# RESUMEN TEMA 4.

## 4.A. Estructuras definidas por el usuario en JavaScript

## 1.- Estructuras de datos.

En los lenguajes de programación existen **estructuras de datos** especiales, que nos sirven para guardar información más compleja que una variable simple.

**Tipos** de estructuras de datos (listas, pilas, colas, árboles, conjuntos,…), que se pueden utilizar para almacenar datos, pero una estructura de las más utilizadas en todos los lenguajes es el **array**.

El **array**, es como una variable o **zona de almacenamiento continuo**, donde podemos introducir varios valores en lugar de solamente uno, como ocurre con las variables normales.

Los **arrays** se suelen denominar matrices o vectores. Desde el punto de vista lógico, una matriz se puede ver como un conjunto de elementos ordenados en fila (o filas y columnas si tuviera dos o más dimensiones).

Se puede considerar que todos los arrays son de una dimensión: la dimensión principal, pero los elementos de dicha fila pueden a su vez **contener** otros arrays o matrices, lo que nos permitiría hablar de **arrays multidimensionales** (los más fáciles de imaginar son los de una, dos y tres dimensiones).

Los **arrays** son una estructura de datos, adecuada para situaciones en las que el **acceso** a los datos se realiza de forma **aleatoria** e **impredecible**.

Por el contrario, si los elementos pueden estar ordenados y se va a utilizar **acceso secuencial** sería más adecuado usar una **lista**, ya que esta estructura puede **cambiar de tamaño** fácilmente **durante la ejecución** de un programa.

Los **arrays** nos permiten **guardar un montón de elementos** y **acceder** a ellos de manera **independiente**.

Cada elemento es **referenciado** por la **posición** que ocupa dentro del array. Dichas posiciones se llaman **índices** y siempre son **correlativos**.

Existen **tres formas** de indexar los elementos de un array:

* **indexación base-cero(0):** en este modo, el primer elemento del array será la componente 0, es decir tendrá el **índice 0**.
* **indexación base-uno(1):** en este modo, el **primer** elemento tiene el índice **1**.
* **indexación base-n(n):** este modo, es un modo versátil de indexación, en el que el índice del **primer elemento** puede ser elegido **libremente**.

En **JavaScript** cuando trabajamos con índices numéricos utilizaremos la indexación base-cero ( 0 ).

El objeto **Array** descrito se denomina genéricamente **array escalar** debido a que sus elementos se **referencian** utilizando un **número**.

El **array asociativo** (también llamado mapa, diccionario, hash, etc). Cada elemento es referenciado utilizando **claves** que son cadenas de caracteres. Es un **conjunto de pares clave-valor** y así se pueden referenciar valores utilizando cualquier tipo de clave ya que las claves ...

Una **diferencia** importante entre ambos tipos de arrays es que los **escalares** se pueden **ordenar** por el **índice** (ya que este es numérico) en cambio los **asociativos NO tienen orden**.

Este tipo de array se corresponde con el objeto **Object** que, como veremos, se utilizar para crear objetos definidos por el usuario. Así, el nombre de la **propiedad o método** de un objeto definido por el usuario se corresponde con **clave del elemento y el valor de esa propiedad** o el código de ese método se corresponde con el valor del elemento.

## 1.- Estructuras de datos

## 1.1.- Objeto Array

Array escalares, aquellos cuyo índice es numérico. Un array escalar es una estructura de datos proporcionadas para almacenar y manipular colecciones de datos. Los arrays en JavaScript son muy versátiles.

En programación, un array se define como una colección ordenada de datos.

JavaScript emplea un montón de arrays internamente para gestionar los objetos HTML en el documento, propiedades del navegador, etc. Ej: document.links[0]

Dentro del objeto **document** tenemos colecciones (anchors[], forms[], links[], images[]) será un array que contendrá las referencias de todas las anclas, formularios, enlaces e imágenes del documento.

