**ÍNDICE**

**Jerarquía de Objetos nativos de Javascript**

*Objeto Date*

*Objeto Math...*

*Objeto Number*

*Objeto String*

*Map*

*Set*

**Objetos importantes**

*Windows*

*Document*

**Gestion de ventanas**

*Abrir una Ventana*

*Cerrar una Ventana*

*Comunicacion entre Ventanas*

**3.1. OBJETOS NATIVOS DE JAVASCRIPT**

### OBJETO DATE

|  |  |
| --- | --- |
|  | **Constructores** |
| new **Date**() | Obtiene la fecha del momento actual. |
| new **Date**(*str*) | Convierte el texto con formato YYYY/MM/DD HH:MM:SS a fecha. |
| new **Date**(*num*) | Convierte el número num, en formato Tiempo UNIX, a fecha UTC. |
| new **Date**(*y, m, d, h, min, s, ms*) | Crea una fecha UTC a partir de componentes numéricos\*. |
|  | **Métodos similares a constructores** |
| Date.**now**() | Devuelve el Tiempo UNIX de la fecha actual. Equivalente a +new Date(). |
| Date.**parse**(*str*) | Convierte una cadena de fecha a Tiempo UNIX. Equivalente a +new Date(str). |
|  | **Getter de fechas** |
| .**getDay**() | Devuelve el día de la semana: OJO: 0 Domingo, 6 Sábado. |
| .**getFullYear**() | Devuelve el año con 4 cifras. |
| .**getMonth**() | Devuelve la representación interna del mes. OJO: 0 Enero - 11 Diciembre. |
| .**getDate**() | Devuelve el día del mes. |
| .**getHours**() | Devuelve la hora. OJO: Formato militar; 23 en lugar de 11. |
| .**getMinutes**() | Devuelve los minutos. |
| .**getSeconds**() | Devuelve los segundos. |
| .**getMilliseconds**() | Devuelve los milisegundos. |
| .**getTime**() | Devuelve el unix timestamp: segundos transcurridos desde 1/1/1970. |
| .**getTimezoneOffset**() | Diferencia horaria (en min) de la hora local respecto a UTC |
|  | **Setter de fechas** |
| .**setFullYear**(*year*)  .**setFullYear**(*y, m, d*) | Altera el año de la fecha, cambiándolo por year. Formato de 4 dígitos. |
| .**setMonth**(*month*)  .**setMonth**(*m, d)* | Altera el mes de la fecha, cambiándolo por month. Ojo: 0-11 (Ene-Dic). |
| .**setDate**(*day*) | Altera el día de la fecha, cambiándolo por day. |
| .**setHour**(*hour*) .**setHour**(*h, m, s, ms)* | Altera la hora de la fecha, cambiándola por hour. |
| .**setMinutes**(*min*)  .**setMinutes**(*m, s, ms*) | Altera los minutos de la fecha, cambiándolos por min. |
| .**setSeconds**(*sec*)  .**setSeconds**(*s, ms*) | Altera los segundos de la fecha, cambiándolos por sec. |
| .**setMilliseconds**(*ms*) | Altera los milisegundos de la fecha, cambiándolos por ms. |
| .**setTime**(*ts*) | Establece una fecha a partir del **tiempo Unix** ts. |
|  | **Formato de fechas** |
| .**toDateString**() | Devuelve formato sólo de fecha: Fri Aug 24 2018 |
| .**toLocaleDateString**() | Idem al anterior, pero en el formato regional actual: 24/8/2018 |
| .**toTimeString**() | Devuelve formato sólo de hora: 00:23:24 GMT+0100 … |
| .**toLocaleTimeString**() | Idem al anterior, pero en el formato regional actual: 0:26:37 |
| .**toISOString**() | Devuelve la fecha en el formato ISO 8601: 2018-08-23T23:27:29.380Z |
| .**toJSON**() | Idem al anterior, pero asegurándose que será compatible con JSON. |
| .**toUTCString**() | Devuelve la fecha, utilizando UTC . |

|  |
| --- |
| ***// INSTANCIAR***  **var** f = **new** **Date** (); *// Fecha actual*  var f = **new** **Date** (*'2018/01/30 23:30:14'*); *// Fecha mediante cadena* var f = **new** **Date** (*872817240000*); *// Fecha mediante timestamp* var f = **new** **Date** (*y, m, d, h, min, s, ms*); *// Fecha por componentes numéricos* var f = **new** **Date** *(2020,11,17);*  ***// GETTER*** *const f = new* **Date***("2018/01/30 15:30:10.999");*  *f.***getDay***(); // 2 (Martes)*  *f.***getDate***(); // 30*  *f.***getMonth***(); // 0 (Enero)*  *f.***getFullYear***(); // 2018*  *f.***getHours***(); // 15*  *f.***getMinutes***(); // 30*  *f.***getSeconds***(); // 10*  *f.***getMilliseconds***(); // 999*  *f.***getTimezoneOffset***(); // 0*  *f.***getTime***(); // 1517326210999 (Tiempo Unix)*  ***// SETTER*** *const f = new* **Date***("2018/01/30 15:30:10.999");*  *f.***setDate***(15); // Cambia a 15/01/2018 15:30:10.999 (Devuelve 1516030210999)*  *f.***setMonth***(1); // Cambia a 15/02/2018 15:30:10.999 (Devuelve 1518708610999)*  *f.***setFullYear***(2020); // Cambia a 15/02/2020 15:30:10.999 (Devuelve 1581780610999)*  *f.***setHours***(21); // Cambia a 15/02/2020 21:30:10.999 (Devuelve 1581802210999)*  *f.***setMinutes***(00); // Cambia a 15/02/2020 21:00:10.999 (Devuelve 1581800410999)*  *f.***setSeconds***(3); // Cambia a 15/02/2020 21:00:03.999 (Devuelve 1581800403999)*  *f.***setMilliseconds***(79); // Cambia a 15/02/2020 21:00:03.079 (Devuelve 1581800403079)*  *f.***setTime(***872817240000); // Cambia a 29/08/1997 02:14:00.000 (Devuelve 872817240000)*  *// Devolver en formato texto, mes actual const MESES = [ "Enero", "Febrero","Marzo","Abril","Mayo","Junio","Julio",*  *"Agosto","Septiembre","Octubre","Noviembre","Diciembre"]; const f = new* **Date***(); MESES[f.***getMonth***()];*  f.**getTime**() *// Tiempo en timestamp*  ***// SETTER***  f.**setDate**() *// Modificar día*  f.**setMonth**() *// Modificar mes*  f.**setFullYear**() *// Modificar año*  f.**setSecond**() *// Modificar los segundos*  f.**setMinutes**() *// Modificar los minutos*  f.**setHours**() *// Modificar las horas*  ***// FORMATOS***  f.**toDateString**() *// Fri Aug 24 2018*  f.**tolocaleDateString**() *// 24/8/2018*  f.**tolocaleTimeString**() *// 0/26/37* |

