Desarrollo Entorno Cliente

# 3.A. Modelo de objetos predefinidos en JavaScript.

## 1.- Objetos JavaScript del navegador

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), describe el contenido del documento como un conjunto de objetos, sobre los que un programa de JavaScript puede interactuar.

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.

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

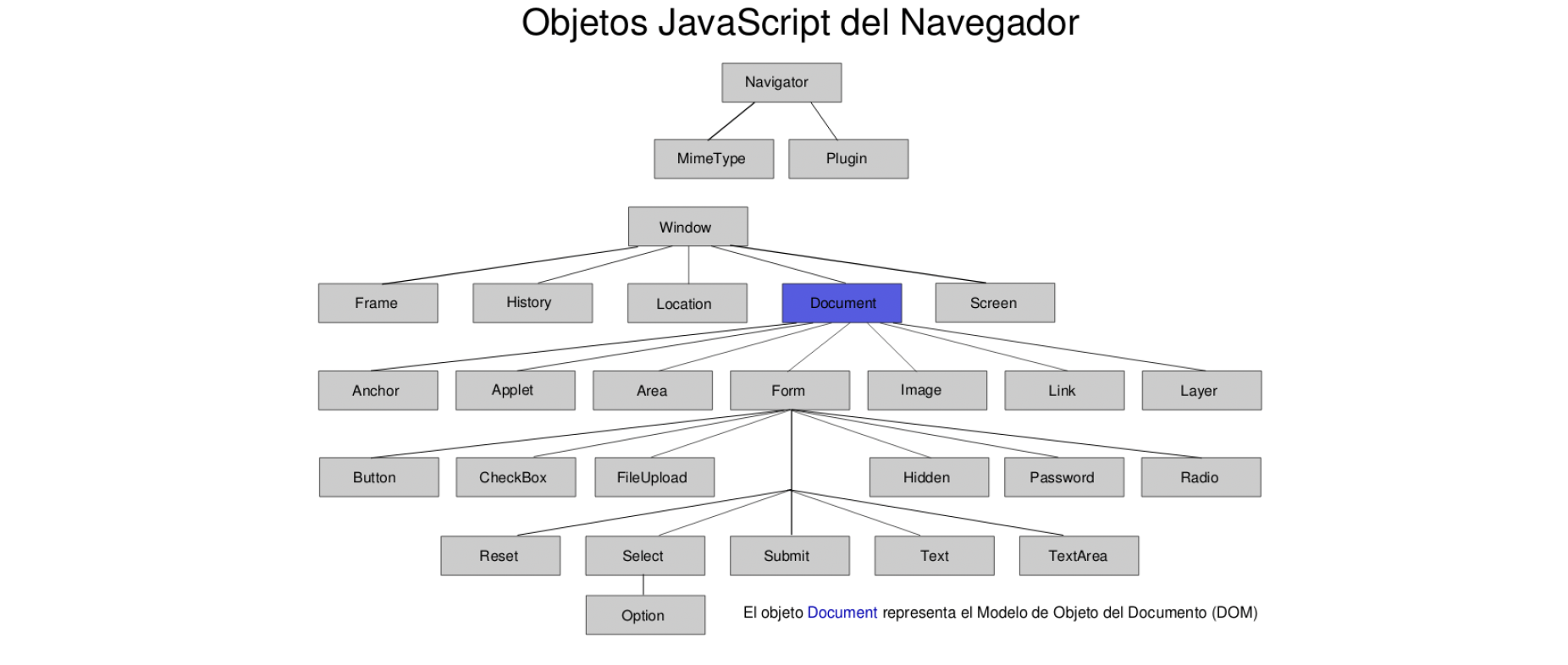
La forma de acceder a una propiedad de un objeto: nombreObjeto.propiedad

La forma de acceder a un método de un objeto: nombreObjeto.metodo([parámetros opcionales])

Podemos acceder a una propiedad o método de un objeto como si se tratara de un array: nombreObjeto[‘nombre propiedad’]

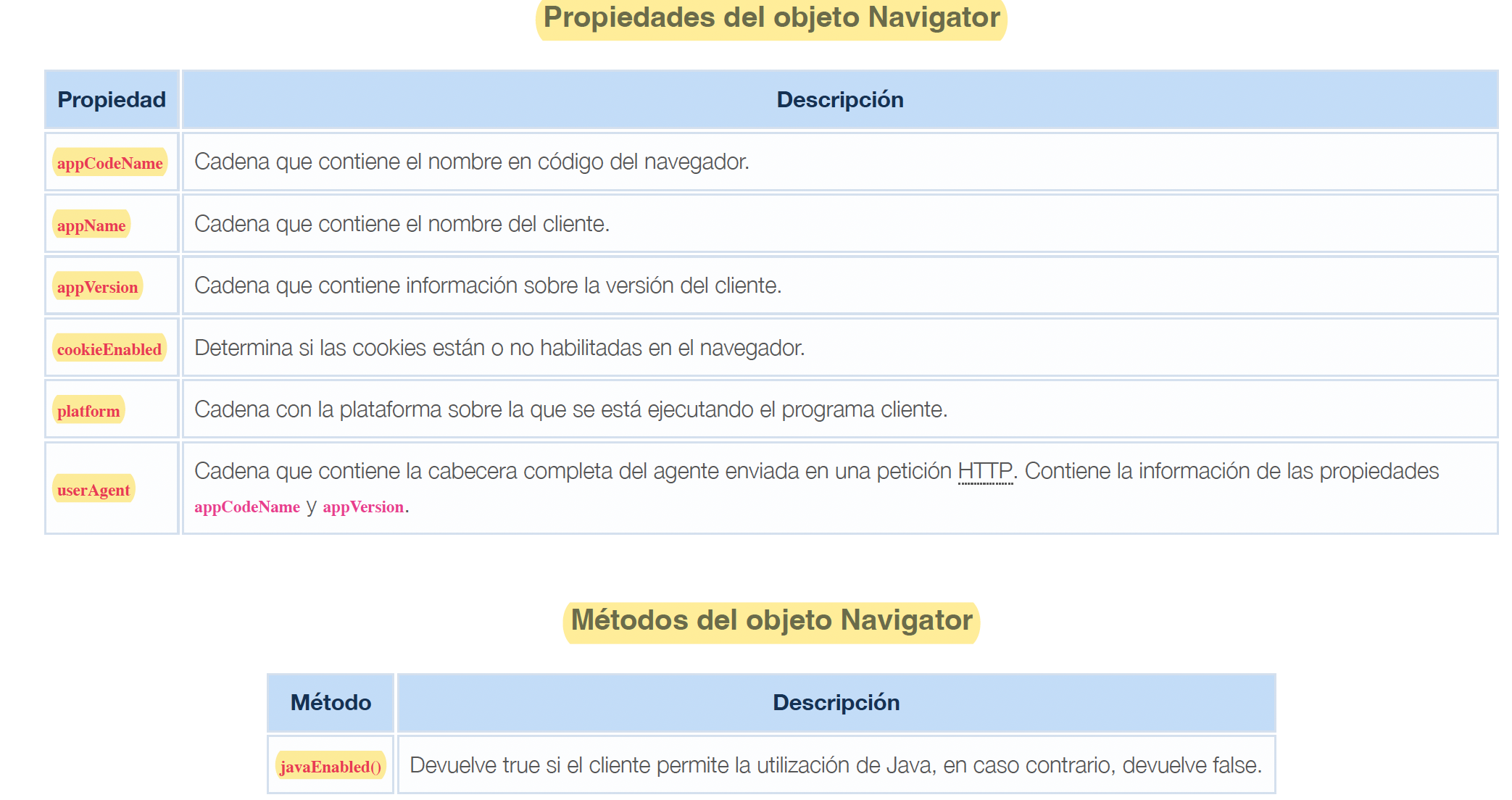
Objetos de alto nivel: window, location, navigator y document.

