Desarrollo de Aplicaciones Web

Práctica 6: JavaScript

1. Objetivos

- Aprender el lenguaje de programación JavaScript.
- Aprender a manejar el DOM de una página web para manipular su contenido.
- Aprender a validar un formulario con el lenguaje de programación JavaScript.

2. Recursos

¿Cuál es la sintaxis de JavaScript? ¿Qué palabras clave existen?

- **W3Schools**[1]: cursos de aprendizaje y guías de referencia de diversas tecnologías empleadas en la programación web. Incluye un tutorial y temas avanzados sobre JavaScript.
- **ECMA-262 ECMAScript Language Specification**[2]: estándar en el que se basa el lenguaje JavaScript.
- **Mozilla Developer Center JavaScript**[3]: información sobre JavaScript según los desarrolladores del proyecto Mozilla.
- **ECMAScript support in Opera**[4]: información sobre JavaScript según los desarrolladores del navegador Opera.
- **JavaScript Cheat Sheet**[5]: resumen de los métodos y funciones principales, la sintaxis de las expresiones regulares y el objeto XMLHttpRequest.
- **JavaScript Reference**[6]: fichas resumen de los métodos y funciones principales. Incluye una tabla con las versiones que aceptan los principales navegadores.

¿Cómo puedo comprobar que el código que escribo es correcto?

- **Firebug**[7]: extensión (*add-on*) para el navegador Mozilla Firefox, es una herramienta esencial para cualquier desarrollador web. Incluye: inspector y editor de HTML, CSS, JavaScript y DOM; regla para tomar mediciones; monitor de actividad de red y depurador de JavaScript.
- **JSLint, The JavaScript Verifier**[8]: aplicación en página web que verifica el código JavaScript, posee múltiples opciones.
- **JavaScript Lint**[9]: aplicación para descargar y ejecutar en local que emplea el motor del navegador Mozilla Firefox.
- **JavaScript Editor**[10]: potente editor de código JavaScript. Es de pago.

[1] http://www.w3schools.com/
[2] http://www.ecma-international.org/publications/standards/Ecma-262.htm
[3] http://developer.mozilla.org/en/JavaScript
[4] http://www.opera.com/docs/specs/js/ecma/
[5] http://www.addedbytes.com/cheat-sheets/javascript-cheat-sheet/
[6] http://javascript-reference.info/
[7] http://getfirebug.com/
[8] http://www.jslint.com/
[9] http://www.javascriptlint.com/
[10] http://www.c-point.com/javascript_editor.php

¿Cómo puedo esconder mi código JavaScript? No se puede, pero puedes emplear un "ofuscador" para ponérselo difícil al que quiera copiar tu código. Estas herramientas también sirven para comprimir el código JavaScript y reducir su tiempo de descarga.

- **iWeb Toolkit: Javascript Compiler**[11]: encriptador de código JavaScript.
- **Lista de ofuscadores para JavaScript**[12]: lista con enlaces a varios ofuscadores.

¿Qué es el DOM?

- **W3C Documento Object Model (DOM)**[13]: especificación oficial del DOM según el W3C.
- **Mozilla Developer Center Gecko DOM Reference**[14]: guía de referencia de los objetos del DOM disponible en el motor Gecko empleado por varios navegadores web como Mozilla Firefox.
- **Support for DOM2 Core and XML modules in Opera**[15] y **DOM 2 HTML Objects supported in Opera**[16]: objetos, propiedades y métodos del DOM que admite el navegador Opera.

3. ¿Qué tengo que hacer?

En esta práctica tienes que emplear JavaScript para validar los formularios que has realizado en las prácticas anteriores. En concreto, tienes que programar las siguientes restricciones:

Página principal Contiene un formulario (nombre de usuario y contraseña) para acceder como usuario registrado. Antes de enviar el formulario, debes comprobar que el usuario ha escrito algo en ambos campos, pero evita que el usuario escriba únicamente espacios en blanco o tabuladores.

Página con el formulario de registro como nuevo usuario Contiene un formulario con los datos necesarios para registrarse (nombre de usuario, contraseña, repetir contraseña, dirección de email, sexo, fecha de nacimiento, ciudad y país de residencia, foto). Antes de enviar el formulario, debes realizar las siguientes comprobaciones:

- **nombre de usuario**: sólo puede contener letras del alfabeto inglés (en mayúsculas y minúsculas) y números; no puede comenzar con un número; longitud mínima 3 caracteres y máxima 15.
- **contraseña**: sólo puede contener letras del alfabeto inglés (en mayúsculas y minúsculas), números, el guion y el guion bajo (subrayado); al menos debe contener una letra en mayúscula, una letra en minúscula y un número; longitud mínima 6 caracteres y máxima 15.
- **repetir contraseña**: su valor debe coincidir con el escrito en el campo **contraseña**.
- **dirección de email**: no puede estar vacío, hay que comprobar que cumple el siguiente patrón de una dirección de email (es una simplificación del estándar):
  - El formato de un correo electrónico es `parte-local@dominio`.
  - La longitud máxima de `parte-local` es 64 caracteres. La longitud máxima de `dominio` es 255 caracteres. Cada una de las partes tiene una longitud mínima de 1 carácter.
  - La `parte-local` es una combinación de las letras mayúsculas y minúsculas del alfabeto inglés, los dígitos del 0 al 9, los caracteres imprimibles `!#$%&'*+-/=?^_`{|}~` y el punto. El punto no puede aparecer ni al principio ni al final y tampoco pueden aparecer dos o más puntos seguidos.
  - El `dominio` es una secuencia de uno o más `subdominio` separados por un punto.
  - La longitud máxima de `subdominio` es 63 caracteres.
  - Un `subdominio` es una combinación de las letras mayúsculas y minúsculas del alfabeto inglés, los dígitos del 0 al 9 y el guion. El guion no puede aparecer ni al principio ni al final.

[11] http://www.virtualpromote.com/tools/javascript-encrypt/
[12] http://sentidoweb.com/2006/10/11/lista-de-ofuscadores-para-javascript.php
[13] http://www.w3.org/DOM/
[14] http://developer.mozilla.org/en/Gecko_DOM_Reference
[15] http://www.opera.com/docs/specs/js/ecma/dom/index.dml
[16] http://www.opera.com/docs/specs/js/ecma/dom/html/index.dml

    - La longitud máxima de una dirección de correo electrónico es 254 caracteres[17].
- **sexo**: se debe elegir un valor.
- **fecha de nacimiento**: comprobar que es una fecha válida y que la persona tenga al menos 18 años recién cumplidos el mismo día en que se registra.
- El resto de campos no indicados se pueden quedar vacíos.

