Gestión de formularios en JavaScript

1) El objeto form.
    1) Formas de selección del objeto Form.
    2) Acceso a propiedades y métodos del formulario.
2) Objetos relacionados con formularios.
    1) Objeto input tipo texto.
    2) Objeto input tipo checkbox.
    3) Objeto input tipo radiobutton.
    4) Objeto input tipo select.
3) Envío y validación de formularios.
4) Expresiones regulares y objetos RegExp.
5) Las cookies.

1. Objeto form

Los formularios, son el principal medio de introducción de datos en una aplicación web.

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**.

JavaScript añade a los formularios dos características muy interesantes:

- Examen y validación de las entradas de usuario directamente, en el lado del cliente.
- Mostrar mensajes instantáneos, con información de la entrada del usuario.

Ver ejemploFormulario.html

1.1. Formas de selección del objeto Form (I)

Si tenemos el siguiente código HTML:

```
<div id="menulateral">
<form id="id_contactar" name="contactar" action="buscar.php">…
</form>
</div>
```

Ejemplo para acceso al formulario: ejAccesoForm-Elements-Submit-Reset-Action.html

Método 1. A través del nombre del formulario (tiene que ser único).

```
var formulario=document.contactar;
var formulario=contactar;
```

Método 2. A través del id del formulario (tiene que ser único).

```
var formulario=id_contactar;
```

!!OJO: NO FUNCIONA!! `var formulario=document.id_contactar;`

Método 3. Con el método `getElementById` (tiene que ser único)

```
var formulario=document.getElementById("id_contactar");
```

1.1. Formas de selección del objeto Form (II)

Método 4. A través del nombre del tag del elemento HTML.

```
var formularios = document.getElementsByTagName("form");
```

formularios será un array de formularios, ya que puede haber más de 1 en el documento HTML

```
var primerFormulario = formularios[0]; // primer formulario del documento
```

De forma reducidua:

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

Método 5. A través del id del elemento HTML padre en el que está el formulario, y después obtenemos el array de formularios que hay dentro de este elemento padre.

```
var menu=document.getElementById("menulateral");

var formularios=menu.getElementsByTagName("form");

// formularios contenidos en el menu lateral

var primerFormulario= formularios[0];

// primer formulario en el menú lateral
```

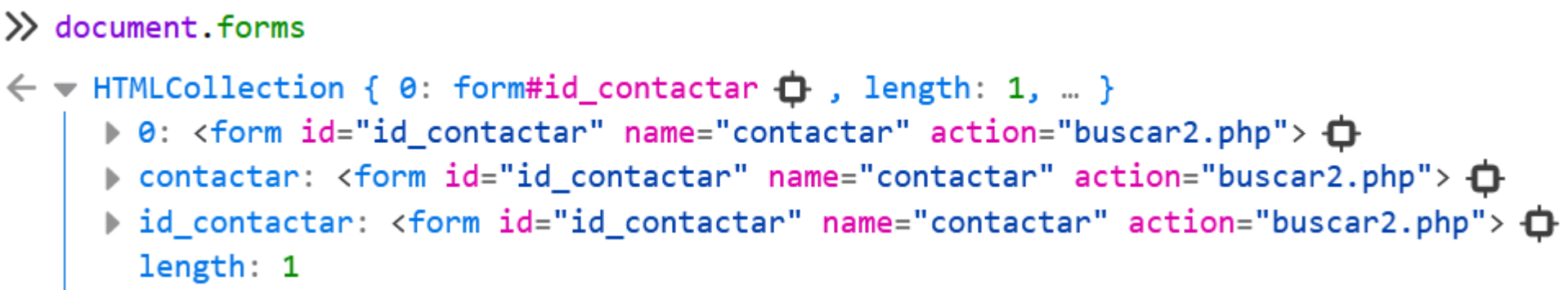
1.1. Formas de selección del objeto Form (III)

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

Método 6. A través del array de formularios del objeto document. Equivalente al método 4.

```
var formularios = document.forms; // Todos los formularios del documento
var miformulario = formularios[0]; //1°formulario del documento
O bien: ------------------------------------
var miformulario = document.forms[0]; //1°formulario del documento
o bien: ------------------------------------
var formularios = document.forms; //Todos los formularios del documento
var miformulario = formularios["contactar"];//form con name "contactar"
var miformulario = formularios["id_contactar"];//form con id "contactar"
```

Observa en consola: Colección `length 1`, indexable por `0`, `contactar` e `id_contactar`



1.2. Acceso a propiedad y métodos de form (I).

Los atributos action, target, method, enctype son propiedades del formulario.