Jerarquía de objetos del navegador:



## 1.1.- Objeto navigator

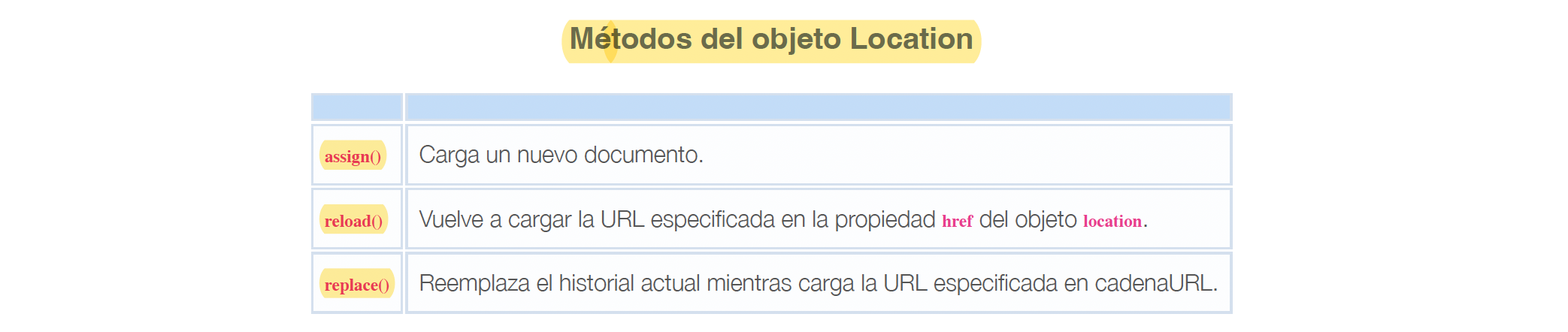
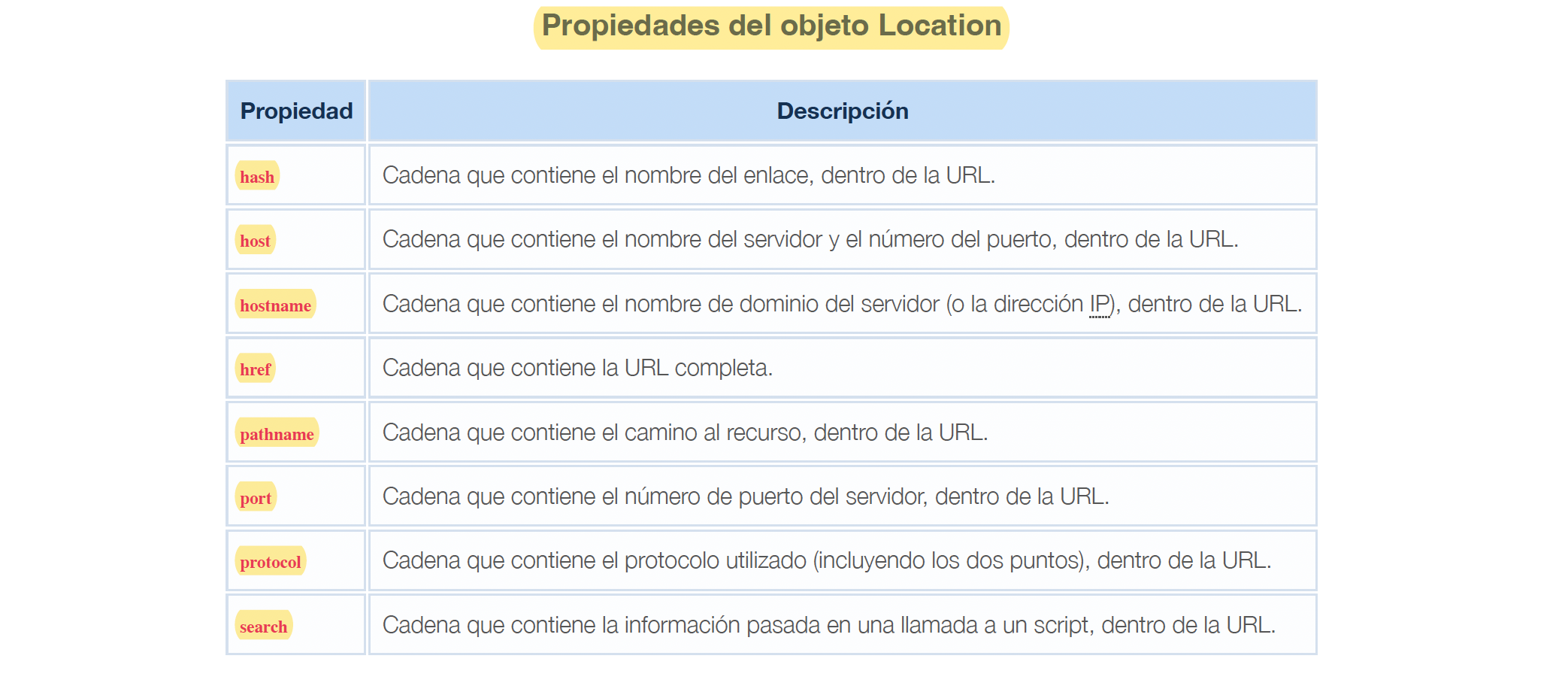
Este objeto navigator, contiene información sobre el navegador que estamos utilizando cuando abrimos una URL o un documento local.



Mas información: <https://developer.mozilla.org/es/docs/Web/API/Navigator>

## 1.2.- Objeto location.

El objeto location contiene información referente a la URL actual. Es parte del objeto window y accedemos a él a través de la propiedad window.location.



Más información del objeto **location**: <https://developer.mozilla.org/es/docs/Web/API/Location>

## 1.3.- Objeto Windows.

En la jerarquía de objetos, tenemos en la parte superior los objetos: **navigator** y **window**.

El objeto **windows** es el contenedor principal de todo el contenido que se visualiza en una ventana o pestaña del navegador.

Tan pronto como se abre una ventana 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 la sección del contenido, las dimensiones de la ventana, así como todo lo que rodea al área de contenido (barras de desplazamiento, barra de herramientas, barra de estado, etc).

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**: los navegadores más modernos, tienen la posibilidad de crear nuevas pestañas así como nuevas ventanas de navegador (para JavaScript ambos son objetos window).

**Acceso a propiedades y métodos.**

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

Window.nombrePropiedad

Window.nombreMétodo([ parámetros])

Los parámetros 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 ])

**Self** se dejará para scripts más complejos en los que tengamos múltiples marcos y ventanas.

Podemos omitir el objeto **window**, en referencia a los objetos dentro de esta ventana.

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.

