



■ PROGRAMACION III

Clase Teórica: UNIDAD III

DOCENTES DE CATEDRA:

- ❖ Esp. Cecilia E. Gallardo
- ❖ Esp. Marta Miranda
- ❖ Lic. Oscar Quinteros
- ❖ Julián Acosta Argañaraz





UNIDAD III: PROGRAMACIÓN DEL LADO DEL CLIENTE

Lenguajes de Programación del lado del Cliente

JavaScript



Lenguajes de programación del lado del cliente. Definición

- Los lenguajes del lado del cliente surgen por la necesidad de incorporar a las páginas web elementos dinámicos y comportamientos programados.
- Como el lenguaje **HTML** es incapaz de proporcionar el control de los elementos dinámicos de una web, se recurre a incluir pequeños programas o **scripts** en el código de la página web que serán **interpretados** y **ejecutados** por el navegador.
- La programación del lado del cliente tiene como principal ventaja que la ejecución de la aplicación se delega al cliente, con lo cual se evita recargar al servidor de trabajo.
- Un lenguaje del lado cliente es totalmente independiente del servidor.



Lenguajes de programación del lado del cliente: Scripts

- Un **script** en el lado del cliente (browser) es un programa que puede acompañar a un documento HTML o que puede estar incluido en él.
- Las instrucciones del programa se ejecutan cuando se carga el documento, o cuando se desencadena algún evento de usuario.
- Los scripts ofrecen la posibilidad de extender los documentos HTML de maneras activas e interactivas. Por ejemplo:
 - Modificar los contenidos del documento dinámicamente.
 - Procesar datos de formularios a medida que éstos se introducen.
 - Controlar los eventos que se producen en el documento: movimientos de foco sobre elementos, del ratón, etc.



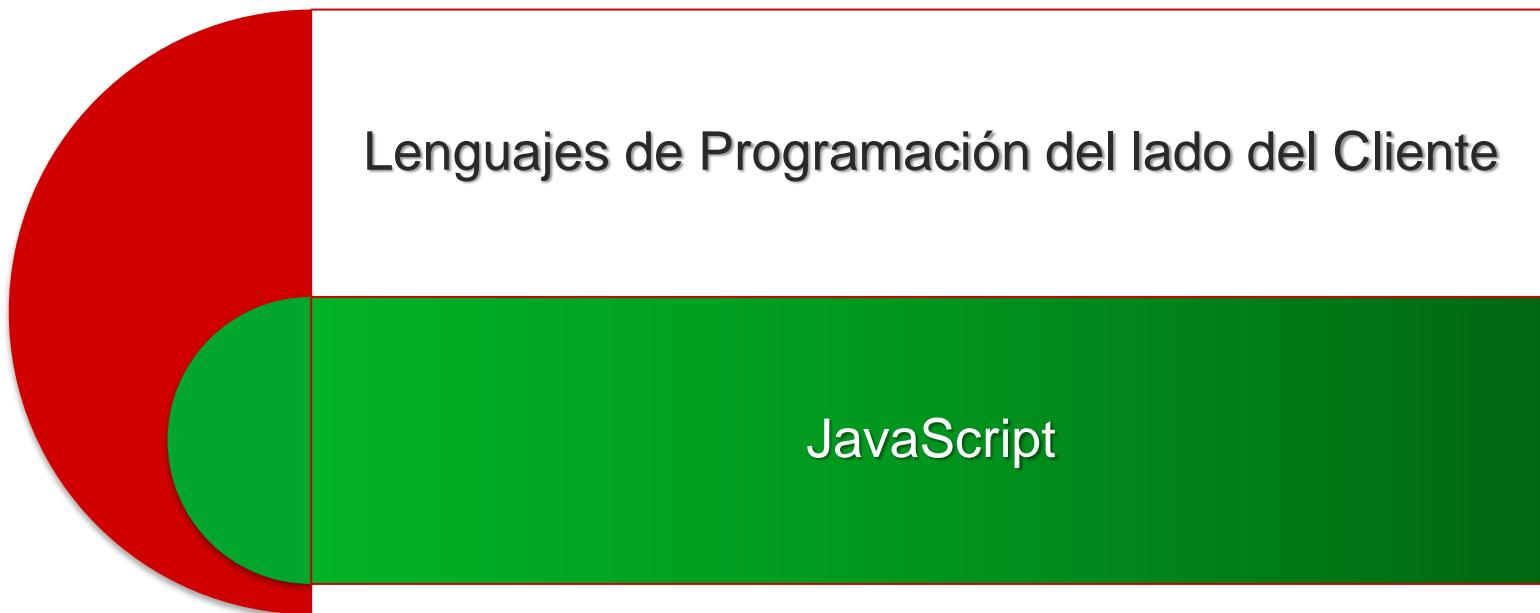
Lenguajes de programación del lado del cliente: Ejemplos

Ejemplos de algunos lenguajes de script son:

- **ECMAScript:** Lenguaje de scripting que soporta el estándar ECMA-262 (European Computer Manufacturers Association).
- **ActionScript:** Lenguaje de script de Macromedia para la aplicación Flash.
- **JScript:** es la implementación de Microsoft de *ECMAScript*. Está disponible mediante Internet Explorer y el Windows Scripting Host.
- **VBScript:** es un lenguaje interpretado por el Windows Scripting Host de Microsoft. Su sintaxis refleja su origen como variación del lenguaje de programación Visual Basic.
- **JavaScript:** es una *implementación* que realizó la empresa Netscape del estándar ECMAScript.



UNIDAD III: PROGRAMACIÓN DEL LADO DEL CLIENTE





JavaScript. Definición

En la actualidad, se considera a JavaScript como uno de los 3 lenguajes estándar que todos los desarrolladores web deben conocer:

1. **HTML** para definir el contenido de las páginas web
2. **CSS** para especificar el diseño de las páginas web
3. **JavaScript** para programar el comportamiento de las páginas web



JavaScript. Definición

- JavaScript es un lenguaje de scripting (o interpretado) por lo que no es necesario compilar los programas para ejecutarlos.
- JavaScript es código de programación que puede ser insertado en páginas HTML y puede ser ejecutado por todos los navegadores.
- A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación **Java**, pero se inspiró en éste.
- JavaScript fue inventado por Brendan Eich. Se publicó desde la compañía del navegador Netscape en 1995.
- Luego fue adoptado por **ECMA** (*European Computer Manufacturers Association*) desde el año 1997.
- **ECMA-262** es el nombre oficial del estándar y ECMAScript es el nombre oficial del lenguaje.

<http://www.ecma-international.org/publications/standards/Ecma-262.htm>



JavaScript. Características

- JavaScript es un lenguaje interpretado que desde el lado del cliente permite acceder a los objetos que utiliza el navegador y al DOM (Document Object Model), que representa la jerarquía de objetos de un documento HTML.
- JavaScript añade soporte para control de eventos, de tal forma que el programa puede interactuar con el usuario, lo que hace el diseño de las páginas web más flexible, dinámico y con una respuesta más rápida.
- JavaScript es un lenguaje basado en prototipos. No existe el concepto de “clase”, pero interacciona con objetos a los que expone al entorno. JavaScript puede acceder a objetos en el navegador.
- Es posible ejecutar código javascript del lado del servidor. *Rhino*, *Spider Monkey* y *Node.js*, son algunos ejemplos de implementaciones de Javascript o ECMAScript



JavaScript. Características

Concretamente con JavaScript podremos:

- Manipular la página web para habilitar o inhabilitar acciones/campos, completar campos calculados, crear y eliminar controles en forma dinámica, etc.
- Manipular información local sobre la sesión del usuario, por ejemplo, para conocer el tiempo restante antes de que expire
- Conocer el tipo de navegador que tiene el cliente para verificar si nuestra aplicación puede correr en ella o no
- Enriquecer la capacidad que tienen los controles estándar de HTML para mejorar "la experiencia de usuario", como los controles calendario, spinner, gauge, etc.
- Búsquedas con autocompletado.
- Hacer validaciones sencillas y validaciones más complejas



JavaScript. Inclusión en páginas HTML/XHTML

■ INCLUIR JAVASCRIPT EN ELEMENTOS HTML/XHTML

Es el método menos utilizado, ya que consiste en incluir porciones de JavaScript dentro del código HTML de la página:

```
<html>
  <head> ...
  </head>
  <body>
    <p onclick="alert('Un mensaje de prueba')">
      Un párrafo de texto.
    </p>
  </body>
</html>
```

INCONVENIENTES:

Ensucia innecesariamente el código HTML de la página y complica el mantenimiento del código JavaScript.



JavaScript. Inclusión en páginas HTML/XHTML

■ INCLUIR JAVASCRIPT EN EL MISMO DOCUMENTO HTML

El código JavaScript se encierra entre etiquetas <script> y se puede incluir en cualquier parte del documento, aunque se recomienda definir el código JavaScript dentro de la cabecera del documento:

```
<html>
  <head>
    <script>
      alert("Un mensaje de prueba");
    </script>
  </head>
  ...
</html>
```

Anteriormente se debía utilizar el atributo type="text/javascript", pero ya no es obligatorio dado que JavaScript es el lenguaje de scripts por defecto en HTML y todos los navegadores modernos

INCONVENIENTES:

Si se quisiera hacer una modificación en el bloque de código, es necesario modificar todas las páginas que incluyen ese mismo bloque de código JavaScript.



JavaScript. Inclusión en páginas HTML/XHTML

■ DEFINIR JAVASCRIPT EN UN ARCHIVO EXTERNO

Las instrucciones JavaScript se pueden incluir en un archivo externo con extensión “.js”, los cuales son enlazados mediante la etiqueta **<script>** en los documentos HTML . Cada documento HTML puede enlazar tantos archivos JavaScript como necesite.

```
<html>
  <head>
    <script src="/js/codigo.js">
    </script>
  </head>
  ...
</html>
```

VENTAJAS:

Simplifica el código XHTML de la página. Se puede reutilizar y mantener de manera inmediata el mismo código JavaScript en todas las páginas del sitio web.