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

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

Propiedades de form:

- **acceptCharset:** ajusta o devuelve el valor del atributo accept-charset del formulario. Especifica la codificación de caracteres utilizada para enviar los datos al servidor. Normalmente UTF-8. No todos los navegadores aceptan todos los tipos de codificaciones.
- **action**: ajusta o devuelve el valor del atributo action. Indica la URL del script que se va a ejecutar (un script de servidor)
- **enctype**: ajusta o devuelve el valor del atributo enctype. Especifica cómo va a codificarse los datos para ser enviados. Solo para el método post. Por defecto: application/x-www-form-urlencoded (todos los datos son codificados)
- **method**: ajusta o devuelve el valor del atributo method: get o post.
- **name:** ajusta o devuelve el valor del atributo name.
- **target**: ajusta o devuelve el valor del atributo target. Indica dónde se va a ver la respuesta que se recibe del script: “_blank: en una nueva ventana o pestaña”, “_self: en la ventana actual”, “_parent: en el frame padre”, “_top: en el body de la ventana (sustituyendo el actual)”
- **length**: : devuelve el número de elementos contenidos en el formulario. (label no es un elemento, los botones, input, fieldset, etc.. sí).

1.2. Acceso a propiedad y métodos de form (II).

Propiedades de form:

- **`elements[]` :** array con los elementos del formulario.

Métodos de form:

- **`reset()` :** resetea un formulario.
- **`submit()`:** envía un formulario.

Ejemplo para observar `elements[]`, de ejecución de `submit()`, `reset()`, y envío de datos del formulario a través de `action`: ejAccesoForm-Elements-Submit-Reset-Action.html

2. Objetos relacionados con formularios (I).

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

Para acceder al objeto elemento de un formulario podemos utilizar estas referencias:

`id-del-elemento` (Cuidado no se puede con name-del-elemento solo)

`name-del-formulario.name-del-elemento` (dentro del objeto formulario sí)

`name-del-formulario.id-del-elemento`

`id-del-formulario.name-del-elemento` (dentro del objeto formulario sí)

`id-del-formulario.id-del-elemento`

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

Se aconseja poner un id único a cada elemento y que coincida con el nombre (name) del elemento.

2. Objetos relacionados con formularios (II).

```
<form id=”id_form” name=”n_form”>

<input type="radio" id="c_madrid" name="ciudad"
value="Madrid">
```

- **c_madrid** -> id-del-elemento
- **n_form.ciudad** → name-del-formulario.name-del-elemento
- **n_form.c_madrid** → name-del-formulario.id-del-elemento
- **id_form.ciudad** → id-del-formulario.name-del-elemento
- **id_form.c_madrid** → id-del-formulario.id-del-elemento
- **document.getElementById("c_madrid ")**→ document.getElementById("id-del-elemento")

2. Objetos relacionados con formularios (III).

Tipos de elementos de un formulario:

- **label → no es un tipo de elemento del formulario.**
- **text** → caja de texto.
- **radio** → Permite seleccionar una opción de un número limitado de opciones.
- **checkbox** → Permite seleccionar 0 o más opciones de un número limitado de opciones.
- **submit** → botón para ejecutar lo indicado en el atributo action de <form>
- **select** → crea un elemento drop-down con varias opciones.
- **textarea** → como text pero con n filas

```
<textarea name="message" rows="10" cols="30"></textarea>
```

- **button** → botón para ejecutar lo indicado según sus atributos.
- **fieldset** → para agrupar varios elementos.
- **list, datalist** → como text pero que indica al usuario una lista de opciones para que elija una o ponga otra libremente.
- **output** → es un elemento que sacará el resultado de un cálculo.
  https://www.w3schools.com/tags/tryit.asp?filename=tryhtml5_output
- **passwd** → caja de texto para introducir clave.

2. Objetos relacionados con formularios (IV).

Más tipos de elementos de un formulario:

- **date** → es un elemento que sacará una ayuda para introducir una fecha.
- **datetime-local** → es un elemento que sacará una ayuda para introducir la fecha y hora local (no contempla time zone, huso horario)
- **email** →elemento para introducir un email. El navegador testea que cumple el formato
- **file** →elemento por el que el navegador saca un navegador de archivos para elegir un fichero.
- **month** → elemento por el que el navegador saca una ayuda para seleccionar un año y un mes
- **time** → un elemento que hace que el navegador saque una ayuda para seleccionar una hora sin huso horario.
- **number** → sale una caja de texto con flechitas para poner un número, se puede indicar máximo y mínimo. En este caso aparecerá en esta caja de texto solo el rango válido.
- **Range** → saca un campo gráfico para elegir con un selector el valor numérico deseado.
- **hidden** → un elemento que estará oculto en el formulario pero cuyo dato aparece en el código y que va a ser enviado al servidor

Las propiedades y métodos de cada uno de estos elementos se pueden consultar en https://www.w3schools.com/jsref/ (menú de la izquierda – HTML Objects)

2.1 Objetos input de tipo texto (I)

`<label for="fname">First name:</label>` `(no es elemento del form)`

`<input type="text" id="fname" name="fname" value="John">`

Hay varios elementos input de tipo texto: **text, hidden, textarea, passwd, email, ….**

Cuando se envían los datos de un formulario a un programa en el lado del servidor, se envían los atributos **name ,** junto con los valores (contenido del atributo **value**) **value es una cadena.**

