

Gestión de eventos y formularios en JavaScript.



Caso práctico

Antonio, afronta una de las partes más interesantes en su estudio de JavaScript. Se trata de la gestión de los eventos en JavaScript y cómo validar formularios.



Antonio estaba deseando llegar a esta sección, ya que su trabajo de desarrollo en el proyecto se va a centrar en muchas de las cosas que va a ver ahora. Va a tener que validar todos los formularios de la web antigua y tendrá que hacerlo con JavaScript, dando mensajes al usuario, sobre los posibles errores que se vaya encontrando durante la validación. La validación de un formulario es primordial, ya que lo que se busca es que cuando los datos sean enviados al servidor, vayan lo más coherentes posible.

Gracias a la gestión de eventos, **Antonio** podrá, por ejemplo, capturar acciones del usuario cuando pulse un botón o cuando pase el ratón por zonas del documento, al introducir datos, etc.

Juan está también muy ilusionado con los progresos de **Antonio** durante todo este tiempo, y le facilita toda la documentación y algunos ejemplos interesantes de validaciones de datos con JavaScript.

En esta unidad de trabajo se explica la forma de acceso a los formularios, a sus propiedades y métodos. Se ven los objetos principales relacionados con los formularios y se explica la forma de pasar a una función la referencia a un objeto.

Se hace una introducción a lo que son los eventos y se analizan los diferentes modelos de registro de eventos en JavaScript, para terminar viendo el orden de disparo de los eventos.

Una vez vistos formularios y eventos se pasa a una parte más práctica en la que se explica cómo llevar a cabo la validación de un formulario y cómo utilizar expresiones regulares para validar el contenido de los campos de tipo texto de nuestros formularios.

Por último se hace una introducción a lo que son las cookies y cómo gestionarlas con JavaScript.



[Ministerio de Educación y Formación Profesional](#) (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- El objeto Form.



Caso práctico

Los formularios, son el principal medio de introducción de datos en una aplicación web, y el principal punto de interactividad con el usuario.

Es aquí donde JavaScript, nos va a permitir aportar toda esa interactividad a los elementos estáticos de un formulario HTML.



Es muy importante conocer al detalle todos los objetos y métodos de los formularios, ya que un porcentaje muy alto de la interactividad que se produce en una página web, proviene del uso de formularios.

La forma de acceso a un formulario, cómo referenciarlo, su estructura desde el punto de vista de JavaScript y su contenido, será lo que **Antonio** estudiará en este momento.

La mayor parte de interactividad entre una página web y el usuario tiene lugar a través de un formulario. Es ahí donde nos vamos a encontrar con los campos de texto, botones, checkboxes, listas, etc. en los que el usuario introducirá los datos, que luego se enviarán al servidor.

En este apartado verás cómo identificar un formulario y sus objetos, cómo modificarlos, cómo examinar las entradas de usuario, enviar un formulario, validar datos, etc.

Los formularios y sus controles, son objetos del DOM que tienen propiedades únicas, que otros objetos no poseen. Por ejemplo, un formulario tiene una propiedad `action`, que le indica al navegador donde tiene que enviar las entradas del usuario, cuando se envía el formulario. Un control `select` posee una propiedad llamada `selectedIndex`, que nos indica la opción de ese campo que ha sido seleccionada por el usuario.

JavaScript añade a los formularios dos características muy interesantes:

- ✓ JavaScript permite examinar y validar las entradas de usuario directamente, en el lado del cliente.
- ✓ JavaScript permite dar mensajes instantáneos, con información de la entrada del usuario.

El objeto de JavaScript `Form`, es una propiedad del objeto `document`. Se corresponderá con la etiqueta `<form>` del HTML. Un formulario podrá ser enviado llamando al método `submit` de JavaScript, o bien haciendo click en el botón `submit` del formulario.



Reflexiona

¿Si, por ejemplo, usamos JavaScript para validar un formulario, será necesario también validar esos datos en el lado del servidor?

¿Qué pasaría con nuestras validaciones si por ejemplo desactivamos JavaScript en el navegador?

¿Y que pasaría si alguien programa una copia de nuestro formulario en otro servidor web para enviar datos a nuestro servidor sin validarlos previamente?

1.1.- Formas de selección del objeto Form.

Dentro de un documento tendremos varias formas de selección de un formulario.

Si partimos del siguiente ejemplo:

```
<div id="menulateral">
  <form id="contactar" name="contactar" action="...">...</form>
</div>
```



[Everaldo Coelho \(GNU/GPL\)](#)

Tendremos los siguientes métodos de selección del objeto `Form` en el documento:

Método 1

Método 2

Método 3

Método 1

A través del método `getElementById()` del DOM, nos permite acceder a un objeto a través de su atributo ID. Tendremos que tener la precaución de asignar id únicos a nuestros objetos, para evitar que tengamos objetos con id repetidos.

Ejemplo:

```
let formulario = document.getElementById("contactar");
```

Método 2

A través del método `getElementsByName()` del DOM, el cuál nos permite acceder a un objeto a través de la etiqueta HTML que queramos. Por ejemplo para acceder a los objetos con etiqueta `form` haremos:

```
let formularios = document.getElementsByName("form");
let primerFormulario = formularios[0]; // primer formulario del documento
```

o también todo en una única línea:

```
let primerFormulario = document.getElementsByTagName("form")[0] ;
```

Otra posibilidad interesante que te permite el método anterior, es la de buscar objetos con un padre determinado, por ejemplo:

```
let menu = document.getElementById("menulateral");  
let formularios = menu.getElementsByTagName("form"); // formularios contenidos  
let primerFormulario = formularios[0]; // primer formulario en el menu
```

Método 3

Otro método puede ser a través de la colección `forms[]` del **objeto** `document`. Esta colección es un array, que contiene la referencia a todos los formularios que tenemos en nuestro documento.

Por ejemplo:

```
let formularios = document.forms; // la referencia a todos los formularios del documento  
let miFormulario = formularios[0]; // primer formulario del documento
```

o bien:

```
let miFormulario = document.forms[0]; // primer formulario del documento
```

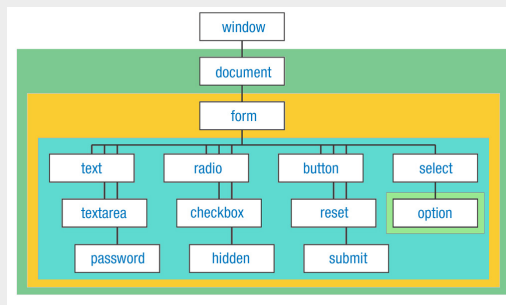
o bien:

```
let miFormulario = formularios["contactar"]; // referenciamos al formulario con su nombre
```

1.2.- El formulario como objeto y contenedor.

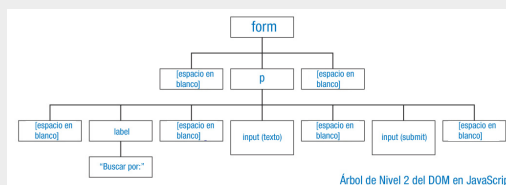
Debido a que el DOM ha ido evolucionando con las nuevas versiones de JavaScript, nos encontramos con que el objeto `Form` está dentro de dos árboles al mismo tiempo. En las nuevas definiciones del DOM, se especifica que `Form` es el padre de todos sus nodos hijos, incluidos objetos y textos, mientras que en las versiones antiguas `Form` sólo era padre de sus objetos (`input`, `select`, `button` y elementos `textarea`).