Evidentemente, algunas de las validaciones planteadas se pueden implementar con HTML[18] y con el Constraint validation API[19]. Por ejemplo, se puede emplear el atributo `pattern` para definir una expresión regular, se puede emplear `type="email"` para comprobar que un correo electrónico es válido, se puede emplear `type="date"` para forzar al usuario a introducir una fecha válida, etc. Sin embargo, el objetivo de esta práctica es aprender a utilizar JavaScript, por lo que **NO SE DEBE HACER USO DE ESAS FACILIDADES QUE PROPORCIONA HTML EN ESTA PRÁCTICA**, aunque en muchos casos sea la mejor opción[20]. Por tanto, **se deben modificar los formularios para que no actúen las validaciones de HTML**. De este modo se podrá comprobar el correcto funcionamiento de las validaciones mediante JavaScript.

A la hora de mostrar los mensajes de error, existen básicamente cuatro mecanismos diferentes:

1. Un simple `alert()` de JavaScript que muestra todo los mensajes de error juntos. Utilizar un `alert()` por cada mensaje de error es una solución muy mala.
2. Un diálogo modal.
3. En la propia página, al principio del formulario o al lado de cada campo que tenga un error.
4. En línea, conforme se vaya introduciendo la información, pero evidentemente hay que evitar que se envíe el formulario si hay algún error.

Además de mostrar los mensajes de error, también debes cambiar la presentación visual de los campos con datos erróneos. Puedes cambiar la presentación del campo, de la etiqueta del campo o de ambos, lo que sea más útil para el usuario.

Importante: para realizar la validación de formularios lo más correcto es emplear expresiones regulares, pero en esta práctica no tienes que utilizar expresiones regulares, tienes que programar para cada situación el algoritmo de validación apropiado.

Por último, en la página **"Solicitar álbum"** (**"Solicitar folleto"**), además de la tabla de tarifas que ya tenía la página, se tiene que mostrar una tabla con posibles costes de un álbum (folleto) según los diferentes parámetros (número de páginas, número de fotos, color y resolución), tal como se puede ver en la Figura 1. Esta tabla se tiene que calcular mediante JavaScript y se tienen que aplicar las tarifas que se proporcionaron en una práctica anterior. La tabla con posibles costes de un álbum (folleto) se debe mostrar en la misma página **"Solicitar álbum"** (**"Solicitar folleto"**), pero como ocupa mucho espacio, se debe poder mostrar y ocultar al pulsar un botón.

¿Cómo está calculada la tabla? Fíjate bien en los valores que contiene, al principio pensarás que está mal, pero la tabla está bien, se aplica un sistema de bloques de número de páginas, similar al sistema de bloques de consumo que se aplica en las facturas de agua.

Esta tabla NO SE CALCULA a partir de lo que un usuario seleccione en el formulario que aparece en la página **"Solicitar álbum"** (**"Solicitar folleto"**). La tabla se calcula (lo que se tiene que calcular son los costes, los valores con euros) a partir de los valores que se indican en la Figura 1 en las dos primeras columnas y en la fila de encabezados. Por ejemplo, en la primera fila de la tabla, el valor 12,00 € es la respuesta a la pregunta "¿cuál es el coste de un álbum (folleto) con 1 página, 3 fotos, en blanco y negro y 150-300 dpi?".

Importante: la tabla se tiene que crear con las funciones del DOM (*Document Object Model*) que permiten añadir contenido a una página web nodo a nodo. Es decir, no vale añadir el contenido como una cadena mediante `innerHTML` u otro mecanismo similar. Y el contenido debe estar calculado mediante la aplicación de las tarifas, no vale añadir directamente los números que se muestran en la tabla de la Figura 1. Es decir, se debe realizar un cálculo que en cada fila de la tabla tenga en cuenta el número de páginas, el número de fotos, la resolución y si la impresión es en blanco y negro o en color.

[17] Evidentemente, no tiene sentido que se diga que la longitud máxima de dominio es 255 caracteres, pero la longitud total es 254 caracteres. Esta incongruencia se debe a un error que existe en el estándar que impide que se cumplan otros estándares relacionados. La explicación está en RFC Errata to 3696: `https://www.rfc-editor.org/errata/eid1690`

[18] `https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation`

[19] `https://developer.mozilla.org/en-US/docs/Web/API/Constraint_validation`

[20] En versiones antiguas de los navegadores web es posible que las validaciones nativas de HTML no funcionen, por lo que validar mediante JavaScript tiene mucho sentido.

		Blanco y negro		Color	
Número de páginas	Número de fotos	150-300 dpi	450-900 dpi	150-300 dpi	450-900 dpi
1	3	12,00 €	12,60 €	13,50 €	14,10 €
2	6	14,00 €	15,20 €	17,00 €	18,20 €
3	9	16,00 €	17,80 €	20,50 €	22,30 €
4	12	18,00 €	20,40 €	24,00 €	26,40 €
5	15	19,80 €	22,80 €	27,30 €	30,30 €
6	18	21,60 €	25,20 €	30,60 €	34,20 €
7	21	23,40 €	27,60 €	33,90 €	38,10 €
8	24	25,20 €	30,00 €	37,20 €	42,00 €
9	27	27,00 €	32,40 €	40,50 €	45,90 €
10	30	28,80 €	34,80 €	43,80 €	49,80 €
11	33	30,40 €	37,00 €	46,90 €	53,50 €
12	36	32,00 €	39,20 €	50,00 €	57,20 €
13	39	33,60 €	41,40 €	53,10 €	60,90 €
14	42	35,20 €	43,60 €	56,20 €	64,60 €
15	45	36,80 €	45,80 €	59,30 €	68,30 €

Figura 1: Tabla con posibles costes de un álbum (folleto de publicidad) impreso

Importante:

- **El código que programes debe ser eficaz, eficiente y modular.**
- **La aplicación debe ser usable, debe ayudar al usuario a completar lo que desea hacer. Por ejemplo, los mensajes de error al rellenar un formulario deben ser claros y significativos.**

4. ¿Cómo lo hago?

4.1. Inclusión de JavaScript en una página web

JavaScript es un lenguaje de script que comparte la misma sintaxis básica que los lenguajes de programación C, C++ y Java. El código de JavaScript se puede incluir de tres formas en una página web:

- Mediante la etiqueta `<script>`.
- En un enlace (etiqueta `<a>`) en el atributo `href`.
- En un atributo que representa un evento, como `onclick` u `onsubmit`.

Por ejemplo, la siguiente página web contiene dos líneas de código JavaScript que se ejecutan automáticamente al cargar la página web; el atributo `type` de la etiqueta `<script>` indica el lenguaje de programación empleado en el código de script[21]:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>Prueba de JavaScript</title>
<script type="text/javascript">
<!--
window.alert("Mensaje 1");
```

[21] En HTML5 ya no es necesario utilizar el atributo `type`.

