Tema 3. Modelo de objetos predefinidos en JavaScript.

# 1.- Objetos de más alto nivel en JavaScript.

El Modelo de Objetos del Documento (DOM), permite ver el mismo documento de otra manera, describiendo el contenido del documento como un conjunto de objetos, sobre los que un programa de JavaScript puede interactuar.

Según el W3C, el Modelo de Objetos del Documento es una interfaz de programación de aplicaciones (API), para documentos válidos HTML y bien construidos XML. Define la estructura lógica de los documentos, y el modo en el que se acceden y se manipulan.

Un objeto es una entidad con una serie de **propiedades** que definen su estado, y unos **métodos** (funciones), que actúan sobre esas propiedades.

**La forma de acceder a una propiedad de un objeto es la siguiente**:

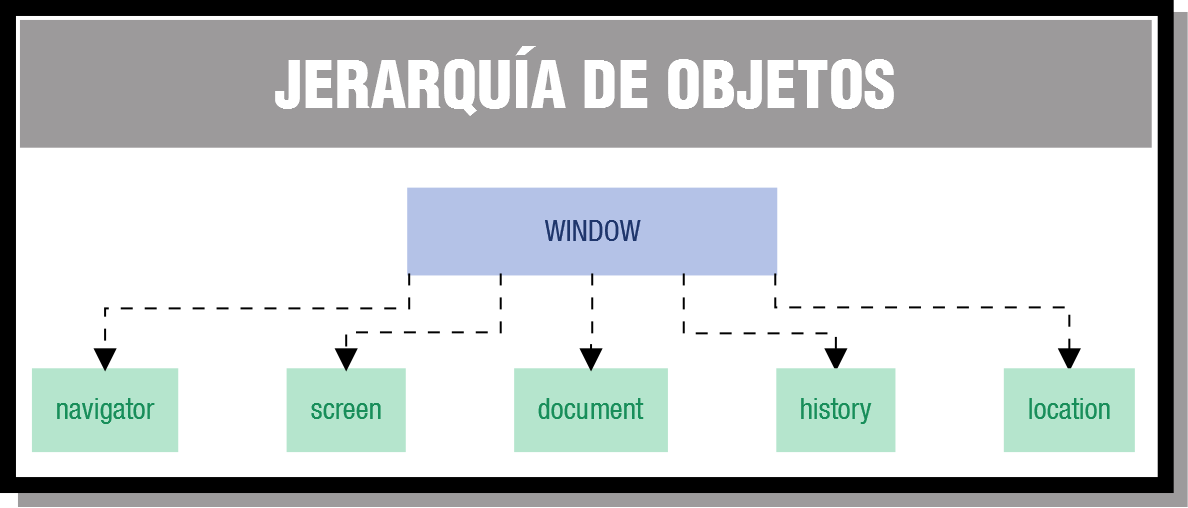
nombreobjeto.propiedad

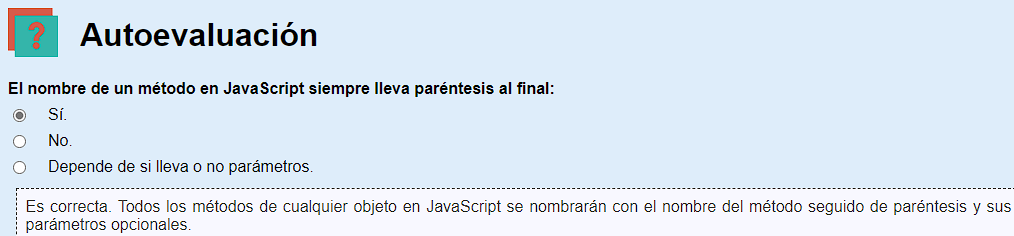
**La forma de acceder a un método de un objeto es la siguiente**:

nombreobjeto.metodo([parámetros opcionales])

**También podemos referenciar a una propiedad de un objeto, por su índice en la creación**. Los índices comienzan por 0.

Nos enfocaremos en objetos de alto nivel: **window, location, navigator y document**. El objetivo es prepararte para profundizar en las propiedades y métodos, gestores de eventos, etc. Que encontrarás en unidades posteriores.





## 1.1.- Objeto window.

Este objeto es el contenedor principal de todo el contenido que se visualiza en el navegador. Tan pronto como se abre una ventana (window) en el navegador, incluso aunque no se cargue ningún documento en ella, este objeto window ya estará definido en memoria.