Jerarquía de nivel 0 del DOM para formularios y controles:



[raveiga](#) (CC BY)

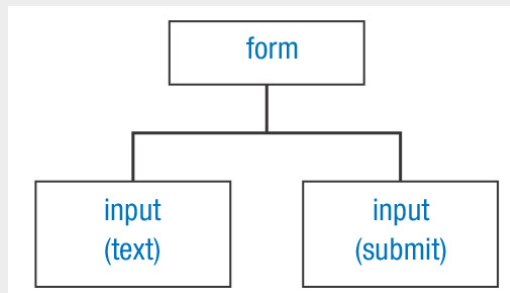
Árbol de nivel 2 del DOM para un formulario típico



Árbol de Nivel 2 del DOM en JavaScript

[raveiga](#) (CC BY)

Árbol de nivel 0 para el mismo formulario



[raveiga](#) (CC BY)

1

2

3

Los dos árboles que te mostré anteriormente pueden ser útiles para diferentes propósitos. El árbol del DOM de nivel 2, se puede utilizar para leer y escribir en todo el documento con un nivel muy fino de 🗑 granuralidad. El árbol del DOM de nivel 0, hace muchísimo más fácil leer y escribir los controles del formulario.



Citas para pensar

"Son las palabras y las fórmulas, más que la razón, las que crean la mayoría de nuestras opiniones."

LE BON, Gustave

1.3.- Acceso a propiedades y métodos del formulario.

Los formularios pueden ser creados a través de las etiquetas HTML, o utilizando JavaScript y métodos del DOM. En cualquier caso se pueden asignar atributos como `name`, `action`, `target` y `enctype`. Cada uno de estos atributos es una propiedad del objeto `Form`, a las que podemos acceder utilizando su nombre en minúsculas, por ejemplo:



[Everaldo Coelho](#) (GNU/GPL)

```
let paginaDestino = objetoFormulario.action;
```

Para modificar una de estas propiedades lo haremos mediante asignaciones, por ejemplo:

```
objetoFormulario.action = "http://www.educacion.gob.es/recepcion.php";
```

Estas dos instrucciones las podemos recomponer usando referencias a objetos:

```
let paginaDestino = document.getElementById("id").action;  
document.forms[0].action = "http://www.educacion.gob.es/recepcion.php";
```

Propiedades del objeto Form

Propiedad	Descripción	W3C
<code>acceptCharset</code>	Ajusta o devuelve el valor del atributo <code>accept-charset</code> en un formulario.	Sí
<code>action</code>	Ajusta o devuelve el valor del atributo <code>action</code> en un formulario.	Sí
<code>enctype</code>	Ajusta o devuelve el valor del atributo <code>enctype</code> en un formulario.	Sí
<code>length</code>	Devuelve el número de elementos en un formulario.	Sí
<code>method</code>	Ajusta o devuelve el valor del atributo <code>method</code> en un formulario.	Sí
<code>name</code>	Ajusta o devuelve el valor del atributo <code>name</code> en un formulario.	Sí
<code>target</code>	Ajusta o devuelve el valor del atributo <code>target</code> en un formulario.	Sí

Métodos del objeto Form

Método	Descripción	W3C
--------	-------------	-----

Método	Descripción	W3C
reset()	Resetea un formulario.	Sí
submit()	Envía un formulario.	Sí

Propiedad `form.elements[]`

La propiedad `elements[]` de un formulario es una colección, que contiene todos los objetos `input` dentro de un formulario. Esta propiedad es otro array, con todos los campos `input` en el orden en el cual aparecen en el código fuente del documento.

Generalmente, es mucho más eficaz y rápido referenciar a un elemento individual usando su ID, pero a veces, los scripts necesitan recorrer cada elemento del formulario, para comprobar que se han introducido sus valores correctamente.

Por ejemplo, empleando la propiedad `elements[]`, podemos hacer un bucle que recorra un formulario y si los campos son de tipo texto, pues que los ponga en blanco:

```
let miFormulario = document.getElementById("contactar"); // guardamos la referencia del fo
if (! miFormulario) return false; // Si no existe ese formulario devuelve false.
for (let i=0; i< miFormulario.elements.length; i++) {
    if (miFormulario.elements[i].type == "text") {
        miFormulario.elements[i].value = "";
    }
}
```

2.- Objetos relacionados con formularios.



Caso práctico

Una vez visto cómo referenciar a un formulario en JavaScript, tenemos que saber cómo acceder a cada uno de los elementos u objetos, que contiene ese formulario.

Cada uno de los elementos de un formulario, son objetos en JavaScript que tendrán propiedades y métodos, que nos permitirán realizar acciones sobre ellos. Gracias a esos métodos y propiedades, podremos realizar acciones como validar el contenido de un formulario, marcar o desmarcar una determinada opción, mostrar contenido de un campo u ocultarlo, etc.

Juan le indica a **Antonio**, que es muy importante que comprenda esta parte, ya que es la base de los procesos de validación que estudiará más adelante, y es una de las tareas que más veces va a tener que realizar en su proyecto de actualización de la web.



Para poder trabajar con los objetos de un formulario, lo primero que necesitas saber es, cómo referenciar a ese objeto. Eso puedes hacerlo directamente a través de su ID, o bien con su nombre de etiqueta, empleando para ello los métodos del DOM nivel 2. O también se puede hacer usando la sintaxis del DOM nivel 0, y construyendo la jerarquía que comienza por `document`, luego el formulario y finalmente el control.

Lo mejor es identificar cada uno de los objetos con un atributo `id` que sea único, y que no se repita en el documento, así para acceder a cualquier objeto dentro de nuestro documento o formulario lo haremos con:

```
document.getElementById("id-del-control");
```

o

```
document.nombreFormulario.name-del-control;
```

Por ejemplo, si consideramos un ejemplo sencillo de formulario:

```
<form id="formularioBusqueda" action="cgi-bin/buscar.pl">
  <p>
    <input type="text" id="entrada" name="cEntrada">
    <input type="submit" id="enviar" name="enviar" value="Buscar...">
  </p>
</form>
```

Las siguientes referencias al campo de texto entrada, serán todas válidas:

```
document.getElementById("entrada");
document.formularioBusqueda.cEntrada;
document.formularioBusqueda.elements[0];
document.forms["formularioBusqueda"].elements["cEntrada"];
document.forms["formularioBusqueda"].cEntrada;
```

Aunque muchos de los controles de un formulario tienen propiedades en común, algunas propiedades son únicas a un control en particular. Por ejemplo, en un objeto `select` tienes propiedades que te permiten conocer la opción que está actualmente seleccionada. Al igual que los `checkboxes` o los botones de tipo `radio`, que también disponen de propiedades para saber cuál es la opción que está actualmente seleccionada.



Autoevaluación

A la hora de identificar los objetos en un formulario lo más recomendable es que el atributo `id` y el atributo `name` sean iguales y que no se repitan los `id` en todo el documento:

- ☐ Verdadero.
- ☐ Falso.

Es correcto, ya que de esta forma podremos referenciar a cualquier objeto del documento sin tener ningún tipo de conflicto.

Es incorrecto, es una de las recomendaciones que deberemos seguir para evitar problemas a la hora de programar con JavaScript.

Solución

1. Opción correcta
2. Incorrecto

2.1.- Objeto input de tipo texto.


Cada uno de los 4 elementos de tipo texto de los formularios: `text`, `password`, `hidden` y elementos `textarea`, son un elemento dentro de la jerarquía de objetos. Todos los elementos, excepto los tipos `hidden`, se mostrarán en la página, permitiendo a los usuarios introducir texto y seleccionar opciones.