Para crear un objeto array, usaremos el constructor new Array( ). Por ejemplo:

var miarray= new Array();

**Creación de un array.**

Los objetos de tipo array disponen de una propiedad que nos indica su longitud (length). Si queremos precisar el tamaño del array durante la inicialización, podríamos hacerlo pasaándole un parámetro al constructor.

var personas = new Array(40);

La propiedad **length** se ajustará automáticamente al nuevo tamaño del array.

personas[53] = "Irene Sáinz Veiga";

longitud = personas.length; // asignará a la variable longitud el valor 54

**Introducir datos en un array**

Introducir datos en un array, es tan simple como crear una serie de sentencias de asignación, una por cada elemento del array.

sistemaSolar = new Array( );

sistemaSolar[0] = "Mercurio";

sistemaSolar[1] = "Venus";

Esta forma tediosa, pero una vez que las posiciones del array están cubiertas con los datos, acceder a esa información nos resultará muy fácil.

unPlaneta = sistemaSolar[2]; // almacenará en unPlaneta la cadena "Tierra".

Mediante el constructor es otra forma de crear un array. En lugar de escribir cada sentencia de asignación para cada elemento, lo podemos hacer creando lo que se denomina un **“array denso”,** aportando al constructor **array(),** los datos a cubrir separados por comas:

sistemaSolar = new array

("Mercurio","Venus","Tierra","Marte","Jupiter","Saturno","Urano","Neptuno");

Los corchetes sustituyen a la llamada al constructor **new Array().** Hay que tener cuidado con esta forma de creación que quizás no funcione en navegadores antiguos.

Otra forma, **array mixto (objeto literal)**, comenzando y terminando con **llaves {}**.

var datos = { "numero": 42, "mes" : "Junio", "hola" : "mundo", 69 : "96" };

**Recorrido de un array**

Empleando un bucle **for**.

Empleando un bucle **while**.

Empleando la sentencia **for each in** (JS 1.6 o superior)

**Borrado de elementos de un array**

Operador **delete**. Al borrar un elemento del array, se eliminará su **índice** en la lista de índices del array, pero **no se reducirá la longitud** del **array**.

El proceso de borrar una entrada del array **no libera** la memoria ocupada por esos datos necesariamente. El intérprete de JavaScript, encargado de gestionar la colección de basura en memoria, se encargará de **liberar** memoria ocupada cuando la necesite.

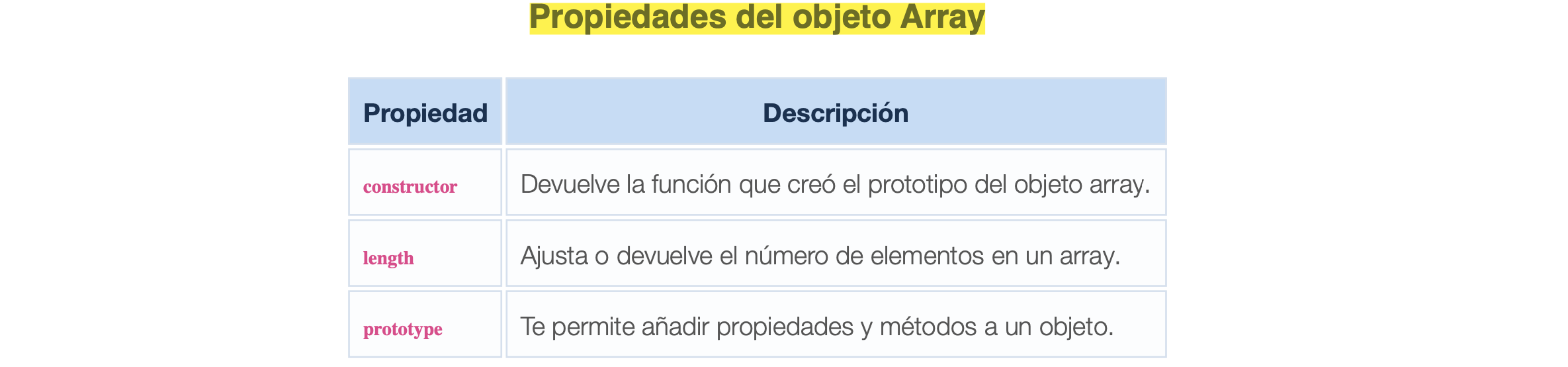
Si se quiere tener un mayor control sobre la eliminación de elementos de un array, deberías considerar usar el método **splice**(índice, número de elementos a eliminar), que está soportado por la mayoría de los navegadores. Este método se puede usar en cualquier array, y te permite eliminar un elemento o una secuencia de elementos de un array, provocando que la longitud del **array se ajuste** al nuevo número de elementos.

