



# *Frameworks Frontend*



## Desarrollo de Aplicaciones Web: JavaScript, jQuery, Vue y Angular

Carlos Delgado Hita  
Departamento de Ciencias de la Computación  
Universidad de Alcalá (Madrid)  
[carlos.delgado@uah.es](mailto:carlos.delgado@uah.es)



# Índice

---

- Introducción y Contexto.
- HTML.
- HTTP.
- HTML DOM.
- JavaScript.
- Selección, Filtros, Eventos y Efectos con jQuery.
- Comunicación con el Servidor.
- Ajax con jQuery.
- JSON.
- Fundamentos de Vue.js
- Fundamentos de Angular



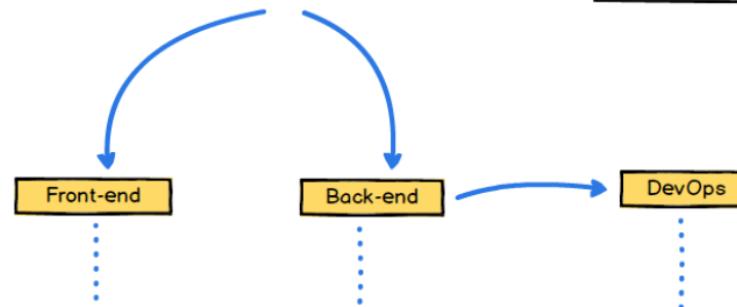
# Contexto: Hoja de Ruta del Desarrollador Web (I)

Required for any path

- Git - Version Control
- Basic Terminal Usage
- Data Structures & Algorithms
- Github
- Licenses
- Semantic Versioning
- SSH
- HTTP/HTTPS and APIs
- Design Patterns
- Character Encodings

## Web Developer in 2021

Choose your path



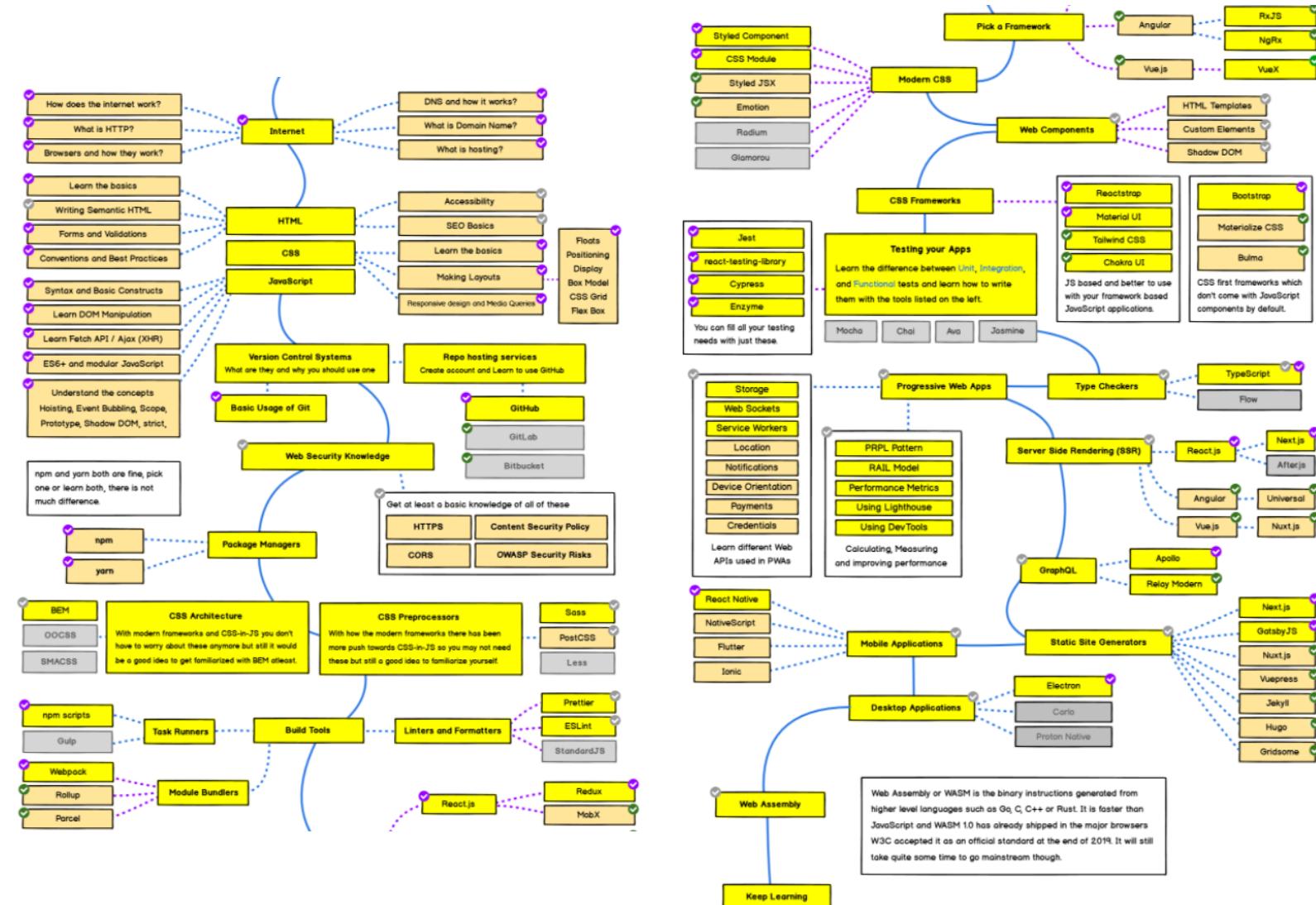
Find the detailed version of this roadmap along with resources and other roadmaps

<http://roadmap.sh>

Fuente: <https://github.com/kamranahmedse/developer-roadmap>



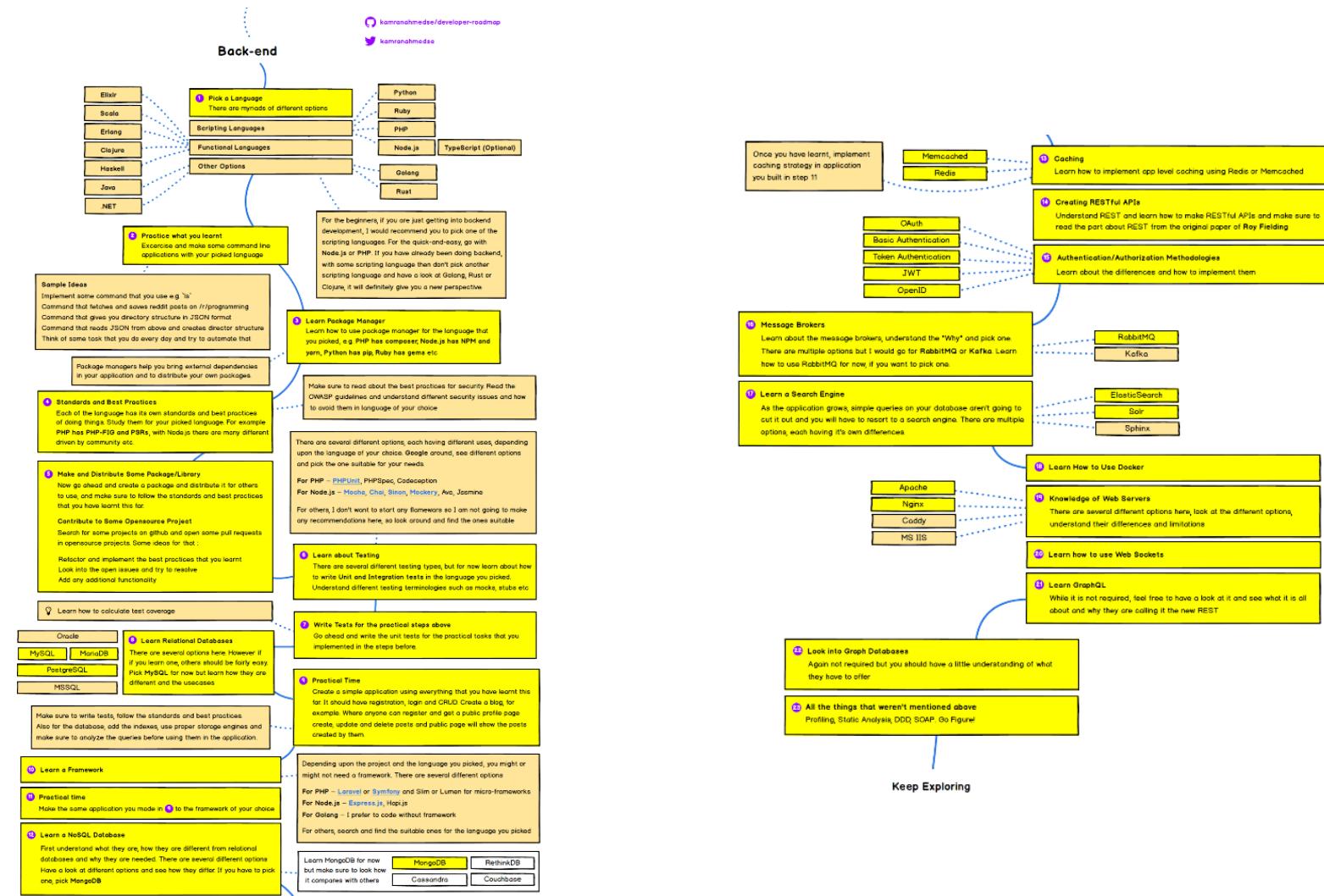
# Contexto: Hoja de Ruta del Desarrollador Web (II)



Fuente: <https://github.com/kamranahmedse/developer-roadmap>



# Contexto: Hoja de Ruta del Desarrollador Web (III)

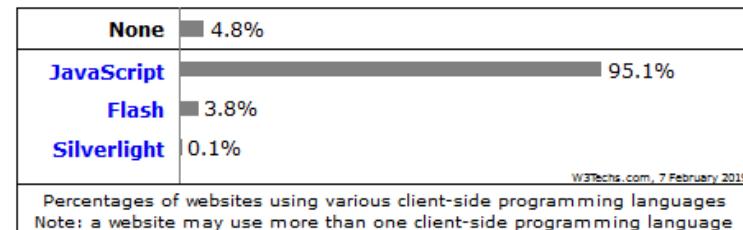
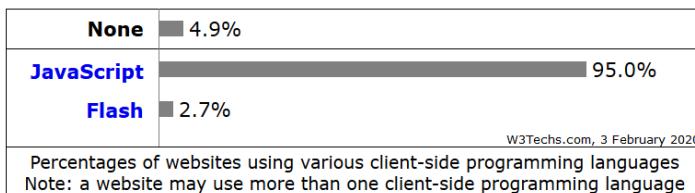
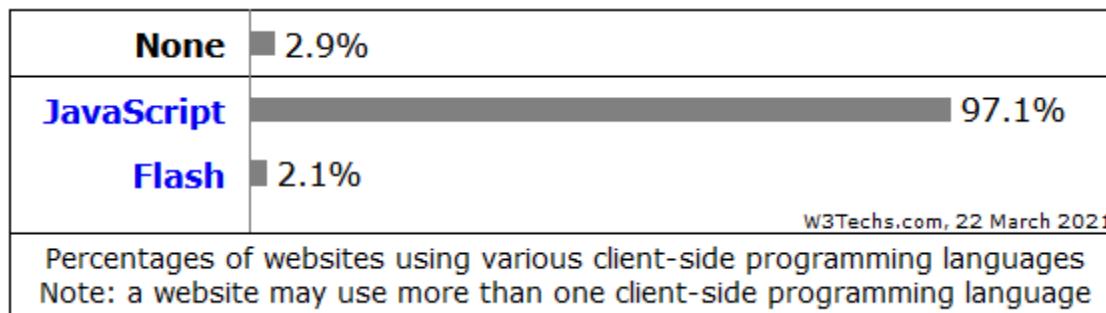


Fuente: <https://github.com/kamranahmedse/developer-roadmap>



# Contexto: Lenguajes de Cliente

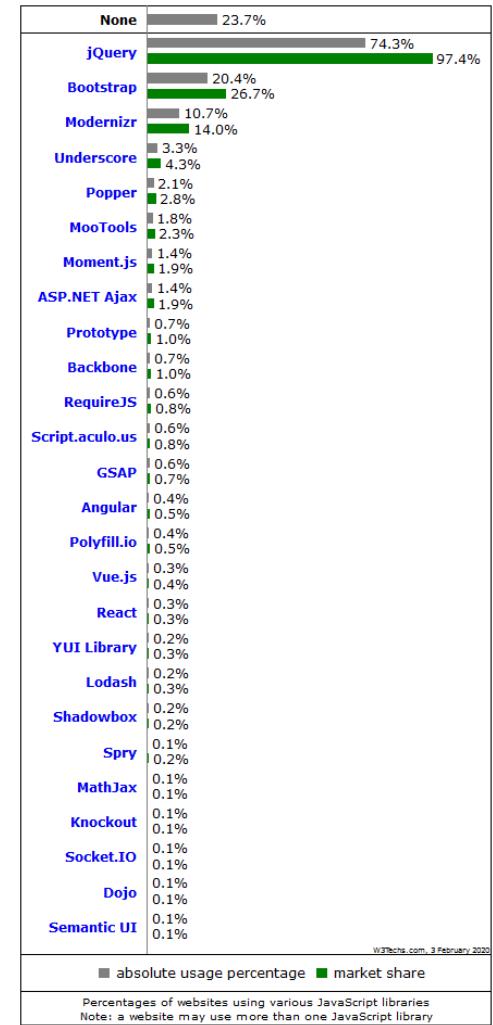
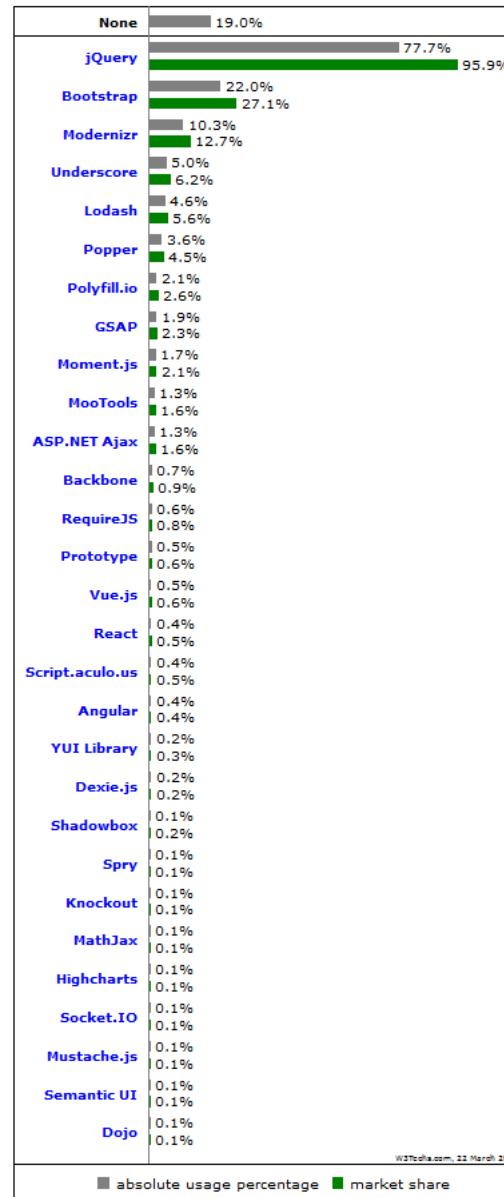
## Estadística de uso de lenguajes de programación del lado del cliente





