

JavaScript. Nivel III

agosto, 2019



Objetivos del nivel

- Aprender a crear objetos.
- Aprender a usar JQuery
- Aprender a usar la notación JSON

Prerrequisitos del nivel

- JavaScript Nivel II

Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADI F1). Para obtener más información sobre este u otros cursos visite nuestro sitio Web www.cadif1.com, escribanos a la dirección de correo cadi@cadif1.com o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños.
Copyright 2019. Todos los derechos reservados.



Contenido del nivel

Capítulo 1. Objetos en Java Script

- 1.1.- Introducción.
- 1.2.- Acceso a Los Atributos.
- 1.3.- Manipulando Objetos.

Capítulo 2. Constructores

- 2.1.- Constructores.
- 2.2.- Constructores Con Parámetros.
- 2.3.- Arreglos de Objetos.

Capítulo 3. Clases

- 3.1.- Definición de Clases.
- 3.2.- Constructores.
- 3.3.- Instanciación.

Capítulo 4. Métodos

- 4.1.- Definición.
- 4.2.- Ejecución.
- 4.3.- Parámetros y Valor de Retorno.

Capítulo 5. Json. Parte 1

- 5.1.- Objetos Literales.
- 5.2.- Objetos Anidados.
- 5.3.- Archivos Json.

Capítulo 6. Json. Parte 2

- 6.1.- Stringify.
- 6.2.- Parse.

Capítulo 7. Almacenamiento Local. Parte 1

- 7.1.- Introducción.
- 7.2.- Opciones de Almacenamiento Local.
- 7.3.- Usando Cookies.

Capítulo 8. Almacenamiento Local. Parte 2

- 8.1.- Localstorage vs Sessionstorage.
- 8.2.- Manipular Datos en el Localstorage.
- 8.3.- Manipular Datos en el Sessionstorage.

Capítulo 9. Introducción jQuery

- 9.1.- ¿Qué es jQuery?.
- 9.2.- Insertar Jquery en la Página Html.
- 9.3.- La Función Jquery().

Capítulo 10. Usando Jquery

- 10.1.- Uso de Selectores.
- 10.2.- Eventos.

Capítulo 11. Manipular el Html Con Jquery

- 11.1.- Get.
- 11.2.- Set.
- 11.3.- Add.
- 11.4.- Remove.

Capítulo 12. Manipular Estilos Con Jquery

- 12.1.- Manipular Estilos Css.
- 12.2.- Asignar Clases de Css.
- 12.3.- Manipular Tamaños.

Capítulo 13. Efectos Con Jquery. Parte 1

- 13.1.- Fade.
- 13.2.- Slide.
- 13.3.- Animate.

Capítulo 14. Efectos Con Jquery. Parte 2

- 14.1.- Callback.
- 14.2.- Encadenamiento.
- 14.3.- Stop.

Capítulo 15. Manejo de Fechas

- 15.1.- Date.



Capítulo 1. OBJETOS EN JAVA SCRIPT

1.1.- Introducción

En JavaScript casi todo se maneja como un objeto, inclusive, hasta ahora se han manejado varios de ellos durante el curso:

- Números (la clase Number)
- Arreglos (la clase Array)
- Strings (la clase String)
- Funciones matemáticas (clase Math)
- Expresiones regulares
- Window
- Document
- Form

Por ejemplo, al necesitar conocer el tamaño de un arreglo se utiliza el atributo "length" o para agregar un elemento al final de un arreglo se utiliza el método "push". Muchos objetos son creados automáticamente al ejecutar una página, pero otros pueden ser creados a discreción por el programador. Se asumirá que el público lector de este material tiene conocimientos de programación orientada a objetos.

JavaScript provee varios mecanismos para crear objetos:

- instanciando la clase Object.
- usando funciones constructoras.
- definiendo clases.
- definiendo objetos literales.

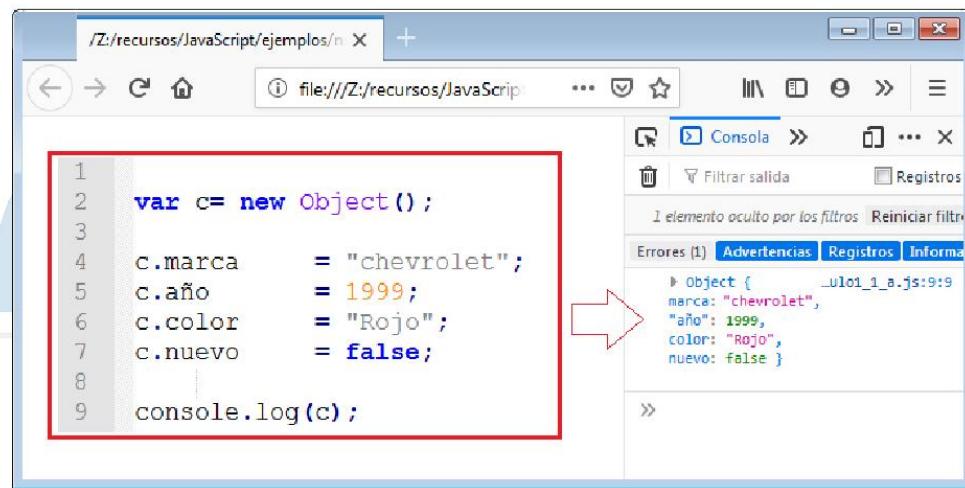
La primera forma de crear objetos es instanciando (creando un objeto de una clase) la clase Object, que es parte de JavaScript. Por ejemplo:

```
var x = new Object();
```

En el ejemplo anterior, la variable "x" toma el valor que retorna la ejecución de la función "Object" que inicializa a un nuevo objeto creado con la instrucción "new". La clase Object posee métodos pre definidos que pudieran usarse eventualmente.

1.2.- Acceso a Los Atributos

Al nuevo objeto que esta naciendo se le pueden asignar nuevos atributos con sus respectivos valores, por ejemplo, usando la variable objeto y el operador "." (punto), seguido del nombre del atributo, el operador "=" seguido del valor. En el siguiente ejemplo se crea un objeto y se le agregan 4 atributos, para luego mostrar el objeto completo en la consola:



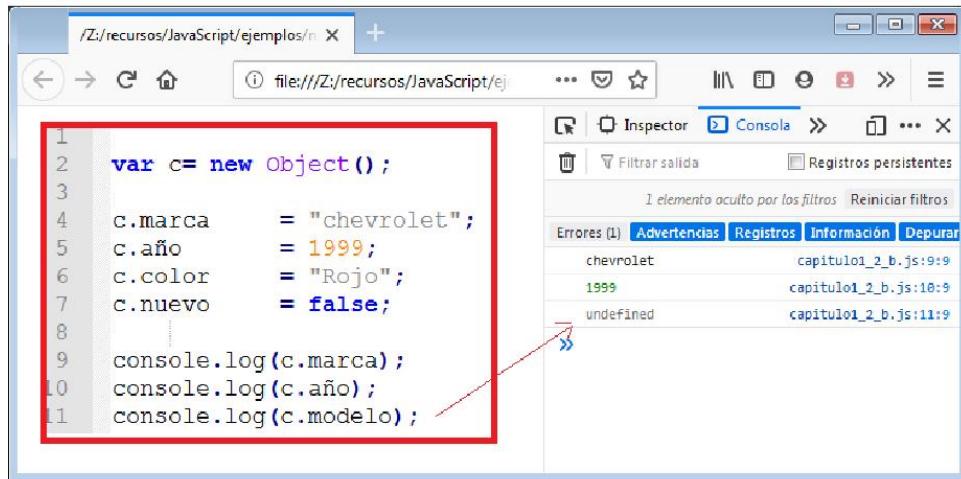
The screenshot shows a browser window with developer tools open. The left pane contains a code editor with the following JavaScript code:

```
1 var c = new Object();
2
3 c.marca = "chevrolet";
4 c.año = 1999;
5 c.color = "Rojo";
6 c.nuevo = false;
7
8 console.log(c);
```

A red box highlights the code from line 1 to line 8. An arrow points from the right side of the code editor to the right pane, which is a developer tools console. The console displays the output of the `console.log` statement:

```
Object { marca: "chevrolet", año: 1999, color: "Rojo", nuevo: false }
```

En el ejemplo anterior, se crearon nuevos atributos. También se puede usar el operador punto para acceder al valor de los atributos que ya posee el objeto. Si el atributo no existe, retornará "undefined". Por ejemplo:



A screenshot of a browser's developer tools showing the 'Console' tab. On the left, the code editor displays the following JavaScript:

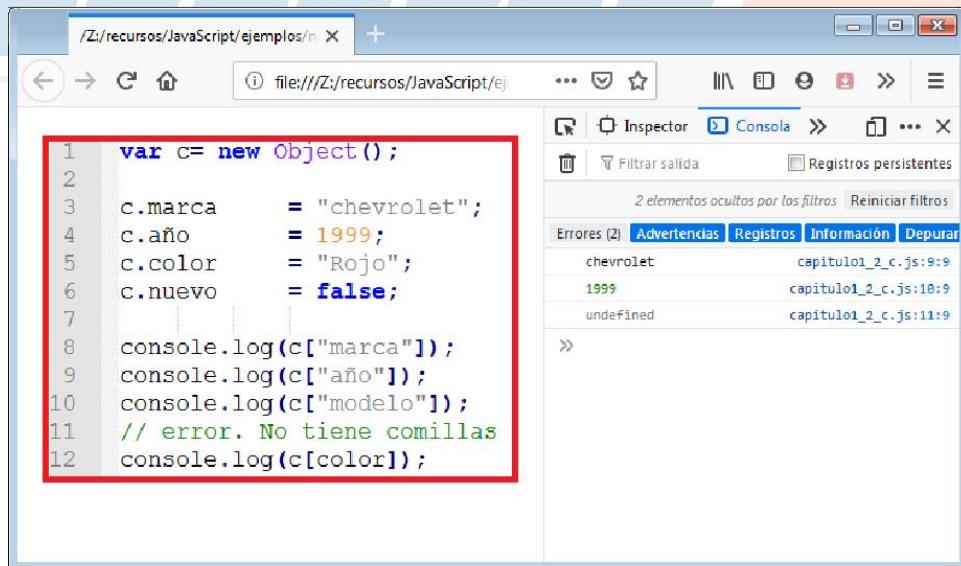
```

1 var c= new Object();
2
3 c.marca      = "chevrolet";
4 c.año        = 1999;
5 c.color       = "Rojo";
6 c.nuevo       = false;
7
8 console.log(c.marca);
9 console.log(c.año);
10 console.log(c.modelo);
11

```

The code uses dot notation to access object properties. The output in the console shows three entries: 'chevrolet', '1999', and 'undefined'. A red box highlights the code from line 1 to line 11. A red arrow points from the word 'modelo' in line 11 to the 'undefined' entry in the console output.

Otra forma de acceder a los atributos de un objeto es con la notación de arreglos (usando corchetes), asumiendo que el objeto es un arreglo y que cada posición de este arreglo es un atributo del objeto, al cual se accede usando el nombre del atributo. Por ejemplo:



A screenshot of a browser's developer tools showing the 'Console' tab. On the left, the code editor displays the following JavaScript:

```

1 var c= new Object();
2
3 c.marca      = "chevrolet";
4 c.año        = 1999;
5 c.color       = "Rojo";
6 c.nuevo       = false;
7
8 console.log(c["marca"]);
9 console.log(c["año"]);
10 console.log(c["modelo"]);
11 // error. No tiene comillas
12 console.log(c[color]);

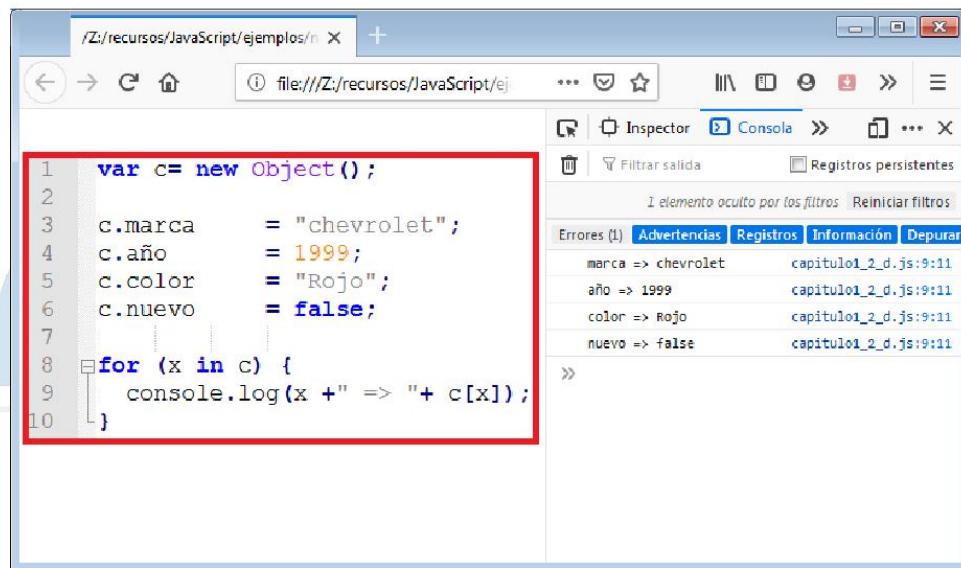
```

The code uses bracket notation to access object properties. The output in the console shows two entries: 'chevrolet' and '1999'. A red box highlights the code from line 1 to line 12. A red arrow points from the word 'color' in line 12 to the 'undefined' entry in the console output.