```
alert("Mensaje 2");
// -->
</script>
</head>
<body>
<p>
Una página web sencilla.
</p>
</body>
</html>
```

En el ejemplo anterior, el código JavaScript está encerrado en un comentario de HTML para que no sea interpretado por aquellos navegadores que no reconocen JavaScript[22]. Además, la llamada al método `alert()` que muestra un cuadro de diálogo con un mensaje se realiza de dos formas equivalentes: a través del objeto `window` y directamente.

En JavaScript hay muchas formas de escribir un código para lograr un objetivo. Por ejemplo, el siguiente código muestra tres formas distintas de asociar una función al evento `click` de un botón (fíjate que en el siguiente ejemplo ya no aparece `type="text/javascript"`):

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Prueba de click</title>
<script>
function funcionClick(msg) {
       alert("El mensaje que debo mostrar es: " + msg);
}

function load() {
       document.getElementById("boton2").onclick = function(){funcionClick("Botón 2")};
       document.getElementById("boton3").addEventListener("click", function(){funcionClick("
           Botón 3")});
}

document.addEventListener("DOMContentLoaded", load, false);
</script>
</head>
<body>
<p>
<input type="button" value="Con atributo onclick" id="boton1" onclick="funcionClick('Botón 1')">
<br> <!-- Recuerda, mejor no usar <br> -->
<input type="button" value="Con onclick en JS" id="boton2">
<br>
<input type="button" value="Con addEventListener en JS" id="boton3">
</p>
</body>
</html>
```

En este ejemplo se han usado las funciones anónimas de JavaScript[23] para pasar información específica a la función manejadora del evento `click`. Además, se ha empleado el evento `DOMContentLoaded`[24] para iniciar la ejecución del código JavaScript. En este ejemplo también se podría haber empleado el evento `load`[25], que también se puede emplear a través de `window.onload`:

```
<script>
function funcionClick(msg) {
       alert("El mensaje que debo mostrar es: " + msg);
}
```

[22] Esto era muy necesario hace algunos años, pero en la actualidad se puede omitir sin problema y en los siguientes ejemplos ya no aparece.

[23] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions#The_function_expression_function_expression

[24] https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded_event

[25] https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event

```
function load() {
       document.getElementById("boton2").onclick = function(){funcionClick("Botón 2")};
       document.getElementById("boton3").addEventListener("click", function(){funcionClick("
           Botón 3")});
}

document.addEventListener("load", load, false);
</script>
```

Los eventos `DOMContentLoaded` y `load` pueden ser intercambiables en muchas situaciones, pero no son iguales:

- `DOMContentLoaded` se activa cuando el DOM está listo (la página se ha cargado).
- `load` se activa cuando se cargan la página y todos los recursos que incluya (imágenes, estilos y otros recursos similares).

4.2. Validación de formularios

El DOM es una API destinada a trabajar con documentos HTML y XML. El DOM está desarrollado por el W3C.

Existen varias formas de acceder a un formulario a través del DOM[26]:

- `document.forms[n]`, donde `n` es la posición que ocupa el formulario en la lista de formularios del documento, comenzando desde 0.
- `document.forms["nom"]`, donde `nom` es el nombre que se ha asignado al formulario con el atributo `name` o `id`.
- `document.nom`, donde `nom` es el nombre que se ha asignado al formulario con el atributo `name`.

Una vez que se ha seleccionado un formulario, existen varias formas de acceder a los campos que contiene (`formu` es un objeto que representa un formulario concreto y que ha sido seleccionado con alguno de los métodos explicados en el párrafo anterior):

- `formu.elements[n]`, donde `n` es la posición que ocupa el campo en la lista de campos del formulario, comenzando desde 0.
- `formu.elements["nom"]`, donde `nom` es el nombre que se ha asignado al campo con el atributo `name` o `id`.
- `formu.nom`, donde `nom` es el nombre que se ha asignado al campo con el atributo `name` o `id`.

Una vez que se ha seleccionado un campo, se tiene que consultar la propiedad `value` para obtener el valor que almacena en un instante concreto.

En el siguiente ejemplo hay un formulario que contiene tres campos de texto para escribir el nombre de tres jugadores, el formulario no se puede enviar si no se han escrito los tres nombres:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>Validación un formulario</title>
<script>
function valida(f) {
var ok = true;
var msg = "Debes escribir algo en los campos:\n";

if(f.elements[0].value == "")
{
msg += "- Jugador 1\n";
ok = false;
}
```

[26] Existe alguna forma más que se empleará en la próxima práctica.

```
if(f.elements["jugador2"].value == "")
{
msg += "- Jugador 2\n";
ok = false;
}

if(f.jugador3.value == "")
{
msg += "- Jugador 3\n";
ok = false;
}

if(ok == false)
alert(msg);
else
alert("Todo correcto, se envía el formulario");

return ok;
}
</script>
</head>
<body>
<form id="miForm" action="" method="get" onsubmit="return valida(this)">
<p>
Jugador 1: <input type="text" id="jugador1">
<br> <!-- Recuerda, mejor no usar <br> -->
Jugador 2: <input type="text" id="jugador2">
<br>
Jugador 3: <input type="text" id="jugador3">
<br>
<input type="submit" value="Enviar">
<input type="reset" value="Borrar">
</p>
</form>
</body>
</html>
```

En el ejemplo anterior, cuando se llama a la función `valida()` desde el evento `onsubmit` se le pasa como valor del parámetro `this`, que es un objeto que representa al formulario desde el que se invoca al método. En este ejemplo, se ha empleado el atributo `id` en vez del atributo `name` para identificar cada elemento, la forma correcta de identificar de forma única un elemento de HTML.

La función `valida()` debe devolver `true` o `false` según la validación haya sido correcta o haya fallado. Este valor debe ser devuelto al navegador: el valor `true` significa "continúa y envía el formulario al servidor", mientras que `false` significa "cancela y no envíes el formulario al servidor". Por ello, en el código se debe escribir `onsubmit="return valida(this)"` para reenviar el valor devuelto al navegador.

Importante: hasta aquí se ha explicado el método tradicional de acceder al contenido de un formulario, está explicado por razones históricas, pero no es la forma adecuada de hacerlo en la actualidad y por lo tanto, no lo tienes que hacer así en la práctica.

En DOM Level 1, publicado en octubre de 1998, se añadieron tres funciones que permiten acceder a los elementos de una página web:

- `document.getElementById(id)`[27]: devuelve el elemento con el identificador indicado. **¡Cuidado!**: recuerda que JavaScript es un lenguaje sensible a mayúsculas y minúsculas, `Id` en `getElementById()` se tiene que escribir con la letra "I" en mayúscula y la letra "d" en minúscula, si se escribe de otra forma, como por ejemplo "ID", fallará.

- `elementoPadre.getElementsByName(nombre)`[28]: devuelve un array de elementos con el nombre indicado en el atributo `name`, los elementos se buscan a partir de `elementoPadre` (si `elementoPadre` es `document`, la búsqueda se realiza en toda la página web).

[27] `https://www.w3.org/TR/REC-DOM-Level-1/level-one-html.html#method-getElementById`
[28] `https://www.w3.org/TR/REC-DOM-Level-1/level-one-html.html#method-getElementsByName`

- `elementoPadre.getElementsByTagName(etiqueta)`[29]: devuelve un array de elementos con la etiqueta indicada, los elementos se buscan a partir de `elementoPadre` (si `elementoPadre` es `document`, la búsqueda se realiza en toda la página web).

El ejemplo anterior se puede escribir de una forma más correcta usando las funciones que se recomiendan en la actualidad:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>Validación un formulario</title>
<script>
function $(id) {
return document.getElementById(id);
}

function valida(event) {
var ok = true;
var msg = "Debes escribir algo en los campos:\n";

if($("jugador1").value == "")
{
msg += "- Jugador 1\n";
ok = false;
}

if($("jugador2").value == "")
{
msg += "- Jugador 2\n";
ok = false;
}

if($("jugador3").value == "")
{
msg += "- Jugador 3\n";
ok = false;
}

if(ok == false) {
alert(msg);
event.preventDefault();
}
else
alert("Todo correcto, se envía el formulario");
}

function load() {
$("miForm").addEventListener("submit", valida);
}

document.addEventListener("DOMContentLoaded", load, false);
</script>
</head>
<body>
<form id="miForm" action="" method="get">
<p>
Jugador 1: <input type="text" id="jugador1">
<br> <!-- Recuerda, mejor no usar <br> -->
Jugador 2: <input type="text" id="jugador2">
<br>
Jugador 3: <input type="text" id="jugador3">
```

