INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

INGENIERÍA EN SISTEMAS COMPUTACIONALES

MATERIA: TECNOLOGÍAS PARA LA WEB

PROFESORA: RIVERA DE LA ROSA MONICA

PRESENTA:

RAMIREZ BENITEZ BRAYAN

GRUPO: 2CM14

OBJETOS DE JAVASCRIPT

CIUDAD DE MEXICO A 21 DE OCTUBRE DE 2021

String – Cadena de caracteres

El objeto String se utiliza para representar y manipular una secuencia de caracteres. Las cadenas son útiles para almacenar datos que se pueden representar en forma de texto. Algunas de las operaciones más utilizadas en cadenas son verificar su lenght, para construirlas y concatenarlas usando operadores de cadena + y +=, verificando la existencia o ubicación de subcadenas con indexOF() o extraer subcadenas con el método substring().

Las cadenas se pueden crear como primitivas, a partir de cadena literales o como objetos, usando el constructor String()

```
const string1 = "Una cadena primitiva";
const string2 = 'También una cadena primitiva';
const string3 = `Otra cadena primitiva más`;

 const string4 = new String("Un objeto String");
```

Hay dos formas de acceder a un carácter individual en una cadena. La primera es con el método charAt()

```
return 'cat'.charAt(1) // devuelve "a"
return 'cat'[1] // devuelve "a"
```

Cuando se usa la notación entre corchetes para acceder a los caracteres, no se puede intentar eliminar o asignar un valor a estas propiedades.

Para comparar cadenas se utilizan operadores

```
let a = 'a'
let b = 'b'
if (a < b) { // true
  console.log(a + ' es menor que ' + b)
} else if (a > b) {
  console.log(a + ' es mayor que ' + b)
} else {
  console.log(a + ' y ' + b + ' son iguales.')
}
```

Se puede lograr un resultado similar usando el método localeCompare() heredado por las instancias de String. Debemos tener en cuenta que a == b compara las cadenas en a y b por ser igual en la forma habitual que distingue entre mayúsculas y minúsculas. Para comparar sin tener en cuenta los caracteres en mayúsculas o minúsculas:

```
function isEqual(str1, str2)
{
    return str1.toUpperCase() === str2.toUpperCase()
} // isEqual
```

En esta función se utilizan mayúsculas en lugar de minúsculas, debido a problemas con ciertas conversiones de caracteres UTF-8.

JavaScript distingue entre objetos String y valores de primitivas string. (Lo mismo ocurre con Booleanos y Números).

Las cadenas literales (denotadas por comillas simples o dobles) y cadenas devueltas de llamadas a String en un contexto que no es de constructor (es decir, llamado sin usar la palabra clave new) son cadenas primitivas. JavaScript automáticamente convierte las primitivas en objetos String, por lo que es posible utilizar métodos del objeto String en cadenas primitivas. En contextos donde se va a invocar a un método en una cadena primitiva o se produce una búsqueda de propiedad, JavaScript ajustará automáticamente la cadena primitiva y llamará al método o realizará la búsqueda de la propiedad.

```
let s_prim = 'foo'
let s_obj = new String(s_prim)

console.log(typeof s_prim) // Registra "string"
console.log(typeof s_obj)  // Registra "object"
```

Las primitivas de String y los objetos String también dan diferente resultado cuando se usa eval(). Las primitivas pasadas a eval se tratan como código fuente; Los objetos String se tratan como todos los demás objetos, devuelven el objeto. Por ejemplo:

```
let s1 = '2 + 2'              // crea una string primitiva
let s2 = new String('2 + 2')  // crea un objeto String
console.log(eval(s1))         // devuelve el número 4
console.log(eval(s2))         // devuelve la cadena "2 + 2"
```

Por estas razones, el código se puede romper cuando encuentra objetos String y espera una string primitiva en su lugar, aunque generalmente los autores no necesitan preocuparse por la distinción.

Un objeto String siempre se puede convertir a su contraparte primitiva con el método valueOf().

```
console.log(eval(s2.valueOf()))  // devuelve el número 4
```

Los caracteres especiales se pueden codificar mediante notación de escape

Código	Salida
`\xxx` (donde *xxx* es de 1 a 3 dígitos octales; rango de `0-377`)	Punto de código Unicode/carácter ISO-8859-1 entre `U+0000` y `U+00FF`
`\'`	Comilla sencilla
`\"`	Comilla doble
`\\`	Barra inversa
`\n`	Nueva línea
`\r`	Retorno de carro
`\v`	Tabulación vertical