# Contexto: Lenguajes de Cliente

## Estadística de uso de librerías JS

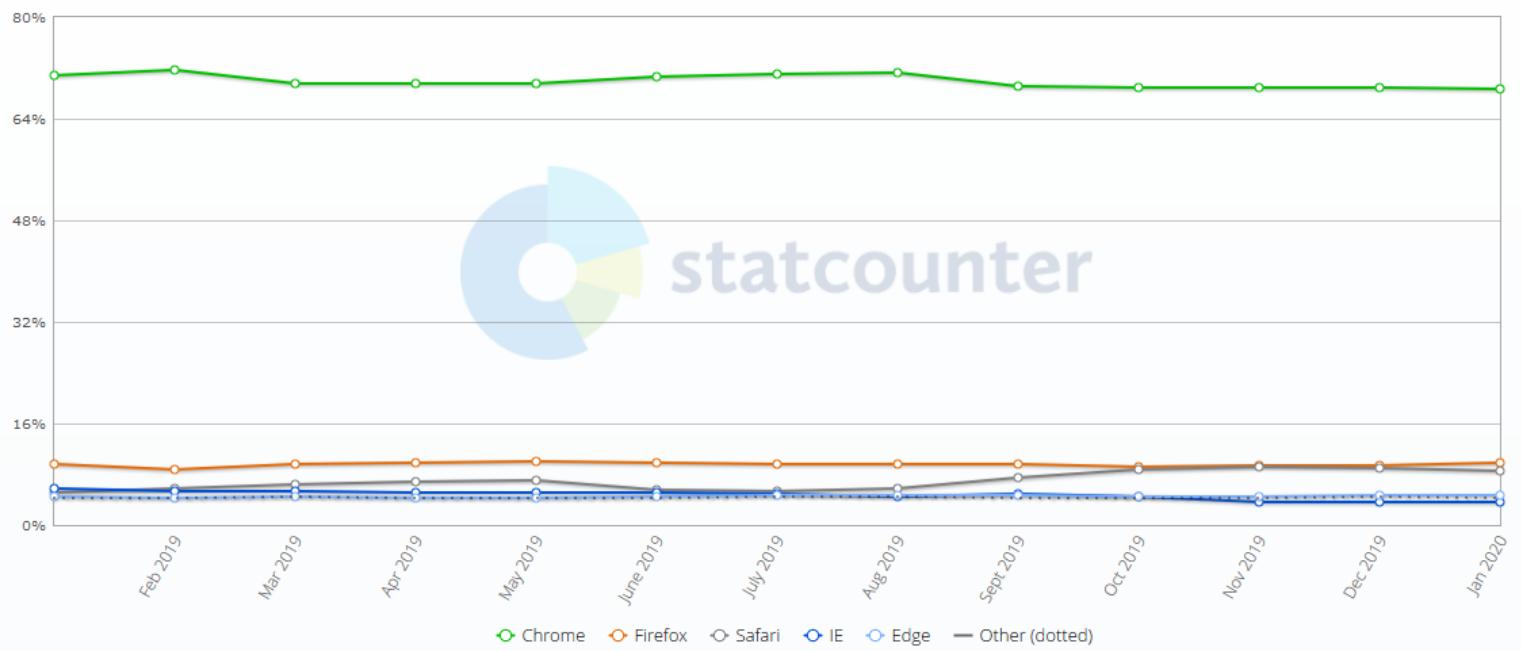




# Contexto: Navegadores (evolución)

Desktop Browser Market Share Worldwide  
Jan 2019 - Jan 2020

Edit Chart Data



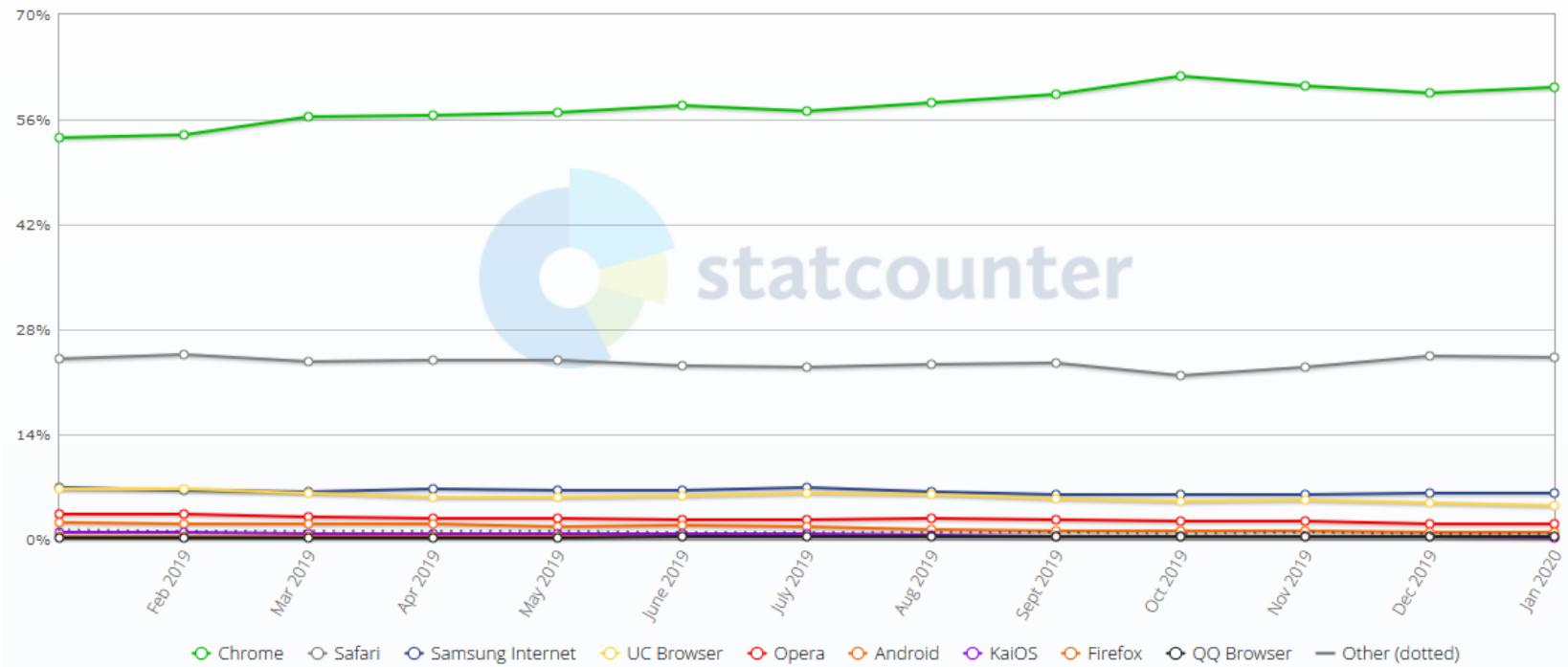
Fuente



# Contexto: Navegadores (evolución)

Mobile, Tablet & Console Browser Market Share Worldwide  
Jan 2019 - Jan 2020

Edit Chart Data



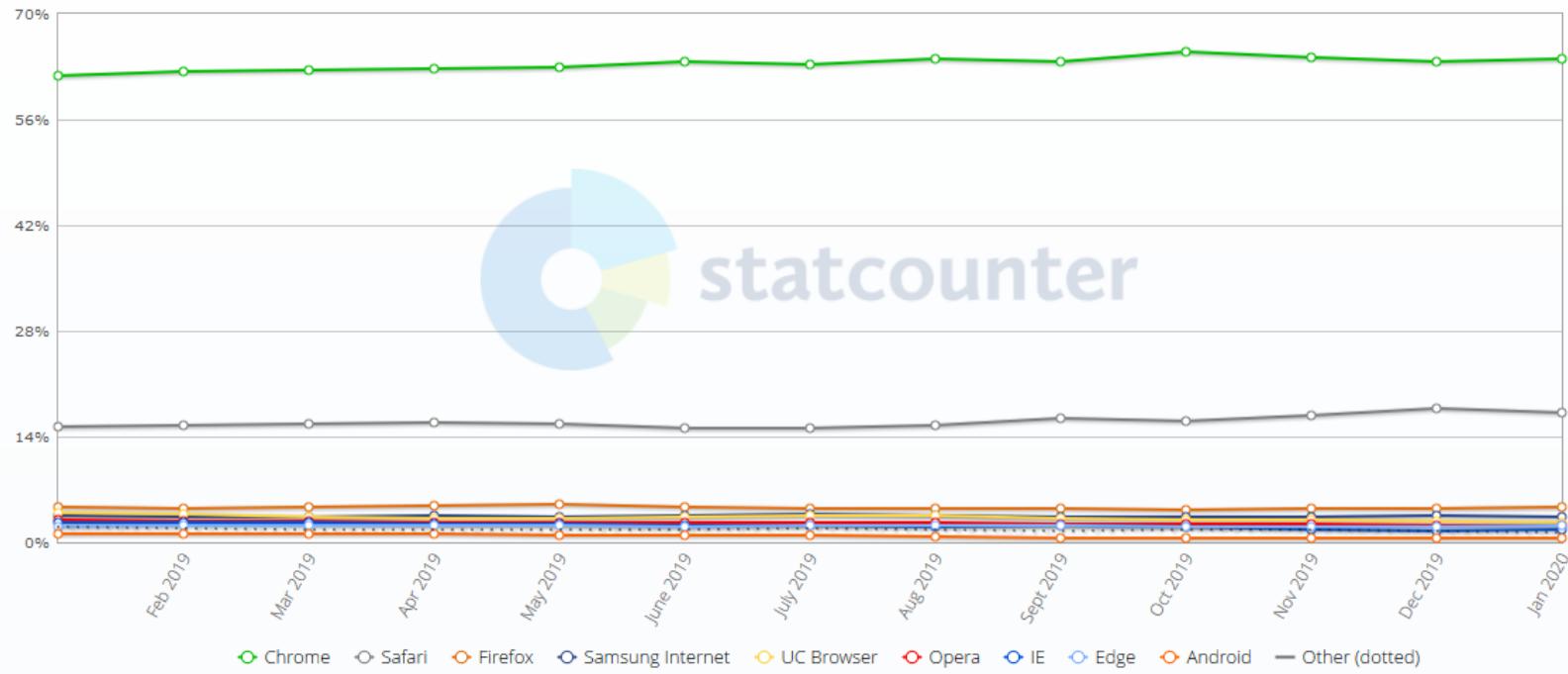
Fuente



# Contexto: Navegadores (evolución)

Browser Market Share Worldwide  
Jan 2019 - Jan 2020

Edit Chart Data



Fuente



## *¿Hacia dónde se dirige la Web?*

---

- El inicio de la World Wide Web se produjo hacia 1989, y consiste en un sistema de documentos de hipertexto enlazados y accesibles mediante Internet.
- El usuario visualiza los contenidos mediante el navegador Web.
- Inicialmente los contenidos eran “panfletos” creados por el propietario de cada página que el usuario podía leer (y acceder a sus enlaces).
- Posteriormente surgieron tecnologías para ampliar la capacidad de los navegadores y la experiencia del usuario (Ej. Applets, JavaScript, Flash..)
- Tendencia actual:
  - Soporte de contenidos multimedia
  - Los contenidos son creados por los propios usuarios (Web 2.0+)
  - “Todo está en la Web”
  - **Aplicaciones Web: Pretenden imitar al comportamiento de las aplicaciones que puede ejecutar el usuario en su máquina local (Aplicaciones Web Progresivas)**



# *Aplicaciones Web vs. Aplicaciones de Escritorio*

- Las aplicaciones Web son ubicuas.
- No es necesaria instalación.
- Intrínsecamente multiplataforma.
- Ligeras.
- **Electron:** Framework para construir aplicaciones de escritorio con forma de aplicaciones Web (HTML, CSS, JS).
- **Xamarin, PhoneGap:** Aplicaciones híbridas móviles.
- Aplicaciones Web progresivas: **Service workers, push, ...**

- Aplicaciones de página única (**SPA**):
  - Página Web cargada al inicio
  - Recursos obtenidos al inicio o bajo demanda (lazy-loading)
  - El cliente se encarga de los cambios en el estado de navegación



# *SPAs: Ventajas e Inconvenientes*

- Ventajas:

- No necesitan ningún plugin externo
- Cantidad reducida de recursos para ejecutarse
- Experiencia más fluida para el usuario

- Desventajas:

- Dificultad para ser indexadas en motores de búsqueda (los crawlers no suelen ejecutar JS)
- No se comportan adecuadamente frente a ciertos eventos gestionados por el navegador (atrás, conservar la posición del scroll en páginas visitadas, etc.)



# Introducción: El framework jQuery



- jQuery es una librería ligera de JavaScript.
- Propósito: Facilitar el uso de JavaScript en aplicaciones Web. Se especializa en proporcionar atajos para múltiples tareas comunes, de manera que con una (o unas pocas) líneas se pueda obtener el mismo resultado que con muchas líneas de código JavaScript.
- Simplifica algunas de las tareas “más pesadas” de JavaScript, como la manipulación del DOM o las llamadas AJAX.
- La librería de jQuery contiene las siguientes características:

Manipulación de HTML/DOM

Manipulación de CSS

Eventos HTML

Efectos y animaciones

AJAX

Otras utilidades



- jQuery es el framework de JavaScript más utilizado en la actualidad.
- Al igual que otras librerías incluye muchas herramientas para facilitar el trabajo del desarrollador.
- Se puede descargar de <http://jquery.com>
- jQuery** es software libre. Posee la Licencia MIT y la Licencia Pública General de GNU v2.
- Admite Plugins para aumentar la funcionalidad.



# *El framework jQuery*



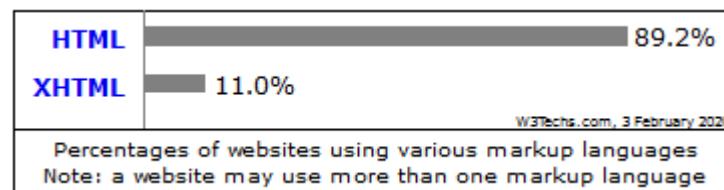
- Existen ampliaciones, como, **jQuery UI**, librería de componentes para jQuery que añaden comportamientos, widgets y efectos. De esta manera se consigue que la aplicación web tenga una apariencia muy próxima a una aplicación tradicional de escritorio.