## OBJETO MATH

|  |  |  |
| --- | --- | --- |
|  | **Constantes** | |
| Math**.PI** | **Número PI** | |
| Math**.SQRT2** | **Raíz cuadrada de 2** | |
|  | **Métodos matemáticos** | |
| Math**.abs(***x***)** | **Devuelve el valor absoluto de x. |x|** | |
| Math**.exp(***x***)** | **Exponente, Devuelve el número e elevado a x.** e**x** | |
| Math**.max(***a, b, c...***)** | **Devuelve el número más grande de los indicados por parámetro.** | |
| Math**.min(***a, b, c...***)** | **Devuelve el número más pequeño de los indicados por parámetro.** | |
| Math**.sqrt(***x)* | **Devuelve la raíz cuadrada de x.** | |
|  | **Aleatorio** | |
| Math**.random(**) | **Devuelve un número al azar entre 0 y 1 *(con 16 decimales)*** | |
|  | **Métodos de redondeo** | |
| Math**.round(***x***)** | **Devuelve el redondeo de x**  Entero más cercano | |
| Math**.ceil(***x***)** | **Devuelve el redondeo superior de x. Entero más alto** | |
| Math**.floor(***x***)** | **Devuelve el redondeo inferior de x. Entero más bajo** | |
| Math**.trunc(***x***)** | **Devuelve la parte entera** | |
| Math.**PI** | | *//Constantes*  ***//MÉTODOS*** |
| Math.**abs***(-5)* | | *//5,Valor absoluto* |
| Math.**max***(1,2,3,4)* | | *//4, Máximo de la lista* |
| Math.**min***(1,2,3,4)* | | *//1, Mínimo de la lista* |
| Math.**pow***(x,y)* | | *// x^y* |
| Math.**sqrt***(2)* | | *// Raíz cuadrada de 2* |
| Math.**floor***(4.7)* | | *//4, parte entera, con redondeo inferior* |
| Math.**ceil***(4.7)* | | *//5, parte entera, con redondeo superior* |
| Math.**round***(4.7)* | | *//5, parte entera, con redondeo al más cercano* |
| Math.**trunc***(4.7)* | | *//4, devuelve parte entera(truncamiento)* |
| Math.**random**(); | | *// Número al azar entre [0, 1) con 16 decimales* |
| **var** x = Math.**floor**(Math.**random***() \* 5);* | | *// Número entre 0 y 5.* |

## OBJETO NUMBER

**Number**, es utilizado para valores enteros y decimales.

• **NaN** → **N***ot* **a N***umber*

|  |  |  |
| --- | --- | --- |
|  | **Constantes** | |
| Number**.POSITIVE\_INFINITY** | **Infinito positivo: +∞** | |
| Number**.NEGATIVE\_INFINITY** | **Infinito negativo: -∞** | |
|  | **Comprobar números** | |
| Number**.isFinite(***n***)** | **Comprueba si n es un número finito.** | |
| Number**.isInteger(***n***)** | **Comprueba si n es un número entero.** | |
| Number**.isNaN(***n***)** | **Comprueba si n no es un número.** | |
|  | **Conversión numérica** | |
| Number**.parseInt(***s*) | **Convierte una cadena de texto s en un número entero.** | |
| Number**.parseFloat(***s***)** | **Convierte una cadena de texto s en un número decimal.** | |
|  | **Representación numérica** | |
| **.toExponential(***n***)** | **Convierte el número a notación exponencial con n decimales.** | |
| **.toFixed(***n***)** | **Convierte el número a notación de punto fijo con n decimales.** | |
| **.toPrecision(***p***)** | **Utiliza p dígitos de precisión en el número.** | |
| **var** n = 4; | | *// Literal Numérico* |
| **var** nObj = **new** **Number**(4); | | *// Objeto Numérico* |
| Number.**MAX\_VALUE** | | *//Costantes* |
| Number.**MIN**\_**VALUE** Number.**NaN** | | **//MÉTODOS** |
| Number.**isNaN***(NaN);* | | *//true, es un not a number* |
| Number.**isNaN***(4);* | | *//false, es un número* |
| Number.**isInteger***(4);* | | *//true, Es un entero* |
| Number.**isInteger***(4.7);* | | *//false,Es un decimal* |
| Number.**parseInt***('4');* | | *// Pasar a entero la cadena ‘4’* |
| Number.**parseInt***('11101', 2);* | | *// 29, antes se especificó en binario(b=2)* |
| (1234).**toString**(); | | *// “1234”, pasa a cadena el valor numérico.* |
| (1234).**toString**(2); | | *// “101101”, pasa a binario el valor numérico.* |
| (1.5).**toFixed**(3); | | *//1.500 , Punto fijo con 3 decimales* |
| *(1.5).***toExponential***(2);* | | *// "1.50e+0" en exponencial* |
| *(1.5).***toFixed***(2);* | | *// "1.50" en punto fijo* |
| *(1.5).***toPrecision***(1);* | | *// "2"* |
| **typeof** n; | | *// number* |

## OBJETO STRING

|  |  |
| --- | --- |
|  | **Propiedades** |
| **. length** | **Devuelve el número de carácteres de la variable de tipo string en cuestión.** |
|  | **Métodos posicionales** |
| .**charAt** (*pos*) | Devuelve el carácter en la posición pos de la variable. Similar a [] |
| .**concat** (*str1, str2...*) | Devuelve el texto de la variable unido a str1, a str2… Similar a + |
| .**indexOf** (*str*) | Devuelve la primera posición del texto str. |
| .**indexOf** (*str, from*) | Idem al anterior, partiendo desde la posición from. |
|  | **Métodos de búsqueda** |
| . **includes**(*s, from*) | Comprueba si el texto contiene el subtexto s desde la posición from. |
| . **search**(*regex*) | Busca si hay un patrón que encaje con regex y devuelve la posición. |
| . **match**(*regex*) | Idem a la anterior, pero devuelve las coincidencias encontradas. |
|  | **Métodos de transformar** |
| . **repeat**(*n*) | Devuelve el texto de la variable repetido n veces. |
| . **toLowerCase**() | Devuelve el texto de la variable en minúsculas. |
| . **toUpperCase**() | Devuelve el texto sin espacios a la izquierda y derecha. |
| . **trim**() | Devuelve el texto sin espacios a la izquierda y derecha. |
| . **replace**(*str|regex, newstr*) | Reemplaza la primera aparición del texto str por newstr. |
| . **replaceAll**(*str|regex, newstr*) | Reemplaza todas las apariciones del texto str por newstr. |
| . **substr**(*ini, len*) | Devuelve el subtexto desde la posición ini hasta ini+len. |
| . **split**(*sep|regex, limit)* | Separa el texto usando sep como separador, en limit fragmentos. |
| . **padStart**(*len, str*) | Rellena el principio de la cadena con str hasta llegar al tamaño len. |
| . **padEnd**(*len, str*) | Rellena el final de la cadena con str hasta llegar al tamaño len. |
|  | **Concatenación** |
| **+** | Concatenación de cadenas y variables |
| **Backticks ${}** | Concatenación de cadenas y variables (*en ESMAC2015)* |

