CURSO DE

JAVASCRIPT

MATERIAL PARA EL ALUMNO

v.1.2: Septiembre 2018

Autores:
Maximiliano Firtman, ITMaster Academy

Edición Revisada: Maximiliano Firtman, ITMaster Academy

INDICE

INDICE **3**

INTRODUCCIÓN A JAVASCRIPT **6**

¿QUÉ ES? 6

ORIGEN 6

VERSIONES 7

INTEGRACIÓN EN UNA PÁGINA WEB **8**

Difiriendo la ejecución de código (defer) *9*

Ejecución de código asincrónico (async) *10*

TRABAJANDO CON LA CONSOLA 10

EXPRESIONES, OPERADORES Y SENTENCIAS **13**

SENTENCIAS 13

EXPRESIONES 13

Condicional if *13*

Condicional múltiple *14*

Bucle Mientras *14*

Bucle Hasta *14*

Bucle For *15*

VARIABLES **16**

Arrays *17*

AGREGADOS EN ECMASCRIPT 6 17

OPERADORES **18**

FUNCIONES **19**

TRABAJANDO CON OBJETOS **20**

OBJETOS DEL LENGUAJE 20

Array *20*

Date *20*

String *21*

DOCUMENT OBJECT MODEL (DOM) **22**

ESQUEMA 22

OBJETO WINDOW 23

OBJETO DOCUMENT 24

ACCEDIENTO A DATOS DE FORMULARIO 25

OTROS OBJETOS DEL NAVEGADOR 25
History *25*
Navigator *25*
Screen *25*
Location *26*
VALIDACIÓN DE FORMULARIOS 27
VALIDANDO CONTENIDO DE CAMPOS DE TEXTO 27
ACCEDIENDO A CHECKBOXES 28
ACCEDIENDO A RADIO BUTTONS 28
ACCEDIENDO A LISTAS Y COMBOS 29
Agregando y Eliminando Opciones *31*
JSON 32
SINTAXIS 32
VENTAJAS 34
JSON CON FUNCIONES 34
CONVIRTIENDO JSON 35
TRABAJANDO CON IMÁGENES 36
CACHÉ DE IMÁGENES 36
TRABAJANDO CON STRINGS 38
TIPS ADICIONALES 39
MODIFICANDO ESTILOS CSS 39
ACCESO A PROPIEDADES 39
REDIRECCIÓN 39
CÓDIGO CRONOMETRADO 40
REESCRIBIENDO PORCIONES DE CÓDIGO HTML 41
BUENAS PRÁCTICAS EN NOMENCLATURA 42
JQUERY 44
INSTALANDO LA LIBRERÍA 44
FUNDAMENTOS DE LA LIBRERÍA 44
FUNCIONES DE JQUERY 46
Encadenamiento de funciones *46*
Funciones de HTML *47*
Funciones de CSS *48*
Mostrando y Ocultando elementos *48*
Funciones de Eventos *49*
Evento ready *49*
AJAX 51

AJAX CON JQUERY 51
AJAX CON FETCH 52

INTRODUCCIÓN A JAVASCRIPT

¿Qué es?

JavaScript es un lenguaje de programación basado en scripts. Esto significa que, a diferencia de otros lenguajes, JavaScript no se compila ni genera un archivo ejecutable (como un .exe de Windows). El código fuente de este lenguaje se distribuye con la aplicación, por ejemplo web y, en lugar de compilarse, se interpreta al momento de ejecutarse. De esta forma, los errores se encuentran recién al momento de ejecutar la aplicación.

JavaScript nos permite crear programas que se ejecutan en:

- Una página web, dando interactividad a HTML y CSS
- Una Progressive Web App: una app para Android, iPhone, Mac y Windows
- Un servidor o una computadora, usando Node.js
- Una casa inteligente, como un televisor o un Raspberry.pi para controlar el hogar
- Robots y pequeños controladores

Importante: JavaScript no es Java: Java sí es un lenguaje compilado y si bien tienen nombres parecidos no tienen relación. Son lenguajes completamente distintos, para fines distintos y es importante no confundirlos.

Origen

En la web, al comienzo, las páginas eran solamente archivos HTML estáticos, sin posibilidad alguna de interacción con el usuario o de programación. Todo el contenido estaba "estampado" sobre la página web. Luego de unos años, los desarrolladores se sintieron muy limitados con HTML y allí surgió JavaScript junto con el navegador Netscape Navigator.

Hoy el nombre oficial del lenguaje es ECMAScript (ES) pero se lo conoce más cotidianamente por JavaScript (JS) que es una marca registrada de Oracle, los mismos que tienen Java. Por lo tanto, ECMAScript y JavaScript son el mismo lenguaje.

Versiones

Es común hablar de versiones de ECMAScript en lugar de JavaScript. Hoy las versiones más comunes son ECMAScript 5 y ECMAScript 6 (ES6 o ES2015), ésta última lanzada en 2015 y hoy ya soportada por casi todos los navegadores, aunque hoy se está publicando una versión por año, por ejemplo ES8/ES2017 publicada en 2017, pero al inicio no soportada por todos los navegadores.

Qué se puede hacer y qué no en el navegador

Con JavaScript trabajando en una página web podremos:

- Validar el ingreso de datos en un Formulario
- Escribir HTML en forma dinámica, que cambie según ciertas variables
- Cambiar las propiedades de objetos HTML
- Reaccionar ante algunos eventos que ocurran en una página web
- Saber información del teléfono o computadora del usuario
- Guardar información persistente en el dispositivo

Con JavaScript NO podremos:

- Acceder a Bases de Datos en un servidor
- Ejecutar código en el servidor
- Enviar e-mails
- Ocultar nuestro Código Fuente
- Compilar nuestro Código Fuente
- Acceder a archivos del disco del usuario o impresora en forma directa

INTEGRACIÓN EN UNA PÁGINA WEB

Cómo comentamos, JavaScript funciona dentro de una página HTML, por lo que debemos escribir el código dentro del archivo fuente HTML. El código JavaScript debe ser encerrado entre un tag HTML llamado <script>. Por ejemplo:

```
<!doctype html>
<html>
<head>
 <script>
    alert('Hola mundo!');
 </script>
</head>
<body>
Primera Prueba de JavaScript
</body>
</html>
```

Como podemos ver en el ejemplo, El código alert('Hola mundo!') se encuentra entra una marca HTML de apertura y una marca de cierre. La función alert nos mostrará un mensaje en una ventana.

Todo el código hasta llegar al tag </script> será considerado por el navegador como código programado en lenguaje JavaScript. Podremos crear tantas etiquetas <script> como cantidad de código queramos intercalar en nuestra página web.

Uniendo JavaScript

Además de poder insertar varios tags <script> dentro de un HTML, tenemos la posibilidad de utilizar archivos externos que posean el código JavaScript aparte. De esta forma, podremos utilizar el mismo código en varios archivos HTML sin tener que copiar todo el código en cada uno. En el caso de alguna modificación en el código estará disponible para todos los HTML.

El tag para leer archivos externos es el mismo, con la propiedad SRC indicando el nombre del archivo, generalmente con extensión .js

```
<script src="micodigo.js"></script>
```

El archivo micodigo.js deberá ser un archivo de texto plano con el código que queremos ejecutar sin etiquetas HTML.

Algunas ventajas muy importantes de utilizar código externo son:

- **Reutilización de código:** diferentes documentos pueden utilizar el mismo código y, además, una vez cargado en el caché producirá una carga más rápida para todo documento que lo utilice.