<http://jqueryui.com>



# HTML

- HTML (**HyperText Markup Language**) es el lenguaje que define el formato y apariencia de las páginas web.
- Los elementos de HTML se componen de **etiquetas** entre ángulos dentro de la página. En general van pareadas `<html> ... </html>`, aunque algunas pueden aparecer solas: `<br>`
- El navegador interpreta las etiquetas y su contenido para generar el contenido de la página web.
- Es posible embeber imágenes y objetos en HTML, así como usar hojas de estilos en cascada (CSS) para definir la apariencia de los elementos de la página.
- Diseñado inicialmente por Tim Berners-Lee en 1980.



Fuente



# HTML

- Algunas etiquetas (clásicas) de HTML:



Tag	Description
<html> ... </html>	Declares the Web page to be written in HTML
<head> ... </head>	Delimits the page's head
<title> ... </title>	Defines the title (not displayed on the page)
<body> ... </body>	Delimits the page's body
<h n> ... </hn>	Delimits a level <i>n</i> heading
<b> ... </b>	Set ... in boldface
<i> ... </i>	Set ... in italics
<center> ... </center>	Center ... on the page horizontally
<ul> ... </ul>	Brackets an unordered (bulleted) list
<ol> ... </ol>	Brackets a numbered list
<li> ... </li>	Brackets an item in an ordered or numbered list
 	Forces a line break here
<p>	Starts a paragraph
<hr>	Inserts a horizontal rule
	Displays an image here
<a href="..."> ... </a>	Defines a hyperlink



# HTML

- Los elementos HTML pueden tener atributos, que consisten en pares nombre-valor que se incluyen en la etiqueta de inicio, después del nombre del elemento.

- Algunos atributos comunes a distintos elementos HTML son:

- **ID**: Identificador único del elemento. Muy útil para poder acceder al mismo.
- **CLASS**: Permite la clasificación de los elementos en categorías.
- **STYLE**: Permite definir reglas de estilo para ese elemento concreto en la página.
- **TITLE**: Proporciona un título para el elemento.



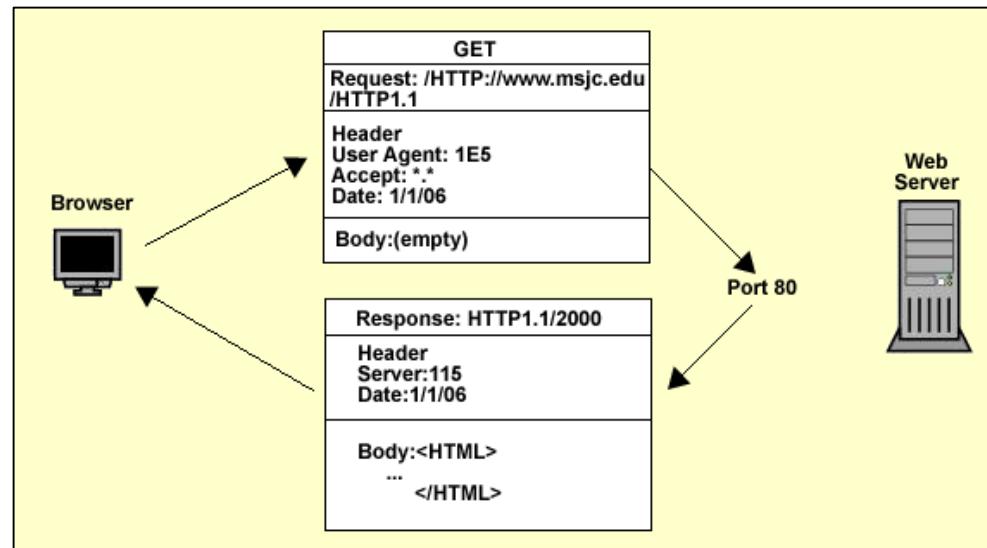


# HTTP



- **Hypertext Transfer Protocol:** Protocolo de nivel de aplicación utilizado en las comunicaciones de la Web para la transferencia de datos e hipertexto.
- Gestiona el envío de peticiones por parte del cliente y de las respuestas por parte del servidor.
- Los recursos se identifican mediante su URL.

- Anatomía de una petición/respuesta:





# Métodos HTTP

- HTTP ofrece una serie de métodos de comunicación. Algunos de ellos:



Method	Description
GET	Request to read a Web page
HEAD	Request to read a Web page's header
PUT	Request to store a Web page
POST	Append to a named resource (e.g., a Web page)
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Reserved for future use
OPTIONS	Query certain options



## Métodos HTTP

- Los más comunes:



**GET:** Envía los datos (claves y valores) en la propia URL, dejando el cuerpo de la petición vacío. Las peticiones pueden quedar en caché. La longitud de los datos a enviar es limitada. No es adecuado para el envío de información sensible.

**POST:** Envía los datos (claves y valores) en el cuerpo de la petición. Las peticiones no se almacenan en caché. Longitud arbitraria de datos.



# Cabeceras HTTP

- Las cabeceras proporcionan información sobre la transacción HTTP (petición o respuesta). Algunas de ellas son:



Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The page encodings the client can handle
Accept-Language	Request	The natural languages the client can handle
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Cookie	Request	Sends a previously set cookie back to the server
Date	Both	Date and time the message was sent
Upgrade	Both	The protocol the sender wants to switch to
Server	Response	Information about the server
Content-Encoding	Response	How the content is encoded (e.g., gzip)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Last-Modified	Response	Time and date the page was last changed
Location	Response	A command to the client to send its request elsewhere
Accept-Ranges	Response	The server will accept byte range requests
Set-Cookie	Response	The server wants the client to save a cookie



# DOM (*Modelo de Objetos de Documento*)

---

- Estándar del W3C (World Wide Web Consortium)

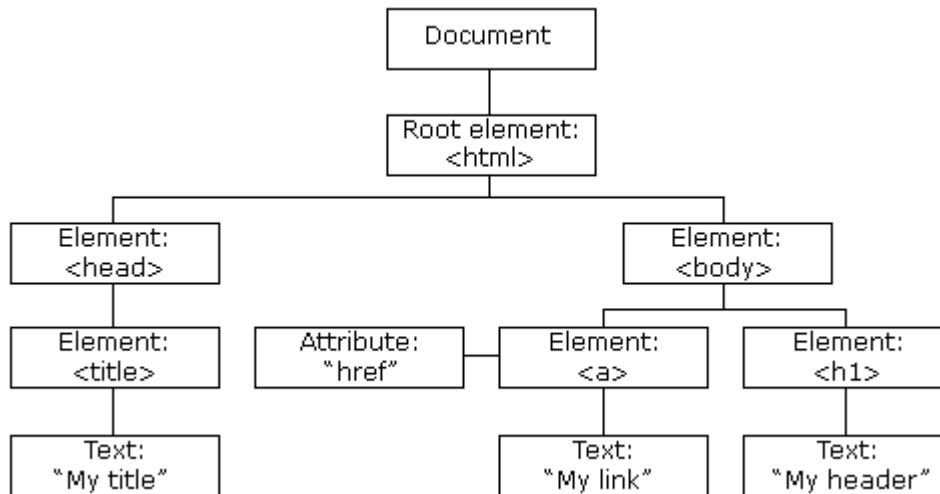
*“The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.”* ([www.w3.org/DOM/](http://www.w3.org/DOM/))

- Define métodos de acceso a documentos.
- Distingue:
  - Core DOM: Modelo para todo tipo de documentos
  - XML DOM: Modelo para documentos XML
  - HTML DOM: Modelo para documentos HTML
- En el **HTML DOM**:
  - Se definen los elementos HTML como **objetos**
  - Se definen las **propiedades** de los elementos HTML
  - Se definen los **métodos** de acceso a los elementos HTML
  - Se definen los **eventos** para los elementos HTML



# HTML DOM

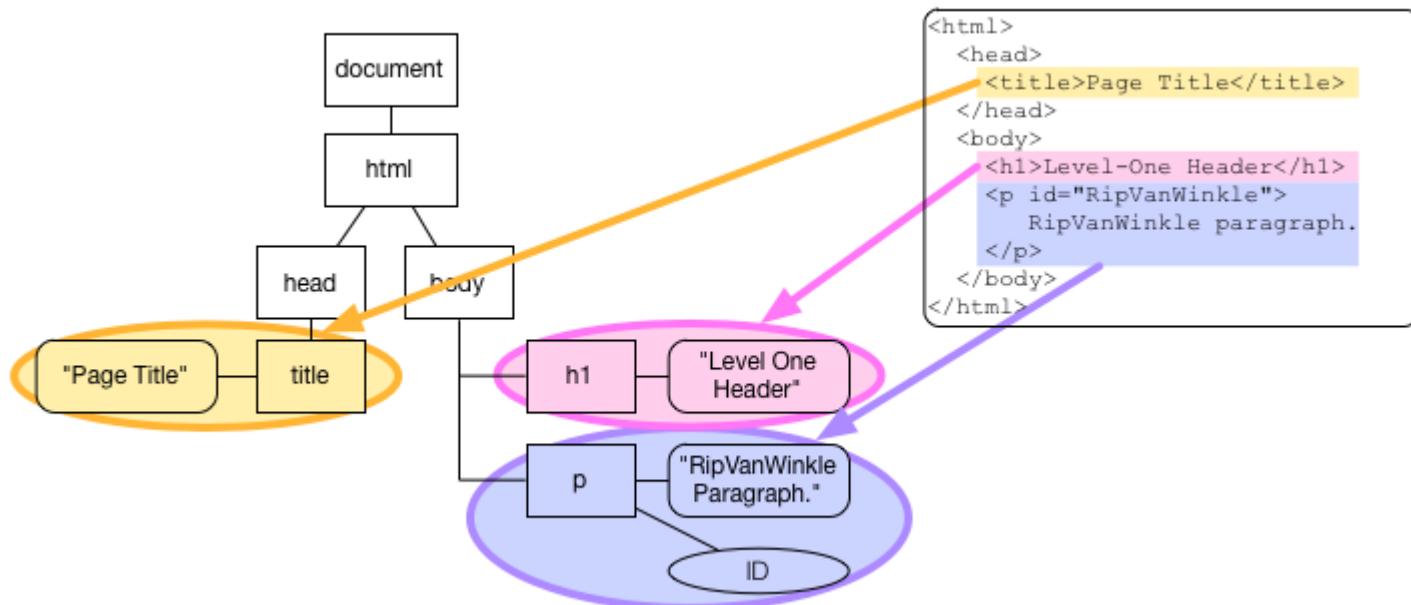
- Cuando un documento HTML se carga en el navegador web, se convierte en una estructura de objetos de documento donde todos los elementos son nodos de un árbol. Por ejemplo:





# HTML DOM

- Los nodos del árbol DOM se corresponden directamente con las distintas partes del documento HTML original.





- Todos los nodos HTML son nodos de **Elemento**.
- Todos los atributos HTML son nodos de **Atributo**.
- El texto dentro de los elementos HTML forma nodos de **Texto**.
- Los comentarios son nodos de **Comentario**.
- El objeto **NamedNodeMap** en HTML DOM representa una colección desordenada de objetos de Atributo.
- **NodeList** representa una lista ordenada de nodos de Elemento.
- En el nivel superior se encuentra en nodo **window**, asociado con la ventana del navegador que está mostrando el documento. El nodo **document** es el que contiene todos los nodos del documento. Además **document** es un atributo de **window**.
- Los eventos que se generan a partir de los elementos HTML también poseen atributos. Dichos atributos dependen del tipo de evento.



# HTML DOM

- Algunas propiedades y métodos de los nodos de Elemento.

Property / Method	Description
<u>element.appendChild()</u>	Adds a new child node, to an element, as the last child node
<u>element.attributes</u>	Returns a NamedNodeMap of an element's attributes
<u>element.childNodes</u>	Returns a NodeList of child nodes for an element
<u>element.className</u>	Sets or returns the class attribute of an element
<u>element.cloneNode()</u>	Clones an element
<u>element.compareDocumentPosition()</u>	Compares the document position of two elements
<u>element.dir</u>	Sets or returns the text direction of an element
<u>element.firstChild</u>	Returns the first child of an element
<u>element.getAttribute()</u>	Returns the specified attribute value of an element node
<u>element.getAttributeNode()</u>	Returns the specified attribute node
<u>element.getElementsByTagName()</u>	Returns a collection of all child elements with the specified tagname
<u>element.hasAttribute()</u>	Returns true if an element has the specified attribute, otherwise false
<u>element.hasAttributes()</u>	Returns true if an element has any attributes, otherwise false
<u>element.hasChildNodes()</u>	Returns true if an element has any child nodes, otherwise false
<u>element.id</u>	Sets or returns the id of an element



# HTML DOM

- Más propiedades y métodos de los nodos de **Elemento**:

Property / Method	Description
<u><a href="#"><b>element.innerHTML</b></a></u>	Sets or returns the content of an element
<u><a href="#"><b>element.insertBefore()</b></a></u>	Inserts a new child node before a specified, existing, child node
<u><a href="#"><b>element.isEqualNode()</b></a></u>	Checks if two elements are equal
<u><a href="#"><b>element.isSameNode()</b></a></u>	Checks if two elements are the same node
<u><a href="#"><b>element.isSupported()</b></a></u>	Returns true if a specified feature is supported on the element
<u><a href="#"><b>element.lastChild</b></a></u>	Returns the last child of an element
<u><a href="#"><b>element.nextSibling</b></a></u>	Returns the next node at the same node tree level
<u><a href="#"><b>element.nodeName</b></a></u>	Returns the name of an element
<u><a href="#"><b>element.nodeValue</b></a></u>	Sets or returns the value of an element
<u><a href="#"><b>element.ownerDocument</b></a></u>	Returns the root element (document object) for an element
<u><a href="#"><b>element.parentNode</b></a></u>	Returns the parent node of an element
<u><a href="#"><b>element.previousSibling</b></a></u>	Returns the previous element at the same node tree level
<u><a href="#"><b>element.removeAttribute()</b></a></u>	Removes a specified attribute from an element



# HTML DOM

- Algunas propiedades y métodos de los nodos **NamedNodeMap** y de **Atributo**:

Property / Method	Description
<u>attr.isId</u>	Returns true if the attribute is of type Id, otherwise it returns false
<u>attr.name</u>	Returns the name of an attribute
<u>attr.value</u>	Sets or returns the value of the attribute
<u>attr.specified</u>	Returns true if the attribute has been specified, otherwise it returns false
<u>nodemap.getNamedItem()</u>	Returns a specified attribute node from a NamedNodeMap.
<u>nodemap.item()</u>	Returns the node at a specified index in a NamedNodeMap
<u>nodemap.length</u>	Returns the number of nodes in a NamedNodeMap
<u>nodemap.removeNamedItem()</u>	Removes a specified attribute node
<u>nodemap.setNamedItem()</u>	Sets the specified attribute node (by name)