delete oceanos[2];

oceanos.splice(2,1); // las posiciones del array resultante serán 0, 1 y 2.

El operador **delete**, se recomienda para arrays que usen texto como índices del array, ya que de esta forma se producirán menos confusiones a la hora de borrar los elementos.

Propiedades y métodos



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

## 1.2.- Arrays paralelos.

Cuando tenemos dos o más arrays, que utilizan el mismo índice para referirse a términos homólogos,se denominan **arrays paralelos.**

Para que los arrays paralelos sean homogéneos, éstos tendrán que tener la misma longitud, ya que de esta forma se mantendrá la consistencia de la estructura lógica creada.

Entre las ventajas del uso de **arrays paralelos** tenemos:

* Se pueden usar en lenguajes que soporten solamente arrays, como tipos primitivos y no registros (como puede ser JavaScript).
* Son fáciles de entender y utilizar.
* Pueden ahorrar una gran cantidad de espacio, en algunos casos evitando complicaciones de sincronización.
* El recorrido secuencial de cada posición del array, es extremadamente rápido en las máquinas actuales.

## 1.3.- Arrays multidimensionales.

Una alternativa a los arrays paralelos es la simulación de un array multidimensional.

var datos = [["Cristina","Seguridad",24],["Catalina","Bases de Datos",17],["Vieites","Sistemas Informáticos",28],["Benjamin","Redes",26]];

Para acceder a un dato en particular, de un array de arrays, se requiere que hagamos una doble referencia. La primera referencia, será a unaposición del array principal, y la segunda referencia, a una posición del array almacenado dentro de esa casilla del array principal.

document.write("<br/>Quien imparte Bases de Datos? "+datos[1][0]); // Catalina

Si queremos **imprimir** toda la información del array **multidimensional**, tal y como hicimos en el apartado anterior podríamos hacerlo con un bucle **for**.

## 2.- Creación de funciones.

Una función es la definición de un conjunto de acciones pre-programadas. Las funciones se llaman a través de eventos o bien mediante comandos desde nuestro script.

Las funciones son una herramienta muy potente en los lenguajes de programación, ya que le van a permitir realizar tareas de una manera mucho más organizada, y además le permitirán reutilizar un montón de código en sus aplicaciones.

Siempre que sea posible, tienes que diseñar funciones que puedas reutilizar en otras aplicaciones, de esta forma, tus funciones se convertirán en pequeños bloques constructivos que te permitirán ir más rápido en el desarrollo de nuevos programas.

En **JavaScript** no vamos a distinguir entre **procedimientos** (que ejecutan acciones), o **funciones** (que ejecutan acciones y devuelven valores). En JavaScript siempre se llamarán funciones.

Una función es capaz de devolver un valor a la instrucción que la invocó, pero esto no es un requisito obligatorio en JavaScript. Cuando una función devuelve un valor, la instrucción que llamó a esa función, la tratará como si fuera una expresión.

Si nuestra función va a devolver algún valor emplearemos la palabra reservada **return**, para hacerlo.

Los nombres que puedes asignar a una función, tendrán las mismas restricciones que tienen los elementos HTML y las variables en JavaScript.

Puedes usar palabras compuestas como chequearMail o calcularFecha, y fíjate que las funciones suelen llevar un verbo.