|  |  |
| --- | --- |
| **var** s = ‘*cadena’*; | *// Literal Cadena* |
| **var** *sObj* = **new** **String***(‘cadena’)* | *// Objeto String*  ***// PROPIEDADES*** |
| s.**length** | *// 6, número de carścteres* |
| *“Hola”.***lenght** | *// 4, número de carácteres* |
| s **[0]** | *//c,primer caracter*  ***// MÉTODOS*** |
| s.**charAt***(1)* | *// c, carácter en la posición 1* |
| s.**indexOf***(‘den’)* | *// 3, posición 1ª ocurrencia cadena ‘den’, -1 no encontrado* |
| s.**concat***(‘33’,’44’)* | *// cadena3344, concatena todas las cadenas* |
| *"Manz".***concat***("i", "to");* | *// 'Manzito'* |
| *"Manz".***includes***("an");* | *// true ('Manz' incluye 'an')* |
| *"Hola a todos".***search***(/o/g);* | *// busca globalmente las "o", 1, devuelve posición de la 1ª o* |
| *"Hola a todos".***match***(/o/g);* | *// ['o', 'o', 'o'], las 3 "o" que encuentra* |
| *"Na".***repeat***(5);* | *// 'NaNaNaNaNa'* |
| *"MANZ".***toLowerCase***();* | *// 'manz'* |
| *"manz".***toUpperCase***();* | *// 'MANZ'* |
| *" Hola ".***trim***();* | *// 'Hola'* |
| *"Amigo".***replace***("A", "Ene");* | *// 'Enemigo'* |
| *"Dispara".***replace***("a", "i");* | *// 'Dispira' (sólo reemplaza la primera aparición)* |
| *"Dispara".***replace***(/a/g, "i");* | *// 'Dispiri' (reemplaza todas las ocurrencias)* |
| *"Submarino".***substr***(3);* | *// 'marino' (desde el 3 en adelante)* |
| *"Submarino".***substr***(3, 1);* | *// 'm' (desde el 3, hasta el 3+1)* |
| *(1:2:3:4)****.*split**(*‘:’*) | *// Separamos por “,” [1,2,3,4]* |
| *"Código".***split***("");* | *// ['C', 'ó', 'd', 'i', 'g', 'o'] (6 elementos)* |
| *"5".***padStart***(6, "0");* | *// '000005'* |
| *"A".***padEnd***(5, "·");* | *// 'A····'* |
| *const sujeto = "frase";*  *const adjetivo = "concatenada"; "Una " + sujeto + " bien " + adjetivo;* | *// Concatenación antigua* |
| *`Una ${sujeto} mejor ${adjetivo}`* | *// Concatenación actual* |

## MAP

Es una colección de parejas de [clave,valor]. Un objeto en Javascript es un tipo particular de *Map* en que las claves sólo pueden ser texto o números.

let persona = new Map([

['nombre', 'Agustin'],

['apellido', 'Aguilera'],

['edad', 99]

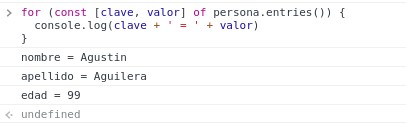
;

])

for (const [clave, valor] of persona.entries()) {

console.log(clave + ' = ' + valor)

}

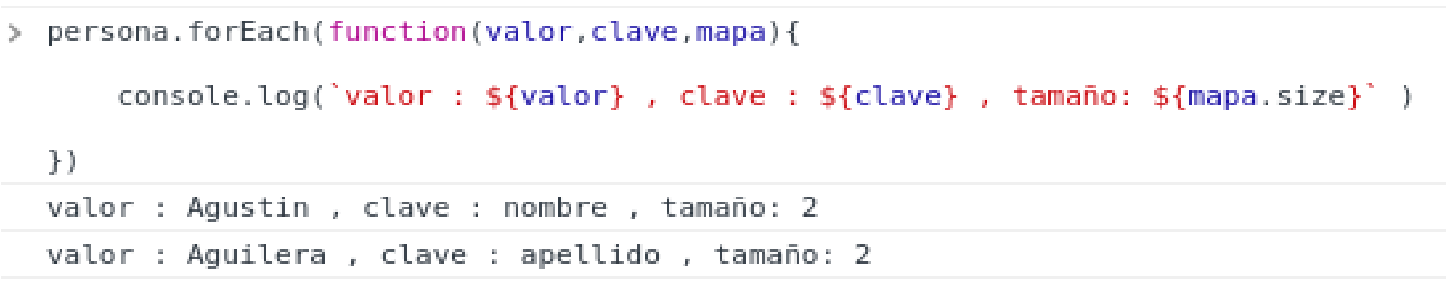


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **Map** | | |
| .***set*** (*clave*, *valor*) | | *almacena el valor asociado a la clave.* | | |
| .***get*** (*clave*) | | devuelve el valor de la clave. Será “undefined” si la clave no existe en map. | | |
| .***has*** (*clave*) | | *Devuelve truesi la clave existe en map,false si no existe.* | | |
| .***delete*** (*clave*) | | *elimina el valor de la clave.* | | |
| .***clear*** () | | *elimina todo de map.* | | |
| .***size*** | | *tamaño, devuelve la cantidad actual de elementos.* | | |
| let persona = new Map(); persona.set('nombre', "Agustin"); persona.set('apellido', "Aguilera"); persona.set('edad', 99);  persona.get("edad"); | | // 99 |
| persona.size | | // 3 |
| persona.delete(“edad”) persona.size | | // 2 |

Para recorrer los valores del map utilizando el método **foreach**:

|  |
| --- |
| persona.forEach(function(valor,clave,mapa){ console.log(`valor : ${valor} ,  clave : ${clave} ,  tamaño: ${mapa.size}` )  }) |

Siendo el resultado de la ejecución:



Otra alternativa sería utilizar **for…..of**

## SET

Es como un *Map* pero que no almacena los valores sino sólo la clave. Podemos verlo como una colección que no permite duplicados. Tiene la propiedad **size** que devuelve su tamaño y los métodos **.add** (añade un elemento), **.delete** (lo elimina) o **.has** (indica si el elemento pasado se encuentra o no en la colección) y también podemos recorrerlo con **.forEach**.

|  |  |
| --- | --- |
| **Map** | |
| .***size*** | *Tamaño, devuelve la cantidad actual de elementos.* |
| .***set*** (*valor*) | *Almacena el valor asociado a la clave.* |
| .***get*** (*valor*) | Devuelve el valor de la clave. Será “undefined” si la clave no existe en map. |
| .***has*** (*valor*) | *Devuelve true si el valor existe en el set, false si no existe.* |
| .***delete*** (*valor*) | *Elimina el valor del set* |
| .***clear*** () | *Elimina todo los valores de la colección* |
| ~~.~~***~~get~~*** ~~(~~*~~valor~~*~~)~~ | **¡¡ Función no implementada en la colección !!** |

Una forma sencilla de eliminar los duplicados de un array es crear con él un *Set*:

|  |
| --- |
| let ganadores = ['Márquez', 'Rossi', 'Márquez', 'Lorenzo', 'Rossi', 'Márquez', 'Márquez']; let ganadoresNoDuplicados = new Set(ganadores); // {'Márquez, 'Rossi', 'Lorenzo'}  // volvemos a convertirlo en un Array.  let ganadoresNoDuplicados = Array.from(new Set(ganadores)); // ['Márquez, 'Rossi',  'Lorenzo'] |

## OBJETOS IMPORTANTES

### Windows

El objeto Window es un objeto que tiene propiedades y controla elementos de lo que ocurre en la “ventana” del navegador.

Los métodos que estudiamos en el tema anterior como alert, prompt, etc. forman parte del objeto Window. Para hacer llamada a estos métodos no hace falta nombrar explícitamente Window (el navegador ya se encarga de ello).