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.**

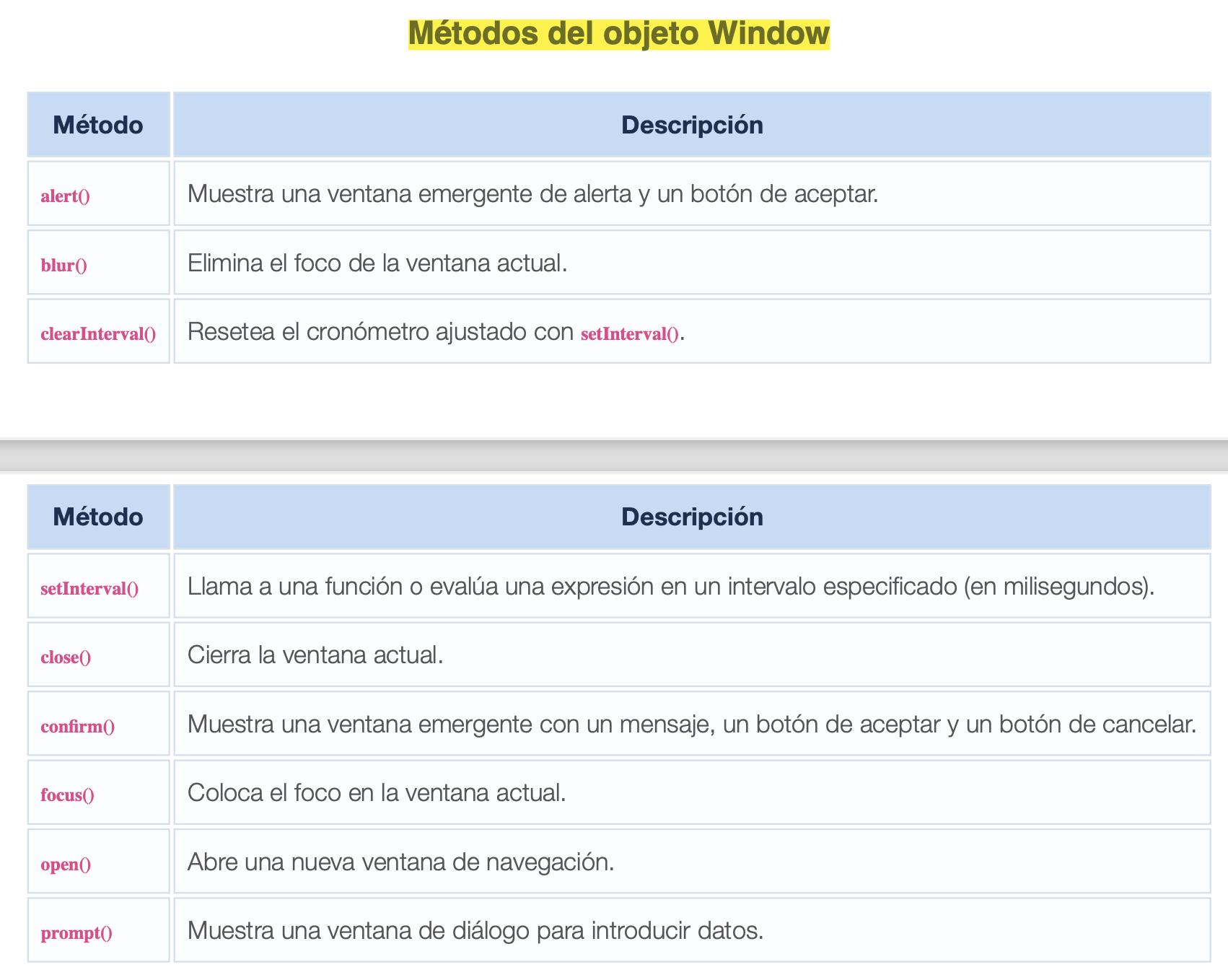
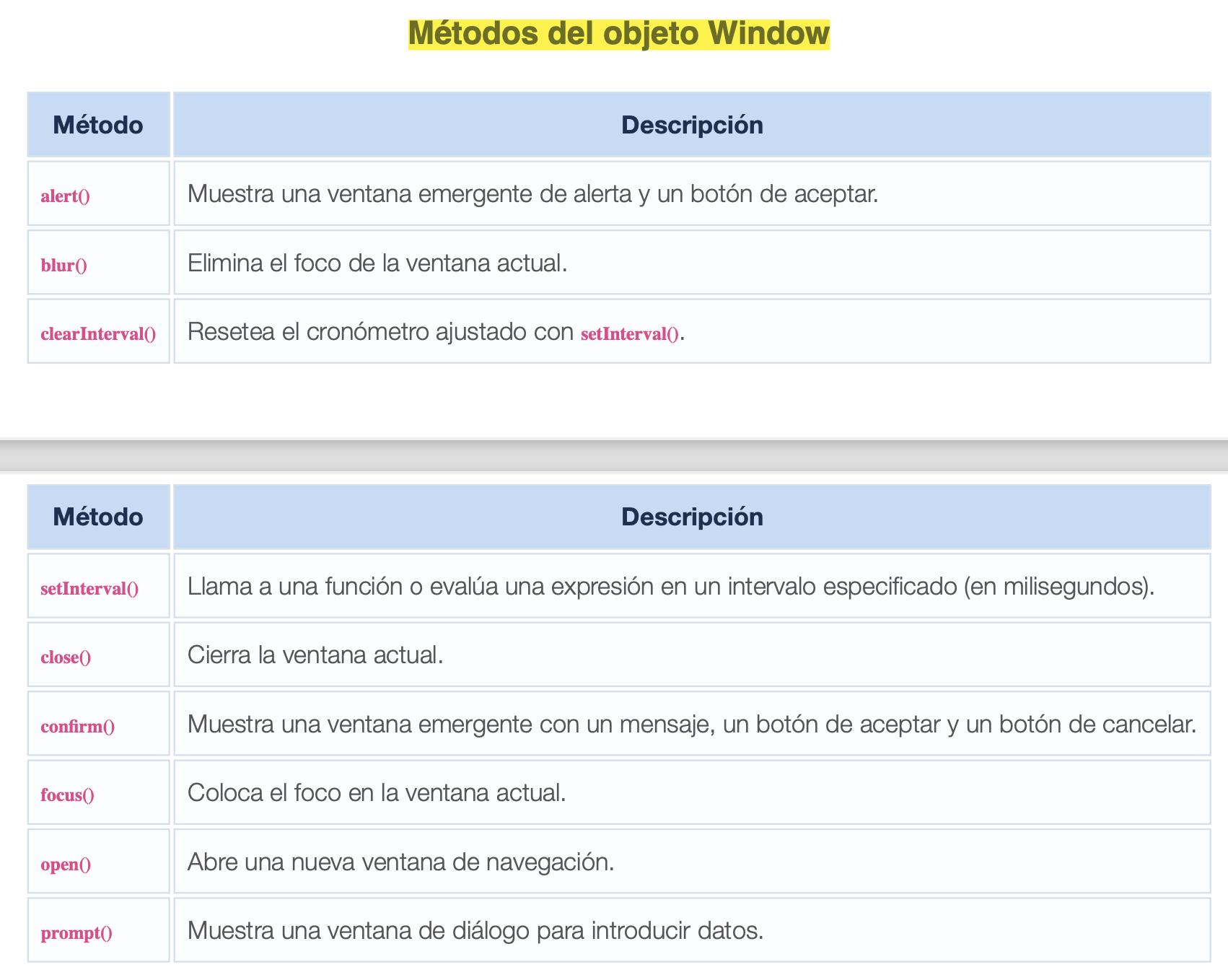
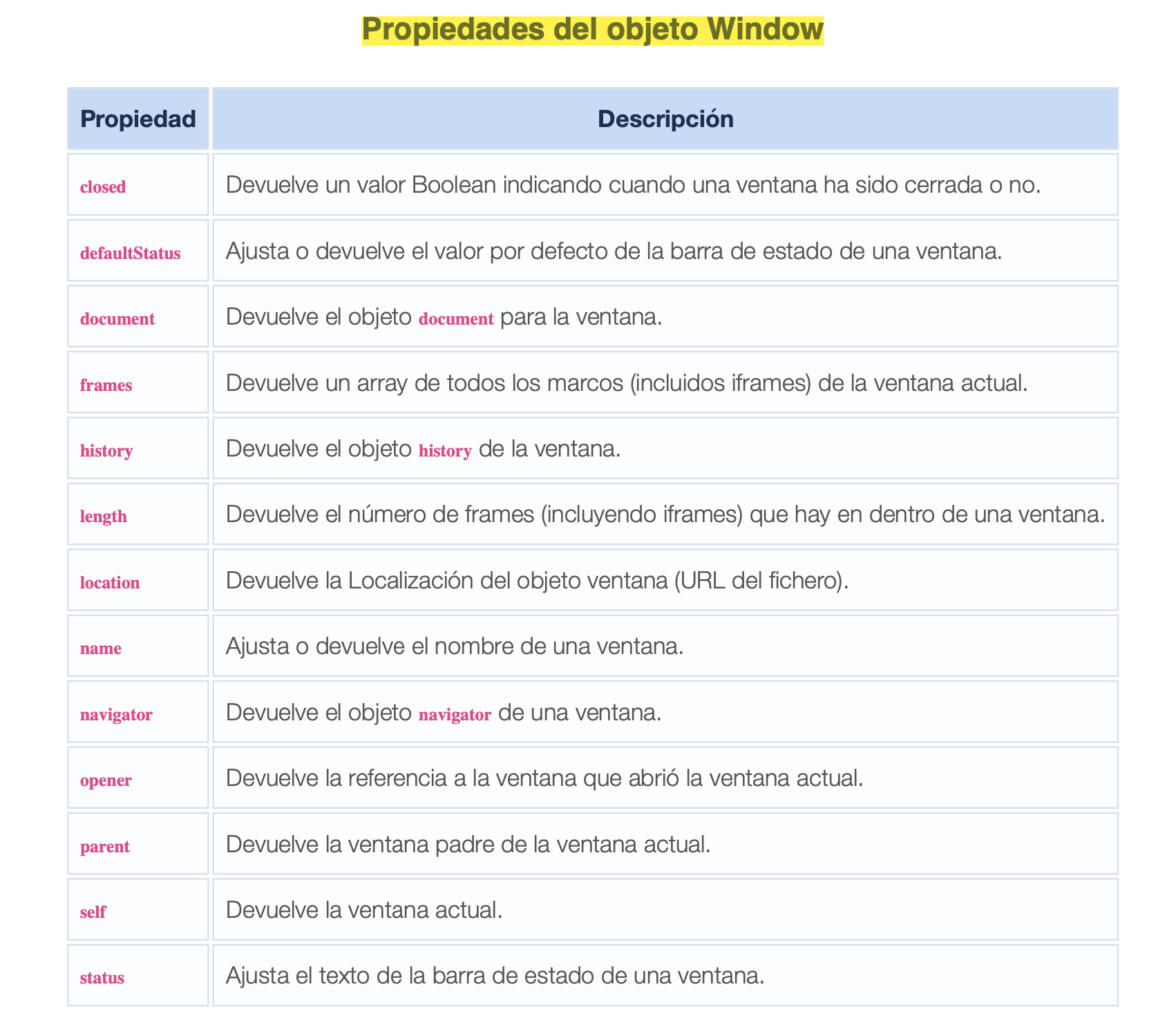
El método que genera una nueva ventana es **window.open().**

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):

Var subVentana = window.open(“nueva.html”, “nueva”, “height=800, width=600”);

Lo importante de esa instrucción, es la asignación que hemos hecho en la varible subVentana. Y podremos referenciar: subVentana.close()

