

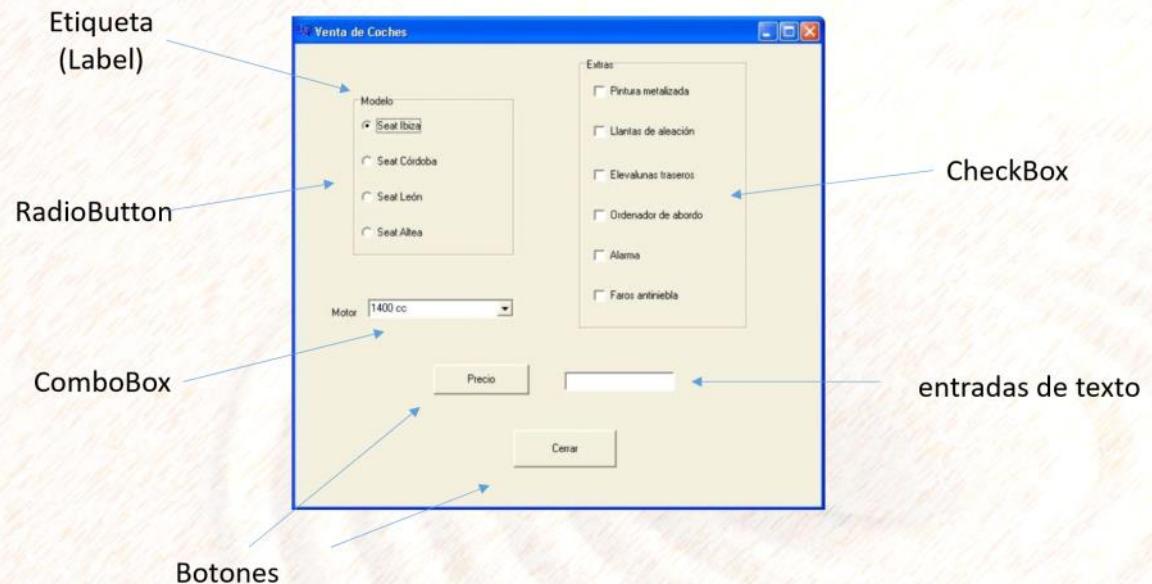
Interfaz Web. Formularios. Imágenes

Formularios en JS. Validación.

Formularios en JQ.

Imágenes. Otros detalles de interfaz

Fundamentos. Interfaces gráficos de usuario.

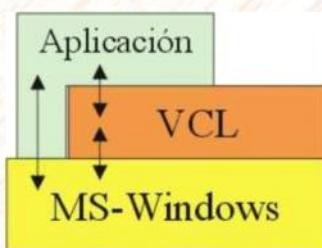


Fundamentos. GUI en Windows.



En lugar de las aplicaciones,
es el S.O el que maneja los
recursos gráficos del sistema

Hay un conjunto de funciones
denominado **API de Windows** que
puede utilizar cualquier aplicación
cada vez que desee manejar algún
recurso gráfico.



Biblioteca de componentes
visuales, dependiendo del entorno
concreto que utilicemos

De él
deriva

Formularios (Forms)



- sin necesidad de que exista un script en el servidor, es posible incorporar la información a un mensaje de correo (Poco recomendable)
- gracias a un script, es posible incorporar la información a una base de datos

Formularios HTML

<form>...</form> Incluye todo el contenido del formulario

Atributos **method** **action** **enctype**

<input> type text / email / URL

First Name:

Last Name:

City:

State:

<input> type submit / reset / button

<textarea> </textarea>

Comments:

<input> type checkbox / radio

<select> <option>valor</option>..... </select>

<input> type number / range / date...

Acceso a los formularios

Js

- 1 Automáticamente se crea un **array forms** con referencia a todos los formularios de la página

el elemento de cada uno de los formularios incluye un **array elements** con referencia a todos los elementos

Problema:

cambian los valores si se cambia el orden de los formularios originales

```
document.forms[i].elements[j];
```

- 2 Referencia a los formularios y sus elementos por su nombre (único y requerido)

```
document.[formulario].[elemento]
```

- 3 Referencia mediante las funciones DOM

Formularios: Propiedades

Js

Una vez que tenemos referencia a un formulario, con sus elementos, las propiedades de estos son

- type →
 - <input>: atributo type → text / email / URL
submit / reset
checkbox / radio
number / range / date
 - <select>: select-on
 - <textarea>: textarea
 - form → referencia directa al formulario al que pertenece el elemento.
 - name → atributo name de HTML (solo lectura)
 - value → atributo value de HTML (lectura/escritura)
- <input type="text"> y <textarea>: texto escrito
botones: texto que se muestra en el botón