Algunos de los métodos mas importantes no estudiados previamente son:

* **setTimeout(cadenaFuncion,tiempo):**

*Este método ejecuta la llamada a la función proporcionada por la cadena (se puede construir una cadena que lleve parámetros) y la ejecuta pasados los milisegundos que hay en la variable tiempo. Devuelve un identificador del “setTimeout” que nos servirá para referenciarlo si deseamos cancelarlo. SetTimeout solo ejecuta la orden una vez.*

* **setInterval(cadenaFunción, tiempo):**

*Exactamente igual que setTimeout, con la salvedad de que no se ejecuta una vez, sino que se repite cíclicamente cada vez que pasa el tiempo proporcionado.*

* **clearTimeout / clearInterval (id):**

*Se le pasa el identificador del timeout/interval y lo anula.*

|  |
| --- |
| // Creamos un intervalo que cada 15 segundos muestra mensaje hola  Var idA=setInterval(“alert(‘hola’);”,15000);  // Creamos un timeout que cuando pasen 3 segundos muestra mensaje adios  Var idB=setTimeout(“alert(‘adios’);”,3000);  // Creamos un timeout que cuando pasen 5 segundos muestra mensaje Var idC=setTimeout(“alert(‘estonoseve’);”,5000); // Cancelamos el ultimo timeout clearTimeout(idC); |

### 

### Document

Cada documento cargado en una ventana del navegador, será un objeto de tipo document. El objeto document proporciona a los scripts, el acceso a todos los elementos HTML dentro de una página.

Este objeto forma parte además del objeto window , y puede ser accedido a través de la propiedad window.document o directamente document (ya que podemos omitir la referencia a la window actual).

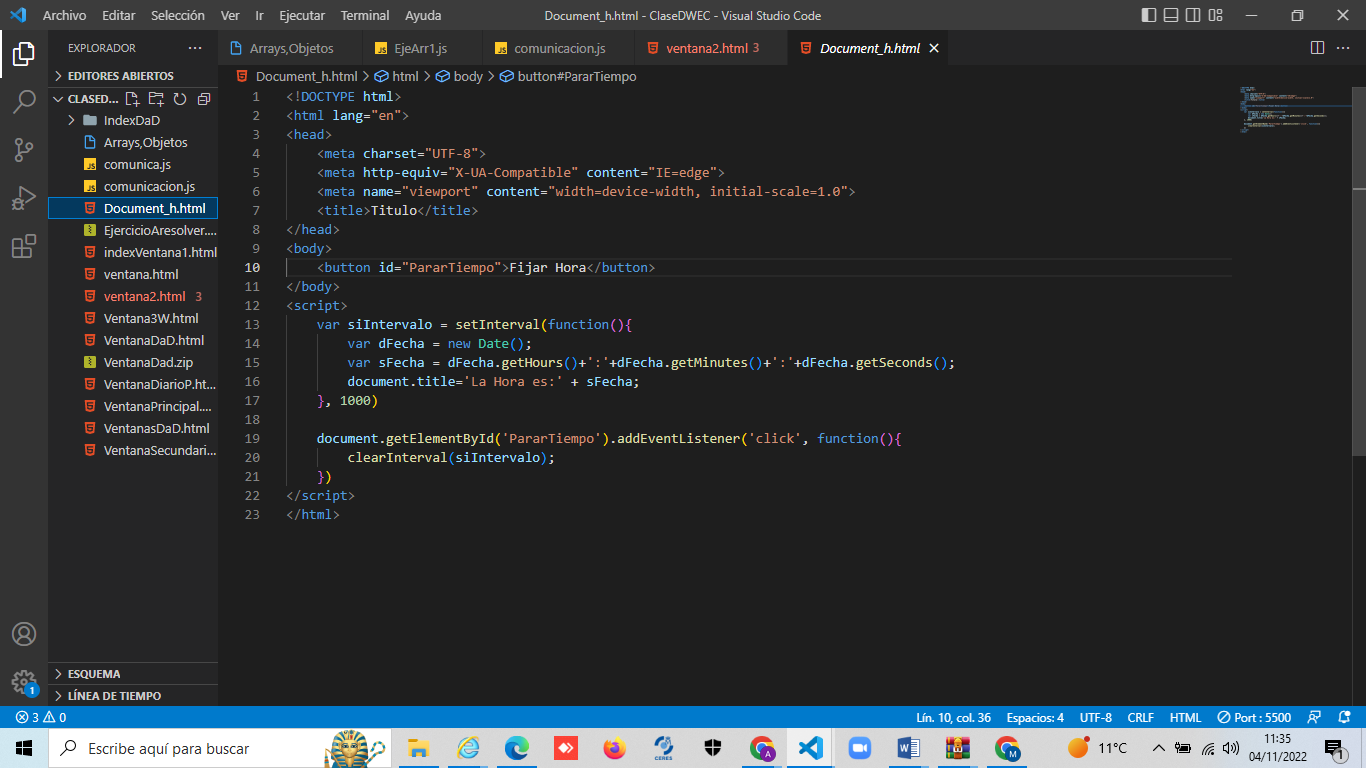
El objeto document nos permite acceder a las siguientes **colecciones**:

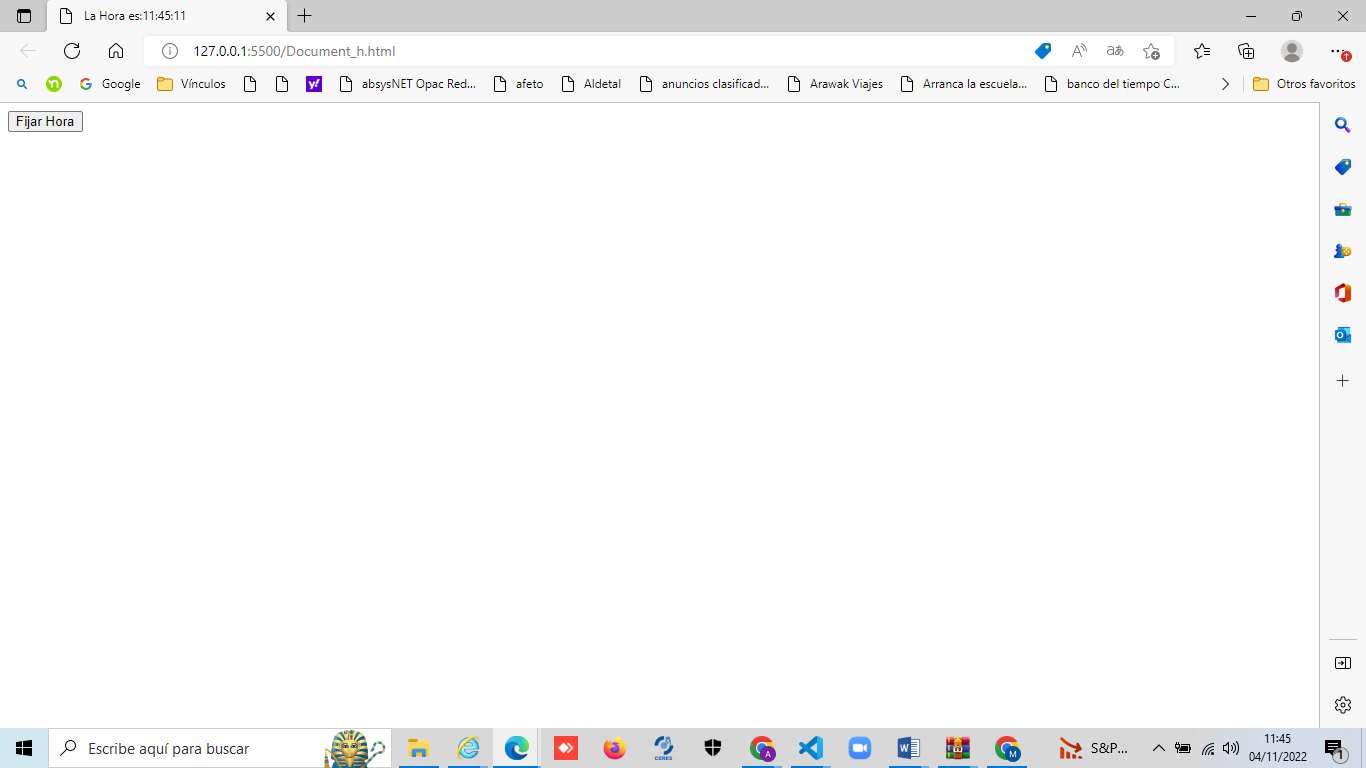
|  |  |
| --- | --- |
| **Colección** | **Descripción** |
| **anchors[]** | *Es un array que contiene todos los hiperenlaces del documento.* |
| **applets[]** | *Es un array que contiene todos los applets del documento.* |
| **forms[]** | *Es un array que contiene todos los formularios del documento.* |
| **images[]** | *Es un array que contiene todas las imágenes del documento.* |
| **links[]** | *Es un array que contiene todos los enlaces del documento.* |

