Desarrollo Web en entorno cliente

UT 3. Objetos predefinidos en Javascript

T.S. en Desarrollo de Aplicaciones Web

IES Escultor J.L. Sánchez

Contenido

1. Objetos predefinidos en Javascript

En JS existen una serie de objetos predefinidos. Entre otros, encontramos los siguientes: *String*, *Math*, *Number*, *Boolean*, *Date*, *Array*

- A pesar de que existen objetos definidos para encapsular los tipos primitivos *number*, *boolean* y *string*, resulta más rápido, siempre que sea posible, trabajar con los tipos primitivos.

String

Almacena un conjunto de caracteres. El primer carácter ocupa la posición 0 del *string*.
Como ya sabemos, los valores de tipo *string* van encerrados entre comillas simples o dobles.

Los objetos de tipo *String* disponen de **propiedades** y **métodos**; los literales de tipo *string* también pueden hacer uso de estas propiedades y métodos, ya que JS los convierte de forma temporal en objetos.

Algunas **propiedades** de *String*:

Propiedades del objeto String	
Propiedad	**Descripción**
`length`	Contiene la longitud de una cadena.

Para instanciar un objeto de tipo *String* se utiliza el constructor: *new String(string s)*

Ejemplo:

```
var s= new String("cadena de caracteres en JS");
```

Algunos **métodos** de *String*:

Métodos del objeto String	
Métodos	**Descripción**
`charAt()`	Devuelve el carácter especificado por la posición que se indica entre paréntesis.
`charCodeAt()`	Devuelve el Unicode del carácter especificado por la posición que se indica entre paréntesis.
`concat()`	Une una o más cadenas y devuelve el resultado de esa unión.

`fromCharCode()`	Convierte valores Unicode a caracteres.
`indexOf()`	Devuelve la posición de la primera ocurrencia del carácter buscado en la cadena.
`lastIndexOf()`	Devuelve la posición de la última ocurrencia del carácter buscado en la cadena.
`match()`	Busca una coincidencia entre una expresión regular y una cadena y devuelve las coincidencias o null si no ha encontrado nada.
`replace()`	Busca una subcadena en la cadena y la reemplaza por la nueva cadena especificada.
`search()`	Busca una subcadena en la cadena y devuelve la posición dónde se encontró.
`slice()`	Extrae una parte de la cadena y devuelve una nueva cadena.
`split()`	Divide una cadena en un array de subcadenas.
`substr()`	Extrae los caracteres de una cadena, comenzando en una determinada posición y con el número de caracteres indicado.
`substring()`	Extrae los caracteres de una cadena entre dos índices especificados.
`toLowerCase()`	Convierte una cadena en minúsculas.
`toUpperCase()`	Convierte una cadena en mayúsculas.

Un objeto de tipo *String* es invariable; por tanto, los métodos que devuelven un nuevo objeto con el resultado, no modifican el objeto implícito.

El método *substring()* extrae y devuelve caracteres de la cadena implícita desde la *posición inicial* indicada en el parámetro, hasta la *posición final - 1* indicada en el segundo parámetro. Si se omite el segundo parámetro, se extraen y devuelven caracteres desde la posición inicial hasta el final de la cadena implícita. No acepta parámetros negativos. El segundo parámetro es opcional.

El método *slice()* extrae y devuelve caracteres de la cadena implícita desde la *posición inicial* indicada en el parámetro, hasta la *posición final - 1* indicada en el segundo parámetro. Admite parámetros negativos; en ese caso, comienza a extraer caracteres desde el final del *string*. El segundo parámetro es opcional.

El método *substr()* extrae y devuelve caracteres del string implícito, empezando desde la posición indicada en el primer parámetro y tantos caracteres como indique el segundo. Si el primer parámetro es negativo, la extracción de caracteres comienza desde el final de la cadena. El segundo parámetro es opcional, pero si existe no puede ser negativo.