[29] https://www.w3.org/TR/REC-DOM-Level-1/level-one-core.html#method-getElementsByTagName

```
<br>
<input type="submit" value="Enviar">
<input type="reset" value="Borrar">
</p>
</form>
</body>
</html>
```

En este ejemplo, la función `valida()` tiene el parámetro `event` que representa el evento `submit`. Un evento posee el método `preventDefault()`[30] que cancela la propagación del evento, que en este caso significa cancelar el envío del formulario. Otro método relacionado es `stopPropagation()`[31].

Con HTML5 han aparecido nuevas funciones para manipular el DOM, pero se tienen que usar con mucho cuidado para que no se produzcan problemas de compatibilidad con navegadores web antiguos. De las nuevas funciones, las más interesantes son:

- `elementoPadre.getElementsByClassName(nombres)`: devuelve un array de elementos que poseen los nombres de clases indicados, los elementos se buscan a partir de `elementoPadre` (si `elementoPadre` es `document`, la búsqueda se realiza en toda la página web). Este método está definido en **W3C DOM4**[32].
- `elementoPadre.querySelector(selector)`: devuelve el primer elemento que cumple el selector indicado, el elemento se busca a partir de `elementoPadre`. El selector emplea la sintaxis de selectores de CSS. Este método está definido en **W3C Selectors API Level 1**[33].
- `elementoPadre.querySelectorAll(selector)`: devuelve un array de elementos que cumplen el selector indicado, los elementos se buscan a partir de `elementoPadre`. El selector emplea la sintaxis de selectores de CSS. Este método está definido en **W3C Selectors API Level 1**.

En definitiva, existen múltiples formas de acceder a un formulario y a los controles de un formulario. El siguiente código muestra 15 formas distintas de acceder al valor que contiene un cuadro de texto:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>Diferentes modos de acceso</title>
<script>
function load() {
console.log(document.forms[0].elements[0].value);
console.log(document.forms[0].elements["usuario"].value);
console.log(document.forms[0].usuario.value);

console.log("----");

console.log(document.forms["miForm"].elements[0].value);
console.log(document.forms["miForm"].elements["usuario"].value);
console.log(document.forms["miForm"].usuario.value);

console.log("----");

console.log(document.miForm.elements[0].value);
console.log(document.miForm.elements["usuario"].value);
console.log(document.miForm.usuario.value);

console.log("----");

console.log(document.getElementById("usuario").value);
console.log(document.getElementsByName("usuario")[0].value);
console.log(document.getElementsByTagName("input")[0].value);
console.log(document.getElementsByClassName("cuadro-texto")[0].value);
```

[30] https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault
[31] https://developer.mozilla.org/en-US/docs/Web/API/Event/stopPropagation
[32] http://www.w3.org/TR/dom/
[33] http://www.w3.org/TR/selectors-api/

```
console.log(document.querySelector("#usuario").value);
console.log(document.querySelectorAll("input")[0].value);
}