Métodos del objeto INPUT de tipo texto

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

Este método sirve para que aparezca el contenido seleccionado:
https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_text_select

2.1 Objetos input de tipo texto (II)

Propiedades del objeto INPUT de tipo texto

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

`size` (`size` de HTML) es la cantidad de caracteres que se pueden ver a la vez en la caja de texto. **`maxLength`** (`maxlength` de HTML) es el n.º máximo de caracteres que el usuario puede introducir (https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_text_size2) Ej: CAMBIAR maxLength

`defaultValue` corresponde a la propiedad `value` de HTML. La propiedad `value` cambia al variar el usuario la caja de texto, pero `defaultValue` no. De esta forma si se puede saber si el usuario ha cambiado el valor inicial de la caja de texto comparando `defaultValue` con `value`. El `value` del HTML se puede establecer o cambiar el establecido con `defaultValue`, y aparecerá el valor dado en la caja de texto (solo si el usuario no ha cambiado previamente el contenido de la caja de texto) (https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_text_defaultvalue2)

Ejemplo de input-text-value-defaultValue

2.2 Objeto input tipo checkbox (I)

```
<form id=”formulario”>
   <input type="checkbox" id="c_madrid" name="c_madrid" value="Madrid">
   <label for="c_madrid">Madrid</label><br>
   <input type="checkbox" id="c_sevilla" name="c_sevilla" value="Sevilla"
checked>
   <label for="c_sevilla">Sevilla</label><br>
   <input type="checkbox" id="c_cuenca" name="c_cuenca" value="Cuenca">
   <label for="c_cuenca">Cuenca</label><br>
</form>
```

El texto que hay en **value 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.**

Si la opción no está marcada no se enviará con el formulario su value.

2.2 Objeto input tipo checkbox (II)

La propiedad **`checked`** devuelve true si el checkbox está marcado, y false si no.

La propiedad **`defaultChecked`** guardará true si se ha declarado en HTML `checked` en el `checkbox`.

Propiedades del objeto INPUT de tipo checkbox

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

Ejercicios 1 y 2: observa en el ejercicio 1 qué datos se envían al servidor.

2.3 Objeto input radio button(I)

```
<form id=”formulario”>
        <input type="radio" id="c_madrid" name="ciudad" value="Madrid">
        <label for="c_madrid">Madrid</label><br>
        <input type="radio" id="c_sevilla" name="ciudad" value="Sevilla">
        <label for="c_sevilla">Sevilla</label><br>
        <input type="radio" id="c_cuenca" name="ciudad" value="Cuenca">
        <label for="c_cuenca">Cuenca</label><br>
</form>
```

Todos **los input de tipo radio** que están **relacionados tienen el mismo name**. Al hacer esto el navegador crea un **array con la lista de elementos que tienen el mismo name**. **Referenciamos a este array por dicho name**. Por ejemplo:

`formulario.ciudad` → es un array de 3 elementos

`formulario.ciudad.length` → es 3.

`formulario.ciudad.value` → tendrá el value del radio button checkeado (Madrid si se ha seleccionado el radio button de Madrid, ... o undefined si no se ha seleccionado ninguno)

`formulario.c_sevilla.checked` o `formulario.ciudad[1].checked` → true si seleccionado.

`formulario.c_sevilla.defaultChecked` o `formulario.ciudad[1].defaultChecked` → será false porque no tiene el atributo HTML checked

Para cada elemento del radio sí existe `checked` y `defaultChecked`, pero no para la referencia al conjunto del radio button

`formulario.ciudad.checked` y `formulario.ciudad.defaultChecked` → no existe

Ejercicio 3, 4, 5, 6, 7

2.4 Objeto input select(I)

```
<form action="/action_page.php">
  <label for="cars">Choose a car:</label>
  <select id="cars" name="cars">
    <option value="volvo">Volvo</option>
    <option value="saab" selected>Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
  </select>
</form>
```

- **El objeto select tiene una propiedad que es options[] que contiene todos sus elementos option.**

`cars.options` → **colección que** guarda los datos de todas las opciones

`cars.selectedIndex` → guarda el número de la opción seleccionada. No existe `checked, defaultChecked, defaultSelected, ….`

`cars.value` → guarda el value de la opción seleccionada.

`cars.options[1].selected:` → vale `true/false` si está seleccionada o no la opción 1.

- La propiedad **`text`** guarda el texto que aparece en HTML (Volvo, Saab, Fial, Audi):

`cars.options[cars.selectedIndex].text` → guarda texto de la opc. seleccionada

- La propiedad **value** es lo que valdrá name si ha sido seleccionada dicha opción (volvo,saab,fiat, audi)

`cars.options[cars.selectedIndex].value` → guarda `value` de opción seleccionada. Es equivalente a **`cars.value.`**

Ejercicio 8 A

2.4 Objeto input select(II)

```
<form action="/action_page.php">
  <label for="cars">Choose a car:</label>
  <select id="cars" name="cars" multiple>
    <option value="volvo">Volvo</option>
    <option value="saab" selected>Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
  </select>
</form>
```

- **Si el select tiene la opción `multiple` activada significa que el usuario puede seleccionar más de una opción (pulsando `Ctrl`)**

`cars.selectedOptions` → **colección que** guarda los datos de todas las opciones seleccionadas

En este caso, la propiedad **`selectedIndex`** no tiene tanto sentido. Guardará el índice de la primera opción seleccionada

Ejercicio 8 B

4. Envío y validación de formularios

La validación de un formulario es un proceso que consiste en chequear un formulario y comprobar que todos sus datos han sido introducidos correctamente.