Se puede acceder a cada uno de los atributos de un objeto usando la instrucción "for...in", de la forma general:

```
for (variable in objeto) {
    // se hace algo con el atributo
}
```

Donde "variable" almacenará el nombre de cada uno de los atributos y "objeto" es el objeto que las contiene.



The screenshot shows a browser window with developer tools open. The code in the console is:

```
1 var c = new Object();
2
3 c.marca = "chevrolet";
4 c.año = 1999;
5 c.color = "Rojo";
6 c.nuevo = false;
7
8 for (x in c) {
9     console.log(x + " => " + c[x]);
10 }
```

The output in the console shows the properties and their values:

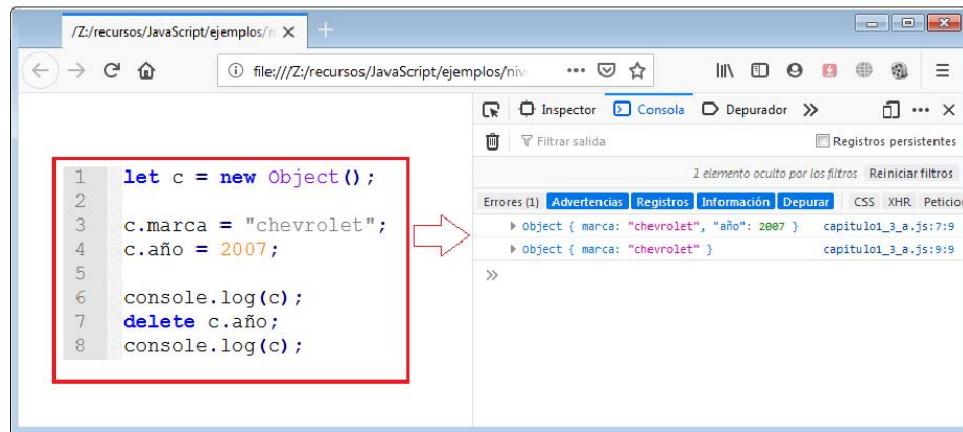
```
marca => chevrolet
año => 1999
color => Rojo
nuevo => false
```

1.3.- Manipulando Objetos

Además de agregar atributos a un objeto, se pueden eliminar atributos en tiempo de ejecución. Se logra usando la palabra reservada "delete", de la siguiente forma:

```
delete objeto.atributo;
```

Delete elimina el atributo y el valor. Por ejemplo:



A screenshot of a browser's developer tools showing the JavaScript console. On the left, there is a code editor window containing the following JavaScript code:

```

1 let c = new Object();
2 
3 c.marca = "chevrolet";
4 c.año = 2007;
5 
6 console.log(c);
7 delete c.año;
8 console.log(c);

```

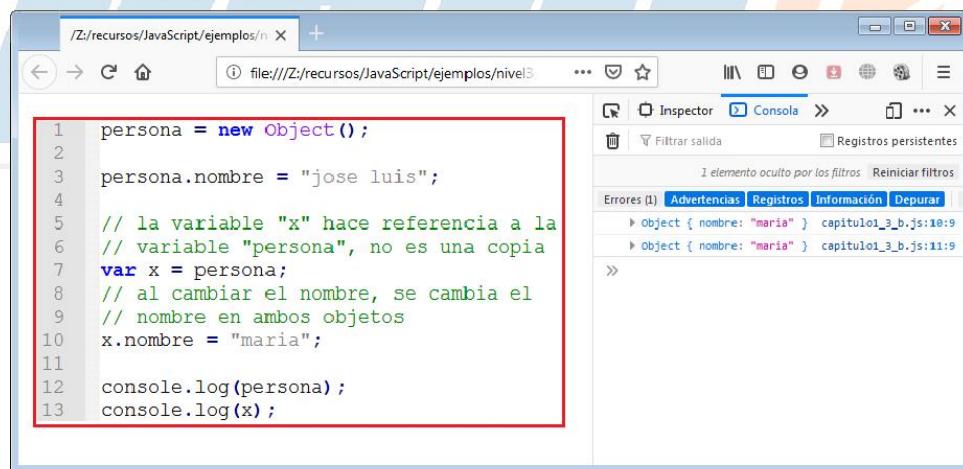
The code is highlighted with a red box around the first two lines. An orange arrow points from this box to the right side of the screen, where the developer tools interface is shown. In the bottom right corner of the tools, there is a log pane displaying the output of the console logs:

```

Object { marca: "chevrolet", año: 2007 }    capítulo1_3_a.js:7:9
Object { marca: "chevrolet" }                  capítulo1_3_a.js:9:9
>

```

Los objetos en JavaScript son "mutables", es decir, son direccionalos por referencia y no por valor. Por ejemplo, si la variable "persona" contiene la referencia a un objeto, al asignar su valor a otra variable, no se crea una copia del objeto



A screenshot of a browser's developer tools showing the JavaScript console. On the left, there is a code editor window containing the following JavaScript code:

```

1 persona = new Object();
2 
3 persona.nombre = "jose luis";
4 
5 // la variable "x" hace referencia a la
6 // variable "persona", no es una copia
7 var x = persona;
8 
9 // al cambiar el nombre, se cambia el
10 // nombre en ambos objetos
11 x.nombre = "maria";
12 
13 console.log(persona);
14 console.log(x);

```

The code is highlighted with a red box around the first two lines. An orange arrow points from this box to the right side of the screen, where the developer tools interface is shown. In the bottom right corner of the tools, there is a log pane displaying the output of the console logs:

```

Object { nombre: "maria" }    capítulo1_3_b.js:10:9
Object { nombre: "maria" }    capítulo1_3_b.js:11:9
>

```

Después de la asignación, ambas variables "persona" y "x" son el mismo objeto, por lo tanto, cualquier cambio a los atributos de la variable "x" afectaran a "persona" y viceversa.

Capítulo 2. CONSTRUCTORES

2.1.- Constructores

Otra forma de crear objetos es usando una función constructora. Es equivalente a hacer un "new Object()", pero hace más versátil y reusable el trabajo. Para lograrlo, se define una función con un nombre arbitrario (que generalice los objetos que se van a crear) y adentro de ésta, se asigna un valor a cada atributo que tendrá el objeto que se desea crear, haciendo uso de la palabra reservada "this", que hace referencia al nuevo objeto que se está creando. Por ejemplo:

```
1  function Vehiculo()
2  {
3      this.marca = "chevrolet";
4      this.año = 1999;
5      this.color = "Rojo";
6      this.nuevo = false;
7  }
8
9  var c = new Vehiculo();
```

Adentro de la función constructora se pueden asignar a los atributos valores literales o valores retornados por una función, por ejemplo, números aleatorios.

2.2.- Constructores Con Parámetros

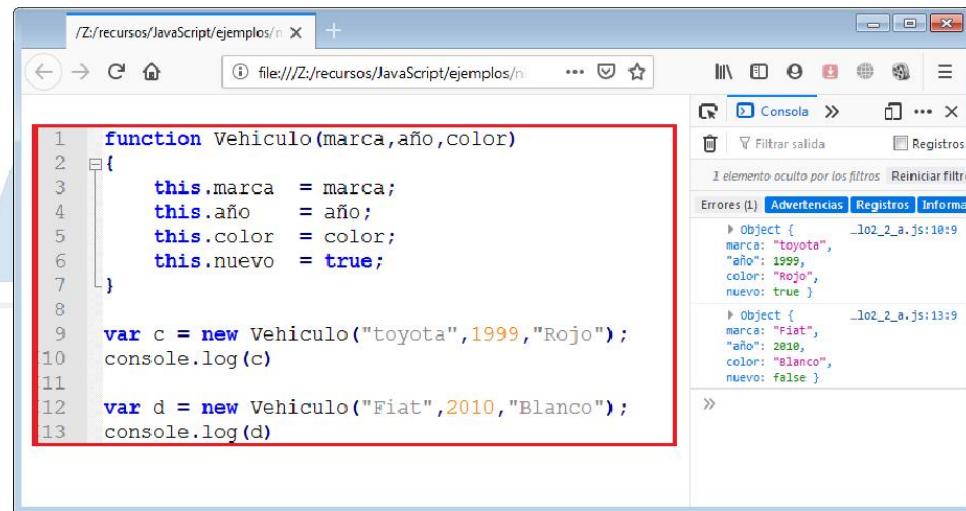
Algunos de los valores iniciales de los atributos pueden ser parametrizados, es decir, especificados al momento de crear el objeto. Esto hará que la función constructora sea más reusable. Para lograrlo, la función constructora debe recibir parámetros. Por ejemplo:

```

1   function Vehiculo(marca,año,color)
2   {
3       this.marca = marca;
4       this.año = año;
5       this.color = color;
6       this.nuevo = true;
7   }

```

Al momento de llamar a la función, se deben enviar los parámetros actuales, haciendo que el nuevo objeto que nace tenga en sus atributos los valores recibidos por parámetro. Por ejemplo:



The screenshot shows a browser window with developer tools open. The code in the script pane is:

```

1   function Vehiculo(marca,año,color)
2   {
3       this.marca = marca;
4       this.año = año;
5       this.color = color;
6       this.nuevo = true;
7   }
8
9   var c = new Vehiculo("toyota",1999,"Rojo");
10  console.log(c)
11
12  var d = new Vehiculo("Fiat",2010,"Blanco");
13  console.log(d)

```

The output in the console pane shows two objects created by the function:

```

Object {
  marca: "toyota",
  año: 1999,
  color: "Rojo",
  nuevo: true }

Object {
  marca: "Fiat",
  año: 2010,
  color: "Blanco",
  nuevo: false }

```

2.3.- Arreglos de Objetos

En los niveles anteriores se han usado arreglos de valores simples: numéricos, lógicos y strings. Sin embargo, también se pueden crear arreglos de objetos. Este evitaría tener arreglos paralelos para cada dato que se desea almacenar. Por ejemplo, si se necesita almacenar los datos (marca, año y color) de 3 vehículos, se pueden crear 3 arreglos con 3 valores cada uno (un arreglo para cada dato), por ejemplo:

The screenshot shows a browser window with the URL `file:///Z:/recursos/JavaScript/ejemplos/nivel3/capitulo2_3_a.js`. The console tab is active, displaying the output of the following JavaScript code:

```

1 // Con arreglos paralelos:
2
3 // arreglo de marcas
4 var marcas = ["Toyota", "Fiat", "Ford"];
5 // arreglo de años
6 var años = [1999, 2010, 2009];
7 // arreglo de colores
8 var colores = ["Blanco", "Azul", "Blanco"];
9
10 console.log("El vehículo " + marcas[0] +
11     " es color " + colores[0] +
12     " y del año " + años[0]);

```

The console output is: `El vehículo Toyota es color Blanco y del año 1999`.