Formularios: Eventos

Js

onclick →
<input type="button">
<input type="submit">
<input type="reset">
<input type="image">

onchange →
<input type="text">
<textarea>
<select>

onfocus → se selecciona cualquier elemento

onblur → deja de estar seleccionado un elemento
(perdida del foco)

Formularios: parámetros en los eventos

Js

```
<form action="#" method="post">  
<p><input type="text" name="date"  
    onBlur="validatePass(this);">  
    <input name="button" type="button" value="Probar">  
    ...  
</form>
```

onblur → deja de estar seleccionado
un elemento
(perdida del foco)

```
function validatePass(campo) {  
    if (campo.value ...)  
    ...}
```

Empleando *this*, le pasamos
a la función que maneja el
evento el elemento que lo
desencadena, con todas sus
propiedades

Obtener el valor: campo texto y textarea

Js

Cuadro de texto y textarea

```
var valor = document.form_name.element_name.value  
var valor = document.getElementById("id_name").value
```

```
<form name="form_1">...  
<input type="text" name="texto_1" id="texto_1" />  
var valor = document.form_1.texto_1.value  
var valor = document.getElementById("texto_1").value;
```

```
<textarea name="párrafo_1" id="párrafo_1"></textarea>  
var valor = document.form_1.texto_1.value  
var valor = document.getElementById("parrafo").value;
```

Obtener el valor: radiobutton

Js

Los radiobutton son un grupo de elementos y lo que importa es cual esta seleccionado: lo averiguamos recorriendo un **array** buscando el valor true

```
recorrer document.form_name.elements[i].checked  
recorrer document.getElementById("id_name")[i].checked
```

```
<input type="radio" value="si" name="pregunta" id="pregunta_si"/> SI  
<input type="radio" value="no" name="pregunta" id="pregunta_no"/> NO  
<input type="radio" value="nsnc" name="pregunta" id="pregunta_nsnc"/> NS/NC
```

```
var elementos = document.getElementsByName("pregunta");  
for(var i=0; i<elementos.length; i++) {  
    alert(" Elemento: " + elementos[i].value +  
          "\n Seleccionado: " + elementos[i].checked);  
}
```

Obtener el valor: checkbox

Js

Checkbox

```
var valor document.form_name.element_name.checked  
var valor document.getElementById("id_name").checked
```

```
<input type="checkbox" value="condiciones" name="condiciones"  
id="condiciones"/>  
He leido y acepto las condiciones
```

```
var elemento = document.getElementById("condiciones");  
alert(" Elemento: " + elemento.value + "\n  
Seleccionado: " + elemento.checked);
```

Obtener el valor: select

Js

Select

options: array de nodos creado automáticamente que contiene la referencia a todos los <option>
selectedIndex: índice de la opción seleccionada.

En dos pasos

```
var select document.getElementById("id_name").selectedIndex  
var opcion document.getElementById("id_name").options[select]
```

En un paso

```
var opcion document.getElementById("id_name").  
options[document.getElementById("id_name").selectedIndex]
```

Y finalmente

```
// Obtener el valor y el texto de la opción seleccionada  
var textoSeleccionado = opcion.text;  
var valorSeleccionado = opcion.value;
```

Obtener el valor: select. Ejemplo

Js

```
<select id="opciones" name="opciones">
  <option value="1">Primer valor</option>
  <option value="2">Segundo valor</option>
  <option value="3">Tercer valor</option>
  <option value="4">Cuarto valor</option>
</select>
```



```
var opcion = document.getElementById("opciones").options[document.getElementById("opciones").selectedIndex];

var textoSeleccionado = opcion.text; // Segundo valor
var valorSeleccionado = opcion.value; // 2
```

Formularios: foco y envío

JS

Foco: Establecer el foco en un elemento de un formulario

Método focus() en el elemento

Envío: Evitar el envío duplicado de un formulario

Propiedad disabled= true una vez pulsado el botón de enviar

En el manejador del clic del botón "enviar" añadimos

```
this.disabled= true  
this.value="Enviando"
```

Y para enviar realmente el formulario

```
this.form.submit()
```

Ejercicio (23a)

Datos de un formulario (1)

Realizar un formulario que incluya campos y áreas de texto (como en el ejercicio 20a de HTML) .

Al pulsar submit, se recogerán los valores y se presentarán sustituyendo al formulario y con la mejor estética posible

First Name:

Last Name:

City:

State:

Comments:

Objetivo: Familiarizarnos con el uso de formularios y la captura de los datos introducidos en ellos.

Ejercicio (23b)

Datos de un formulario (2).

Realizar un formulario que incluya campos y áreas de texto, junto con opciones de selección: checkbox, radiobutton y listas (como en el ejercicio 20b de HTML).