`\t`	Tabulación
`\b`	Retroceso
`\f`	Avance de página
`\u`*XXXX* (donde *XXXX* son 4 dígitos hexadecimales; rango de `0x0000-0xFFFF`)	Unidad de código UTF-16/punto de código Unicode entre `U+0000` y `U+FFFF`
`\u{`*X*`}` ... `\u{`*XXXXXX*`}` (donde *X*...*XXXXXX* es de 1 a 6 dígitos hexadecimales; rango de `0x0-0x10FFFF`)	Unidad de código UTF-32/punto de código Unicode entre `U+0000` y `U+10FFFF`
`\x`*XX* (donde *XX* son 2 dígitos hexadecimales; rango de `0x00-0xFF`)	Punto de código Unicode/carácter ISO-8859-1 entre `U+0000` y `U+00FF`

En lugar de tener líneas que se prolongan interminablemente, es posible que desees dividir específicamente la cadena en varias líneas en el código fuente sin afectar el contenido real de la cadena. hay dos maneras de conseguirlo.

Puedes usar el operador + para agregar varias cadenas juntas, así:

```
let longString = "Esta es una cadena muy larga que necesita " +
                 "que dividimos en varias líneas porque " +
                 "de lo contrario, mi código es ilegible."
```

El carácter de barra invertida (\) al final de cada línea para indicar que la cadena continúa en la siguiente línea. No debe haber ningún espacio ni ningún otro carácter después de la barra invertida (a excepción de un salto de línea) o como sangría; de lo contrario, no trabajará.

```
let longString = "Esta es una cadena muy larga que necesita \
que dividimos en varias líneas porque \
de lo contrario, mi código es ilegible."
```

Constructor

String()

Crea un nuevo objeto String. Realiza la conversión de tipos cuando se llama como función, en lugar de como constructor, lo cual suele ser más útil.

Métodos estáticos

String.fromCharCode(*num1* [, ...[, *numN*]])

Devuelve una cadena creada utilizando la secuencia de valores Unicode especificada.

String.fromCodePoint(*num1* [, ...[, *numN*]])

Devuelve una cadena creada utilizando la secuencia de puntos de código especificada.

String.raw()

Devuelve una cadena creada a partir de una plantilla literal sin formato.

Propiedades de la instancia

String.prototype.length

Refleja la length de la cadena. Solo lectura.

Métodos de instancia

String.prototype.charAt(*index*)

Devuelve el carácter (exactamente una unidad de código UTF-16) en el *índex* especificado.

String.prototype.charCodeAt(*index*)

Devuelve un número que es el valor de la unidad de código UTF-16 en el *index* dado.

String.prototype.codePointAt(*pos*)

Devuelve un número entero no negativo que es el valor del punto de código del punto de código codificado en UTF-16 que comienza en la *pos* especificada.

String.prototype.concat(*str*[, ...*strN*])

Combina el texto de dos (o más) cadenas y devuelve una nueva cadena.

String.prototype.includes(*searchString* [, *position*])

Determina si la cadena de la llamada contiene *searchString*.

String.prototype.endsWith(*searchString*[, *length*])

Determina si una cadena termina con los caracteres de la cadena *searchString*.

String.prototype.indexOf(*searchValue*[, *fromIndex*])

Devuelve el índice dentro del objeto String llamador de la primera aparición de *searchValue*, o -1 si no lo encontró.

String.prototype.lastIndexOf(*searchValue*[, *fromIndex*])

Devuelve el índice dentro del objeto String llamador de la última aparición de *searchValue*, o -1 si no lo encontró.

String.prototype.localeCompare(*compareString*[, *locales*[, *options*]])

Devuelve un número que indica si la cadena de referencia *compareString* viene antes, después o es equivalente a la cadena dada en el orden de clasificación.

String.prototype.match(*regexp*)

Se utiliza para hacer coincidir la expresión regular *regexp* con una cadena.

String.prototype.matchAll(*regexp*)

Devuelve un iterador de todas las coincidencias de *regexp*.

String.prototype.normalize([*form*])

Devuelve la forma de normalización Unicode del valor de la cadena llamada.

String.prototype.padEnd(*targetLength*[, *padString*]) (en-US)

Rellena la cadena actual desde el final con una cadena dada y devuelve una nueva cadena de longitud *targetLength*.

String.prototype.padStart(*targetLength*[, *padString*])

Rellena la cadena actual desde el principio con una determinada cadena y devuelve una nueva cadena de longitud *targetLength*.

String.prototype.repeat(*count*)

Devuelve una cadena que consta de los elementos del objeto repetidos *count* veces.

String.prototype.replace(*searchFor*, *replaceWith*)

Se usa para reemplazar ocurrencias de *searchFor* usando *replaceWith*. *searchFor* puede ser una cadena o expresión regular, y *replaceWith* puede ser una cadena o función.