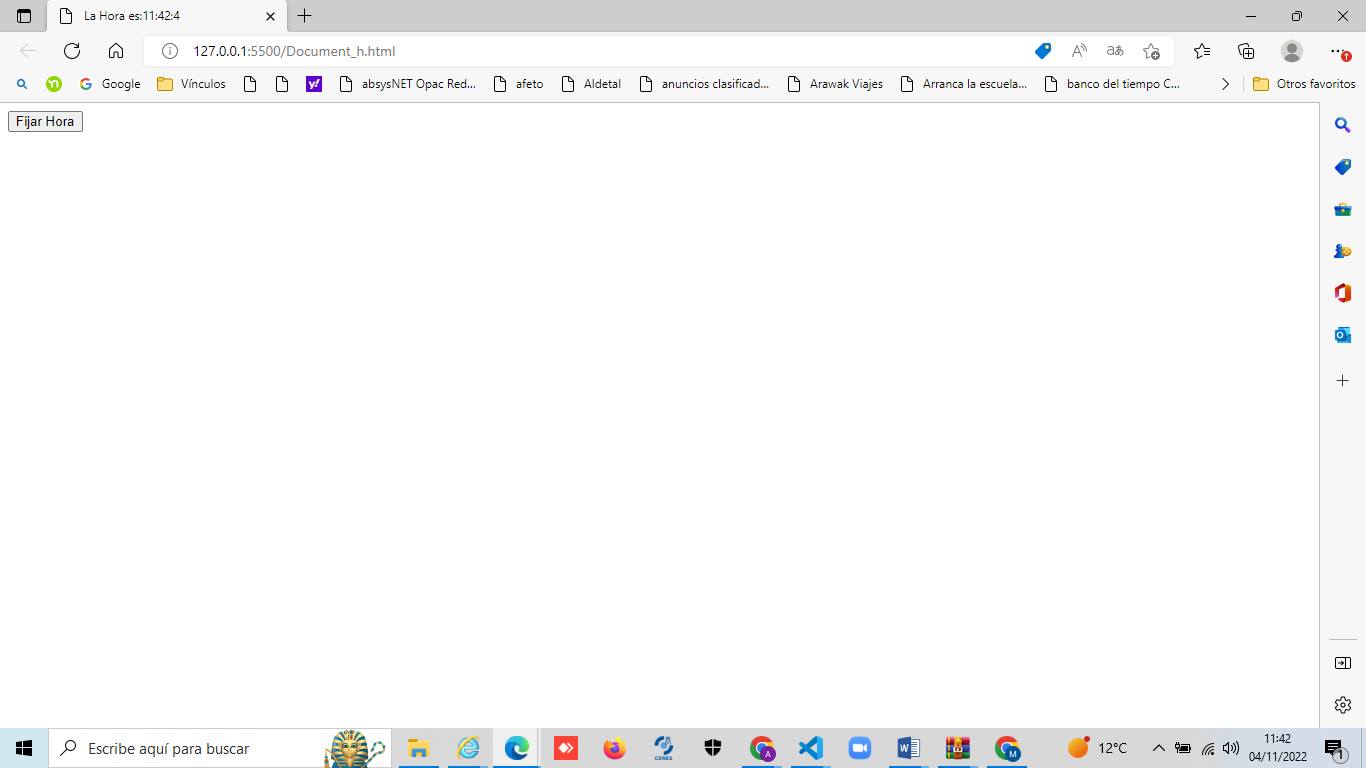
El objeto document nos permite acceder a las siguientes **propiedades y métodos**:

|  |  |
| --- | --- |
| **Propiedad** | **Descripción** |
| ***Cookie*** | *Devuelve todos los nombres/valores de las cookies en el documento* |
| ***Domain*** | *Cadena que contiene el nombre de dominio del servidor que cargó el documento.* |
| ***lastModified*** | *Devuelve la fecha y hora de la última modificación del documento* |
| ***readyState*** | *Devuelve el estado de carga del documento actual* |
| ***Referrer*** | *Cadena que contiene la URL del documento desde el cuál llegamos al documento actual.* |
| ***Title*** | *Devuelve o ajusta el título del documento.* |
| ***URL*** | *Devuelve la URL completa del documento.* |
| **Método** | **Descripción** |
| **getElementById()** | *Para acceder a un elemento identificado por el id escrito entre paréntesis.* |
| **open()** | *Abre el flujo de escritura para poder utilizar document.write() o document.writeln en el documento.* |
| **close()** | *Cierra el flujo abierto previamente con document.open().* |
| **write()** | *Para poder escribir expresiones HTML o código de JavaScript dentro de un documento.* |

*Ejemplo*







## GESTIÓN DE VENTANAS

JavaScript permite gestionar diferentes aspectos relacionados con las ventanas como por ejemplo, abrir nuevas ventanas al presionar un botón. Cada una de estas ventanas tiene un tamaño, posición y estilo diferente. Aclarar que estas ventanas emergentes suelen tener un contenido dinámico.

### Abrir una Ventana

Es una operación muy común en las páginas web y en algunas ocasiones se abren sin que el usuario haga nada. HTML permite abrir nuevas ventanas pero no permite ningún control posterior sobre ellas.

Con JavaScript es posible abrir una ventana vacía mediante el método open():

nuevaVentana = window.open();

El método **open(),** cuenta con cuatro parámetros y todos son opcionales:

* URL.
* Nombre de la ventana.
* Colección de atributos que definen la apariencia de la ventana.
* True (*URL reemplaza al documento actual*), false (*lo* *añade*)

nuevaVentana=**window**.**open**(“http://www.misitioWeb.com/ads”,“Publicidad”, “height=100, width=100”);

### Cerrar una Ventana

Para cerrar una ventana se puede invocar el método **close()**:

|  |
| --- |
| myWindow1.document.write('<input type=button value=Cerrar  onClick=**window**.**close**()>'); |