En ES6, el objeto **String** incluye algunos nuevo métodos; por ejemplo:

- *startsWith*(): comprueba si una cadena comienza por otra
- *endsWith*(): comprueba si una cadena finaliza por otra
- *includes*(): comprueba si una cadena está incluida en otra
- *repeat*(): repite una cadena *n* veces

Más información:
https://www.w3schools.com/jsref/jsref_obj_string.asp

Ejemplos de uso de objetos *String*:

```
var cadena="El parapente es un deporte de riesgo medio";
document.write("La longitud de la cadena es: "+ cadena.length + "<br/>");
document.write(cadena.toLowerCase()+ "<br/>");
document.write(cadena.charAt(3)+ "<br/>");
document.write(cadena.indexOf('pente')+ "<br/>");
document.write(cadena.substring(3,16)+ "<br/>");
```

Actividad: ¿Qué resultados se obtienen al ejecutar las instrucciones del ejemplo anterior?

El tipo primitivo *string* vs *objetos String*

Se recomienda trabajar con variables de tipo primitivo en lugar de objetos de tipo *String* ya que el manejo de objetos ralentiza la velocidad de ejecución.

Por otra parte, a la hora de comparar referencias a objetos hay que tener en cuenta que éstas apuntan a direcciones de memoria:

```
var x = new String("John");
var y = new String("John");
// (x == y) es false porque x e y referencian direcciones de memoria
//distintas
```

Templates literales en ES6

Los **template literals** son literales de texto que nos permite utilizar varias lineas e interpolar expresiones. Se delimitan con el carácter de comillas o tildes invertidas (` `) , en lugar de las comillas simples o dobles. Para interpolar expresiones se utiliza la sintaxis *${expresión}*

Ejemplo de uso de cadenas de más de una línea:

```
//ES6
// Cadenas de texto que ocupan varias lineas
console.log(`línea 1 de la cadena de texto
línea 2 de la cadena de texto`);
// línea 1 de la cadena de texto
// línea 2 de la cadena de texto
```

Ejemplo de uso de interpolación de string:

```
let precio = 2950 ;
const IVA = 1.21

let cadena = `El precio es ${precio*IVA} euros` ;
console .log(cadena);
```

Number

Number encapsula el tipo primitivo *number.* Normalmente, el tipo primitivo es suficiente para trabajar en JS.

Algunas **propiedades** de *Number*:

Propiedad	Descripción
MAX_VALUE	Devuelve el número más alto disponible en JavaScript.
MIN_VALUE	Devuelve el número más pequeño disponible en JavaScript.
POSITIVE_INFINITY	Representa infinito positivo (se devuelve en caso de overflow).
NEGATIVE_INFINITY	Representa infinito negativo (se devuelve en caso de overflow)
NaN	Representa el valor Not-a-Number

Algunos **métodos** de *Number*:

Para instanciar un objeto de tipo *Number* se utiliza el constructor: *new Number(value)*

Método	Descripción
Number.isFinite(value)	Devuelve *true* si el valor es de tipo *Number* y equivale a un número finito
Number.isInteger(value)	Devuelve *true* si el valor es de tipo *Number* y es, además, si es un número entero.
Number.isNaN(value)	Devuelve *true* si el valor es de tipo *Number* y equivale a *NaN*
toFixed(x)	Convierte el número a *string* con *x* decimales. El valor de *x* por defecto es 0
toPrecision(x)	Convierte el número a *string* con una longitud *x*, añadiendo un punto decimal o 0 si fuese necesario.
toString()	Convierte el número en *string*. Por parámetro se puede indicar la base de numeración a utilizar.
valueOf()	Devuelve el valor primitivo del número

Algunos de estos métodos existen como funciones globales de JS (que estudiaremos más adelante). El funcionamiento de unos y otros es semejante, aunque existen algunas diferencias.

Es importante revisar a qué versión de JS pertenecen los métodos y, en consecuencia, la versión del navegador que lo soporta.