El **campo de influencia de este objeto**, abarca también las dimensiones de la ventana, así como todo lo que rodea al área de contenido: las barras de desplazamiento, barra de herramientas, barra de estado, etc.

Debajo del objeto **window**, tenemos otros objetos como **navigator, screen, history y location**, y el objeto **document**. Este objeto **document** será el que contendrá toda la jerarquía de objetos, que tengamos dentro de nuestra página HTML.

**Atención**: Para JavaScript tanto las ventanas de navegador, como las pestañas, son ambos objetos window.

Acceso a propiedades y a métodos

Para acceder a las propiedades y métodos del objeto window, **lo podemos hacer de diferentes formas**:

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

**Un objeto window también se podrá referenciar mediante la palabra self**, cuando estamos haciendo la referencia desde el propio documento contenido en esa ventana.

self.nombrePropiedad

self.nombreMetodo( [parametros] )

Debido a que el objeto **window** siempre estará presente cuando ejecutemos nuestros scripts, podremos omitirlo, en referencias a los objetos dentro de esa ventana. Es decir, si escribimos:

nombrePropiedad

nombreMetodo( [parametros] )

**También funcionaría sin ningún problema, porque se asume que esas propiedades o métodos, son del objeto de mayor jerarquía** (el objeto window) en el cual nos encontramos.

### 1.1.1.- Gestión de ventanas.

Un **script** que esté ejecutándose en una de las ventanas principales del navegador, **podrá crear o abrir nuevas sub-ventanas**.

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

**Por ejemplo**: **var** subVentana=window.open("nueva.html","nueva","height=800,width=600");

**De esta forma podremos** a lo largo de nuestro código, **referenciar a la nueva ventana desde el script original de la ventana principal**. Si quisiéramos cerrar la nueva ventana desde nuestro script, simplemente tendríamos que hacer:

subVentana.close()

**Ejemplo**: [ver aquí](https://aula21-my.sharepoint.com/personal/5634224_alu365_murciaeduca_es/Documents/Documentos/FP/DAW/Curso%202/Desarrollo%20web%20en%20entorno%20cliente/Evaluacion%201/Tema%203/Recursos/DWEC03_CONT_R02_Ejemplo1/DWEC03_CONT_R02_Ejemplo1.html).

### 1.1.2.- Propiedades y métodos.

El objeto **window** representa una ventana abierta en un navegador. El navegador crea un objeto **window** para el documento HTML, y un objeto **window** adicional para para cada marco.

Propiedades del objeto Window

|  |  |
| --- | --- |
| **Propiedad** | **Descripción** |
| **closed** | Devuelve un valor Boolean indicando cuando una ventana ha sido cerrada o no. |
| **defaultStatus** | Ajusta o devuelve el valor por defecto de la barra de estado de una ventana. |
| **document** | Devuelve el objeto **document** para la ventana. |
| **frames** | Devuelve un array de todos los marcos (incluidos iframes) de la ventana actual. |
| **history** | Devuelve el objeto **history** de la ventana. |
| **length** | Devuelve el número de frames (incluyendo 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. |
| **opener** | Devuelve la referencia a la ventana que abrió la ventana actual. |
| **parent** | Devuelve la ventana padre de la ventana actual. |
| **self** | Devuelve la ventana actual. |
| **status** | Ajusta el texto de la barra de estado de una ventana. |

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()** | Resetea el cronómetro ajustado con **setInterval()**. |
| **setInterval()** | Llama a una función o evalúa una expresión en un intervalo especificado (en milisegundos). |
| **close()** | Cierra la ventana actual. |
| **confirm()** | Muestra una ventana emergente con un mensaje, un botón de aceptar y un botón de cancelar. |
| **focus()** | Coloca el foco en la ventana actual. |
| **open()** | Abre una nueva ventana de navegación. |
| **prompt()** | Muestra una ventana de diálogo para introducir datos. |

## 1.2.- Objetos location.

El objeto **window** contiene información referente a la URL actual. Es **parte del objeto window**.