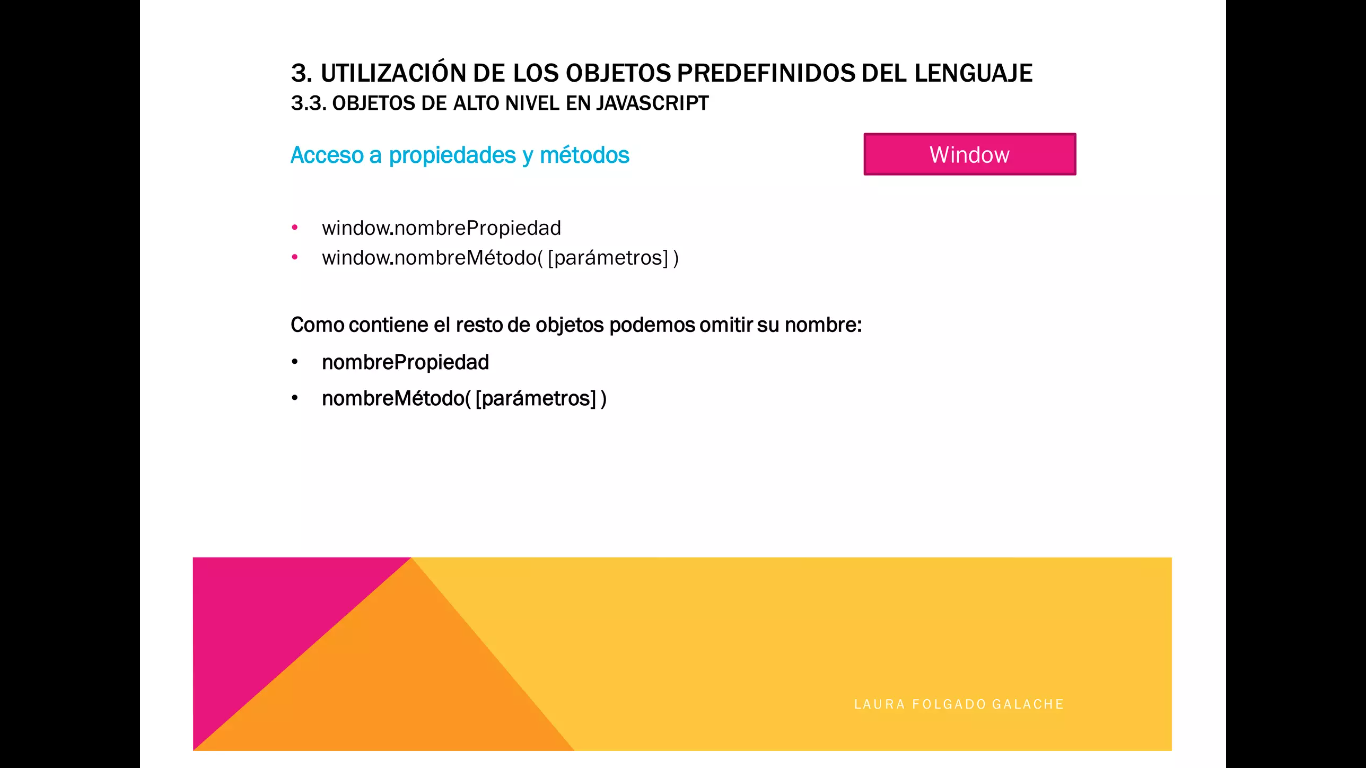
### Comunicación entre Ventanas

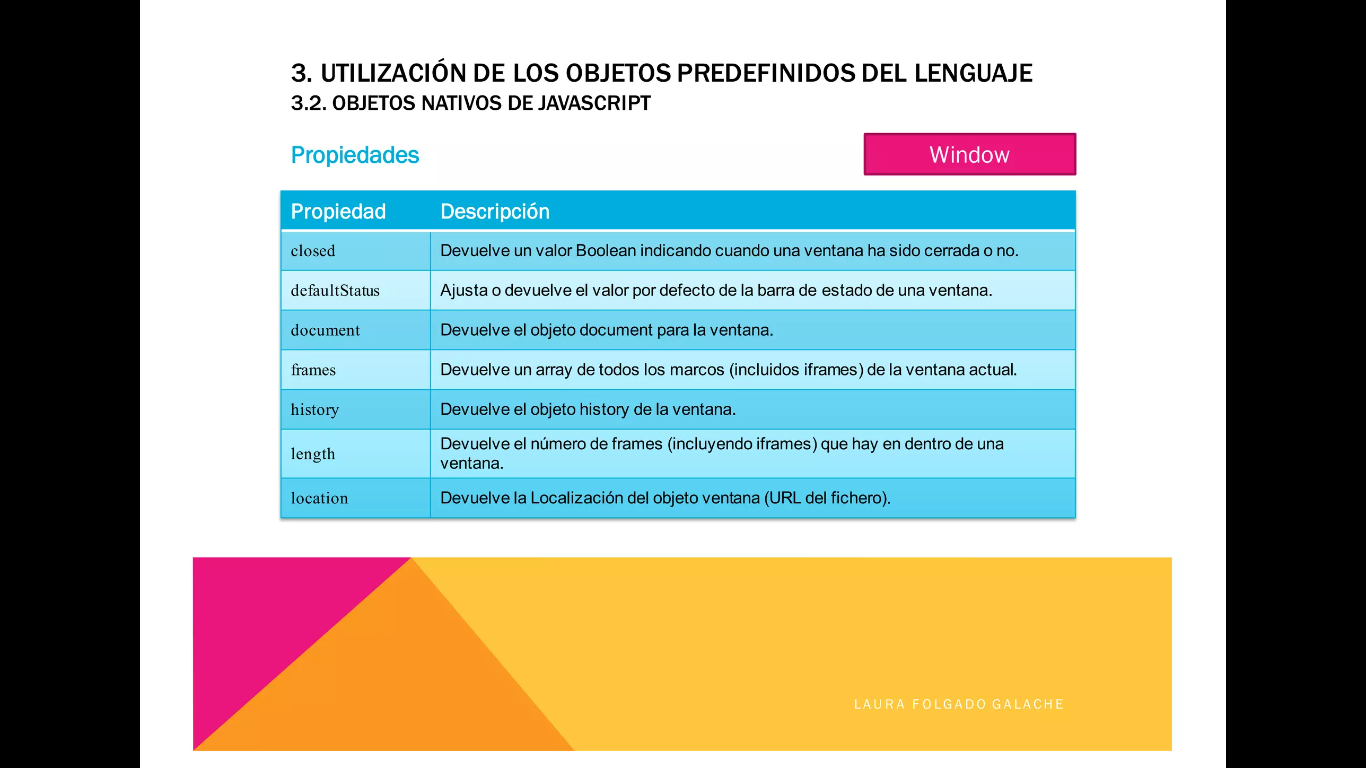
Desde una ventana se pueden abrir o cerrar nuevas ventanas. La primera se denomina ventana principal, mientras que las segundas se denominan ventanas secundarias. Desde la ventana principal se puede acceder a las ventanas secundarias.