Para poder usar estos objetos dentro de nuestros scripts de JavaScript, simplemente será suficiente con asignar un atributo `id`, a cada uno de los elementos. Te recomiendo que asignes a cada objeto de tu formulario un atributo `id` único y que coincida con el `name` de ese objeto.



[Everaldo Coelho \(GNU/GPL\)](#)

Cuando se envían los datos de un formulario a un programa en el lado del servidor, lo que en realidad se envía son los atributos `name`, junto con los valores (contenido del atributo `value`) de cada elemento. Sin lugar a dudas, la propiedad más utilizada en un elemento de tipo texto es por lo tanto `value`. Un script podrá recuperar y ajustar el contenido de la propiedad `value` en cualquier momento. Por cierto, el contenido de un `value` es siempre una cadena de texto, y quizás puedas necesitar realizar conversiones numéricas si quieres realizar operaciones matemáticas con esos textos.

En este tipo de objetos, los gestores de eventos (que verás más adelante) se podrán  disparar de múltiples formas: por ejemplo, cuando pongamos el foco en un campo (situar el cursor dentro de ese campo) o modifiquemos el texto (al introducir el texto y salir del campo).

Ejemplo de acceso al valor del campo de texto

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
    <title>DWEC05 - Propiedad VALUE de un objeto de tipo texto</title>
    <script type="text/javascript">
      const convertirMayusculas = () => {

        /*
         * En este ejemplo accedemos a la propiedad value de un objeto con id nombre
         * su contenido actual pero convertido a mayúsculas con el método toUpperCase
         */
        document.getElementById("nombre").value=document.getElementById("nombre").value.toUpperCase()
      }
    </script>
  </head>
  <body>
    <h1>Propiedad VALUE de un objeto INPUT de tipo TEXT</h1>
    <form id="formulario" action="pagina.php">
      <p>
        <label for="nombre">Nombre y Apellidos: </label>
        <input type="text" id="nombre" name="nombre" value="" size="30" onblur="convertirMayusculas()" />
      </p>
    </form>
  </body>
</html>
```

```
Introduce tu Nombre y Apellidos y haz click fuera del campo.  
</p>  
</form>  
</body>  
</html>
```

Propiedades del objeto INPUT de tipo texto

Propiedad	Descripción	W3C
defaultValue	Ajusta o devuelve el valor por defecto de un campo de texto.	Sí
form	Devuelve la referencia al formulario que contiene ese campo de texto.	Sí
maxLength	Devuelve o ajusta la longitud máxima de caracteres permitidos en el campo de tipo texto	Sí
name	Ajusta o devuelve el valor del atributo <code>name</code> de un campo de texto.	Sí
readOnly	Ajusta o devuelve si un campo es de sólo lectura, o no.	Sí
size	Ajusta o devuelve el ancho de un campo de texto (en caracteres).	Sí
type	Devuelve el tipo de un campo de texto.	Sí
value	Ajusta o devuelve el contenido del atributo <code>value</code> de un campo de texto.	Sí

Métodos del objeto INPUT de tipo texto

Metodo	Descripción	W3C
select()	Selecciona el contenido de un campo de texto.	Sí

Además de los métodos anteriores, los campos de tipo texto también soportan todas las propiedades estándar, métodos y eventos.



Para saber más

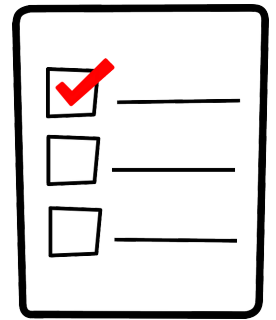
En los siguientes enlace podrás ampliar información sobre las propiedades y métodos de los objetos `input` y encontrarás más ejemplos.

[Más información sobre las propiedades estándar, métodos y eventos.](#)

[Más información y ejemplos sobre objeto input](#)

2.2.- Objeto input de tipo checkbox.

Un `checkbox` es también un objeto muy utilizado en los formularios, pero algunas de sus propiedades pueden que no sean muy intuitivas. En los botones de un formulario la propiedad `value` nos mostrará el texto del botón, pero en un `checkbox` la propiedad `value` es un texto que está asociado al objeto. Este texto no se mostrará en la página, y su finalidad es la de asociar un valor con la opción actualmente seleccionada. Dicho valor será el que se enviará, cuando enviemos el formulario.



[OpenClipart-Vectors](#) ([Pixabay License](#))

Por ejemplo:

```
<label for="cantidad">Si desea recibir 20 Kg marque esta opción: </label>
<input type="checkbox" id="cantidad" name="cantidad" value="20 Kg">
```

Si chequeamos este `checkbox` y enviamos el formulario, el navegador enviará el par `name/value` "cantidad" y "20 Kg". Si el `checkbox` no está marcado, entonces este campo no será enviado en el formulario. El texto del `label` se mostrará en la pantalla pero las etiquetas `label` no se envían al servidor. Para saber si un campo de tipo `checkbox` está o no marcado, disponemos de la propiedad `checked`. Esta propiedad contiene un valor `booleano`: `true` si el campo está marcado o `false` si no está marcado. Con esta propiedad es realmente sencillo comprobar o ajustar la marca en un campo de tipo `checkbox`.

Veamos el siguiente ejemplo muy simple, pero en el que manejamos dicha propiedad:

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      const marcar = () => {
        document.getElementById("verano").checked = true;
      }
      const desmarcar = () => {
        document.getElementById("verano").checked = false;
      }
    </script>
  </head>
  <body>
    <form action="" method="get">
      <label for="verano">¿Te gusta el verano?</label>
      <input type="checkbox" id="verano" name="verano" value="Si"/>
    </form>
    <button onclick="marcar()">Marcar Checkbox</button>
    <button onclick="desmarcar()">Desmarcar Checkbox</button>
  </body>
</html>
```


Propiedades del objeto INPUT de tipo checkbox

Propiedad	Descripción	W3C
<code>checked</code>	Ajusta o devuelve el estado <code>checked</code> de un <code>checkbox</code> .	Sí
<code>defaultChecked</code>	Devuelve el valor por defecto del atributo <code>checked</code> .	Sí
<code>form</code>	Devuelve la referencia al formulario que contiene ese campo <code>checkbox</code> .	Sí
<code>name</code>	Ajusta o devuelve el valor del atributo <code>name</code> de un <code>checkbox</code> .	Sí
<code>type</code>	Nos indica que tipo de elemento de formulario es un <code>checkbox</code> .	Sí
<code>value</code>	Ajusta o devuelve el valor del atributo <code>value</code> de un <code>checkbox</code> .	Sí

2.3.- Objeto input de tipo radio.

El ajustar un grupo de objetos de tipo radio desde JavaScript requiere un poco más de trabajo. Para dejar que el navegador gestione un grupo de objetos de tipo radio, deberemos asignar el mismo atributo `name` a cada uno de los botones del grupo. Podemos tener múltiples grupos de botones de tipo radio en un formulario, pero cada miembro de cada grupo tendrá que tener el mismo atributo `name` que el resto de compañeros del grupo.