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.

La validación de un formulario en el lado del cliente puede **ahorrar algunas idas y vueltas al servidor** 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.

Podremos hacer la validación a medida que vamos metiendo datos en el formulario, por ejemplo campo a campo, o cuando se pulse el botón de envío del formulario.

La ventaja de **JavaScript te permite dar mensajes instantáneos, con información de la entrada del usuario.**

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.

4. Envío y validación de formularios

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.
- **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.

4. Envío y validación de formularios(V)

Estructura recomendada para la **validación global de todo el formulario:**

```
window.onload=iniciar;

    //cuando el documento esté cargado nos aseguramos que las asignaciones de eventos no fallarán ya que todos los objetos
    están disponibles.

function iniciar()

   //Al hacer click en el botón enviar tendrá que llamar a la la función validar que se encargará de validar el formulario.
    Evento click lo programamos en la fase de burbujeo (false)

   document.getElementById("enviar").addEventListener('click',validar,false);

function validar()

  // Si el usuario confirma el envío del formulario se ejecutará la acción por defecto del botón de envío (el submit)

Evento por defecto asociado al botón de "enviar" (type=submit) que en este caso lo que hace por defecto es enviar un form

    If (validarcion1 && validacion2 && ….validaciónN && confirm("¿Deseas enviar el formulario?"))

        return true;

    else

       //Cancelamos el evento de envío por defecto asignado al boton de submit enviar.

        event.preventDefault();

        return false;// Salimos de la función devolviendo false.

//-----------------------------------------------------------//

function validacion1()

function validacion2()

...

function validacionN()
```

5. Expresiones regulares y objetos RegExp

Las expresiones regulares son patrones para ser buscados en una cadena de texto. simplificándose enormemente la tarea de búsqueda de patrones y procesamiento de los mismos (como sustitución) por su potencial de programación.

Las expresiones regulares en javascript son **objetos**.

Ejemplo: patrón de las palabras Aloe Vera, en ese orden y separadas por uno o más espacios en medio:

```
var patron=/Aloe\s+Vera/;
```

Una expresión regular es un patrón que lo pueden cumplir muchas cadenas. En este ejemplo en las cadenas “Aloe Vera”, “Aloe Vera”, “Aloe Vera”, etc. se encuentra este patrón.

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

Hay dos formas de definir un expresión regular:

- Mediante una **expresión literal**:

```
var  expresion = /patron/modificadores;
//No llevan dobles comillas
```

- Mediante el **objeto RegExp**:

```
var expresion= new RegExp(“/patron/”, modificadores)
```

5.1. Modificadores

Las expresiones regulares se componen de dos partes:

- **Patrón de la expresión**
- **Modificadores.**

Los modificadores son opcionales y sirven para definir el comportamiento de la expresión regular. Son los siguientes (y no están activados por defecto):

- **`g`(global)**: El patrón se aplicará a toda la cadena en lugar de detenerse al encontrar la primera correspondencia correcta. Es decir, se devolverán todas las ocurrencias encontradas. Tiene sentido con métodos `match()` y `replace()` de `String`, no con métodos `exec()` y `test()` de `RegExp`, ni con el método `search()` de `String`.

Ejemplo: https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_regexp_g_string (probar con g y sin g)

- **`i`(insensible)** a mayúsculas y minúsculas.
- **`m`(multilinea)**: varía el comportamiento de los caracteres especiales `^` y `$` de las expresiones regulares. Cuando está activado el modificador `m` si la cadena en la que se busca un patrón tiene varias líneas (por ejemplo en el caso de `textarea`), la cadena es tratada como varias líneas, con tantos comienzos y finales de línea como líneas haya. Por lo que si `m` está activado, al utilizar el carácter `^cad` se va a buscar `cad` en cada comienzo de línea, y lo mismo con el carácter `$`. Por defecto, aunque `cad` tuviera varias líneas sería tratado como una sola línea, con un solo comienzo de línea y un solo fin de línea.