document.addEventListener("DOMContentLoaded", load, false);
</script>
</head>
<body>
<form id="miForm" name="miForm" action="" method="get">
<p>
Usuario: <input type="text" id="usuario" name="usuario" class="cuadro-texto" value="Nombre
    usuario">
</p>
</form>
</body>
</html>
```

En la Figura 2 se muestra la salida del código anterior por la consola de JavaScript de un navegador web.

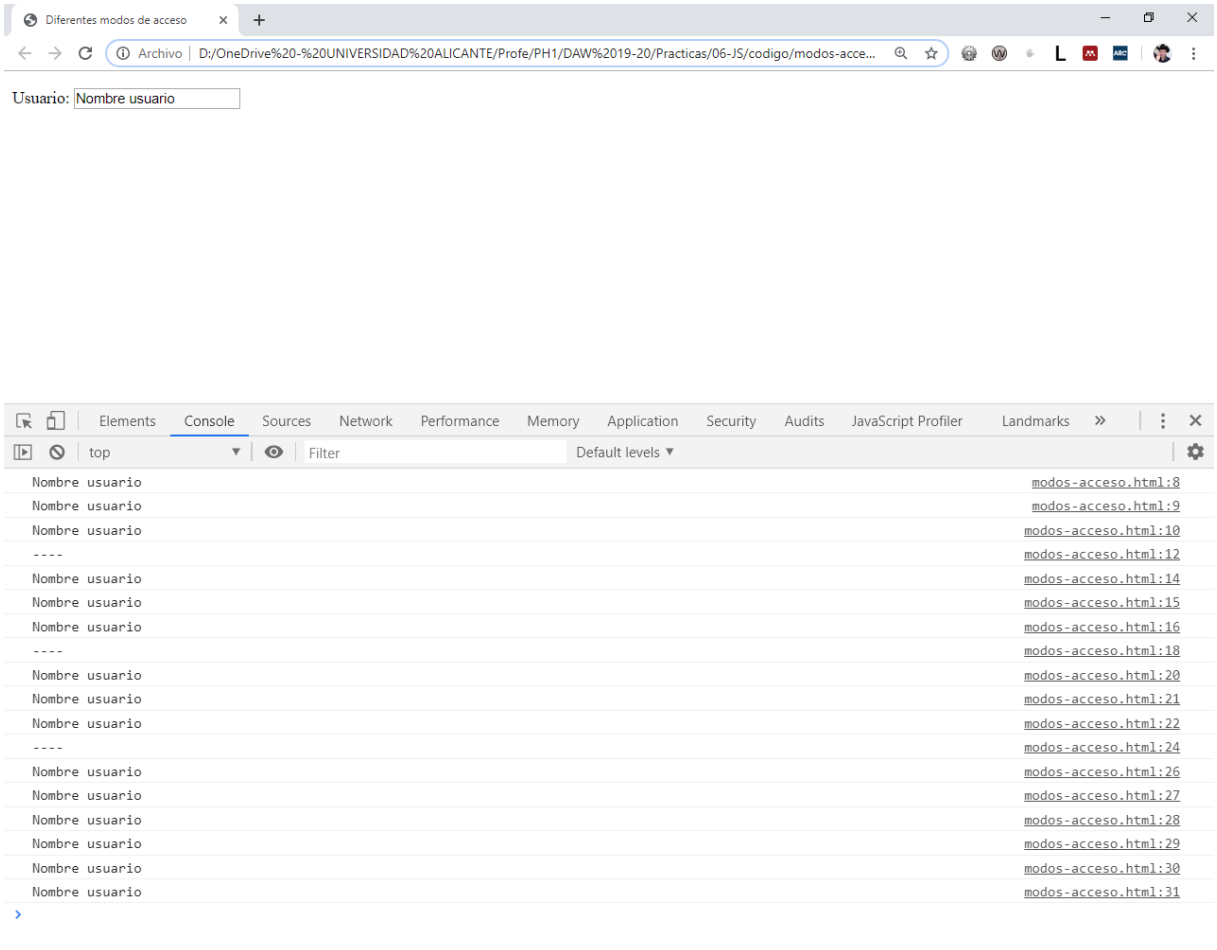

Figura 2: Diferentes modos de acceso a los controles de un formulario

4.3. Manipulación de una página web

El DOM proporciona una representación en forma de árbol de un documento y permite su manipulación (añadir un elemento nuevo, borrar un elemento, mover un elemento de sitio). Algunas funciones útiles del DOM son:

- `document.createElement(etiqueta)`: crea un nuevo elemento según la etiqueta indicada.
- `document.createTextNode(cadena)`: crea un nuevo nodo de texto con la cadena indicada.
- `elementoPadre.appendChild(elementoNuevo)`: añade `elementoNuevo` a `elementoPadre` al final de su lista de elementos hijo.
- `elementoPadre.insertBefore(elementoNuevo, elementoReferencia)`: añade `elementoNuevo` en la lista de elementos hijo de `elementoPadre` antes de `elementoReferencia`.
- `elementoPadre.removeChild(elementoHijo)` y `elementoHijo.remove()`: elimina `elementoHijo` del DOM (lo elimina de la lista de elementos hijo de `elementoPadre`), pero sigue existiendo si existe una referencia a él[34].

El siguiente ejemplo muestra cómo se pueden añadir párrafos de texto nuevos a una página web con el texto que introduzca el usuario en la misma página web:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>Contenido generado por el usuario</title>
<script>
function nuevoTexto() {
	// Obtiene el div que tiene id="parrafos"
	var d = document.getElementById("parrafos");

	// Crea un nuevo elemento de tipo párrafo
	var np = document.createElement("p");

	// Inserta el texto en el contenido del nuevo párrafo
	texto = document.createTextNode(document.getElementById("texto").value);
	np.appendChild(texto);

	// Añade el párrafo al final de los elementos que contenga el div
	d.appendChild(np);

	return false;
}

function load() {
	document.getElementById("boton").addEventListener("click", nuevoTexto);
}

document.addEventListener("DOMContentLoaded", load, false);
</script>
</head>
<body>
<p>
Texto: <input type="text" id="texto">
<br> <!-- Recuerda, mejor no usar <br> -->
<input type="button" value="Añade texto" id="boton">
</p>
<div id="parrafos">
<!-- Inicialmente vacío -->
</div>
</body>
</html>
```

[34] Ambas funciones realizan lo mismo y el resultado final es el mismo, la diferencia está en la forma de invocación: https://stackoverflow.com/a/36999675

4.4. Más formas de manipular una página web

Las funciones anteriores son suficientes para realizar la mayoría de las manipulaciones. Sin embargo, existen funciones y propiedades adicionales que permiten escribir un código más claro y sencillo de mantener.

Para definir el contenido textual de un elemento se puede crear un nodo de texto con la función `document.createTextNode()`[35], tal como se ha mostrado en el ejemplo anterior. Sin embargo, también se puede acceder al contenido textual de un elemento, tanto para lectura como escritura, a través de la propiedad (atributo) `textContent`[36]. La propiedad `textContent` no se debe confundir con `innerText`[37], `innerHTML`[38] y `outerHTML`[39].

Existen funciones para realizar manipulaciones sobre tipos de elementos concretos en una página web. Por ejemplo, los métodos `insertRow()`[40] e `insertCell()`[41] crean y añaden a una tabla una fila o una celda nueva respectivamente. El siguiente código muestra dos formas de crear una tabla, mediante los métodos anteriores o mediante `document.createElement()`:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>Validación un formulario</title>
<style>
table, tr, th, td {
border: 1px solid #000;
}
</style>
<script>
function $(id) {
return document.getElementById(id);
}

function metodo1() {
var tabla = document.createElement("table");
var titulo = tabla.createCaption();
titulo.textContent = $("filas").value + " x " + $("columnas").value;

for(f = 0; f < $("filas").value; f++) {
var fila = tabla.insertRow();
for(c = 0; c < $("columnas").value; c++) {
var celda = fila.insertCell();
celda.textContent = Math.random().toFixed(2);
}
}

document.body.appendChild(tabla);
}

function metodo2() {
var tabla = document.createElement("table");
var titulo = document.createElement("caption");
titulo.textContent = $("filas").value + " x " + $("columnas").value;
tabla.appendChild(titulo);

for(f = 0; f < $("filas").value; f++) {
var fila = document.createElement("tr");
for(c = 0; c < $("columnas").value; c++) {
var celda = document.createElement("td");
celda.textContent = Math.random().toFixed(2);
```

[35] https://developer.mozilla.org/en-US/docs/Web/API/Document/createTextNode
[36] https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent
[37] https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/innerText
[38] https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML
[39] https://developer.mozilla.org/en-US/docs/Web/API/Element/outerHTML
[40] https://developer.mozilla.org/en-US/docs/Web/API/HTMLTableElement/insertRow
[41] https://developer.mozilla.org/en-US/docs/Web/API/HTMLTableRowElement/insertCell

```
fila.appendChild(celda);
}
tabla.appendChild(fila);
}

document.body.appendChild(tabla);
}

function load() {
$("metodo1").addEventListener("click", metodo1);
$("metodo2").addEventListener("click", metodo2);
}