JavaScript. Soporte en Navegadores

- Algunos navegadores (aunque muy pocos en la actualidad), puede ser que no dispongan de soporte completo de JavaScript, otros permiten bloquear parcial o completamente el uso de JavaScript.
- Si una página web requiere JavaScript para su correcto funcionamiento, se puede incluir un mensaje de aviso indicando que se debería activar JavaScript.
- El lenguaje HTML define la etiqueta **<noscript>** para mostrar un mensaje al usuario cuando su navegador no puede ejecutar JavaScript. Se puede incluir dentro de la etiqueta **<head>** o **<body>** :

```
<body>
  <noscript>
    <p>Bienvenido a Mi Sitio</p>
    <p>La página que Ud está viendo
       requiere para su funcionamiento
       el uso de JavaScript.
    </p>
  </noscript>
</body>
```



JavaScript. Sintaxis Básica

- En JavaScript no se define el *tipo* de las variables, es decir que una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.
- Las sentencias se separan por un punto y coma “;”
- JavaScript es sensible a mayúsculas e ignora espacios adicionales.
- Se pueden incluir comentarios de una o varias líneas (// y /*..*/ respectiv)
- Identificadores y palabras reservadas:

break - delete - function - return - typeof - case - do - if - switch - var catch - else - in - this - void - continue - false - instanceof - throw while - debugger - finally - new - true - with - default - for - null - try arguments - eval



JavaScript. Variables

- Antes de usar variables se deben declarar con la palabra reservada “**var**” y las mismas pueden opcionalmente ser inicializadas al momento de la declaración.
- Si no se inicializa toma por defecto un valor especial en JavaScript, que es “**undefined**”.

Ejemplo: **var nombreauto;** **var nombreauto="Volvo";**

- No son tipadas, por lo tanto durante su tiempo de ejecución pueden tomar diferentes valores de diferentes tipos.
- Si se vuelve a declarar una variable en JavaScript, no perderá su valor. En el ejemplo, la variable “nombreauto” aún tendrá el valor “Volvo”, después de la ejecución de las dos sentencias siguientes:

var nombreauto="Volvo"; var nombreauto;



JavaScript. Tipos de variables

- JavaScript divide los distintos tipos de variables en dos grupos: **tipos primitivos** y **tipos objetos**.

■ TIPOS PRIMITIVOS

- **undefined.** Corresponde a aquellas variables que no fueron declaradas y también a las variables que han sido declaradas pero que aún no se les asignó un valor.
- **boolean.** Sólo almacenan uno de los dos valores: true o false.
- **number.** JavaScript tiene un solo tipo numérico y todos los números se representan como valores de coma flotante.

var x1=34; var x2=35.50; var y=123e5; (y=12300000) **var y1=123e-5;**
(y1= 0.00123)

- **string.** Permiten almacenar cualquier sucesión de caracteres Unicode, dentro de comillas dobles o simples.

var carname="Volvo XC60"; var carname='Fiat 500';



JavaScript. Tipos de variables

■ TIPOS PRIMITIVOS

JavaScript es un lenguaje de programación "**no tipado**", lo que significa que una misma variable puede guardar diferentes tipos de datos a lo largo de la ejecución de la aplicación:

```
var x; // x tiene un valor indefinido  
  
var x = 5; // Ahora x es un numero  
  
var x = "John"; // Aqui x es un String
```



JavaScript. Tipos de variables

TIPOS OBJETOS

- **Object.**
 - **Array**
 - **Date**
 - **Function**
 - **Null**
- Todos los objetos JavaScript heredan propiedades y métodos de un “prototype” (prototipo).
 - Los objetos Date heredan de **Date.prototype**. Los objetos Array heredan de **Array.prototype**. Los objetos de persona heredan de **Person.prototype**.
 - A su vez, **Date.prototype**, **Array.prototype** y **Person.prototype** heredan de **Object.prototype**.



JavaScript. Funciones

- Una función es un bloque de código que se ejecutará cuando "alguien" lo convoque, ya sea cuando ocurra un evento o desde cualquier parte del código JavaScript.
- **SINTAXIS:**
function nombreFuncion ()
{ código a ser ejecutado }
- **LLAMADO A UNA FUNCIÓN CON ARGUMENTOS:**

```
<button onclick="myFunction('Luis Perez','ingeniero')">Clic aquí</button>
<script>
    function myFunction(nombre, profesion) {
        alert("Bienvenido" + nombre + ", de profesión: " + profesion);
    }
</script>
```



JavaScript. Funciones

■ FUNCIONES CON UN VALOR DE RETORNO:

```
function myFunction(a,b) {  
    return a*b;  
}  
var myVar=myFunction(4,3); // myVar= 12
```

- **VARIABLES LOCALES:** una variable declarada (utilizando **var**) dentro de una función JavaScript tiene un ámbito **LOCAL** y sólo se puede acceder a la misma desde esa función. Se eliminan cuando finaliza la función.
- **VARIABLES GLOBALES:** las variables declaradas fuera de una función, tienen un ámbito **GLOBAL**, independientemente de si se define utilizando la palabra reservada **var** o no y todos los scripts y funciones en la página web pueden acceder a ellas. Se eliminan cuando se cierra la pagina.

Por otra parte, si se asigna un valor a una variable que **no ha sido declarada**, la misma será automáticamente declarada como variable **global**, incluso si se ejecuta dentro de una función.



JavaScript. Funciones

EJEMPLO DE VARIABLES DE ALCANCE LOCAL Y GLOBAL

```
var indice= 0,5; // variable de alcance global

function calculoIndice (a,b) {
    var medida = 0,8 // variable de alcance local
    return (a*medida + b*medida) * indice;
}

var myVar= calculoIndice(4,3); // myVar= 2,48
```



JavaScript. Expresiones de Funciones

- Una función de JavaScript también se puede definir con una *expresión* y a la vez se puede almacenar en una variable:

```
var x = function (a,b) { return a * b; }  
var z = x(4,3); // z= 12
```

FUNCION ANONIMA

Las funciones almacenadas en variables no necesitan un nombre. Siempre se invocan usando el nombre de la variable.

- Las *expresiones de funciones* se pueden hacer "invocando a sí mismas", y ejecutándose automáticamente sin ser llamada.

```
(function () {  
    return 5 * 3;  
} )();
```

FUNCION ANONIMA AUTOINVOCADA



JavaScript. Tipos de variables

■ TIPOS OBJETOS

- **Object.** Los objetos JavaScript se definen entre llaves y sus **propiedades** se especifican como pares “**nombre: valor**” separados por comas. Los **métodos** se almacenan en propiedades como definición de funciones.

```
var persona = {  
    nombre : "Luis",  
    apellido : "Perez",  
    edad : 50,  
    nombreCompleto: function() {return this.nombre+ " " + this.apellido;}  
};
```

```
persona.nombreCompleto() + " tiene " + persona.edad + " años.";
```



JavaScript. Tipos de variables

■ TIPOS OBJETOS

- **Object.** Para crear varios objetos del “mismo tipo” (similar a una clase), utilizar una función **constructor de objetos**:

```
function Persona(nombre, apellido, edad) {  
    this.nombre = nombre;  
    this.apellido = apellido;  
    this.edad = edad;  
    this.cambioNombre = function (nombre) {  
        this.nombre = nombre;  
    };  
};  
  
var persona1 = new Persona("John", "Doe", 50);  
var persona2 = new Persona("Sally", "Rally", 48);  
persona2.cambioNombre("Holly"); //persona2.nombre -> Holly
```



JavaScript. Tipos de variables

■ TIPOS OBJETOS: OBJECT

- El constructor anterior crea una función “**cambioNombre**” para cada uno de los objetos que se creen con este constructor.
- Si creamos un array con 1000 instancias de **Persona** se crearán 1000 funciones distintas (e independientes) => **Eficiencia indeseable**
- **Solución:** utilizar el atributo heredado “**prototype**” que es un objeto vacío y simple pero potente:

```
function Persona(nombre, apellido, edad) {
```

```
    this.nombre = nombre;
```

```
    this.apellido = apellido;
```

```
    this.edad = edad;
```

```
};
```

```
Persona.prototype.cambioNombre = function (nombre) {
```

```
    this.nombre = nombre;
```

```
};
```

La función pasa a ser parte del prototipo y no de la instancia



JavaScript. Tipos de variables

■ TIPOS OBJETOS

- **Array.** En JavaScript, los **arrays** siempre utilizan indices numerados, comenzando por el cero. *Ejemplos de creación de array:*

```
var cars = ["Saab","Volvo"];
```

```
var cars = new Array("Saab","Volvo");
var misc = [{}, true, "a"];
```

```
var cars=new Array();
cars[0]="Saab";
cars[1]="Volvo";
```