Al pulsar submit o el botón correspondiente, se recogerán los valores y se presentarán sustituyendo al formulario y con la mejor estética posible

Objetivo: Familiarizarnos con el uso de formularios y la captura de los datos introducidos en ellos.

Select definido dinámicamente

Js

Select:

Engloba un conjunto de opciones (*options*) que se cargan en un *array*

```
for (var i = 0; i < aDatos.length; i++) {  
    oDomSelect.innerHTML += "<option>" + aDatos[i] + "</option>";  
}
```

Una selección previa (*radiobutton* o *select*) determina el conjunto concreto de datos que se cargan en un *select* determinado

- se define el manejador del evento *change* del primer *select*
- en la función correspondiente se cargan los datos de un *array* en el siguiente *select*

Ejercicio (24)

Selects encadenados.

Realizar un formulario que incluya dos listas de selección (*select*) . Ambos se cargarán dinámicamente; el primero con diversos géneros de novela y el segundo con diversos autores correspondientes al género seleccionado.

Añadir un botón que sólo esté activo cuando esten completa s ambas selecciones y que al pulsarlo muestre un mensaje

Objetivo: Familiarizarnos con el uso de formularios y la captura de los datos introducidos en ellos.

Formularios: validación

Nuevos atributos en **HTML5** required / pattern

Validación **tradicional** mediante JavaScript

Condiciones básicas: elementos requeridos

Condiciones avanzadas: elementos que coinciden con un patrón: Expresiones regulares

Combinar **ambas**: comprobamos si el navegador soporta required.
Podemos hacerlo utilizando **Modernizr** (<http://modernizr.com/>)

Validación en HTML5



Los atributos HTML5 permiten definir ciertas operaciones de validación mediante atributos HTML

required → indica un campo que debe ser completado para que se envíe el formulario

pattern → permite añadir una **expresión regular** que se comprobará antes de enviar

```
<input type="text" name="zipCode" value=""  
pattern="[0-9]{5}([-][0-9]{4})?" required="required" />
```

A nivel de *form*

novalidate → desactiva la validación y sobrescribe los campos *required*

Validación y estilos



En la validación sin scripts se usan también las pseudoclases de CSS3

- :required** → permiten definir el estilo que se aplicará a los campos requeridos
- :invalid** → define el estilo de los campos que no son validos al cargar el formulario (e.g, requeridos vacíos); predomina sobre el anterior y cambia cuando se introduce un valor válido
- :valid** → define el estilo de los campos que no son validos

Actualmente, los **formatos aplicados al validar**, incluidos los de los textos emergentes, no pueden modificarse mediante estilos CSS estándares

Validación HTML5 y JS

Js

API de validación de restricciones (Constraint Validation API)

La validación definida en HTML se realiza de dos maneras

mediante el método ***checkValidity()*** de un *input*, *textarea*, *select* o *button*, que evalúa específicamente el elemento correspondiente

mediante el método ***checkValidity()*** de un formulario : **validación estática**

al principio del método ***submit()***, cuando este es disparado por el clic en un control de tipo *submit*: **validación interactiva** (mensajes emergentes)

Plantea ciertos problemas si el formulario carece de *submit*

Validación y submit

JS

API de validación de restricciones (Constraint Validation API)

La **validación interactiva** aparece inmediatamente por delante de la respuesta al **evento submit** disparado por el **formulario** como consecuencia de un I clic en un control de tipo *submit*



Si se define una función manejadora del evento *submit* del formulario:

- la función se desencadena justo después de la validación
- en ella es posible deshabilitar el comportamiento por defecto de envío y hacer cualquier operación con los datos (ya validados)

Ejercicio (25a)

Validación de un formulario con HTML5 / JS (1).

En el primero de los formularios realizados realizar una validación HTML5 que incluya la obligatoriedad de llenar campos de texto nombre y apellidos (Opcionalmente hacer que coincidan con el patrón de una expresión regular). Aplicar los estilos adecuados para hacer la validación lo más interactiva posible

Al pulsar **submit**, en caso de que todos los datos sean correctos, NO se enviara el formulario, sino que se presentaran los resultados en pantalla, como en ejercicios anteriores

Objetivo: Familiarizarnos con la validación de los formularios mediante HTML5. Emplear pseudo-clases *:required*, *:invalid*.

Expresiones regulares



Expresiones regulares

secuencia de caracteres que forman un patrón de búsqueda, método por medio del cual se pueden realizar búsquedas dentro de cadenas de caracteres.