- **Fácil manutención:** el tener el código en módulos facilita enormemente la manutención del código y trabajar de manera independiente diferentes áreas de nuestra aplicación y tener separadas la presentación de la lógica.

Si se definiera la variable en un archivo .js directamente dentro de la etiqueta <script>, esta variable estaría disponible para cualquier otro archivo JS que se cargue posteriormente.

Difiriendo la ejecución de código (defer)

Para diferir la ejecución de un código JavaScript hasta que la página HTML se haya cargado y desplegado por completo, se puede agregar el siguiente modificador: defer.

```
<script defer src="hola_mundo.js" ></script>
```

Para los navegadores que soportan HTML5, esta opción está disponible solamente para código que se carga externamente.

Más recomendable que utilizar esta opción, es colocar el código que quiera diferirse al final del HTML. Así, se garantiza este comportamiento en cualquier versión de navegador y tanto para el código integrado al documento como para el que se carga externamente.

Ejecución de código asincrónico (async)

Esta opción aplica solamente a código cargado externamente, cuando el intérprete se encuentra con esta opción, no espera que el código JavaScript se cargue y ejecute para seguir representando el resto del documento. Esta opción es soportada por los navegadores: Chrome, Opera, Firefox y de Internet Explorer 10 en adelante.

```
<script async src="hola_mundo.js" ></script>
```

Trabajando con la Consola

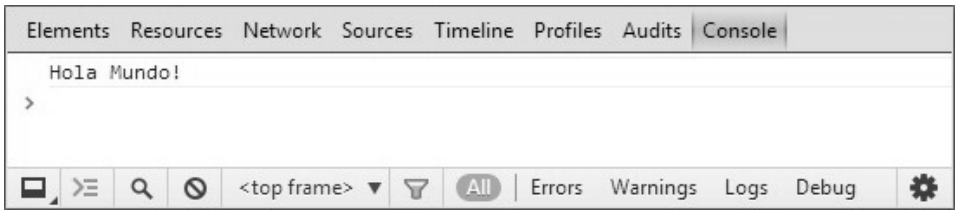
Mientras trabajamos, utilizaremos un método de JavaScript para mostrarlo en la consola de depuración. En cualquier navegador con una página abierta, al hacer clic en F12 se abre una ventana para depuración de código. Una vez abierta la ventana, buscamos en ella el botón **Consola** (o *Console* si lo tenemos en inglés) y le damos clic para abrir la consola propiamente dicha y ver allí los mensajes. Para enviar los mensajes, utilizaremos el método: **console.log.**

Empleando este método, los mensajes solamente serán visibles en la consola y el usuario no verá ningún tipo de representación visual del mensaje en el HTML.

Utilizamos este código:

```
console.log("Hola Mundo!");
```

Tenemos por resultado en Chrome:



El colocar el código en el <head> nos asegura que el mismo estará disponible para cuando se cargue el <body>.

Manejando eventos

Si bien en JavaScript podremos escribir código y será ejecutado en forma secuencial como cualquier lenguaje, la potencia de JavaScript radica en trabajar con capturadores o manejadores de eventos, que son funciones que se ejecutarán cuando hay algún cambio de estado en el HTML u ocurre algo, por ejemplo: se presiona un botón, se cierra la página, o se selecciona una opción de una lista.

La lista de eventos posibles y los objetos que permiten capturarlos cambia de versión a versión de los navegadores. Las más comunes son las siguientes:

Evento	Descripción	Objetos que permiten su captura
onload	Se ejecuta al cargarse el elemento, como por ejemplo al cargar la página	Todos, en general se usa sobre el <body>
onbeforeunload	Se ejecuta al cerrarse la página	<body>
onmouseover	Se ejecuta al pasar el mouse sobre el elemento	Todos, como <a>, <img>, <section>, <button>
onmouseout	Se ejecuta, luego de onMouseOver, cuando el mouse sale del foco del elemento	Todos, como <a>, <img>, <section>, <button>
onclick	Se ejecuta al realizar clic sobre un objeto	Todos, como <a>, <button>, <input>
onchange	Al cambiar el contenido de un objeto	Típicamente elementos de formulario, como <input>, <select>, <textarea>

Existen muchos eventos más para cada tipo de elemento HTML.

Con los eventos podremos definir una función, por ejemplo, que se ejecutará cuando un usuario haga clic en un botón. Si quisiéramos realizar un botón que diga PRESIONAME y que se ejecute el mensaje de Hola Mundo visto previamente deberíamos utilizar el siguiente código:

```
<script>
  function alerta() {
    alert('Hola mundo!');
```

```
  }
 </script>
<form>
<input type="button" value="PRESIONAME" onclick="alerta()">
</form>
```

Como vemos, se asigna el nombre de la función dentro del tag HTML a capturar y se le agregan paréntesis al final () que significa “ejecutar la función alerta”).

EXPRESIONES, OPERADORES Y SENTENCIAS

Sentencias

La sintaxis de las expresiones JavaScript es similar a C, Java y PHP. Permite acceder a objetos y sus métodos y propiedades a través del punto "." y los strings o cadenas de caracteres permite trabajarlos con comillas simples (') o comillas dobles ("), pero no ambas simultáneamente. Las sentencias JavaScript deberían finalizar con punto y coma (;) pero, al no ser un lenguaje estricto puede prescindir de ellas. Las condiciones lógicas deben siempre definirse entre paréntesis ( ). Por último, podemos mencionar que las estructuras complejas como funciones, while, if se deben ingresar entre llaves ({ }).

JavaScript es un lenguaje CASE SENSITIVE. Por lo tanto, debemos tener mucho cuidado con el uso de mayúsculas y minúsculas, tanto en el nombre de funciones, estructuras del lenguaje, variables y nombres de eventos.

Los comentarios en JavaScript se definen por línea utilizando doble barra // delante. También es posible definir comentarios de múltiples líneas utilizando /* para abrir y */ para cerrar el comentario.

Expresiones

Veamos las estructuras con ejemplos:

Condicional if

```
if (nombre == "Raul") {
  alert('Hola Raúl');
} else {
  alert('Quién sos?');
}
```

Condicional múltiple

```
switch(nombre) {
  case "Raúl":
    alert('Hola Raúl');
    break;
  case "José":
    alert('Hola José');
    break;
  default:
    alert('¿Quién sos?');
}
```

Bucle Mientras

```
var num=0;
while (num<=2) {
  alert('Hola!!');
  num++;
}
```

Bucle Hasta

```
var num=0;
do {
  alert('Hola!!');
  num++;
} while (num <= 2);
```

Bucle For

```
for (var i=0; i<20; i++) {
  console.log(“El número actual es “ + i);
}
```

En el bucle for indicamos entre paréntesis, las sentencias de inicio, la condición de salida del bucle y la condición que se va a repetir luego de cada ciclo.

For, en JavaScript, nos provee también de otra construcción llamada for in, que nos permite iterar entre todas la propiedades de un objeto, en el caso de una persona, por ejemplo:

```
for (var i in vector) {
   sum = sum + vector[i]
}
```

Desde la versión ECMAScript 6 (ES6), también se incorpora la construcción for of que nos permite iterar entre una colección de elementos, por ejemplo:

```
for (var propiedad of persona) {
   console.log(“La persona tiene “ + propiedad);
}
```

VARIABLES

Una variable es un lugar en la memoria del dispositivo donde guardamos información, puede ser un número, un nombre, datos de un producto que vendemos, o cualquier otra cosa.