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



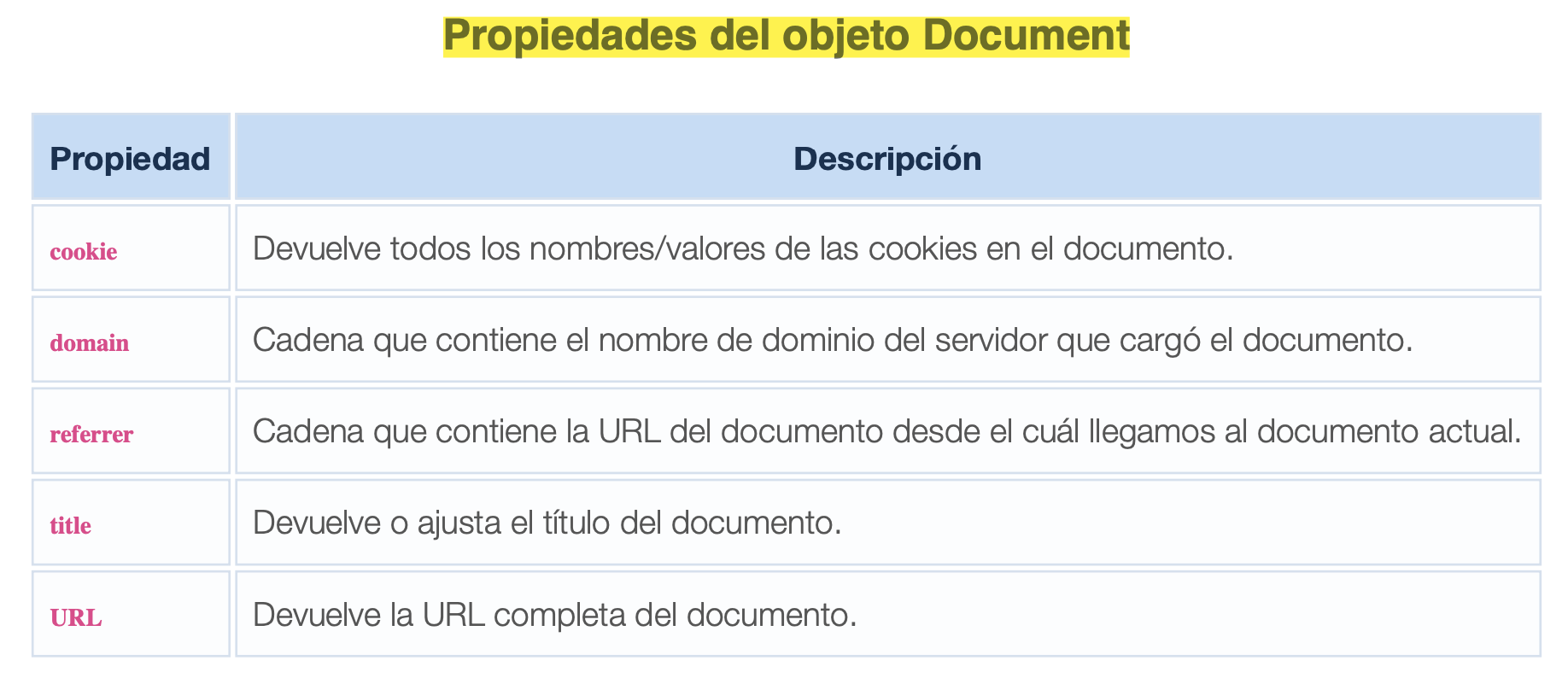
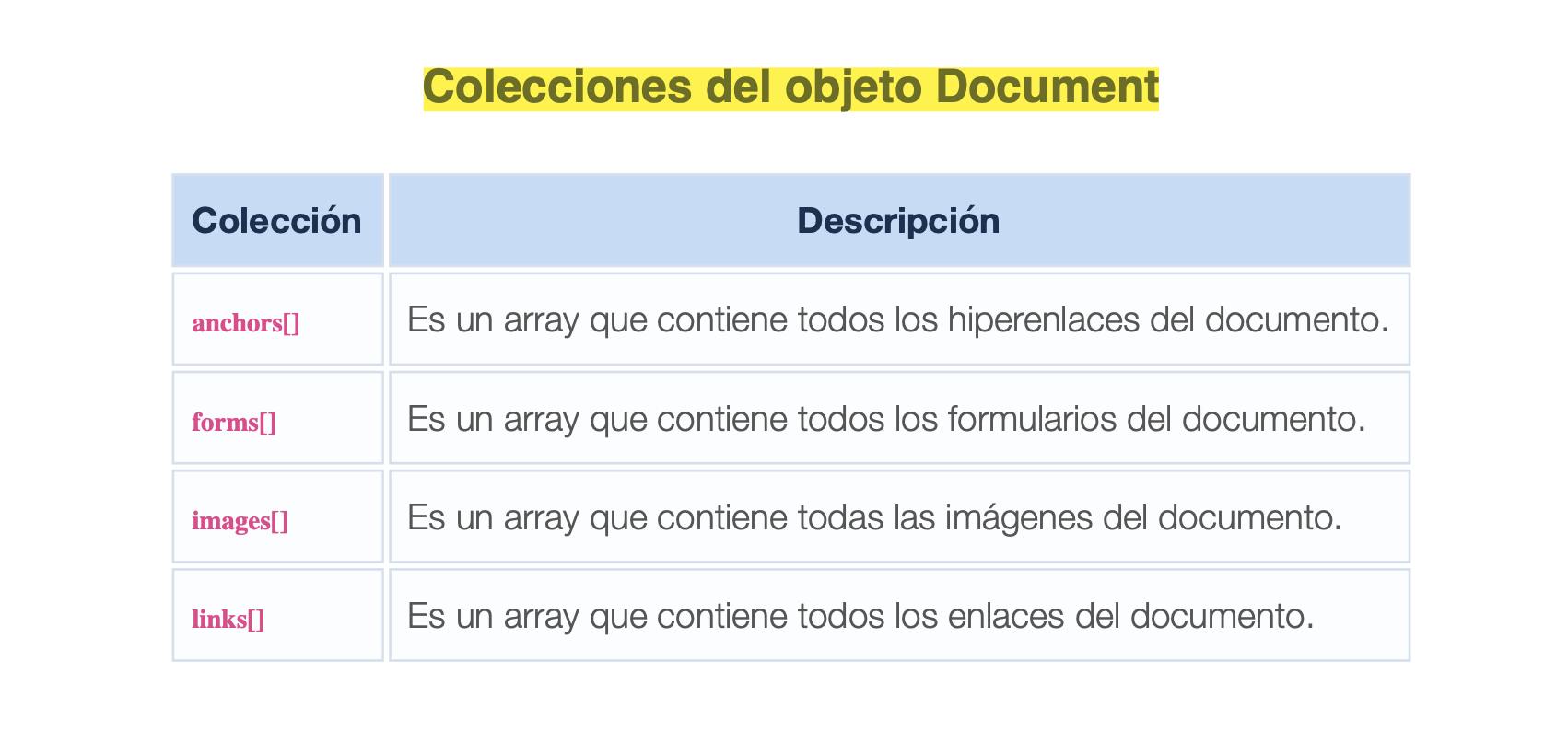
Más información: <https://developer.mozilla.org/es/docs/Web/API/Window>

## 1.4.- Object Document

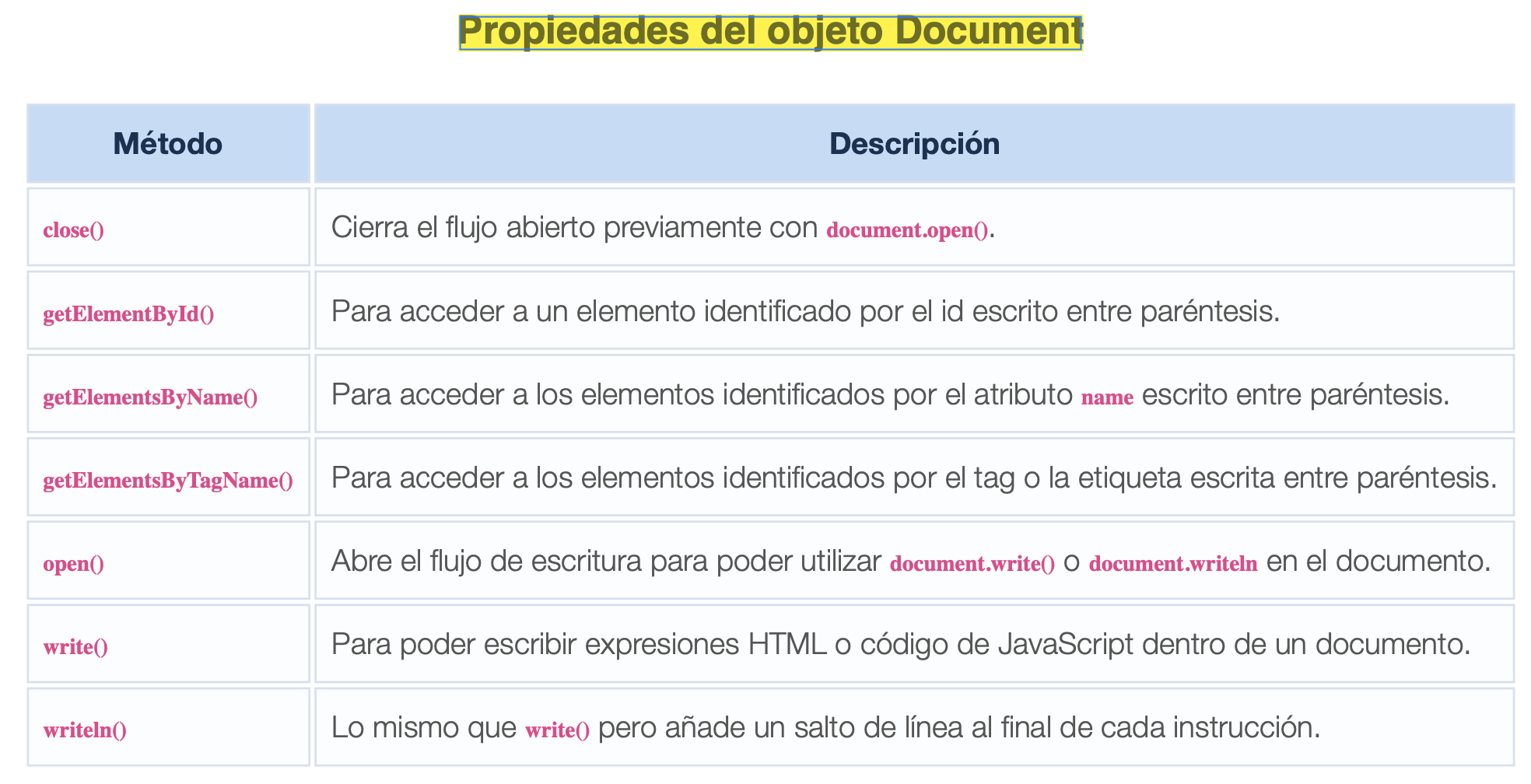
Cada documento cargado en una ventana del navegador, será un objeto de tipo **document**.