# HTML DOM

- Algunas propiedades y métodos del objeto raíz **document**:

Property / Method	Description
<u><a href="#">document.baseURI</a></u>	Returns the absolute base URI of a document
<u><a href="#">document.body</a></u>	Returns the body element of the document
<u><a href="#">document.createAttribute()</a></u>	Creates an attribute node
<u><a href="#">document.createElement()</a></u>	Creates an Element node
<u><a href="#">document.createTextNode()</a></u>	Creates a Text node
<u><a href="#">document.getElementById()</a></u>	Returns the element that has the ID attribute with the specified value
<u><a href="#">document.getElementsByName()</a></u>	Accesses all elements with a specified name
<u><a href="#">document.getElementsByTagName()</a></u>	Returns a NodeList containing all elements with the specified tagname
<u><a href="#">document.images</a></u>	Returns a collection of all the images in the document
<u><a href="#">document.lastModified</a></u>	Returns the date and time the document was last modified
<u><a href="#">document.links</a></u>	Returns a collection of all the links in the document
<u><a href="#">document.renameNode()</a></u>	Renames the specified node
<u><a href="#">document.title</a></u>	Sets or returns the title of the document
<u><a href="#">document.URL</a></u>	Returns the full URL of the document



# HTML DOM: Eventos

---

- HTML DOM proporciona funcionalidad para asignar manejadores de eventos sobre los elementos HTML de la página web.
- Objetivo: Unificar los procedimientos de manejo de eventos en distintos navegadores.
- Permite asociar funciones con los eventos sobre los elementos de la página, de manera que se ejecute el código asociado por el desarrollador.
- Distintos tipos
  - *Teclado*
  - *Ratón*
  - *Objetos HTML*
  - *Formularios*
  - *Interfaz*
  - *Táctiles*
  - *Etc.*



# HTML DOM: Eventos Teclado y Ratón

Property	Description	DOM
<u>onclick</u>	The event occurs when the user clicks on an element	2
<u>ondblclick</u>	The event occurs when the user double-clicks on an element	2
<u>onmousedown</u>	The event occurs when a user presses a mouse button over an element	2
<u>onmousemove</u>	The event occurs when the pointer is moving while it is over an element	2
<u>onmouseover</u>	The event occurs when the pointer is moved onto an element	2
<u>onmouseout</u>	The event occurs when a user moves the mouse pointer out of an element	2
<u>onmouseup</u>	The event occurs when a user releases a mouse button over an element	2

Attribute	Description	DOM
<u>onkeydown</u>	The event occurs when the user is pressing a key	2
<u>onkeypress</u>	The event occurs when the user presses a key	2
<u>onkeyup</u>	The event occurs when the user releases a key	2



# HTML DOM: Eventos Objeto/Marco y Formulario

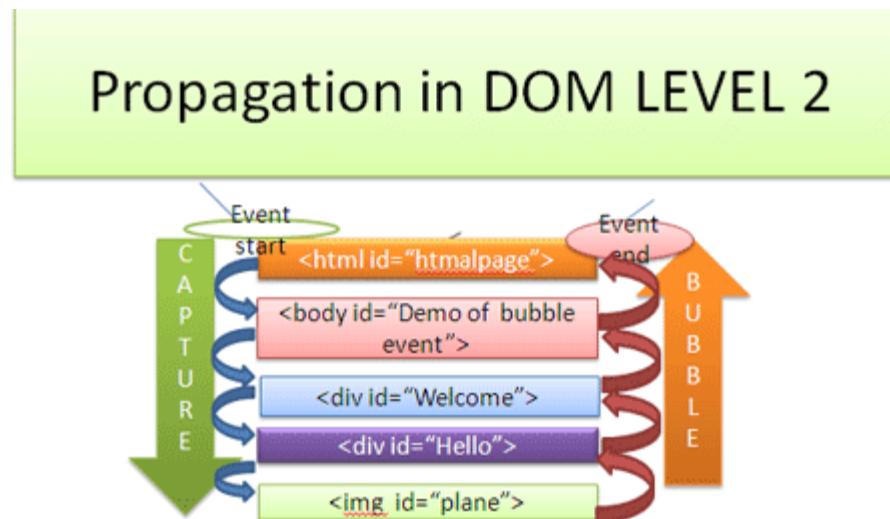
Attribute	Description	DOM
<u>onabort</u>	The event occurs when an image is stopped from loading before completely loaded (for <object>)	2
<u>onerror</u>	The event occurs when an image does not load properly (for <object>, <body> and <frameset>)	
<u>onload</u>	The event occurs when a document, frameset, or <object> has been loaded	2
<u>onresize</u>	The event occurs when a document view is resized	2
<u>onscroll</u>	The event occurs when a document view is scrolled	2
<u>onunload</u>	The event occurs once a page has unloaded (for <body> and <frameset>)	2

Attribute	Description
<u>onblur</u>	The event occurs when a form element loses focus
<u>onchange</u>	The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <select>, and <textarea>)
<u>onfocus</u>	The event occurs when an element gets focus (for <label>, <input>, <select>, textarea>, and <button>)
<u>onreset</u>	The event occurs when a form is reset
<u>onselect</u>	The event occurs when a user selects some text (for <input> and <textarea>)



# HTML DOM: Eventos – Captura y Burbujeo

- Son dos modelos para la generación de eventos.
- Situación: un elemento HTML dentro (hijo) de otro, y ambos tienen un manejador para el mismo evento. **¿Cuál de los dos se dispara primero?**
- El procesado de eventos se realiza en dos fases en todos los navegadores (excepto IE<9): En la fase de **captura** el evento se propaga desde el elemento externo hacia adentro, y en la fase de **burbujeo** recorre el camino contrario.





# HTML DOM: Eventos – Captura y Burbujeo

- HTML DOM proporciona funcionalidad para especificar si se desea que el manejador del evento sea disparado en la fase de captura o en la fase de burbujeo, es decir, primero en el elemento de nivel más alto o primero en el de nivel más bajo.

Method	Description
<code>initEvent()</code>	Specifies the event type, whether or not the event can bubble, whether or not the event's default action can be prevented
<code>preventDefault()</code>	To cancel the event if it is cancelable, meaning that any default action normally taken by the implementation as a result of the event will not occur
<code>stopPropagation()</code>	To prevent further propagation of an event during event flow

Method	Description
<code>addEventListener()</code>	Allows the registration of event listeners on the event target (IE8 = attachEvent())
<code>dispatchEvent()</code>	Allows to send the event to the subscribed event listeners (IE8 = fireEvent())
<code>removeEventListener()</code>	Allows the removal of event listeners on the event target (IE8 = detachEvent())

Method	Description
<code>handleEvent()</code>	Called whenever an event occurs of the event type for which the EventListener interface was registered



# HTML5 – Web Components – Shadow DOM

- Permite encapsular un subárbol DOM dentro de un nodo, aislando del resto del documento.

Fragmento: Uso de shadow DOM.

```
<style>
  p { background: lightgreen; color: black; }
</style>
<div id="mi-huesped"></div>
<p>Estoy fuera de la raíz oculta</p>

<script>
  var elem = document.getElementById("mi-huesped");
  var raizOculto = elem.attachShadow({ mode: "open" });
  raizOculto.innerHTML =
    `<p>Estoy dentro de la raíz oculta</p>
  `;
</script>
```

Estoy dentro de la raíz oculta

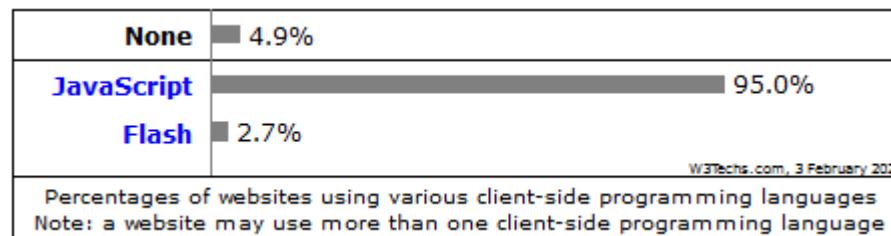
Estoy fuera de la raíz oculta

Dentro de un elemento con Shadow DOM, los estilos, clases e identificadores son locales a dicho elemento, lo que proporciona al desarrollador un mayor control del desarrollo de la aplicación, pudiendo seguir un enfoque más modular



# Fundamentos de JavaScript

- JavaScript es un lenguaje interpretado y multiparadigma (orientado a objetos, imperativo, funcional), usado comúnmente como parte de navegadores web.
  - Interacción con el Usuario
  - Control del navegador
  - Comunicación síncrona/asíncrona
- Desarrollado originalmente por Brendan Eich y distribuido en 1995.
- Permite: funciones recursivas, funciones anónimas, clausuras, etc.
- Posee un tipado muy sencillo, con un conjunto de tipos primitivos y objetos que son pares clave/valor, donde cada valor apunta a un tipo primitivo o a otro objeto.



Fuente



# Evolución de JavaScript

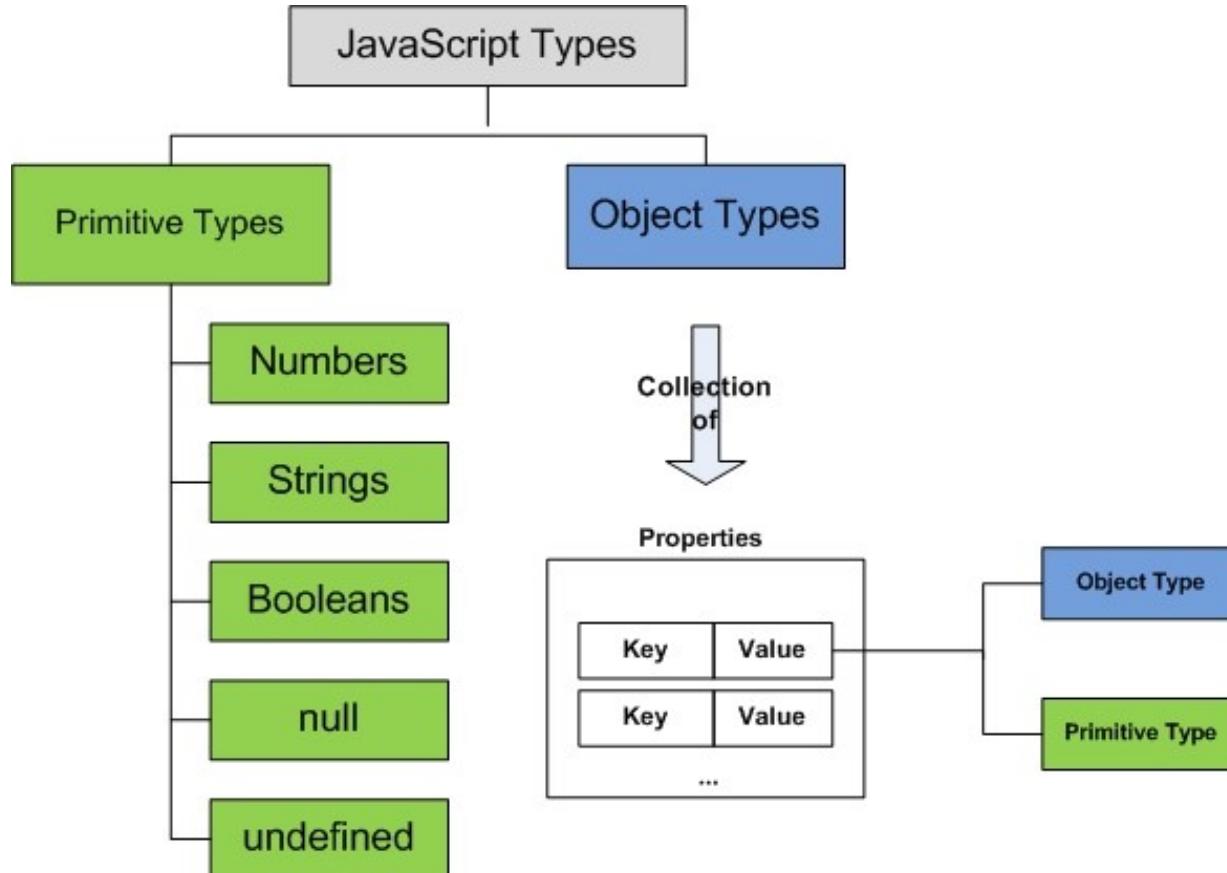
- Se utiliza integrado en un navegador web, permitiendo el desarrollo de interfaces de usuario mejoradas y páginas web dinámicas.
- Javascript ha ido evolucionando y aumentando su funcionalidad de forma paralela a la Web.
- Inicialmente: Utilización de marcos ocultos para conseguir una comunicación con el servidor en la que el servidor enviará información para actualizar una parte de la página principal.
- Posteriormente se introdujeron en JavaScript los marcos flotantes, que pueden ser generados de forma dinámica, facilitando la planificación de la aplicación Web.
- En la versión 5 del navegador MS Internet Explorer se introdujo la interfaz **XMLHttpRequest**, rápidamente adoptada por el resto de navegadores.





# Fundamentos de JavaScript

JS





# HTML y JavaScript

HTML



- JavaScript proporciona al desarrollador métodos para acceder y/o modificar elementos HTML mediante el DOM.
- La forma de ejecutar el código JavaScript es mediante scripts embebidos en el código HTML de las páginas web.
- Ejemplo:

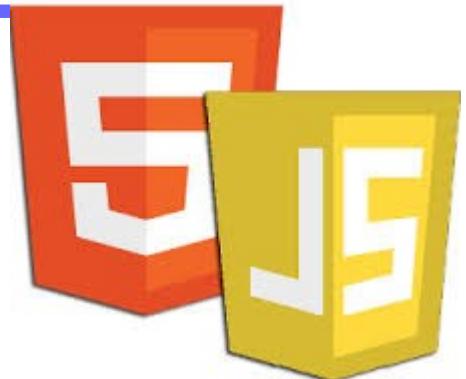
```
<input id="boton1" type="button" value="Pinchame" onclick="miFuncion(); ">
```

- Existe una tendencia a desaconsejar el uso de código JavaScript de manera *inline* en en código HTML como en el ejemplo anterior, y sustituirlo por el modelo no obtrusivo.