Con arreglos de objetos, se crea un solo arreglo con 3 objetos, donde cada objeto tendrá los valores de los 3 datos. El siguiente código muestra el ejemplo anterior, pero usando un arreglo de objetos:

The screenshot shows a browser window with the URL `file:///Z:/recursos/JavaScript/ejemplos/nivel3/capitulo2_3_b.js`. The console tab is active, displaying the output of the following JavaScript code:

```

1 // con arreglos de objetos:
2
3 var vehiculos = []; // un solo arreglo
4
5 // primer vehículo
6 vehiculos.push(new Vehiculo("Toyota", 1999, "Blanco"));
7 // segundo vehículo
8 vehiculos.push(new Vehiculo("Fiat", 2010, "Azul"));
9 // tercer vehículo
10 vehiculos.push(new Vehiculo("Ford", 2009, "Blanco"));
11
12 console.log("El vehículo " + vehiculos[0].marca +
13     " es color " + vehiculos[0].color +
14     " y del año " + vehiculos[0].año);

```

The console output is: `El vehículo Toyota es color Blanco y del año 1999`.

Se pueden crear los objetos usando un ciclo y obteniendo datos del usuario usando "prompt" (o de inputs en formularios):

```
1 var vehiculos = [];// un solo arreglo
2
3 for (let i=1;i<=3;i++)
4 {
5     let marca =prompt("Marca:");
6     let año   =prompt("Año:");
7     let color =prompt("Color:");
8     vehiculos.push(new Vehiculo(marca,año,color));
9 }
10 console.log("El vehiculo "+vehiculos[0].marca+
11           " es color " +vehiculos[0].color+
12           " y del año " +vehiculos[0].año);
```

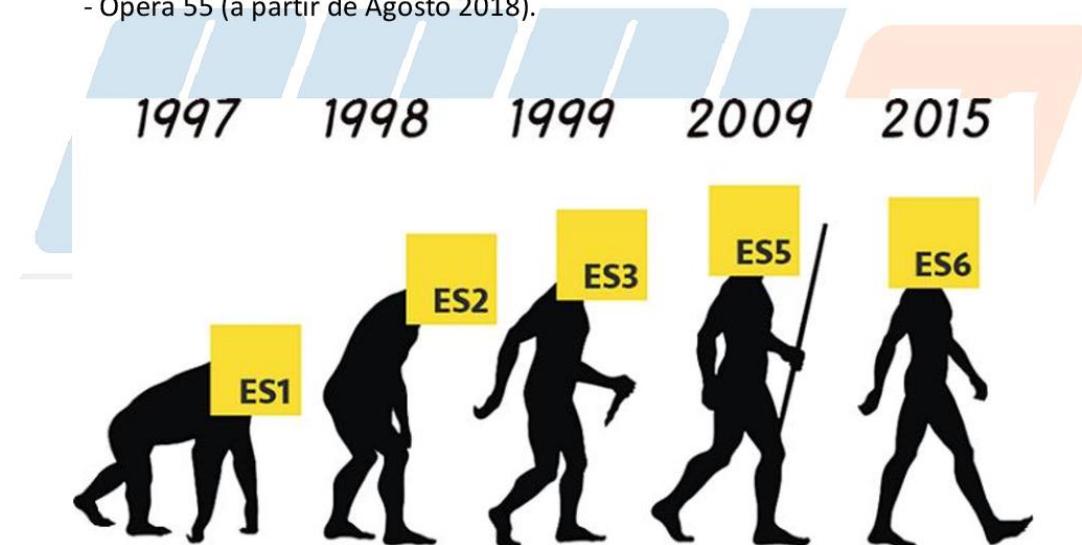


Capítulo 3. CLASES

3.1.- Definición de Clases

Las clases (como tradicionalmente se manejan en lenguajes de programación como Php, Java, C++, etc) de JavaScript fueron introducidas en ECMAScript 6 (ES6), también conocido como ECMAScript 2015, que es soportado por los navegadores más importantes en las siguientes versiones:

- Chrome 58 (a partir de Enero 2017).
- Edge 14 (a partir de Agosto 2016).
- Firefox 54 (a partir de Marzo 2017).
- Safari 10 (a partir de Julio 2016).
- Opera 55 (a partir de Agosto 2018).



Las clases son "funciones especiales". Una manera de definir una clase es mediante una declaración de clase. Para declarar una clase, se utiliza la palabra reservada "class" y un nombre arbitrario, por ejemplo:

```
1 // clase Vehiculo
2 class Vehiculo
3 {
4
5 }
6
7 // clase Cliente
8 class Cliente
9 {
10
11 }
```

3.2.- Constructores

El método constructor es un método especial de una clase para crear e inicializar un objeto. Es equivalente a la "función constructora" vista en el capítulo previo. Solo puede haber un método especial con el nombre "constructor" en una clase. Por ejemplo:

```
1 class Vehiculo
2 {
3     constructor()
4     {
5         this.marca = "chevrolet";
6         this.color = "rojo";
7         this.año = 1999;
8         this.nuevo = true;
9     }
10 }
```

Los constructores de una clase también pueden tener parámetros:

```
1 class Vehiculo
2 {
3     constructor(marca,color,año)
4     {
5         this.marca = marca;
6         this.color = color;
7         this.año = año;
8         this.nuevo = true;
9     }
10 }
```

3.3.- Instanciación

Para instanciar una clase (crear un nuevo objeto de la clase), se usa el nombre de la clase como función constructora luego de la palabra "new". Por ejemplo:

The screenshot shows a browser window with developer tools open. The code in the console is:

```
1 class Vehiculo
2 {
3     constructor()
4     {
5         this.marca = "chevrolet";
6         this.color = "rojo";
7         this.año = 1999;
8         this.nuevo = true;
9     }
10 }
11 var c = new Vehiculo();
12 console.log(c);
```

The output in the console is:

```
▶ object { ...03_1_a.js:12:9
marca: "chevrolet",
color: "rojo",
"año": 1999,
nuevo: true }
```

Si el constructor tiene parámetros, se deben pasar al momento de hacer "new":

```

1 class Vehiculo
2 {
3     constructor(marca,color,año)
4     {
5         this.marca = marca;
6         this.color = color;
7         this.año = año;
8         this.nuevo = true;
9     }
10 }
11
12 var c= new Vehiculo("toyota","rojo",1999);
13 console.log(c);

```

The screenshot shows a browser developer tools console. On the left, there is a code editor window with the above JavaScript code. On the right, the 'Console' tab is selected, showing the output of `console.log(c)`:

```

Object { marca: "toyota", color: "rojo", año: 1999, nuevo: true }

```

Un atributo de un objeto puede ser un objeto de otra clase, por ejemplo:

```

1 class Cliente
2 {
3     constructor(cedula,nombre)
4     {
5         this.cedula=cedula;
6         this.nombre=nombre;
7     }
8 }
9 class Vehiculo
10 {
11     constructor(marca,color,año)
12     {
13         this.marca = marca;
14         this.color = color;
15         this.año = año;
16         this.nuevo = true;
17         this.dueño = new Cliente("1234","jose");
18     }
19 }
20
21 var c= new Vehiculo("toyota","rojo",1999);
22 console.log(c.dueño)
23 console.log(c);

```

The screenshot shows a browser developer tools console. On the left, there is a code editor window with the above JavaScript code. On the right, the 'Console' tab is selected, showing the output of `console.log(c.dueño)` and `console.log(c)`:

```

Object { cedula: "1234", nombre: "jose" }
Object { año: 1999, color: "rojo", dueño: Object { cedula: "1234", nombre: "jose" }, nuevo: true }

```

Capítulo 4. MÉTODOS

4.1.- Definición

Un método es una función asociada a un objeto. La forma de definir los métodos depende de la forma de crear un objeto. Si el objeto se crea usando la clase Object, la forma de definir un método es la siguiente:

```
1  var c= new Object();
2
3
4  c.marca      = "chevrolet";
5  c.año        = 1999;
6  c.color       = "Rojo";
7  c.nuevo       = false;
8  c.saludar    = function ()
9  {
10    console.log("Esto es un método");
11 }
```

Si se utiliza una función constructora, la forma de definir un método es la siguiente:

```
1  function Vehiculo()
2  {
3    this.marca  = "chevrolet";
4    this.año    = 1999;
5    this.color  = "Rojo";
6    this.nuevo  = false;
7    this.saludar = function ()
8    {
9      console.log("Esto es un método");
10     }
11 }
```

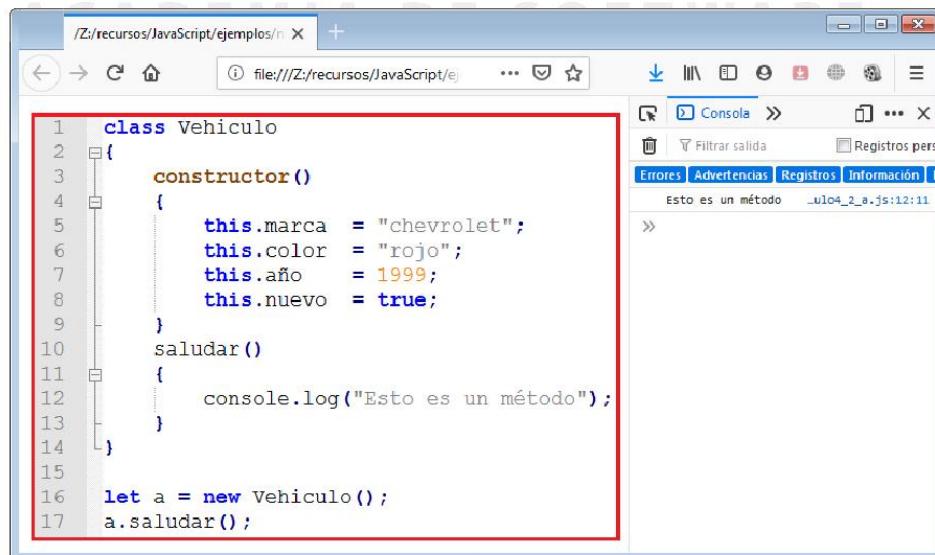
Si se define una clase, la forma de definir un método es la siguiente:

```

1  class Vehiculo
2  {
3      constructor()
4      {
5          this.marca = "chevrolet";
6          this.color = "rojo";
7          this.año = 1999;
8          this.nuevo = true;
9      }
10     saludar()
11     {
12         console.log("Esto es un método");
13     }
14 }
```

4.2.- Ejecución

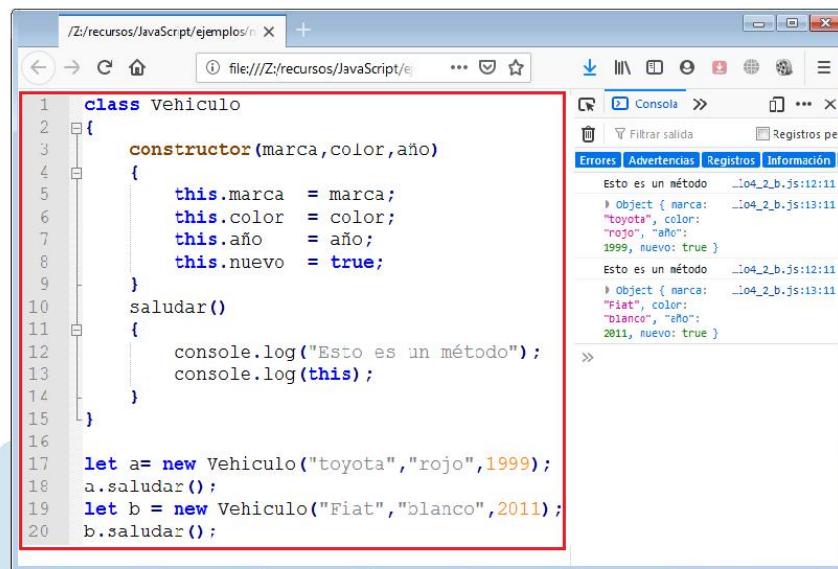
Para ejecutar los métodos, independientemente de la forma como se defina, se debe hacer igual que con los atributos: objeto.método(), por ejemplo:



```

1  class Vehiculo
2  {
3      constructor()
4      {
5          this.marca = "chevrolet";
6          this.color = "rojo";
7          this.año = 1999;
8          this.nuevo = true;
9      }
10     saludar()
11     {
12         console.log("Esto es un método");
13     }
14 }
15
16 let a = new Vehiculo();
17 a.saludar();
```