El objeto **document**, proporciona el acceso a todos los elementos HTML dentro de una página.

Forma parte del objeto **window**, y puede ser accedido a través de la propiedad **window**.**document** o directamente **document**.



METODOS



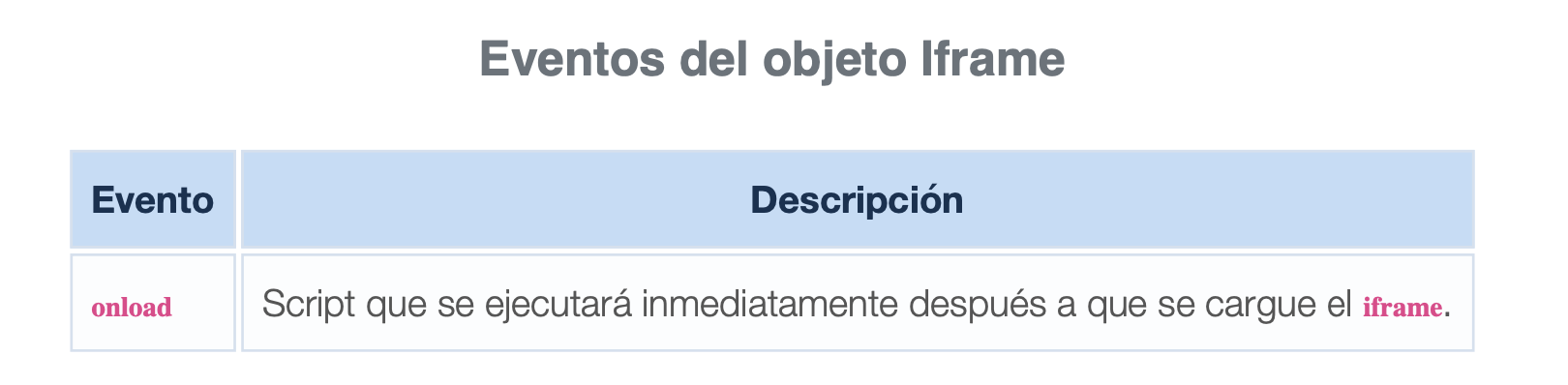
Más información: <https://developer.mozilla.org/es/docs/Web/API/Document>

## 2.- Marcos

Aunque los marcos son elementos de una página cada vez más en desuso, pueden resultar interesantes para realizar algunos proyectos.

Previo al estándar HTML5, para definir los marcos se utilizaba la etiqueta frame para representar a un marco HTML. La etiqueta **<frame>** identifica una ventana particular, dentro de un conjunto de marcos (frameset).

Actualmente en HTML5, la etiqueta <frame> no está soportada y se ha sustituido por la etiqueta **<iframe>**.



## 2.1.- Jerarquías

**Marcos: padres e hijos.**

Cuando trabajamos con marcos o iframes, podemos referenciar a las ventanas como: **frame, top y parent.**

Aunque el uso de **marcos** o iframes es completamente válido en HTML, en términos de usabilidad no se recomiendan, por lo que su uso está en verdadero declive. El problema fundamental con los marcos, es que la **URL** principal de nuestro navegador no cambia y los buscadores como Google, Bing, etc no idexan bien los frames.

## 2.2.- Comunicación entre marcos

**Referencias padre-hijos**

Desde el momento en el que el documento padre contiene uno o más marcos, ese documento padre mantiene un array con sus marcos hijo. Podemos acceder a un marco a través de la sintaxis de array o por el nombre que le hemos dado a ese marco, por el id o con el atributo **name** que hemos puesto en la marca **<frame>.**

**Ejemplo:**

[window.]frames[n].objeto-función-variable-nombre

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

[window.]nombreDelMarco.objeto-función-variable-nombre

El índice numérico n, que indica el número de frame, está basado en el orden en el que aparecen en el documento frameset. Se recomienda que pongamos un nombre a cada frame en dicho documento, ya que así la referencia a utilizar será mucho más fácil.

**Referencias hijo-padre**

Es más común enlazar scripts al documento padre (**frameset**), ya que éste se carga una vez y permanecerá cargado con los mismos datos, aunque hagamos modificaciones dentro de los marcos.

Desde el punto de vista de un documento hijo (aquel que está en un frame), su antecesor en la jerarquía será denominado el padre (**parent**).

parent.objeto-función-variable-nombre

Si el elemento al que accedemos en el padre es una función que devuelve un valor, el valor devuelto será enviado al hijo sin ningún tipo de problemas.

var valor=parent.NombreFuncion();

Además como la ventana padre está en el top de la jerarquía de ventanas, opcionalmente podríamos escribir:

var valor=top.NombreFuncion();

**Referencias Hijos-Hijos.**

Una de las propiedades de cualquier ventana o marco es su padre (parent – el cuál será null cuando estamos hablando de una ventana sin hijos). Por lo tanto, la forma de comunicar dos ventanas o marcos hermanos va a ser siempre referenciándolos a través de su padre, ya que es el único nexo de unión entre ambos (los dos tienen el mismo padre).

Podemos utilizar alguno de los siguientes formatos:

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 sub-ventanas, empleando el método **open()** del objeto **window**.

Cada objeto **window** tiene una propiedad llamada **opener**.

Esta propiedad contiene la referencia a la ventana o marco, que ha abierto ese objeto **window** empleando el método **open()**. Para la ventana principal el valor de opener será **null**. Debido a que opener es una referencia válida a la ventana padre que abrió las otras, podemos emplearlo para iniciar la referencia a objetos de la ventana original (padre) desde la ventana hija. Es semejante a lo que vimos con frames, pero en este caso es entre ventanas independientes del navegador.

## 3.- Objetos nativos en JavaScript

Una de las características del lenguaje JavaScript es que es un lenguaje orientado a objetos, específicamente orientado a prototipos. Objetos **String, Math, Number, Boolean y Date.**

## 3.1.- 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.

Podremos emplear las comillas dobles o simples:

var cadena = '<input type="checkbox" name="coche" />Audi A6';

var cadena = "<input type='checkbox' name='coche' />Audi A6";

Si queremos emplear comillas dobles al principio y fin de la cadena, y que en el contenido aparezcan también comillas dobles, tendríamos que escaparlas con \”, por ejemplo:

var cadena = "<input type=\"checkbox\" name=\"coche\" />Audi A6";

Cuando estamos hablando de cadenas muy largas, podríamos concatenarlas con + (Más.)= (Igual.)