document.addEventListener("DOMContentLoaded", load, false);
</script>
</head>
<body>
<p>Filas: <input type="number" id="filas" value="1" min="1"></p>
<p>Columnas: <input type="number" id="columnas" value="1" min="1"></p>
<p><input type="button" value="Método 1" id="metodo1"> <input type="button" value="Método 2" id
    ="metodo2"></p>
</body>
</html>
```

En el ejemplo anterior también se crea el título de la tabla de dos formas distintas, mediante el método específico `createCaption()`[42] o mediante el método genérico `document.createElement()`.

Además de `appendChild()` e `insertBefore()`, existen algunos métodos nuevos que facilitan el proceso de añadir contenido a una página web:

- `elemento.after()`[43]: inserta un conjunto de nodos justo después del elemento indicado.
- `elementoPadre.append()`[44]: inserta un conjunto de nodos al final de la lista de elementos hijo de `elementoPadre`. Es parecido a `appendChild()`, pero existen algunas diferencias.
- `elemento.before()`[45]: inserta un conjunto de nodos justo antes del elemento indicado. Es parecido a `insertBefore()`, pero existen algunas diferencias.
- `elementoPadre.prepend()`[46]: inserta un conjunto de nodos al principio de la lista de elementos hijo de `elementoPadre`. Es parecido a `insertBefore()`, pero existen algunas diferencias.

Estos métodos se deben usar con cuidado, ya que puede ser que no funcionen en todos los navegadores web.

4.5. Modificación de CSS desde JavaScript

Así como se puede usar JavaScript para cambiar el HTML de una página web, también se puede usar JavaScript para cambiar los estilos CSS. Para ello se puede emplear el CSS Object Model (CSSOM)[47], pero a continuación se van a explicar dos propiedades del DOM, `style`[48] y `className`[49].

Una vez seleccionado un elemento de la página, la propiedad `style` permite acceder a las propiedades CSS del elemento. Por ejemplo, para cambiar el color de fondo de un párrafo que tiene el identificador `mensaje`:

```
var parrafo = document.getElementById("mensaje");
parrafo.style.backgroundColor = "yellow";
```

[42] https://developer.mozilla.org/en-US/docs/Web/API/HTMLTableElement/createCaption
[43] https://developer.mozilla.org/en-US/docs/Web/API/ChildNode/after
[44] https://developer.mozilla.org/en-US/docs/Web/API/ParentNode/append
[45] https://developer.mozilla.org/en-US/docs/Web/API/ChildNode/before
[46] https://developer.mozilla.org/en-US/docs/Web/API/ParentNode/prepend
[47] https://developer.mozilla.org/en-US/docs/Web/API/CSS_Object_Model
[48] https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style
[49] https://developer.mozilla.org/en-US/docs/Web/API/Element/className

Cuando la propiedad CSS es solo una palabra, como `margin` o `padding`, se usa el mismo nombre de la propiedad para cambiar el estilo en JavaScript. Sin embargo, si la propiedad CSS tiene guiones (-), como `margin-left` o `padding-right`, el nombre de la propiedad CSS se debe convertir al estilo "camelCase". Por eso, en el ejemplo anterior, en vez de `background-color` se debe escribir `backgroundColor`.

La propiedad `style` tiene la misma prioridad en CSS que una declaración de estilo en línea establecida a través del atributo `style` en el elemento.

En el Cuadro 1 se muestran algunos ejemplos de propiedades CSS y cómo se escribe cada propiedad en JavaScript.

Propiedad en CSS	**Propiedad en JS**
background-color	backgroundColor
border-radius	borderRadius
color	color
font-family	fontFamily
margin	margin
text-align	textAlign

Cuadro 1: Propiedades en CSS y JavaScript

El siguiente ejemplo muestra un acertijo en el que para visualizar la respuesta se puede pulsar en un botón que muestra u oculta la respuesta al modificar la propiedad `display`:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Cambia el color con style</title>
<script>
function load() {
	document.getElementById("boton1").addEventListener("click", function(){
		if(document.getElementById("boton1").value == "Ver respuesta (con style)") {
			document.getElementById("respuesta").style.display = "inline";
			document.getElementById("boton1").value = "Ocultar respuesta (con style)";
		}
		else {
			document.getElementById("respuesta").style.display = "none";
			document.getElementById("boton1").value = "Ver respuesta (con style)";
		}
	});
}

document.addEventListener("DOMContentLoaded", load, false);
</script>
<style>
#respuesta {
	display: none;
}
</style>
</head>
<body>
<p>
¿Qué ser, provisto de una sola voz, camina primero de cuatro patas por la mañana, después sobre
    dos patas al mediodía y finalmente con tres patas al atardecer?
</p>
<p>
<input type="button" value="Ver respuesta (con style)" id="boton1">
</p>
<p>Respuesta: <span id="respuesta">El hombre</span></p>
</body>
</html>
```

La propiedad `className`[50] permite obtener y modificar la propiedad `class` de un elemento. Las diferentes clases que se aplican a un elemento se separan con espacios en blanco.

El siguiente ejemplo muestra un acertijo en el que para visualizar la respuesta se puede pulsar en un botón que muestra u oculta la respuesta al modificar la propiedad `className`:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Cambia el color con className</title>
<script>
function load() {
	document.getElementById("boton2").addEventListener("click", function(){
		if(document.getElementById("boton2").value == "Ver respuesta (con className)") {
			document.getElementById("respuesta").className = "visible";
			document.getElementById("boton2").value = "Ocultar respuesta (con
			    className)";
		}
		else {
			document.getElementById("respuesta").className = "noVisible";
			document.getElementById("boton2").value = "Ver respuesta (con className)";
		}
	});
}

document.addEventListener("DOMContentLoaded", load, false);
</script>
<style>
.visible {
	display: inline;
}