Propiedades del objeto Location

|  |  |
| --- | --- |
| **Propiedad** | **Descripción** |
| **hash** | Cadena que contiene el nombre del enlace, dentro de la URL. |
| **host** | Cadena que contiene el nombre del servidor y el número del puerto, dentro de la URL. |
| **hostname** | Cadena que contiene el nombre de dominio del servidor (o la dirección IP), dentro de la URL. |
| **href** | Cadena que contiene la URL completa. |
| **pathname** | Cadena que contiene el camino al recurso, dentro de la URL. |
| **port** | Cadena que contiene el número de puerto del servidor, dentro de la URL. |
| **protocol** | Cadena que contiene el protocolo utilizado (incluyendo los dos puntos), dentro de la URL. |
| **search** | Cadena que contiene la información pasada en una llamada a un script, dentro de la URL. |

Métodos del objeto Location

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

## 1.3.- Objeto navigator.

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

Propiedades del objeto Navigator

|  |  |
| --- | --- |
| **Propiedad** | **Descripción** |
| **appCodeName** | Cadena que contiene el nombre en código del navegador. |
| **appName** | Cadena que contiene el nombre del cliente. |
| **appVersion** | Cadena que contiene información sobre la versión del cliente. |
| **cookieEnabled** | Determina si las cookies están o no habilitadas en el navegador. |
| **platform** | Cadena con la plataforma sobre la que se está ejecutando el programa cliente. |
| **userAgent** | Cadena que contiene la cabecera completa del agente enviada en una petición  HTTP. Contiene la información de laspropiedades **appCodeName** y **appVersion**. |

Métodos del objeto Navigator

|  |  |
| --- | --- |
| **Método** | **Descripción** |
| **javaEnabled()** | Devuelve true si el cliente permite la utilización de Java, en caso contrario, devuelve false |

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

## 1.4.- Objeto 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**.

Colecciones del objeto Document

|  |  |
| --- | --- |
| **Colección** | **Descripción** |
| **anchors[]** | Es un array que contiene todos los hiperenlaces 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. |

Propiedades del objeto Document

|  |  |
| --- | --- |
| **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. |
| **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. |

Propiedades del objeto Document

|  |  |
| --- | --- |
| **Método** | **Descripción** |
| **close()** | Cierra el flujo abierto previamente con **document.open()**. |
| **getElementById()** | Para acceder a un elemento identificado por el id escrito entre paréntesis. |
| **getElementsByName()** | Para acceder a los elementos identificados por el atributo **name** escrito entre paréntesis. |
| **getElementsByTagName()** | Para acceder a los elementos identificados por la etiqueta escrita entre paréntesis. |
| **open()** | Abre el flujo de escritura para poder utilizar **document.write()** o **document.writeln** en el documento. |
| **write()** | Para poder escribir expresiones HTML o código de JavaScript dentro de un documento. |
| **writeln()** | Lo mismo que **write()** pero añade un salto de línea al final de cada instrucción. |

# 2.- Marcos.

Un objeto **frame** representa un marco HTML. La etiqueta **<frame>** identifica una ventana particular, dentro de un conjunto de marcos (frameset). Para cada **<frame>**, se creará un objeto **frame**. Todo lo anterior, se aplicará también al objeto **Iframe <iframe>**.

Propiedades del objeto Frame/iframe

|  |  |
| --- | --- |
| **Propiedad** | **Descripción** |
| **align** | Cadena que contiene el valor del atributo **align** (alineación) en un **iframe**. |
| **contentDocument** | Devuelve el objeto documento contenido en un **frame**/**iframe**. |
| **contentWindow** | Devuelve el objeto **window** generado por un **frame**/**iframe**. |
| **frameBorder** | Cadena que contiene el valor del atributo **frameborder** (borde del marco) de un **frame**/**iframe**. |
| **height** | Cadena que contiene el valor del atributo **height** (altura) de un **iframe**. |
| **longDesc** | Cadena que contiene el valor del atributo **longdesc** (descripción larga) de un **frame**/**iframe**. |
| **marginHeight** | Cadena que contiene el valor del atributo **marginheight** (alto del margen) de un **frame**/**iframe**. |
| **marginWidth** | Cadena que contiene el valor del atributo **marginwidth** (ancho del margen) de un **frame**/**iframe**. |
| **name** | Cadena que contiene el valor del atributo **name** (nombre) de un **frame**/**iframe**. |
| **noResize** | Cadena que contiene el valor del atributo **noresize** de un **frame**/**iframe**. |
| **scrolling** | Cadena que contiene el valor del atributo **scrolling** (desplazamiento) de un **frame**/**iframe**. |
| **src** | Cadena que contiene el valor del atributo **src** (origen) de un **frame**/**iframe**. |
| **width** | Cadena que contiene el valor del atributo **width** (ancho) de un **iframe**. |

Eventos del objeto frame/iframe

* **onLoad**: Script que se ejecutará inmediatamente después a que se cargue el frame/iframe.