[Ciker-Free-Vector-Images](#) (Pixabay License)

Cuando le asignamos el mismo `name` a varios elementos en un formulario, el navegador lo que hace es crear un array con la lista de esos objetos que tienen el mismo `name`. El contenido del atributo `name` será el nombre del array. Algunas propiedades, se las podremos aplicar al grupo como un todo; otras en cambio, tendremos que aplicárselas a cada elemento del grupo y lo haremos a través del índice del array del grupo. Por ejemplo, podemos ver cuantos botones hay en un grupo `radio`, consultando la propiedad `length` de ese grupo:

```
objetoFormulario.nombregrupo.length;
```

Y si queremos acceder a la propiedad `checked` de un botón en particular, lo haremos accediendo a la posición del array y a la propiedad `checked`:

```
objetoFormulario.nombregrupo[0].checked; // Accedemos a la propiedad checked del primer b
```

Ejemplo:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>DWEC05 - Trabajando con objetos input de tipo radio</title>
    <script type="text/javascript">
      const mostrarDatos = () => {
        for (let i=0;i<document.formulario.actores.length; i++) {
          if (document.formulario.actores[i].checked)
            alert(document.formulario.actores[i].value);
        }
      }
    </script>
  </head>
  <body>
    <h1>Trabajando con objetos input de tipo radio</h1>
    <form name="formulario" action="stooges.php">
      <fieldset>
```

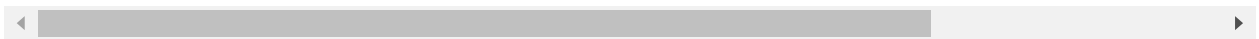
```
<legend>Selecciona tu actor favorito:</legend>
<label for="actor1">Willis</label>
<input type="radio" name="actores" id="actor1" value="Walter Bruce Willis - 19
<label for="actor2">Carrey</label>
<input type="radio" name="actores" id="actor-2" value="James Eugene Jim Carre
<label for="actor3">Tosar</label>
<input type="radio" name="actores" id="actor-3" value="Luis Tosar - 13 de Oct
<input type="button" id="consultar" name="consultar" value="Consultar Más Dat
</fieldset>
</form>
</body>
</html>
```



Para saber más

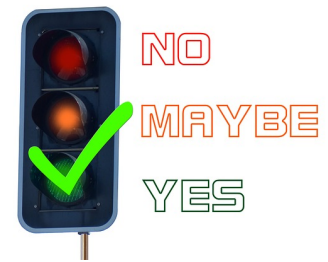
En el siguiente enlace podrás profundizar en las propiedades y métodos de este objeto.

[Propiedades y métodos del objeto `radio`.](#)



2.4.- Objeto select.

Uno de los controles más complejos que te puedes encontrar en formularios, es el objeto `select`. Un objeto `select` está compuesto realmente de un array de objetos `option`. El objeto `select` se suele mostrar como una lista desplegable en la que seleccionas una de las opciones, aunque también tienes la opción de selecciones múltiples, según definas el objeto en tu documento. Vamos a ver cómo gestionar una lista que permita solamente selecciones sencillas.




[geralt](#) (Pixabay License)

Algunas propiedades pertenecen al objeto `select` al completo, mientras que otras, por ejemplo, sólo se pueden aplicar a las opciones individuales dentro de ese objeto. Si lo que quieres hacer es detectar la opción seleccionada por el usuario, y quieres usar JavaScript, tendrás que utilizar propiedades tanto de `select`, como de `option`.

La propiedad más importante del objeto `select` es la propiedad `selectedIndex`, a la que puedes acceder de las siguientes formas:

```
objetoFormulario.nombreCampoSelect.selectedIndex;  
document.getElementById("objetoSelect").selectedIndex;
```

El valor devuelto por esta propiedad, es el  índice de la opción actualmente seleccionada, y por supuesto recordarte que los índices comienzan en la posición 0. Siempre que queramos acceder a la opción actualmente seleccionada, recuerda que tendrás que usar esta propiedad.

Las opciones tienen dos propiedades accesibles que son `text` y `value`, y que te permitirán acceder al texto visible en la selección y a su valor interno para esa opción (ejemplo: `<option value="OU">Ourense</option>`). Veamos las formas de acceso a esas propiedades:

```
objetoFormulario.nombreCampoSelect.options[n].text;  
objetoFormulario.nombreCampoSelect.options[n].value;
```

o también usando `document.getElementById("objetoSelect").options[n].text` Ó `.value`

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta http-equiv="content-type" content="text/html; charset=utf-8">  
    <title>DWEC05 - Trabajando con un objeto Select</title>  
    <script type="text/javascript">  
      const consultar = () => {  
        let provincias = document.getElementById("provincias");  
        let texto = provincias.options[provincias.selectedIndex].text;  
        let valor = provincias.options[provincias.selectedIndex].value;  
        alert(`Datos de la opción seleccionada:\n\nTexto: ${texto}\nValor: ${valor}`);  
      };  
    </script>  
  </head>  
</html>
```

```

    }
  </script>
</head>
<body>
  <h1>Trabajando con un objeto Select</h1>
  <form id="formulario">
    <p>
      <label for="provincias">Seleccione provincia: </label>
      <select name="provincias" id="provincias">
        <option value="AL">Almería</option>
        <option value="JA">Jaen</option>
        <option value="GR">Granada</option>
        <option value="MA">Málaga</option>
        <option value="SE">Sevilla</option>
        <option value="CO">Córdoba</option>
        <option value="CA">Cádiz</option>
        <option value="HU">Huelva</option>
      </select>
    </p>
    <p>
      Selecciona una opción y pulsa el botón.
    </p>
    <input type="button" name="boton" value="Consultar información de la opción" onclick=
  </form>
</body>
</html>

```



Debes conocer

En el siguiente enlace podrás ampliar conocimientos sobre las propiedades y métodos del objeto `Select`.

[Propiedades y métodos del objeto `Select`.](#)

2.5.- Pasando objetos a las funciones usando this.

En los ejemplos que hemos visto anteriormente cuando un gestor de eventos (`onclick`, `onblur`,...) llama a una función, esa función se encarga de buscar el objeto sobre el cuál va a trabajar. En JavaScript disponemos de un método que nos permite llamar a una función, pasándole directamente la referencia del objeto, sin tener que usar variables globales o referenciar al objeto al comienzo de cada función.



[johnhain](#) (Pixabay License)

Para conseguir hacerlo necesitamos usar la palabra reservada *this*, la cuál hace referencia siempre al objeto que contiene el código de JavaScript en donde usamos dicha palabra reservada. Por ejemplo, si programamos una función para un botón, que al hacer click haga algo, si dentro de esa función usamos la palabra *this*, entonces estaremos haciendo referencia al objeto en el cuál hemos hecho click, que en este caso será el botón. El uso de *this* nos permite evitar usar variables globales, y el programar scripts más genéricos.