.noVisible {
	display: none;
}
</style>
</head>
<body>
<p>
¿Qué ser, provisto de una sola voz, camina primero de cuatro patas por la mañana, después sobre
    dos patas al mediodía y finalmente con tres patas al atardecer?
</p>
<p>
<input type="button" value="Ver respuesta (con className)" id="boton2">
</p>
<p>Respuesta: <span id="respuesta" class="noVisible">El hombre</span></p>
</body>
</html>
```

Además de la propiedad `className`, también existe la propiedad `classList`[51], que devuelve una lista con las clases de un elemento. La lista se puede consultar y modificar con los elementos `add()`, `contains()`, `remove()`, `replace()` y `toggle()`.

4.6. Cuadro de diálogo modal

Según el sitio web "Can I use...?"[52], desde marzo 2022 se puede emplear `<dialog>`[53] en la mayoría de los dispositivos y navegadores web (Figura 3). Este elemento permite crear un cuadro de diálogo modal de forma nativa, con muy poco código CSS y JavaScript.

[50] Se emplea "className" y no "class" por que esta última es una palabra reservada de JavaScript.
[51] `https://developer.mozilla.org/en-US/docs/Web/API/Element/classList`
[52] `https://caniuse.com/?search=dialog`
[53] `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/dialog`

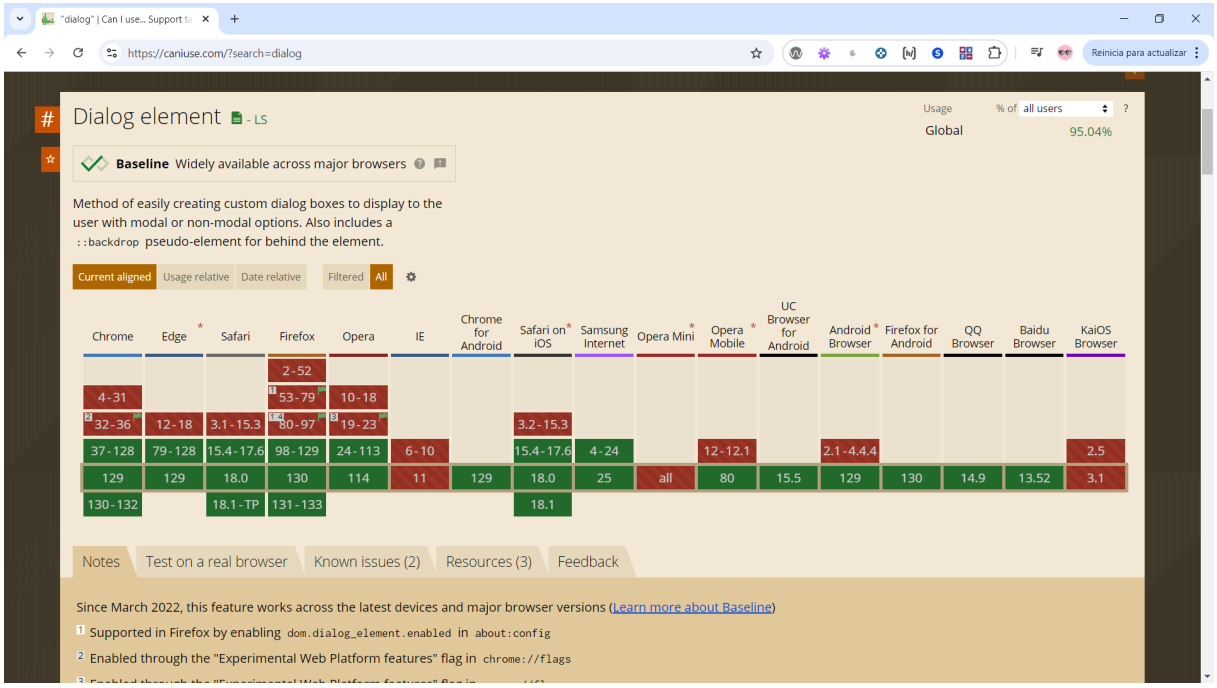

Figura 3: Soporte del elemento dialog en los principales navegadores web según el sitio web "Can I use...?"

El API del elemento dialog[54] incluye los métodos `close()`, `show()` y `showModal()`. Además, se le puede aplicar el pseudoelemento `::backdrop` de CSS para configurar la presentación de todo lo que queda detrás del cuadro de diálogo.

El siguiente ejemplo muestra una página web con un botón "Iniciar sesión" que al pulsarse muestra el cuadro de diálogo que se puede ver en la Figura 4; este cuadro de diálogo es modal, por lo que no se puede interactuar con el resto de la página y se puede cerrar al pulsar en el botón "Cancelar" o al pulsar la tecla Escape:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Cuadro de diálogo modal</title>
<script>
function load() {
	document.getElementById("iniciar").addEventListener("click", () => {
		document.getElementById("dialogo").showModal();
	});

	document.getElementById("cancelar").addEventListener("click", () => {
		document.getElementById("dialogo").close();
	});
}

document.addEventListener("DOMContentLoaded", load, false);
</script>
<style>
dialog::backdrop {
	background: #fff5;
	backdrop-filter: blur(4px);
}
</style>
</head>
<body>
```

[54] https://developer.mozilla.org/en-US/docs/Web/API/HTMLDialogElement

Figura 4: Ejemplo de cuadro de diálogo modal con dialog

```
<p>
<button type="button" id="iniciar">Iniciar sesión</button>
</p>

<dialog id="dialogo">
<form action="entrar.php" method="post">
<h2>Inicio de sesión</h3>
<p><label>Nombre de usuario: <input type="text" name="nombre"></label></p>
<p><label>Contraseña: <input type="password" name="contra"></label></p>
<p><button type="button" id="cancelar">Cancelar</button> <button type="submit">Entrar</button></
    p>
</form>
</dialog>
</body>
</html>
```

En HTML5 también existen los popovers, que se pueden confundir con los cuadro de diálogo, pero presentan algunas diferencias importantes[55].

5. Recomendaciones

Muy importante: en la realización de las prácticas **no debes** emplear un *framework* de JavaScript.

Cuanta mayor separación haya entre las distintas partes que componen una página web (HTML, CSS y JavaScript) mucho mejor. Por ello, el código de JavaScript escríbelo en un fichero aparte e inclúyelo en aquellas páginas donde lo necesites con la etiqueta `<script src=""></script>`.

En el sitio web W3Schools puedes encontrar el apartado **JavaScript Form Validation**[56] que explica algunos tipos de validación sencilla de un formulario. Además de esta página existen muchas otras más como:

- **How-To: Form validation with JavaScript**[57]: explica paso a paso como realizar una validación sencilla. Además, explica cómo validar botones de radio, casillas de verificación y listas desplegables.
- **The Art of Web: JavaScript**[58]: explica cómo realizar algunos tipos de validación especiales (contraseña, fecha y hora, tarjeta de crédito).

Para manipular las cadenas en JavaScript existe el objeto `String`. Consulta la página **JavaScript String Object Reference**[59] en W3Schools para conocer los métodos y propiedades que posee este objeto. Algunos métodos que puedes necesitar son:

- `charAt()`: devuelve el carácter situado en la posición indicada.

[55] https://frontendmasters.com/blog/whats-the-difference-between-htmls-dialog-element-and-popovers/
[56] http://www.w3schools.com/js/js_form_validation.asp
[57] http://www.elated.com/articles/form-validation-with-javascript/
[58] http://www.the-art-of-web.com/javascript/
[59] http://www.w3schools.com/jsref/jsref_obj_string.asp

- `charCodeAt()`: devuelve un número que indica el valor Unicode del carácter situado en la posición indicada. Los primeros 128 códigos de Unicode encajan directamente con los códigos de caractéres de la codificación ASCII.
- `endsWith()`: determina si una cadena termina con los caracteres de otra cadena, devolviendo true o false según corresponda.
- `String.fromCharCode()`: método estático que devuelve una cadena creada mediante el uso de una secuencia de valores Unicode especificada.
- `includes()`: determina si una cadena de texto puede ser encontrada dentro de otra cadena de texto, devolviendo true o false según corresponda.
- `indexOf()`: devuelve la posición de la primera aparición de una cadena en otra cadena, buscando hacia delante desde la posición indicada.
- `lastIndexOf()`: devuelve la posición de la última aparición de una cadena en otra cadena, buscando hacia atrás desde la posición indicada.
- `replace()`: reemplaza unos caracteres por otros caracteres en una cadena.
- `slice()`: extrae una parte de una cadena.
- `split()`: divide una cadena en partes según los caracteres indicados.
- `startsWith()`: determina si una cadena empieza con los caracteres de otra cadena, devolviendo true o false según corresponda.
- `toLowerCase()`: convierte una cadena a minúsculas.
- `toUpperCase()`: convierte una cadena a mayúsculas.