Una recomendación que te hacemos, es la de que las funciones sean muy específicas, es decir que no realicen tareas adicionales a las inicialmente propuestas en esa función.

Para realizar una llamada a una función lo podemos hacer con:

nombreFuncion( ); // Esta llamada ejecutaría las sentencias programadas dentro de la función.

Otro ejemplo de uso de una función en una asignación:

variable=nombreFuncion( ); // En este caso la función devolvería un valor que se asigna a la variable.

Las funciones en JavaScript también son objetos, y como tal tienen **métodos y propiedades**. Un método, aplicable a cualquier función puede ser toString(), el cuál nos devolverá el código fuente de esa función.

## 2.1.- Parámetros.

Cuando se realiza una llamada a una función, muchas veces es necesario pasar parámetros (también conocidos como argumentos). Este mecanismo nos va a permitir enviar datos entre instrucciones.

Para pasar parámetros a una función, tendremos que escribir dichos parámetros entre paréntesis y separados por comas.

A la hora de definir una función que recibe parámetros, lo que haremos es, escribir los nombres de las variables que recibirán esos parámetros entre los paréntesis de la función.

Veamos el siguiente ejemplo:

function saludar(a,b) {

alert("Hola " + a + " y "+ b +".");

}

Si llamamos a esa función desde el código:

saludar("Martin","Silvia"); // Mostraría una alerta con el texto: Hola Martin y Silvia.

Los parámetros que usamos en la definición de la función a y b, no usan la palabra reservada var para inicializar dichas variables. Esos parámetros a y b serán variables locales a la función , y se inicializarán automáticamente en el momento de llamar a la función, con los valores que le pasemos en la llamada.

Más información: <https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Functions>

## 2.2.- Ámbito de las variables.

Las variables que se definen fuera de las funciones se llaman variables globales.

Las variables que se definen dentro de las funciones, con la palabra reservada **var**, se llaman variables locales.

Una variable global en JavaScript tiene una connotación un poco diferente, comparando con otros lenguajes de programación.

Para un script de JavaScript, el alcance de una variable global, se limita al documento actual que está cargado en la ventana del navegador o en un frame. Sin embargo cuando inicializas una variable como variable global, quiere decir que todas las instrucciones de tu script (incluidas las instrucciones que están dentro de las funciones), tendrán acceso directo al valor de esa variable. Todas las instrucciones podrán leer y modificar el valor de esa variable global.

En el momento que una página se cierra, todas las variables definidas en esa página se eliminarán de la memoria para siempre. Si necesitas que el valor de una variable persista de una página a otra, tendrás que utilizar técnicas que te permitan almacenar esa variable (como las cookies, o bien poner esa variable en el documento frameset, etc.).

Aunque el uso de la palabra reservada var, para inicializar variables es opcional.

Una variable local será definida dentro de una función. Antes viste que podemos definir variables en los parámetros de una función (sin usar **var**), pero también podrás definir nuevas variables dentro del código de la función. En este caso, **si que se requiere** el uso de la palabra reservada **var** cuando definimos una variable local, ya que de otro modo, esta variable será reconocida como una variable global.

El alcance de una variable local está solamente dentro del ámbito de la función. Ninguna otra función o instrucciones fuera de la función podrán acceder al valor de esa variable.

Reutilizar el nombre de una variable global como local es uno de los **bugs** más sutiles y por consiguiente más difíciles de encontrar en el código de JavaScript. La variable local en momentos puntuales ocultará el valor de la variable global, sin avisarnos de ello. Como recomendación, no reutilices un nombre de variable global como local en una función, y tampoco declares una variable global dentro de una función, ya que podrás crear fallos que te resultarán difíciles de solucionar.

## 2.3.- Funciones anidadas.

Los navegadores más modernos nos proporcionan la opción de anidar unas funciones dentro de otras. Es decir podemos programar una función dentro de otra función.