A las variables en JavaScript no es necesario asignarles qué tipo de datos tendrán dentro. De esta forma, con sólo utilizar o asignar un valor a una variable por primera vez, ésta será creada del tipo que le asignemos

Típicamente, declaramos variables en JavaScript usando la instrucción “var”. Si usamos el igual (=) podemos asignar un valor en el mismo momento que creamos la varilable, más allá que podemos cambiarle su contenido luego.

```
var numero1;
var numero2;
var nombre = “ITMaster”;
```

Las variables viven en dos posibles contextos: global o función. Si usamos var fuera de cualquier función, simplemente en el código JavaScript, los contenidos de esas variables van a existir y pueden ser accedidos desde cualquier lado del código hasta que se cierre el navegador, el tab o se refresque la página.

Si usamos var adentro de una función, esa variable será accedida sólo dentro de esa función

```
var nombre = “Este dato es global, se puede usar en todo el código”;

function miFuncion() {
   var texto = “Esta variable sólo sirve dentro de la función”;
}
```

Las variables pueden guardar números (number), textos (string), datos buleanos si/no (boolean), funciones, objetos (objects) y colecciones de datos (arrays).

Arrays

Algunos tipos de dato más complejos son objetos y éstos se inicializan con new, como ser los vectores o arrays, indicándole entre paréntesis la cantidad de elementos.

```
var paises = new Array(20);
```

podremos acceder a cada posición del vector indicando su posición con corchetes[]. En JavaScript no hace falta definir el largo de un vector; se lo puede definir como new Array() y luego, ir asignando valores a cada posición; JavaScript automáticamente generará el tamaño adecuado y lo irá cambiando durante la ejecución.

Los vectores en JavaScript comienzan desde la posición 0, hasta la longitud-1, por ejemplo de 0 a 19 serían 20 elementos.

Agregados en ECMAScript 6

La versión ES6 agrega dos instrucciones: let y const. Con let también creamos variables pero en el bloque de código más cercano. Por ejemplo, si creamos una variable con let dentro de un if o un for sólo es accesible dentro de ese bloque de código. Con const creamos constantes, son posiciones en memoria que una vez que le ponemos un valor, no se puede cambiar.

```
const porcentajeIVA = 21;
if (porcentajeIVA>19) {
    let mensaje = “El IVA es alto”;
}
```

OPERADORES

Los operadores matemáticos disponibles son +, -, *, /, % (módulo), ++ (incremento), -- (decremento). Por ejemplo, al indicar **num++** equivale a indicar **num = num + 1**. El operador para asignar es el igual (=) definiendo: nombre_de_variable = valor. También se puede asignar utilizando += y sus equivalentes para otras operaciones. Por ejemplo, a += b equivale a realizar a = a + b.

Los operadores de comparación, para utilizar por ejemplo, en una condición lógica son: <, >, <=, >=. La igualación se trabaja con un doble carácter igual (==) y el distinto con !=. JavaScript es poco estricto con los tipos de dato, de esta forma si igualamos un número 2 con un string conteniendo un 2 nos dará verdadero ya que el contenido es el mismo. Existe otro comparador, llamado igualdad estricta, que se define con triple carácter igual (===) que, además, compara los tipos.

Por último, veremos los operadores lógicos. And se representa con &&, Or con || (pipes) y Not con !.

FUNCIONES

Las funciones se definen en JavaScript en cualquier porción del HTML y puede ser invocada en cualquier tag <script> o en la captura de un evento. Todos los códigos que pertenezcan a un mismo HTML comparten la definición de un programa JavaScript. No pueden existir dos funciones con el mismo nombre dentro de un mismo HTML. Se utiliza la sentencia return para devolver un valor en la función.

Las funciones se escriben de la siguiente forma:

```
function sumaUno(num) {
  return num + 1;
}
```

No todas las funciones están obligadas a retonar un valor.

Para invocar a una función que recibe argumentos usamos el nombre de la función paréntesis y los valores dentro de los paréntesis, por ejemplo:

```
var valor = sumaUno(12);
// Nuestra variable tendrá un 13
```

Existen también las funciones literales o funciones anónimas, que no llevan nombre y se envían o asignan en el momento de querer ser usadas, por ejemplo:

```
setTimeout( function() {
  // Esta es una función anónima
}, 100);
```

TRABAJANDO CON OBJETOS

Para trabajar con objetos podremos trabajar con sus propiedades (variables) o métodos (funciones). Para ambos se accede mediante el nombre del objeto seguido de un punto "." y el nombre de la variable o método. Por ejemplo:

```
var form = document.getElementById(“form1”);
form.getElementById(“cantidad”).value = 0;
form.submit();
```

Este ejemplo pone en 0 un campo de texto llamado cantidad en un formulario HTML con id form1 y luego acciona el formulario con el método submit. Los nombres cantidad y form1 son los que les dimos a los objetos dentro del HTML con la propiedad "id" de cada tag.

Objetos del Lenguaje

Array

Posee atributos como length para conocer la cantidad de elementos actuales de un vector y métodos como sort que ordena alfabéticamente el contenido.

Date

Permite crear un objeto de tipo fecha. Por ejemplo

```
var fecha = new Date();
var fecha = new Date(ano, mes, dia);
```

Luego con métodos como setMonth(), setDay(), setYear() se podrán definir cada valor y con las variantes get de las funciones se podrán consultar los valores de una fecha. También se pueden definir horas, minutos y segundos.

String

En JavaScript los String son objetos y solemos trabajarlos con literales, usando comillas simples o dobles. Una vez que tenemos un objeto de este tipo tenemos acceso a varias funciones, como

charAt(posición) que nos devolverá el carácter en la posición indicada; **indexOf(abuscar)** que devuelve la posición de un substring abuscar dentro del string al que invocamos**; substr(indice, longitud)** que devuelve un substring con el índice como posición de comienzo.

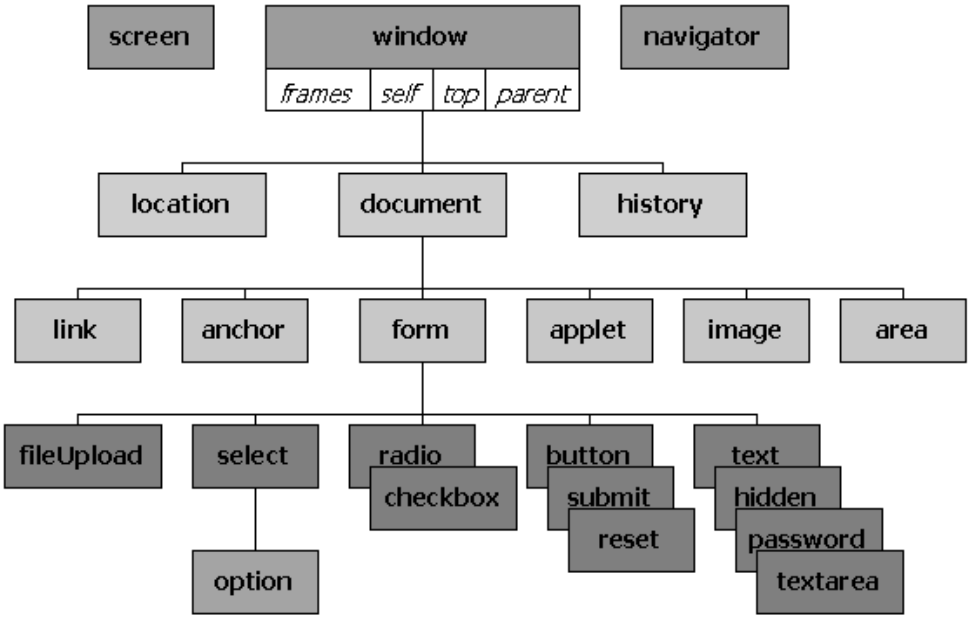
DOCUMENT OBJECT MODEL (DOM)

Esquema

Como estamos trabajando con una página web, los objetos a los que podremos referenciar son los elementos que pueden estar disponibles en una página web. Algunos de ellos son únicos en un mismo html, como ser document que referencia al documento HTML en sí (etiqueta body), window que representa a la ventana y otros pueden ser múltiples, como ser formularios, imágenes o botones. En éstos casos accederemos a ellos a través de un nombre que definiremos dentro del tag html con la propiedad "id", por ejemplo:

```
<img src="prueba.gif" id="logo">
```

El esquema de objetos disponibles suele ser parecido al siguiente:



Objeto Window

Con este objeto podremos manejar las propiedades y métodos disponibles para una ventana, por ejemplo, su ancho, alto; podremos generar y abrir ventanas nuevas, así como cerrarlas e imprimirlas. Al ser window la clase madre de todas, podemos invocar a sus métodos sin indicar window delante.