PATRONES

Caracteres → Se representan a si mismos
Meta caracteres . * ? + [] () { } ^ \$ | \



- AWK (Unix/Linux): comandos grep, sed ...
- Java, varias bibliotecas
- JavaScript soporte integrado: objetos **RegExp**
- Perl: el lenguaje que hizo crecer a las expresiones regulares
- PCRE: biblioteca de ExReg para C, C++, Visual Basic
- PHP dos tipos diferentes
- Python: soporte mediante su librería <regex>.
- .Net Framework: clases para utilizarlas

Meta caracteres



. \ | ^ \$? * + { } [] ()

. cualquier carácter excepto nueva línea
\ caracteres especiales escapados
| alternancia: un patrón u otro

posicionamiento → ^ patrón al principio de la cadena evaluada
\$ patrón al final de la cadena evaluada

cuantificadores → ? lo precedente 0 o una vez (= opcional)
* lo precedente 0 o más veces
+ lo precedente 1 o más veces
{ } número de veces de lo precedente

{n}: n ocurrencias.
{n,m}: no menos de n ocurrencias y no más de m.
{n,}: un mínimo de n ocurrencias.

Meta caracteres: Clases



Clases [] → conjunto de caracteres válidos o no

- **Simples:** una serie de caracteres [aeiou]
- **De negación:** excepciones, incluyendo al comienzo el circunflejo (^) [^n]
- **De rango:** un rango de elementos separados por un guión (-): [a-z]
- **Predefinidas:**

.	Cualquier carácter menos salto de línea y retorno de carro [^\n\r]
\d	un dígito del 0 al 9; equivale a [0-9]
\w	cualquier carácter alfanumérico (letras del alfabeto latino y de números arábigos); equivale a [0-9_A-Za-z]
\s	un espacio en blanco [\t\n\x0B\f\r]
\D	cualquier carácter que no sea un dígito del 0 al 9 [^0-9]
\W	cualquier carácter no alfanumérico
\S	cualquier carácter que no sea un espacio en blanco [^ \t\n\x0B\f\r]

Meta caracteres: grupos



agrupaciones → () múltiples elementos como grupo de captura

permiten combinar diversas clases en patrones más complejos, aplicando cuantificadores al grupo completo.

[0-9][a-z]{2} → dígito seguido de 2 letras

([0-9][a-z]){2} → dígito / letras 2 veces

lookaheads
(búsqueda anticipada) → una clase precedida por **?=** o **?!**, que expresan si la clase debe existir o no tras un grupo especificado.
(es una condición de la búsqueda, pero no parte del grupo a buscar)

(?=.*\d) → cualquier carácter dígito aparecerá en el grupo

Carácteres especiales o escapados



\t	tabulador.
\r	"retorno de carro" o "regreso al inicio" o sea el lugar en que la línea vuelve a iniciar.
\n	"nueva línea" el carácter por medio del cual una línea da inicio
\e	tecla "Esc" o "Escape
\f	salto de página
\v	tabulador vertical
\x	caracteres ASCII
\u	caracteres unicode
\A	Representa el inicio de la cadena. No un carácter sino una posición
\Z	Representa el final de la cadena. No un carácter sino una posición
\b	Marca el inicio y el final de una palabra
\B	Marca la posición entre dos caracteres alfanuméricos o dos no-alfanuméricos

Regex: Herramientas (1)



regexpal 0.1.4 – a JavaScript regular expression tester

Case insensitive (i) ^\$ match at line breaks (m) Dot matches all (s; via [RegExp](#))

Write your regex here. Its syntax will be `/[A-Zñ][a-zñ]+/g`

Expression share save flags

Write your test data here. Matches alternate

Text 17 matches

Welcome to [RegExr](#) v2.0 by gskinner.com!

Edit the Expression & Text to see matches. Roll over matches or the expression for details. Undo mistakes with ctrl-z. Save & Share expressions with friends or the Community. A full Reference & Help is available in the Library, or watch the video Tutorial.

[RegExr.](#)

herramientas para probar expresiones regulares. Por un lado, tenemos el texto objetivo y, por otro, la expresión regex que se aplica en JavaScript

Regex: Herramientas (2)



[RegExper](#) nos permite construir el esquema que representa gráficamente el significado de la expresión regular

REGEXPER
You thought you only had two problems...