Ejemplo: crear un textarea y buscar el patrón `^hola`

5.2. Caracteres especiales en el patrón de las expresiones regulares. Clases de caracteres

.	Cualquier carácter excepto nueva línea	/a.e/	Que aparezca cualquier carácter, excepto nueva línea entre la a y la e: “ape” y “axe”.
\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		
\s	Coincide con un espacio en blanco		
\S	Coincide con un carácter que no es un espacio en blanco		
\t	Un tabulador		
\r	Un retorno de carro		
\d	Dígitos del 0 al 9		
\D	Cualquier carácter que no sea un dígito		

\w: cualquier carácter alfanumérico del alfabeto latino básico (no incluida ñ, tildes, etc) mayúsculas o minúsculas, incluido guión bajo

\s: espacio en blanco, tabulador, retorno de línea, avance de página, , etc

Los metacaracteres son caracteres que tienen un significado especial, a diferencia de los caracteres regulares que no tienen ningún significado especial, sino únicamente su valor literal.

<u>En los patrones de expresiones regulares tenemos los siguientes metacaracteres</u>:

```
. * + ? ^ $ { } ( ) | [ ] \ < ! = - \b \B \s \S \w \W \d \D \t \r \n \v \f
```

**** Para escapar un carácter especial y, por tanto, no sea interpretado, y se trate como un literal. Por ejemplo: `/\./` : se busca el carácter **.** y no cualquier carácter

5.2. Caracteres especiales en el patrón de las expresiones regulares
Cuantificadores

Carácter	Coincidencias	Patrón	Ejemplo de cadena
*	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").

Ejemplo ? : /ap?/ → la coincidencia del patrón en **`apple`** es **`ap`**, y en **`and`** es **`a`**.

Ejemplo ? : /ap?l/ → **`apple`** no tiene ninguna coincidencia con el patrón. **`aple`** sí, y **`ale`** sí.

Caracteres especiales en el patrón de las expresiones regulares. Aserciones

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".

\b: es un límite de palabra (son posiciones en una cadena de texto que indican el inicio o el final de una palabra, y que se encuentran entre un carácter de palabra y un carácter que no es de palabra.):

- `/na\b/` buscará la cadena seguida de un carácter que no es de palabra, es decir, al final de una palabra
- `/\bna/` buscará la cadena precedida de un carácter que no es de palabra, es decir, al comienzo de una palabra

Si queremos buscar palabras del alfabeto inglés:

`/\w+/i` Para `"esto es un 3"` devuelve `esto` `es` `un` `3`

No es necesario `/\b\w+\b/` ya que los caracteres no incluidos en `\w` hacen de límite de palabra, por lo que funciona igual. Para `"esto es un 3"` devuelve `esto` `es` `un` `3`

NOTA: No funcionará indistintamente con palabras de otros alfabetos como el español. Ya que hay que tener en cuenta que un carácter que no es de palabra es cualquier carácter que no sea del alfabeto básico látino, número o guión bajo (`\w`) , por lo que si la palabra empieza por un carácter no latino (vocales con tilde, ñ, guión alto, etc) no funciona. P.ej: buscamos palabras del alfabeto español:

`/\b[\wáéíóúüñ]+\b/i` Para `"esto ñakjlñ"` Devuelve `esto` `ñakjlñ` -> no funciona, ya que lo que hay entre la `ñ` y la `a` es límite de palabra, ya que la `ñ` no es carácter de palabra.

`/[\wáéíóúüñ]+/i` Para `"esto ñakjlñ"` devuelve `esto` `ñakjlñ` -> **Solución que sí funciona**

Palabras españolas de 5 a 8 caracteres: `/[\wáéíóúüñ]{3,5}/i` Para `"es ñacas"` devuelve `ñacas`

Caracteres especiales en el patrón de las expresiones regulares. Aserciones

`\B`: al contrario. Por ejemplo:

- `/na\B/` buscará la cadena pero no seguida de un límite de palabra. O lo que es lo mismo, no buscará la cadena al final de una palabra, la cadena podrá estar al comienzo o en medio de una palabra pero no al final.
- `/\Bna/` buscará la cadena pero no precedida de un límite de palabra. O lo que es lo mismo, no buscará la cadena al comienzo de una palabra, la cadena podrá estar al final o en medio pero no al comienzo.

5.2. Caracteres especiales en el patrón de las expresiones regulares. Aserciones

cad1(?=cad2) → que aparezca cad1 seguido de cad2. La ocurrencia sería cad1. Ejemplo: /e(?=\scla)/ (e seguido de uno o más espacios en blanco y después cla en la cadena "Hoy es el primer dia d**e** clase" sí hay coincidencia.

```
var patron=/e(?=\sclase)/;

var str="Final de clase";

demo.innerHTML+=patron.exec(str); /devuelve e


var patron=/e(?=\sclase)/;

var str="Final de hora";

demo.innerHTML+=patron.exec(str); /no devuelve nada. Hay e
pero no seguido de " clase"


var patron=/e\sclase/;

var str="Final de clase";

demo.innerHTML+=patron.exec(str); /devuelve e clase
```

5.2. Caracteres especiales en el patrón de las expresiones regulares. Aserciones

`cad1(?!cad2)` → que aparezca cad1 si no está seguido de cad2. La ocurrencia sería cad1.

`(?<=cad2)cad1` → que aparezca cad1 si está precedido de cad2. La ocurrencia sería cad1. No soportada en todos los navegadores.