Algunos ejemplos:

window.open(url, nombre, propiedades);

Abrirá una ventana en la dirección designada en url, con un nombre y las propiedades invocadas dentro de un string. Por ejemplo,

```
window.open("popup.htm","ventana",
"toolbar=no,directories=no,menubar=no,status=yes");
```

Las propiedades disponibles son:

- toolbar[=yes|no]
- location[=yes|no]
- directories[=yes|no]
- status[=yes|no]
- menubar[=yes|no]
- scrollbars[=yes|no]
- resizable[=yes|no]
- width=pixels
- height=pixels

Es importante invocar a esta función luego de una acción del usuario, como por ejemplo el click de un botón porque los navegadores modernos bloquean la apertura de estas ventanas popup.

window.alert(texto)

Nos permite mostrar una ventana nueva de alerta por pantalla.

var respuesta = window.confirm(texto)

Nos permite mostrar un texto y que el usuario seleccione Ok o Cancelar para confirmar una acción. Este método nos devuelve true o false, según la selección del usuario.

var respuesta = window.prompt(texto, inicial)

Este método nos permite pedirle al usuario que ingrese cierto valor. Se ofrecerá el mensaje como texto de referencia y si indicamos un string inicial será el cargado de forma predeterminada

Objeto Document

Este objeto también es único como Window y nos permitirá acceder a funciones útiles para el contenido del documento (dentro de la ventana). El método más antiguo es write. Este método nos permitirá escribir directamente a la salida HTML. Es decir, escribiremos código html que se analizará por el navegador en el momento que invoquemos a este método. De esta forma, podremos escribir código HTML de forma dinámica utilizando JavaScript.

```
document.write('<b>Hola Mundo JavaScript</b>');
```

Lo más usual es interactuar con el documento HTML a través de los tags para luego manipularlos. Los usos más comunes son buscar un elemento en el HTML por su atributo id (getElementById), o buscarlo por un selector CSS (getQuerySelector), por ejemplo:

```
// busca un elemento con id=”logo” en el HTML y nos deja manipularlo
// en JavaScript
var logo = document.getElementById(“logo”);

// busca un elemento video con class=”trailer” usando selector CSS
var video = document.querySelector(“video.trailer”);

// cuando buscamos muchos elementos, en este caso todos los <li>
// que pudieran existir dentro de un elemento con id menu
var elementosLista = document.querySelectorAll(“#menu li”);
```

Accediento a datos de formulario

Para acceder a los valores que posean los campos de texto o listas de un formulario, podemos hacerlo a través del nombre del campo definido en la propiedad id. Por ejemplo para la etiqueta <input type="text" name="nombre" id=”nombre”>, podremos acceder al valor que haya ingresado el usuario referenciando a:

```
var nombre = document.getElementById(“nombre”).value
```

Los elementos que sean del tipo radio o checkbox podrán verificarse con la propiedad lógica **checked** que nos dirá si un elemento está activado o no. Los elementos tipo lista (tag select), nos permitirán acceder a cada una de las opciones con un vector llamado options y la opción seleccionada por el usuario la conoceremos utilizando **selectedIndex**. También es posible leer el value.

Otros objetos del navegador

History

Nos permitirá acceder a propiedades del historial del navegador, por ejemplo **history.back()** retrocederá al navegador a la página inmediata anterior que el usuario navegó. **history.go(-2)** retrocederá 2 páginas.

Navigator

Nos permitirá conocer detalles sobre el navegador sobre el cual estamos ejecutando la aplicación. Por ejemplo: **navigator.appName** nos dará el nombre y **navigator.appVer** su versión.

Screen

Nos permitirá acceder a tres propiedades de la pantalla del usuario: height (alto) y width (ancho)

Location

Este objeto nos permitirá manejar la ubicación de la barra de direcciones del navegador. Es un objeto dependiente de document, por lo que lo accederemos con document.location. Por ejemplo: **document.location.href** nos dará la dirección actual de la página o **document.location.host** que nos dará sólo la parte del servidor de la URL.

VALIDACIÓN DE FORMULARIOS

Una de las aplicaciones más útiles de JavaScript y las páginas web es poder validar los datos ingresados en un formulario. De esta forma, podemos comprobar si un campo de un formulario HTML fue completado o si tiene el mínimo requerido, antes de proceder a enviar la información a un servidor.

La propiedad a la que más accederemos de un campo de texto es value. Esta propiedad nos permite leer qué valor ha ingresado el usuario, así como poder cambiar el valor que tiene un campo de texto.

Sugerencia: Todos los objetos que estén en un formulario tienen un método llamado focus() que nos permitirá que el foco de la página y el cursor del usuario se pare sobre el objeto sobre el cual lo estamos invocando. Es útil para mostrar dónde el usuario se ha equivocado, por ejemplo.

Validando contenido de campos de texto

Por ejemplo, el siguiente código validaría si el campo de texto está vacío:

```
if (document.getElementById(“apellido”).value == '') {
  alert('El apellido está vacío')
}
```

También podremos acceder a valores como size (tamaño en ancho), maxLength (cantidad máxima de caracteres que puede leer), por ejemplo:

document.getElementById(“apellido”).size = 20;

document.getElementById(“apellido”).maxLength = 40;

Los campos que son de tipo password tienen las mismas propiedades y métodos que un campo de tipo texto ya que se basan en el mismo concepto.

Los campos hidden sólo permiten acceder a la propiedad value para leerlo o modificarlo. Recordemos que cualquier cambio en este campo es invisible al usuario y nos permitirá enviar información oculta al servidor.

Accediendo a Checkboxes

Los checkbox son los casilleros de selección que nos permitirán saber si una opción está o no seleccionada. También este tipo de campo se lo accede a través de su nombre, y la propiedad que verificamos es checked, que nos dará true si el campo está seleccionado y false en caso contrario.

Por ejemplo, suponiendo:

```
<input type="checkbox" name="acepta" value="si" id="acepta">
<label for="acepta">Acepto las condiciones</label>
```

El siguiente código JavaScript verificaría si está seleccionado o no

```
if (document.getElementById(“acepta”).checked) {
   alert('Está seleccionado')
} else {
   alert('No está seleccionado')
}
```

Accediendo a Radio Buttons

Los radio buttons son los casilleros de selección donde podremos seleccionar una opción entre varias disponibles. Por ejemplo, el sexo del usuario podría ser un objeto de este tipo, definido de la siguiente forma:

```
<input type="radio" name="sexo" value="F" id=”sexoF”>
<label for=”sexoF”>Femenino</label>

<input type="radio" name="sexo" value="M" id=”sexoM”>
<label for=”sexoM”>Masculino</label>
```

Los radiobuttons, al tener más de una opción dentro del mismo nombre (en este caso, name), se acceden a través de cada elemento.