Si queremos concatenar el contenido de una variable dentro de una cadena de texto emplearemos el símbolo + :

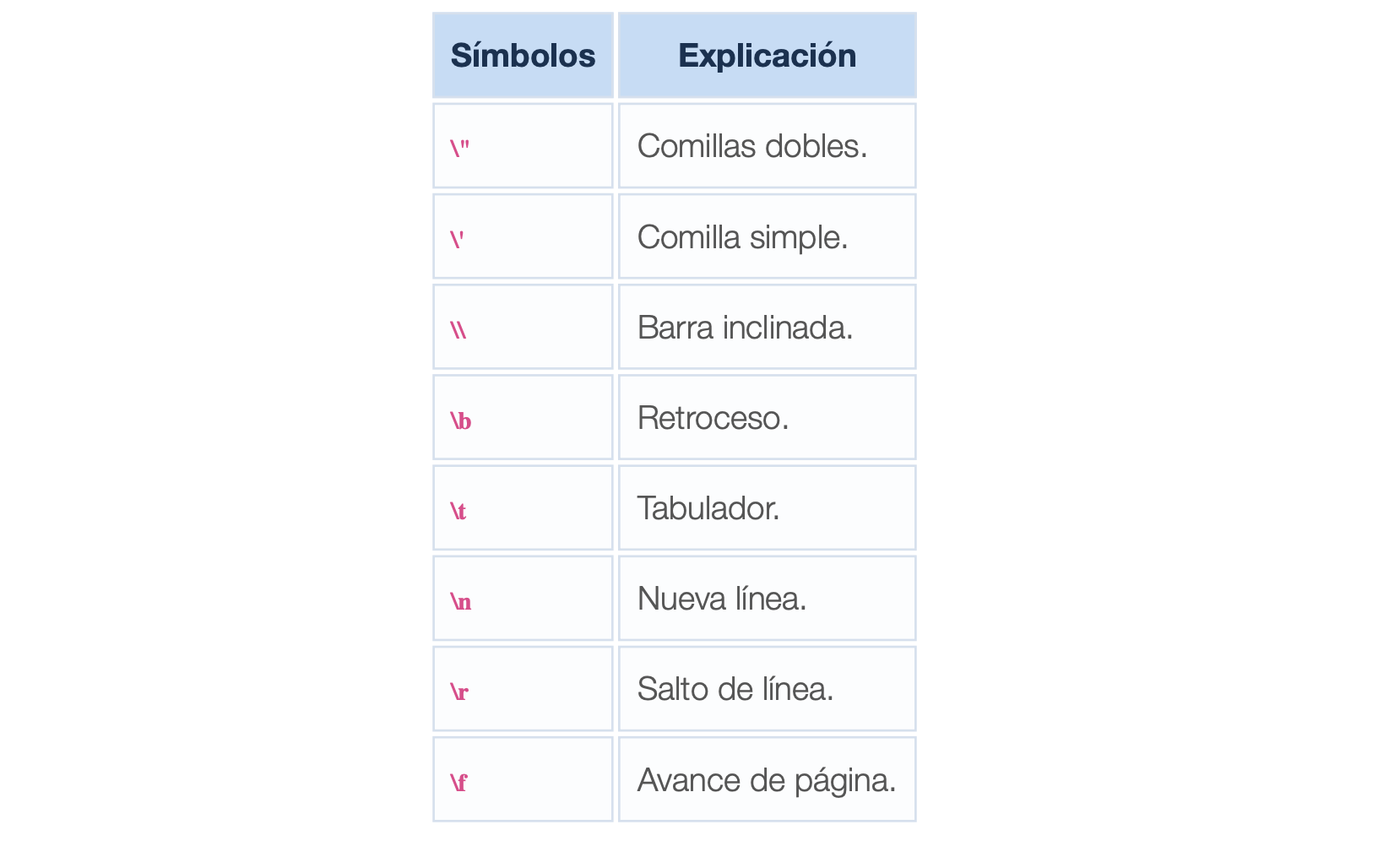
nombreEquipo = prompt("Introduce el nombre de tu equipo favorito:","");

var mensaje= "El " + nombreEquipo + " ha sido el campeón de la Copa del Rey!";

alert(mensaje);

**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.



Para crear un objeto String lo podremos hacer de la siguiente forma:

var miCadena = new String("texto de la cadena");

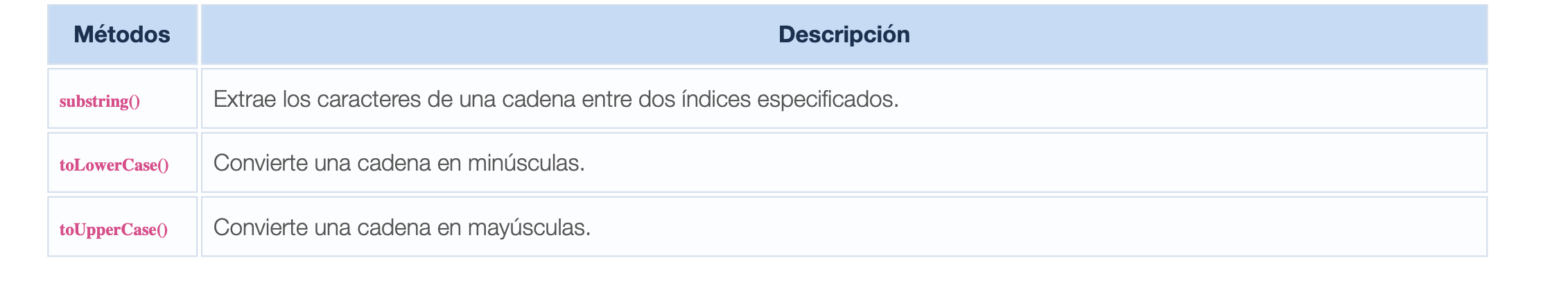
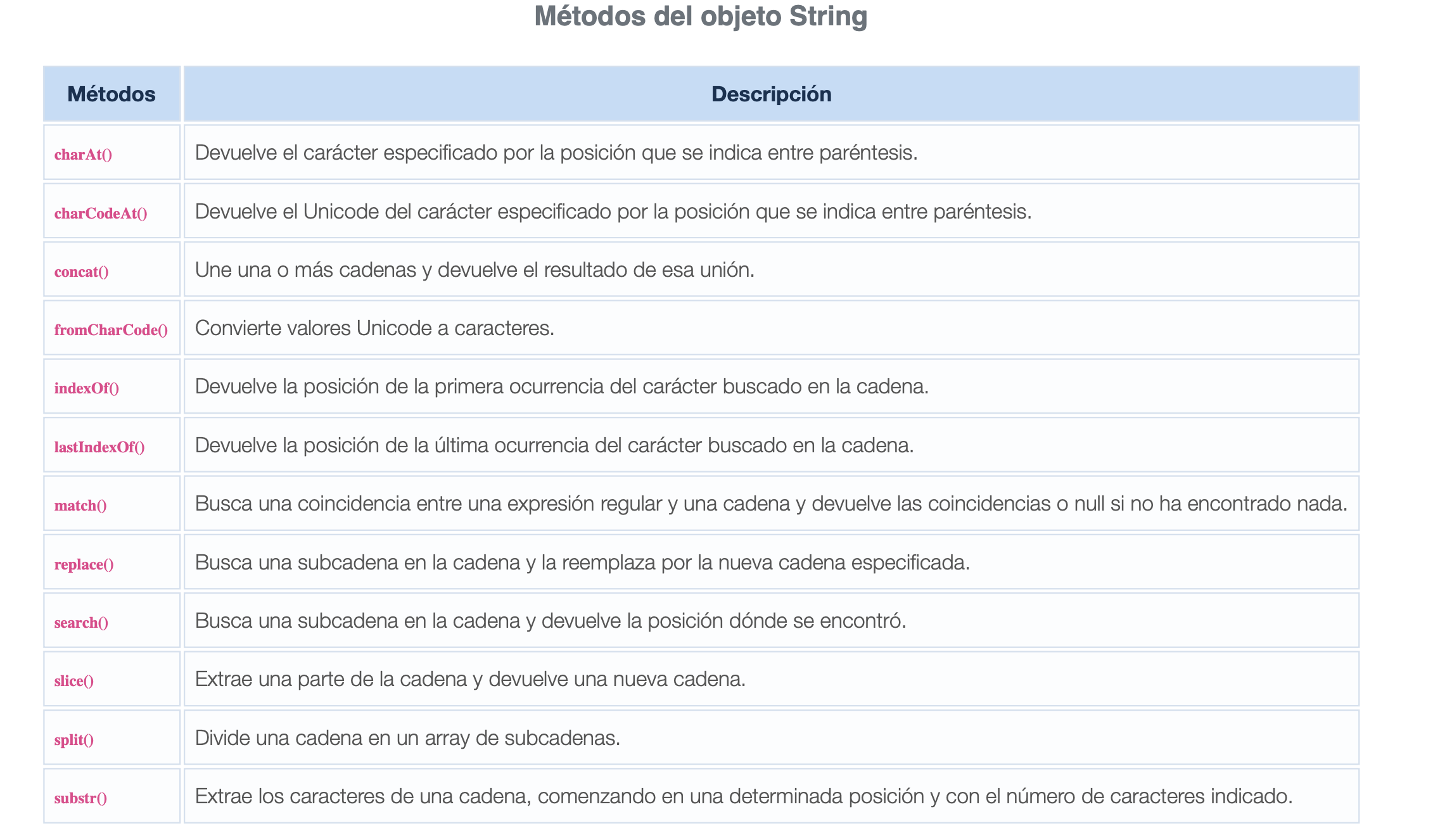
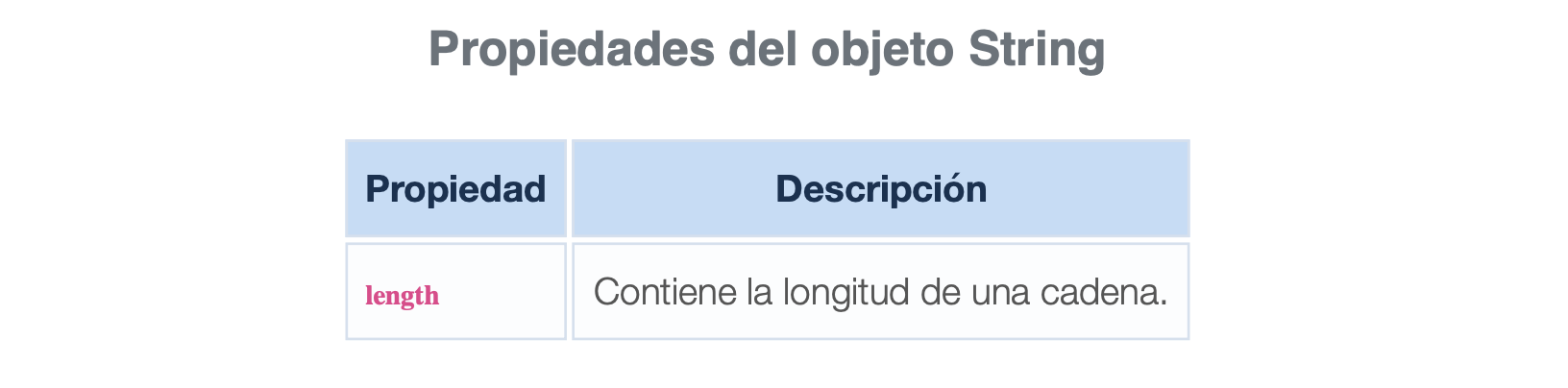
O también se podría hacer:

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:

cadena.propiedad;

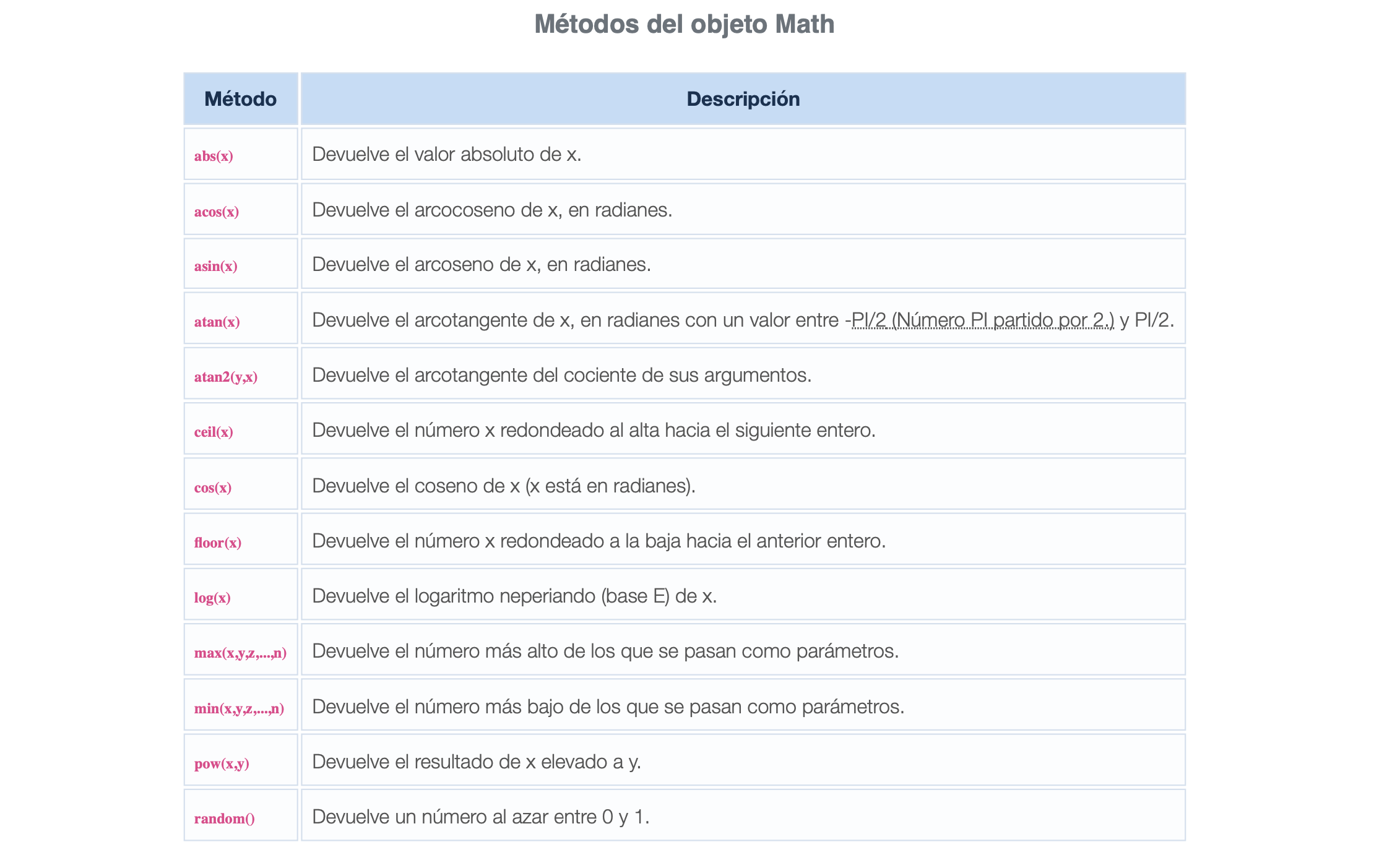
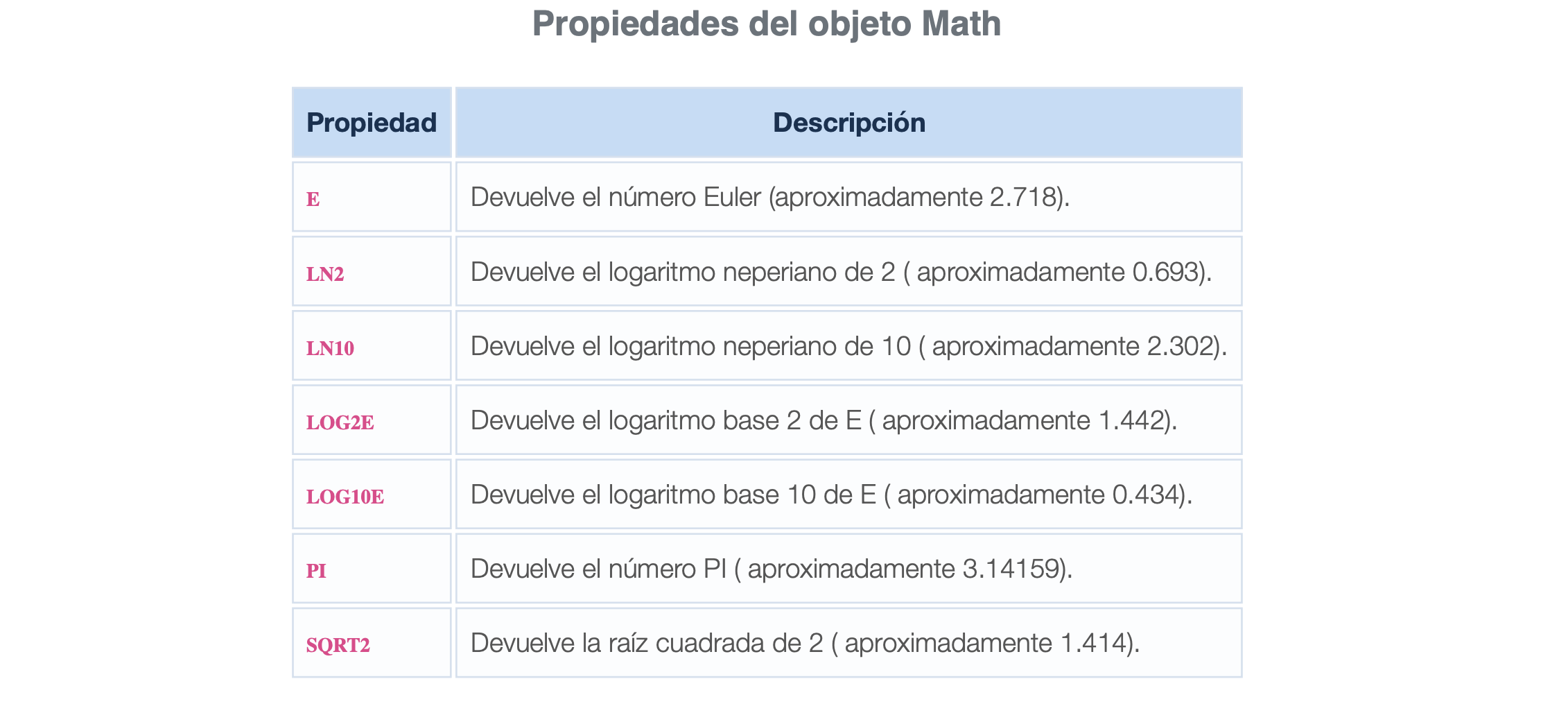
cadena.metodo( [parámetros] );



Más información: <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String>

## 3.2.- Objeto Math.

Disponemos de un objeto Math en JavaScript, que nos permite realizar operaciones matemáticas. El objeto **Math** no es 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.