# HTML y JavaScript

HTML



- La etiqueta **<script>...</script>** permite definir un script del lado del cliente directamente sobre HTML o en un archivo externo (local o mediante su url).
- El orden de los archivos de script añadidos puede ser importante!!
- Ejemplos:

```
<script type="text/javascript">  
    document.getElementById('miId').innerHTML += '<p>Hola</p>';  
</script>
```

```
<script type="text/javascript" src="miArchivoJS.js"></script>
```

```
<script type="text/javascript" src="http://miurl.com/miJS.js"></script>
```



# HTML y JavaScript

HTML



- Cuando el navegador carga un sitio web con la etiqueta `<script> ...</script>`:
  - 1) Se obtiene el documento HTML
  - 2) Comienza a construirse el árbol DOM
  - 3) Se encuentra la etiqueta `<script>`
  - 4) **El navegador espera a recibir el script (se bloquea)**
  - 5) Cuando se recibe el script se ejecuta
  - 6) Se continúa formateando el documento
- Este comportamiento puede variar con el navegador
- Recomendación (antiquada): Colocar los `<script>` al final de `<body>`
  - Conviene empezar a cargar los script lo antes posible para mejor experiencia del usuario
- Solución: Propiedades **async** y **defer** de `<script>`



# HTML y JavaScript

HTML



- **async** establece carga asíncrona, no bloqueante
- El script se descarga y se ejecuta cuando esté disponible

```
<script type="text/javascript" src="path/to/script1.js" async></script>
<script type="text/javascript" src="path/to/script2.js" async></script>
```

↑  
*El script 2 se podría ejecutar antes que el 1*

- **defer** hace que los scripts se descarguen lo antes posible y se ejecuten en orden. Sólo se empiezan a ejecutar después de haber cargado todo el documento.

```
<script type="text/javascript" src="path/to/script1.js" defer></script>
<script type="text/javascript" src="path/to/script2.js" defer></script>
```



# HTML y JavaScript

HTML



- Los eventos se pueden manejar con JavaScript de varias formas:

- *Definiendo el manejador “en línea” en el documento HTML. Ejemplo:*

```
<input id="boton1" type="button" value="Pínchame" onclick="miFuncion(); "/>
```

- *Asignando la propiedad correspondiente del DOM. Ejemplo:*

```
//- Usando un puntero a una función:
```

```
document.getElementById("boton1").onclick = miFuncion;
```

```
//- Usando una función anónima:
```

```
document.getElementById("boton1").onclick = function () {  
    alert('Hola'); };
```



# HTML y JavaScript

HTML



- Asociando el manejador utilizando `addEventListener`. Este procedimiento permite asociar un número arbitrario de funciones al mismo manejador de evento. Se puede combinar con los procedimientos anteriores. *Ejemplo:*

```
var el = document.getElementById("boton1");
if (el.addEventListener)
    el.addEventListener("click", miFuncion, false);
else if (el.attachEvent)
    el.attachEvent('onclick', miFuncion);
```

→ IE, versiones anteriores a 9

→ IE 9+ y resto de navegadores



- El uso de `addEventListener` pertenece a DOM nivel 2. Se indica en la llamada el evento a observar, la función que lo manejará y un parámetro booleano que establece este parámetro establece:

- Que la función se debe invocar durante la **fase de captura**: valor **true** o **1**
- Que la función se debe invocar durante la **fase de burbujeo**: valor **false** o **0**
- MS IE no soporta invocaciones en la fase de captura para versiones anteriores a IE 9



- En caso de que sea necesario prevenir la propagación de un evento se puede utilizar la funcionalidad ofrecida por el DOM:

```
function pararPropagacion(event)
{
    if (event.stopPropagation) {
        event.stopPropagation();
    }
    else if(window.event) {
        window.event.cancelBubble=true;
    }
}
```

→ IE, versiones anteriores a 9

→ IE 9+ y resto de navegadores



- Es una práctica relativamente común en JavaScript el declarar los manejadores de eventos en el cuerpo del manejador del evento **window.onload**
- Evita la inclusión de código “en línea”
- El evento **window.onload** no se dispara hasta que todo el árbol DOM del documento, las imágenes y los recursos adicionales se hayan cargado

```
window.onload = function () {  
    document.getElementById("boton1").onclick = miFuncion;  
}
```



# *HTML y JavaScript*

**HTML**



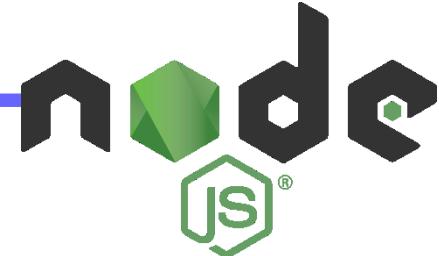
- Ejecución periódica de una función JavaScript: **setInterval** (método del objeto window)
- Permite indicar la función a ejecutar y el período de repetición, expresado en ms. Devuelve un identificador que se puede utilizar posteriormente para cancelar las ejecuciones con **clearInterval**.

```
var nreps = 1;  
function repetir(){  
    alert("Repeticion " + nreps++);  
}  
var id = setInterval(repetir, 5000);  
  
...  
clearInterval(id);
```



- Para seleccionar de elementos HTML desde JavaScript se pueden utilizar los métodos y atributos proporcionados por el DOM para tal efecto: childNodes, getElementById, getElementsByTagName, getElementsByName, etc.
- La palabra reservada **this** en JavaScript hace referencia al objeto dueño de la función en la que se utiliza **this**.
- Por ejemplo, si existe un elemento con el atributo id="miDiv" en HTML se puede hacer:

```
window.onload = function() {  
  
    document.getElementById("miDiv").onclick = function(){this.innerText =  
    "Texto Cambiado";};  
  
};
```



- JavaScript en el servidor!
- Utiliza el motor V8 de JavaScript de Código Abierto de Google
- Basado en un modelo orientado a eventos no bloqueante (asíncrono)
- La programación es distinta del lado del cliente (ej. No hay implementación del DOM).
- Basado en módulos.
- No recomendado para aplicaciones web intensivas en uso de CPU en el servidor.

```
// Load the http module to create an http server.  
var http = require('http');  
  
// Configure our HTTP server to respond with Hello World to all requests.  
var server = http.createServer(function (request, response) {  
    response.writeHead(200, {"Content-Type": "text/plain"});  
    response.end("Hello World\n");  
});  
  
// Listen on port 8000, IP defaults to 127.0.0.1  
server.listen(8000);  
  
// Put a friendly message on the terminal  
console.log("Server running at http://127.0.0.1:8000/");
```



- La utilización de jQuery se basa en llamadas a la función:

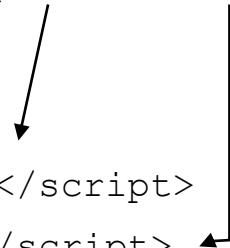
```
jQuery(function() {  
    ....  
});
```

- Donde se suele utilizar el símbolo \$ como atajo:

```
$ (function() {  
    ....  
});
```

- Para utilizar jQuery sólo hay que incluir el script (local o url) en la página HTML:

```
<script type="text/javascript" src="jquery-3.3.1.js"> </script>  
<script src="http://code.jquery.com/jquery-3.3.1.js"></script>
```





- Para asegurar la carga de los elementos HTML antes de ejecutar el script:

```
$ (document) .ready(function () {  
  
    // ...  
  
} );
```

- `$ (document) .ready()` es un evento específico de jQuery (Usa `DOMContentLoaded` si está disponible en el navegador, y si no lo emula), y se diferencia de `window.onload` en que se dispara cuando el navegador ha construido el árbol DOM a partir de los elementos HTML, pero no espera a que las imágenes se hayan cargado completamente. Ante páginas web pesadas la utilización de `$ (document) .ready()` puede dar una sensación de mayor fluidez.
- Similar a la captura del evento `DOMContentLoaded`



- jQuery facilita enormemente la tarea de seleccionar uno o varios elementos HTML simultáneamente. Para ello se emplean los **selectores**.
  - Selector universal: `$ ("*")` – Selecciona todos los elementos
  - Selector de clase: `$ (" . nombreClase ")` – Selecciona todos los elementos con la clase `nombreClase` (`atributo class="nombreClase"`)
  - Selector de elemento: `$ ("elemento")` – Selecciona todos los elementos con la etiqueta especificada
  - Selector de identificador: `$ (" #identificador ")` – Selecciona el elemento con el identificador especificado (`atributo id`)
  - Selector múltiple: `$ (" selector1, selector2, ..., selectorN ")` – Selecciona la combinación de elementos que cumplan alguno de los selectores

```
// e.j.: selecciona un elemento: <span id="footer"></span>
$('#footer');
// Equivalente a document.getElementById("footer")
```



- Se pueden utilizar varios selectores simultáneamente separándolos por comas, como `$('#header, div, .footer')`
- Otros selectores (atributos):
  - `[nombre="valor"]`: Selecciona los elementos que tienen el atributo especificado exactamente con el valor indicado (Ej:  `$("input[value='Hot Fuzz']").next().text("Hot Fuzz");` )
  - `[nombre != "valor"]`: Selecciona los elementos que no tienen el atributo especificado, o en caso de tenerlo el valor asignado es distinto al que se indica en el selector
  - `[nombre]`: Selecciona los elementos que tienen el atributo asignado
  - `[nombre*="valor"]`: Selecciona los elementos cuyo atributo contiene la cadena indicada (el valor indicado es una subcadena del valor del atributo)
  - `[nombre$="valor"]`: Selecciona los elementos que tienen el atributo indicado y cuyo valor termina en la subcadena indicada. La comparación que se realiza diferencia entre mayúsculas y minúsculas.
  - `[nombre^="valor"]`: Selecciona los elementos que tienen el atributo indicado y su valor comienza exactamente con la subcadena indicada.



- **Selectores jerárquicos:** se usan cuando, además del valor de ciertos atributos de los nodos, interesa su posición en el DOM:
  - `$("selector1 selector2")`: Selecciona todos los elementos que satisfacen el segundo selector, pero que además son descendientes de los nodos que satisfacen el primer selector.
  - `$("selectorPadre > selectorHijo")`: Selecciona todos los elementos que satisfacen el selectorHijo, pero que además son hijos directos de elementos que satisfagan el selectorPadre.
  - `$("selector1 ~ selector2")`: Selecciona todos los elementos que satisfagan el selector2 que se encuentren después de elementos hermanos que satisfagan el selector1.
  - `$("selector1 + selector2")`: Selecciona todos los elementos que satisfacen el segundo selector que están inmediatamente precedidos por un elemento que satisfaga el primer selector. Para considerarlos como que uno precede al otro, deben estar en el mismo nivel del árbol.



- **Filtros:** Se utilizan para refinar más las búsquedas y selección de nodos. Se utilizan anteponiendo dos puntos antes de su nombre y se pueden combinar con los anteriores.

```
$('div:first'); // Selecciona el primer DIV.  
$('div:last'); // Selecciona el último DIV.  
$('div:not(#box)'); // Selecciona DIV que no tenga un id="box".  
$('div:even'); // Selecciona los DIV pares.  
$('div:odd'); // Selecciona los DIV impares.  
$('div:eq(4)'); // Selecciona el 5º índice, contado de 0 a 4.  
$('div:gt(2)'); // Selecciona todos los DIV después del 3º  
$('div:lt(2)'); // Selecciona todos los DIV antes del tercero.  
$(':header'); // Selecciona los elementos: h1, h2, h3 etc...  
$(':visible'); // Selecciona los elementos visibles  
// Selecciona los elementos con el texto especificado  
$('div:contains(texto)');  
// Selecciona elementos que no tienen ni texto ni elementos.  
$('div:empty');  
// e.j.: Selecciona los DIV que tengan un IMG de clase 'box'.  
$('div:has(img.box)');
```



- El resultado de hacer selecciones con jQuery (como `var elem1 = $(".miClase");`) son objetos jQuery.
- Estos objetos pueden ser manipulados mediante una serie de métodos y propiedades:
  - `Objeto.length`: número de elementos contenidos en el objeto
  - `Objeto.context`: el nodo DOM pasado al método JQuery para realizar la consulta. Si no se pasó ninguno, tendrá como valor el documento.
  - `Objeto.html ()`: obtiene o establece el contenido del HTML del elemento (similar a la propiedad nativa `innerHTML` del DOM)
  - `Objeto.addClass (clase)`:Añade la clase especificada a todos los elementos seleccionados.
  - `Objeto.removeClass (clase)`: Elimina la clase especificada de los elementos seleccionados.
  - `Objeto.hasClass (clase)`: Devuelve un booleano indicando si el elemento contiene la clase indicada.
  - `Objeto.toggleClass (clase)`: Añade la clase especificada si no existe, o la elimina si ya está presente en el elemento.
  - `Objeto.text ()`: obtiene o establece el contenido del elemento en forma de texto (al utilizarlo como setter, no se devuelven las etiquetas HTML)
  - `Objeto.attr ()`: obtiene o establece un atributo del elemento. Tiene un primer argumento obligatorio para especificar el nombre del atributo. Si queremos utilizarlo como setter, se introduce el valor a establecer como segundo argumento.



# Objetos jQuery



- `Objeto.removeAttr()`: Elimina el atributo de los elementos implicados.
- `Objeto.width()`: obtiene o establece el valor de la anchura del elemento
- `Objeto.height()`: obtiene o establece el valor de la altura del elemento
- `Objeto.position()`: devuelve un objeto con información sobre la posición del primer elemento de la selección, relativa al primer nodo padre con posicionamiento
- `Objeto.offset()`: devuelve un objeto con información sobre la posición del primer elemento de la selección, relativa al documento completo
- `Objeto.val()`: obtiene o establece el valor para elementos de formulario
- `Objeto.eq(i)`: devuelve el elemento que se encuentra en la posición i de la lista de resultados, en forma de objeto jQuery.
- `Objeto.get(i)`: devuelve el elemento que se encuentra en la posición i de la lista de resultados, pero devolviendo directamente el nodo DOM asociado. Podemos también acceder en forma de array al nodo DOM utilizando la notación de corchetes, de modo que escribiríamos `Objeto[i]`
- `Objeto.children()`: devuelve los hijos inmediatos de los nodos. Se le puede pasar de forma opcional un selector, de modo que filtra sólo los hijos que cumplan dicho selector.
- `Objeto.parent()`: devuelve los elementos padre de la lista de nodos actual
- `Objeto.parents()`: devuelve los ancestros de cada nodo de la lista actual
- `Objeto.prev()`: obtiene el hermano que va inmediatamente antes en el árbol del documento para cada nodo de los del objeto
- `Objeto.next()`: obtiene el hermano que va inmediatamente después en el árbol del documento para cada nodo
- `Objeto.is()`: testea el objeto con un selector, elemento u objeto jQuery y devuelve true si al menos una de las selecciones devuelve true



# Ejercicios



- Escribir un script que obtenga el número de párrafos existentes en la página y lo muestre en un mensaje de alerta.
- Sustituir el mensaje de alerta del punto anterior por un texto dentro de un contenedor situado al final de la página.
- Eliminar el contenedor anterior y utilizar los elementos del DOM para mostrar el mensaje al principio o al final de la página.
- Hacer que todos los párrafos de la página aparezcan en negrita.