`(?<!cad2)cad1` → que aparezca cad1 si no está precedido de cad2. La ocurrencia sería cad1. No soportada en todos los navegadores.

5.2. Caracteres especiales en el patrón de las expresiones regulares. Grupos, rangos y |

`[...]`	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".

`[a-c]` Cualquier carácter del rango, a,b,c

`[A-D]` Cualquier carácter del rango: A,B,C,D

`[4-7]` Cualquier carácter del rango: 4,5,6,7

cad1|cad2→ que aparezca o cad1 o cad2. Ejemplo: /rojo|verde/ en la cadena "Cuadro rojo" si encontrará una coincidencia. En la cadena "Cuadro verde" también. En la cadena "Cuadro amarillo" no.

```
var patron=/rojo|verde|azul/;

var str="Pajaro rojo";  //Cumple con el patrón.
```

`():` para crear subexpresiones y aplicar funciones a ellas. Por ejemplo:

```
var patron=/(rojo|verde|azul)3/ //se busca rojo, verde o azul seguido de 3

var str="Pajaro rojo3";  //Cumple con el patrón.
```

Otro ejemplo:

`/(le)+/g`: buscará `le` 1 o más veces. En la cadena `lelere` encontrará `lele`

Métodos del objeto RegExp

`exec()`	Busca la coincidencia en una cadena. Devolverá la primera coincidencia.
`test()`	Busca la coincidencia en una cadena. Devolverá true o false.

El **método `exec()`** se ejecutará así:

```
var str = "Is this it?";

//expresion→ que comience por is ignorando mayúsculas y minúsculas.
var expresion = /^is/i;
```

Devolverá en un array la 1ª ocurrencia de `expresion` encontrada en `str` aunque haya más de 1. **NO devolverá más ocurrencias aún utilizando modificador g. Si no encuentra ninguna ocurrencia devuelve null.**

```
var result = expresion.exec(str);;  // result será Is
```

El **método `test()`** se ejecutará así:

```
var str = "Is this it?";

//expresion→ que comience por is ignorando mayúsculas y minúsculas.
var expresion = /^is/i;
```

Devuelve `true` si encuentra alguna ocurrencia de `expresion` en `str`, o `false` si no hay ninguna.

```
var result = expresion.test(str);; //devuelve true
```

Ejercicios 9, 10, 11, 12, 13,14

Métodos de String que usan expresiones regulares

`cadena.`**`match`**`(expresionRegular)`**`:`** devuelve un `array` con todas las ocurrencias del patrón encontradas en la cadena, si la expresión regular tiene el modificador `g` activado. Si no tiene activado el modificador `g` devuelve un `array` con solo la primera ocurrencia. Si no encuentra ninguna ocurrencia del patrón en la cadena devuelve `null`.

`cadena.`**`replace`**`(expresionRegular, nuevacadena)`**`:`** reemplaza todas las ocurrencias del patrón encontradas en la cadena por `nuevacadena`, si la expresión regular tiene el modificador `g` activado. Si no tiene activado el modificador `g` reemplaza solo la primera ocurrencia. Si no encuentra ninguna ocurrencia del patrón en la cadena devuelve la cadena sin modificar.

`cadena.`**`search`**`(expresionRegular)`**`:`** devuelve la posición en la que se encuentra la primera ocurrencia del patrón encontrada en la cadena. Si no encuentra ninguna ocurrencia del patrón en la cadena devuelve `-1`. No devuelve más posiciones aunque la expresión regular tenga el modificador `g` activado.

constructor RegExp()

Hasta ahora hemos creado las expresiones regulares literales, pero se puede crear una expresión regular con el constructor `RegExp()`. La expresión regular se compondrá llamando al constructor y pasándole 2 argumentos de tipo `String` (que pueden ser variables):

- Primer argumento: un string que guardará el patrón (sin las `/`)
- Segundo argumento: un string que guardará los modificadores. Este argumento es opcional.

```
var patron="[0-3]";
var modificadores="g";
var expr=new RegExp(patron,modif)    //1° argumento el patron y 2° modificadores
var texto="hola3"
expr.test(texto) // devuelve true porque encuentra un 3 en el texto
```

Cuando creamos una expresión regular con `RegExp` el navegador ha de crear a partir de ella una expresión regular literal, y esto le lleva más tiempo.

Sin embargo utilizar RegExp es necesario cuando la expresión regular no es conocida de antemano, por ejemplo si la facilita el usuario a través de un input, o va variando a lo largo del programa.