Más información sobre *Number*:
https://www.w3schools.com/jsref/jsref_obj_number.asp

Ejemplos de uso de *Number*:

```
var num = new Number(13.3714);
document.write(num.toPrecision(3)+"<br>");
document.write(num.toFixed(1)+"<br>");
document.write(num.toString(2)+"<br>");
document.write(num.toString(8)+"<br>");
document.write(num.toString(16)+"<br>");
document.write(Number.MIN_VALUE +"<br>");
document.write(Number.MAX_VALUE +"<br>");
```

Actividad: ¿Qué resultados se obtienen al ejecutar las instrucciones las instrucciones de ejemplo?

Más ejemplos:

```
Number.isInteger(0) //true
Number.isInteger(0.5) //false
Number.isInteger('123') //false
Number.isFinite(123) //true
Number.isFinite(-1.23) //true
Number.isFinite(5-2) //true
Number.isFinite(0) //true
Number.isFinite('123') //false
Number.isFinite('Hello') //false
Number.isNaN(123) //false
Number.isNaN('NaN') //false
Number.isNaN(NaN) //true
Number.isNaN(0 / 0) //true
```

Math

Math dispone de una serie de propiedades y métodos estáticos que implementan operaciones con números.

Algunas **propiedades** de *Math*

Propiedad	Descripción
E	Devuelve el número E
PI	Devuelve el número PI
SQRT	Devuelve la raíz cuadrada de 2 (aproximadamente 1.414).

Algunos **métodos** de *Math*:

Método	Descripción
Math.abs(x)	Devuelve el valor absoluto de x.
Math.acos(x)	Devuelve el arcocoseno de x, en radianes.
Math.asin(x)	Devuelve el arcoseno de x, en radianes.
Math.atan(x)	Devuelve el arcotangente de x, en radianes con un valor entre -PI/2 y PI/2.
Math.ceil(x)	Devuelve el número x redondeado al alta hacia el siguiente entero.
Math.cos(x)	Devuelve el coseno de x (x está en radianes).
Math.floor(x)	Devuelve el número x redondeado a la baja hacia el anterior entero.
Math.max(x1,x2,…)	Devuelve el número más alto de los que se pasan como parámetros.
Math.min(x1,x2,…)	Devuelve el número más bajo de los que se pasan como parámetros.
Math.pow(x,y)	Devuelve el resultado de x elevado a y.
Math.radom()	Devuelve un número al azar entre 0 y 1.
Math.round(x)	Redondea x al entero más próximo.
Math.sin(x)	Devuelve el seno de x (x está en radianes).
Math.sqrt(x)	Devuelve la raíz cuadrada de x.
Math.tan(x)	Devuelve la tangente de un ángulo (en radianes)
Math.trunc(x)	Devuelve la parte entera del número x (ES6)

Más información: https://www.w3schools.com/jsref/jsref_obj_math.asp

Ejemplos:

```
var a = Math.ceil(5); //5
var b = Math.ceil(5.1); //6
var c = Math.ceil(-5.1); //-5
var d = Math.ceil(-5.9); //-5
var e = Math.floor(5); //5
var f = Math.floor(5.1); //5
var g = Math.floor(-5.1); //-6
var h = Math.floor(-5.9); //-6
var i = Math.tan(90); // -1.995200412208242
var j = Math.max(0, 150, 30, 20, 38); //150
var k = Math.pow(0, 1); //0
var l = Math.pow(2, 4); //16
```

Boolean

El objeto *Boolean* encapsula valores booleanos.
Una de sus posibles utilidades es la de conseguir valores booleanos a partir de datos de cualquier otro tipo.
Dependiendo de lo que reciba el constructor de la clase *Boolean,* el valor del objeto booleano que se crea será verdadero o falso, de la siguiente manera:

- Se inicializa a *false* cuando no se pasa valor al constructor, o se pasa una cadena vacía, el número 0, el valor *false*, *NaN*, *null* o un valor indefinido.
- Se inicializa a *true* cuando recibe cualquier valor entrecomillado o cualquier número distinto de 0.