# Ejercicios



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Selectores</title>
        <script type="text/javascript" src=". jquery.min.js"></script>
        <script type="text/javascript" src=". /00-selectores.js"></script>
    </head>
    <body>
        <h1>Ejercicios básicos con selectores</h1>
        <p>Primer párrafo.</p>
        <p>Segundo párrafo.</p>
        <div><p>Tercer párrafo.</p></div>
        <div id="insertar"></div>
    </body>
</html>
```

**HTML**

```
$(document).ready(function() {

    // Ejercicio 1:
    alert('Hay ' + jQuery('p').length + ' párrafos en el documento.');

    // Ejercicio 2:
    jQuery('#insertar').append('Hay ' + jQuery('p').length + ' párrafos en el documento.');

    // Ejercicio 3:
    jQuery('body').prepend('Hay ' + jQuery('p').length + ' párrafos en el documento.');

    // Ejercicio 4:
    $('p').each(function() {
        $(this).html("<b>" + $(this).html() + "</b>");
    });

});
```

**JS**



# Ejercicios



- Dada la siguiente tabla se desea escribir un script que haga que, para mejorar su legibilidad todas las filas impares mantengan el fondo claro, mientras que las pares aparezcan sobre un fondo oscuro.

```
<table border="1" style="width:30%">
  <caption>Relación de personal</caption>
  <tr>
    <th>Nombre</td>
    <th>Apellidos</td>
    <th>Titulación</td>
  </tr>
  <tr>
    <td>José</td>
    <td>López</td>
    <td>Ingeniero de Telecomunicación</td>
  </tr>
  <tr>
    <td>Antonio</td>
    <td>García</td>
    <td>Licenciado en Matemáticas</td>
  </tr>
  <tr>
    <td>María</td>
    <td>Hernández</td>
    <td>Ingeniero Informático</td>
  </tr>
  <tr>
    <td>Luisa</td>
    <td>Domínguez</td>
    <td>Licenciada en Medicina</td>
  </tr>
</table>
```

**HTML**



# Ejercicios



```
$ (document) .ready(function() {  
  
    alert('Procesando la tabla..');  
  
    jQuery('tr:odd').attr('style',"background-color:grey");  
    // Sería mejor utilizar estilos desde un archivo externo  
    //jQuery('tr:odd').addClass('miClase');  
  
});
```

**JS**



# Eventos en jQuery



- jQuery proporciona el método on() para gestionar la manipulación de eventos.

```
.on(eventos, [selector], [datos], manejador);
```

Siendo cada uno de los argumentos:

- eventos: una cadena de caracteres indicando el nombre del evento (o más de un evento separados por espacios) que se quiere capturar
- selector: un selector para filtrar la lista de elementos hijos del elemento que van a hacer que se lance el evento
- datos: cualquier tipo de datos (un número, string, objeto..) que se quiera pasar en el objeto del evento cuando se llame al manejador
- manejador: función que se ejecutará cuando se produzca el evento. Recibe como argumento un objeto de evento
- Para desasociar un evento se utiliza el método off(), pasando el nombre del evento que se quiere desasociar.

```
$( 'ul#miLista' ).off("click");
```



# Eventos en jQuery



- El método `.on()` en combinación con los correspondientes selectores puede asociar un manejador a todos los elementos seleccionados, incluyendo además **a todos los elementos que en un futuro satisfagan dicho selector.** Esta funcionalidad reemplaza a `.live()` que desde la versión 1.7 de jQuery está en desuso.

```
$('.someClass').live('click', doSomething);
```



```
$(document).on('click', '.someClass', doSomething);
```

- También proporciona métodos propios para el tratamiento de eventos. Algunos de ellos:

<code>click()</code>	Asocia una función al evento click de los elementos de la selección
<code>focusin()</code>	Asocia una función cuando el elemento seleccionado (o algún descendiente) obtiene el foco
<code>focusout()</code>	Asocia una función cuando el elemento seleccionado (o algún descendiente) pierde el foco
<code>blur()</code>	Asocia una función cuando el elemento seleccionado pierde el foco. El evento no se propaga a los hijos.
<code>mousedown()</code>	Asocia una función a los elementos seleccionados cuando el usuario pulsa un botón del ratón
<code>mouseover()</code>	Asocia una función a los elementos seleccionados cuando el usuario sitúa el cursor del ratón encima
<code>mouseout()</code>	Asocia una función a los elementos seleccionados cuando el cursor del ratón sale
<code>keydown()</code>	Desencadena un evento cuando se pulsa una tecla
<code>keypress()</code>	Desencadena un evento cuando se escribe un carácter



# Ejercicio



- Escribir un script para que al hacer click en un botón con la etiqueta “Mostrar Texto” se muestre un texto en un párrafo bajo dicho botón. Cuando el texto esté visible la etiqueta del botón debe ser “Ocultar Texto” y al volver a hacer click se ocultará el texto, volviendo a la situación inicial.



# Ejercicio



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
    <head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Eventos</title>
        <script type="text/javascript" src=".//jquery.min.js"></script>
        <script type="text/javascript" src=".//02-eventos.js"></script>
    </head>
    <body>
        <h1>Ejercicio básico con eventos</h1>
        <p><input type="button" id="botonMostrar" value="Mostrar Texto"></p>
        <p id="textoOculto" >
            <b>Texto para mostrar</b><br>
            Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.
        </p>
    </body>
</html>
```

**HTML**

```
$(document).ready(function() {
    jQuery('#textoOculto').hide();

    jQuery('#botonMostrar').click(function() {
        if(jQuery('#textoOculto').is(':visible')) {
            $('#textoOculto').hide();
            $('#botonMostrar').attr('value', 'Mostrar Texto');
        } else {
            $('#textoOculto').show();
            $('#botonMostrar').attr('value', 'Ocultar Texto');
        }
    });
});
```

**JS**



- La librería jQuery ofrece métodos para llevar a cabo efectos visuales que sirven para mejorar la percepción visual de la página.
- Existen tres métodos para mostrar y ocultar elementos:
  - `show()`: muestra el elemento
  - `hide()`: oculta el elemento
  - `toggle()`: Si está oculto lo muestra, si está visible lo oculta
- Estos métodos pueden recibir los argumentos:
  - Primer argumento: “slow”, “normal”, “fast” o número de miliseg.
  - Segundo argumento: “swing” y “linear” para tipo de interpolación
  - Tercer argumento: función a ejecutar al finalizar

```
$("#clickme").click(function() {  
    $("#miElem").show( "slow", "linear", function() {  
        // Animacion completa --> Hacer algo  
    } );  
} );
```



# Efectos y animaciones en jQuery



- Otros métodos de animaciones, similares a los anteriores, son:
  - fadeIn () : el elemento aparece (aumenta la opacidad)
  - fadeOut () : el elemento desaparece (disminuye la opacidad)
  - fadeToggle () : Si está oculto se muestra, si está visible se oculta progresivamente
  - slideUp () : el elemento se oculta mediante desplazamiento
  - slideDown () : el elemento aparece mediante desplazamiento
  - slideToggle () : Si está oculto se muestra, si está visible se oculta desplazándose
- El método **.animate()** permite variar cualquier propiedad css:

```
.animate( properties [, duration] [, easing] [, complete] );
```

- properties se refiere a las propiedades CSS que la animación modificará
- duration determina cuánto durará la animación
- easing determina la velocidad a la que la animación progresará en diferentes puntos de la animación (la función de interpolación). Esta propiedad se puede especificar por cada propiedad CSS o de manera general, dependiendo de cuál versión de la función se utilice.
- complete especifica la función de callback que se llamará cuando la animación



- Ejemplo:

```
$ ("#miElem").animate({  
    'top': '200px',  
    'left': '100px',  
    'opacity': 0.5  
}, 1000, function() {  
    //La función ha terminado --> hacer lo que sea  
}  
);
```

- La duración se debe especificar en milisegundos (1 segundo en este ejemplo).



# Ejercicio



- Utilizando la imagen del logo de jQuery y un botón hacer que al pulsar el mismo la imagen aparezca o desaparezca con una transición suave.
- Hacer que al llevar el puntero del ratón sobre la imagen se desate una animación en la que se reduzca su tamaño y posteriormente aumente hasta alcanzar el estado inicial.



# Ejercicio



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Animaciones</title>
        <script type="text/javascript" src=". jquery.min.js"></script>
        <script type="text/javascript" src=". /03-animaciones.js"></script>
    </head>
    <body>
        <h1>Ejercicio básico con animaciones</h1>
        <p><input type="button" id="boton" value="Ocultar Imagen"></p>
        </img><br>
    </body>
</html>
```

**HTML**

```
$(document).ready(function() {
    // Ejercicio 1:
    jQuery('#boton').click(function() {
        if(jQuery('#imagen').is(':visible'))
            $('#boton').attr('value', 'Mostrar Imagen');
        else
            $('#boton').attr('value', 'Ocultar Imagen');
        $('#imagen').fadeToggle();
    });

    // Ejercicio 2:
    jQuery('#imagen').hover(function() {
        $('#imagen').animate({width: "toggle"}, 1000);
    });
});
```

**JS**



# Otras funciones de jQuery



- Funciones sobre objetos y arrays nativos de JavaScript:

**`$.inArray(valor, array);`** Devuelve el índice en el que se encuentra, o -1 si no existe. Acepta un tercer argumento opcional para indicar un índice inicial.

**`$.grep(array, función);`** Se ejecuta la función pasando cada elemento de la lista, incluyendo en el array resultante los elementos para los que devuelva true.

**`$.merge(array1, array2);`** Junta el contenido de dos arrays preservando el orden, incluyendo los elementos del segundo al final del primer array. Modifica el primer array.

**`$.each(colección, callback);`** La función será llamada para cada elemento de la colección. Esta función recibe como argumentos el índice del elemento de la colección (o el nombre de la propiedad, en caso de estar iterando sobre un objeto), y el elemento en sí sobre el que se está iterando.



# Interacción con el Usuario: jQuery UI



- Ampliación de jQuery, muy fácil de usar

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Tecnologías de Cliente Web</title>
  </head>
  <body>
    <h1>Ejemplo - Draggable (jQuery UI)</h1>
    <div id="draggable" class="ui-widget-content">
      <p>Texto móvil...</p>
    </div>
  </body>
  <script type="text/javascript" src="jquery.min.js"> </script>
  <script type="text/javascript" src="jquery-ui-1.10.4.custom.js"> </script>
  <script type="text/javascript" src="ejemploDraggable.js"> </script>
</html>
```

**HTML**

```
$(function () {
    $("#draggable").draggable();
});
```

**JS**



# ¿Qué entendemos como AJAX?



- Traducción literal: Javascript y XML Asíncronos (**Aynchronous Javascript And XML**)
- En términos más amplios, AJAX se refiere a la técnica de realizar una aplicación Web que se ejecute en el navegador Web mientras se mantiene una comunicación asíncrona en segundo plano con el servidor, modificando los elementos de la página principal que requieran las acciones del usuario.
  - Paradoja: Ni Javascript ni XML son estrictamente necesarios en una aplicación AJAX.
- AJAX no es un lenguaje de programación, ni hace referencia a la utilización de tecnologías concretas. Se refiere a una metodología en la interacción cliente-servidor.

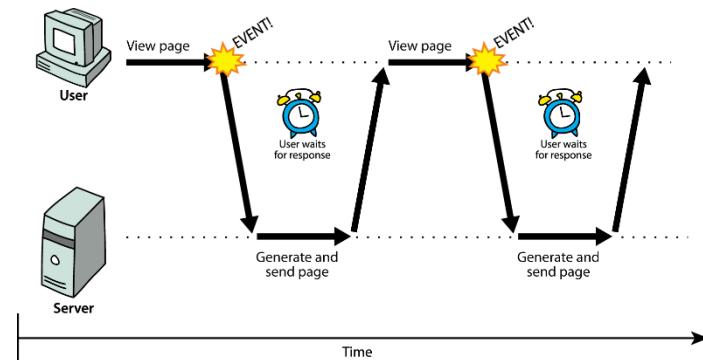


# Comunicación asíncrona con el servidor

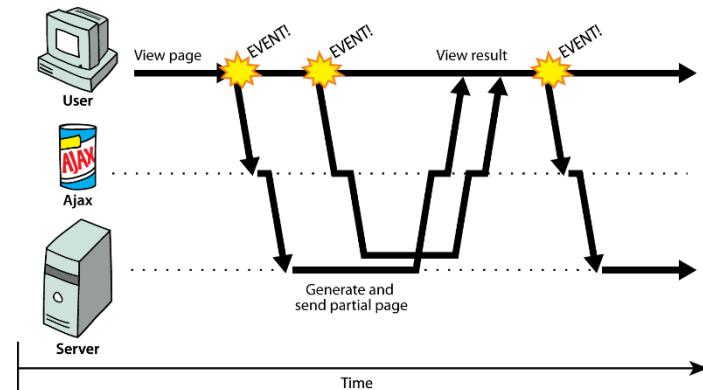


- Comunicación en segundo plano: Se evitan esperas en el cliente, propiciando una sensación de fluidez.

## COMUNICACIÓN SÍNCRONA:

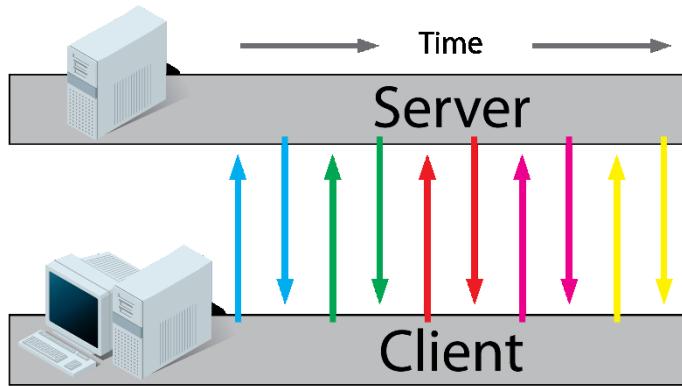


## COMUNICACIÓN ASÍNCRONA:

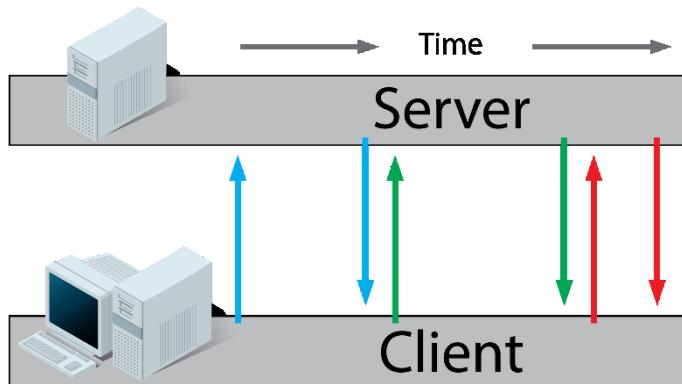




# Retención de la respuesta: Long-Polling



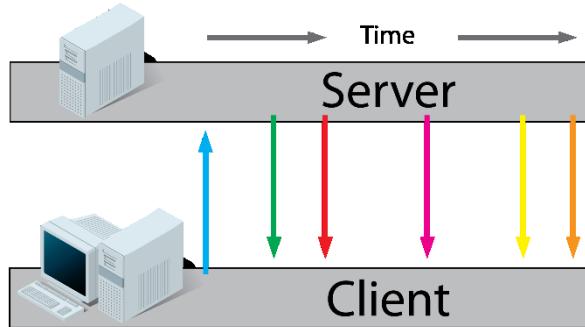
Ajax Polling



Ajax Long-Polling

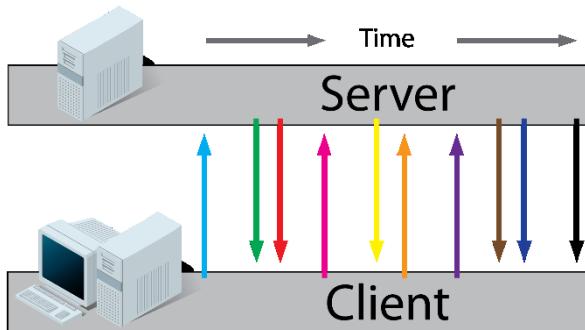


# Otras tecnologías



## HTML5 Server Sent Events

El cliente abre una comunicación con el servidor, y éste envía un evento al cliente cuando hay nuevos datos disponibles.