En los métodos se puede acceder a los datos de los atributos del objeto que llama el método. Esto se logra con el identificador "this":



The screenshot shows a browser window with developer tools open. The code in the editor is:

```

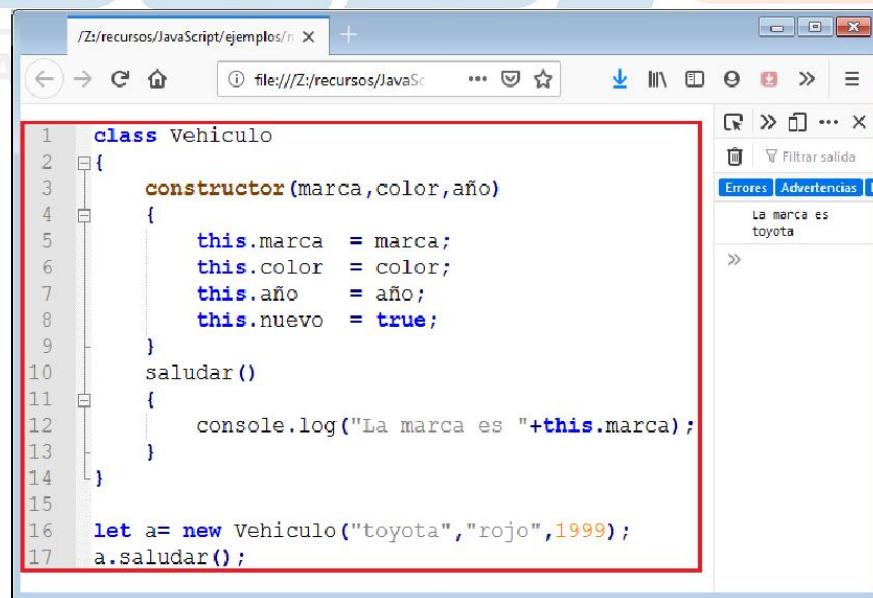
1 class Vehiculo
2 {
3     constructor(marca,color,año)
4     {
5         this.marca = marca;
6         this.color = color;
7         this.año = año;
8         this.nuevo = true;
9     }
10    saludar()
11    {
12        console.log("Esto es un método");
13        console.log(this);
14    }
15}
16
17 let a= new Vehiculo("toyota","rojo",1999);
18 a.saludar();
19 let b = new Vehiculo("Fiat","blanco",2011);
20 b.saludar();

```

The output in the console tab shows two objects created from the Vehiculo class:

- Object { marca: "toyota", color: "rojo", año: 1999, nuevo: true }
- Object { marca: "Fiat", color: "blanco", año: 2011, nuevo: true }

Si se necesita acceder al valor de un atributo, se utiliza this un punto y el atributo.



The screenshot shows a browser window with developer tools open. The code in the editor is:

```

1 class Vehiculo
2 {
3     constructor(marca,color,año)
4     {
5         this.marca = marca;
6         this.color = color;
7         this.año = año;
8         this.nuevo = true;
9     }
10    saludar()
11    {
12        console.log("La marca es "+this.marca);
13    }
14}
15
16 let a= new Vehiculo("toyota","rojo",1999);
17 a.saludar();

```

The output in the console tab shows the result of the console.log statement:

La marca es toyota

4.3.- Parámetros y Valor de Retorno

Al ser un método una función como cualquier otra (con la diferencia que está asociada a un objeto), los métodos también pueden recibir parámetros. En el siguiente ejemplo, el método "calcularDescuento" recibe por parámetro un "precio". Al llamar al método (línea 21), se pasa el valor del parámetro:

The screenshot shows a browser window with developer tools open. The main area displays the following JavaScript code:

```
1 class Vehiculo
2 {
3     constructor(marca,color,año)
4     {
5         this.marca = marca;
6         this.color = color;
7         this.año = año;
8         this.nuevo = true;
9     }
10    calcularDescuento(precio)
11    {
12        console.log("El descuento es:");
13        if (this.año > 2000)
14            console.log(precio*0.05);
15        else
16            console.log(precio*0.15);
17    }
18 }
19
20 let a= new Vehiculo("toyota","rojo",1999);
21 a.calcularDescuento(5000);
```

The code defines a class `Vehiculo` with a constructor and a method `calcularDescuento`. The method logs the discount based on the car's age. Line 21 creates an instance of `Vehiculo` and calls its `calcularDescuento` method with the argument `5000`. The output on the right side of the developer tools shows the result: `El descuento es: 750`.

En el bloque de un método puede haber instrucciones de cualquier tipo: operaciones aritméticas, condiciones, ciclos, etc. El método puede retornar un valor que se calcule internamente, usando la palabra reservada "return". Por ejemplo:

```
1  class Vehiculo
2  {
3      constructor(marca,color,año)
4      {
5          this.marca = marca;
6          this.color = color;
7          this.año = año;
8          this.nuevo = true;
9      }
10     calcularDescuento(precio)
11     {
12         if (this.año > 2000)
13             return precio*0.05;
14         else
15             return precio*0.15;
16     }
17 }
18
19 let a= new Vehiculo("toyota","rojo",1999);
20 let d=a.calcularDescuento(5000);
21 console.log("El descuento es:"+d);
```

ACADEMIA DE SOFTWARE

Capítulo 5. JSON. PARTE 1

5.1.- Objetos Literales

JSON (JavaScript Object Notation) es un formato de peso ligero para intercambio de datos entre aplicaciones. Es fácil para los humanos leerlo y escribirlo y es fácil para las máquinas generarla e interpretarla (parse). Es un formato de texto completamente independiente del lenguaje de programación que lo utilice, pero usa convenciones que son familiares a los programadores de C, C++, Java, Python, entre otros.

JSON está construido en base a dos estructuras:

- Una colección de pares nombre/valor.
- Una lista ordenada de valores (arreglos).

La notación JSON encierra la colección de pares de nombre/valor entre { } y las listas ordenadas entre [].

Usar la notación JSON en el código JavaScript es otra forma de crear objetos. A esto se le llama "objetos literales". El siguiente ejemplo muestra cómo declarar un objeto literal con varios atributos. Si el valor numérico de un atributo esta encerrado entre comillas, se asume que es un String, en caso contrario, se maneja como número:

```
<script language="javascript">
    var objeto=
    {
        "nombre":"jose luis",
        "apellido":"rojas dellan",
        "edad":34
    }
</script>
```

Un objeto literal puede contener métodos en su definición. Por ejemplo:

```
<script language="javascript">
    var objeto=
    {
        "nombre":"jose luis",
        "apellido":"rojas dellan",
        "edad":34,
        "saludar": function ()
        {
            alert("Mi nombre es "+this.nombre);
        }
    }
    objeto.saludar();
</script>
```

5.2.- Objetos Anidados

Uno de los atributos de un objeto puede ser un arreglo. En ese caso, deben usarse los corchetes [] al asignarle valor o valores al atributo (también pueden quedar dos corchetes vacíos indicando que el arreglo no tiene elementos). Para acceder a los elementos del arreglo se debe colocar el objeto, el atributo y la posición del valor que se desea acceder:

```
<script language="javascript">
    var objeto=
    {
        "nombre":"jose luis",
        "apellido":"rojas dellan",
        "edad":34,
        "notas": [15,18,20,14],
        "saludar": function ()
        {
            alert("Mi nombre es "+this.nombre);
        }
    }
    objeto.saludar();
    alert("La primera nota fue "+objeto.notas[0]);
</script>
```

Un objeto también puede tener como atributo otro objeto. En ese caso, deben colocarse otro par de llaves en la declaración del atributo. Para acceder a los valores de los atributos del objeto interno se debe colocar otro punto y el nombre del atributo. La forma general es:

objeto.objetointerno.atributo.

El siguiente es un ejemplo:

```
<script language="javascript">
    var objeto=
    {
        "nombre":"jose luis",
        "apellido":"rojas dellan",
        "edad":34,
        "notas": [15,18,20,14],
        "fecha": {
            "dia":11,
            "mes":1,
            "anno":1980
        },
        "saludar": function ()
        {
            alert("Mi nombre es "+this.nombre);
        }
    }
    objeto.saludar();
    alert("Nacio el dia "+objeto.fecha.dia);
</script>
```

En la notación JSON, para definir un arreglo de objetos o una colección (o lista) de elementos se usan los corchetes. El siguiente es un ejemplo de un arreglo de objetos:

```
<script language="javascript">
    var alumnos =
    [
        {
            "nombre": "jose",
            "nota": 15
        },
        {
            "nombre": "maria",
            "nota": 17
        }
    ];
    alert("La nota del primer alumno fue "+alumnos[0].nota);
    alert("El nombre del segundo alumno es "+alumnos[1].nombre);
</script>
```

5.3.- Archivos Json

JSON es un formato que se ha extendido en la industria del software, al punto tal de usarlo para sustituir antiguas formas de almacenar configuraciones de aplicaciones, tales como archivos .ini., aprovechando su simplicidad y extendido uso por los programadores. Es por eso que es cada vez más común encontrar archivos de texto con extensión ".json" que contienen información útil.

En la siguiente imagen se muestra un ejemplo de un archivo JSON utilizado por TypeScript (un lenguaje de programación basado en JavaScript), para conocer cómo debe comportarse al ejecutarse.

```
tsconfig.json
1  {
2    "compileOnSave": false,
3    "compilerOptions": {
4      "baseUrl": "./",
5      "outDir": "./dist/out-tsc",
6      "sourceMap": true,
7      "declaration": false,
8      "module": "es2015",
9      "moduleResolution": "node",
10     "emitDecoratorMetadata": true,
11     "experimentalDecorators": true,
12     "importHelpers": true,
13     "target": "es5",
14     "typeRoots": [
15       "node_modules/@types"
16     ],
17     "lib": [
18       "es2018",
19       "dom"
20     ]
21   }
22 }
```

ACADEMIA DE SOFTWARE

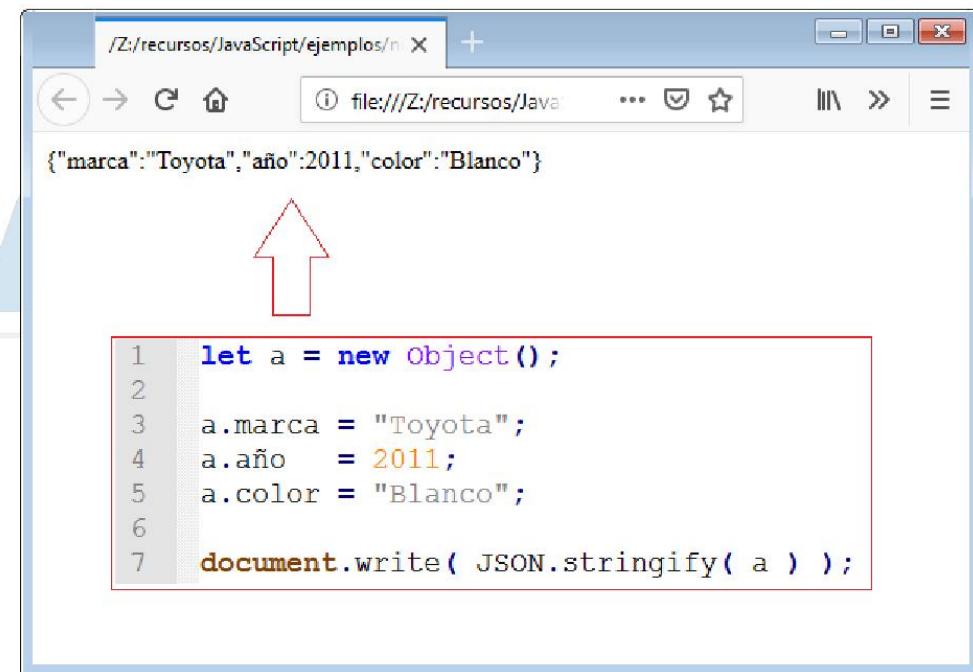
Capítulo 6. JSON. PARTE 2

6.1.- Stringify

JavaScript ofrece una clase para trabajar con objetos JSON. Su nombre es JSON y ofrece 2 métodos sencillos:

- stringify: convierte un objeto en string.
- parse: convierte un string en un objeto.