Cuando no tenemos funciones anidadas, cada función que definamos será accesible por todo el código, es decir serán funciones globales. Con las funciones anidadas, podemos encapsular la accesibilidad de una función dentro de otra y hacer que esa función sea privada o local a la función principal.

function principalA() {

// Sentencias

function internaA1() {

// Sentencias

}

// Sentencias

}

Una buena opción para aplicar las funciones anidadas, es cuando tenemos una secuencia de instrucciones que necesitan ser llamadas desde múltiples sitios dentro de una función, y esas instrucciones sólo tienen significado dentro del contexto de esa función principal. En otras palabras, en lugar de romper la secuencia de una función muy larga en varias funciones globales, haremos lo mismo pero utilizando funciones locales.

function hipotenusa(a, b) {

function cuadrado(x) {

return x\*x;

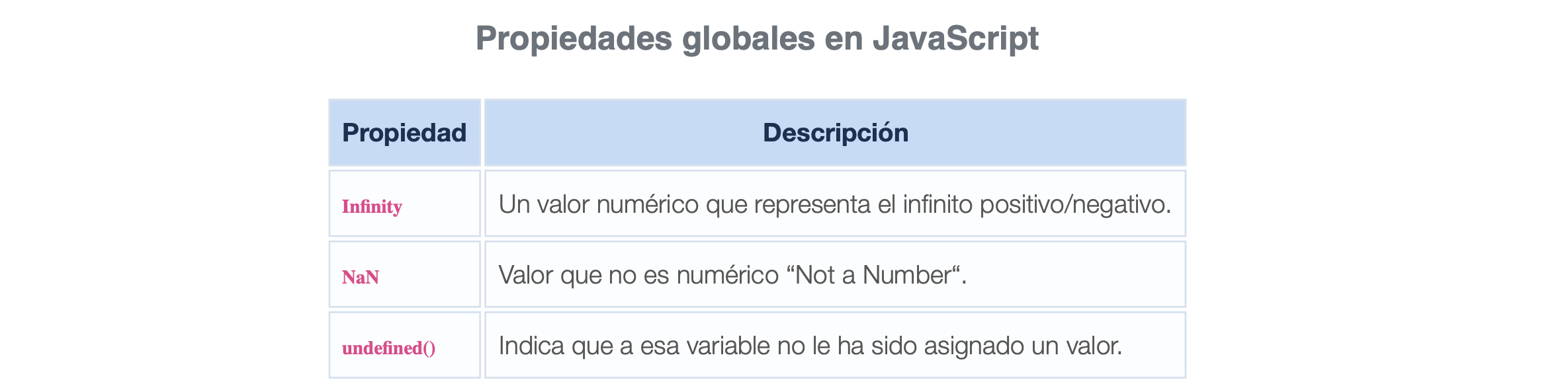
}

return Math.sqrt(cuadrado(a) + cuadrado(b));

}

## 2.4.- Funciones predefinidas del lenguaje.

Si te das cuenta todos los métodos en realidad son funciones (llevan siempre paréntesis con o sin parámetros). Pues bien en JavaScript, disponemos de algunos elementos que necesitan ser tratados a escala global y que no pertenecen a ningún objeto en particular (o que se pueden aplicar a cualquier objeto).



3.- Creación de objetos definidos por el usuario.

JavaScript también te da la oportunidad de crear tus propios objetos en memoria, objetos con propiedades y métodos que tú puedes definir a tu antojo. Estos objetos no serán elementos de la página de interfaz de usuario, pero sí que serán objetos que podrán contener datos (propiedades) y funciones (métodos), cuyos resultados si que se podrán mostrar en el navegador. El definir tus propios objetos, te permitirá enlazar a cualquier número de propiedades o métodos que tú hayas creado para ese objeto. Es decir, tú controlarás la estructura del objeto, sus datos y su comportamiento.