## HTML5 Websockets

El cliente abre una comunicación con el servidor. Cualquiera de los dos puede enviar mensajes al otro cuando haya nuevos datos disponibles.



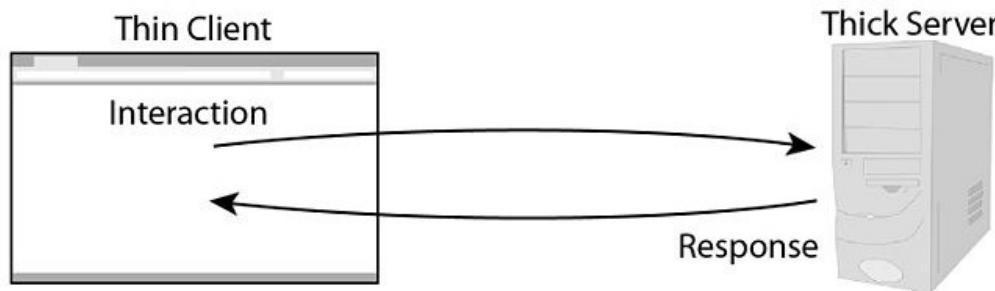
# Aplicaciones Web tradicionales “Thin Client”

## Stateful Server

El servidor debe seguir el estado de los clientes y devolver la página a visualizar de acuerdo al mismo..

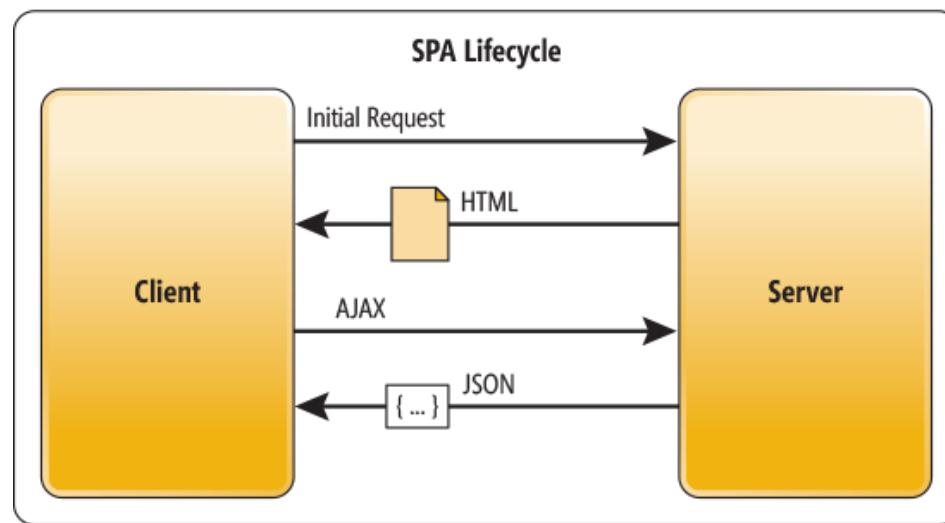
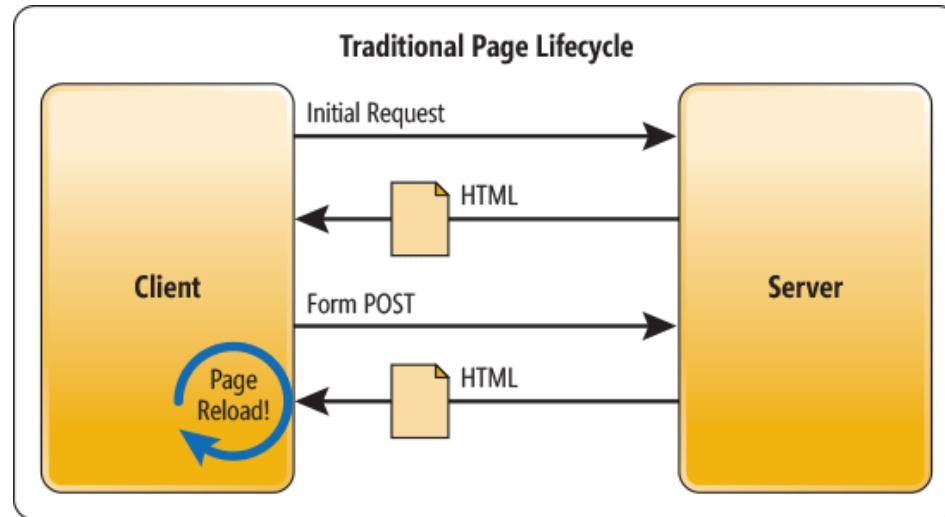
## Stateless Server

El servidor no almacena el estado de los clientes clientes. Este estado debe formar parte de los datos que los clientes transmiten.



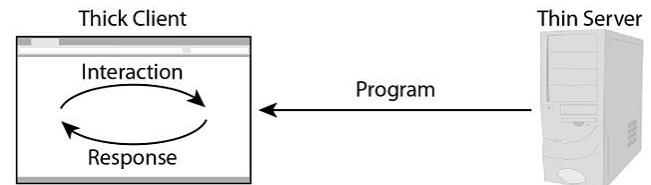


# Aplicaciones de página única (SPA)



## Arquitectura Thin Server

El servidor proporciona servicios Web.





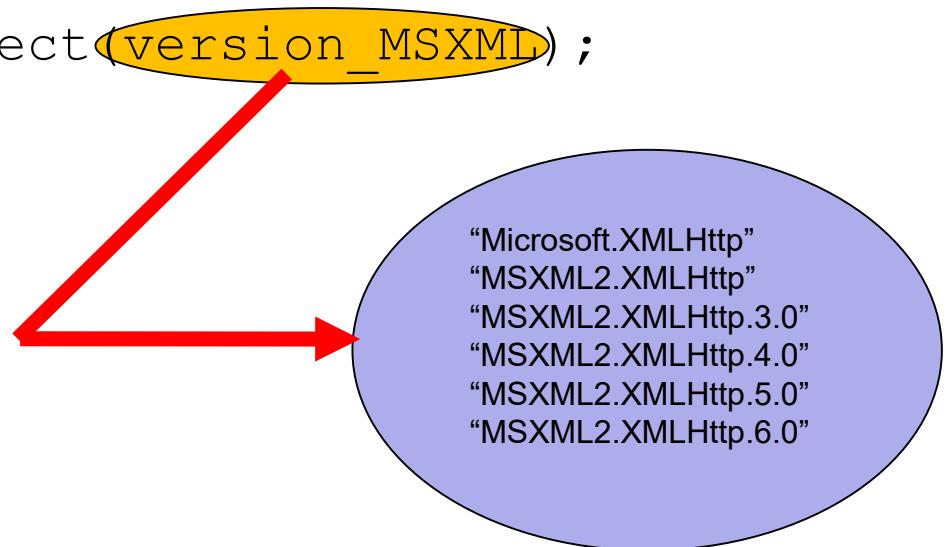
# LEGACY: Creación XMLHttpRequest



- Dependiente del navegador:

**MS Internet Explorer (versiones antiguas):**

```
var obj=new ActiveXObject(version_MSXML);
```



**Firefox, Opera, Safari, Chrome, versiones modernas IE, etc.:**

```
Var obj = new XMLHttpRequest();
```



# LEGACY: Inicializ. XMLHttpRequest



- Después de haber creado el objeto:

```
obj.open(arg1,arg2,arg3);
```

Tipo de petición a realizar  
“GET”  
“POST”

true: Petición asíncrona  
false: Petición síncrona

URL destino  
Ej. “www.miservidor.com/miscript.php”

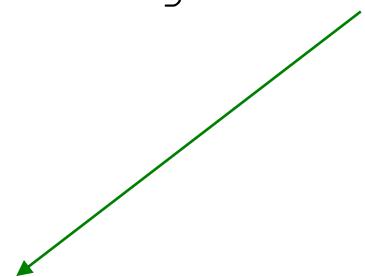


# LEGACY: Capturando cambios de estado



- Después de haber inicializado el objeto se especifica la función JavaScript (en el navegador) que se ejecutará cuando llegue la respuesta del servidor (con los datos pedidos):

```
obj.onreadystatechange=miFuncion;
```



```
function miFuncion() {  
  
    // BLAH BLAH BLAAAHHH  
    // (Se utilizan los datos devueltos por el servidor)  
    // (Se actualizan partes de la página principal)  
}
```



# LEGACY: Comprobación del estado



- La función asignada a onreadystatechange se invoca siempre que cambie la propiedad readyState del objeto XMLHttpRequest.
- Conviene comprobar el estado de esta propiedad para realizar las acciones pertinentes:

0 - Sin inicializar - El objeto se ha creado pero no se ha llamado al método open()  
1 - Cargando - Se ha llamado al método open() pero la petición todavía no se ha enviado.  
2 - Cargado - La petición se ha enviado  
3 - Se ha recibido una respuesta parcial  
4 - Completo - Se han recibido los datos y se ha cerrado la conexión



# LEGACY: Envío de la petición



- Después de haber configurado el manejador de la función de retorno se puede enviar la petición:

```
obj.send(cuerpo);
```

Es el cuerpo de la petición a enviar. Tenemos que realizar un tratamiento diferente si es una petición GET o POST

GET: Los parámetros se pasan al servidor en la URL. El cuerpo de la petición está vacío: **obj.send(null);**

POST: Los parámetros se pasan al servidor en el cuerpo de la petición:  
**obj.send(param);**

```
param="parametro1=valor1&parametro2=valor2&.....&parametron=valorn";
```

Además, debemos configurar las cabeceras:

```
obj.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
obj.setRequestHeader("Content-length", param.length);
obj.setRequestHeader("Connection", "close");
```



# Herramientas para AJAX con jQuery



## .load(url [, datos] [, función])

Realiza una petición al servidor, accediendo a la url especificada y pasando los parámetros (opcionales) indicados, en formato de clave/valor. Se puede especificar, también de manera opcional, una función que se ejecutará cuando se haya recibido satisfactoriamente la respuesta del servidor. El resultado es situado automáticamente en el elemento desde que se realizó la llamada.

*Ejemplo: Al hacer click sobre un botón cargar el contenido del archivo demo\_test.txt en un elemento tipo <div> con identificador div1:*

```
$ ("button") .click(function() {  
    $ ("#div1") .load ("demo_test.txt") ;  
} ) ;
```



# *Herramientas para AJAX con jQuery*



## **.loadIfModified(url [, datos] [, funcion])**

Esta llamada es similar a load(), pero con la particularidad de que sólo se cargan los datos si ha habido modificaciones en los mismos desde la llamada anterior.



# Herramientas para AJAX con jQuery



## **jQuery.get(url [, datos] [, funcion] [, tipo])**

Esta llamada realiza una petición HTTP GET al servidor, accediendo a la url especificada y pasando los parámetros (opcionales) indicados, en formato de clave/valor. Se puede especificar, también de manera opcional, una función que se ejecutará cuando se haya recibido satisfactoriamente la respuesta del servidor. También se puede indicar el tipo de los datos que se transmiten a la función, que serán "xml", "html", "script", "json", "jsonp" o "text".

```
$.get("text.html", funcion(data) {  
    $("#resultado").html(data);  
} );
```



# *Herramientas para AJAX con jQuery*



## **jQuery.post(url [, datos] [, funcion] [, tipo])**

Esta llamada es similar a .get(), pero con la particularidad de que los datos se transmiten en este caso utilizando el método HTTP POST.

## **jQuery.getScript(url [, funcion])**

Esta llamada carga un script del servidor utilizando GET y lo ejecuta. Opcionalmente se puede pasar una función como argumento que se ejecutará si la respuesta se ha recibido satisfactoriamente.

```
$.getScript("test.js");
```



# Herramientas para AJAX con jQuery



## jQuery.ajax(opciones)

Este método permite hacer una consulta AJAX completa, controlando todas las opciones posibles. La forma general de la llamada es:

```
$ .ajax ({  
    opcion_1: valor_1,  
    opcion_2: valor_2,  
    .....  
    opcion_n: valor_n  
} );
```

Donde muchas de las posibles opciones son opcionales (la única obligatoria es "url"). Los nombres de las opciones son los que se presentan a continuación.



# Herramientas para AJAX con jQuery



Opción	Descripción
<u>url</u>	Cadena de caracteres con la dirección de la consulta.
<u>type</u>	Cadena de caracteres que contiene el método a usar para la consulta. por defecto es GET.
<u>dataType</u>	Cadena de caracteres que indica el formato de los datos enviados. Si no se indica nada jQuery usa el tipo MIME para determinar el formato adecuado: responseXML o responseText. Los tipos disponibles son: "xml", "html", "script", "json".
<u>ifModified</u>	Valor booleano que indica si el servidor debe comprobar si los datos a devolver son distintos a los de la última consulta antes de devolverlos con éxito. Por defecto es false.
<u>timeout</u>	Número de milisegundos tras el cual la consulta se debe considerar como fallida.
<u>global</u>	Valor booleano que permite lanzar la ejecución del visor de eventos global de AJAX. Por defecto true. Con un valor false se ignoran los desencadenadores de eventos como ajaxStart() o ajaxStop().
<u>beforeSend</u>	Una función que se debe ejecutar antes de enviar la consulta. Esto permite modificar el objeto XMLHttpRequest antes de que se envíe para especificar, por ejemplo, encabezados HTTP personalizados.
<u>error</u>	Función que se debe ejecutar si falla la consulta. La función tiene tres argumentos: el objeto XMLHttpRequest, una cadena de caracteres que describe el tipo de error que se ha producido y un objeto exception, si se ha creado.