Date

Se utiliza para trabajar con fechas y horas.
Hay 4 formas de instanciar un objeto de tipo *Date*:

```
var d = new Date(); //instante actual
var d = new Date(milliseconds);
var d = new Date(dateString); //fecha indicada en la cadena, en formato JS
var d = new Date(year, month, day, hours, minutes, seconds, milliseconds);
```

- Un objeto de tipo *Date* representa un instante de tiempo expresado en milisegundos (desde el 1 de enero de 1970).
- El mes de enero se corresponde con el número 0.
- El último constructor permite omitir los 4 últimos parámetros.

Formatos de fechas en JS:

- **ISO 8601**: *AAAA-MM-DD*
- Formato **corto**: *MM/DD/YYYY*
- Formato **largo**: *MMM DD YYYY*
  - El mes se puede escribir completo o abreviado (en inglés), en mayúscula o minúscula
  - El mes y el día puede estar en cualquier orden.
  - Se puede utilizar la coma (',') como separador

Algunos **métodos** de *Date*:

Método	Descripción
getDate()	Devuelve el día del mes (de 1-31).
getDay()	Devuelve el día de la semana (de 0-6).
getFullYear()	Devuelve el año (4 dígitos).
getHours()	Devuelve la hora (de 0-23).
getMilliseconds()	Devuelve los milisegundos (de 0-999).
getMinutes()	Devuelve los minutos (de 0-59).
getMonth()	Devuelve el mes (de 0-11).
getSeconds()	Devuelve los segundos (de 0-59).
getTime()	Devuelve los milisegundos desde media noche del 1 de Enero de 1970.
Date.parse(f)	Transforma la cadena de fecha *f* en milisegundos

setDate(d)	Ajusta el día del mes del objeto (de 1-31).
setFullYear(y)	Ajusta el año del objeto (4 dígitos).
setHours(h)	Ajusta la hora del objeto (de 0-23).
setMinutes(m)	Ajusta los minutos del objeto (de 0-59).
setMonth(m)	Ajusta el mes del objeto (de 0-11).
setSeconds()	Ajusta los segundos del objeto (de 0-59).
setTime(ms)	Establece una fecha, dada en milisegundos, a partir del 1 de enero de 1970
toLocaleDateString()	Devuelve la fecha en forma de string y con el formato local
toLocaleTimeString()	Devuelve la hora en forma de string y con el formato local

Más información sobre *Date*:

https://www.w3schools.com/jsref/jsref_obj_date.asp

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Date

2. Objetos del navegador en Javascript

Existen una serie de objetos predefinidos en Javascript que están relacionados con el navegador. Entre otros, encontramos los siguientes: **window**, **location**, **navigator** e **history**

window

Representa una ventana abierta en un navegador.

Este objeto ofrece una serie de métodos y propiedades para controlar la ventana del navegador. Con ellos podemos controlar el aspecto de la ventana, la barra de estado, abrir ventanas secundarias, etc.

Además de ofrecer control, el objeto *window* da acceso a otros objetos como el documento (la página web que se está visualizando), el historial de páginas visitadas o los distintos *frames* de la ventana. De modo que para acceder a cualquier otro objeto de la jerarquía deberíamos empezar por el objeto *window*. Tanto es así que Javascript entiende perfectamente que la jerarquía empieza en *window* aunque no lo señalemos.

Sintaxis para acceder a las propiedades y métodos del objeto *window*:

```
window.nombrePropiedad
window.nombreMétodo( parámetros)
```

Debido a que el objeto *window* siempre estará presente cuando se ejecute un script, se puede omitir la palabra *window* en referencias a los objetos dentro de la ventana principal:

nombrePropiedad
nombreMétodo(parámetros)

Un script no creará nunca explícitamente la ventana principal de un navegador. Es el usuario, quien realiza esa tarea abriendo una URL en el navegador. Sin embargo, un script que esté ejecutándose en una de las ventanas principales del navegador, sí que podrá crear o abrir nuevas sub-ventanas.