- **Date.** Un tipo Date consiste en un año, mes, dia, hora, minuto, segundo y milisegundos. *Formas de creación:*

```
new Date() // crea un nuevo objeto con la fecha y hora actual
```

```
new Date(milliseconds)
```

```
new Date(dateString)
```

```
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```



JavaScript. Tipos de variables

■ TIPOS OBJETOS

- **Null.** En JavaScript null es "nada". Se supone que es algo que no existe.
- **Function.** Las funciones son tambien objetos en JavaScript.

■ DIFERENCIAS ENTRE UNDEFINED Y NULL

- **typeof undefined // undefined**
- **typeof null // object**
- **null === undefined // false**
- **null == undefined // true**



JavaScript. Operadores

- OPERADORES DE ASIGNACIÓN: dado $x=10$ e $y=5$:

Operador	Ejemplo	Igual que	Resultado
$+$	$x=y$		$x=5$
$+=$	$x+=y$	$x=x+y$	$x=15$
$-=$	$x-=y$	$x=x-y$	$x=5$
$*=$	$x*=y$	$x=x*y$	$x=50$
$/=$	$x/=y$	$x=x/y$	$x=2$
$%=$	$x\%=y$	$x=x\%y$	$x=0$

- OPERADORES LÓGICOS: dado $x=true$ e $y=false$

Operador	Descripción	Ejemplo
!	Negación	$!(x==y)$ es true
$\&\&$	AND	$(x < 10 \&\& y > 1)$ es true
$\ $	OR	$(x==5 \ y==5)$ es false

JavaScript. Operadores

- OPERADOR CONDICIONAL:** dado **x=10** e **y=5**:

Sintaxis	Ejemplo
<code>variablename=(condition)?value1:value2</code>	<code>x=(age<18)?"Muy joven":"Bastante mayor";</code>

- OPERADORES DE COMPARACIÓN:** se utilizan en instrucciones lógicas para determinar la igualdad o diferencia entre variables o valores. Dado **x=5**:

Operador	Descripción	Ejemplo	Resultado
<code>==</code>	Igual que	<code>x==8</code>	false
<code>====</code>	exactamente igual (igual valor e igual tipo)	<code>x===="5"</code>	false
<code>!=</code>	No igual que	<code>x!=8</code>	true
<code>!==</code>	No es igual (diferente valor o diferente tipo)	<code>x!=="5"</code>	True
<code>>, >=</code>	Mayor que, mayor o igual que	<code>x>=8</code>	false
<code><, <=</code>	Menor que, menor o igual que	<code>x<8</code>	true



JavaScript. Operadores Aritméticos

Operador	Descripción	Ejemplo	Resultado de x	Resultado de y con valor = 5
+	Suma	$x=y+2$	7	5
-	Resta	$x=y-2$	3	5
*	Multiplicación	$x=y*2$	10	5
/	División	$x=y/2$	2.5	5
%	Modulo (resto de la división)	$x=y \% 2$	1	5
++	Incremento	$x=++y$ $x=y++$	6 5	6 6
--	Disminución	$x=--y$ $x=y--$	4 5	4 4



JavaScript. Operadores de tipo

Operador	Descripción	Ejemplo	Resultado de x	Resultado de y con valor = 5
typeof	Retorna el tipo de una variable	x=y+2	7	5
instanceof	Retorna “true” si un objeto es una instancia de un tipo de objeto	x=y-2	3	5



JavaScript. Estructuras de Control

■ SENTENCIAS CONDICIONALES:

```
if (condition) {  
    sentencias  
}
```

```
if (condition) {  
    sentencias  
} else {  
    sentencias }
```

```
if (condition1) {  
    sentencias  
} else if (condition2)  
{ sentencias }  
else { sentencias }
```

```
switch(expresion) {  
    case n:  
        sentencias1  
        break;  
    case n:  
        sentencias2  
        break;  
    default:  
        sentencias  
}
```

■ BUCLE FOR:

```
for (statement 1; statement 2; statement 3)  
{ sentencias }
```

```
for ( i = 0; i < 5; i++ )  
{ x=x + "El numero es " + i + "<br>"; }
```

Statement 1 se ejecuta antes de que comience el bucle.

Statement 2 define la condición para ejecutar el bucle.

Statement 3 se ejecuta cada vez, después de que el bucle se ha ejecutado.



JavaScript. Estructuras de Control

- **BUCLE FOR/IN:** itera a través de las propiedades de un objeto

```
var person={ fname:"John",lname:"Doe",age:25 };
```

```
for (x in person) { txt=txt + person[x]; }
```

```
for (indice in array) { ... }
```

- **BUCLES WHILE / Do WHILE:**

```
while (condicion) { sentencias }
```

```
do { sentencias } while (condicion)
```

- **SENTENCIA BREAK:** permite terminar de forma abrupta un bucle

```
for(i in letras) {  
    if(letras[i] == 'a') { break; } ...}
```

- **SENTENCIA CONTINUE:** permite saltarse algunas repeticiones del bucle

```
for(i in letras) {  
    if(letras[i] == 'a') { continue; } ...}
```



JavaScript. DOM HTML

- DOM (Document Object Model) es un API estandar de W3C que define una estructura para acceder a documentos. Ver referencia en <http://w3schools.com/jsref/default.asp>
- El estándar W3C DOM está separado en 3 partes:
 - Core DOM – modelo estándar para documentos de todo tipo
 - XML DOM - modelo estándar para documentos XML
 - HTML DOM - modelo estándar para documentos HTML
- A través del DOM HTML, JavaScript puede acceder y manipular a todos los elementos de un documento HTML.
- Cuando se carga una página, los browsers transforman automáticamente la página, construyendo un DOM como un árbol de **objetos (nodos)**

JavaScript. DOM HTML

DOM

Document Object Model

document

Root element:
<html>

Element:
<head>

Element:
<title>

Element:
<body>

Element:
<h1>

Text:
"A heading"

Element:
<a>

Attribut:
href

Text:
"Link text"

Cada etiqueta HTML se transforma en un nodo de tipo "Elemento"

El nodo tipo "Texto" contiene el texto encerrado por la etiqueta HTML.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <h1>A heading</h1>
    <a href="..">Link text</a>
  </body>
</html>
```



JavaScript. DOM: tipos de nodos

- La especificación completa de DOM HTML define alrededor de 12 tipos de nodos, aunque las páginas HTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:
 - **document:** nodo raíz del que derivan todos los demás nodos del árbol y que representa una página web en si misma.
 - **element:** representa cada una de las etiquetas HTML. Es el único nodo que puede contener atributos y del que pueden derivar otros nodos.
 - **attr:** representa cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par atributo=valor.
 - **text:** nodo que contiene el texto encerrado por una etiqueta HTML.
 - **comment:** representa los comentarios incluidos en la página HTML.

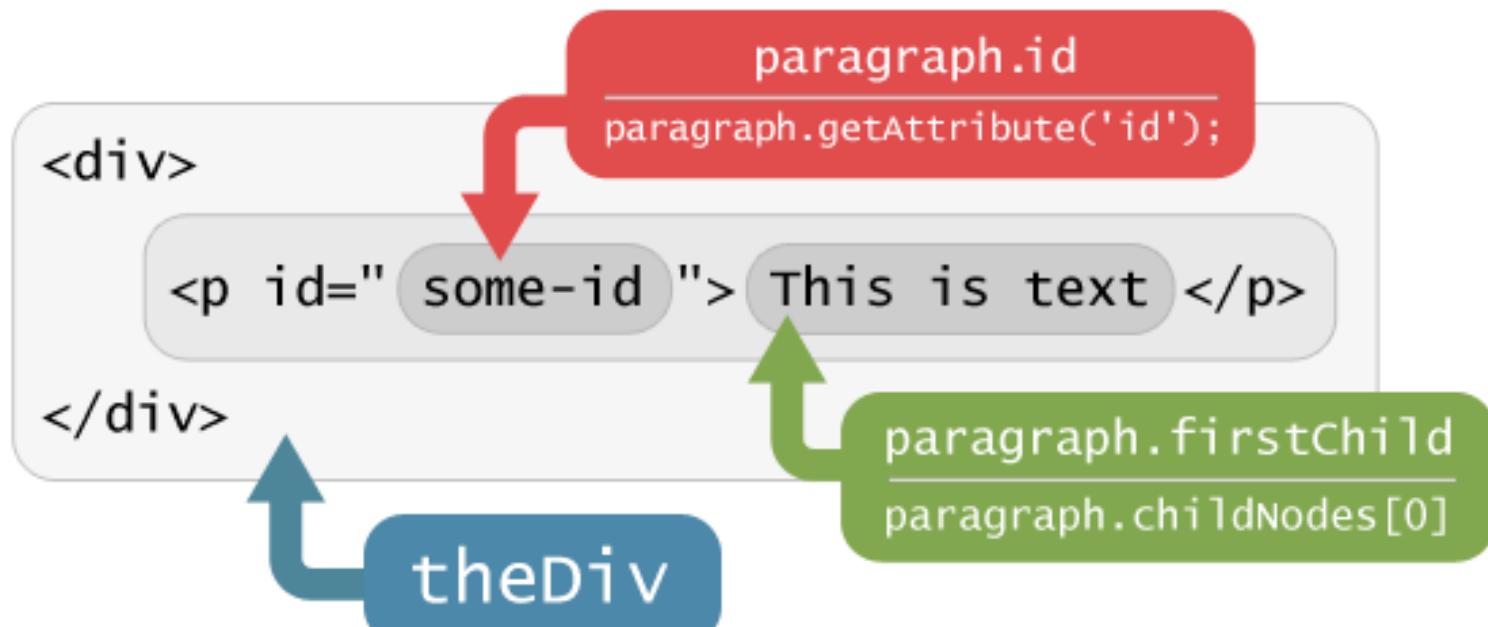


JavaScript. DOM: acceso directo a nodos

- **getElementById()**: método que retorna el elemento con el atributo ID especificado. Sintaxis: `document.getElementById("nombreId");`
- **getElementsByName()**: método que retorna una colección de todos los elementos cuya etiqueta sea igual que el parámetro que se le pasa a la función. Sintaxis: `document.getElementsByTagName("tagname");`
- **getElementsByName()**: es similar al anterior, pero en este caso se buscan los elementos cuyo atributo **name** sea igual al parámetro proporcionado. `document.getElementsByName("name");`
- **getElementsByClassName()** : método que retorna una colección de todos los elementos con el nombre mismo nombre de clase. Sintaxis: `document.getElementsByClassName ("classname");`
- **querySelector(), querySelectorAll()**: métodos que devuelven el primer elemento o una lista de elementos, que coinciden con un selector CSS específico. Sintaxis: `document.querySelector(CSS selectors); document.querySelectorAll(CSS selectors)`

