

Modelo de objetos predefinidos en JavaScript.



Caso práctico

Antonio ha completado correctamente la fase de introducción y fundamentos básicos del lenguaje JavaScript, y ahora comienza a investigar en las características de los objetos predefinidos en JavaScript.



Estos objetos le van a permitir gestionar ventanas, marcos, propiedades de los navegadores, de las URL, etc. en JavaScript.

Además, también va a poder realizar operaciones matemáticas, de fecha y de cadenas, con otros tantos objetos nativos del lenguaje JavaScript.

Antonio tiene una pequeña reunión con **Ada** y con su tutor **Juan**, para comentar los progresos realizados hasta este momento y se pone manos a la obra con esta nueva sección.

En esta unidad de trabajo profundizaremos en los objetos de más alto nivel en JavaScript, los cuales estarán disponibles en todas nuestras aplicaciones. Veremos como gestionar los marcos o frames y la forma de comunicarnos entre múltiples ventanas que tengamos abiertas en el navegador.

Para terminar con la unidad veremos los objetos nativos en JavaScript, aquellos que están definidos en el propio lenguaje y que nos van a permitir realizar operaciones matemáticas complejas, trabajar con cadenas de texto, operaciones de fecha y hora, etc.



[Ministerio de Educación y Formación Profesional](#) (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Objetos de más alto nivel en Javascript.



Caso práctico

Bajo la tutoría de **Juan, Antonio** se dispone a profundizar en los objetos básicos y de más alto nivel de JavaScript. Estos objetos, los encontrará en prácticamente la mayoría de aplicaciones que haga con JavaScript, por lo que será fundamental, que tenga muy claras las características y diferentes funcionalidades que estos objetos le van a aportar a sus aplicaciones.



Una página web, es un documento HTML que será interpretado por los navegadores de forma gráfica, pero que también va a permitir el acceso al código fuente de la misma.

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 contruidos XML. Define la estructura lógica de los documentos, y el modo en el que se acceden y se manipulan.

Ahora que ya has visto en la unidad anterior, los fundamentos de la programación, vamos a profundizar un poco más en lo que se refiere a los objetos, que podremos colocar en la mayoría de nuestros documentos.

Definimos como **objeto**, 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.

En esta unidad, nos enfocaremos en objetos de alto nivel, que encontrarás frecuentemente en tus aplicaciones de JavaScript: `window`, `location`, `navigator` y `document`. El objetivo, no es solamente indicarte las nociones básicas para que puedas comenzar a realizar tareas sencillas, sino también, el prepararte para profundizar en las propiedades y métodos, gestores de eventos, etc. que encontrarás en unidades posteriores.

En esta unidad, verás solamente las propiedades básicas, y los métodos de los objetos mencionados anteriormente.

Te mostramos aquí el gráfico del modelo de objetos de alto nivel, para todos los navegadores que permitan usar JavaScript.



Es muy importante que tengas este gráfico en mente porque va a ser la guía a lo largo de toda esta unidad.

En esta unidad no hablaremos del uso de `frames` ya que su uso está totalmente desaconsejado por motivos de accesibilidad y usabilidad, por lo que te recomendamos no usarlos.



Autoevaluación

El nombre de un método en JavaScript siempre lleva paréntesis al final:

- ☐ Sí.
- ☐ No.
- ☐ Depende de si lleva o no parámetros.

Es correcta. Todos los métodos de cualquier objeto en JavaScript se nombrarán con el nombre del método seguido de paréntesis y sus parámetros opcionales.

Es incorrecta. Si no lleva paréntesis se consideraría una propiedad.

No es correcta. Aunque no lleve parámetros tendría que llevar los paréntesis igualmente.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto

1.1.- Objeto window.

En la jerarquía de objetos, tenemos en la parte superior el objeto `window`.

Este objeto está situado justamente ahí, porque 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.



[Everaldo Coelho \(GNU/GPL\)](#)

Además de la sección de contenido del objeto `window`, que es justamente dónde se cargarán los documentos, 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.