El método que genera una nueva ventana es *window.open()*. Este método contiene varios parámetros que definen las características de la nueva ventana; entre otros: la URL del documento a abrir, el nombre de esa ventana y su apariencia física (tamaño, color,etc.).

Por ejemplo, la siguiente instrucción abre una nueva ventana de un tamaño determinado y con el contenido de un documento HTML:

```
var subVentana=window.open("nueva.html","nueva","height=800,width=600");
```

Guardando la nueva ventana creada en la variable *subVentana* se podrá referenciar desde el script original de la ventana principal. Por ejemplo, si quisiéramos cerrar la nueva ventana desde nuestro script, simplemente tendríamos que hacer: *subVentana.close()*
Aquí sí que es necesario especificar *subVentana*, ya que si escribiéramos *window.close()* o *close()* estaríamos cerrando nuestra propia ventana (previa confirmación), pero no la *subVentana* que creamos en los pasos anteriores.

Algunas **propiedades** del objeto *window*:

Propiedad	Descripción
closed	Devuelve un valor `Boolean` indicando cuando una ventana ha sido cerrada o no.
document	Devuelve el objeto `document` de la ventana
frameElement	Devuelve el elemento *frame o iframe* en el que está incluida la ventana; si la ventana no está incluida en un marco, devuelve *null*.
frames	Devuelve un array de todos *frames* e *iframes* de la ventana actual.
history	Devuelve el objeto `history` de la ventana.
length	Devuelve el número de *frames* e *iframes* que hay en dentro de una ventana.
location	Devuelve la localización del objeto ventana (URL del fichero).
name	Ajusta o devuelve el nombre de una ventana.
navigator	Devuelve el objeto `navigator` de una ventana, objeto que contiene información sobre el navegador.
opener	Si desde una ventana se ha abierto otra, *opener* devuelve una referencia a la ventana "generadora" o ventana desde la que se invocó la apertura de la ventana actual. Si es la ventana inicial de navegación, *opener* devuelve *null*.
parent	Referencia a la ventana que aloja una subventana (es decir, ventana que aloja un *frame* o *iframe* típicamente). Si no existe *parent*, devuelve una referencia a sí misma (la propia ventana).
self	Devuelve una referencia a la ventana actual.
status	Ajusta o devuelve el texto de la barra de estado de una ventana.

Algunos **métodos** del objeto *window*:

Método	Descripción
alert()	Muestra una ventana emergente de alerta y un botón de aceptar.
blur()	Elimina el foco de la ventana actual.
clearInterval(id)	Resetea el cronómetro ajustado con `setInterval()`
setInterval(function, milliseconds)	Llama a una función periódicamente en un intervalo especificado (en milisegundos). El método *setInterval ()* continuará llamando a la función hasta que se llame *clearInterval ()* o se cierre la ventana. El valor *ID* devuelto por *setInterval()* se utiliza como parámetro para el método *clearInterval().*
close()	Cierra la ventana actual.
confirm()	Muestra una ventana emergente con un mensaje, un botón de aceptar y un botón de cancelar. Devuelve un valor booleano: *true* si se pulsa aceptar o *false* si se pulsa cancelar.
focus()	Coloca el foco en la ventana actual.
open(url, name, parametros)	Abre una nueva ventana de navegación. Tiene parámetros opcionales: – *url* -> url a cargar en la nueva ventana. – *name*-> nombre, opcional, puede usarse para fijar atributo target en un link. – *parametros* es una lista de parámetros como *left* ó *top* para posición, *height* o *width* para dimensiones, …
prompt()	Muestra una ventana de diálogo para introducir datos.
resizeBy(width, height)	Redimensiona la ventana con los valores indicados por parámetro (en pixels) añadiéndolos o restándolos a su tamaño actual
resizeTo(width, height)	Redimensiona la ventana con los valores indicados por parámetro (en pixels)
setTimeout(function,milisegundos)	Llama a una función después de transcurridos los milisegundos indicados. El valor *ID* devuelto por *setTimeout()* se utiliza como parámetro para el método *clearTimeout().*
clearTimeout(id)	Borra el temporizador establecido por *setTimeout()*

- ✓ Los métodos *blur()* y *focus()* puede que no funcionen como se espera en todos los navegadores debido a la configuración del usuario.