El siguiente ejemplo, verificaría si está seleccionada la primera opción (femenino):

```
if (document.getElementById(“sexoF”).checked) {
   alert('Femenino')
}
```

Accediendo a Listas y Combos

Las listas, permiten que el usuario seleccione una opción o más (en el caso de listas múltiples) de un listado predefinido de opciones. El tag que administra las listas es SELECT, pero, a su vez, existe un tag HTML que maneja cada una de las opciones disponibles para una lista.

El siguiente es un ejemplo de lista:

```
<select name="continente" id="continente">
  <option value="AM">America</option>
  <option value="AF">Africa</option>
  <option value="EU">Europa</option>
  <option value="AS">Asia</option>
  <option value="OC">Oceanía</option>
</select>
```

Atención: Recordemos que, para el formulario el valor de cada opción es lo definido en value (en este ejemplo, las dos letras) y no el texto que sí se muestra al usuario.

Podremos acceder a las propiedades y métodos de la lista a través de su nombre, en este caso: continente.

En nuestro ejemplo, sería document.getElementById(“continente”) o document.querySelector(“select#continente”)

Las listas poseen la propiedad length que nos dará la cantidad de opciones que posee y options nos da un vector (array) con todas las opciones disponibles. Otra propiedad muy útil es selectedIndex que nos devolverá la opción seleccionada actualmente. En el caso de que no exista ninguna opción seleccionada, nos devolverá -1.

Si accedemos luego a options[0], options[1], etc. podremos acceder a cada opción en particular que poseen sus propiedades. Cada opción representa a cada tag OPTION. Las propiedades más útiles son value que nos dará el valor que posee esa opción, text que posee el texto que esa opción muestra al usuario (recordemos que puede ser distinto a value) y selected que nos indica si está actualmente seleccionado o no. Selected es útil para listas múltiples donde podremos recorrer todos los elementos del vector options y verificar cuáles están seleccionados.

Por ejemplo, podremos mostrar en una alerta qué continente ha seleccionado el usuario con el siguiente código.

```
var actual = document.getElementById(“continente”).selectedIndex;
alert(document.getElementById(“continente”).options[actual].text);
```

De la misma forma, podemos hacer un combo de menú de links, por ejemplo:

```
<form name="form">
<select id="link" onchange="navegar()">
   <option value="/">Escoja una sección</option>
   <option value="/noticias">Noticias</option>
   <option value="/deportes">Deportes</option>
   <option value="/clima">Clima</option>
   </select>
</form>
<script>
function navegar() {
   window.location = document.getElementById(“link”).value;
}
</script>
```

Agregando y Eliminando Opciones

JavaScript nos da la posibilidad de agregar y eliminar opciones de una lista o combo cuando la página está ya cargada.

Para crear una opción debemos utilizar un constructor de objetos que lleva una sintaxis como el siguiente ejemplo:

```
var nuevaopcion = new Option ('Antártida', 'AN', false, false);
```

En este ejemplo, estamos creando una nueva opción llamada Antártida que poseerá valor AN y que no estará seleccionada por defecto, ni seleccionada actualmente en la lista.

Una vez creada la opción la tenemos en una variable y no dentro de nuestra lista (recordemos que podemos tener más de una lista). Para agregarla al final, debemos hacerlo en la posición que nos devuelva length (recordemos que comienzan desde 0 los vectores), por ejemplo:

```
var posicion = document.getElementById(“continente”).length;
document.getElementById(“continente”).options[posicion] = nuevaopcion;
```

Para eliminar una opción de la lista, debemos asignar null a la opción, por ejemplo, si quisiéramos eliminar la opción Europa (posición 3), deberíamos utilizar el siguiente código:

document.getElementById(“continente”).options[2] = null;

Al ejecutar esta sentencia, las opciones que estaban luego de la eliminada decrementan en uno su posición.

JSON

JSON significa JavaScript Object Notation, o notación de objetos de JavaScript, y se pronuncia como el nombre inglés Jason. En países de habla hispana también es común nombrarlo en español como Jota-son. No es más que una forma muy potente de definir objetos y su contenido, que tomó un significado nuevo a partir de las aplicaciones AJAX.

Su notación es tan sencilla que en la actualidad se utiliza en muchos lenguajes para transporte liviano de objetos, incluyendo PHP, C#, Java, Visual Basic y Objective-C.

Sintaxis

Un objeto con notación JSON está encerrado entre llaves {} y contiene propiedades separadas por comas, cuyos nombres deberían estar encerrados entre comillas (aunque la mayoría de las implementaciones los acepta sin ellas). Cada propiedad tiene un valor separado por dos puntos (:) y cada valor puede ser:

- Un string literal encerrado entre comillas.
- Un número.
- Un valor lógico true o false.
- Un array.
- Otro objeto JSON.
- Una función (compatible sólo con JavaScript).

Veámoslo mejor en un ejemplo:

```
{
    "nombre": "Lionel",
    "apellido": "Messi",
    "edad": 35
}
```

El estándar estricto indica que los nombres de las propiedades deben estar encerrados entre comillas, aunque a esta altura de la evolución de los browsers se puede prescindir de ellas en situaciones controladas:

```
{
    nombre: "Lionel",
    apellido: "Messi",
    edad: 35
}
```

Es probable que esta sintaxis nos recuerde la utilizada en las hojas de estilo CSS. Es similar, sólo que las propiedades se separan por comas y no por punto y coma y, además, no se coloca la última coma.

Un vector se puede enunciar con rapidez entre corchetes [ ] con sus elementos separados por coma. Por ejemplo:

```
{
    nombre: "John",
    apellido: "Doe",
    edad: 25,
    hijos: ["Mary", "Sean"]
}
```

Asimismo, una propiedad puede contener otro JSON abriendo otro par de llaves { }. Por ejemplo:

```
{
    nombre: "John",
    apellido: "Doe",
    edad: 25,
    hijos: ["Mary", "Sean"],
    pareja: {
        nombre: "Lisa"
    }
}
```

Todos los espacios, las tabulaciones y los saltos de línea son opcionales, pero simplifican la lectura.

A veces, también se considera JSON cuando se define sólo un vector. Por ejemplo:

```
var paises = ["Argentina", "México", "España"];
```

Asimismo, cada elemento de un vector en un JSON puede ser otro JSON.

```
var carreras = [ {nombre: "Programador Android", duracion: 4},
                 {nombre: "Programador Web", duracion: 5}
];
console.log(carreras[0].nombre); // Imprime Mobile
```

Ventajas

Las ventajas son muchas: es sencillo y rápido de escribir, su lectura es comprensible y se puede navegar por el objeto con notación de punto, como cualquier objeto JavaScript:

```
var cliente = {
    nombre: "John",
    apellido: "Doe",
    edad: 25,
    hijos: ["Mary", "Sean"],
    pareja: {
        nombre: "Lisa"
    }
};
console.log("El cliente " + cliente.nombre + " tiene " +
    cliente.hijos.length + " hijos y su pareja se llama " +
    cliente.pareja.nombre);
```

Un JSON, como todo objeto JavaScript, puede recorrerse con un for in para saber todas las propiedades que posee.