Cómo se ve en el gráfico de la jerarquía de objetos, debajo del objeto `window` tenemos otros objetos como el `navigator`, `screen`, `history`, `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 En los navegadores más modernos, los usuarios tienen la posibilidad de abrir las páginas tanto en nuevas pestañas dentro de un navegador, como en nuevas ventanas de navegador. Para JavaScript tanto las ventanas de navegador, como las pestañas, son ambos objetos `window`.

Acceso a propiedades y métodos.

Para acceder a las propiedades y métodos del objeto `window`, lo podremos hacer de diferentes formas, dependiendo más de nuestro estilo, que de requerimientos sintácticos. Así, la forma más lógica y común de realizar esa referencia, incluiría el objeto `window` tal y como se muestra en este ejemplo:

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

Como puedes ver, los parámetros van entre corchetes, indicando que son opcionales y que dependerán del método al que estemos llamando.

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.nombreMétodo( [parámetros] )
```

Podremos usar cualquiera de las dos referencias anteriores, pero intentaremos dejar la palabra reservada `self`, para scripts más complejos en los que tengamos múltiples marcos y ventanas.

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

```
nombrePropiedad  
nombreMétodo( [parámetros] )
```

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



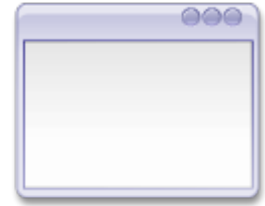
Citas para pensar

“Sólo cerrando las puertas detrás de uno se abren ventanas hacia el porvenir.”

SAGAN, Françoise.

1.1.1.- Gestión de ventanas.

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



[Everaldo Coelho](#) (GNU/GPL)

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

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

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

Lo importante de esa instrucción, es la asignación que hemos hecho en la variable `subVentana`. De esta forma podremos a lo largo de nuestro código, referenciar a la nueva ventana 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 si que es necesario especificar `subVentana`, ya que si escribiéramos `window.close()`, `self.close()` o `close()` estaríamos intentando cerrar nuestra propia ventana (previa confirmación), pero no la `subVentana` que creamos en los pasos anteriores.

Véase el siguiente ejemplo que permite abrir y cerrar una sub-ventana:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
    <title>Apertura y Cierre de Ventanas</title>
    <script type="text/javascript">
      const inicializar = () => {
        document.getElementById("crear-ventana").onclick=crearNueva;
        document.getElementById("cerrar-ventana").onclick=cerrarNueva;
      }

      let nuevaVentana;

      const crearNueva = () => {
        nuevaVentana = window.open("http://www.google.es", "", "height=400,width=800");
      }

      const cerrarNueva = () => {
        if (nuevaVentana) {
          nuevaVentana.close();
        }
      }
    </script>
  </head>
  <body>
    <div>
      <button id="crear-ventana">Crear Nueva Ventana
    </div>
    <div>
      <button id="cerrar-ventana">Cerrar Nueva Ventana
    </div>
  </body>
</html>
```



```
        nuevaVentana.close(); nuevaVentana = null;
    }
}
</script>
</head>
<body onLoad="inicializar()">
<h1>Abrimos y cerramos ventanas</h1>
<form>
    <input type="button" id="crear-ventana" value="Crear Nueva Ventana">
    <input type="button" id="cerrar-ventana" value="Cerrar Nueva Ventana">
</form>
</html>
```

1.1.2.- Propiedades y métodos.

El objeto `window` representa una ventana abierta en un navegador. Si un documento contiene marcos (`<frame>` o `<iframe>`), el navegador crea un objeto `window` para el documento HTML, y un objeto `window` adicional para cada marco.



[quinn.anyu](#) (CC BY-SA)

Propiedades del objeto Window

Propiedad	Descripción
<code>closed</code>	Devuelve un valor Boolean indicando cuando una ventana ha sido cerrada o no.
<code>defaultStatus</code>	Ajusta o devuelve el valor por defecto de la barra de estado de una ventana.
<code>document</code>	Devuelve el objeto <code>document</code> para la ventana.
<code>frames</code>	Devuelve un array de todos los marcos (incluidos iframes) de la ventana actual.
<code>history</code>	Devuelve el objeto <code>history</code> de la ventana.
<code>length</code>	Devuelve el número de frames (incluyendo iframes) que hay en dentro de una ventana.
<code>location</code>	Devuelve la Localización del objeto ventana (URL del fichero).
<code>name</code>	Ajusta o devuelve el nombre de una ventana.
<code>navigator</code>	Devuelve el objeto <code>navigator</code> de una ventana.
<code>opener</code>	Devuelve la referencia a la ventana que abrió la ventana actual.
<code>parent</code>	Devuelve la ventana padre de la ventana actual.
<code>self</code>	Devuelve la ventana actual.
<code>status</code>	Ajusta el texto de la barra de estado de una ventana.