Por ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
    <title>Uso de la palabra reservada this</title>
    <script type="text/javascript">
      const identificar = (objeto) => {
        let nombre = objeto.name;
        let id = objeto.id;
        let valor = objeto.value;
        let tipo = objeto.type;
        alert(`Datos del campo pulsado:\n\nNombre: ${nombre}\nID: ${id}\nValor: ${valor}\nTipo: ${tipo}`);
      }
    </script>
  </head>
  <body>
    <h1>Trabajando con this</h1>
    <form id="formulario">
      <p>
        <label for="nombre">Nombre: </label>
        <input type="text" name="nombre" id="nombre" value="Bob" onclick="identificar(this)" />
        <label for="apellidos">Apellidos: </label>
        <input type="text" name="apellidos" id="apellidos" value="Esponja" onclick="identificar(this)" />
        <label for="edad">Edad: </label>
        <input type="number" name="edad" id="edad" value="18" onclick="identificar(this)" />
        <label for="pais">País: </label>
        <label for="pais1">España: </label>
        <input type="radio" name="pais" id="pais1" value="ES" onclick="identificar(this)" />
        <label for="pais2">Francia: </label>
        <input type="radio" name="pais" id="pais2" value="FR" onclick="identificar(this)" />
      </p>
      <p>
        Haga click en cada uno de los campos para ver más información.
      </p>
    </form>
  </body>
</html>
```

```
</p>  
</form>  
</body>  
</html>
```

En este ejemplo, cada vez que hagamos click en alguno de los objetos, llamaremos a la función `identificar()` y a esa función le pasaremos como parámetro *this*, que en este caso será la referencia al objeto en el cuál hemos hecho click. La función `identificar()` recibe ese parámetro, y lo almacena en la variable `objeto`, la cuál le permite imprimir todas las referencias al `name`, `id`, `value` y `type`. En el siguiente apartado veremos los eventos y allí te mostraré otro uso de *this* por ejemplo dentro de la función `identificar()` sin tener que pasar ningún parámetro.



Citas para pensar

"Nunca consideres el estudio como un deber, sino como una oportunidad para penetrar en el maravilloso mundo del saber."

EINSTEIN, Albert



3.- Eventos.



Caso práctico

La mayor parte de las veces que un usuario realiza acciones en un formulario está generando eventos. Por ejemplo, cuando hace click con el ratón, cuando sitúa el cursor en un campo, cuando mueve el ratón sobre algún objeto, etc.



Con JavaScript podremos programar que, cuando se produzca alguno de esos eventos, realice una tarea determinada. Es lo que se conoce en programación como **capturar un evento**.

Los modelos de registro de esos eventos, así como el orden en el que esos eventos se generan, es la parte que va a estudiar **Antonio** ahora mismo. Esta parte le ayudará mucho cuando tenga que capturar eventos que se produzcan en secuencia, o cuando quiera cancelar un evento, etc.

Hay que tener en cuenta que, sin eventos prácticamente no hay scripts. En casi todas las páginas web que incorporan JavaScript, suele haber eventos programados que disparan la ejecución de dichos scripts. La razón es muy simple, JavaScript fue diseñado para añadir interactividad a las páginas: el usuario realiza algo y la página reacciona.

Por lo tanto, JavaScript necesita detectar de alguna forma las acciones del usuario para saber cuándo reaccionar. También necesita saber las funciones, que queremos que ejecute cuando se produzcan esas acciones.

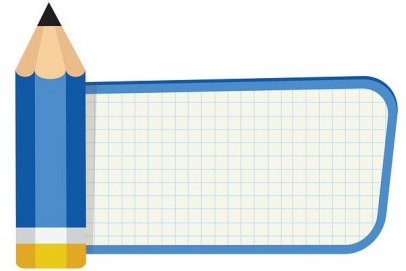
Cuando el usuario hace algo se produce un evento. También habrá algunos eventos que no están relacionados directamente con acciones de usuario: por ejemplo el evento de carga (`load`) de un documento, que se producirá automáticamente cuando un documento ha sido cargado.

A todo ello nos dedicaremos en este apartado, aunque ya en ejemplos anteriores los has ido utilizando casi sin darte cuenta.

3.1.- Modelo de registro de eventos en línea.

En el modelo de registro de eventos en línea, el evento es añadido como un atributo más a la etiqueta HTML, como por ejemplo:

```
<a href="pagina.html" onClick="alert('Has pulsado en el enla
```



[Kidaha \(Pixabay License\)](#)

Cuando hacemos click en el enlace, se llama al gestor de eventos `onClick` (al hacer click) y se ejecuta el script; que contiene en este caso una alerta de JavaScript. También se podría realizar lo mismo pero llamando a una función:

```
const alertar = () => {  
    alert("Has pulsado en el enlace");  
}  
...  
<a href="pagina.html" onClick="alertar()">Pulsa aqui</a>
```

No uses el modelo de registro de eventos en línea

Este modelo no se recomienda, y aunque lo has visto en ejemplos que hemos utilizado hasta ahora, tiene el problema de que estamos mezclando la estructura de la página web con la programación de la misma, y lo que se intenta hoy en día es separar la programación en JavaScript, de la estructura HTML, por lo que este modelo no nos sirve.

En el ejemplo anterior, cuando haces click en el enlace se mostrará la alerta y a continuación te conectará con la `pagina.html`. En ese momento desaparecerán de memoria los objetos que estaban en un principio, cuando se programó el evento. Ésto puede ser un problema, ya que si por ejemplo la función a la que llamamos, cuando se produce el evento, tiene que realizar varias tareas, éstas tendrían que ser hechas antes de que nos conecte con la nueva página.

Éste modo de funcionamiento ha sido un principio muy importante en la gestión de eventos. Si un evento genera la ejecución de un script y además también se genera la acción por defecto para ese objeto entonces:

- 1.- El script se ejecutará primero.
- 2.- La acción por defecto se ejecutará después.

Evitar la acción por defecto

A veces es interesante el bloquear o evitar que se ejecute la acción por defecto. Por ejemplo, en nuestro caso anterior podríamos evitar que nos conecte con la nueva `pagina.html`. Cuando programamos un gestor de eventos, ese gestor podrá devolver un valor booleano `true` o `false`. Eso tendremos que programarlo con la instrucción `return true|false`. False quiere decir

"no ejecutes la acción por defecto". Por lo tanto nuestro ejemplo quedará del siguiente modo:

```
<a href="pagina.html" onClick="alertar(); return false">Pulsa aqui</a>
```

De esa forma, cada vez que pulsemos en el enlace realizará la llamada a la función `alertar()` y cuando termine ejecutará la instrucción `"return false"`, que le indicará al navegador que no ejecute la acción por defecto asignada a ese objeto (en este caso la acción por defecto de un hiperenlace es conectarnos con la página `href` de destino).

También se puede utilizar el método `preventDefault()` del evento para lo mismo.

3.2.- Modelo de registro de eventos tradicional.

En los navegadores antiguos, el modelo que se utilizaba era el modelo en línea. Con la llegada de DHTML, el modelo se extendió para ser más flexible. En este nuevo modelo el evento pasa a ser una propiedad del elemento, así que por ejemplo los navegadores modernos ya aceptan el siguiente código de JavaScript:

```
elemento.onclick = hacerAlgo; // cuando el usuario haga cli
```



[OpenClipart-Vectors](#) (Pixabay License)

Esta forma de registro, no fue estandarizada por el W3C, pero debido a que fue ampliamente utilizada por Netscape y Microsoft, todavía es válida hoy en día. La ventaja de este modelo es que podremos asignar un evento a un objeto desde JavaScript, con lo que ya estamos separando el código de la estructura. Fíjate que aquí los nombres de los eventos si que van siempre en minúsculas.

Para eliminar un gestor de eventos de un elemento u objeto, le asignaremos *null*:

```
elemento.onclick = null;
```

Otra gran ventaja es que, como el gestor de eventos es una función, podremos realizar una llamada directa a ese gestor, con lo que estamos disparando el evento de forma manual. Por ejemplo:

```
elemento.onclick(); // Al hacer ésto estamos disparando el evento click de forma manual y
```

Sin paréntesis

Fíjate que en el registro del evento no usamos paréntesis (). El método *onclick* espera que se le asigne una función completa. Si haces: `element.onclick = hacerAlgo();` la función será ejecutada y el resultado que devuelve esa función será asignado a *onclick*. Pero ésto no es lo que queremos que haga, queremos que se ejecute la función cuando se dispare el evento.

3.3.- Modelo de registro avanzado de eventos según W3C.

El W3C en la especificación del DOM de nivel 2, pone especial atención en los problemas del modelo tradicional de registro de eventos. En este caso ofrece una manera sencilla de registrar los eventos que queramos, sobre un objeto determinado.

La clave para poder hacer todo eso está en el método `addEventListener()`.

Este método tiene tres **argumentos**: el **tipo de evento**, la **función a ejecutar** y un **valor** booleano (`true` o `false`), que se utiliza para indicar cuándo se debe capturar el evento: en la fase de captura (`true`) o de burbujeo (`false`).



[OpenClipart-Vectors](#) (Pixabay License)

```
elemento.addEventListener('evento', función, false|true);
```

Por ejemplo para registrar la función `alertar()` de los ejemplos anteriores, haríamos:

```
document.getElementById("miEnlace").addEventListener('click', alertar, false);
const alertar = () => {
    alert(`Te conectaremos con la página: ${this.href}`);
}
```

La ventaja de este método, es que podemos añadir tantos eventos como queramos. Por ejemplo:

```
document.getElementById("miEnlace").addEventListener('click', alertar, false);
document.getElementById("miEnlace").addEventListener('click', avisar, false);
document.getElementById("miEnlace").addEventListener('click', chequear, false);
```

Por lo tanto, cuando hagamos click en `miEnlace` se disparará la llamada a las tres funciones. Por cierto, el W3C no indica el orden de disparo, por lo que no sabemos cual de las tres funciones se ejecutará primero. **Fíjate también, que el nombre de los eventos al usar `addEventListener` no lleva 'on' al comienzo.**

También se pueden usar funciones anónimas (sin nombre de función), haciendo uso de la función flecha como ya venimos haciendo:

```
element.addEventListener('click', () => {
    this.style.backgroundColor = '#cc0000';
}, false);
```

Uso de la palabra reservada `this`

La palabra reservada `this`, tiene exactamente la misma funcionalidad que hemos visto en el modelo tradicional.

¿Qué eventos han sido registrados?

Uno de los problemas de la implementación del modelo de registro del W3C, es que no podemos saber con antelación, los eventos que hemos registrado a un elemento.

En el modelo tradicional si hacemos: `alert(elemento.onclick)`, nos devuelve *undefined*, si no hay funciones registradas para ese evento, o bien el nombre de la función que hemos registrado para ese evento. Pero en este modelo no podemos hacer eso.

El **W3C** en el reciente **nivel 3 del DOM**, introdujo un método llamado `eventListenerList`, que almacena una lista de las funciones que han sido registradas a un elemento.

Para eliminar un evento de un elemento, usaremos el método `removeEventListener()`:

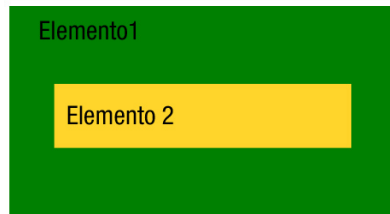
```
elemento.removeEventListener('evento', función, false|true);
```

Para cancelar un evento, este modelo nos proporciona el método `preventDefault()`.

3.4.- Orden de disparo de los eventos.

Imagina que tenemos un elemento contenido dentro de otro elemento, y que tenemos programado el mismo tipo de evento para los dos (por ejemplo el evento `click`). ¿Cuál de ellos se disparará primero? Sorprendentemente, esto va a depender del tipo de navegador que tengamos.

El problema es muy simple. Imagina que tenemos el siguiente gráfico:



y ambos tienen programado el evento de `click`. Si el usuario hace click en el `elemento2`, provocará un click en ambos: `elemento1` y `elemento2`. ¿Pero cuál de ellos se disparará primero?, ¿cuál es el orden de los eventos?

Modelo W3C

W3C decidió que, cuando se produce un evento en su modelo de eventos, primero se producirá la fase de captura hasta llegar al elemento de destino, y luego se producirá la fase de burbujeo hacia arriba. Este modelo es el estándar, que todos los navegadores deberían seguir para ser compatibles entre sí.

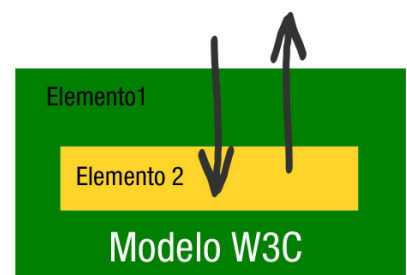
Tú podrás decidir cuando quieres que se registre el evento: en la fase de captura o en la fase de burbujeo. El tercer parámetro de `addEventListener` te permitirá indicar si lo haces en la **fase de captura** (`true`), o en la **fase de burbujeo** (`false`).

Por ejemplo:

```
elemento1.addEventListener('click', hacerAlgo1, true);
elemento2.addEventListener('click', hacerAlgo2, false);
```

Si el usuario hace click en el `elemento2` ocurrirá lo siguiente:

- 1.- El evento de click comenzará en la fase de captura. El evento comprueba si hay algún ancestro del `elemento2` que tenga un evento de `onclick` para la fase de captura (`true`).
- 2.- El evento encuentra un `elemento1.hacerAlgo1()` que ejecutará primero, pues está programado a `true`.
- 3.- El evento viajará hacia el destino, pero no encontrará más eventos para la fase de captura. Entonces el evento pasa a la fase de burbujeo, y ejecuta `hacerAlgo2()`, el cuál hemos registrado para la fase de burbujeo (`false`).
- 4.- El evento viaja hacia arriba de nuevo y chequea si algún ancestro tiene programado un evento para la fase de burbujeo. Éste no será el caso, por lo que no hará nada más.



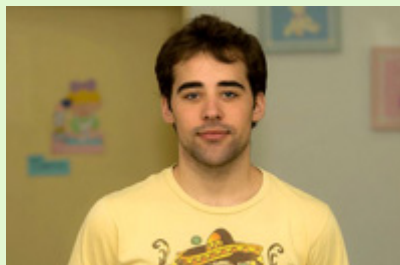
Para **detener la propagación del evento** en la fase de burbujeo, disponemos del método `stopPropagation()`. En la fase de captura es imposible detener la propagación.

4.- Envío y validación de formularios.



Caso práctico

La validación de un formulario, es una de las típicas tareas que tendrá que desarrollar un programador web. Esta tarea de validación es la que más veces tendrá que realizar **Antonio** en la modernización del portal web, por lo que quiere prestar mucha atención al ejemplo de validación de un formulario con JavaScript, comentado por **Juan**, en el que explica todas las partes del código de validación.



En ese ejemplo, se utilizan muchas de las referencias que ha visto en esta unidad, y puesto que es un ejemplo básico, es una buena base para una posible ampliación con los requerimientos que se le piden en el proyecto de actualización del portal.

La validación de un formulario es un proceso que consiste en chequear un formulario y comprobar que todos sus datos han sido introducidos correctamente. Por ejemplo, si tu formulario contiene un campo de texto en el que hay que escribir un e-mail, sería interesante comprobar si ese e-mail está escrito correctamente, antes de pasar al siguiente campo.

Hay dos métodos principales de validación de formularios: en el **lado del servidor** (usando scripts CGI, PHP, ASP, etc.) y en el **lado del cliente** (generalmente usando JavaScript).

La validación en el lado del servidor es más segura, pero a veces también es más complicada de programar, mientras que la validación en el lado del cliente es más fácil y más rápida de hacer (el navegador no tiene que conectarse al servidor para validar el formulario, por lo que el usuario recibirá información al instante sobre posibles errores o fallos encontrados).

La idea general que se persigue al validar un formulario, es que cuando se envíen los datos al servidor, éstos vayan correctamente validados, sin ningún campo con valores incorrectos.

A la hora de programar la validación, podremos hacerlo a medida que vamos metiendo datos en el formulario, por ejemplo campo a campo, o cuando se pulse el botón de envío del formulario.

JavaScript añade a tus formularios dos características muy interesantes:

- ✓ JavaScript te permite examinar y validar las entradas de usuario directamente, en el lado del cliente.
- ✓ JavaScript te permite dar mensajes instantáneos, con información de la entrada del usuario.

La validación de datos del usuario en la entrada, generalmente suele fallar en alguna de las 3 siguientes categorías:

- ✓ **Existencia:** comprueba cuando existe o no un valor.
- ✓ **Numérica:** que la información contenga solamente valores numéricos.
- ✓ **Patrones:** comprueba que los datos sigan un determinado patrón, como el formato de un e-mail, una fecha, un número de teléfono, un número de la seguridad social, etc.

JavaScript también se puede utilizar para modificar los elementos de un formulario, basándose en los datos introducidos por el usuario: tal como cubrir un campo de selección con una lista de nombres de ciudades, cuando una determinada provincia está seleccionada, etc.

Una parte muy importante que no debes olvidar al usar JavaScript con formularios, es la posibilidad de que el usuario desactive JavaScript en su navegador, por lo que JavaScript no debería ser una dependencia en la acción de envío de datos desde un formulario.

"Acuérdate de que JavaScript está para mejorar, no para reemplazar".

La validación de un formulario en el lado del cliente puede ahorrar algunas idas y vueltas a la hora de enviar los datos, pero aún así, tendrás que realizar la validación de datos en el servidor, puesto que es allí realmente donde se van a almacenar esos datos y el origen de los mismos puede venir por cauces que no hemos programado.



Citas para pensar

"Cada hombre puede mejorar su vida mejorando su actitud."

TASSINARI, Héctor

4.1.- Ejemplo sencillo de validación de un formulario.

Como te comenté anteriormente el objeto de validar un formulario es comprobar, que antes del envío del mismo, todos los campos poseen valores correctos, evitando así que el usuario tenga que volver de nuevo al formulario, si es que no se pudo validar correctamente en el servidor. Eso no evita que también tengamos que hacer validación en el servidor, ya que recuerda, que el usuario podrá desactivar JavaScript en su navegador, con lo que la validación que hemos programado en JavaScript no funcionaría.

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>DWEC05 - Validación de un Formulario</title>
    <script type="text/javascript">
      const validar = () => {
        let edad = document.getElementById('edad').value;
        if ( edad >= 18 && confirm("¿Deseas enviar el formulario?"))
          return true;
        else
        {
          alert('Lo siento, pero no eres mayor de edad.')
          this.event.preventDefault();
          return false;
        }
      }
    </script>
    <style type="text/css">
      label{
        width: 150px;
        float:left;
        margin-bottom:5px;
      }
      input, select {
        width:150px;
        float:left;
        margin-bottom:5px;
      }

      fieldset{
        background:#CCFF99;
        width:350px;
      }

      .error{
        border: solid 2px #FF0000;
      }
    </style>
  </head>
  <body>
    <fieldset>
```

```

<legend>DWEC05 - Validación de un Campo </legend>
<form name="formulario" id="formulario" action="http://www.google.es" method="get"
  <label for="edad">Edad:</label>
  <input name="edad" type="text" id="edad" maxlength="3" />
  <input type="reset" name="limpiar" id="limpiar" value="Limpiar" />
  <input type="submit" name="enviar" id="enviar" value="Enviar" onClick="valida
</form>
</fieldset>
</body>
</html>

```



Reflexiona

Te recomiendo que también veas el vídeo en el que se realiza la validación de un formulario, siguiendo el modelo de registro de eventos en línea. Aunque este modelo se utiliza cada vez menos, todavía te encontrarás muchísimos ejemplos que hacen uso del mismo.



Autoevaluación

¿Cuáles de los siguientes métodos son, los que se utilizan en el modelo de registro de eventos de la W3C?

☐ `addEventListener()`.

☐ `onClick()`.

☐ `removeEventListener()`.

☐ `attachEvent()`.

☐ `detachEvent()`.



Solución

1. Correcto
2. Incorrecto
3. Correcto
4. Incorrecto
5. Incorrecto

5.- Expresiones regulares y objetos RegExp.



Caso práctico

Antonio ha comenzado a trabajar en un pequeño formulario del proyecto, y se da cuenta de que en uno de los campos tiene que solicitar al cliente que introduzca los datos en un formato determinado: los datos irán en grupos de caracteres separados por guiones y algunos datos tendrán que llevar paréntesis.



Antonio se pone en contacto con **Juan**, y le pregunta cómo puede hacer para validar que los datos sigan un formato o estructura en un campo de texto. **Juan** le indica que hay dos formas principales: una lenta y una rápida.

La forma lenta, consiste en trabajar con esa cadena de texto, y con los métodos del objeto `String` comprobar que se cumple el formato solicitado. Éste método implicará bastante lógica de programación por lo que dependiendo de la complejidad del formato a chequear habrá que evaluar varias condiciones.

La forma rápida, y recomendada por **Juan**, es el uso de expresiones regulares. De esta forma se podría realizar la validación solicitada, por muy complicada que sea, en dos o tres líneas prácticamente, pero eso sí, tendrá que aprender como crear una expresión regular y los caracteres utilizados en expresiones regulares. Pero el esfuerzo merecerá la pena de sobra, ya que el tiempo que le llevará a **Antonio** validar un campo empleando el método más lento, será el tiempo que necesitará para aprender expresiones regulares.

Las expresiones regulares son patrones de búsqueda, que se pueden utilizar para encontrar texto que coincida con el patrón especificado.

Ejemplo de búsqueda de una cadena de texto sin usar expresiones regulares:


```
let texto = "La línea de alta velocidad llegará pronto a toda España,";
let subcadena = "velocidad";
let indice = texto.indexOf(subcadena);    // devuelve 17, índice de donde se encuentra la su
if (indice !== -1)
    // correcto, se ha encontrado la subcadena
    ....
```

Este código funciona porque estamos buscando una subcadena de texto exacta. ¿Pero qué pasaría si hiciéramos una búsqueda más general? Por ejemplo si quisiéramos buscar la

cadena "car" en textos como "cartón", "bicarbonato", "practicar", ...?

Cuando estamos buscando cadenas que cumplen un patrón en lugar de una cadena exacta, necesitaremos usar expresiones regulares. Podrías intentar hacerlo con funciones de `String`, pero al final, es mucho más sencillo hacerlo con expresiones regulares, aunque la sintaxis de las mismas es un poco extraña y no necesariamente muy amigable.

En JavaScript las expresiones regulares se gestionan a través del objeto `RegExp`.

Para crear un  literal del tipo `RegExp`, tendrás que usar la siguiente sintaxis:

```
let expresion = /expresión regular/;
```

La expresión regular está contenida entre la barras `/`, y fíjate que no lleva comillas. Las comillas sólo se pondrán en la expresión regular, cuando formen parte del patrón en sí mismo.