JSON con funciones

Un JSON, como no podía ser de otra manera, también puede contener funciones, que pueden transportarse como cualquier otra variable mientras que tanto remitente como receptor de la función sea JavaScript:

```
var cliente = {
    nombre: "Juan",
    limpiar: function() {
        this.nombre = "";
    }
}
cliente.limpiar();
```

Esta metodología se usa para encapsular funcionalidad en un objeto en lugar de crear funciones que son globales al contexto de ejecución. Esto emula el concepto de paquetes o nombres de espacio en otros lenguajes. Por ejemplo:

```
var FuncionesUtiles = {
    funcion1: function() { },
    funcion2: function() {}
}
```

Convirtiendo JSON

Los navegadores modernos incluyen una metodología simple para convertir un objeto en memoria al formato JSON (texto) y un texto en formato JSON a su objeto nuevamente. Para ello, existe un objeto JSON con los métodos stringify (traducido, algo así como 'convertir a texto') y parse respectivamente.

```
var textoConJSON = JSON.stringify(objeto);
var objetoRestaurado = JSON.parse(textoConJSON);
```

De esta manera podemos recibir un archivo en formato JSON y en lugar de tratarlo como texto, lo convertimos a un objeto que pueda ser manipulado con sintaxis de punto.

TRABAJANDO CON IMÁGENES

A las imágenes, como a los formularios, podremos accederlas mediante un nombre que le definamos en la propiedad "id" de cada una. Podremos cambiar sus propiedades, como ser el archivo al que apunta la imagen (y generar así un efecto roll-over) o cambiar su tamaño.

Por ejemplo:

```
<img src="logo.gif" id="logo">
```

Aquí tenemos una imagen a la que podremos accederla mediante el objeto llamado logo. Las propiedades disponibles más utilizadas son: border, height (alto), width (alto), src (URL del gráfico asociado).

Por ejemplo, logo.width = logo.width * 2 duplicaría el ancho del logo y logo.src = 'logo2.gif' cambiaría el gif que ese muestra, por ejemplo, se podría indicar en un evento onmouseover y onmouseout para generar un efecto.

Caché de Imágenes

Se puede, adicionalmente realizar una previa carga de las imágenes para no generar un efecto extraño al cambiar la URL de una imagen, dado que debe cargar en el momento la imagen.

Dentro de un evento, como onmouseover se puede acceder a un objeto especial llamado “this” que nos permite acceder a las propiedades del propio objeto.

Para ello, se utiliza un script como el siguiente generando imágenes vía código:

```
<script>
var image1 = new Image(200,200);
image1.src = 'image1.gif';
var image2 = new Image(200,200);
image2.src = 'image2.gif';
```

```
</script>

<img src="image1.gif" name="logo"
     onmouseover="this.src=image2.src"
     onmouseout="this.src=image1.src">
```

TRABAJANDO CON STRINGS

En JavaScript, los Strings o cadenas de texto no son tipos de datos básicos, sino que son objetos, por lo que también podremos acceder a métodos y atributos/propiedades de una variable tipo String. Veamos algunos de ellos:

- length(): Nos devuelve la longitud de la cadena de texto
- substring(desde, hasta): Nos devuelve una porción de una cadena de texto
- charAt(posicion): Nos devuelve el carácter en la posición indicada
- indexOf(loquebusco): Busca una cadena dentro de la cadena principal y devuelve la posición donde se encontró o devuelve -1 si no existe.
- lastIndexOf(loquebusco): Igual al anterior sólo que busca la última ocurrencia de lo que buscamos.
- split(delimitador): Separa un string en un vector de strings, separándo el original por el carácter delimitador que se especifique, por ejemplo si tenemos "Jorge, Perez" y hacemos split(","), nos dará un vector donde Jorge será el primer elemento y Perez el segundo.
- toLowerCase(): Convierte el string a minúsculas
- toUpperCase(): Convierte el string a mayúsculas

El siguiente ejemplo validaría una dirección de e-mail (en forma básica) de un formulario, si tiene arroba divide el usuario del dominio y si no tiene el carácter @ muestra un error:

```
var email = document.getElementById(“email”).value;
var marca = email.indexOf('@');
if (marca == -1) {
   window.alert('Dirección de E-mail inválida');
   document.getElementById(“email”).focus();
} else {
   var usuario = email.substring(0,marca);
   var dominio = email.substring(marca+1,email.length);
}
```

TIPS ADICIONALES

Modificando estilos CSS

Desde JavaScript también es posible acceder a un atributo especial de todos los nodos HTML, se trata de style, que permite definir un estilo CSS inline. JavaScript representa los estilos CSS de un elemento con un objeto con notación de punto. Así, todas las propiedades CSS tienen su contraparte JavaScript con un ligero cambio de sintaxis. En CSS hay propiedades, como text-align, que llevan guion; éstas se transforman en textAlign.

Ejemplos

```
// Cambia el color de fondo del body (documentElement)
document.documentElement.style.backgroundColor = "blue";
// Cambia el tamaño de tipografía de un elemento
document.getElementById("elemento").style.fontSize = "larger";
```

Acceso a Propiedades

En los objetos, además de utilizar el "." para indicar una propiedad, por ejemplo document.bgColor, podremos hacerlo a través de corchetes con la propiedad indicada entre comillas, como si fuera un vector de índices alfanuméricos: document['bgColor']. Esto es útil para acceder en forma dinámica a diferentes propiedades, teniendo la propiedad en una variable String.

Redirección

Si queremos que un script redirija a un usuario a otra página, por ejemplo, cuando presiona un botón, podemos utilizar el objeto location dentro de window. Por ejemplo

```
<script>
function ir() {
   window.location.href = "http://www.yahoo.com.ar"
}
```

```
</script>
<input type="button" onclick="ir()">
```

Código cronometrado

En JavaScript podemos definir una acción a realizarse en el futuro, por ejemplo, en 10 segundos a partir de la ejecución del código. Para ello llamamos a la función setTimeout del objeto window que recibe dos parámetros, la acción a ejecutar (dada como función) y la cantidad de milisegundos que debe esperar para ejecutarla. El siguiente ejemplo saludará al usuario en 5 segundos a partir de la ejecución del código (que puede estar dentro de un evento de botón o en una función).

```
function saludar() {
    alert(“hola”)
}
window.setTimeout(saludar, 5000);
```

Para cancelar la ejecución de una tarea programada, como ésta, debemos asignársela a una variable, por ejemplo:

```
<script>
var bomba = window.setTimeout(function() { alert(‘Boom!’ } ),10000);
</script>

<button type="button" onClick="clearTimeout(bomba)">
Apretame, te queda poco tiempo
</button>
```

Este ejemplo, muestra un mensaje “Boom!” a no ser que el usuario presione el botón antes de los 10 segundos de ejecutado el primer código.

Para generar un evento que se ejecute cada 10 segundos, debemos crearnos una función, luego crear un TimeOut que la ejecute y, dentro de dicha función, crear un nuevo TimeOut llamandose a si misma en otros 10 segundos.