/[A-Z][a-zA-Z]+/

Display

Created by [Jeff Avallone](#) // Generated images licensed

Otras expresiones regulares



URL: `^(ht|f)tp(s?)\:\.\//[0-9a-zA-Z]([-\.\w]*[0-9a-zA-Z])*(:(0-9)*)*(\//?)([a-zA-Z0-9\-.\.?\.\'\\\+\&%\$#_]*)?$/`

Número tarjeta de crédito `^((67\d{2})|(4\d{3})|(5[1-5]\d{2})|(6011))(-?\s?\d{4}){3}|(3[4,7])\ \d{2}-?\s?\d{6}-?\s?\d{5}$`

Código postal `^([1-9]{2}|[0-9][1-9]| [1-9][0-9])[0-9]{3}$`

<http://webintenta.com/validacion-con-expresiones-regulares-y-javascript.html>

A1

Expresiones regulares en JS

Js

Objetos RegExp

```
var RegEx_nombre = /expresión regular$/;
var RegEx_nombre = new RegExp(" expresión regular $")
```

Método match()

```
campo.value.match(RegEx_nombre)
```

Devuelve un array con las coincidencias que ha encontrado, o un null si no hay coincidencias

Método search()

```
campo.value.search(RegEx_nombre)
```

Devuelve la posición del primer carácter de la primera coincidencia o -1 si no hay ninguna

Formularios: validación (1)

Js

Procedimientos basados exclusivamente en la **presencia de datos**

Validar un campo de texto obligatorio

Validar que un *checkbox* ha sido seleccionado

(Validar que un *radiobutton* ha sido seleccionado)

(Validar que se ha seleccionado una opción de una lista)

Formularios: validación

JS

Procedimientos basados en **expresiones regulares**

Validar un campo de texto con valores numéricos

Validar un número de DNI

Validar un número de teléfono

Validar una dirección de email

Validar una fecha

Validar una contraseña y su fortaleza

Validación y retro-compatibilidad

Js

Podemos facilitarlo utilizando **Modernizr**
(<http://modernizr.com/>)

Enlazamos el script correspondiente, en nuestro sitio o en una CDN

<https://cdnjs.cloudflare.com/ajax/libs/modernizr/2.8.3/modernizr.js>

El objeto *Modernizr* que se crea contiene propiedades booleanas que indican la compatibilidad con las etiquetas o atributos HTML5



if (Modernizr.input.required)

Devolverá true siempre que el navegador soporte el atributo *required* de HTML5

Ejercicio (25d)

Validación de un formulario (2).

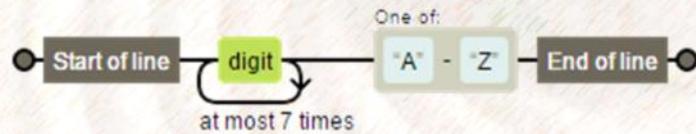
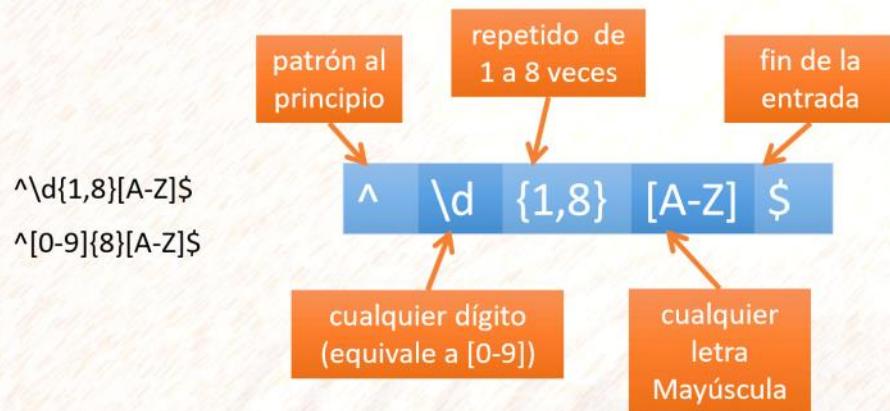
Con los formularios ya realizados realizar una validación que incluya la obligatoriedad de llenar campos de texto con determinado patron (e.g. nombre y apellidos, correo, *password*) y campos de tipo *checkbox*.

Se puede comprobar al pulsar *submit* o cuando el usuario abandona los campos correspondientes. En caso de que falten datos se informara al usuario dentro del formulario y con la mejor estética posible.

Realizar el formulario empleando **HTML5** y al mismo tiempo hacerlo compatible con un **navegador antiguo** (e.g. IE 8, que puede simularse desde IE 10)

Objetivo: Familiarizarnos con la validación de los formularios.