Ejemplo `new RegExp()`: crea un fichero HTML con los siguientes elementos:

- **Un textarea libre para que el usuario pueda introducir el texto que desee.**
- **Un input tipo text para introducir la cadena que se desea buscar en el texto anterior.**
- **Un input tipo button "Buscar" que al pulsarlo se muestren todas las cadenas del texto del textarea que cumplen con el patrón ignorando mayúsculas y minúsculas.**

Ejercicios expresiones regulares

Crea un input tipo texto para cada uno los siguientes datos, y un botón que al pulsarlo se valida el contenido de cada uno de los campos y se indique al usuario si es correcto o no:

- Campo nombre: no puede tener números, ni signos de puntuación.
- Campo apellidos: no puede tener números, ni signos de puntuación.
- Campo teléfono internacional con el formato: un código de 1 a 3 dígitos, precedido por el signo "+" o "00" y después 9 dígitos.
- Campo día con valores 1-31
- Campo email: teniendo en cuenta que debe haber una @, y en el dominio al menos ha de haber un ., y terminado en un número de caracteres de 2 a 10. Y teniendo en cuenta los caracteres permitidos en una dirección email.
- Campo para un número binario:
- Campo para un número hexadecimal.
- Campo para un número decimal (positivo o negativo) con 2 decimales.

Ejercicios expresiones regulares.

- **Busca en un texto** todas las cadenas que terminan en `n` o `s`.
- **Busca en un texto** los colores que aparecen pero en singular
- **Busca en un texto** compuesto por datos con esta estructura `"dato1=valor1,dato2=valor2,dato3=valor3,….”`, el valor del dato `"dato2”`
- **Busca en un texto** que es una lista de códigos de grupos `NAAA-X` siendo `N` el curso, `AAA` el código de la etapa educativa, y `X` la letra del grupo:
  - El número de grupos de ESO.
  - El número de grupos de 1º de la ESO.

6. Las cookies

Las HTTP cookies es un mecanismo para guardar datos de forma temporal en el lado del cliente.

Se utilizan en las aplicaciones interactivas y permitan "recordar" las acciones que se realizaron previamente.

Las cookies se guardan en un fichero que está situado en un directorio especial del disco duro. La localización de este fichero depende de cada navegador Web. Cada navegador guarda las cookies que se han generado en las aplicaciones ejecutadas con él, y no puede acceder a las cookies generadas por otros navegadores.

Las cookies se almacenan y se leen mediante métodos de JavaScript.

Por cada cookie se almacenan esta información y en este orden:

- Dominios que pueden acceder a la cookie (puede incluir subdominios o no en función de cómo se guarde la cookie)
- Información de si es necesaria una conexión http segura para acceder a la cookie.
- Trayectoria de las URL que podrán acceder a la cookie dentro de los dominios especificados anteriormente.
- Fecha de caducidad de la cookie.
- Nombre de una entrada de datos.
- Cadena de texto asociada a ese nombre de entrada de datos.

6. Las cookies

MUY IMPORTANTE: Si el código JavaScript de un dominio crea una cookie, otro código JavaScript de otro dominio no podrá acceder a ella.

Las cookies tendrán una **fecha de caducidad**, ya que **algunos navegadores tienen limitado el número máximo de cookies que pueden almacenar**. Será el propio navegador el encargado de borrar las cookies caducadas. También es posible eliminar las cookies selectivamente através del menú de opciones o de configuración del navegador.

MUY IMPORTANTE: Para que **las cookies funcionen** correctamente la aplicación web que las genera y las consulta se ha de ejecutar desde el protocolo http/https. En definitiva **tienes que instalar la aplicación en un servidor web** e invocarla desde el navegador a través del protocolo http/https.

6.1. Gestión y uso de cookies.

Para guardar una cookie, podemos utilizar una asignación simple con la propiedad `document.cookie`.

La cookie es una serie de datos con la siguiente estructura y se guarda asignando esos datos a `document.cookie`

```
document.cookie  =  "nombreCookie=datosCookie
                    [;  expires=horaformatoGMT]
                    [;  path=ruta]
                    [;  domain=nombreDominio]
                    [;  secure]";
```

Los datos están separados unos de otros por "`;`" (excepto el último).

6.1. Gestión y uso de cookies.

Los datos **`expires, path, domain`** y **`secure`** **son opcionales**, si no se indica el navegador guardará la cookie con los **valores por defecto.** Estos datos sirven para:

- **`expires:`** fecha en formato GMT que indica la fecha de caducidad de la cookie. Si no se indica el valor por defecto, o bien, caducará al terminar la sesión (al cerrar el navegador), o bien, el navegador puede asignar una fecha de caducidad por defecto. Sin embargo, los navegadores web pueden usar la restauración de sesiones, lo que hace que la mayoría de las cookies de sesión sean permanentes, como si el navegador nunca se cerrara.

- **`path`**: ruta que tiene que tener la página web, dentro de los dominios permitidos, para que ésta pueda leer la cookie. Si no se indica se asignará la ruta de la página web desde la que se creó la cookie. **NOTA:** Si se intenta acceder a la cookie desde un dominio permitido pero no desde una ruta permitida dentro de ese dominio), se podrá acceder a la cookie para sobreescribir o eliminarla, pero no para leerla.