La función "stringify" recibe por parámetro un objeto creado de cualquiera de las formas vistas anteriormente y retorna un string en forma JSON. Por ejemplo:



The screenshot shows a web browser window with the URL `file:///Z:/recursos/Java`. The page content displays the JSON string `{"marca": "Toyota", "año": 2011, "color": "Blanco"}`. A red arrow points from this string up towards the browser's address bar. Below the browser window, a code editor window is visible, showing the following JavaScript code:

```
1 let a = new Object();
2
3 a.marca = "Toyota";
4 a.año = 2011;
5 a.color = "Blanco";
6
7 document.write( JSON.stringify( a ) );
```

Si un objeto tiene entre sus datos a otro objeto, al convertir el objeto a String se generará el objeto interno como parte del String completo, como se muestra en el siguiente ejemplo, donde el objeto "c" tiene un atributo con el nombre "dueño" que es de la clase Cliente:

The screenshot shows a browser window with the URL `file:///Z:/recursos/JavaScript/ejemplos/n`. The page displays a JSON object in the address bar:

```
{"marca": "toyota", "color": "rojo", "año": 1999, "nuevo": true, "dueño": {"cedula": "1234", "nombre": "jose"}}
```

Below the JSON object is a block of JavaScript code:

```
1 class Cliente
2 {
3     constructor(cedula, nombre)
4     {
5         this.cedula=cedula;
6         this.nombre=nombre;
7     }
8 }
9 class Vehiculo
10 {
11     constructor(marca, color, año)
12     {
13         this.marca = marca;
14         this.color = color;
15         this.año = año;
16         this.nuevo = true;
17         this.dueño = new Cliente("1234", "jose");
18     }
19 }
20
21 var c= new Vehiculo("toyota", "rojo", 1999);
22 document.write( JSON.stringify( c ) );
```

A red arrow points upwards from the JSON object in the browser's address bar towards the first line of the JavaScript code.

6.2.- Parse

La función "parse" recibe por parámetro un "string" con formato JSON e intenta convertirlo en un objeto JavaScript para manipularlo de ahí en adelante como objeto. Los siguientes son ejemplos de conversión a objetos de 2 cadenas:

A screenshot of a browser's developer tools console. The code in the script pane is:

```

1 let s = '{"nombre":"Jose Luis","edad":39}';
2 let c = JSON.parse(s);
3 // se muestra el nombre del objeto
4 console.log(c.nombre);
5
6 let x = '{"nombre":"Pedro","notas":[15,12,20]}';
7 let y = JSON.parse(x);
8 // se muestra la primera nota del objeto
9 console.log(y.notas[1]);

```

The output pane shows the results of the execution:

```

Jose Luis
12
»

```

Si "parse" no logra convertir el string en un objeto, se generará una excepción del tipo SyntaxError (error en tiempo de ejecución), que detendrá el script a menos que se maneje la excepción. Las instrucciones posteriores a la línea que genera la excepción no se ejecutarán. Por ejemplo:

A screenshot of a browser's developer tools console. The code in the script pane is:

```

1 // falta una comilla doble para cerrar el
2 // atributo edad
3 let x = '{"nombre":"Pedro Perez","edad:39"}';
4
5 // producira un error
6 let y = JSON.parse(x);

```

An arrow points from the error message in the errors panel to the incomplete JSON string in the code.

The errors panel shows:

```

syntaxError:
  JSON.parse:
    unterminated string
  at line 1 column 34
  of the JSON
  data [Saber más]
»

```

Para evitar que el script se detenga por completo, se debe manejar la excepción usando una instrucción "try catch". Si la conversión no se da exitosamente, se ejecutará el bloque en las llaves del "catch". Por ejemplo:

A screenshot of a browser's developer tools window, specifically the 'Console' tab. The code in the editor is:

```
1 // falta una comilla doble para cerrar el
2 // atributo edad
3 let x = '{"nombre":"Pedro Perez","edad:39}';
4
5 try {
6     // producira un error
7     let y = JSON.parse(x);
8 } catch (e) {
9     console.log("Error al intentar convertir");
10 }
```

The line `let y = JSON.parse(x);` is highlighted with a red box. In the console output, there is one entry:

Error al intentar convertir a JSON
capítulo6_2_c.js:9:11



Capítulo 7. ALMACENAMIENTO LOCAL. PARTE 1

7.1.- Introducción

El almacenamiento en una aplicación Web puede realizarse en 2 lugares:

- en el servidor (server side): en archivos o bases de datos en el disco duro del servidor donde se ejecuta el BackEnd de la aplicación.
- en el cliente (client side): en el navegador donde se ejecuta la aplicación.

El almacenamiento en el servidor se utiliza cuando se desean almacenar datos fuera del contexto del equipo donde se ejecuta la aplicación (PC, Teléfono, TV, etc), para tenerlos disponibles al acceder desde otro equipo. Los datos deben enviarse desde la aplicación (FrontEnd) al servidor (Backend), para ser almacenados en este último. Este tipo de almacenamiento es un tema que escapa del alcance del curso, debido a que requiere el uso de un lenguaje de servidor, como: PHP, Ruby, Java, o NodeJs (JavaScript ejecutándose fuera del navegador).

El almacenamiento local, se utiliza cuando se desean guardar datos de la aplicación, de modo que estén disponibles aunque no haya conexión con el Backend. Algunos usos de este almacenamiento son:

- mantener una sesión de usuario abierta aunque se cierre el navegador.
- recordar con qué cuentas se ha autenticado previamente un usuario en una aplicación (como lo hace google).
- guardar un historial de búsquedas realizadas.
- almacenar datos mientras no haya conexión con el Backend, para ser enviados cuando se restaure la conexión.

La desventaja del almacenamiento local, es que los datos estarán disponibles solo en el navegador donde se ejecuta la aplicación.

7.2.- Opciones de Almacenamiento Local

Para el almacenamiento local existen 4 opciones:

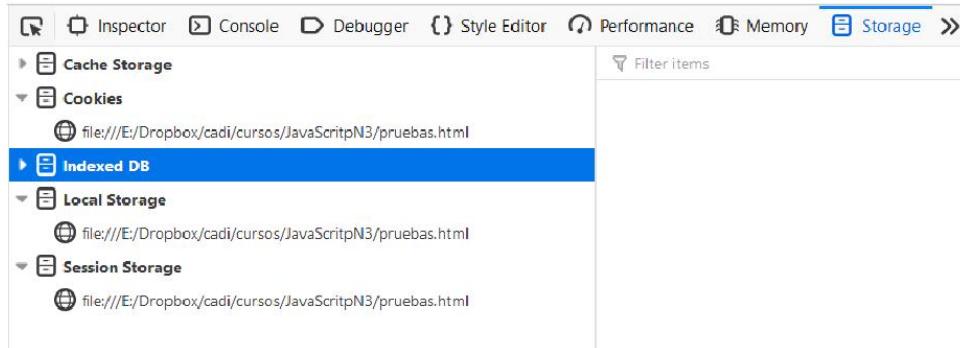
- Cookies.
- LocalStorage (disponible a partir de HTML5).
- SessionStorage (disponible a partir de HTML5).
- IndexDb

En este nivel del curso no se trabajará con IndexDB.

En la siguiente tabla comparativa se pueden apreciar las diferencias entre cada uno de los tipos de almacenamiento disponibles:

Característica	Cookies	LocalStorage	SessionStorage
Tamaño máximo	4 KB	5 MB (depende del navegador)	5 MB
Tipos de datos soportados	String	String	String
Bloqueable por el usuario	Si	Si	Si
Opción de auto expiración	Si	No	Si
Soporte por navegadores	Alto	Alto	Alto
Accesible del lado del servidor	Si	No	No
Se transfiere en cada petición HTTP	Si	No	No
Modificable por el usuario final	Si	Si	Si
Se puede acceder desde código de	Cliente y Servidor	Solo Cliente	Solo Cliente
Eliminable / borrible desde	Cliente y Servidor	Solo Cliente	Solo Cliente
Disponible es	Cualquier página del mismo dominio	Cualquier página del mismo dominio	Solo La pestaña donde se creo
Tiempo de vida	Según se especifique	Hasta que se borre	Hasta que cierre la pestaña
Almacenamiento seguro	No	No	No

Todos los valores almacenados en cualquiera de estos medios pueden visualizarse y manipularse en el navegador. Cada navegador tiene su forma de hacerlo. En Firefox se visualizan en la ventana de desarrollador, en la opción "Storage" o "Almacenamiento", como se puede observar en la siguiente imagen:



7.3.- Usando Cookies

Las cookies son el mecanismo de almacenamiento usado desde casi los inicios de la Web. Son manipulables desde el lado del servidor y desde el lado del cliente. Pueden crearse y destruirse desde ambos lugares. Todas las cookies son enviadas en las peticiones al servidor, por eso, son muy utilizadas para almacenar información del usuario autenticado y saber en el Backend (el servidor), quien está haciendo peticiones.

Para guardar u obtener valores almacenados en Cookies, se usa el atributo "cookie" del objeto "document":

```
document.cookie = "variable=valor"
```

Al crear una cookie, se establece el nombre y el valor inicial de la misma, por ejemplo:

```
document.cookie = "username=jose"
```

En el ejemplo anterior, se está creando una Cookie con el nombre "username" y el valor "jose". En Google Chrome no se pueden crear cookies ejecutando una pagina que no esté alojada en un dominio, es decir, probando en archivos locales no funcionara la creación de cookies.

A las cookies se les puede establecer un momento de expiración, lo que indicará que la cookie existirá hasta antes de que se alcance la fecha/hora de expiración. Por defecto, si al crear la cookie no se especifica el momento de expiración, éste se establece en 0, conocido también como expiración "Session". Cuando una cookie tiene el momento de

expiración en el valor "session", dependiendo del navegador, la variable será eliminada al momento de cerrar la ventana del navegador (a excepción de Firefox).

Para establecer el momento de expiración de una cookie, se asigna el atributo "expires", asignándole un valor de fecha/hora en formato UTC, por ejemplo:

```
document.cookie = "username=jose; expires=Thu, 01 Jan 2019 00:00:00 UTC;  
path=/";
```



Capítulo 8. ALMACENAMIENTO LOCAL. PARTE 2

8.1.- Localstorage vs Sessionstorage

Como se explicó anteriormente, a partir de HTML 5 se crearon 2 nuevas formas de almacenamiento local: LocalStorage y SessionStorage. Ambos son muy similares:

- trabajan con pares clave/valor.
- tienen una capacidad de almacenamiento mayor al de las cookies.
- poseen una interfaz sencilla para crear, acceder y eliminar valores.

LocalStorage y sessionStorage son útiles para persistir información no sensible entre páginas. Nunca debería guardarse información sensible, como pueden ser las contraseñas o información que puedan comprometer la seguridad del usuario.

La diferencia entre localStorage y sessionStorage, es que en localStorage se guarda la información de forma persistente en el navegador, es decir, si el usuario cierra el navegador y vuelve a entrar en éste, la información aun seguirá disponible. En cambio, en sessionStorage, la información sólo dura lo que dure la sesión del navegador. Al cerrar la ventana se perderá la información.

También es importante señalar que el sessionStorage sólo sirve para la ventana o pestana que se está viendo. Si el usuario abre un enlace de la aplicación en una nueva ventana o un nuevo TAB, esta información tampoco estará disponible.

8.2.- Manipular Datos en el Localstorage

Para manipular los datos del "localStorage", los navegadores ofrecen una interfaz en ese nombre, que contiene los métodos necesarios para realizar las operaciones básicas:

- localStorage.setItem("clave","valor"): se utiliza para guardar un "valor" identificado con el nombre "clave".
- localStorage.getItem("clave"): se utiliza para consultar un valor identificado con una "clave". Si la clave no existe, retornará "undefined".

- `localStorage.removeItem("clave")`: se utiliza para eliminar un valor identificado con una "clave".
- `localStorage.clear()`: limpia todos los valores almacenados en el `localStorage`.