Expresiones regulares: DNI

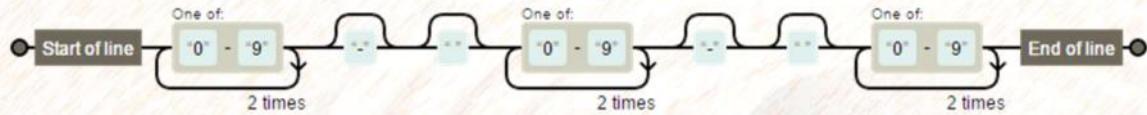


Número teléfono



Para los actuales teléfonos españoles, el formato puede ser NNN-NNN-NNN

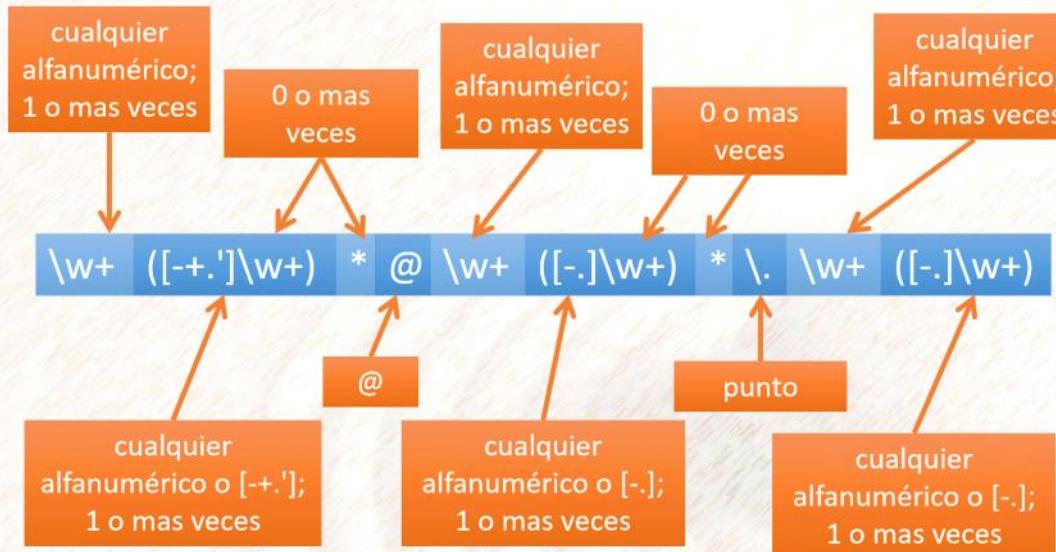
$^{\text{[0-9}\{3\}-? \text{[0-9}\{3\}-? \text{[0-9}\{3\}}\$}$



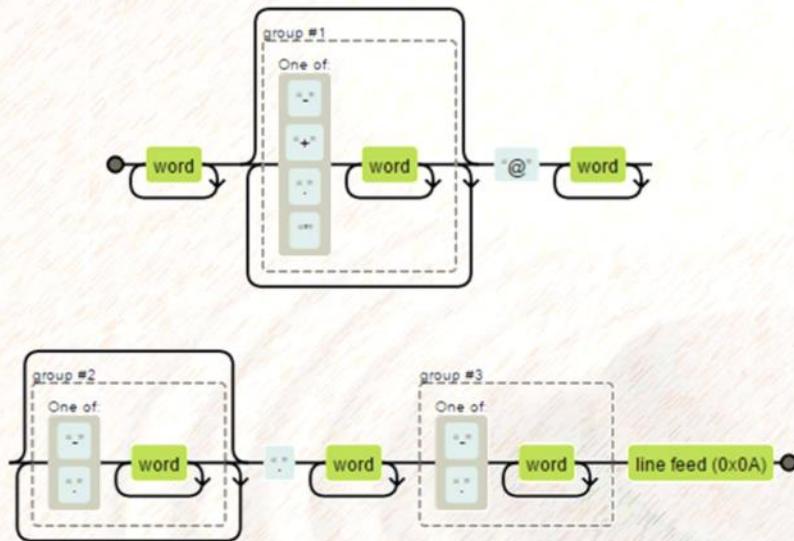
O anteponiendo el prefijo

$^{\text{[0-9}\{2,3\}-? \text{[0-9}\{3\}-? \text{[0-9}\{3\}-? \text{[0-9}\{3\}}\$}$

Expresiones regulares: correo@



Expresiones regulares: correo@



\w+([-.\']\w+)*@\w+([-.\']\w+)*\.\w+([-.\']\w+)

Validación de fechas

Js

- **Validar el formato**
- Validar el valor correcto (i.e. día válido según mes y año)
- Almacenar la variable Date correspondiente

Como existen diversos formatos, conviene indicar, junto al campo de datos, cual es el admitido, e.g. DD/MM/AA(AA)

`^\d{1,2}\/\d{1,2}\/\d{2,4}$`



Validación de fechas

Js

- Validar el formato
- **Validar el valor correcto (i.e. día válido según mes y año)**
- Almacenar la variable Date correspondiente

variables numéricas: nDia, nMes, nAño
creamos una variable tipo fecha

```
var date = new Date(nAño,nMes-1,nDia)
```

extraemos el mes de la variable tipo fecha

```
var nDateMes = date.getMonth();
```

comprobamos que el mes extraído es el mismo que el suministrado

```
var nDateMes + 1 == nMes ;
```

Ejercicio (25b)

Validación de una fecha con HTML5 / JS y con HTML / JS.