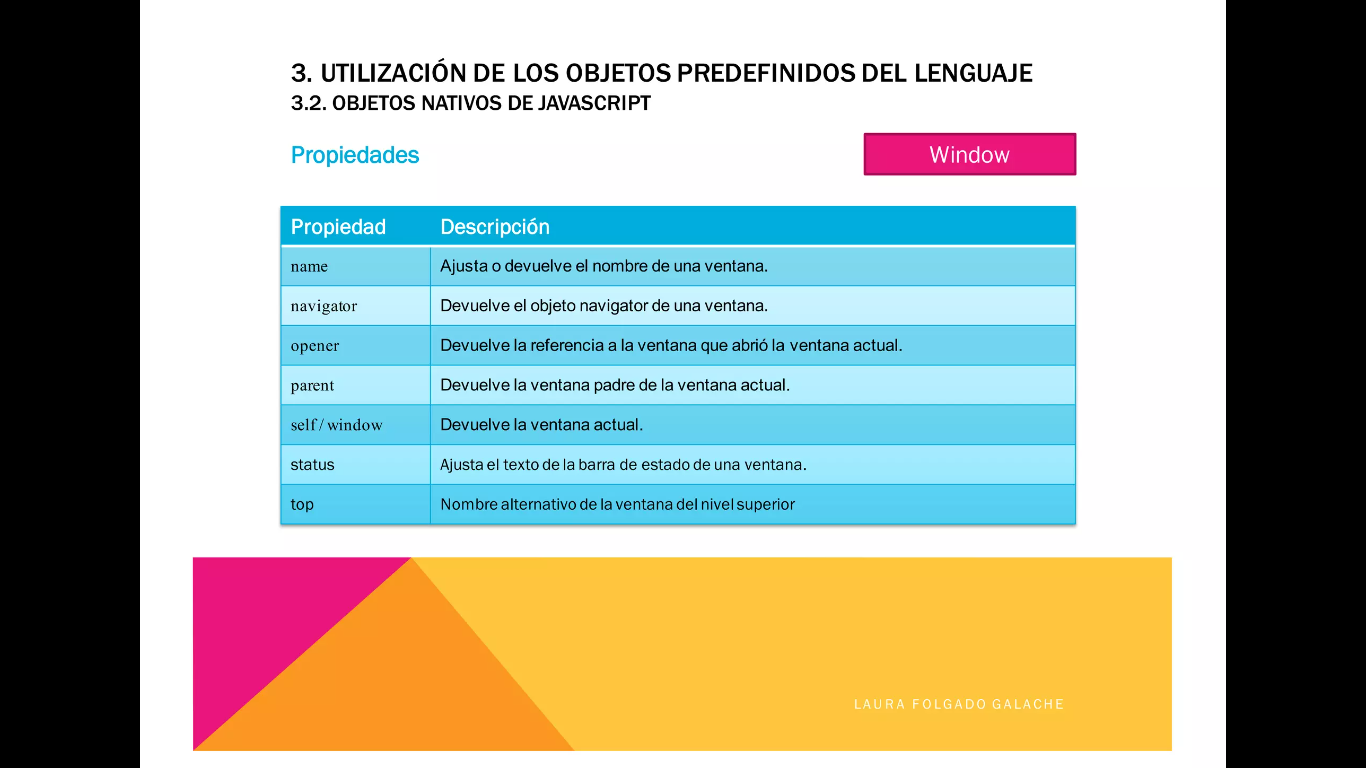
En el siguiente ejemplo se muestra cómo acceder a una ventana secundaria:

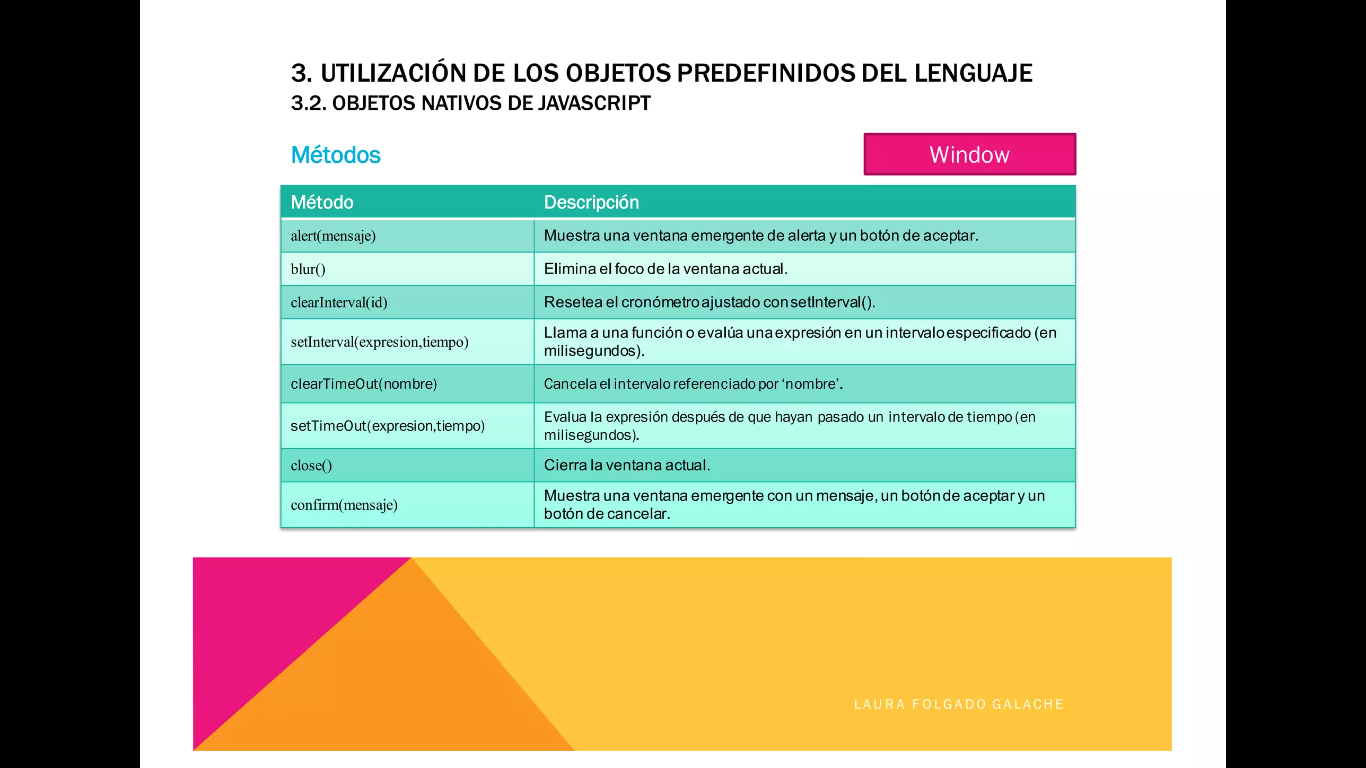


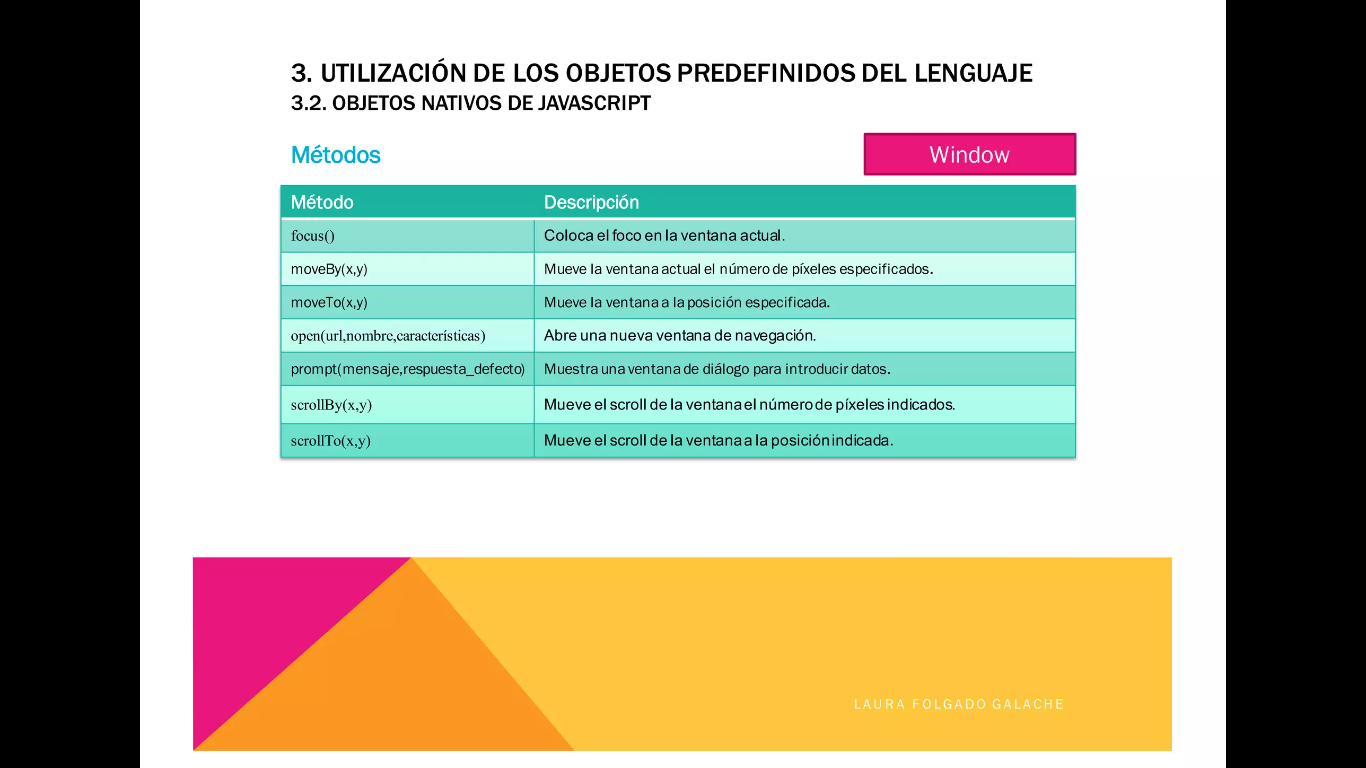
### Propiedades y métodos. Ejemplos.