Para trabajar con fechas en JavaScript existe el objeto `Date`. Consulta la página **JavaScript Date Object Reference**[60] en W3Schools para conocer los métodos y propiedades que posee este objeto. Algunos métodos que puedes necesitar son:

- `getDate()`: devuelve el día del mes (de 1 a 31).
- `getMonth()`: devuelve el mes (de 0 a 11).
- `getFullYear()`: devuelve el año, como un número de cuatro dígitos.
- `getTime()`: devuelve una fecha y hora (instante de tiempo) como el número de milisegundos desde la medianoche del 1 de enero de 1970.
- `setDate()`: fija el día del mes (de 1 a 31).
- `setMonth()`: fija el mes (de 0 a 11).
- `setFullYear()`: fija el año, como un número de cuatro dígitos.
- `setTime()`: fija una fecha y hora (instante de tiempo) como el número de milisegundos desde la medianoche del 1 de enero de 1970.

El objeto `Date` se puede emplear para validar una fecha introducida por el usuario. ¿Cómo? Utiliza la fecha introducida por el usuario para crear un objeto de tipo `Date` y compara la fecha introducida con los valores que devuelve el objeto.

Cuando una página contiene un error de JavaScript, los navegadores suelen informar de alguna manera al usuario. Por ejemplo, la siguiente página contiene un error, ya que el bloque de código de la función `prueba()` no está cerrado (falta una llave).

[60] `http://www.w3schools.com/jsref/jsref_obj_date.asp`

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>Prueba de JavaScript</title>
<script>
function prueba() {
window.alert("Mensaje 1");
alert("Mensaje 2");

return false;
</script>
</head>
<body>
<p>
Un texto. Pulsa <a href="#" onclick="return prueba()">aquí</a>
para ejecutar la función.
</p>
</body>
</html>
```

En los navegadores actuales existe la posibilidad de consultar los mensajes de error de JavaScript a través de la consola de las herramientas para desarrolladores. En la Figura 5 se muestra el mensaje de error que muestra el navegador Google Chrome y en la Figura 6 el mensaje de error que muestra el navegador Mozilla Firefox.

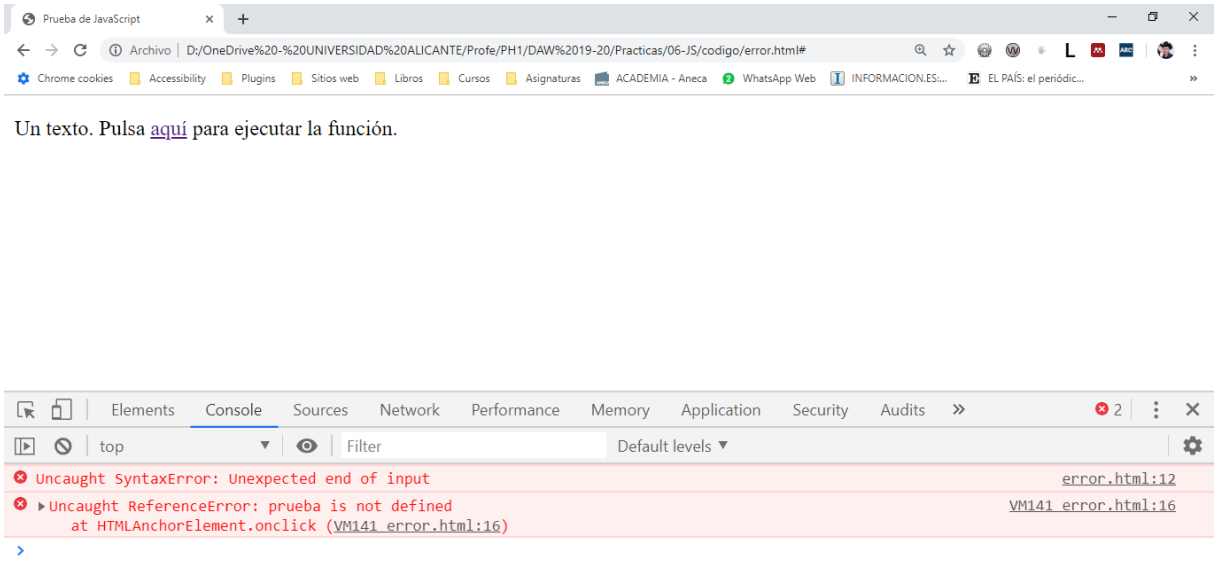

Figura 5: Consola de errores en Google Chrome

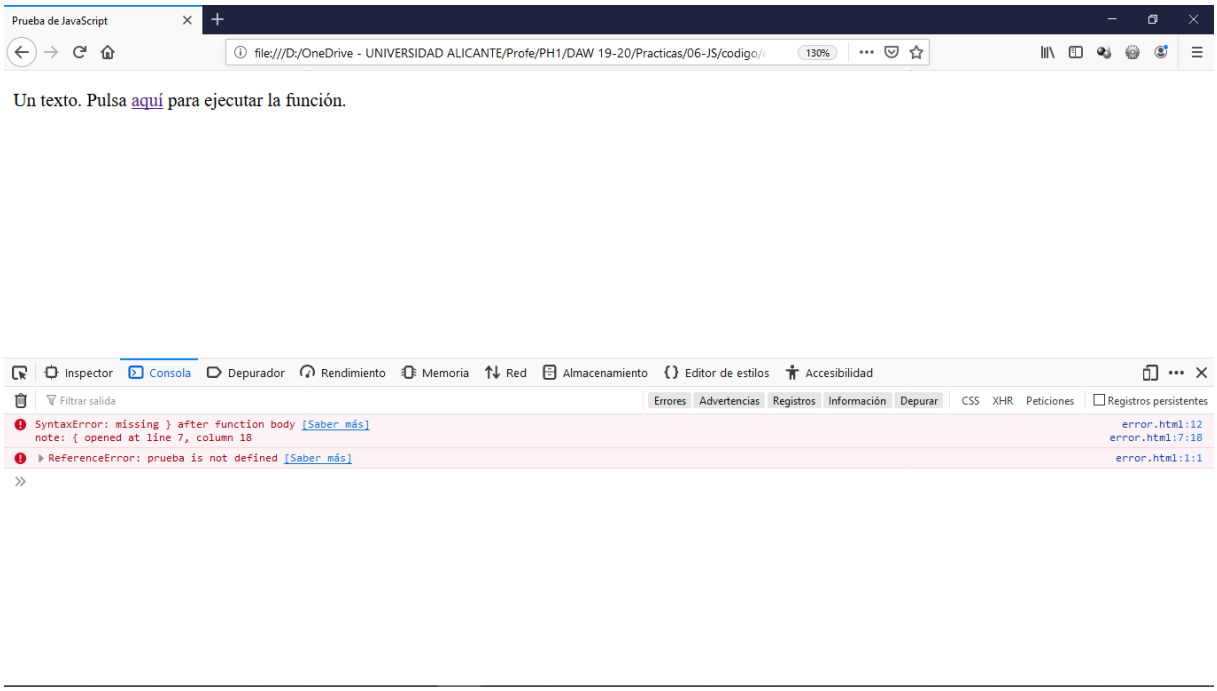

Figura 6: Consola de errores en Mozilla Firefox