Creamos una página con 2 formularios.

- En el primero recogemos y validamos una fecha con HTML5
- En el segundo no utilizamos HTML5

En cada caso habrá un botón "probar fecha", que mostrará si es incorrecto o la fecha en formato DD/MM/AA indicando el día de la semana correspondiente

Como anteriormente, al pulsar **submit**, en caso de que todos los datos sean correctos, NO se enviara el formulario, sino que se presentaran los resultados en pantalla.

Objetivo: Familiarizarnos con la validación de fechas mediante HTML5. Emplear pseudo-clases *:required*, *:invalid*. Compararlo con la validación sin HTML5

Contraseña segura

Js

Elementos necesarios
para una contraseña
"segura" (de fortaleza
razonable)

- Letras minúsculas
- Letras mayúsculas
- Números
- Caracteres no alfanuméricos
- No menos de 8 caracteres
- No incluir palabras del diccionario

En HTML5, el *input type = "password"* **enmascara los caracteres** que el usuario teclea (solo en su presentación por pantalla)

En HTML anterior se puede reproducir este proceso capturando cada evento de teclado del campo input

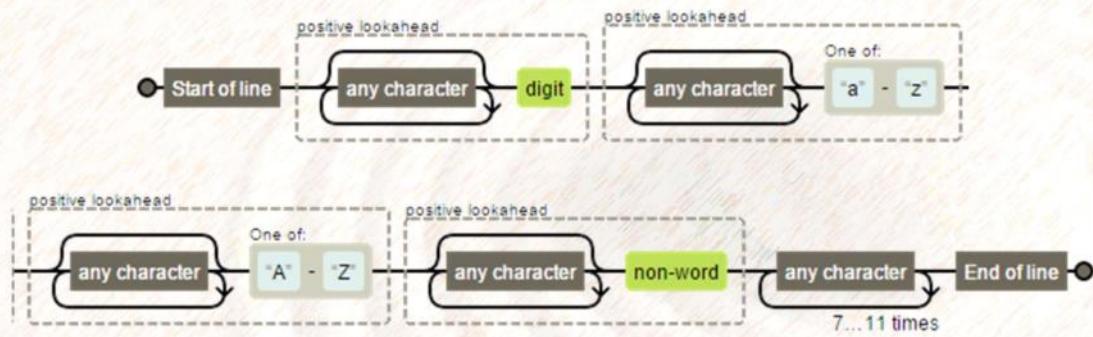
El envío de contraseñas debe hacerse siempre en **POST** y con algún mecanismo de **encriptación** (i.e. https)

Contraseña segura



Los requisitos de contraseña segura, excepto el último, se recogen en una expresión regular que utiliza *lookaheads*

`^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*\W).{8,12}$`



Validación y contraseñas

Js

API de validación de restricciones (Constraint Validation API)

Si además del ajuste al patrón, se necesita comprobar la igualdad entre 2 entradas, es interesante cambiar el comportamiento de la segunda:

- se define un manejador del evento input, incorporado en HTML5
- en el se evalúa la igualdad entre las dos entradas
- se genera un mensaje "a medida en el API de validación de restricciones

```
var msg = "";
if (domPass1.value != domPass2.value) {
    msg = "Las contraseñas no coinciden"
}
domPass2.setCustomValidity(msg)
```

Ejercicio (25c)

Validación de una fecha con HTML5 / JS .

Realizamos un formulario que solicite por duplicado una contraseña y valida que cumpla los criterios de seguridad mencionados.

Como anteriormente, al pulsar **submit**, en caso de que todos los datos sean correctos, NO se enviara el formulario, sino que se presentaran los resultados en pantalla

Objetivo: Familiarizarnos con la validación de los formularios.

Contraseñas y expresiones regulares

```
<script type="text/javascript">
<!--
function validatePass(campo) {
    var RegExPattern = /(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{8,10})$/;
    var errorMessage = 'Password Incorrecto.';
    if ((campo.value.match(RegExPattern)) && (campo.value!=='')) {
        alert('Password Correcto');
    } else {
        alert(errorMessage);
        campo.focus();
    }
//-->
</script>
```

Probar
Entre 8 y 10 caracteres,
por lo menos un digito y un alfanumérico,
y no puede contener caracteres espaciales

```
<form action="#" method="post">
<p><input type="text" name="date"
    onblur="validatePass(this);">
    <input name="button" type="button"
        value="Probar"> <br>
```

Entre 8 y 10 caracteres, por lo menos un
digito y un alfanumérico, y no puede
contener caracteres espaciales
</form>

Validación de formularios en HTML5

sábado, 14 de julio de 2018 18:49

La llegada de HTML5 supuso un cambio radical en la validación de los formularios.