Ejemplos sobre el funcionamiento del objeto *window* en la *w3schools*:
https://www.w3schools.com/jsref/obj_window.asp

location

El objeto *location* contiene información referente al URL actual.

Este objeto es parte del objeto *window* y se accede a él a través de la propiedad *window.location*.

Algunas **propiedades** del objeto *location*:

Propiedad	Descripción
hash	Establece o devuelve la parte # del URL. Cuando se utiliza para establecer la parte #, no incluye el signo #
host	Establece o devuelve el nombre del host y el número del puerto, dentro de la URL. Si el número de puerto no está especificado en la URL (o si se trata de un puerto predeterminado como 80 o 443), algunos navegadores no mostrarán nada.
hostname	Establece o devuelve el nombre del host, dentro de la URL.
href	Establece o devuelve el nombre el URL.
pathname	Establece o devuelve el camino al recurso dentro del URL
port	Establece o devuelve el número de puerto del servidor, dentro de la URL. Si el número de puerto no está especificado en la URL (o si se trata de un puerto predeterminado como 80 o 443), algunos navegadores mostrarán 0 o nada.
protocol	Establece o devuelve el protocolo utilizado (incluyendo los dos puntos), dentro de la URL.
search	Establece o devuelve los parámetros (incluido ?) del URL

Métodos de *location*

Método	Descripción
assing(URL)	Carga un nuevo documento.
reload()	Vuelve a cargar la URL especificada en la propiedad `href` del objeto `location`.
replace(newURL)	Reemplaza el historial actual mientras carga la URL especificada en *newURL*.

La diferencia entre *replace()* y *assign()* es que *replace()* elimina la URL del documento actual del historial del documento, lo que significa que no es posible utilizar el botón "volver" para volver al documento original.

Ejemplos sobre el funcionamiento del objeto *location* en *w3schools*:
https://www.w3schools.com/jsref/obj_location.asp

navigator

Este objeto contiene información sobre el navegador que estamos utilizando cuando abrimos una URL o un documento local.

Algunas **propiedades** del objeto *navigator*:

Propiedad	Descripción
appName	Devuelve el nombre del navegador
geolocation	Devuelve un objeto *Geolocation* que puede usarse para obtener información sobre la ubicación (latitud, longitud) desde donde el usuario navega. Algunos navegadores no responden bien. Otros restringen o piden permiso al usuario por considerar que puede atentar contra su privacidad.
language	Devuelve un *string* representativo del lenguaje del navegador. Algunos navegadores no disponen de *language* pero en su lugar tienen disponible *userLanguage*. Otros navegadores no reconocen ni una ni otra forma.
online	Devuelve un valor booleano (*true* o *falso*) que indica si se está o no con conexión a internet

Información sobre el objeto *navigator*:
https://www.w3schools.com/jsref/obj_navigator.asp

history

El objeto *history* contiene las URLs visitadas por el usuario (dentro de la ventana del navegador)

Este objeto forma parte del objeto *window* y se accede a él a través de *window.history*

Propiedades del objeto *history*

Propiedad	Descripción
length	Devuelve el número de URLs en la lista del historial de la ventana actual del navegador

Métodos del objeto *history*

Método	Descripción
back()	Carga el URL anterior al actual
forward()	Carga el URL siguiente al actual
go(number|URL)	Carga el URL especificado en el parámetro

Ejemplos de uso de *history*:

https://www.w3schools.com/jsref/obj_history.asp

3. Referencias web

- https://www.w3schools.com/jsref/default.asp
- https://desarrolloweb.com/articulos/826.php
- https://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206