JavaScript. DOM: acceso directo a nodos

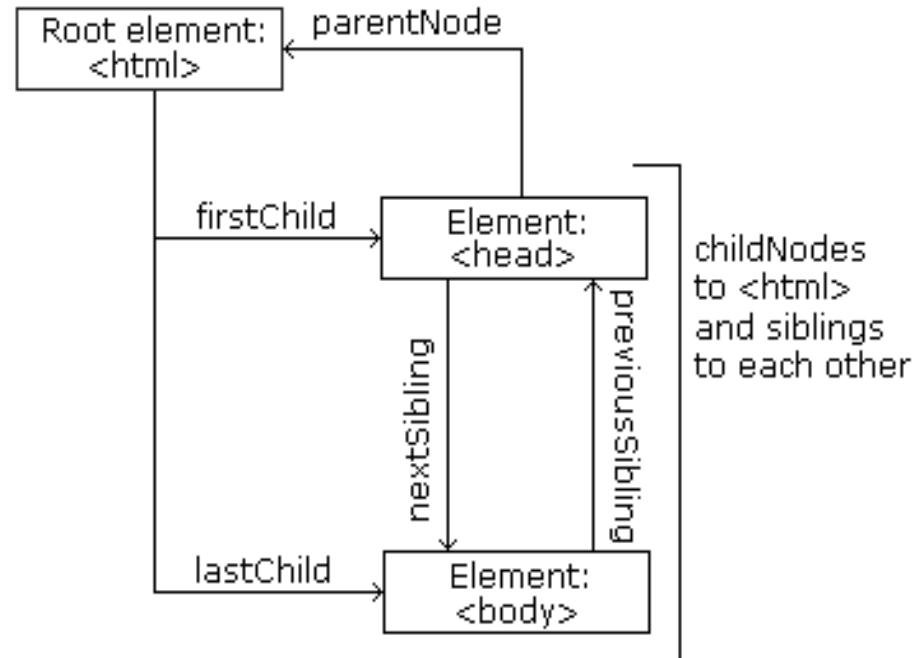
```
var theDiv = document.getElementsByTagName('div')[0];  
var paragraph = theDiv.getElementById('some-id');
```



JavaScript. Recorriendo el DOM

El API DOM proporciona muchas propiedades inherentes a todos los nodos y que permiten acceder a nodos relacionados / cercanos :

- **Node.childNodes**: devuelve un array con todos los hijos del elemento.
- **Node.firstChild**: devuelve el primer elemento del array de nodos hijos. Es lo mismo que acceder de la forma: `Element.childNodes[0]`
- **Node.lastChild**: devuelve el último elemento del array de nodos hijos. También se puede obtener de la forma: `Element.childNodes[Element.childNodes.length-1]`
- **Node.parentNode**: devuelve el nodo padre del elemento actual.
- **Node.nextSibling**: devuelve el siguiente nodo en el mismo nivel dentro del árbol DOM.





JavaScript. Recorriendo el DOM

```
var theDiv = document.getElementsByTagName('div')[0];
```

```
<div>
```

```
    var p = theDiv.firstChild;
```

```
    <p> This is text </p>
```

```
    var ul = p.nextSibling;
```

```
    <ul>
```

```
        <li>Apple</li>    ul.childNodes[0]
```

```
        <li>Pear</li>     ul.childNodes[1]
```

```
        <li>Melon</li>   ul.childNodes[2]
```

```
    </ul>
```

```
</div>
```

Tener en cuenta que si hubiera una línea en blanco entre `` y ``, esto será otro nodo de tipo texto!!!



JavaScript. DOM: creación de nodos

- **Creación de nodos:** añade el nuevo elemento como el último hijo del padre y consta de 4 pasos:
 - Creación de un nodo de tipo Element que represente al elemento. Ejemplo: **var parrafo = document.createElement("p");**
 - Creación de un nodo de tipo Text que represente el contenido del elemento. Ejemplo: **var contenido = document.createTextNode("Hola!");**
 - Añadir el nodo Text como nodo hijo del nodo Element. Ejemplo: **parrafo.appendChild(contenido);**
 - Añadir el nodo Element a la página, en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento. Ejemplo: **document.body.appendChild(parrafo);**



JavaScript. DOM: eliminación y reemplazo de nodos

- **Eliminación de nodos:** para eliminar un nodo, debe conocer quien es el padre del elemento:

```
var parrafo = document.getElementById("p1");
parrafo.parentNode.removeChild(parrafo);
```

- **Reemplazo de nodos:**

```
<div id="div1">
  <p id="p1">Parrafo1</p>
  <p id="p2">Parrafo2</p>
</div>
<script>
  var parrafo = document.createElement("p");
  var nodo = document.createTextNode("Nuevo");
  parrafo.appendChild(nodo);

  var parent = document.getElementById("div1");
  var child = document.getElementById("p1");
  parent.replaceChild(parrafo,child);
</script>
```



JavaScript. DOM: modificación de elementos

MODIFICACION HTML = Cambio de elementos, atributos, estilos, y eventos:

- **Cambio de contenido HTML:** se utiliza la propiedad “innerHTML”
[\(https://www.w3schools.com/jsref/dom_obj_all.asp\)](https://www.w3schools.com/jsref/dom_obj_all.asp)

```
<html>
  <body>
    <p id="p1">Hello World!</p>
    <script>
      document.getElementById("p1").innerHTML="Nuevo texto!";
    </script>
  </body>
</html>
```

- **Cambio de estilos HTML:**
[\(https://www.w3schools.com/jsref/dom_obj_style.asp \)](https://www.w3schools.com/jsref/dom_obj_style.asp)

```
<html>
  <body>
    <p id="p2">Hello world!</p>
    <script>
      document.getElementById("p2").style.color="blue";
    </script>
  </body>
</html>
```



JavaScript. DOM: acceso directo a atributos

- Los atributos HTML de los elementos de la página se transforman automáticamente en propiedades de los nodos.
- Para acceder a su valor, simplemente se indica el nombre del atributo HTML detrás del nombre del nodo.

```
var enlace = document.getElementById("enlace");
alert(enlace.href); // muestra http://www.unsitio.com
<a id="enlace" href="http://www.unsitio.com">Enlace</a>
```



JavaScript. Modelo de eventos DOM HTML

- JavaScript puede utilizar el modelo de programación basada en eventos.
- En este tipo de programación, los scripts responden a la acción del usuario procesando esa información y generando un resultado.
- Ejemplo de eventos son: (http://w3schools.com/jsref/dom_obj_event.asp)
 - Cuando un usuario hace un clic en el ratón (`onclick`)
 - Cuando una página web se ha cargado completamente (`onload`)
 - Cuando el ratón se mueve sobre un elemento (`onmouseover`)
 - Deseleccionar un elemento que se ha modificado (`onchange`)
 - Cuando se envía un formulario HTML (`onsubmit`)

```
<!DOCTYPE html>
<html>
  <body>
    <h1 onclick="this.innerHTML='Ooops!'">Click en el texto!</h1>
  </body>
</html>
```



JavaScript. Manejadores de eventos

- Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado.
- Las funciones o código JavaScript que se definen para cada evento se denominan "manejador de eventos" o “event handlers”.
- Cada elemento HTML define su propia lista de posibles eventos que se le pueden asignar.
- Un mismo tipo de evento (por ejemplo, click con el botón izquierdo del ratón) puede estar definido para varios elementos HTML diferentes y un mismo elemento HTML puede tener asociados varios eventos diferentes.
- El nombre de cada evento se construye mediante el prefijo **on**, seguido del nombre en inglés de la acción asociada al evento.
- Así por ejemplo, el evento de hacer click un elemento con el mouse se denomina **onclick** y el evento asociado a la acción de mover el mouse se denomina **onmousemove**.



JavaScript. Manejadores de eventos

- Para vincular una función de javascript a un evento de objeto existen 3 formas de indicar los manejadores:

1) Manejadores como atributos de los elementos HTML:

```
<body onload="alert('La página se ha cargado completamente');">
```

```
<div id="contenidos" style="border:thin solid silver"  
onmouseover="this.style.borderColor='black';">
```

*La variable especial llamada **this** se crea automáticamente y se utiliza en los eventos para referirse al elemento HTML que ha provocado el evento. No se puede seguir utilizando esta variable en funciones externas.*

2) Manejadores como funciones JavaScript externas:

```
<body onload="funcionMuestraMensaje()">
```

3) Manejadores "semánticos": seleccionar el objeto y agregarle una función de callback a dicho evento.



JavaScript. Manejadores de eventos

3) Manejadores "semánticos":

El ejemplo:

```
<input id="botclick" type="button" value="Click Aquí"  
      onclick="alert('Gracias por pinchar');" />
```

Se puede transformar en:

```
// Función externa  
  
function muestraMensaje() {  
    alert('Gracias por pinchar');  
}  
  
// Asignar la función externa al elemento  
  
document.getElementById("botclick").onclick = muestraMensaje;  
  
// Elemento HTML  
  
<input id="botclick" type="button" value="Click Aquí" />
```

* Nombre de la función SIN paréntesis!!
* Si se añaden los paréntesis, se estaría ejecutando la función y guardando el valor devuelto por la misma en la propiedad **onclick** del elemento.



