula el valor de 4 elevado a 5.

5. METODO random

El método random devuelve un valor aleatorio entre 0 y 1. En este caso no necesitamos pasar argumentos. Este método utiliza un algoritmo interno para realizar el cálculo de este valor.

```
<script>
    alert(Math.random())
</script>
```

6. METODO round

El método round devuelve el valor de un número redondeado al entero más cercano. La forma de uso será la siguiente:

```
<script>
    let redondeo1 = Math.round(4.6)
    let redondeo2 = Math.round(4.5)
    let redondeo3 = Math.round(4.4)
    alert(`El primer valor es ${redondeo1}
    el segundo es ${redondeo2} y el tercero ${redondeo3}`)
</script>
```

La sentencia anterior nos guardará en redondeo el valor 5 en los dos primeros casos y 4 en el tercero. Como podemos ver, a partir del decimal 5 redondea siempre hacia arriba.

7. METODO trunc

El método trunc devuelve la parte entera del número, se produce la eliminación de los dígitos fraccionarios.

```
<script>
    alert(Math.trunc(5.369))
</script>
```

Estas instrucciones da como resultado 5, simplemente elimina la parte decimal y deja solo la parte entera.

8. AJUSTAR EL NÚMERO DE DECIMALES DE UN NÚMERO

En este caso lo que vamos a utilizar es una función, que se aplica a un objeto de tipo numérico, no es un método estático, luego no pertenece a la clase Math, pero puede resultar interesante su uso.

Para ajustar el número de decimales podemos utilizar la función **toFixed(),** esta función lo que hace es redondear el número a los decimales que especificamos. Veamos algún ejemplo:

var numero=78.99;
numero.toFixed(0); //devuelve 79, realiza un redondeo y con cero lugares decimales
numero.toFixed(1); //devuelve 79.0, realiza un redondeo y con un decimal
numero.toFixed(2); //devuelve 78.99, en este caso no hay redondeo, porque teníamos 2 decimales
numero.toFixed(3); //devuelve 78.990, en este caso se añade simplemente un cero