Algunos ejemplos de como utilizar los métodos de `localStorage`:

```
// almacena un valor el valor "John" asociado a la clave "name"  
localStorage.setItem( "name", "John" );
```

```
// obtiene un valor almacenado en la clave "name"  
localStorage.getItem( "name" );
```

```
// elimina el valor asociado a la clave "name"  
localStorage.removeItem( "name" );
```

```
// borrar todos los valores almacenados  
localStorage.clear();
```

Para trabajar con objetos (almacenarlos y recuperarlos) deben usarse las funciones de JSON stringify y parse:

```
// se crea un objeto literal  
var libro = {  
    name: "Cookies are old",  
    author: "Gordan"  
}  
// se convierte o serializado el objeto "libro" en string y se guarda asociado con la  
// clave "libro"  
localStorage.setItem( "libro", JSON.stringify( libro ) );
```

```
// se recupera el valor de la clave "libro" como un string y se convierte en objeto  
nuevamente  
var post = JSON.parse( localStorage.getItem( "post" ) );
```

8.3.- Manipular Datos en el Sessionstorage

Para el sessionStorage también hay una interfaz con el mismo nombre y aplican exactamente los mismos métodos que el localStorage. Algunos ejemplos de como usar sessionStorage son:

```
// almacena un valor el valor "John" asociado a la clave "name"  
sessionStorage.setItem( "name", "John" );
```

```
// obtiene un valor almacenado en la clave "name"  
sessionStorage.getItem( "name" );
```

```
// elimina el valor asociado a la clave "name"  
sessionStorage.removeItem( "name" );
```

```
// borrar todos los valores almacenados  
sessionStorage.clear();
```



Capítulo 9. INTRODUCCIÓN JQUERY

9.1.- ¿Qué es jQuery?

jQuery es librería de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejo de eventos, desarrollar animaciones y agregar interacción con la técnica AJAX en páginas web. jQuery, al igual que otras librerías ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta librería se logran grandes resultados en menos tiempo y espacio.



jQuery es considerado una librería que sirve como base para la programación avanzada de aplicaciones. Posee una serie de funciones para realizar tareas habituales.

Los programadores utilizan las librerías para no tener que desarrollar las tareas más básicas, puesto que en la librería ya hay implementaciones que están probadas, funcionan y no se necesitan volver a programar.



jQuery posee las siguientes características:

- Selección de elementos DOM.
- Interactividad y modificaciones del árbol DOM, incluyendo soporte para CSS 1 y CSS 3.
- Manejar y manipular eventos.
- Manipulación de la hoja de estilos CSS.
- Efectos y animaciones.
- Compatible con los navegadores Mozilla Firefox 2.0+, Internet Explorer 6+, Safari 3+, Opera 10.6+ y Google Chrome 8+.
- AJAX.

La librería jQuery en resumen aporta las siguientes ventajas:

- Cross-Browser: es un framework compatible con muchos navegadores. Con jQuery la mayor parte del código que escribas funcionará perfectamente en los principales navegadores incluyendo IE 6.

- Selectores CSS3: jQuery puede trabajar perfectamente con los selectores de la nueva especificación de CSS3.
- Funciones de utilidad: en jQuery se incluye un número de funciones de utilidad que no se incluyen en JavaScript.
- Plugins: el framework jQuery es extensible, esto significa que se puede ampliar para diseñar nuevos plugins que se puedan reutilizar en diferentes proyectos.
- Nos provee de un mecanismo para la captura de eventos.
- Provee un conjunto de funciones para animar el contenido de la página en forma
 - muy sencilla.
- Integra funcionalidades para trabajar con AJAX.

9.2.- Insertar Jquery en la Página Html

Lo primero que se debe hacer es descargar la librería de JQuery. El sitio oficial ofrece dos posibilidades para descargar, una llamada PRODUCTION, que es la adecuada para páginas web en producción, puesto que está minimizada y ocupa menos espacio, con lo que la carga del archivo será más rápida.

La otra posibilidad es descargar la versión DEVELOPMENT, que tiene el código sin comprimir, por lo que ocupa más espacio, pero se podrá leer la implementación de las funciones.



Una vez que se ha descargado la librería jQuery, se coloca en la carpeta raíz y lo siguiente es incluir la librería en los documentos HTML. Esto se logra haciendo referencia al archivo de script en la cabecera del documento.

```
<!doctype html>
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>jQuery</title>
    <script type="text/javascript" src="jquery.js"></script>
</head>
<body>
    <h1>jQuery</h1>
</body>
</html>
```

Por otra parte, también se puede hacer referencia a la librería directamente desde un CDN (Content Delivery Network que son servidores donde se almacenan archivos para compartir). El CDN ofrece una ventaja de rendimiento al cargar jQuery en servidores repartidos por todo el mundo, ya que si el visitante de su página web ha descargado una copia de jQuery del mismo CDN, no tendrá que volverla a descargar. Para utilizar el CDN oficial de jQuery, simplemente se hace referencia al archivo directamente desde <http://code.jquery.com> en la etiqueta de script:

```
<!doctype html>
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>jQuery</title>
    <script
        type="text/javascript"
        src="//code.jquery.com/jquery-1.11.0.min.js">
    </script>
</head>
<body>
    <h1>jQuery</h1>
</body>
</html>
```

Existen otros CDN en la web. En el siguiente ejemplo se muestran algunos CDN que también puede utilizar:

```
<!-- CDN de Google -->
<script type="text/javascript" src="//ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js" />
</script>
<script type="text/javascript" src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js" />
</script>
<!-- CDN de Microsoft -->
<script type="text/javascript" src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-2.1.1.min.js" />
</script>
```

Cuando se está trabajando localmente (desarrollo), se recomienda incluir la librería desde la ruta local del archivo .js (ejemplo: nombre-carpeta/jquery.js), una vez que el proyecto esté listo para llevarlo a producción, se debe colocar la ruta absoluta de cualquiera de los CDN nombrados anteriormente.

Esto se hace con la finalidad de agilizar los tiempos de cargas cuando se trabaja en desarrollo, ya que se accede a la librería directamente. En producción se coloca la ruta del CDN para no sobrecargar el servidor con otra petición, y si el visitante anteriormente ha visitado un sitio web que esté utilizando la misma dirección del CDN, ya tiene descargado la librería jQuery.

El CDN más utilizado y recomendado es el de Google, por tener servidores en toda parte del mundo.

9.3.- La Función Jquery()

Todas las instrucciones jQuery empiezan con la llamada a la función "jQuery()", ya que es la puerta de entrada al framework jQuery. Esto es así para evitar que los nombres de las funciones entren en conflicto con otras funciones de otras bibliotecas enlazadas en nuestro proyecto. Así podemos decir que la biblioteca está contenida en un namespace

(espacio de nombre) llamado jQuery. jQuery ofrece un alias más corto, `$()`, el signo del dólar. El alias se utiliza mucho más debido a que es más corto.

Esta función es la encargada de buscar y seleccionar los elementos del DOM que coinciden con el selector pasado como parámetro. Es una manera de encapsular los elementos DOM del documento en un objeto jQuery al que se le pueden aplicar todos los métodos definidos en el framework. La función por lo tanto devuelve un objeto jQuery con el/los elementos coincidentes. La función `$()` por sí sola no realiza nada, simplemente encapsula los elementos en objetos jQuery para poder utilizar los métodos del framework, por lo que esta función se suele utilizar en combinación con otros métodos.

Técnicamente jQuery es un objeto que define un conjunto de funciones (ó métodos) en él. La instrucción típica de jQuery sigue el siguiente esquema:

`jQuery("selector").metodo(parametro);`

es equivalente a

`$(“selector”).metodo(parametro);`

Cuando se programan ciertas acciones complejas con Javascript, es necesario que la página haya terminado de cargar y esté lista para ser manipulada, como crear elementos, quitarlos, cambiar sus propiedades, etc. Si no se espera que la página esté lista para recibir instrucciones, se obtendrán errores de Javascript en la ejecución.

Es por eso que se programa el evento "onload" del documento. En el siguiente ejemplo, se muestra un alert cuando se termina de cargar el documento.

```
window.onload = function ()  
{  
    |   alert("El documento está listo");  
}
```

No es posible interactuar de forma segura con el contenido de una página hasta que el documento no se encuentre preparado para su manipulación. jQuery permite detectar dicho estado a través de la declaración:

```
$(document).ready()
```

De forma tal que el bloque se ejecutará sólo una vez que la página esté disponible. Se trata de detectar el momento en que la página está lista para recibir comandos Javascript que hacen uso del DOM.

\$(document).ready();

Con \$(document) se obtiene una referencia al documento (la página web) que se está cargando. Luego, con el método ready() se define un evento, que se desata al quedar listo el documento para realizar acciones sobre el DOM de la página.

Luego se programa una función que se ejecute al cargar la página, como se muestra en el siguiente ejemplo:

```
$(document).ready(function()
{
    console.log('El documento está listo');
});
```

Capítulo 10. USANDO JQUERY

10.1.- Uso de Selectores

Lo más simple que podemos hacer con jQuery es obtener algunos elementos y hacer algo con ellos. Si usted entiende los selectores CSS, se le va hacer muy sencillo obtener algunos elementos, sólo tiene que pasar el selector apropiado para su posterior uso. Existen varias formas de seleccionar elementos de la página, los más usados son:

- Selector de elemento
- Selector de id
- Selector de clase

Al seleccionar un elemento, jQuery retorna un objeto que hace referencia a este. Luego de seleccionado el elemento, se puede manipular como quiera. Uno de las tareas más sencillas que se puede realizar con un elemento es esconderlo con `hide()` o mostrarlo `show()`.

Los selectores de elemento seleccionan todos los elementos que tengan una etiqueta HTML coincidente con la etiqueta pasada como argumento. Se utilizan de la siguiente forma:

`$(“elemento”)`

Internamente llama a la función `getElementsByTagName()` de JavaScript para devolver los elementos coincidentes.

```
$('h1').hide(); //Selecciona todas las estiquetas H1  
$('p').hide(); //Selecciona todas las estiquetas p  
$('div').hide(); //Selecciona todas las estiquetas div  
$('strong').hide(); //Selecciona todas las estiquetas strong
```

El selector de id selecciona el único elemento que tenga un atributo ID cuyo valor sea igual al ID pasado como argumento. Su sintaxis:

`$("#id");`

Internamente llama a la función getElementById() de JavaScript para devolver el elemento coincidente. Recuerda que el atributo id es único, por lo que no puede haber dos elementos con el mismo valor en el atributo id.

```
$('#contenedor').hide(); //Selecciona el elemento con el ID cuyo nombre es contenedor  
$('#menu').hide(); //Selecciona el elemento con el ID cuyo nombre es menu  
$('#sliders').hide(); //Selecciona el elemento con el ID cuyo nombre es sliders  
$('#barra').hide(); //Selecciona el elemento con el ID cuyo nombre es barra
```

El selector de clase selecciona todos los elementos que tengan un atributo class cuyo valor sea igual al pasado como argumento. Su sintaxis es la siguiente:

```
$(".clase");
```

Internamente llama a la función getElementsByClassName() de JavaScript si el navegador la soporta. Si no la soporta comprueba el atributo class de cada elemento para devolver los elementos coincidentes. Se permite el uso de múltiples clases. Internet Explorer 6 no soporta el selector múltiple de clases pero a través de jQuery si funciona. Aquí tenemos una de las ventajas de utilizar jQuery, la implementación de una solución cross-browser.

```
$('.lista').hide(); //Selecciona el elemento con la clase cuyo nombre es lista  
$('.icono').hide(); //Selecciona el elemento con la clase cuyo nombre es icono  
$('.items').hide(); //Selecciona el elemento con la clase cuyo nombre es items  
$('.columna').hide(); //Selecciona el elemento con la clase cuyo nombre es columna
```

Selector múltiple

Sintaxis: \$("selector 1, selector 2, selector 3");

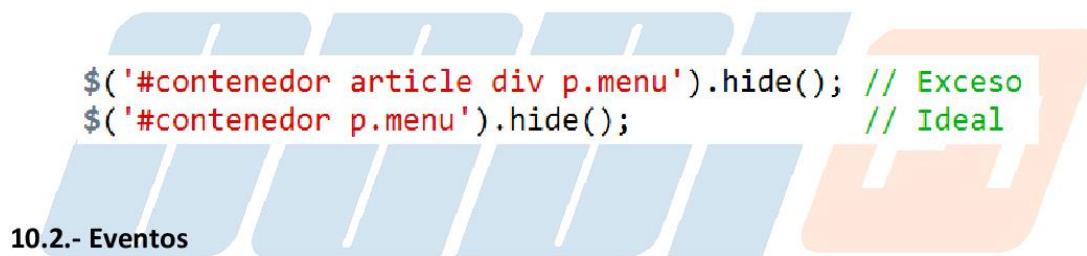
Función: selecciona y combina todos los elementos del documento que coincidan con los selectores pasados como argumento.