*Ejemplos*

<!DOCTYPE html>

<html>

<head>

<title>Ejemplo de JavaScript</title>

<meta charset="UTF-8">

</head>

<body>

<p>Este programa permite analizar la llamada a distintas responsabilidades del objeto window.</p>

<form>

<br>

<input type="button" value="open()" onClick="abrir()">

<br>

<input type="button" value="open con parámetros" onClick="abrirParametros()">

<br>

<input type="button" value="alert" onClick="mostrarAlerta()">

<br>

<input type="button" value="confirm" onClick="confirmar()">

<br>

<input type="button" value="prompt" onClick="cargarCadena()">

</form>

<script>

function abrir() {

let ventana = open();

ventana.document.write("Estoy escribiendo en la nueva ventana<br>");

ventana.document.write("Segunda linea");

}

function abrirParametros() {

let ventana = open('', '', 'status=yes,width=400,height=250,menubar=yes');

ventana.document.write("Esto es lo primero que aparece<br>");

}

function mostrarAlerta() {

alert("Esta ventana de alerta ya la utilizamos en otros problemas.");

}

function confirmar() {

let respuesta = confirm("Presione alguno de los dos botones");

if (respuesta == true)

alert("presionó aceptar");

else

alert("presionó cancelar");

}

function cargarCadena() {

let cad = prompt("cargue una cadena:");

alert("Usted ingreso " + cad);

}

</script>

</body>

</html>

*Particularidades*

Cuando uno define una variable fuera de cualquier función con la palabra clave 'var', luego dicha variable se transforma en un atributo del objeto window, esto no sucede cuando utilizamos la palabra clave 'let':

<script>

var v1 = 10;

let v2 = 20;

document.write(window.v1); //muestra un 10

document.write("<br>")

document.write(window.v2); //muestra undefined

</script>

La ejecución de este script tiene como resultado:

10

undefined

Es otra de las diferencias importantes que tienen las variables definidas con var y let.

*Location*

Cuando le asignamos una nueva dirección a la propiedad location del objeto window, el navegador redirecciona a dicha página.

Primero muestra ventana de confirmación con mensajes.

<!DOCTYPE html>

<html>

<head>

<title>Ejemplo de JavaScript</title>

<meta charset="UTF-8">

</head>

<body>

<button onClick="verificarEntrada()">Visitar 20 minutos?</button>

<script>

function verificarEntrada() {

if (window.confirm('Desea visitar la web 20 minutos?')) {

window.location = 'https://www.20minutos.es';

} else {

window.alert('Hasta la próxima');

}

}

</script>

</body>

</html>

El objeto history almacena todas las páginas que visitamos. Luego, con una serie de funciones, podemos extraer de la memoria de la computadora las páginas ya visitadas, sin tener que pedirlas nuevamente al servidor.

Cuenta con las siguientes funciones:

window.history.back(); //Retrocede a la página anterior

window.history.forward(); //Avanza a la página siguiente almacenada en la

cache de la máquina.

window.history.go(); //Avanza o retrocede en la lista de páginas visitadas.

Llamar a la función back, tiene el mismo comportamiento que presionar el botón "Atrás" del navegador.

El siguiente ejemplo nos permite cargar una segunda página y luego retroceder a la primera página sin tener que solicitarla nuevamente al servidor:

<!DOCTYPE html>

<html>

<head>

<title>Ejemplo de JavaScript</title>

<meta charset="UTF-8">

</head>

<body>

</head>

<body>

<a href="pagina2.html">Ir a la página 2</a>

<br>

<br>

<a href="javascript:avanzar()">Extraer del cache la segunda página</a>

<script>

function avanzar() {

window.history.go(1);

}

</script>

</body>

</html>

En la segunda página, mediante el método go y pasándole un valor negativo, retrocedemos a la primera página sin tener la necesidad de recargarla.

Podemos mejorar el ejemplo accediendo al atributo length (almacena la cantidad de páginas de la lista) del objeto history:

if (history.length>0){

history.go(1); }

else {

alert('no hay otra página en la cache hacia adelante');

}

El siguiente programa muestra el contenido de las cinco propiedades del objeto screen:

<!DOCTYPE html>

<html>

<head>

<title>Ejemplo de JavaScript</title>

<meta charset="UTF-8">

</head>

<body>

<script>

document.write('Valores de las propiedades del objeto screen:<br>');

document.write('availHeight :' + screen.availHeight + '<br>');

document.write('availWidth :' + screen.availWidth + '<br>');

document.write('height :' + screen.height + '<br>');

document.write('width :' + screen.width + '<br>');

document.write('colorDepth :' + screen.colorDepth);

</script>

</body>

</html>

Los valores de las propiedades comunes de los navegadores, se ven en este ejemplo:

<!DOCTYPE html>

<html>

<head>

<title>Ejemplo de JavaScript</title>

<meta charset="UTF-8">

</head>

<body>

<script>

document.write('Valores de las propiedades del objeto navigator:<br>');

document.write('appName :' + navigator.appName + '<br>');

document.write('appVersion :' + navigator.appVersion + '<br>');

document.write('cookieEnabled :' + navigator.cookieEnabled + '<br>');

document.write('plugins :' + navigator.plugins.length + '<br>');

</script>

</body>

</html>

El lenguaje JavaScript permite agrupar funciones y disponerlas en un archivo separado a la página HTML, lo cual aporta beneficios, como:

* Reutilización de funciones de uso común. No tenemos que copiar y pegar sucesivamente las funciones en las páginas en las que necesitamos.
* Facilita el mantenimiento de las funciones al encontrarse en archivos separados.
* Nos obliga a ser más ordenados.

La forma de implementar estos archivos externos en JavaScript es:

1 - Crear un archivo con extensión ejemplofunciones.js y teclear las funciones en la misma.

2.- Codificamos un archivo html que utilizará las funciones contenidas en el archivo ejemplofunciones.js

3.- Con la etiqueta script y la propiedad src importamos el archivo ejemplofunciones.js:

<script src="ejemplofunciones.js"></script>