JavaScript. Manejadores de eventos

3) Manejadores "semánticos":

- Este método requiere que la página se cargue completamente antes de que se utilicen las funciones DOM que asignan los manejadores a los elementos HTML.
- Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento onload:

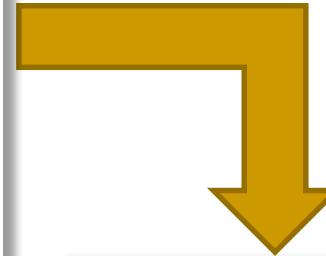
```
window.onload = function() {  
  
    document.getElementById("botclick").onclick =  
        muestraMensaje;  
  
}
```

FUNCION ANONIMA

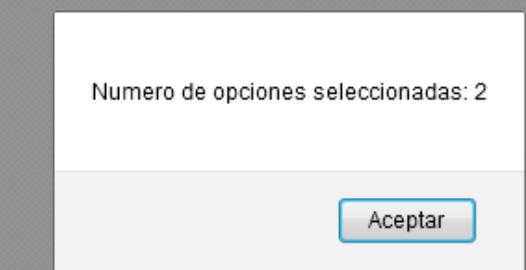
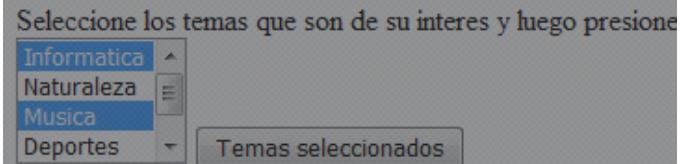


JavaScript. Modelo de eventos DOM HTML

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Prueba Eventos JavaScript</title>
    <script>
        function contar() {
            var seleccionadas = 0;
            var objectSelect=document.getElementById("temas");
            for ( i=0; i < objectSelect.options.length; i++) {
                if (objectSelect.options[i].selected) {
                    seleccionadas++;
                }
            }
            alert('Numero de opciones seleccionadas: '+
                  seleccionadas);
        }
    </script>
</head>
<body>
    <form name="formulario" method="post">
        <label for="doc">Seleccione los temas que son de su
            interes y luego presione el botón</label> <br/>
        <select id="temas" name="temas" multiple="multiple">
            <option selected>Informatica
            <option >Naturaleza
            <option >Musica
            <option >Deportes
            <option >Economía
        </select>
        <input type="button" value="Temas seleccionados"
               onclick="contar();"/>
    </form>
</body>
</html>
```



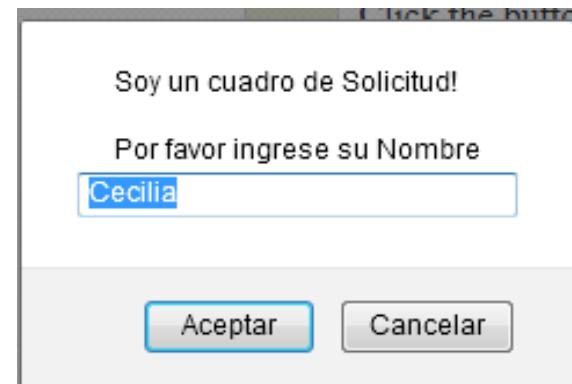
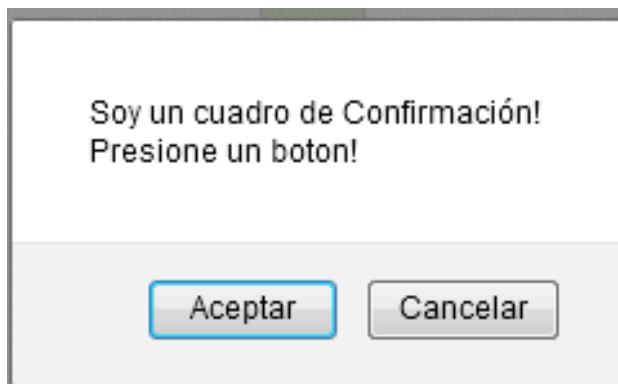
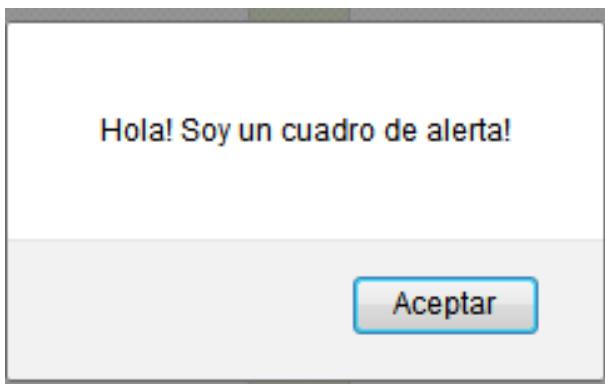
Salida en el Navegador





JavaScript. Ventanas emergentes

- **CUADRO DE ALERTA:** se utiliza para brindar información al usuario.
Sintaxis: `window.alert("texto");`
- **CUADRO DE CONFIRMACIÓN:** se utiliza cuando el usuario debe verificar o aceptar algo. Sintaxis: `window.confirm("texto");`
- **CUADRO DE SOLICITUD:** se utiliza cuando el usuario debe ingresar algún valor. Sintaxis: `window.prompt("texto","valorpordefecto");`





JavaScript. Buenas Prácticas

BUENAS PRACTICAS

- Evitar el uso de **Variables Globales**
- Declarar siempre las **Variables Locales** usadas dentro de una Función, mediante la palabra clave **var**
- Colocar todas las declaraciones de variables en la parte superior de cada script o función, e inicializarlas

```
// Declarar al inicio del script
```

```
var nombre = "", apellido = "", precio = 0, vArray = [],  
vObjecto = {};
```

```
// Declarar al inicio del script
```

```
var i
```

```
// Usar mas tarde
```

```
for (i = 0; i < 5; i++) {...}
```

- Tener cuidado con las conversiones automáticas de tipo. Tener en cuenta que los números pueden convertirse accidentalmente en cadenas o NaN (Not a Number)
- Usar el operador “`==`” para comparación. El operador “`==`” siempre convierte (a tipos coincidentes) antes de la comparación. En cambio el operador “`===`” obliga a comparar los valores y el tipo



JavaScript. Buenas Prácticas

BUENAS PRACTICAS

- Siempre finalizar una instrucción **switch** con un valor predeterminado (**default**)
- Tener en cuenta que en JavaScript no se crea un nuevo ámbito de variables para cada bloque de código (if, for, etc), es decir que una variable definida dentro de un bloque sigue existiendo fuera del mismo.
- Reducir la actividad en bucles. Cada instrucción en un bucle, incluyendo la instrucción **for**, se ejecuta para cada iteración. Colocar por fuera del bucle, todas las declaraciones o asignaciones que se puedan, para que el bucle se ejecute más rápido.

```
var i;  
var l = arr.length;  
for ( i = 0; i < l; i++ ) {...}
```

- Reducir acceso al DOM. Si se espera acceder a un mismo elemento DOM varias veces, guardar el resultado en una variable local y luego hacer referencia a la misma

JavaScript. Actividad 3.1 – Ejercitación



Dado el siguiente doc. HTML, crear un manejador de eventos JavaScript para que cuando se haga un clic en el Botón Aceptar, muestre los datos ingresados en el formulario dentro de la etiqueta `<div id="info">` y luego deshabilite dicho botón.

```
<!DOCTYPE html>
<html>
<body>
<form>
<label for="nombre">Nombre:</label>
<input type="text" id="nombre"/> <br>
<label for="dni">DNI:</label>
<input type="number" id="dni"/> <br>
<input type="button" id="boton1" value="Aceptar">
</form>
<div id="info"></div>
</body>
</html>
```



JavaScript. Validación de Datos de Formularios

- La validación de datos es el proceso que asegura que los datos ingresados mediante los campos de entrada de un formulario, están limpios, correctos y son útiles para la aplicación.
- Tareas de validación típicas son:
 - Se han completado todos los campos obligatorios?
 - Se ha ingresado una fecha válida?
 - Se ha ingresado texto en un campo numérico?
- La validación puede ser definida por varios métodos diferentes, e implementado de diferentes maneras:
 - La validación del **lado del servidor** se realiza mediante un servidor web, **después** de que la entrada ha sido enviado al servidor.
 - La validación del **lado del cliente** se realiza mediante un navegador web, **antes** de enviar la entrada a un servidor web.



JavaScript. Validación de Datos desde el Cliente

FORMAS DE VALIDAR DATOS DE FORMULARIOS DEL LADO DEL CLIENTE:

- 1)** Validación de Formularios mediante JavaScript.
- 2)** Validación de Formularios automática mediante el Navegador y el nuevo concepto de validación de HTML5 denominado “validación de restricciones”, el cual se basa en:
 - 2.1)** Validación de restricción de atributos de la etiqueta <input> de HTML5.
 - 2.2)** Validación de restricción de pseudo selectores CSS.
 - 2.3)** propiedades DOM del API de Validación de restricción de HTML5.
 - 2.4)** Métodos DOM del API de Validación de restricción de HTML5.

<https://html.spec.whatwg.org/#barred-from-constraint-validation>

<http://www.html5rocks.com/en/tutorials/forms/constraintvalidation/>



JavaScript. Validación de Datos desde el Cliente

1) VALIDACIÓN DE FORMULARIOS MEDIANTE JAVASCRIPT.

- Antes de enviar un formulario al servidor, se pueden validar mediante JavaScript, los datos insertados por el usuario.
- Normalmente, la validación de un formulario consiste en llamar a una función de validación cuando el usuario pulsa sobre el botón de envío del formulario.