## 2.1.- Jerarquías.

Marcos: Padres e hijos

El objeto **window** está en la cabeza de la jerarquía y puede tener sinónimos como **self**. Cuando trabajamos con marcos o iframes, podemos **referenciar** a las **ventanas** como **frame, top y parent**.

**El uso de marcos o iframes no se recomienda**, ya que su uso está en verdadero declive (de hecho actualmente ya no se usan, salvo incrustar para los vídeos de YouTube en páginas, ya que siguen usando iframes).

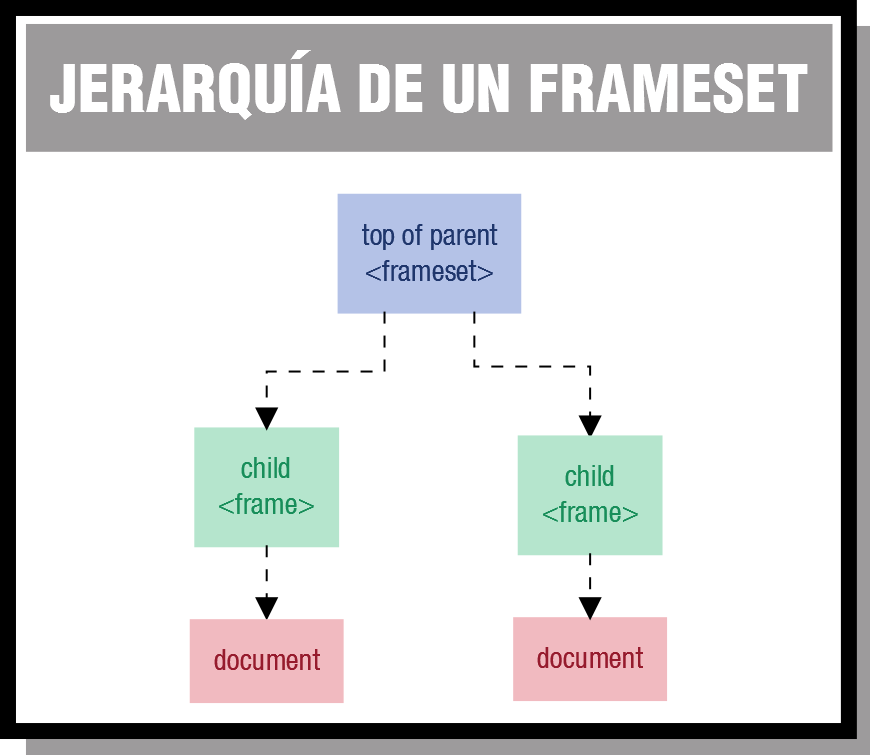
**Inconvenientes de los marcos**:

* Las páginas contenidas en estos marcos no son directamente accesibles, ya que la URL principal de nuestro navegador no cambia, con lo que no tenemos una referencia directa de la página en la que nos encontramos.
* Los buscadores como Google, Bing, etc, no indexan bien los frames.

Texto

Descripción generada automáticamente**Ejemplo de frame**:

Este código divide la ventana del navegador en dos marcos de igual tamaño, con dos documentos diferentes en cada columna. Un frameset establece las relaciones entre los marcos de la colección.



En este gráfico se puede observar que la ventana padre (la que contiene el frameset), **no tiene ningún objeto document**, ya que son **los frames hijos**, los que **sí** **tienen** **objeto document**.