Descripción: se puede especificar cualquier número de selectores pero hay que tener en cuenta que es ineficiente si se quieren seleccionar en el orden en el que están situados en el documento, ya que se seleccionan en el orden en el que van coincidiendo, pudiendo darse una situación de desorden.

```
$(‘h1, #menu, .lista’).hide(); //Selecciona el elemento H1,  
// el elemento con el ID de nombre menu  
// y el elemento con la clase de nombre lista
```

La elección de buenos selectores es un punto importante cuando se desea mejorar el rendimiento del código. Una pequeña especificidad — por ejemplo, incluir el tipo de elemento (como div) cuando se realiza una selección por el nombre de clase — puede ayudar bastante. Por eso, es recomendable darle algunas “pistas” a jQuery sobre en qué lugar del documento puede encontrar lo que desea seleccionar. Por otro lado, demasiada especificidad puede ser perjudicial.

Es importante entender que cualquier selección que realice sólo contendrá los elementos que existían en la página al momento cuando hizo la selección.



jQuery provee métodos para asociar controladores de eventos (en inglés event handlers) a selectores. Cuando un evento ocurre, la función provista es ejecutada. Dentro de la función, la palabra clave this hace referencia al elemento en que el evento ocurre. La función del controlador de eventos puede recibir un objeto. Este objeto puede ser utilizado para determinar la naturaleza del evento o, por ejemplo, prevenir el comportamiento predeterminado de éste.

```
//Vincular un evento utilizando un método reducido
$('p').click(function() {
    console.log('click');
});

//Vincular un evento utilizando el método $.fn.on
$('p').on('click', function() {
    console.log('click');
});
```

Query ofrece métodos para la mayoría de los eventos:

- .click(): al hacer clic sobre un elemento.
- .submit(): cuando se pulsa sobre el botón de enviar de un formulario.
- .dblclick(): al hacer doble clic sobre un elemento.
- .hover(): cuando al pasar el ratón por encima de un elemento. Permite pasar una o dos funciones que se ejecutarán cuando los eventos mouseenter y mouseleave ocurran en el elemento seleccionado. Si se pasa una sola función, ésta será ejecutada en ambos eventos; en cambio si se pasan dos, la primera será ejecutada cuando ocurra el evento mouseenter, mientras que la segunda será ejecutada cuando ocurra mouseleave.
- .mouseenter(): cuando el cursor entra en un elemento.
- .mouseleave(): cuando el cursor sale de un elemento.
- .load() - cuando se termina de cargar el elemento.

Muy a menudo, elementos en una aplicación estarán vinculados a múltiples eventos, cada uno con una función diferente. En estos casos, es posible pasar un objeto dentro de \$.fn.on con uno o más pares de nombres claves/valores. Cada clave será el nombre del evento mientras que cada valor será la función a ejecutar cuando ocurra el evento.

```
//Vincular múltiples eventos a un elemento
$('p').on({
    'click': function() {
        console.log('clickeado');
    },
    'mouseover': function() {
        console.log('sobrepasado');
    }
});
```



Capítulo 11. MANIPULAR EL HTML CON JQUERY

11.1.- Get

Hay muchos métodos que se pueden llamar una vez que haya realizado una selección. Estos métodos generalmente se dividen en dos categorías: los captadores (getters) y definidores (setters). jQuery “sobrecarga” sus métodos, en otras palabras, el método para establecer un valor posee el mismo nombre que el método para obtener un valor. Cuando un método es utilizado para establecer un valor, es llamado método establecedor (en inglés setter). En cambio, cuando un método es utilizado para obtener (o leer) un valor, es llamado obtenedor (en inglés getter).

Los Getters recuperan un pedazo de información de la selección y setters alteran la selección de alguna manera. En casi todos los casos, los getters operan sólo en el primer elemento de una selección (.text () es una notable excepción); setters operan en todos los elementos de una selección, utilizando lo que se conoce como iteración implícita.

Iteración implícita significa que jQuery repite automáticamente todos los elementos de una selección cuando se llama a un método pionero en esa selección. Esto significa que, cuando se quiere hacer algo para todos los elementos de una selección, usted no tiene que llamar a un método definidor (setter) en cada ítems de su selección.

Existen tres métodos para obtener el contenido:

- `text ()`: Obtiene o devuelve el contenido de texto de los elementos seleccionados.
- `html ()`: Obtiene o devuelve el contenido de los elementos seleccionados (incluyendo el formato HTML).
- `val ()`: Obtiene o devuelve el valor de los campos del formulario.

```
// Obtiene el texto del elemento HTML  
// con el id con el valor texto  
$("#btn1").click(function(){  
    $("#texto").text();  
});  
  
// Obtiene el selector HTML con el  
// id con el valor texto  
$("#btn2").click(function(){  
    $("#texto").html();  
});  
  
// Obtiene el valor Cadí del elemento del formulario  
// con el id con el valor texto  
$("#btn3").click(function(){  
    $("#texto").val();  
});
```

11.2.- Set

Existen tres métodos para establecer el contenido:

- `text (nuevoValor)`: Establece o devuelve el contenido de texto de los elementos seleccionados.
- `html (nuevoValor)`: Establece o devuelve el contenido de los elementos seleccionados (incluyendo el formato HTML).
- `val (nuevoValor)`: Establece o devuelve el valor de los campos del formulario.

```
// Agrega el texto Hola mundo! al elemento HTML
// con el id con el valor texto
$("#btn1").click(function(){
    $("#texto").text("Hola mundo!");
});

// Agrega el elemento <strong>Hola mundo!</strong>
// al elemento HTML con el id con el valor texto
$("#btn2").click(function(){
    $("#texto").html("<strong>Hola mundo!</strong>");
});

// Agrega el valor Cadi al elemento del formulario
// con el id con el valor texto
$("#btn3").click(function(){
    $("#texto").val("Cadi");
});
```

Los atributos de los elementos HTML que conforman un documento pueden contener información útil, por eso es importante poder establecer y obtener esa información. El método \$.fn.attr actúa tanto como método establecedor como obtenedor.

ACADEMIA DE SOFTWARE

```
$( 'a' ).attr('href', 'contacto.html');
$( 'a' ).attr({
    'title' : 'Servicios las 24 horas del día',
    'href' : 'contacto.html'
});
```

11.3.- Add

Con jQuery, hay una manera fácil de agregar un nuevo elemento y nuevo contenido. Tenemos cuatro métodos que podemos usar para agregar contenido nuevo:

- append(): Inserta el contenido en el final del elemento seleccionado.
- prepend(): Inserta el contenido en el inicio del elemento seleccionado.
- after(): Inserta contenido después del elemento seleccionado.
- before(): Inserta contenido antes del elemento seleccionado.

```
$("h1").append(" CADI. "); // <h1>jQuery CADI </h1>
$("h1").prepend(" CADI. "); // <h1> CADI jQuery</h1>
$("h1").after(" CADI. "); // <h1>jQuery</h1> CADI
$("h1").before(" CADI. "); // CADI<h1>jQuery</h1>
```

En todos los casos, podemos insertar solo algunos elementos HTML o texto en el principio, ante, después o al final del elemento HTML seleccionado. Sin embargo, los métodos pueden tomar un número infinito de nuevos elementos pasados como parámetros. Los nuevos elementos generados con texto o HTML se pueden hacer con jQuery, o con JavaScript puro y con elementos del DOM.

```
function agregarTexto() {
    var texto1 = "<p>Texto.</p>";           // Creamos el elemento con HTML
    var texto2 = $("<p></p>").text("Texto."); // Creamos el elemento con jquery
    var texto3 = document.createElement("p"); // Creamos el elemento con DOM
    texto3.innerHTML = "Texto.";
    $("p").append(texto1, texto2, texto3);    // Agrega los nuevos elementos
}
```

11.4.- Remove

Existen dos formas de remover elementos de una página: Utilizando `$.fn.remove` o `$.fn.detach`. Cuando deseé remover de forma permanente al elemento, se utiliza el método `$.fn.remove`. Por otro lado, el método `$.fn.detach` también remueve el elemento, pero mantiene la información y eventos asociados al mismo, siendo útil en el caso que necesite reinsertar el elemento en el documento.

Por otro lado, si se desea mantener al elemento pero se necesita eliminar su contenido, es posible utilizar el método `$.fn.empty`, el cual “vaciará” el contenido HTML del elemento.

```
$('#menu').remove(); // Quita el elemento permanentemente  
$('#menu').detach(); // Quita el elemento pero mantiene información y eventos  
$('#menu').empty(); // Mantiene el elemento y vacía el contenido
```



Capítulo 12. MANIPULAR ESTILOS CON JQUERY

12.1.- Manipular Estilos Css

El método css() establece o retorna una o más propiedades del elemento seleccionado. Para retornar el valor específico de la propiedad CSS, usamos la siguiente sintaxis:

```
css("nombreNombrePropiedad");
```

Y para establecer una propiedad CSS, se usa la siguiente sintaxis:

```
css("nombreNombrePropiedad","valor");
```

Para establecer varias propiedades CSS, se usa la siguiente sintaxis:

```
css({"nombreNombrePropiedad":"valor","nombreNombrePropiedad":"valor",...});
```

```
// Retorna el valor del color del fondo  
$("p").css("background-color");  
  
// Establecemos el color del fondo en amarillo  
$("p").css("background-color","yellow");  
  
// Establecemos el color del fondo en amarillo  
// y el tamaño de la fuente a 40px  
$("p").css({"background-color":"yellow","font-size":"40px"});  
  
// Para leerlo más fácilmente, lo podemos tabular  
$("p").css({  
    "background-color":"yellow",  
    "font-size":"40px"  
});
```

jQuery incluye una manera útil de obtener y establecer propiedades CSS a los elementos.

Las propiedades CSS que incluyen como separador un guión del medio, en JavaScript deben ser transformadas a su estilo CamelCase. Por ejemplo, cuando se la utiliza como propiedad de un método, el estilo CSS font-size deberá ser expresado como fontSize. Sin embargo, esta regla no es aplicada cuando se pasa el nombre de la propiedad CSS al método \$.fn.css — en este caso, los dos formatos (en CamelCase o con el guión del medio) funcionarán.

```
// Obtenemos el tamaño de la fuente  
// del elememto con el valor texto  
// y luego lo mostramos en pantalla  
$("#btn1").click(function(){  
    var mostrar = $("#texto").css("font-size");  
    alert(mostrar);  
});
```

12.2.- Asignar Clases de Css

Si se quiere cambiar el código HTML de todos los elementos de la lista en una página se usa el método .html(), que cambia el HTML de todos los elementos de la lista seleccionados.