El nuevo estándar incluía dos mecanismos novedosos que permitían llevar a cabo la validación sin necesidad de utilizar JavaScript:

- las restricciones implícitas incluidas en algunos de los valores semánticamente apropiados del atributo type, como 'URL' y 'email'
- las restricciones explícitas proporcionadas por los atributos correspondientes a la validación, como son required, pattern, min, max, o maxlength

Si cualquiera de estas restricciones no se cumple, el evento submit no se dispara y en su lugar aparecen una serie de mensajes emergentes cuyo aspecto y contenido dependen del navegador.

El problema surge cuando los mensajes no son adecuados o se necesitan definir otras restricciones, diferentes de las contempladas en el estándar. Ambas dificultades se resuelven fácilmente conociendo el funcionamiento del proceso de validación y el API disponible para manejarlo, conocido como API de Validación de Restricciones (Constraint Validation API).

La validación depende la propiedad validity de los controles de formulario y del propio formulario. Se trata de un objeto que a su vez incluye las siguientes propiedades:

- badInput
- customError
- patternMismatch
- rangeOverflow
- rangeUnderflow
- stepMismatch
- tooLong
- tooShort
- typeMismatch
- valueMissing
- valid

La última de las propiedades ajusta su valor en función de todas las demás, y para conócelo existe el método checkValidity, que devuelve el estado de validez true o false del control o del formulario.

Cuando se intenta lanzar un evento submit, antes de que ocurra se evalúan estas propiedades y solo en ausencia de cualquier valor false se dispará finalmente el evento submit.

Validaciones a Medida

Para añadir o modificar las validaciones HTML basta con definir una función manejadora del evento input de un control de formulario en la que estableceremos el valor del mensaje asociado al customError.

Como ejemplo, tomaremos un sencillo formulario que solicita nombre y contraseña, junto con la confirmación de la segunda.

- El nombre es requerido, pero queremos definir el mensaje que se mostrará
- Los dos campos de contraseña son requeridos según una expresión regular, pero además queremos comprobar que ambos coinciden.

Este sería el formulario

```
<label for="Nombre">Nombre:</label>
<input type="text" id="Nombre" name="Nombre" required>
<label for="Clave1">Introduzca su clave de acceso</label>
<input id="Clave1" maxlength="12" name="Clave1" type="password" required pattern="(?=^.{8,}$)((?=.*\d|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$)">
<label for="Clave2">Repita la clave de acceso</label>
<input id="Clave2" maxlength="12" name="Clave2" type="password" required pattern="(?=^.{8,}$)((?=.*\d|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$)">
```

En el código, el primer paso sería necesario definir los nodos del DOM y sus manejadores de eventos

```
domNombre = document.querySelector('#Nombre')
domPass1 = document.querySelector('#Clave1')
domPass2 = document.querySelector('#Clave2')

domNombre.addEventListener('input', validaNombre)
domPass2.addEventListener('input', compararPassw)
```

Lo siguiente son los manejadores de eventos, teniendo en cuenta que su papel no es realmente validar, sino únicamente definir el estado en que se encuentra el customError y su correspondiente mensaje, que se define gracias a la propiedad setCustomValidity(). Por eso se asocian al evento input, dado que no van a tener ningún efecto visible que moleste al usuario.

En el caso del nombre, comprobamos su estado gracias a checkValidity(), para asignarle o no un mensaje de error (incluso cuando es valido le damos el valor de carácter nulo, para resetear validaciones custom previas)

```
validaNombre() {
  let msg = ''
  // 
  this.domNombre.setCustomValidity(msg)
  if(!this.domNombre.checkValidity()){
    msg = 'Es necesario indicar el nombre'
  }
  this.domNombre.setCustomValidity(msg)
}
```

En el caso de la segunda contraseña, se comprueba su igualdad con la primera, y en caso de que no sea así, se define un mensaje de validación personalizado

```
compararPassw() {
  let msg = "";
  if (this.domPass1.value != this.domPass2.value) {
    msg = "Las contraseñas no coinciden"
  }
  this.domPass2.setCustomValidity(msg)
};
```

Cuando se pulsa el botón de enviar se desencadena el proceso de validación interactiva de restricciones, que en respuesta al estado de validity de cada control muestra los mensajes apropiados, en este caso proporcionados por nuestro código