## 2.2.- Comunicación entre marcos.

Referencias Padres a Hijos

* **[window.]frames[n]**.objeto-función-variable-nombre
* **[window.]frames["nombreDelMarco"].**objeto-función-variable-nombre
* **[window.]nombreDelMarco**.objeto-función-variable-nombre

Referencias Hijos a Padre

**parent.**objeto-función-variable-nombre

**Ejemplo**: var valor= parent .NombreFuncion();

Esto es **equivalente** a: var valor=top.NombreFuncion();

Referencias Hijos a Hijos

El único modo de hacer referencias de un hijo a otro, es a través de su padre, que es común a todos, y por tanto, actúa como nexo entre ellos. **Sintaxis**:

* **parent.frames[n].**objeto-función-variable-nombre
* **parent.frames["nombreDelMarco"].**objeto-función-variable-nombre
* **parent.nombreDelMarco.**objeto-función-variable-nombre

## 2.3.- Comunicación entre múltiples ventanas.

Podemos comunicarnos con subventanas:

* Podemos abrirlas con el **método open()** del objeto window.
* **Cada objeto window** tiene una **propiedad** llamada **opener**, que hace referencia a la ventana o marco que la ha abierto.
  + La ventana principal tendrá valor de **null**, ya que ninguna otra la ha abierto.

Texto

Descripción generada automáticamente con confianza media

Texto

Descripción generada automáticamente con confianza baja

# 3.- Objetos nativos en JavaScript.

JavaScript dispone de unos objetos nativos. **Estos son como las clases wrapper en Java**: String, Math, Number, Boolean y Date. Nos serán muy útiles para:

* las conversiones de datos
* trabajar intensivamente con cadenas
* y con fechas y horas

## 3.1.- Objeto String.

Podemos usar comillas simples o dobles para encerrar una cadena.

* **Usamos las comillas simples** cuando queremos incluir comillas dobles en su interior.
  + Si queremos usar comillas simples en su interior, tendremos que hacer uso de la **secuencia de escape \’**.
* **Usamos las comillas dobles** cuando queremos incluir comillas simples en su interior.
  + Si queremos usar comillas dobles en su interior, tendremos que hacer uso de la **secuencia de escape \“**.



* **Podemos concatenar cadenas con el operador**: operador+=.
* **Concatenar contenido de una variable dentro de una cadena**: operador +.

Caracteres especiales o caracteres de escape

|  |  |
| --- | --- |
| **Símbolos** | **Explicación** |
| **\"** | Comillas dobles. |
| **\'** | Comilla simple. |
| **\\** | Barra inclinada. |
| **\b** | Retroceso. |
| **\t** | Tabulador. |
| **\n** | Nueva línea. |
| **\r** | Salto (retorno de carro) de línea. |
| **\f** | Avance (forward) de línea. |

### 3.1.1.- Propiedades y métodos del objeto String.

* Formas de crear un String:
  + var miCadena = new String ("texto de la cadena");
  + var miCadena = "texto de la cadena";

Es decir, cada vez que tengamos una cadena de texto, **en realidad es un objeto String** que tiene propiedades y métodos.

Propiedades de String

* length: devuelve la longitud de la cadena.

Métodos del objeto String

|  |  |
| --- | --- |
| **Métodos** | **Descripción** |
| **charAt()** | Devuelve el carácter especificado por la posición que se indica entre paréntesis. |
| **charCodeAt()** | **Devuelve *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**.  Esta función es especial, ya que solo se puede llamar a ella desde la clase, y no desde el objeto. String.fromCharCode() |
| **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/elimina una parte de la cadena y devuelve una nueva cadena (es decir, la cadena resultante). |
| **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. |

Ejemplo de uso:

Texto, Carta

Descripción generada automáticamente

    cad = 'Cifp Carlos III';

    console.log(cad.charAt(1));//->i

    console.log(cad.toUpperCase());//->CIFP CARLOS III

    console.log(cad.toLowerCase());//->cifp carlos iii

    console.log(cad.indexOf('III'));//->12

    console.log(cad.includes('III'));//->true

    console.log(cad.substring(2,4));//->fp

## 3.2.- Objeto Math.

Ya vimos algunas funciones relacionadas con la **conversión de cadenas a diferentes formatos numéricos** (parseInt, parseFloat…).

A parte de eso, disponemos de un **objeto Math que cuenta con propiedades y métodos útiles**.