Más información: <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Math>

## 3.3.- Objeto Number.

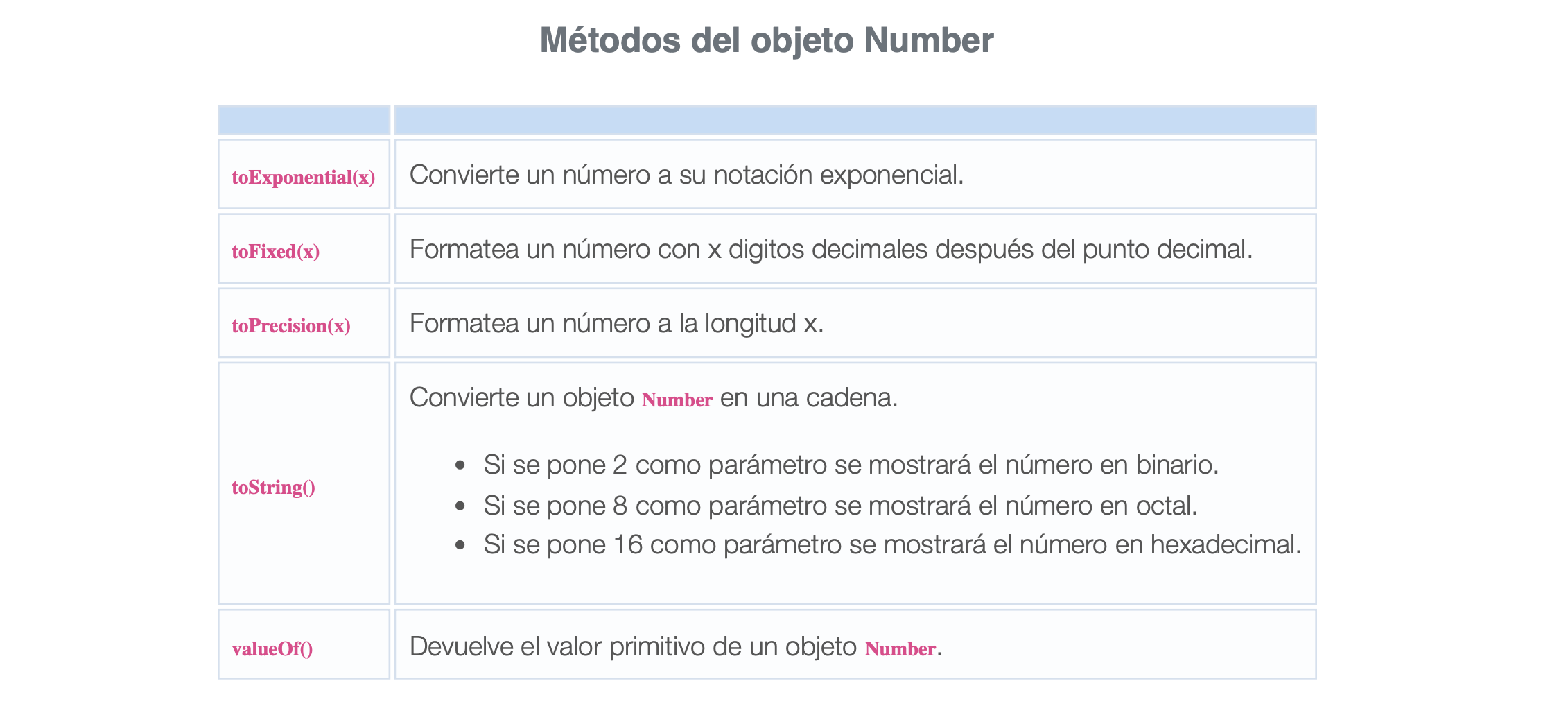
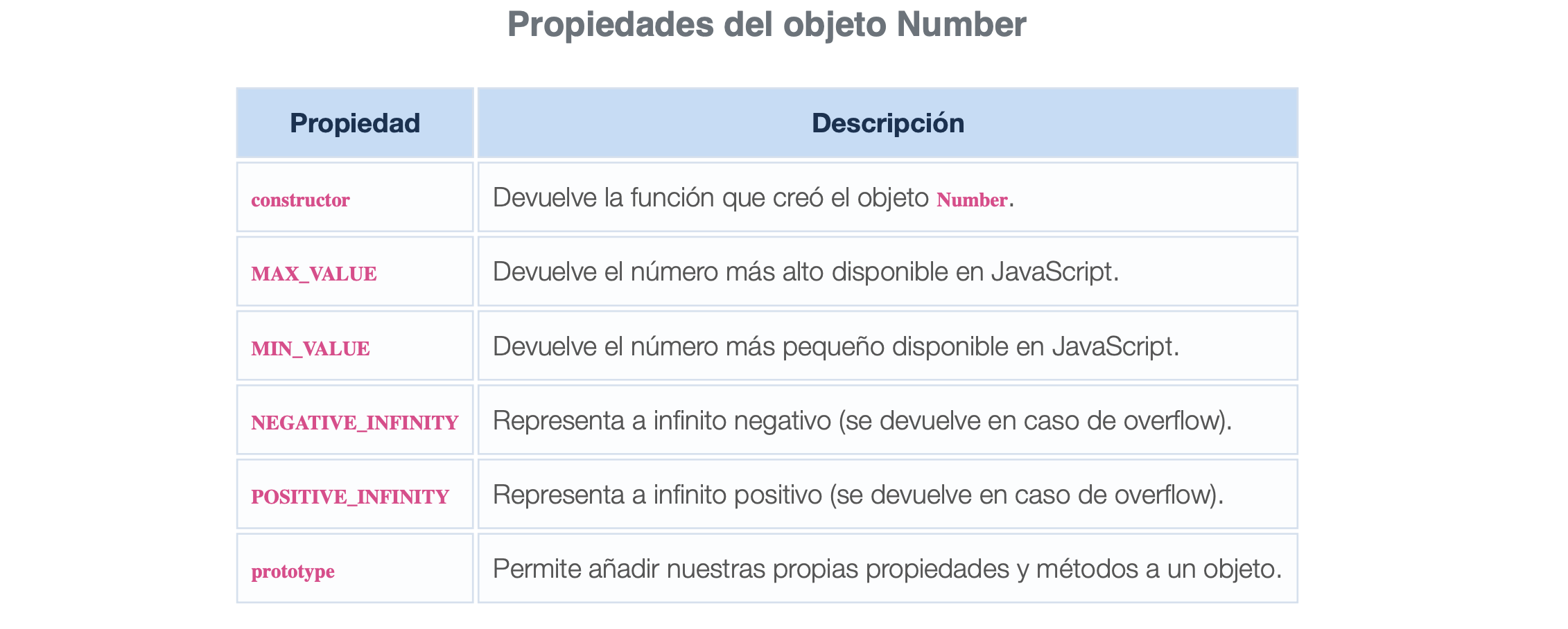
El objeto **Number** se usa muy raramente, ya que para la mayor parte de los casos, JavaScript satisface las necesidades del día a día con los valores numéricos que almacenamos en variables.

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.**

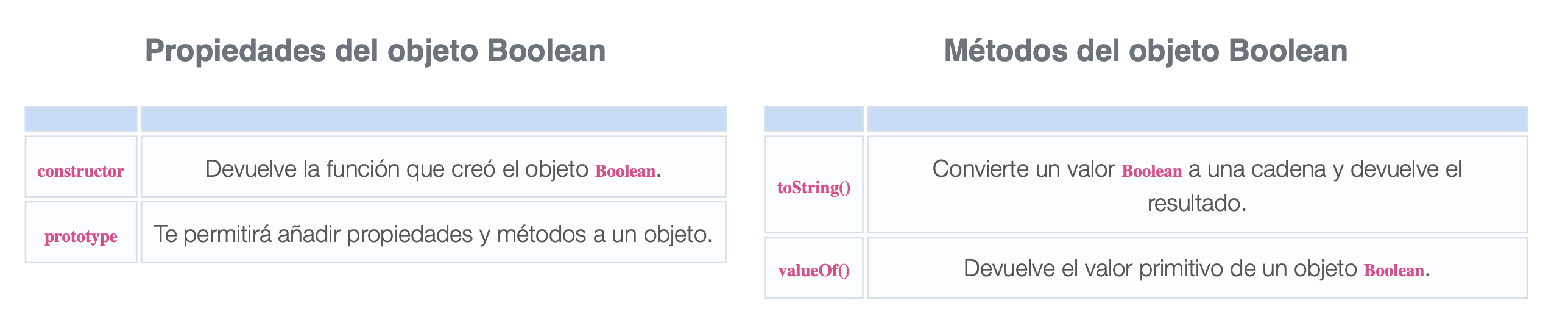
El objeto **Number**, es un objeto envoltorio para valores numéricos primitivos.

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



## 3.4.- Objeto Boolean.

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



## 3.5.- 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):

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.)