- **`domain`**: dominio para el cual la cookie es accesible. **Advertencia:** Si no se especifica, toma como valor por defecto el host del document.location actual, excluyendo subdominios. **Si se especifica domain, los subdominios son siempre incluidos.**

  Por ejemplo: si se establece **domain=mozilla.org** para una cookie, dicha cookie será accesible desde el subdominio **developer.mozilla.org.** Pero sino se hubiese indicado este domain, la cookie no sería accesible desde el subdomino **developer.mozilla.org**

- **`secure`**: si se indica significa solo se enviarán al servidor a través del protocolo seguro HTTPS

6.1. Gestión y uso de cookies.

Para crear o modificar una cookie:

```
document.cookie = "usuario=Martin"
```

Al ejecutar esta sentencia el navegador mira si ya existe esta cookie, si existe sobreescribe el valor, y si no existe la crea.

Si se ejecutaría después:

```
document.cookie="contador=0”
```

Si existe la cookie contador se sobreescribe y si no existe la crea, pero no borra la anterior “usuario”.

Importante: Si se va a modificar una cookie ya existente hay que hacerlo con las opciones `domain, path y secure` con las que se creó, de lo contrario se creará una nueva cookie pero con opciones distintas.

Para consultar las cookies se utiliza la misma propiedad `document.cookie` que te devolverá todas las cookies guardadas.

Para eliminar una cookie: Para forzar la eliminación de una cookie se le asigna como fecha de caducidad la actual o una fecha pasada.

```
document.cookie = "usuario=valor expires=fecha_actual"
```

Ejercicio cookies

Crea una página web llamada `cookie.html` que tenga lo siguiente:

- Un input tipo text llamado `ciudad`.
- Un botón llamado "`Guardar`": cuando el usuario pulse el botón se va a guardar el valor del input "`Ciudad`" introducido por el usuario en una cookie llamada `ciudad`. Esta cookie va a durar 1 minuto.

Ejecuta la aplicación y comprueba que la cookie ya no es accesible tras transcurrir 1 minuto

Después haz lo siguiente.

- Añade a la página web **`cookie.html`** lo siguiente:
  - Modifica la inserción de la cookie `ciudad` para que se cree con el parámetro `path=/`
  - Añade un input llamado `provincia` tipo text.
  - Cuando se pulse el botón "`Guardar`" se guardará lo introducido en el input `provincia` en una cookie llamada `provincia`. Esta cookie solo podrá ser leída por 'aplicaciones' con ruta `/cliente` del sitio web `localhost`.
  - Otro botón llamado `Consultar` que al pulsarlo muestre el valor de ambas cookies.
  - Guarda el fichero `cookie.html` en el directorio `cliente` del sitio web `localhost`.

- Crea la página web "`prueba.html`" con un párrafo HTML, en el que vuelque el contenido de `document.cookie`. Guarda el fichero prueba.html en el directorio `/tienda` del sitio web `localhost`.

Ejecuta la aplicación `localhost/cliente/cookie.html`. Guarda la cookies `provincia` y `ciudad`. Después consulta las cookies. Después ejecuta la aplicación `localhost/tienda/prueba.html`. ¿Qué cookies te muestra?, ¿por qué?

Después: añade a `cookie.html` un botón llamado "`Borrar cookies`" que borre todas las cookies.

Ejercicio cookies

Crea una página web siguiente:

- Tenga un input (con atributo `autocomplete="off"`) con un `<datalist>` llamado `Usuario`:
  - El usuario podrá introducir un nuevo usuario o elegir un usuario de la lista.
  - Cada vez que el usuario introduce un usuario en este campo se guarda en una cookie llamada `usuarioN`, siendo N un número del 1 al 4, y considerando el número mayor el más moderno. El primer usuario introducido ocupará la posición 1, el siguiente usuario introducido ocupará la posición 2, el siguiente usuario introducido ocupará la posición 3, el siguiente la posición 4, y el siguiente usuario introducido ocupará la posición 4 desplazando al que ocupaba la posición 4 a la posición 3, y al que ocupaba la posición 3 a la posición 2 y al que ocupaba la posición 2 a la posición 1, despareciendo el que ocupaba la posición 1.
  - Las opciones de este `<datalist>` se han de construir dinámicamente (mediante métodos de creación de elementos de Javascript) consultando todas las cookies que pueda haber llamadas `usuarioN`. Para ello antes se han de borrar todos los hijos del `datalist`.
- Tenga un input de tipo password llamado `"Clave: "`:
  - El funcionamiento de este input en cuanto a las cookies es igual al del input `"Usuario: "` , pero las cookies generadas serán `clave1, clave2, clave3, clave4.` Y cada una de ellas respectivamente será la clave del usuario guardado en la `cookie usuario1, usuario2, usuario3 y usuario4.`
  - Si en el `<datalist>` de `"Usuario"` el usuario eligió uno de las opciones del `<datalist>` (de una cookie), este input se rellenará solo con la cookie correspondiente `claveN` que se habrá guardado junto con `usuarioN` en una ejecución anterior.
- Un botón llamado `"Enviar"` que guardará los datos introducidos por el usuario y los enviará al servidor.