Las expresiones regulares están hechas de caracteres, solos o en combinación con caracteres especiales, que se proporcionarán para búsquedas más complejas. Por ejemplo, lo siguiente es una expresión regular que realiza una búsqueda que contenga las palabras **Aloe Vera**, en ese orden y separadas por uno o más espacios en medio:

```
let expresion = /Aloe\s+Vera/;
```

Los caracteres especiales en este ejemplo son, la barra invertida (`\`), que tiene dos efectos: o bien se utiliza con un carácter regular, para indicar que se trata de un carácter especial, o se usa con un carácter especial, tales como el signo más (`+`), para indicar que el carácter debe ser tratado literalmente. En este caso, la barra invertida se utiliza con `"s"`, que transforma la letra `s` en un carácter especial indicando un espacio en blanco, un tabulador, un salto de línea, etc. El símbolo `+` indica que el carácter anterior puede aparecer una o más veces.

5.1.- Caracteres especiales en expresiones regulares.

Veamos una tabla con algunos de los caracteres más utilizados para la construcción de Expresiones Regulares:



[OpenIcons](#) ([Pixabay License](#))

Caracteres especiales utilizados en Expresiones Regulares

Carácter	Coincidencias	Patrón	Ejemplo de cadena
^	Al inicio de una cadena	/^Esto/	Coincidencia en "Esto es..."
\$	Al final de la cadena	/final\$/	Coincidencia en "Esto es el final".
*	Coincide 0 o más veces	/se*/	Que la "e" aparezca 0 o más veces: "seeee" y también "se".
?	Coincide 0 o 1 vez	/ap?	Que la p aparezca 0 o 1 vez: "apple" y "and".
+	Coincide 1 o más veces	/ap+/	Que la "p" aparezca 1 o más veces: "apple" pero no "and".
{n}	Coincide exactamente n veces	/ap{2}/	Que la "p" aparezca exactamente 2 veces: "apple" pero no "apabullante".
{n,}	Coincide n o más veces	/ap{2,}/	Que la "p" aparezca 2 o más veces: "apple" y "appple" pero no en "apabullante".
{n,m}	Coincide al menos n, y máximo m veces	/ap{2,4}/	Que la "p" aparezca al menos 2 veces y como máximo 4 veces: "apppppple" (encontrará 4 "p").
.	Cualquier carácter excepto nueva línea	/a.e/	Que aparezca cualquier carácter, excepto nueva línea entre la a y la e: "ape" y "axe".

Carácter	Coincidencias	Patrón	Ejemplo de cadena
[...]	Cualquier carácter entre corchetes	/a[px]e/	Que aparezca alguno de los caracteres "p" o "x" entre la a y la e: "ape", "axe", pero no "ale".
[^...]	Cualquier carácter excepto los que están entre corchetes	/a[^px]/	Que aparezca cualquier carácter excepto la "p" o la "x" después de la letra a: "ale", pero no "axe" o "ape".
\b	Coincide con el inicio de una palabra	^bno/	Que "no" esté al comienzo de una palabra: "novedad".
\B	Coincide al final de una palabra	^Bno/	Que "no" esté al final de una palabra: "este invierno" ("no" de "invierno").
\d	Dígitos del 0 al 9	^d{3}/	Que aparezcan exactamente 3 dígitos: "Ahora en 456".
\D	Cualquier carácter que no sea un dígito	^D{2,4}/	Que aparezcan mínimo 2 y máximo 4 caracteres que no sean dígitos: encontrará la cadena "Ahor" en "Ahora en 456".
\w	Coincide con caracteres del tipo (letras, dígitos, subrayados)	^w/	Que aparezca un carácter (letra, dígito o subrayado): "J" en "JavaScript".
\W	Coincide con caracteres que no sean (letras, dígitos, subrayados)	^W/	Que aparezca un carácter (que no sea letra, dígito o subrayado): "%" en "100%".
\n	Coincide con una nueva línea		Recuerda los caracteres.
\s	Coincide con un espacio en blanco		
\S	Coincide con un carácter que no es un espacio en blanco		
\t	Un tabulador		
(x)	Capturando paréntesis		
\r	Un retorno de carro		Hola mundo mundial.
?=n	Cualquier cadena que está seguida por la cadena n indicada después del igual.		

5.2.- El objeto RegExp.

El objeto `RegExp` es tanto un literal como un objeto de JavaScript, por lo que también se podrá crear usando un constructor:

```
let expresionRegular = new RegExp("Texto Expresión Regular");
```

¿Cuándo usar el literal o el objeto?

La expresión `RegExp` literal es compilada cuando se ejecuta el script, por lo tanto se recomienda usar el literal cuando sabemos que la expresión no cambiará. Una versión compilada es mucho más eficiente.

Usaremos el objeto, cuando sabemos que la expresión regular va a cambiar, o cuando vamos a proporcionarla en tiempo de ejecución.

Al igual que otros objetos en JavaScript, el objeto `RegExp` también tiene sus propiedades y métodos:

Propiedades del objeto RegExp

Propiedad	Descripción
<code>global</code>	Especifica que sea utilizado el modificador "g".
<code>ignoreCase</code>	Especifica que sea utilizado el modificador "i".
<code>lastIndex</code>	El índice donde comenzar la siguiente búsqueda.
<code>multiline</code>	Especifica si el modificador "m" es utilizado.
<code>source</code>	El texto de la expresión regular <code>RegExp</code> .

Métodos del objeto RegExp

Método	Descripción
<code>compile()</code>	Compila una expresión regular.
<code>exec()</code>	Busca la coincidencia en una cadena. Devolverá la primera coincidencia.
<code>test()</code>	Busca la coincidencia en una cadena. Devolverá true o false.



[Clicer-Free-Vector-Images](#)
(Pixabay License)

5.3.- Ejemplos de uso de expresiones regulares.

Aquí te pongo algunos ejemplos de uso de expresiones regulares:

Validar un número de teléfono

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html" />
    <title>Ejemplo de RegExp</title>
    <script type="text/javascript">
      const iniciar = () => {
        document.getElementById("comprobar").onclick
      }
      const comprobarTelefono = () => {
        let telefono = document.getElementById("telefono")
        let patron = /^d{9}$/;
        if (telefono.match(patron))
          alert('Teléfono Correcto!!!!');
        else
          alert('Teléfono INCORRECTO!!!!');
      }
      window.onload = iniciar;
    </script>
  </head>
  <body>
    <form name="formulario">
      <label for="telefono">Telefono:</label>
      <input type="text" name="telefono" id="telefono" />
      <input type="button" name="comprobar" id="comprobar" value="Comprobar" />
    </form>
  </body>
</html>
```



[OpenClipart-Vectors](#) (Pixabay License)

Validación de un número de DNI

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>Ejemplo de RegExp</title>
    <script type="text/javascript">
      const iniciar = () => {
        document.getElementById("comprobar").onclick = comprobarDni;
      }
      const comprobarDni = () => {
```

```
        let dni = document.getElementById("dni").value;
        let patron = /^\\d{8}[A-Z]$/;
        if (dni.match(patron))
            alert('DNI Correcto!!!!');
        else
            alert('DNI INCORRECTO!!!!');
    }
    window.onload = iniciar;
</script>
</head>
<body>
    <form name="formulario">
        <label for="dni">DNI:</label>
        <input type="text" name="dni" id="dni" />
        <input type="button" name="comprobar" id="comprobar" value="Comprobar Formato" />
    </form>
</body>
</html>
```



Debes conocer

En el siguiente enlace podrás validar las expresiones regulares que introduzcas y además te irá explicando cómo analiza dicha expresión regular. Te aconsejo su utilización ya que te va a permitir practicar con ellas y entenderlas mejor.

[Expresiones regulares](#)