También hay que dejar claro que, JavaScript no es un lenguaje orientado a objetos de verdad en sentido estricto. Se considera que, JavaScript es un lenguaje basado en objetos (u orientado a prototipos). La diferencia entre orientado a objetos y basado en objetos es significante, y tiene que ver sobre todo en cómo los objetos se pueden extender.

Un objeto en JavaScript es realmente una colección de propiedades almacenadas en un array asociativo. Las propiedades pueden tener forma de datos, tipos, funciones (métodos) o incluso otros objetos. De hecho sería más fácil de entender un objeto como un array de valores, cada uno de los cuales está asociado a una propiedad (un tipo de datos, método u objeto).

Una función contenida en un objeto se conoce como un método. Los métodos no son diferentes de las funciones que has visto anteriormente, excepto que han sido diseñados para ser utilizados en el contexto de un objeto, y por lo tanto, tendrán acceso a las propiedades de ese objeto. Esta conexión entre propiedades y métodos es uno de los ejes centrales de la orientación a objetos.

Los objetos se crean empleando una función especial denominada constructor, determinada por el nombre del objeto. Ejemplo de una función constructor:

function Coche( ) {

// propiedades y métodos

}

Aunque esta función no contiene código, es sin embargo la base para crear objetos de tipo Coche. Puedes pensar en un constructor como un anteproyecto o **plantilla**, que será utilizada para crear objetos. Por convención, los nombres de los constructores se ponen generalmente con las iniciales de cada palabra en **mayúscula**, y cuando creamos un objeto con ese constructor (instancia de ese objeto), lo haremos empleando minúsculas al principio. Por ejemplo: var unCoche = new Coche( );

La palabra reservada **new** se emplea para crear objetos en JavaScript. Al crear la variable unCoche, técnicamente podríamos decir que hemos creado una instancia de la clase Coche, o que hemos instanciado el objeto Coche, o que hemos creado un objeto Coche, etc. Es decir hay varias formas de expresarlo, pero todas quieren decir lo mismo.

A partir del estándar ECMAScript 2015 aparece la sintaxis de clases que permiten definir objetos utilizando expresiones más apropiadas de un lenguaje orientado a objetos (basado en clases).

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

## 3.1.- Definición de propiedades.

Las propiedades para nuestro objeto se crearán dentro del constructor empleando para ello la palabra reservada **this**.

function Coche( ) {

// Propiedades

this.marca = "Audi A6";

this.combustible = "diesel";

this.cantidad = 0; // Cantidad de combustible en el depósito.

}

La palabra reservada this, se utiliza para hacer referencia al objeto actual, que en este caso será el objeto que está siendo creado por el constructor. Por lo tanto, usarás this, para crear nuevas propiedades para el objeto. El único problema con el ejemplo anterior es, que todos los coches que hagamos del tipo Coche será siempre Audi A6, diésel, y sin litros de combustible en el depósito. Por ejemplo;

var cocheDeMartin = new Coche( );

var cocheDeSilvia = new Coche( );

A partir de ahora, si no modificamos las propiedades del coche de Martin y de Silvia, en el momento de instanciarlos tendrán ambos un Audi A6 a diésel y sin combustible en el depósito.

Lo ideal sería por lo tanto que en el momento de instanciar un objeto de tipo Coche, que le digamos al menos la marca de coche y el tipo de combustible que utiliza. Para ello tenemos que modificar el constructor, que quedará de la siguiente forma:

function Coche(marca, combustible) {

// Propiedades

this.marca = marca;

this.combustible = combustible;

this.cantidad = 0; // Cantidad de combustible inicial por defecto en el depósito.