Métodos del objeto Window

Método	Descripción
<code>alert()</code>	Muestra una ventana emergente de alerta y un botón de aceptar.
<code>blur()</code>	Elimina el foco de la ventana actual.
<code>clearInterval()</code>	Resetea el cronómetro ajustado con <code>setInterval()</code> .
<code>setInterval()</code>	Llama a una función o evalúa una expresión en un intervalo especificado (en milisegundos).
<code>close()</code>	Cierra la ventana actual.
<code>confirm()</code>	Muestra una ventana emergente con un mensaje, un botón de aceptar y un botón de cancelar.
<code>focus()</code>	Coloca el foco en la ventana actual.
<code>open()</code>	Abre una nueva ventana de navegación.
<code>prompt()</code>	Muestra una ventana de diálogo para introducir datos.



Debes conocer

El siguiente enlace amplía información sobre el objeto `Window` y todas sus propiedades y métodos.

[Más información y ejemplos sobre el objeto `Window`.](#)

1.2.- Objeto location.

El objeto `location` contiene información referente a la URL actual.

Este objeto, es parte del objeto `window` y accedemos a él a través de la propiedad `window.location`.



[chrisdlugosz \(CC BY\)](#)

El objeto `location` contiene información referente a la URL actual.

Este objeto, es parte del objeto `window` y accedemos a él a través de la propiedad `window.location`.

Propiedades del objeto Location

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

Métodos del objeto Location

--	--

<code>assign()</code>	Carga un nuevo documento.
<code>reload()</code>	Vuelve a cargar la URL especificada en la propiedad <code>href</code> del objeto <code>location</code> .
<code>replace()</code>	Reemplaza el historial actual mientras carga la URL especificada en cadenaURL.



Citas para pensar

“Mil rutas se apartan del fin elegido, pero hay una que llega a él.”

MONTAIGNE, Michel de.



Debes conocer

El siguiente enlace amplía información sobre el objeto `Location` y todas sus propiedades y métodos.

[Más información y ejemplos sobre el objeto `Location`.](#)

1.3.- Objeto navigator.

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



[Caitlinator](#) (CC BY)

Propiedades del objeto Navigator

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

Métodos del objeto Navigator

Método	Descripción
<code>javaEnabled()</code>	Devuelve true si el cliente permite la utilización de Java, en caso contrario, devuelve false.



Debes conocer

El siguiente enlace amplía información sobre el objeto `Navigator` y todas sus propiedades y métodos.



Autoevaluación

Si queremos introducir datos a través de una ventana de diálogo en nuestra aplicación de JavaScript lo haremos con:

- ☐ La propiedad `input` del objeto `window`.
- ☐ La propiedad `userAgent` del objeto `navigator`.
- ☐ El método `prompt` del objeto `window`.

No es correcta. No existe tal propiedad en el objeto `window`.

Incorrecta. Esa propiedad nos permite consultar información sobre el navegador utilizado.

Muy bien. Este método te permitirá solicitar datos al usuario, mediante una ventana de diálogo que podrás usar en tu aplicación.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

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`, y puede ser accedido a través de la propiedad `window.document` o directamente `document` (ya que podemos omitir la referencia a la `window` actual).



[infilimity](#) (CC BY-NC-SA)

Colecciones del objeto Document

Colección	Descripción
<code>anchors[]</code>	Es un array que contiene todos los hiperenlaces del documento.
<code>forms[]</code>	Es un array que contiene todos los formularios del documento.
<code>images[]</code>	Es un array que contiene todas las imágenes del documento.
<code>links[]</code>	Es un array que contiene todos los enlaces del documento.