**var** x = Math.PI; // Devuelve el número PI.

**var** y = Math.sqrt(16); // Devuelve la raíz cuadrada de 16.

Propiedades del objeto Math

|  |  |
| --- | --- |
| **Propiedad** | **Descripción** |
| **E** | Devuelve el **número Euler** (aproximadamente 2.718). |
| **LN2** | Devuelve el logaritmo **neperiano** **de 2** ( aproximadamente 0.693). |
| **LN10** | Devuelve el logaritmo **neperiano** **de 10** ( aproximadamente 2.302). |
| **LOG2E** | Devuelve el **logaritmo base 2 de E** ( aproximadamente 1.442). |
| **LOG10E** | Devuelve el **logaritmo base 10 de E** ( aproximadamente 0.434). |
| **PI** | Devuelve el **número PI** ( aproximadamente 3.14159). |
| **SQRT2** | Devuelve la **raíz cuadrada de 2** ( aproximadamente 1.414). |

Métodos del objeto Math

|  |  |
| --- | --- |
| **Método** | **Descripción** |
| **abs(x)** | Devuelve el valor absoluto de x. |
| **acos(x)** | Devuelve el arcocoseno de x, en radianes. |
| **asin(x)** | Devuelve el arcoseno de x, en radianes. |
| **atan(x)** | Devuelve el arcotangente de x, en radianes con un valor entre -PI/2 y PI/2. |
| **atan2(y,x)** | Devuelve el arcotangente del cociente de sus argumentos. |
| **ceil(x)** | Devuelve el número x redondeado al alta hacia el siguiente entero. |
| **cos(x)** | Devuelve el coseno de x (x está en radianes). |
| **floor(x)** | Devuelve el número x redondeado a la baja hacia el anterior entero. |
| **log(x)** | Devuelve el logaritmo neperiando (base E) de x. |
| **max(x,y,z,...,n)** | Devuelve el número más alto de los que se pasan como parámetros. |
| **min(x,y,z,...,n)** | Devuelve el número más bajo de los que se pasan como parámetros. |
| **pow(x,y)** | Devuelve el resultado de x elevado a y. |
| **random()** | Devuelve un número al azar entre 0 y 1. |
| **round(x)** | Redondea x al entero más próximo. |
| **sin(x)** | Devuelve el seno de x (x está en radianes). |
| **sqrt(x)** | Devuelve la raíz cuadrada de x. |
| **tan(x)** | Devuelve la tangente de un ángulo. |

Ejemplos de uso:

Texto, Carta

Descripción generada automáticamente

## 3.3.- Objeto Number.

Este objeto se usa **muy raramente**. Para la mayor parte de los casos, JavaScript satisface las necesidades del día a día con valores numéricos almacenados en variables.

Sin embargo, el objeto Number posee información y capacidades muy interesantes.

* El objeto **Number** es un objeto envoltorio para valores numéricos primitivos.
* Son creados con el constructor **new Number()**.

Propiedades del objeto Number

|  |  |
| --- | --- |
| **Propiedad** | **Descripción** |
| **constructor** | Devuelve la función que creó el objeto **Number**. |
| **MAX\_VALUE** | Devuelve el número más alto disponible en JavaScript.  Este es 1.79E+308. Cualquier nº mayor que este nº será considerado como infinito positivo. |
| **MIN\_VALUE** | Devuelve el número más pequeño disponible en JavaScript.  Este es 2.22E-308. Cualquier nº menor que este nº será considerado como infinito negativo. |
| **NEGATIVE\_INFINITY** | Representa a infinito negativo (se devuelve en caso de overflow). |
| **POSITIVE\_INFINITY** | Representa a infinito positivo (se devuelve en caso de overflow). |
| **prototype** | Permite añadir nuestras propias propiedades y métodos a un objeto. |

Los números y sus valores están definidos internamente en JavaScript, como valores de doble precisión y de 64 bits.

Métodos del objeto Number

|  |  |
| --- | --- |
| **Método** | **Descripción** |
| **toExponential(x)** | Convierte un número a su notación exponencial. |
| **toFixed(x)** | Formatea un número con x digitos decimales después del punto decimal. |
| **toPrecision(x)** | Formatea un número a la longitud x. |
| **toString()** | Convierte un objeto **Number** en una cadena.   * Si se pone 2 como parámetro se mostrará el número en binario. * Si se pone 8 como parámetro se mostrará el número en octal. * Si se pone 16 como parámetro se mostrará el número en hexadecimal. |
| **valueOf()** | Devuelve el valor primitivo de un objeto **Number**. |