```
<form ... onsubmit="return validacion()"> ... </form>
```

```
function validacion() {  
    // Validar un campo de texto obligatorio  
    var valor = document.getElementById("campo").value;  
    if( valor == null || valor.length == 0 ) {  
        alert('[ERROR] El campo debe tener un valor de...');  
        return false;  
    }  
    ....  
    return true;  
}
```



JavaScript. Validación de Datos desde el Cliente

2) VALIDACIÓN DE FORMULARIOS AUTOMÁTICA MEDIANTE EL NAVEGADOR.

2.1) Validación de restricciones de atributos de la etiqueta <input> de HTML.

Atributo	Descripción
disabled	Especifica que el elemento input debe estar deshabilitado
max	Especifica el máximo valor de un elemento input
min	Especifica el mínimo valor de un elemento input
pattern	Especifica el patrón del valor de un elemento input
required	Especifica que el campo input requiere un valor
type	Especifica el tipo de un elemento input



JavaScript. Validación de Datos desde el Cliente

2) VALIDACIÓN DE FORMULARIOS AUTOMÁTICA MEDIANTE EL NAVEGADOR.

2.1) Validación de restricciones de atributos de la etiqueta <input> de HTML.

Atributo Pattern: permite especificar una expresión que un campo determinado debe coincidir. Algunos patrones útiles:

- **Patrón de URL:** `input type="url" pattern="https?://.+"`

- **Patrón de dirección IPv4:**

`input type="text" pattern="\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"`

- **Fecha(dd/mm/yyyy or mm/dd/yyyy):**

`input type="text" pattern="\d{1,2}/\d{1,2}/\d{4}"`

- **Precio:** `input type="text" pattern="\d+(\.\d{2})?"`

- **Latitud/Longitud:** `input type="text" pattern="-?\d{1,3}(\.\d+)"`

JavaScript. Validación de Datos desde el Cliente

2) VALIDACIÓN DE FORMULARIOS AUTOMÁTICA MEDIANTE EL NAVEGADOR.

2.1) Validación de restricciones de atributos de la etiqueta <input> de HTML.

Atributo Required: indica que el elemento es obligatorio. Ejemplo:

- <input title="Se necesita un nombre" type="text" name="nombre" required/>

Nombre:

The screenshot shows a web page with a text input field labeled "Nombre:". The input field is highlighted with a red border, indicating it is invalid. Below the input field is a tooltip-style message box with a yellow exclamation mark icon. The message reads "Completa este campo" and "Se necesita un nombre". A blue arrow points from the word "required" in the list item above to the red border of the input field.



JavaScript. Validación de Datos desde el Cliente

2) VALIDACIÓN DE FORMULARIOS AUTOMÁTICA MEDIANTE EL NAVEGADOR.

2.2) Validación de restricciones de Pseudo Selectores CSS.

Selector	Descripción
:disabled	Selecciona los elementos de entrada que tienen especificado el atributo "disabled"
:invalid	Selecciona los elementos de entrada con valores inválidos
:optional	Selecciona los elementos de entrada que no tienen especificado el atributo "required"
:required	Selecciona los elementos de entrada que poseen el atributo "required"
:valid	Selecciona los elementos de entrada que poseen valores válidos



JavaScript. Validación de Datos desde el Cliente

2) VALIDACIÓN DE FORMULARIOS AUTOMÁTICA MEDIANTE EL NAVEGADOR.

2.3) Propiedades DOM del API de Validación de restricciones HTML5.

Propiedad	Descripción
validity	Objeto que contiene propiedades booleanas relacionadas a la validez de un elemento de entrada.
validationMessage	Contiene el texto del mensaje que el navegador mostrará cuando una validación resulte falsa.
willValidate	Indica si un elemento <input> debe ser validado.



JavaScript. Validación de Datos desde el Cliente

2.3.1) La Propiedades *Validity*, de un elemento <input> contiene un numero de propiedades relacionadas a la validez del dato:

Propiedad	Descripción
customError	Se establece en true, si se establece un mensaje de validez personalizado.
patternMismatch	Se establece en true, si el valor de un elemento no coincide con su atributo “pattern”.
rangeOverflow	Se establece en true, si el valor de un elemento es mayor que su atributo “max”.
rangeUnderflow	Se establece en true, si el valor de un elemento es menor que su atributo “min”.
stepMismatch	Se establece en true, si el valor de un elemento no es válido de acuerdo a su atributo “step”.
tooLong	Se establece en true, si el valor de un elemento excede su atributo “maxLength”.
typeMismatch	Se establece en true, si el valor del elemento es inválido de acuerdo a su atributo “type”.



JavaScript. Validación de Datos desde el Cliente

2.3.1) Propiedades **Validity**:

Propiedad	Descripción
valueMissing	Se establece en true, si un elemento con un atributo "required", no tiene ningún valor.
valid	Se establece en true, si el valor de un elemento es válido.

```
<input id="id1" type="number" max="100">
<button onclick="myFunction()">OK</button>
<p id="demo"></p>
<script>
    function myFunction() {
        var txt = "";
        if (document.getElementById("id1").validity.rangeOverflow)
            {txt = "Valor demasiado alto";}
        document.getElementById("demo").innerHTML = txt;}
</script>
```





JavaScript. Validación de Datos desde el Cliente

2.3.1) Propiedades *Validity*:

Screenshot of a browser developer tools console showing the `ValidityState` object properties for a numeric input field.

The console output shows:

```

true
▼ ValidityState {valid: false, customError: false, badInput: true, stepMismatch: false, rangeOverflow: false...}
  badInput: true
  customError: false
  patternMismatch: false
  rangeOverflow: false
  rangeUnderflow: false
  stepMismatch: false
  tooLong: false
  tooShort: false
  typeMismatch: false
  valid: false
  valueMissing: false
▶ __proto__: ValidityState

```

The page displays a form element with the following attributes:

Ingrese un numero
de 0 a 10 OK?

ENVIAR

The browser's status bar indicates "Select an element in the page to inspect it."

The status bar also shows the following ValidityState object properties:

```

▼ ValidityState {valid: false, customE...
  badInput: false
  customError: false
  patternMismatch: false
  rangeOverflow: true
  rangeUnderflow: false
  stepMismatch: false
  tooLong: false
  tooShort: false
  typeMismatch: false
  valid: false
  valueMissing: false
▶ __proto__: ValidityState

```



JavaScript. Validación de Datos desde el Cliente

2) VALIDACIÓN DE FORMULARIOS AUTOMÁTICA MEDIANTE EL NAVEGADOR.

2.4) Métodos DOM del API de Validación de restricciones de HTML5.

Propiedad	Descripción
checkValidity()	Retorna true si un elemento de entrada contiene datos válidos.
setCustomValidity()	Establece el valor de la propiedad “validationMessage” de un elemento de entrada.

```

<input id="id1" type="number" min="100" max="300">
<button onclick="myFunction () ">OK</button>
<p id="demo"></p>
<script>
    function myFunction() {
        var inpObj = document.getElementById("id1");
        if (inpObj.checkValidity() == false) {
            document.getElementById("demo").innerHTML =
                inpObj.validationMessage; } }
</script>

```





JavaScript. Validación de Datos desde el Cliente

2) VALIDACIÓN DE FORMULARIOS AUTOMÁTICA MEDIANTE EL NAVEGADOR.

2.4) Métodos DOM del API de Validación de restricciones de HTML5.

Ejemplo Validación personalizada

```
1. <label>Email:</label>
2. <input type="email" id="email_addr" name="email_addr">
3. <label>Repeat Email Address:</label>
4. <input type="email" id="email_addr_repeat"
   name="email_addr_repeat" oninput="check(this)">

5. <script>
6. function check(input) {
7.   if (input.value !=
     document.getElementById('email_addr').value) {
8.     input.setCustomValidity('The two email addresses must
      match.');
9.   } else {
10.    // input is valid -- reset the error message
11.    input.setCustomValidity('');
12.  }
13.}
14.</script>
```



JavaScript. Manejo de Errores

- Cuando se produce un error, el motor JavaScript normalmente se detiene y genera un mensaje de error. JavaScript generará (**throw**) un error.
- La sentencia **try** permite definir un bloque de código para probar errores mientras se ejecuta.
- La declaración **catch** permite definir un bloque de código que se ejecutará, si se produce un error en el bloque **try**.

```
var txt="";
function message() {
    try {
        adddlert("Welcome guest!");
    } catch(err) {
        txt="Se produjo un error.\n\n";
        txt+="Descripción: " + err.message + "\n\n";
        txt+="Click OK para continuar.\n\n";
        alert(txt); }
}
```

```
<body>
<input type="button"
       value="Ver Msj"
       onclick="message()">
</body>
```



Ej. 3



JavaScript. Manejo de Errores

- La sentencia **throw** permite crear un error personalizado.
- Ejemplo que examina el valor de un campo de entrada. Si el valor es incorrecto, se lanza una excepción (error).

```
<script>
function myFunction() {
    try {
        var x=document.getElementById("demo").value;
        if(x=="") throw "vacio";
        if(isNaN(x)) throw "no es un numero";
        if(x>10)   throw "muy alto";
        if(x<5)   throw "muy bajo";
    } catch(err) {
        var y=document.getElementById("mess");
        y.innerHTML="Error: " + err + ".";
    }
}
</script>
```



```
<p>Ingrese un numero entre 5 y 10:</p>
<input id="demo" type="text">
<button type="button"
onclick="myFunction()">Prueba</button>
<p id="mess"></p>
```

JavaScript. Actividad 3.2 – Ejercitación



Dado el siguiente doc. HTML, crear una función JavaScript que interactúe como sigue: Cuando se seleccione la “opcion1” del Grupo de Opciones 1, se debe habilitar el Grupo de opciones 2 (por defecto debe tener el valor “Opcion1.1”), caso contrario, deshabilitar el Grupo de opciones2 y habilitar el “Campo1”, establecer su valor en vacío y colocar el foco en el mismo.