String.prototype.replaceAll(*searchFor*, *replaceWith*) (en-US)

Se utiliza para reemplazar todas las apariciones de *searchFor* usando *replaceWith*. *searchFor* puede ser una cadena o expresión regular, y *replaceWith* puede ser una cadena o función.

String.prototype.search(*regexp*)

Busca una coincidencia entre una expresión regular *regexp* y la cadena llamadora.

String.prototype.slice(*beginIndex*[, *endIndex*])

Extrae una sección de una cadena y devuelve una nueva cadena.

String.prototype.split([*sep*[, *limit*]])

Devuelve un arreglo de cadenas pobladas al dividir la cadena llamadora en las ocurrencias de la subcadena *sep*.

String.prototype.startsWith(*searchString*[, *length*])

Determina si la cadena llamadora comienza con los caracteres de la cadena *searchString*.

String.prototype.substr()

Devuelve los caracteres en una cadena que comienza en la ubicación especificada hasta el número especificado de caracteres.

String.prototype.substring(*indexStart*[, *indexEnd*])

Devuelve una nueva cadena que contiene caracteres de la cadena llamadora de (o entre) el índice (o indeces) especificados.

String.prototype.toLocaleLowerCase([*locale*, ...*locales*])

Los caracteres dentro de una cadena se convierten a minúsculas respetando la configuración regional actual.

Para la mayoría de los idiomas, devolverá lo mismo que toLowerCase().

String.prototype.toLocaleUpperCase([*locale*, ...*locales*])

Los caracteres dentro de una cadena se convierten a mayúsculas respetando la configuración regional actual.

Para la mayoría de los idiomas, devolverá lo mismo que toUpperCase().

String.prototype.toLowerCase()

Devuelve el valor de la cadena llamadora convertido a minúsculas.

String.prototype.toString()

Devuelve una cadena que representa el objeto especificado. Redefine el método Object.prototype.toString().

String.prototype.toUpperCase()

Devuelve el valor de la cadena llamadora convertido a mayúsculas.

String.prototype.trim()

Recorta los espacios en blanco desde el principio y el final de la cadena. Parte del estándar ECMAScript 5.

String.prototype.trimStart() (en-US)

Recorta los espacios en blanco desde el principio de la cadena.

String.prototype.trimEnd()

Recorta los espacios en blanco del final de la cadena.

String.prototype.valueOf()

Devuelve el valor primitivo del objeto especificado. Redefine el método Object.prototype.valueOf().

String.prototype.@@iterator() (en-US)

Devuelve un nuevo objeto Iterator que itera sobre los puntos de código de un valor de cadena, devolviendo cada punto de código como un valor de cadena.

Date

Sobre la clase Date recae todo el trabajo con fechas en Javascript, como obtener una fecha, el día, la hora actual y otras cosas. Para trabajar con fechas necesitamos instanciar un objeto de la clase Date y con él ya podemos realizar las operaciones que necesitemos. Un objeto de la clase Date se puede crear de dos maneras distintas. Por un lado, podemos crear el objeto con el día y hora actuales y por otro podemos crearlo con un día y hora distintos a los actuales. Esto depende de los parámetros que pasemos al construir los objetos.

Para crear un objeto fecha con el día y hora actuales colocamos los paréntesis vacíos al llamar al constructor de la clase Date.

```
miFecha = new Date()
```

Para crear un objeto fecha con un día y hora distintos de los actuales tenemos que indicar entre paréntesis el momento con que inicializar el objeto. Hay varias maneras de expresar un día y horas válidas, por eso podemos construir una fecha guiándonos por varios esquemas. Estos son dos de ellos, suficientes para crear todo tipo de fechas y horas.

```
miFecha = new Date(año,mes,dia,hora,minutos,segundos)
miFecha = new Date(año,mes,dia)
```

Los valores que debe recibir el constructor son siempre numéricos. Un detalle, el mes comienza por 0, es decir, enero es el mes 0. Si no indicamos la hora, el objeto fecha se crea con hora 00:00:00.

Los objetos de la clase Date no tienen propiedades, pero si métodos.

getDate()

Devuelve el día del mes.

getDay()

Devuelve el día de la semana.

getHours()

Retorna la hora.

getMinutes()

Devuelve los minutos.

getMonth()

Devuelve el mes (atención al mes que empieza por 0).

getSeconds()

Devuelve los segundos.

getTime()

Devuelve los milisegundos transcurridos entre el día 1 de enero de 1970 y la fecha correspondiente al objeto al que se le pasa el mensaje.

getFullYear()

Retorna el año con todos los dígitos. Usar este método para estar seguros de que funcionará todo bien en fechas posteriores al año 2000.

setDate()

Actualiza el día del mes.

setHours()

Actualiza la hora.

setMinutes()

Cambia los minutos.

setMonth()

Cambia el mes (atención al mes que empieza por 0).

setSeconds()

Cambia los segundos.

setTime()

Actualiza la fecha completa. Recibe un número de milisegundos desde el 1 de enero de 1970.

setFullYear()

Cambia el año de la fecha al número que recibe por parámetro. El número se indica completo ej: 2005 o 1995. Utilizar este método para estar seguros de que todo funciona para fechas posteriores a 2000.