}

## 3.2.- Definición de métodos.

Las propiedades son solamente la mitad de la ecuación de la Orientación a Objetos en JavaScript. La otra mitad son los métodos, que serán funciones que se enlazarán a los objetos, para que dichas funciones puedan acceder a las propiedades de los mismos.

function rellenarDeposito (litros) {

// Modificamos el valor de la propiedad cantidad de combustible

this.cantidad = litros;

}

Fíjate que el método rellenarDeposito, que estamos programando a nivel global, hace referencia a la propiedad **this**.cantidad para indicar cuantos litros de combustible le vamos a echar al coche. Lo único que faltaría aquí es realizar la conexión entre el método rellenarDeposito y el objeto de tipo Coche (recuerda que los objetos podrán tener propiedades y métodos y hasta este momento sólo hemos definido propiedades dentro del constructor). Sin esta conexión la palabra reservada this no tiene sentido en esa función, ya que no sabría cuál es el objeto actual.

function Coche(marca, combustible) {

// Propiedades

this.marca = marca;

this.combustible = combustible;

this.cantidad = 0; // Cantidad de combustible inicial por defecto en el depósito.

// Métodos

this.rellenarDeposito = rellenarDeposito;

}

Aquí se ve de forma ilustrada, que los métodos son en realidad propiedades: se declaran igual que las propiedades, por lo que son enmascarados como propiedades en JavaScript. Hemos creado una nueva propiedad llamada rellenarDeposito y se le ha asociado el método rellenarDeposito. Es muy importante destacar que el método rellenarDeposito() se referencia sin paréntesis dentro del constructor, this.rellenarDeposito = rellenarDeposito.

La forma en la que hemos definido el método rellenarDeposito a nivel global, no es la mejor práctica en la programación orientada a objetos. Una mejor aproximación sería definir el contenido de la función rellenarDeposito dentro del constructor, ya que de esta forma los métodos al estar programados a nivel local aportan mayor privacidad y seguridad al objeto en general, por ejemplo:

function Coche(marca, combustible) {

// Propiedades

this.marca = marca;

this.combustible = combustible;

this.cantidad = 0;

// Métodos

this.rellenarDeposito = function (litros) {

this.cantidad=litros;

};

}

Una aproximación aún mejor es utilizar la característica del lenguaje JavaScript de ser un lenguaje orientado a prototipos. Un prototipo en JavaScript es una propiedad que tienen todas las funciones para almacenar propiedades que se asignan a un objeto cuando este es creado con el operador new. De este modo el prototipo almacena propiedades (y por tanto métodos) que, al pertenecer al prototipo, pertenecen por así decirlo a todos los objetos que comparten ese prototipo.

En nuestro ejemplo, al asignar el método rellenarDeposito al prototipo del constructor Coche en lugar de asignarlo al objeto, todos los objetos creados con el constructor, reciben el mismo método rellenarDeposito al contrario que los ejemplos anteriores donde cada objeto Coche tiene su propio método rellenarDeposito. Esto tiene dos implicaciones importantes:

No se duplica código. Esto es, el mismo método se utiliza para todos los objetos con el mismo prototipo Permite implementar la herencia de propiedades, si una propiedad no se encuentra en un objeto, ésta se busca en su prototipo y si no se encuentra, se busca en el prototipo de su prototipo y así sucesivamente hasta terminar de recorrer la cadena de prototipos.

function Coche(marca, combustible) {

// Propiedades

this.marca = marca;

this.combustible = combustible;

this.cantidad = 0;

}

// Métodos

Coche.prototype.rellenarDeposito = function (litros) {

this.cantidad=litros;

};

## 3.3.- Definición de objetos literales.

Otra forma de definir objetos es hacerlo de forma literal definiendo un array asociativo.

Un literal es un valor fijo que se especifica en JavaScript. Un objeto literal será un conjunto, de cero o más parejas del tipo

nombre:valor.

Por ejemplo:

avion = { marca:"Boeing" , modelo:"747" , pasajeros:"450" };

Es equivalente a:

var avion = new Object();

avion.marca = "Boeing";

avion.modelo = "747";

avion.pasajeros = "450";

Para referirnos desde JavaScript a una propiedad del objeto avión podríamos hacerlo con:

document.write(avion.marca); // o txambién se podría hacer con:

document.write(avion["modelo"]);

Objetos:

https://www.w3schools.com/js/js\_object\_definition.asp