Reescribiendo porciones de código HTML

Si utilizamos document.write luego de que la página haya sido cargada (al presionar un botón, por ejemplo) borraremos todo el contenido de la página directamente. Para poder cambiar parte de texto de una página por otro contenido, debemos encerrar este contenido por un tag <DIV> y </DIV>, asignarle un nombre y luego, desde una función acceder a la propiedad innerHTML que nos permite leer y cambiar el contenido de esa porción de nuestra página web.

```
<html>
<body>
<div id="texto">Primer texto</div>
<form name="form1">
    <input type="text" id="nuevoTexto">
    <button type="button" onClick="cambiarTexto()">Cambiar Texto</button>
</form>
</body>
</html>
```

Sobre el HTML anterior, podríamos ejecutar el siguiente código JavaScript y cambiar el texto al clickear en el botón:

```
function cambiarTexto() {
   var nuevoTexto = document.getElementById(“nuevoTexto”).value;
   document.getElementById(“texto”).innerHTML = nuevoTexto;
}
```

Buenas prácticas en Nomenclatura

Es una buena práctica tener una nomenclatura estándar para usar. Esto ayudará a reducir la cantidad de errores y a compartir el código con otros programadores. Una sugerencia utilizada por el mismo lenguaje en varias oportunidades es la siguiente:

• Utilizar notación camello minúscula en variables y funciones. Esto es, la primera letra de todas las variables y funciones debe ser minúscula y todas las palabras siguientes que forman el nombre deben comenzar con mayúscula. Por ejemplo: imprimir(), imprimirNombreCompleto(), cantidadAcumulada.

• En funciones constructoras (clases) utilizar notación camello mayúscula. Esto es, todas las palabras que forman parte del nombre deben comenzar con mayúscula. Por ejemplo: String, ListaDeEmpleados.

• En propiedades de tipo función y en nombre de eventos utilizar minúscula sin separación de palabras. Por ejemplo: onclick, onreadystatechange, onfinishshopping.

• En constantes utilizar mayúsculas y guion bajo para separar palabras. Por ejemplo: PI, MENSAJE_ERROR.

• Prefijos. Si bien hay técnicas para utilizar prefijos en todas las variables (p. ej., si n fuera para números, nTotal sería el nombre correcto), lo ideal en JavaScript es utilizar prefijos cuando su uso realmente valga la pena, por ejemplo, para diferenciar vectores de variables simples. En general se puede utilizar a o v como prefijo.

• Prefijos en Id. Es muy útil utilizar prefijos en los *Id* de los elementos HTML. De esta manera, se pueden identificar y encontrar elementos en el DOM con rapidez. Algunas sugerencias son:

ELEMENTO	PREFIJO	EJEMPLO
a	lnk	lnkProximo
input type="button"	btn	btnGuardar
input type="text"	txt	txtDireccionPostal

input type="checkbox"	chk	chkAceptaCondiciones
select	lst	lstPaises
table	tbl	tblVentas

JQUERY

jQuery es una librería de JavaScript; una librería es un conjunto de funciones y utilidades que nos hacen la vida más fácil cuando programamos. Si podemos mencionar una librería para JavaScript que haya dominado el mercado por años, seguramente sea jQuery de quien estemos hablando.

Surgida como librería de código abierto para múltiples navegadores en 2006, no ha sido hasta mitad de 2008 que su uso se convirtió en una de las librerías más usadas del mercado. Según un estudio independiente (disponible en `trends.builtwith.com/javascript`), a mitad del año 2010 el 40% de los sitios Web utilizaban jQuery. Desde 2015 su uso ha empezado a caer en pos de otras ideas, como React y Angular, pero todavía es muy usada hoy y por eso es importante aunque sea conocer las bases de la librería.

Una de las ventajas, respecto de otras tecnologías, es su arquitectura de agregados o *plugins*, que permite que cualquier desarrollador cree distintas soluciones que utilicen a jQuery y que se adhieran a él, logrando miles de soluciones prácticas gratuitas y simples, disponibles en `plugins.jquery.com`.

Instalando la librería

Si bien la librería puede ser descargada desde https://jquery.com/download/, el uso más simple es incluyendo un script en nuestra web apuntando a un servidor CDN, que es un servidor público que aloja la librería por nosotros, por ejemplo uno de Google para la version 3.3 de jQuery:

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
```

Fundamentos de la librería

Esta librería tiene ciertos conceptos de programación en JavaScript que hace que algunas personas lo consideren casi un lenguaje propio montado sobre JavaScript. Toda la funcionalidad descripta se realiza a través de un solo objeto, y también función, llamado `jQuery`, el cual está representado con un alias por el signo peso o dólar (`$`), un objeto global de JavaScript del que disponemos al incluir la librería. En general, por simplicidad, utilizaremos la versión del signo pesos.

Podemos utilizarlo como función en su versión más simple, el cual recibe un selector de CSS definido como string como primer parámetro. Esta función nos devuelve lo que también se conoce como un objeto jQuery o una consulta. Un objeto jQuery contiene una colección de objetos DOM, por ello también se lo conoce como "consulta de elementos" o "elementos seleccionados". El fundamento y la simplicidad de esta librería se los debemos a este objeto y los métodos que existen para sus instancias.

La consulta de elementos es muy similar al nuevo método DOM document.querySelectorAll. La diferencia radica en lo que nos devuelve, que en el caso de jQuery es un objeto que puede ser encadenado con muchas funciones como veremos más adelante.

Veamos algunos ejemplos:

```
$("#id")
$("ul li p")
$("div.clase")
$("input[type=checkbox]")
```

Analizando el ejemplo, vemos que podemos incluir cualquier tipo de selector CSS dentro de las comillas para hacer una consulta sobre el DOM y traer una colección de todos los elementos que cumplen con tal condición. Cuando queremos traer un solo elemento, siempre debemos acceder a la posición cero, si es que el elemento existe, dado que una consulta jQuery siempre nos devuelve una colección, aun cuando el resultado sea un solo elemento. No obstante, veremos rápidamente que esto no es una complicación, dado que rara vez utilizaremos el accessor de array para trabajar con los elementos.

En el ejemplo anterior, podemos detectar que jQuery soporta selectores de CSS 3, como ser el selector condicional de atributo. En ese caso estaría buscando los elementos de tipo input, cuyo atributo type contenga el valor checkbox. Cabe destacar que jQuery soporta selectores de CSS 3, incluso cuando el navegador sobre el que está ejecutando no soporta CSS 3. Para conocer más sobre todos los selectores CSS 3 disponibles y su funcionamiento, recomendamos el libro "HTML5 y CSS 3" de esta misma colección.

Funciones de jQuery

Cualquier consulta de elementos jQuery, o sea, el objeto que `$(selector)` nos devuelve, posee una selección de métodos realmente muy útiles y simples de usar que automáticamente se aplican a todos los elementos que hayan cumplido con la búsqueda.

Los métodos se dividen según la funcionalidad:

- • Atributos y contenido HTML.
- • Estilos CSS.
- • Datos.
- • Efectos.
- • Eventos.
- • Formulario.
- • Utilitarios.
- • AJAX.

Encadenamiento de funciones

Una de las importantes características de jQuery es que todas las funciones asociadas a una consulta de elementos, además de realizar la acción que le pedimos, retorna el mismo objeto jQuery, con lo cual podemos encadenar funciones.

Por ejemplo, podemos cambiar el contenido de un div y aplicar un css al mismo tiempo

```
$("div").html(“nuevo contenido”).css(“color”, “blue”);
```

Usualmente cuando es así, usamos la siguiente notación:

```
$("div").html(“nuevo contenido”)
        .css(“color”, “blue”);
```

Es importante notar que no debe haber un punto y coma al final de la primera línea, porque el encadnamiento funciona siempre sobre la misma línea.

Aquí es donde se ve la simplicidad de la librería jQuery. Podemos generar comportamiento complejo con apenas una o algunas líneas de código. Algunos desarrolladores también prefieren utilizar una sintaxis de encadenamiento multilínea.

La lista completa de funciones está disponible en `api.jquery.com`. A continuación veremos las funciones más utilizadas.

Funciones de HTML

Estos métodos permiten manipular fácilmente los atributos de los elementos que haya devuelto una consulta jQuery.

Si queremos obtener el valor de un atributo de un solo elemento, en lugar de acceder a la primera posición de la colección y luego acceder al atributo, podemos utilizar la función `attr` que, en su versión inicial, obtiene el valor del atributo del primer elemento de una consulta de elementos. Por ejemplo:

```
var tipoInput = $("input#edad”).attr(“type”);
```

También se puede cambiar el valor con:

```
$(“#nombre”).attr(“maxlength”, “34”);
```

Para obtener el texto que contiene un elemento, podemos usar la función `html` aplicada sobre una consulta de elementos. Esta función sólo reacciona sobre el primero de los elementos en el caso de una consulta con muchos resultados.

La misma función, en caso de enviarle un parámetro, modifica el contenido del elemento (muy similar al uso de `innerHTML` en DOM). Veamos unos ejemplos:

```
// Muestra el contenido del DIV con id=mensaje como HTML
alert($(“div#mensaje”).html());
// Cambia el contenido de un div con el id=mensaje
$(“div#mensaje”).html(“Este es el nuevo contenido”);
```

Si lo que queremos no es reemplazar el contenido, sino agregar algo al inicio o al final del contenido de los selectores, podemos utilizar el método `append` para agregar al final y `prepend` para agregar al inicio. Ambas funciones aceptan como parámetro un *string*, un elemento DOM o un elemento jQuery.

Asimismo, es posible agregar elementos antes y después de otros elementos, utilizando los métodos `after` (después) y `before` (antes). A diferencia de `append` y `prepend`, estos métodos agregan el contenido por fuera de los elementos de la consulta y no dentro.

Dado que la utilización de formularios es uno de los usos más normales en un desarrollo Web, jQuery nos provee de ciertas herramientas que nos permitirán trabajar con formularios más fácilmente. La función `serialize`, que se aplica a cualquier consulta de elementos que posea elementos de formulario (generalmente un elemento `form`), convierte todos los valores de los elementos (`input`, `textarea`, `select`) en un *string* con formato QueryString (del tipo `clave=valor&clave=valor`), útil para enviar vía Internet.

Funciones de CSS

La manipulación de CSS es una de las funciones más prácticas de las funciones de jQuery. Para ello disponemos de los siguientes métodos:

```
var valorViejo = $("#elementoid").css("propiedad");
$("#elementoid").css("propiedad", "nuevo-valor");
$("#elementoid").addClass("clase");
$("#elementoid").removeClass("clase");
```

Mostrando y Ocultando elementos

Vamos a mencionar los métodos `hide` y `show`. Estos métodos sin ningún parámetro ocultan y muestran, respectivamente, a la consulta de elementos sin ningún tipo de animación, modificando la propiedad `display` de CSS.

```
$("#seccion1").hide();
```

Funciones de Eventos

En jQuery tenemos diversas formas de capturar eventos de los elementos del DOM. La primera de ellas es el enlace estático de eventos a partir del método `bind`, que se puede aplicar a cualquier consulta de elementos. Todos los enlaces que estamos analizando utilizan el patrón de diseño "observador"; esto significa que cada vez que nos enlazamos a un evento, estamos agregando un oyente a dicho evento y no estamos reemplazando a oyentes que ya hubiera de antemano. Es decir, podemos tener varias funciones escuchando al mismo evento del mismo elemento DOM.

La forma más simple que veremos de manipular eventos es vía funciones con el nombre del evento es un *string* equivalente al atributo de DOM, sin la palabra `on`. Así, disponemos de funciones como `blur`, `change`, `click`, `dblclick`, `error`, `focus`, `hover`, `keydown`, `keypress`, `keyup`, `load`, `mousedown`, `mouseup`, `load`, `select` y `submit`, entre otras. Estas funciones sólo reciben un parámetro de tipo función, que es el que se ejecutará al momento de ocurrir el evento.

Veamos un ejemplo simple:

```
$("#botonEnviar").click(function() {
    // Hacemos algo cuando el usuario hace click en el botón
});
```

Evento ready

Una de las tareas más comunes de una aplicación AJAX es ejecutar cierto código de manera inicial, cuando la página es cargada. Generalmente, estas acciones se realizan en el evento `load` del documento, ya sea via `window.onload` o via HTML en `<body onload>`. Hay que tener en cuenta que este evento se ejecuta cuando todo el documento está listo, eso implica que el navegador ya descargó todos los recursos externos, incluyendo estilos CSS e imágenes.

Es muy probable que las tareas que queremos realizar en este evento no dependan de las imágenes ni del CSS, por lo que podrían hacerse antes que los recursos estén descargados, ahorrando mucho tiempo y logrando una interfaz más rápida. Navegadores modernos soportan el evento DOMContentLoaded que se ejecuta cuando el DOM está listo y en memoria, aunque los recursos, como imágenes, no necesariamente están disponibles. Pero algunos navegadores antiguos no soportan este evento.

Para simplificar el problema, jQuery crea un nuevo concepto conocido como el evento `ready` (listo), evento que sólo puede aplicarse al documento total y no a un selector.

```
$(document).ready(function() {
  // Todo el documento está cargado
});
```

AJAX

AJAX es una técnica de JavaScript que nos permite consultar información o enviar información desde y hacia un servidor externo. Esto nos permite sin refrescar la página ni enviar al usuario a otra página, mostrar información de fuentes externas (como el clima, la cotización del dólar) o enviar información (como los datos de un formulario que el usuario carga).

La información que descargamos puede ser recibida usualmente como texto, como objeto JSON o como XML. Lo más usual es JSON, para así poder manipularlo como un objeto de JavaScript

Por ejemplo, si tenemos una URL ficticia que nos da la cotización del dólar, como por ejemplo https://cotizaciones.com/dolar y este sitio nos devuelve un JSON como el siguiente:

```
{
   “monedas”: {
       “dolar”: 25,
       “euro”: 28
   }
}
```

Podríamos descargar ese archivo desde Internet y mostrar la información en nuestra página web.

Para evitar robo de información, los navegadores sólo pueden ir a consultar archivos en nuestro propio servidor o en servidores que habilitaron la posibilidad de que otras webs lean su contenido. Los servicios que entregan datos públicos suelen habilitarlo (la habilitación se llama CORS)

AJAX con jQuery

Por muchos años hacer pedidos de tipo AJAX llevaban mucho código. La librería jQuery lo ha simplificado con el uso de varias funciones destinadas a tal fin. La más simple función ($.get) permite leer un JSON de Internet y ejecutar una función cuando los datos han llegado, por ejemplo, podríamos colocar la cotización del dólar en un div llamado “cotizacionDolar”

```
$.get(“https://cotizaciones.com/dolar”, function(respuesta) {
     var dolar = respuesta.monedas.dolar;
     $(“#cotizacionDolar”).html(dolar)
});
```

AJAX con Fetch

Navegadores modernos incluyen una función llamada fetch (pedir), que se parece bastante a $.get de jQuery y permite hacer pedidos AJAX sin necesidad de tener la librería jQuery de forma fácil y rápida. El mismo código pero usando fetch sería:

```
fetch(“https://cotizaciones.com/dolar”)
   .then( function(respuesta) {
       return respuesta.json(); // le decimos que queremos JSON
   })
   .then( function (respuesta) {
     var dolar = respuesta.monedas.dolar;
     document.getElementById(“cotizacionDolar”).innerHTML = dólar;
   });
```

Para navegadores que no soportan fetch se puede agregar una pequeña librería que la haga compatible.

```
<script src="https://unpkg.com/unfetch/polyfill"></script>
```

La lista de navegadores compatibles con fetch está actualizada y disponible en https://caniuse.com/#search=fetch