```
<!DOCTYPE html>
<html> <head> <style> body {margin:5%} form {width: 500px;}
        #izq {float: left} #der {float: right} </style> </head>
<body>
<h2>Formulario con diferentes opciones</h2>
<form>
<div id="izq"> <label>Grupo de Opciones 1</label>
<select id="opciones" onChange="activarDesactivar(this)">
<option value="op1" selected>Opcion1</option>
<option value="op2">Opcion2</option>
<option value="op3">Opcion3</option>
<option value="op4">Opcion4</option>
</select> </div>
<div id="der"> <label>Grupo de Opciones 2</label>
<select id="opciones2">
<option value="op1">Opcion1.1</option>
<option value="op2">Opcion1.2</option>
</select> <br><br>
<label>Campo1</label> <input type="text" id="campo1" disabled> </div>
</form> </body> </html>
```



JavaScript. Actividad 3.2 – Continuación



Vistas de ejemplo según las opciones seleccionadas:

Formulario con diferentes opciones

Grupo de Opciones 1

Grupo de Opciones 2

Campo1

Formulario con diferentes opciones

Grupo de Opciones 1

Grupo de Opciones 2

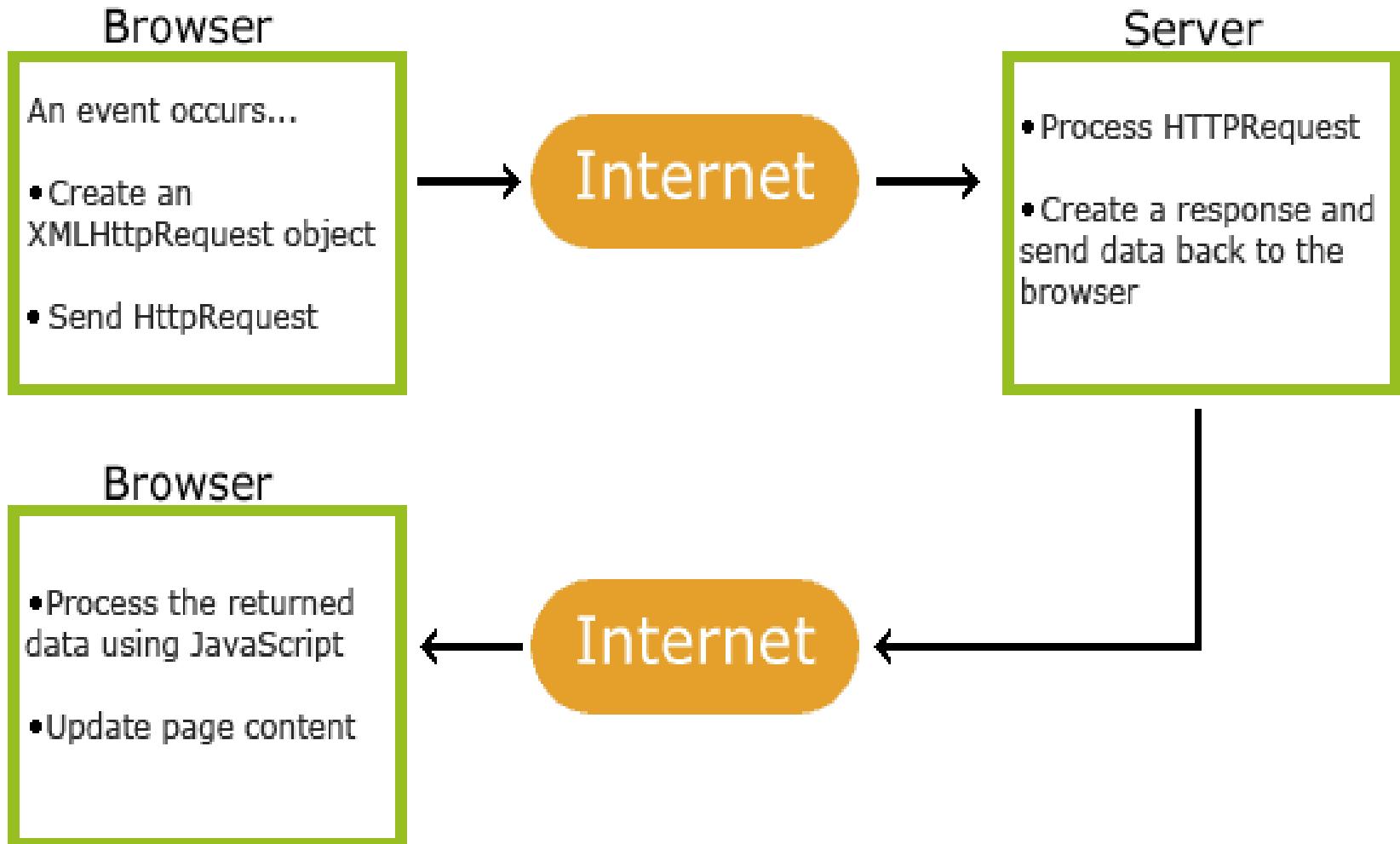
Campo1



JavaScript. Ajax

- **AJAX** significa solamente ***Asynchronous Javascript y XML***.
- Permite que las páginas web se actualicen de forma asíncrona mediante el intercambio de pequeñas porciones de datos con el servidor => se actualizan partes de la página web, sin volver a cargar toda la página.
- Esta capacidad de actualización parcial nos permite alejarnos del paradigma de **request-response**, propio de la programación web tradicional.
- AJAX se basa en estándares de Internet, y utiliza una combinación de:
 - **Objeto XMLHttpRequest** (para intercambiar datos de forma asíncrona con el servidor y el cual es soportado por los navegadores modernos)
 - **JavaScript / DOM** (para mostrar / interactuar con la información)
 - **CSS** (estilos de los datos)
 - **XML** (utilizado a menudo como el formato de transferencia de datos)

JavaScript. Ajax





JavaScript. Ajax: Uso del objeto XMLHttpRequest

■ CREACIÓN DEL OBJETO XMLHttpRequest.

Sintaxis: variable = new XMLHttpRequest();

■ ENVÍO DE UNA SOLICITUD AL SERVIDOR WEB.

Se deben utilizar los métodos **open()** y **send()** del objeto XMLHttpRequest.

Ejemplo:

Variable del tipo
XMLHttpRequest

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Método	Descripción
open(<i>method, url, async</i>)	method: especifica el tipo de solicitud (GET o POST), url: la ubicación del archivo en el servidor y async: si la solicitud debe ser manejada de forma asíncrona o no (true o false).
send()	Envía la solicitud al servidor (utilizada para GET)
send(<i>string</i>)	Envía la solicitud al servidor (utilizada para POST)



JavaScript. Ajax: Uso del objeto XMLHttpRequest

■ EJEMPLO DE SOLICITUD GET CON ENVÍO DE PARÁMETROS:

```
xhttp.open("GET","demo_get2.jsp?fname=Henry&lname=Ford",true);  
xhttp.send();
```

■ EJEMPLO DE SOLICITUD POST CON ENVÍO DE PARÁMETROS.