Ejemplos de uso

Texto

Descripción generada automáticamente con confianza baja

//Métodos

let numdecimal = 3.1416;

console.log(numdecimal.toExponential(2)) //3.14e+0

console.log(numdecimal.toFixed(3)) //3.142

console.log(numdecimal.toString()) //3.1416 (string)

/\*

   Todos los números se guardan en 64 bits, aunque no se utiliza mucho

   en versiones actuales de js existe un nuevo tipo bigint que permite

   almacenar números más grnades, añadiendole una n al final del numero

\*/

let bignum = 1111n;

console.log(typeof bignum); //bignint

//Admite notación Hexadecimal (0x), binaria(0b) y octal(0o)

let numexadecimal = 0xFF;

console.log(numexadecimal) //255

## 3.4.- Objeto Boolean.

Es equivalente a un tipo booleano primitivo. Pero dispone de los siguientes métodos y propiedades:

Propiedades del objeto Boolean

|  |  |
| --- | --- |
| **constructor** | Devuelve la función que creó el objeto **Boolean**. |
| **prototype** | Te permitirá añadir propiedades y métodos a un objeto. |

Métodos del objeto Boolean

|  |  |
| --- | --- |
| **toString()** | Convierte un valor Boolean a una cadena y devuelve el resultado. |
| **valueOf()** | Devuelve el valor primitivo de un objeto Boolean. |

Ejemplos de uso:

Texto

Descripción generada automáticamente

**Para saber más**: [JavaScript Boolean Reference (w3schools.com)](https://www.w3schools.com/jsref/jsref_obj_boolean.asp)

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

## 3.5.- Objeto Date.

El objeto **Date** se utiliza para trabajar con fechas y horas. Los objetos **Date** se crean con **New Date()**. Sin embargo, **hay 4 formas de instanciar** (crear) un objeto de tipo **Date**:

* **var** d = **new** Date();
* **var** d = **new** Date(milisegundos);
* **var** d = **new** Date(cadena de Fecha);
* **var** d = **new** Date(año, mes, [día, horas, minutos, segundos, milisegundos]);
  + // (el mes comienza en 0, Enero sería 0, Febrero 1, etc.)

Propiedades del objeto Date

|  |  |
| --- | --- |
| **Propiedad** | **Descripción** |
| **constructor** | Devuelve la función que creó el objeto **Date**. |
| **prototype** | Te permitirá añadir propiedades y métodos a un objeto |

Métodos del objeto Date

|  |  |  |
| --- | --- | --- |
| **Resumen** | **Método** | **Descripción** |
| **Día** | **getDate()** | Devuelve el **día** del mes (de 1-31). |
| **getDay()** | Devuelve el **día** de la semana (de 0-6). |
| **getUTCDay()** | Devuelve el día de la semana en base a la hora UTC (de 0-6). |
| **getUTCDate()** | Devuelve el día del mes en base a la hora  UTC (de 1-31). |
| **Mes** | **getMonth()** | Devuelve el mes (de 0-11). |
| **Año** | **getUTCFullYear()** | Devuelve el año en base a la hora UTC (4 dígitos). |
| **getFullYear()** | Devuelve el año (4 dígitos). |
| **Tiempo** | **getHours()** | Devuelve la hora (de 0-23). |
| **getMinutes()** | Devuelve los minutos (de 0-59). |
| **getSeconds()** | Devuelve los segundos (de 0-59). |
| **getMilliseconds()** | Devuelve los milisegundos (de 0-999). |
| **getTime()** | Devuelve los milisegundos desde media noche del 1 de Enero de 1970. |
| **getTimezoneOffset()** | Devuelve la diferencia de tiempo entre  GMT y la hora local, en minutos. |
| **Setters** | **setDate()** | Ajusta el día del mes del objeto (de 1-31). |
| **setFullYear()** | Ajusta el año del objeto (4 dígitos). |
| **setHours()** | Ajusta la hora del objeto (de 0-23). |

Algunos ejemplos de uso:

var d = new Date();

document.write(d.getFullYear());

document.write(d.getMonth());

document.write(d.getUTCDay());

var d2 = new Date(5,28,2011,22,58,00);

d2.setMonth(0);

d.setFullYear(2020);

## 3.6.- Expresiones regulares.

Son patrones que describen como tiene que ser una cadena. Existen dos formas de definirlos:

* let patron = new RegExp('expresión\_regular','modificadores') //mod. opcionales (igm)
* let patron = /expresión\_regular/modificadores

El objeto RegExp dispone de un **método test** que comprueba si una cadena cumple el patrón. Lo definamos como lo definamos, tenemos acceso a este método.