También se puede pasar una función con los métodos setters de jQuery. El valor de retorno de esta función, se utiliza como el nuevo valor, y recibe dos argumentos: el índice del elemento en la selección, y el antiguo valor del elemento que está tratando de cambiar. Esto es útil cuando se necesita información sobre el estado actual de un elemento con el fin de configurar correctamente el nuevo estado.

```

var $h1 = $('h1');

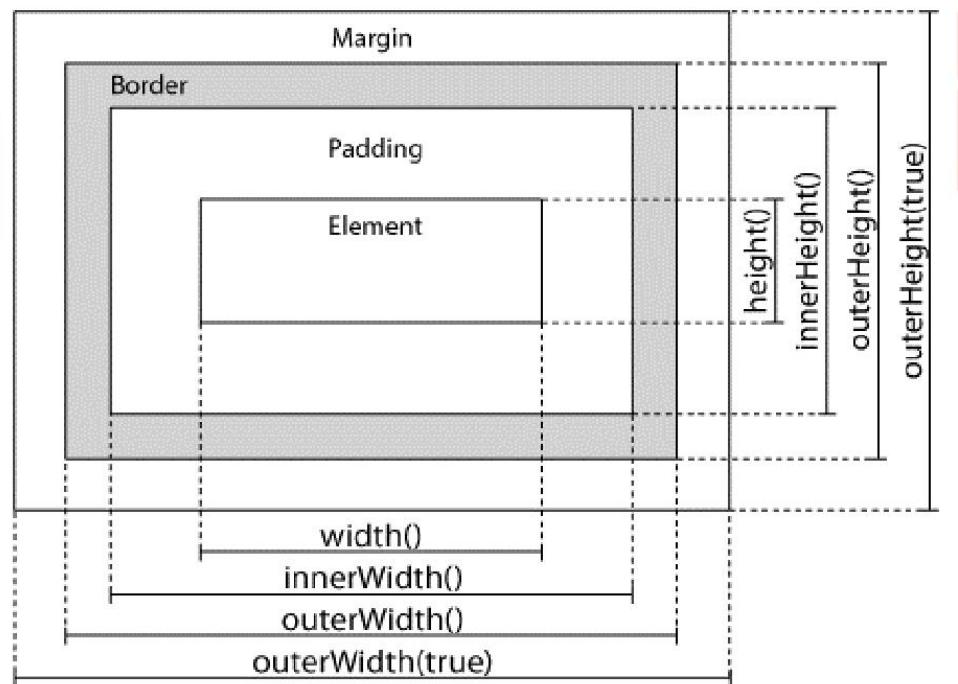
$h1.addClass('cadi');      // Añade la clase cadi
$h1.removeClass('cadi');    // Quita la clase cadi
$h1.toggleClass('cadi');    // Añade o quita la clase cadi,
                           // dependiendo del estado

if ($h1.hasClass('cadi')) { ... } // Verifica si tiene la clase cadi

```

12.3.- Manipular Tamaños

Para manipular las dimensiones de los elementos HTML es importante conocer el modelo de cajas de HTML. La siguiente imagen muestra los elementos que son parte del modelo de cajas, que son propiedades CSS de cualquier etiqueta :



jQuery tiene varios métodos importantes para trabajar con las dimensiones (tamaños) de los elementos:

- width(): establece o devuelve el ancho de un elemento (no incluye el padding, border o margin)
- height(): establece o devuelve la altura de un elemento (no incluye el padding, border o margin).
- innerWidth() devuelve el ancho de un elemento (incluyendo el padding).
- innerHeight() devuelve la altura de un elemento (incluyendo el padding).
- outerWidth() devuelve el ancho de un elemento (incluyendo padding y border)
- outerHeight() devuelve la altura de un elemento (incluyendo padding y border)

Los siguientes son ejemplos de como conocer y establecer dimensiones con JQuery:

```
$('h1').width('50px'); // establece el ancho de todos los elementos H1
$('h1').width(); // obtiene el ancho del primer elemento H1

$('h1').height('50px'); // establece el alto de todos los elementos H1
$('h1').height(); // obtiene el alto del primer elemento H1
```

En el siguiente ejemplo se muestra como usar las propiedades width, innerWidth, height e innerHeight:

```
// En este ejemplo se retorna solo el
// ancho y altura de un elemento específico
$("#inicio").click(function(){
    var anchoAlto="";
    anchoAlto+="Ancho: " + $("#caja").width() + "<br>";
    anchoAlto+="Alto: " + $("#caja").height();
    $("#caja").html(anchoAlto);
});

// En este ejemplo se retorna el ancho y la
// altura más el padding de un elemento específico
$("#inicio").click(function(){
    var anchoAlto="";
    anchoAlto+="Ancho + padding: " + $("#caja").innerWidth() + "<br>";
    anchoAlto+="Alto + padding: " + $("#caja").innerHeight();
    $("#caja").html(anchoAlto);
});
```

Capítulo 13. EFECTOS CON JQUERY. PARTE 1

13.1.- Fade

Con jQuery, agregar efectos a una página es muy fácil. Estos efectos poseen una configuración predeterminada pero también es posible proveerles parámetros personalizados. Además, es posible crear animaciones particulares estableciendo valores de propiedades CSS.

Con jQuery puede desaparecer elementos dentro y fuera de la visibilidad. jQuery tiene los siguientes métodos:

fadeIn()
fadeOut()
fadeToggle()
fadeTo()

La sintaxis para aplicar un efecto sigue la siguiente sentencia:

`$(selector).fadeIn(velocidad,disparador);`

El parámetro de velocidad es opcional y especifica la duración del efecto y puede tomar los siguientes valores: "slow", "fast", o milisegundos. La opción del disparador es una función que se ejecutará luego que el efecto sea completado.

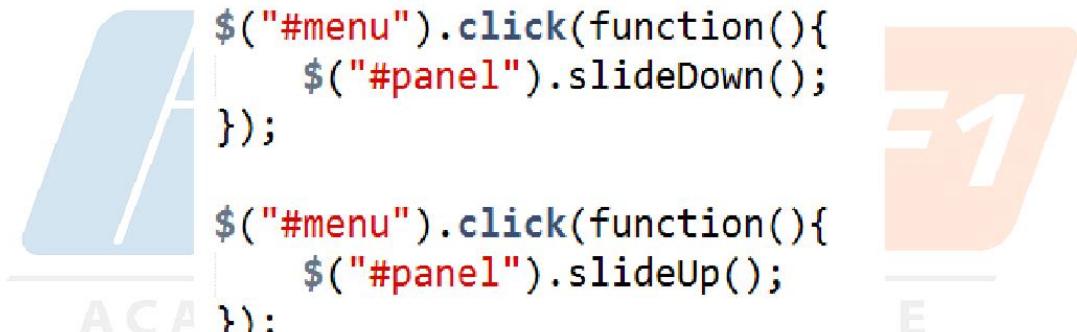
```
$("#menu").click(function(){
    $("#div1").fadeIn();
    $("#div2").fadeOut("slow");
    $("#div3").fadeIn(3000);
});
```

13.2.- Slide

Con jQuery puede crear un efecto de deslizamiento en los elementos hacia arriba y hacia abajo. jQuery tiene los siguientes métodos para los slide:

- slideDown()
- slideUp()
- slideToggle()

El método jQuery slideToggle () alterna entre los métodos slideDown () y slideUp(). Si el elemento se ha deslizado hacia abajo, slideToggle () lo deslizará hacia arriba y viceversa.



```
$("#menu").click(function(){
    $("#panel").slideDown();
});

$("#menu").click(function(){
    $("#panel").slideUp();
});

$("#flip").click(function(){
    $("#panel").slideToggle();
});
```

13.3.- Animate

El método jQuery animate() permite crear animaciones personalizadas. La sentencia es la siguiente:

```
$(selector).animate({parámetros},velocidad,disparador);
```

El parámetro define las propiedades CSS para ser animadas. El parámetro de velocidad especifica la duración del efecto. La velocidad puede tomar los siguientes valores: "slow", "fast", o milisegundos. El disparador es una función opcional que se ejecutará una vez finalizada la animación.

```
// El siguiente ejemplo demuestra un uso sencillo
// del método animado (), se mueve un elemento de
// clase .caja a la derecha hasta que haya alcanzado
// los 250px de la propiedad CSS left
$("#boton").click(function(){
    $(".caja").animate({left: '250px'});
});

// Ahora se le pasan varios parámetros
// Note que múltiples propiedades pueden
// ser animadas en el mismo momento
$("#boton").click(function(){
    $(".caja").animate({
        left: '250px',
        opacity: '0.5',
        height: '150px',
        width: '150px'
    });
});
```

También puede especificar a la propiedad los valores como "show", "hide", o "toggle".

Las propiedades relacionadas al color no pueden ser animadas utilizando el método `$.fn.animate`, pero es posible hacerlo a través de la extensión color plugin.

```
$('span').animate(  
{  
    left : "50px",  
    opacity : 0.25,  
    height : "toggle"  
}, 300, // duración  
function() {  
    console.log('realizado');  
});
```



Capítulo 14. EFECTOS CON JQUERY. PARTE 2

14.1.- Callback

Una función disparadora es ejecutada después que el efecto actual es 100% finalizado. Las sentencias de JavaScript se ejecutan línea por línea. Sin embargo, con los efectos, la siguiente línea del código se puede ejecutar a pesar de que el efecto no ha terminado. Esto puede crear errores. Para evitar esto, se puede crear una función disparadora.

```
// El ejemplo debajo tiene un disparador
// que será ejecutado después que el efecto
// de ocultamiento haya sido completado
$("#ocultar").click(function(){
    $("p").hide("slow",function(){
        alert("El párrafo ha sido ocultado");
    });
});

// A diferencia de este ejemplo que no tiene
// un disparador, y el cuadro de alerta
// se mostrará antes de que se complete
// el efecto de ocultamiento
$("#ocultar").click(function(){
    $("p").hide(1000);
    alert("El párrafo ha sido ocultado");
});
```

14.2.- Encadenamiento

Hasta ahora se ha manipulado con jQuery un elemento con una operación a la vez (uno tras otro). Sin embargo, existe una técnica llamada encadenamiento, que permite ejecutar varios comandos de jQuery, uno tras otro, en el mismo elemento(s). Los encadenamientos permiten correr múltiples métodos jQuery en una única sentencia. De

esta forma, los navegadores no tienen que encontrar el mismo elemento(s) más de una vez. Para encadenar una acción, sólo tiene que anexar la acción a la acción anterior.

```
// El elemento "#cadena" primero cambia a rojo,  
// luego se desliza hacia arriba,  
// y luego se desliza hacia abajo  
$("#cadena").css("color","red").slideUp(2000).slideDown(2000);
```

14.3.- Stop

El método stop() es usado para detener la animación o el efecto antes de que haya finalizado, trabaja sobre todas las funciones de efectos que tiene jQuery, incluyendo los sliding, fading y animaciones personalizadas. Su sintaxis es la siguiente:

```
$(selector).stop(stopAll,goToEnd);
```

El parámetro stopAll es opcional y especifica si también la cola de la animación debe ser limpiado o no. El valor predeterminado es falsa, lo que significa que sólo la animación activa se detuvo, permitiendo que cualquiera animación en la cola pueda llevarse a cabo después. El parámetro goToEnd es opcional y especifica si o no se completa la animación actual inmediatamente. El valor predeterminado es falso. Así que, por defecto, el método stop () mata a la animación actual que se realiza en el elemento seleccionado.

```
$("#boton-parar").click(function(){  
    $("#panel").stop();  
});
```

Capítulo 15. MANEJO DE FECHAS

15.1.- Date

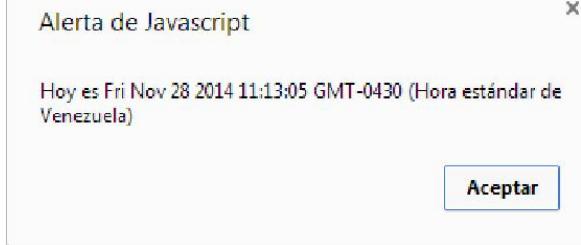
Obtener la fecha de la computadora donde se está ejecutando la página es una tarea que muy comúnmente se realiza. Para esto, se utiliza la clase Date. Esta clase al usarse posee toda la información de fecha y hora: dia, mes, año, hora, minutos, segundos y zona horaria. Para acceder a algunos de estos datos es necesario usar los métodos que posee esta clase. Los métodos que podemos usar es:

- `getFullYear()`: obtiene el año en 4 dígitos de la fecha actual.
- `getYear()`: obtiene el año en 2 dígitos de la fecha actual.
- `getMonth()`: obtiene el mes de la fecha actual.
- `getDate()`: obtiene el día de la fecha actual.
- `getHours()`: obtiene la hora de la fecha actual.
- `getMinutes()`: obtiene los minutos de la fecha actual.
- `getSeconds()`: obtiene los segundos de la fecha actual.
- `toString()`: genera toda la información de la fecha y hora y lo convierte en un string.

Un ejemplo sencillo de cómo usar la clase Date es el siguiente:

```
ACADEMIA DE SOFTWARE
<script language="javascript">
    var today = new Date();

    alert("Hoy es "+today.toString());
</script>
```



Otro ejemplo de cómo usar métodos de la clase Date:

```
<script language="javascript">
    var today = new Date();

    var today = new Date();
    var h = today.getHours();
    var m = today.getMinutes();
    var s = today.getSeconds();

    alert("Son las "+h+ ":" + m + ":" + s);
</script>
```