Propiedades del objeto Document

Propiedad	Descripción
<code>cookie</code>	Devuelve todos los nombres/valores de las cookies en el documento.
<code>domain</code>	Cadena que contiene el nombre de dominio del servidor que cargó el documento.
<code>referrer</code>	Cadena que contiene la URL del documento desde el cuál llegamos al documento actual.
<code>title</code>	Devuelve o ajusta el título del documento.
<code>URL</code>	Devuelve la URL completa del documento.

Propiedades del objeto Document

Método	Descripción
--------	-------------

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



Debes conocer

El siguiente enlace amplía información sobre el objeto `Document` todas sus propiedades y métodos.

[Más información y ejemplos sobre el objeto `Document`.](#)

2.- Objetos nativos en Javascript.



Caso práctico

El lenguaje JavaScript es un lenguaje basado en objetos. A **Antonio** le suena un poco el tema de objetos aunque nunca trabajó intensivamente con ellos. Como todos los lenguajes que incorporan sus funciones para realizar acciones, conversiones, etc., JavaScript también dispone de unos objetos nativos que le van a permitir a **Antonio** el realizar muchas de esas tareas.



Estos objetos, hacen referencia al trabajo con cadenas de texto, operaciones matemáticas, números, valores booleanos y trabajo con fechas y horas.

Ésto le va a ser muy útil para realizar su aplicación ya que tendrá que realizar diferentes tipos de conversiones de datos, trabajar intensivamente con cadenas y por supuesto con fechas y horas.

Aunque no hemos visto como crear objetos, sí que ya hemos dado unas pinceladas a lo que son los objetos, propiedades y métodos.

En esta sección vamos a echar una ojeada a objetos que son nativos en JavaScript, ésto es, aquello que JavaScript nos da, listos para su utilización en nuestra aplicación.

Echaremos un vistazo a los objetos **String**, **Math**, **Number**, **Boolean** Y **Date**.



Citas para pensar

“Si me hubieran hecho objeto sería objetivo, pero me hicieron sujeto.”

BERGAMÍN, José.



Reflexiona

¿Te has parado a pensar alguna vez que nuestro mundo está rodeado de objetos por todas partes?

¿Sabes que prácticamente, todos esos objetos tienen algunas propiedades como pueden ser tamaño, color, peso, tipo de corriente que usan, temperatura, tipo de combustible, etc.?

¿Sabes que también podemos realizar acciones con esos objetos, como pueden ser encender, apagar, mover, abrir, cerrar, subir temperatura, bajar temperatura, marcar número, colgar, etc.?

2.1.- Objeto String.

Una cadena (string) consta de uno o más caracteres de texto, rodeados de comillas simples o dobles; da igual cuales usemos ya que se considerará una cadena de todas formas, pero en algunos casos resulta más cómodo el uso de unas u otras.

Recuerda que de cómo definir las cadenas ya hemos hablado y que no se te olviden las **plantillas de cadenas** que te hacen muy fácil su composición en vez de estar concatenando que a veces resulta engorroso.



faccig (CC BY-SA)

Caracteres especiales o caracteres de escape.

La forma en la que se crean las cadenas en JavaScript, hace que cuando tengamos que emplear ciertos caracteres especiales en una cadena de texto, tengamos que escaparlos, empleando el símbolo \ seguido del carácter.

Vemos aquí un listado de los caracteres especiales o de escape en JavaScript:

Caracteres de escape y especiales en JavaScript

Símbolos	Explicación
\"	Comillas dobles.
\'	Comilla simple.
\\	Barra inclinada.
\b	Retroceso.
\t	Tabulador.
\n	Nueva línea.
\r	Salto de línea.
\f	Avance de página.



Debes conocer

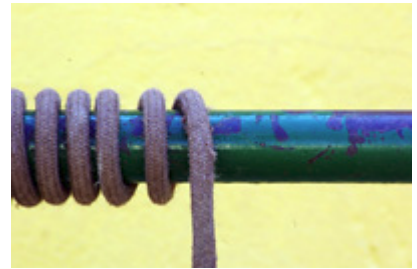
El siguiente enlace amplía información sobre el objeto `String` y todas sus propiedades y métodos.

[Más información y ejemplos sobre el objeto String.](#)

2.1.1.- Propiedades y métodos del objeto String.