Math

Math es un objeto incorporado que tiene propiedades y métodos para constantes y funciones matemáticas. No es un objeto de función. Math funciona con el tipo Number. No funciona con BigInt.

A diferencia de los demás objetos globales, el objeto Math no se puede editar. Todas las propiedades y métodos de Math son estáticos. Usted se puede hacer referencia a la constante *pi* como Math.PI y llamar a la función *seno* como Math.sin(x), donde **x** es el argumento del método. Las constantes se definen con la precisión completa de los números reales en JavaScript.

Propiedades

Math.E

Constante de Euler, la base de los logaritmos naturales, aproximadamente 2.718.

Math.LN2

Logaritmo natural de 2, aproximadamente 0.693.

Math.LN10

Logaritmo natural de 10, aproximadamente 2.303.

Math.LOG2E

Logaritmo de E con base 2, aproximadamente 1.443.

Math.LOG10E

Logaritmo de E con base 10, aproximadamente 0.434.

Math.PI

Ratio de la circunferencia de un círculo respecto a su diámetro, aproximadamente 3.14159.

Math.SQRT1_2

Raíz cuadrada de 1/2; Equivalentemente, 1 sobre la raíz cuadrada de 2, aproximadamente 0.707.

Math.SQRT2

Raíz cuadrada de 2, aproximadamente 1.414.

Metodos

Math.abs(x)

Devuelve el valor absoluto de un número.

Math.acos(x)

Devuelve el arco coseno de un número.

Math.acosh(x)

Devuelve el arco coseno hiperbólico de un número.

Math.asin(x)

Devuelve el arco seno de un número.

Math.asinh(x)

Devuelve el arco seno hiperbólico de un número.

Math.atan(x)

Devuelve el arco tangente de un número.

Math.atanh(x)

Devuelve el arco tangente hiperbólico de un número.

Math.atan2(y, x)

Devuelve el arco tangente del cuociente de sus argumentos.

Math.cbrt(x)

Devuelve la raíz cúbica de un número.

Math.ceil(x)

Devuelve el entero más pequeño mayor o igual que un número.

Math.clz32(x) (en-US)

Devuelve el número de ceros iniciales de un entero de 32 bits.

Math.cos(x)

Devuelve el coseno de un número.

Math.cosh(x) (en-US)

Devuelve el coseno hiperbólico de un número.

Math.exp(x)

Devuelve E^x, donde *x* es el argumento, y E es la constante de Euler (2.718...), la base de los logaritmos naturales.

Math.expm1(x)

Devuelve e^x - 1.

Math.floor(x)

Devuelve el mayor entero menor que o igual a un número.

Math.fround(x) (en-US)

Devuelve la representación flotante de precisión simple más cercana de un número.

Math.hypot([x[, y[, …]]])

Devuelve la raíz cuadrada de la suma de los cuadrados de sus argumentos.

Math.imul(x, y) (en-US)

Devuelve el resultado de una multiplicación de enteros de 32 bits.

Math.log(x)

Devuelve el logaritmo natural (log, también ln) de un número.

Math.log1p(x) (en-US)

Devuelve el logaritmo natural de x + 1 (loge, también ln) de un número.

Math.log10(x)

Devuelve el logaritmo en base 10 de x.

Math.log2(x)

Devuelve el logaritmo en base 2 de x.

Math.max([x[, y[, …]]])

Devuelve el mayor de cero o más números.

Math.min([x[, y[, …]]])

Devuelve el más pequeño de cero o más números.

Math.pow(x, y)

Las devoluciones de base a la potencia de exponente, que es, baseexponent.

Math.random()

Devuelve un número pseudo-aleatorio entre 0 y 1.

Math.round(x)

Devuelve el valor de un número redondeado al número entero más cercano.

Math.sign(x)

Devuelve el signo de la x, que indica si x es positivo, negativo o cero.

Math.sin(x)

Devuelve el seno de un número.

Math.sinh(x) (en-US)

Devuelve el seno hiperbólico de un número.

Math.sqrt(x)

Devuelve la raíz cuadrada positiva de un número.

Math.tan(x)

Devuelve la tangente de un número.

Math.tanh(x)

Devuelve la tangente hiperbólica de un número.