```
xhttp.open("POST","ajax_test.jsp",true);  
xhttp.send("fname=Henry&lname=Ford");
```



JavaScript. Ajax: Uso del objeto XMLHttpRequest

■ RESPUESTA DEL SERVIDOR

○ PROPIEDADES DEL OBJETO XMLHttpRequest

Propiedad	Descripción
responseText	obtiene los datos de respuesta como un String de JavaScript
responseXML	obtiene los datos de respuesta en formato XML

○ MÉTODOS DEL OBJETO XMLHttpRequest

Propiedad	Descripción
getResponseHeader('encabezado')	Devuelve información de un encabezado específico del recurso de servidor
getAllResponseHeaders()	Devuelve la información de todos los encabezados del recurso del servidor



JavaScript. Ajax: Uso del objeto XMLHttpRequest

■ PROCESAMIENTO ASÍNCRONO

- Con AJAX, el código JavaScript no tiene que esperar la respuesta del servidor (asíncrono), mientras puede ejecutar otros scripts a la espera de respuesta del servidor y tratar la misma cuando esté disponible.
- Para tratar la respuesta se deben utilizar las siguientes propiedades del objeto XMLHttpRequest:

Propiedad	Descripción
onreadystatechange	Define una función que se ejecutará automáticamente cada vez que el atributo readyState cambie
readyState	Mantiene el estado del XMLHttpRequest. Toma valores de 0 a 4: 0 : solicitud no inicializada; 1 : conexión de servidor establecida 2 : solicitud recibida; 3 : solicitud en proceso 4 : Solicitud finalizada y respuesta lista
status	Contiene el estado de la solicitud HTTP 200 : "OK" 403 : "Forbidden" 404 : "Page not found"



JavaScript. Ajax: Uso del objeto XMLHttpRequest

■ PROCESAMIENTO ASÍNCRONO

Cuando la propiedad **readyState** es igual 4 y el **status** es 200, la respuesta está lista:

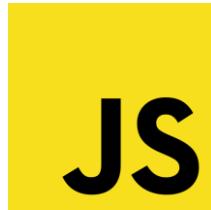
```
function loadDoc() {  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            document.getElementById("myDiv").innerHTML =  
                this.responseText;  
        }  
    };  
    xhttp.open("GET", "ajax_info.txt", true);  
    xhttp.send();  
}
```

Nota: El evento **onreadystatechange** se dispara cinco veces (0-4), un tiempo para cada cambio en **readyState**.

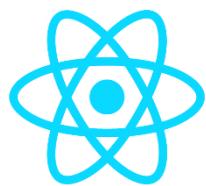
Ver: https://www.w3schools.com/js/tryit.asp?filename=tryjs_ajax_first



JavaScript. Librerías y frameworks



¡ Existen más de 1700 frameworks y librerías de Javascript
a la actualidad !



React



Vue.js



JavaScript. Librerías y frameworks

Para hacer frente a dificultades y procesamientos complejos en el cliente, se han desarrollado una gran cantidad de librerías y Frameworks de JavaScript. Algunos de ellos son:

- **JQUERY:** es el framework de JavaScript más popular en Internet en la actualidad. Provee manipulación de HTML/DOM y CSS, métodos de eventos HTML, efectos y animaciones, AJAX. Ver tutorial: <http://www.w3schools.com/jquery/default.asp>
- **ANGULAR.JS:** es un framework MVW (Model View Whatever) de Google que se ha encargado de traer orden a las aplicaciones JavaScript y potenciar las Arquitecturas SPA (Single Page Applications).
- **REACT.JS:** es la librería de Facebook orientada a la gestión de UI. Usa los conceptos de DataFlows y programación Reactiva para simplificar el proceso de actualización de la vista.



JavaScript. Librerías y frameworks

- **METEOR.JS:** uno de los frameworks JavaScript que está adquiriendo mayor tracción en el mercado y que apoya el concepto de aplicaciones JavaScript Isomórficas, aquellas aplicaciones que pueden ejecutar su código JavaScript en cliente y servidor.
- **MOMENT.JS:** la librería de Javascript orientada a la gestión de fechas, con ella el manejo de variables temporales se ha convertido en algo mucho más sencillo.
- **FOUNDATION.JS:** Uno de los frameworks JavaScript más avanzados del mercado a la hora de diseñar soluciones responsive. Su fuerte uso de Media Queries y un enfoque Mobile First le han catapultado a los primeros puestos.
- Entre otros...



JavaScript. Angular: Definición

¿Qué es ANGULAR?

- Es un **framework** del lado del Cliente, esto quiere decir que sirve para crear aplicaciones que funcionan en un navegador web y que los datos se solicitan al servidor.
- Permite crear Aplicaciones Web de una Sola Página (**SPA**, Single Page Applications)
- Es creado y mantenido por **Google** y es uno de los frameworks del lado del Cliente más populares de Javascript, lo que influye en que su comunidad y ecosistema sean gigantesco.

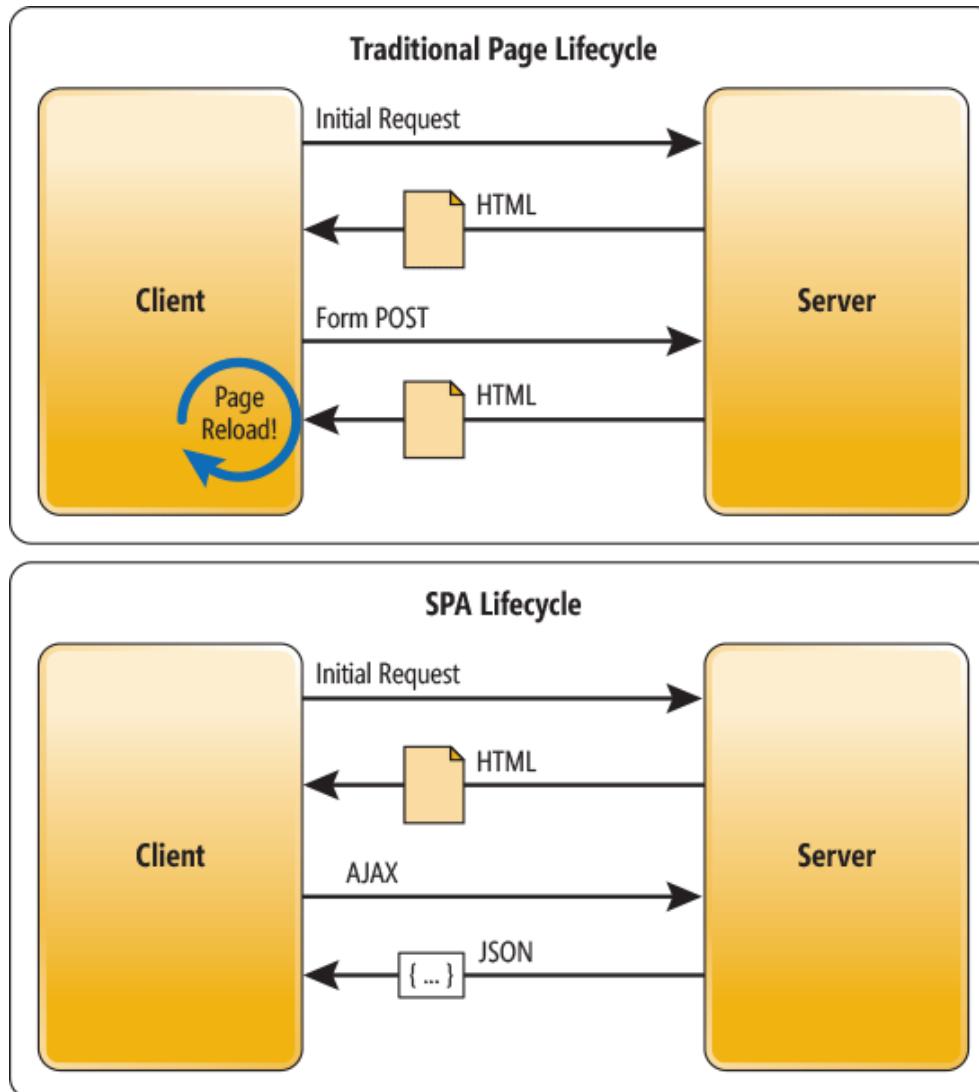
JavaScript. Angular: Usos

Creación de Aplicaciones Web de una Sola Página (**SPA**)

- La carga de datos es dinámica, asíncrona y sin refrescar la pantalla.
- Permite la creación de aplicaciones reactivas que no sobrecarguen el navegador.
- Separación del código en frontend y backend, según las últimas tendencias del desarrollo web.



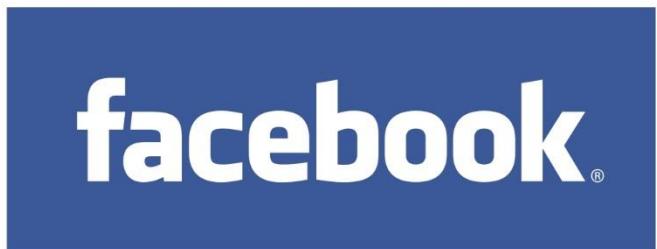
JavaScript. Angular: App Cliente servidor VS App SPA





JavaScript. Angular: Aplicaciones SPA

The screenshot shows the Gmail settings page at <https://mail.google.com/mail/u/1/#settings/accounts>. The 'Accounts and Import' tab is highlighted with a green circle containing the number 2. A green circle containing the number 1 is on the top right of the settings header. A green circle containing the number 3 is on the 'Import mail and contacts' link.



The image shows a desktop computer monitor and a smartphone both displaying the Facebook news feed. The desktop screen shows a detailed view of a post from 'Josephine Hazeleigh' about a camping trip, with comments and likes visible. The smartphone screen shows a similar feed with posts from various users, including one from 'Guillermo Moreno'.



JavaScript. Angular: Ventajas y características

VELOCIDAD Y RENDIMIENTO

Las aplicaciones angulares se cargan rápidamente con el Enrutador de componentes, que ofrece división automática de códigos para que los usuarios solo carguen el código requerido para procesar la vista que solicitan.

PRODUCTIVIDAD

Cree rápidamente vistas de IU con sintaxis de plantilla simple y potente.

Cree animación complejas y de alto rendimiento con muy poco código a través de la API intuitiva de Angular.

DESARROLLAR EN TODAS LAS PLATAFORMAS

Angular permite crear aplicaciones reutilizando el código, sin importar el destino: web, web móvil, dispositivo móvil nativo y escritorio nativo.





JavaScript. Angular: Arquitectura

Componentes

Es una clase que define cómo se comporta una porción de nuestro sistema.

Plantillas

Son archivos .html que definen la visualización de los elementos de la aplicación.

Decoradores

Permiten modificar y configurar dinámicamente atributos de las clases y componentes.

Metadatos

Dan información acerca de las clases y detallan las relaciones. Si tenemos, por ejemplo, un componente y una plantilla, el metadato se encargará de decirle a Angular que ese componente y esa plantilla deben ir juntos.



JavaScript. Angular: Arquitectura

Servicios

Es una clase que permite y facilita la reutilización de código.

Providers

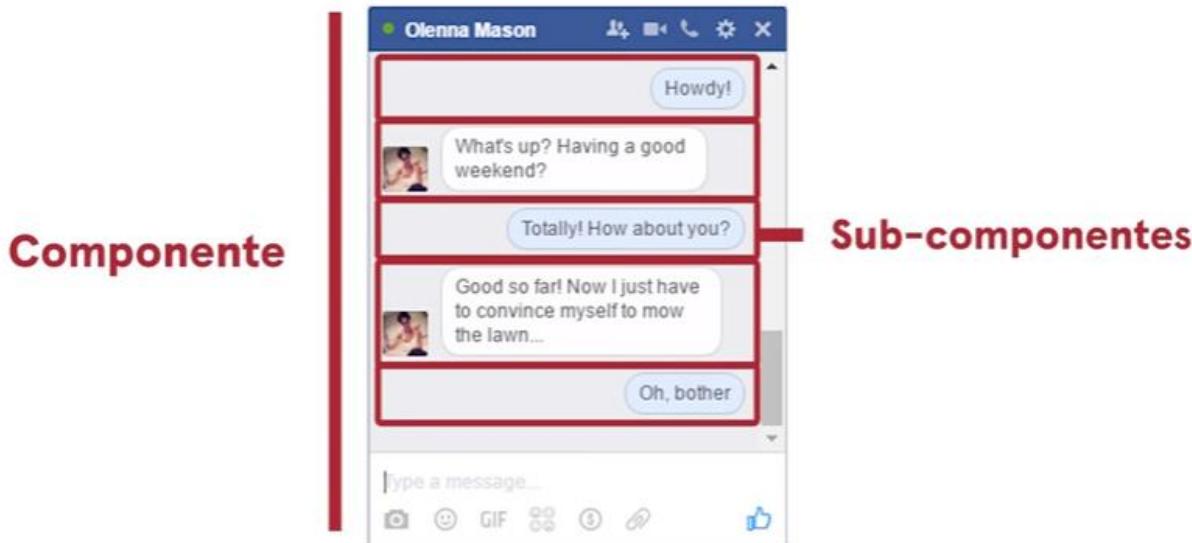
Son servicios que nos proveen de datos o funcionalidades a través de sus métodos. Existen pares provider/servicio que son propios de Angular aunque también podemos desarrollar los nuestros.

Directiva

Básicamente son nuevos atributos que se aplican en nuestro código HTML. Estas directivas se transforman en funcionalidades que actúan directamente sobre el DOM: por ejemplo, mostrar u ocultar un elemento <div>.

JavaScript. Angular: Componente

Es una clase que define cómo se comporta una porción de nuestro sistema. Básicamente, nuestra aplicación Angular será una sumatoria de componentes que se comunican entre sí.



Cada componente tendrá una **vista** por medio de html y **funciones** que definen su forma de actuar frente a eventos.



JavaScript. Angular: Ejemplo de código de Componente

Clase

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app works!';
}
```



JavaScript. Angular: Desarrollos y ejemplos

www.madewithangular.com/

<https://www.w3schools.com/angular/default.asp>

https://www.w3schools.com/angular/tryit.asp?filename=try_ng_app5