Para crear un objeto `String` lo podremos hacer de la siguientes formas:

```
let cadena1 = new String('Primera cadena');
let cadena2 = "Segunda cadena";
let cadena3 = 'Tercera cadena';
let todasCadenas = `${cadena1} - ${cadena2} - ${cadena3}`
```



[Josep Ma. Rosell](#) (CC BY)


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

```
cadena.propiedad;
cadena.metodo( [parámetros] );
```

Propiedades del objeto String

Propiedad	Descripción
<code>length</code>	Contiene la longitud de una cadena.

Métodos del objeto String

Métodos	Descripción
<code>charAt()</code>	Devuelve el carácter especificado por la posición que se indica entre paréntesis.
<code>charCodeAt()</code>	Devuelve el  Unicode del carácter especificado por la posición que se indica entre paréntesis.
<code>concat()</code>	Une una o más cadenas y devuelve el resultado de esa unión.
<code>fromCharCode()</code>	Convierte valores Unicode a caracteres.
<code>indexOf()</code>	Devuelve la posición de la primera ocurrencia del carácter buscado en la cadena.
<code>lastIndexOf()</code>	Devuelve la posición de la última ocurrencia del carácter buscado en la cadena.

Métodos	Descripción
<code>match()</code>	Busca una coincidencia entre una expresión regular y una cadena y devuelve las coincidencias o <code>null</code> si no ha encontrado nada.
<code>replace()</code>	Busca una subcadena en la cadena y la reemplaza por la nueva cadena especificada.
<code>search()</code>	Busca una subcadena en la cadena y devuelve la posición dónde se encontró.
<code>slice()</code>	Extrae una parte de la cadena y devuelve una nueva cadena.
<code>split()</code>	Divide una cadena en un array de subcadenas.
<code>substr()</code>	Extrae los caracteres de una cadena, comenzando en una determinada posición y con el número de caracteres indicado.
<code>substring()</code>	Extrae los caracteres de una cadena entre dos índices especificados.
<code>toLowerCase()</code>	Convierte una cadena en minúsculas.
<code>toUpperCase()</code>	Convierte una cadena en mayúsculas.

2.2.- Objeto Math.

Ya vimos anteriormente algunas funciones, que nos permitían convertir cadenas a diferentes formatos numéricos (`parseInt`, `parseFloat`). A parte de esas funciones, disponemos de un objeto `Math` en JavaScript, que nos permite realizar operaciones matemáticas. El objeto `Math` no posee un constructor (no nos permitirá por lo tanto crear o instanciar nuevos objetos que sean de tipo `Math`), por lo que para llamar a sus propiedades y métodos, lo haremos anteponiendo `Math` a la propiedad o el método. Por ejemplo:



[conskeptical](#) (CC BY-SA)

```
let x = Math.PI;           // Devuelve el número PI.
let 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
<code>abs(x)</code>	Devuelve el valor absoluto de x.
<code>acos(x)</code>	Devuelve el arcocoseno de x, en radianes.
<code>asin(x)</code>	Devuelve el arcoseno de x, en radianes.
<code>atan(x)</code>	Devuelve el arcotangente de x, en radianes con un valor entre $-\frac{\pi}{2}$ y $\frac{\pi}{2}$.

Método	Descripción
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.

2.3.- Objeto Number.

El objeto `Number` se usa muy raramente, ya que para la mayor parte de los casos, JavaScript satisface las necesidades del día a día con los valores numéricos que almacenamos en variables. Pero el objeto `Number` contiene alguna información y capacidades muy interesantes para programadores más serios.



[Darwin Bell](#) (CC BY)

Lo primero, es que el objeto `Number` contiene propiedades que nos indican el rango de números soportados en el lenguaje. El número más alto es $1.79E + 308$; el número más bajo es $2.22E-308$. Cualquier número mayor que el número más alto, será considerado como infinito positivo, y si es más pequeño que el número más bajo, será considerado infinito negativo.

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

El objeto `Number`, es un objeto envoltorio para valores numéricos primitivos.




Los objetos `Number` son creados con `new Number()`.

Propiedades del objeto Number

Propiedad	Descripción
<code>constructor</code>	Devuelve la función que creó el objeto <code>Number</code> .
<code>MAX_VALUE</code>	Devuelve el número más alto disponible en JavaScript.
<code>MIN_VALUE</code>	Devuelve el número más pequeño disponible en JavaScript.
<code>NEGATIVE_INFINITY</code>	Representa a infinito negativo (se devuelve en caso de overflow).
<code>POSITIVE_INFINITY</code>	Representa a infinito positivo (se devuelve en caso de overflow).
<code>prototype</code>	Permite añadir nuestras propias propiedades y métodos a un objeto.

Métodos del objeto Number

<code>toExponential(x)</code>	Convierte un número a su notación exponencial.
<code>toFixed(x)</code>	Formatea un número con x dígitos decimales después del punto decimal.
<code>toPrecision(x)</code>	Formatea un número a la longitud x.

toString()	<p>Convierte un objeto <code>Number</code> en una cadena.</p> <ul style="list-style-type: none">✔ 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 <code>Number</code> .

2.4.- Objeto Boolean.

El objeto `Boolean` se utiliza para convertir un valor no lógico, a un valor lógico (`true` o `false`).



[Everaldo Coelho \(GNU/GPL\)](#)

Propiedades del objeto Boolean

constructor	Devuelve la función que creó el objeto <code>Boolean</code> .
prototype	Te permitirá añadir propiedades y métodos a un objeto.

Métodos del objeto Boolean

toString()	Convierte un valor <code>Boolean</code> a una cadena y devuelve el resultado.
valueOf()	Devuelve el valor primitivo de un objeto <code>Boolean</code> .



Para saber más

En el siguiente enlace podrás encontrar ejemplos sobre el uso del objeto `Boolean`.

[Más información y ejemplos sobre el objeto `Boolean`.](#)



Autoevaluación

¿Para usar un objeto `Math` deberemos instanciarlo antes de poder usarlo?

- ☐ Sí.
- ☐ No.

No es correcto porque este objeto no dispone de constructor, con lo que para usarlo lo llamaremos directamente seguido de su propiedad

o método.

Es correcto porque este objeto no dispone de constructor con lo que no podremos crear instancias del mismo.

Solución

1. Incorrecto
2. Opción correcta

2.5.- Objeto Date.

El objeto `Date` se utiliza para trabajar con fechas y horas. Los objetos `Date` se crean con `new Date()`.

Hay 4 formas de instanciar (crear un objeto de tipo `Date`):

```
let dia1 = new Date();
let dia2 = new Date(milisegundos);
let dia3 = new Date(cadena de Fecha);
let dia4 = new Date(año, mes, día, horas, minutos, segundos, miliseg
```



[Everaldo Coelho](#) (GNU/GPL)

Propiedades del objeto Date

Propiedad	Descripción
constructor	Devuelve la función que creó el objeto <code>Date</code> .
prototype	Te permitirá añadir propiedades y métodos a un objeto.

Métodos del objeto Date

<code>getDate()</code>	Devuelve el día del mes (de 1-31).
<code>getDay()</code>	Devuelve el día de la semana (de 0-6).
<code>getFullYear()</code>	Devuelve el año (4 dígitos).
<code>getHours()</code>	Devuelve la hora (de 0-23).
<code>getMilliseconds()</code>	Devuelve los milisegundos (de 0-999).
<code>getMinutes()</code>	Devuelve los minutos (de 0-59).
<code>getMonth()</code>	Devuelve el mes (de 0-11).
<code>getSeconds()</code>	Devuelve los segundos (de 0-59).
<code>getTime()</code>	Devuelve los milisegundos desde media noche del 1 de Enero de 1970.
<code>getTimezoneOffset()</code>	Devuelve la diferencia de tiempo entre <u>GMT</u> y la hora local, en minutos.

<code>getUTCDate()</code>	Devuelve el día del mes en base a la hora UTC (de 1-31).
<code>getUTCDay()</code>	Devuelve el día de la semana en base a la hora UTC (de 0-6).
<code>getUTCFullYear()</code>	Devuelve el año en base a la hora UTC (4 dígitos).
<code>setDate()</code>	Ajusta el día del mes del objeto (de 1-31).
<code>setFullYear()</code>	Ajusta el año del objeto (4 dígitos).
<code>setHours()</code>	Ajusta la hora del objeto (de 0-23).