Modificadores

* **g**: devuelve todas las coincidencias del patrón.

Ejemplos:

let patron1 = /hola/;

let patron2 = /a/g; //modificador g: Devuelve todas las coincidencias del patrón

let cadena = 'amapola';

console.log(patron1.test(cadena)); //false

console.log(patron2.test(cadena)); //true

Manejo de expresiones regulares de String

El objeto String también tiene **métodos** relacionados con las expresiones regulares: match y replace

console.log( cadena.match(patron1) ); //null

console.log( cadena.match(patron2) ); // ['a','a','a']

console.log(cadena.replace(/a/g,'X')); //XmXpolX

console.log(cadena) //amapola (no modifica la cadena)

### 3.6.1.- Expresiones regulares (II). Clases de caracteres.

Tenemos una serie de clases de caracteres:

|  |  |
| --- | --- |
| **Clase** | **Descripción** |
| \d | Dígito (número). |
| \s | Espacio. |
| \w | Carácter (dígito, letra, guión bajo) |
| . | Cualquier caracter |
| Clase en mayúscula | Si ponemos la letra de la clase en mayúscula, es la clase inversa. |

Ejemplo de uso

let cadena = 'Desarrollo de aplicaciones web 2020';

let patron = /\d/g;

    console.log(patron.test(cadena)) //true

    console.log(cadena.match(patron)) //['2','0','2','0']

Más modificadores

* **Podemos combinar caracteres normales con clases de caracteres**. Ejemplo:

patron = /de\s/;

* **Inicio ^ y fin de cadena $**:

    patron = /^D/;

    console.log(patron.test(cadena)); //true

    patron = /0$/;

    console.log(patron.test(cadena)); //true

    patron = /^D0$/;

    console.log(patron.test(cadena)); //false ¿Reflexiona por que es falso

* **Conjuntos […] y rangos**: cualquier carácter de los que se indica. Los rangos se refiere a cuando queremos referirnos a una serie de números. Por ejemplo:

patron = /[DXP]/

    console.log(patron.test(cadena)); //true : Suficiente con que coincida D

    patron = /[0-5]/

    console.log(patron.test(cadena)); //true : El guión nos permite declarar rangos

    patron = /^[0-5]/

    console.log(patron.test(cadena)); //false : ^ No Comienza con número.

    patron = /[^0-5]/ //false : ^ No Comienza con número.

    console.log(patron.test(cadena));

* **Cuantificadores {n}**: Se aplican al elemento que les precede. n veces (también admite rangos).

    patron = /20{2}/;

    console.log(patron.test(cadena)); //false. Reflexiona por qué

    patron = /(20){2}/;

    console.log(patron.test(cadena)); //true. Los parentesis permiten agrupar

    patron = /(20){3,5}/;

    console.log(patron.test(cadena)); //false. Los rangos se ponen con comas

* + **Cuantificadores abreviados**:
    - +: 1 o más = {1,}
    - \*: 0 o más = {0,}
    - ?: 0 o 1 {0,1}
* **Grupos**: en el anterior ejemplo hemos visto como crear un grupo. Podemos hacer referencia a los grupos capturados con \1, \2…

    cadena = '3R5'

    patron = /(\d)\w\1/;

    console.log(patron.test(cadena)); //false. Reflexiona por que..

* **Búsqueda de caracteres especiales**:
  + Si queremos buscar un punto pondriamos \.

# Tutorías

Texto

Descripción generada automáticamente

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Texto

Descripción generada automáticamente con confianza media

Podemos usar métodos propios de Number aunque no hayamos declarado la variable con new Number:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

//  setInterval, ejecuta una función cada x milisegundos

            function prueba(){

                fecha = new Date()

                document.writeln(fecha.getSeconds()+'<br>');

            }

            setInterval(prueba,1000)

         //   setTimeout, ejecuta una función transcurridos x milisegundos.

            setTimeout(prueba,1000)

Geolocalizacion:

navigator.geolocation.getCurrentPosition(exito)

        function exito (posicion) {

            console.log(posicion);

        }