# Herramientas para AJAX con jQuery

Opción	Descripción
<u>success</u>	Función que se invoca si la consulta se lleva a cabo con éxito. Sólo recibe un argumento: los datos devueltos por el servidor.
<u>complete</u>	Función que se invoca cuando la consulta termina. Tiene dos argumentos: el objeto XMLHttpRequest y una cadena de caracteres describiendo el tipo de éxito de la consulta.
<u>data</u>	Los datos que se envían al servidor, formados por una query string con pares clave/valor.
<u>processData</u>	Valor booleano que indica si los datos de la opción data se deben convertir automáticamente a cadena de caracteres. Por defecto true.
<u>contentType</u>	Cadena de caractres que contiene el MIME de los datos que se envían al servidor. Por defecto se conserva el MIME application/x-www-form-urlencoded
<u>async</u>	Valor booleano que indica si la consulta es asíncrona. Por defecto es true.
<u>cache</u>	Valor booleano que, en caso de ser false, impide colocar en la cache del navegador la página cargada.
<u>password</u>	Si el acceso HTTP de la consulta necesita una contraseña.
<u>username</u>	Si el acceso HTTP de la consulta necesita un nombre de usuario.
<u>statusCode</u>	Permite llamar a una función ante un error con código específico.
<u>xhr</u>	Permite crear el ActiveXObject (IE) o el XMLHttpRequest (resto).



# Herramientas para AJAX con jQuery



**jQuery.getJSON(url [, datos] [, funcion])**

Es un atajo para:

```
$ .ajax ( {  
    dataType: "json",  
    url: url,  
    data: datos,  
    success: funcion  
} ) ;
```

Los datos se envían mediante GET y pueden ser objetos (son convertidos a query string automáticamente)



# Herramientas para AJAX con jQuery



## jQuery.ajaxSetup(Parametros)

Permite definir los parámetros globales para todas las llamadas de la página.

```
$.ajaxSetup( {  
    url: "test.htm",  
    global: false,  
    type: POST  
} );
```

De esta manera se podría realizar una petición al servidor escribiendo sencillamente `$.ajax( {} );`



# *Herramientas para AJAX con jQuery*



## **.ajaxSend(funcion)**

Asigna una función que se ejecutará antes del envío de la consulta AJAX. Este método devuelve un objeto jQuery.

Ejemplo:

```
$ ("#loading") .ajaxSend(function() {  
    $ (this) .show () ;  
} ) ;
```



# Herramientas para AJAX con jQuery



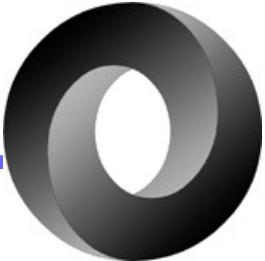
**.ajaxStart(funcion):** Asigna una función que se ejecutará cuando comienza una consulta AJAX.

**.ajaxStop(funcion):** Asigna una función que se ejecutará cuando termina una consulta AJAX.

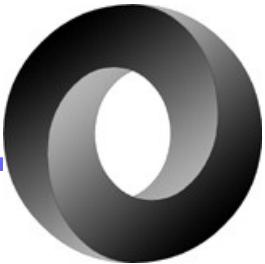
**.ajaxSuccess(funcion):** Asigna una función que se ejecutará cuando termina con éxito una consulta AJAX.

**.ajaxComplete(funcion):** Asigna una función que se ejecutará cuando termina el proceso completo de una consulta AJAX.

**.ajaxError(funcion):** Asigna una función que se ejecutará cuando falla una consulta AJAX.



- JSON es el acrónimo de JavaScript Object Notation.
- Es un formato para el intercambio de datos (alternativa a XML)
- Formato de datos muy ligero basado en un subconjunto de la sintaxis de Javascript (literales de matrices y objetos).
- Las definiciones JSON pueden incluirse dentro de archivos Javascript y acceder a ellas sin ningún análisis adicional como los requeridos con lenguajes basados en XML.

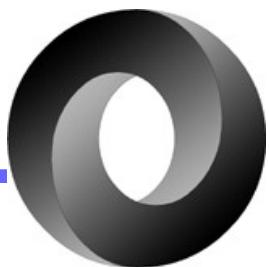


- Funciones nativas para la manipulación de objetos y cadenas JSON en navegadores modernos (Internet Explorer 8+, Firefox 3.1+, Safari 4+, Chrome 3+, ...)
- Para navegadores antiguos: utilizar codificación y decodificación desde jQuery o utilizando la librería:  
<http://www.json.org/json2.js>

```
if(!JSON || !JSON.parse || !JSON.stringify)
    document.write('<script src="json2.js"></script>');
```



# *Conversión de objetos JavaScript en JSON*



- Para codificar los datos en sintaxis JSON pasamos un objeto al método `JSON.stringify()`. Por ejemplo:

```
var persona = {  
    "nombre" : "carlos",  
    "NIF" : "1234567A",  
};
```

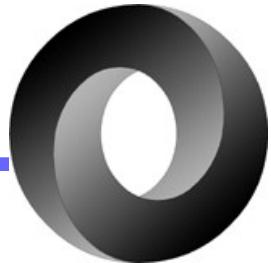
```
cadena=JSON.stringify(persona);
```

El resultado es la siguiente cadena, que puede ser directamente enviada al servidor:

```
{"nombre" : "carlos", "NIF" : "1234567A"}
```



## *Conversión de cadenas JSON a objetos JS*



- Para convertir una cadena JSON recibida del servidor a objeto JavaScript se utiliza el método `JSON.parse()`. Por ejemplo:

```
var personaRecibida = JSON.parse(cadenaRecibida);
```

Con esta sentencia se puede “deshacer” la conversión vista en la diapositiva anterior.

- También existe el método `$.toJSON` de jQuery para codificar. Utiliza `JSON.stringify` con navegadores modernos, y una implementación propia para navegadores anteriores.



# Ejercicio



- Diseñar una aplicación que envíe al servidor un objeto JSON conteniendo los campos “nombre” y “NIF”. El servidor lo recibirá, añadirá otros elementos y devolverá el array contenido el objeto original y los añadidos al cliente, codificados mediante JSON, que serán mostrados al usuario.



# Ejercicio



```
// MANIPULACIÓN ARRAYS JSON:  
$(document).ready(miFuncion);  
  
function miFuncion(){  
    var persona = {  
        "nombre" : "Carlos",  
        "NIF" : "1234567A",  
    };  
  
    $.getJSON("probar_json.php", persona, function(response) {  
  
        var s = "";  
        for(var i=0; i<response.length; i++){  
            s += "nombre: " + response[i].nombre + "\n";  
            s += "NIF: " + response[i].NIF + "\n";  
        }  
        alert(s);  
    });  
}
```

**JS**



# *Vue.js: Introducción*



- Framework front-end de código abierto <https://vuejs.org>
- Orientado a la construcción de SPAs e interfaces Web
- También permite desarrollar aplicaciones de escritorio y móviles junto con el framework Electron.
- Define **componentes** que extienden los elementos básicos HTML.
- Define directivas aplicadas como atributos de componentes.
- Adopción incremental, más fácil de aplicar que frameworks monolíticos.
- Última versión: Vue 3 Core, lanzada a finales de 2020.



# Vue.js: La instancia Vue



- Las aplicaciones Web con Vue.js se basan en la definición de instancias Vue. Esto se lleva a cabo con la función Vue:

```
var vm = new Vue({  
    // opciones  
})
```

**JS**

- Dentro de esta instancia se incluye el objeto **data**, que tiene todas sus propiedades asociadas al sistema de reactividad de Vue (al modificarlas la vista reaccionará cambiando para ajustarse a los nuevos valores)

- Las propiedades son reactivas sólo si existían cuando se creó la instancia de Vue. Con Object.freeze() se puede evitar que se modifiquen propiedades existentes dentro de data.

```
var data = { a: 1 }  
  
// El objeto se agrega a una instancia de Vue  
var vm = new Vue({  
    data: data  
})
```

**JS**



# Vue.js: Hooks de la instancia



- Existen funciones (**Hooks**) que permiten realizar algo en diferentes etapas del ciclo de vida de la instancia. Algunos son:
  - **created**: tras crear una instancia (antes de renderizar plantillas o virtual DOM)
  - **mounted**: acceso a componentes reactivos, plantillas y DOM
  - **updated**: tras cambios en componentes
  - **destroyed**: tras destruir un componente

```
new Vue({  
  data: {  
    a: 1  
  },  
  created: function () {  
    // `this` hace referencia a la instancia vm  
    console.log('a es: ' + this.a)  
  }  
})
```

**JS**



# Vue.js: Renderizado Declarativo



- Con `{{ }}` se realiza la **interpolación de texto**: enlaza el DOM y los datos de manera **reactiva**. No es necesario interactuar con HTML directamente, sino con las instancias de Vue.

```
<script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
...
<div id="app">
  {{ texto }}
</div>
```

**HTML**

```
var app = new Vue({
  el: '#app',
  data: {
    texto: 'Hello Vue!'
  }
})
```

**JS**



# Vue.js: Renderizado Declarativo



- Las directivas de Vue comienzan por **v-** y son atributos que permiten añadir comportamiento reactivo.
- La directiva **v-bind** enlaza el valor de un atributo a datos contenidos en instancias Vue.

```
<div id="app-2">
  <span v-bind:title="message">
    Hover your mouse over me for a few seconds
    to see my dynamically bound title!
  </span>
</div>
```

**HTML**

```
var app2 = new Vue({
  el: '#app-2',
  data: {
    message: 'You loaded this page on ' + new Date().toLocaleString()
  }
})
```

**JS**

- Las llaves dobles no se pueden utilizar dentro de los atributos



# *Vue.js: Enlace de Clases y Estilos*



- Un caso de uso muy frecuente es el enlace del atributo **class** y/o **style** de algún elemento
- Vue incorpora funcionalidad específica para estos enlaces
- Se puede pasar un objeto a **v-bind:class** para asignar clases dinámicamente. Por ejemplo:

```
<div v-bind:class="classObject"></div>
```

**HTML**

```
...
data: {
  classObject: {
    active: true,
    'text-danger': false
  }
}
```

**JS**

- En este caso el elemento tiene la clase **active**. Se puede cambiar la asignación de clases reactivamente.



# *Vue.js: Enlace de Clases y Estilos*



- Para el enlace de estilos podemos utilizar **v-bind:style**
- Para definir el estilo se pasa un objeto JS

```
<div v-bind:style="styleObject"></div>
```

**HTML**

```
...
data: {
  styleObject: {
    color: 'red',
    fontSize: '13px'
  }
}
```

**JS**

- Alternativamente se puede pasar un array en lugar de un objeto tanto a **v-bind:class** como a **v-bind:style** con el mismo resultado



# Vue.js: Más directivas



- Se puede realizar interpolaciones únicas que no se actualizan en el cambio de datos usando la directiva **v-once**
- La directiva **v-bind** enlaza el valor de un atributo a datos contenidos en instancias Vue.

```
<span v-once>Esto nunca cambiara: {{ msg }}</span>
```

**HTML**

- Las llaves dobles interpretan los datos como texto plano, no HTML. Para generar se deberá utilizar la directiva **v-html**

```
<p>Using mustaches: {{ rawHtml }}</p>
<p>Using v-html directive: <span v-html="rawHtml"></span></p>
```

**HTML**



# Vue.js: Directivas Condicionales



- Con **v-if** se condiciona la existencia de un elemento HTML a los datos de alguna instancia Vue.

```
<div id="app-3">  
  <span v-if="seen">Now you see me</span>  
</div>
```

**HTML**

```
var app3 = new Vue({  
  el: '#app-3',  
  data: {  
    seen: true  
  }  
})
```

**JS**

- También puede utilizarse **v-else**, o **v-else-if**

```
<div v-if="type === 'A'">  
  A  
</div>  
<div v-else-if="type === 'B'">  
  B  
</div>  
<div v-else-if="type === 'C'">  
  C  
</div>  
<div v-else>  
  Si no es A, B o C  
</div>
```

**HTML**



# *Vue.js: Directivas Condicionales*



- Se puede aplicar dentro de un elemento <template> que actúa como un envoltorio, afectando a más de un elemento:

```
<template v-if="ok">
  <h1>Título</h1>
  <p>Párrafo 1</p>
  <p>Párrafo 2</p>
</template>
```

**HTML**

- El resultado final renderizado no incluye el elemento <template>
- La directiva **v-show** es una alternativa más simple a v-if. En este caso los elementos siempre se representan en el DOM, y se hace visible o no mediante la propiedad CSS **display** del elemento.



# Vue.js: Directivas de Bucle



- Con **v-for** pueden mostrar datos dinámicamente a partir de los valores de un Array.

```
<div id="app-4">
<ol>
  <li v-for="todo in todos">
    {{ todo.text }}
  </li>
</ol>
</div>
```

**HTML**

```
var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
      { text: 'Learn JavaScript' },
      { text: 'Learn Vue' },
      { text: 'Build something awesome' }
    ]
  }
})
```

**JS**

- Se puede usar *of* en lugar de *in*
- Se puede pasar un segundo argumento (*<li v-for="(todo, index) in todos">*) que contendrá el índice de cada elemento



# Vue.js: Directivas de Bucle



- También se puede usar **v-for** para iterar sobre todas las propiedades de un objeto.
- En el caso de objetos, el segundo y tercer argumento (opcionales) identifican la clave y el índice de la propiedad.

```
<div v-for="(value, key, index) in object">
  {{ index }}. {{ key }}: {{ value }}
</div>
```

**HTML**

```
new Vue({
  el: '#v-for-object',
  data: {
    object: {
      primerNombre: 'John',
      apellido: 'Doe',
      edad: 30
    }
  }
})
```

**JS**



# *Vue.js: Directivas de Bucle*



- Al igual que **v-if**, **v-for** puede ser utilizado en un elemento **<template>**.

```
<ul>
  <template v-for="item in items">
    <li>{{ item.msg }}</li>
    <li class="divider" role="presentation"></li>
  </template>
</ul>
```

**HTML**

- Se puede especificar un rango numérico (por ejemplo de 1 a 10):

```
<div>
  <span v-for="n in 10">{{ n }}</span>
</div>
```

**HTML**



# Vue.js: Directivas de Bucle



- Los filtrados sobre un vector se pueden realizar con **v-for** y una propiedad calculada o un método convencional:

```
<li v-for="n in numerosImpares">{{ n }}</li>
```

**HTML**

```
data: {  
    numeros: [ 1, 2, 3, 4, 5 ]  
,  
computed: {  
    numerosImpares: function () {  
        return this.numeros.filter(function (numero) {  
            return numero % 2 === 0  
        })  
    }  
}
```

**JS**

```
<li v-for="n in even(numeros)">{{ n }}</li>
```

**HTML**

```
data: {  
    numeros: [ 1, 2, 3, 4, 5 ]  
,  
methods: {  
    even: function (numeros) {  
        return numeros.filter(function (numero) {  
            return numero % 2 === 0  
        })  
    }  
}
```

**JS**



# Vue.js: Directivas Condicionales + Bucle



- No se recomienda usar **v-if** junto con **v-for** en el mismo elemento. En su lugar se pueden utilizar propiedades calculadas que hagan el filtrado.

```
<ul>
  <li
    v-for="user in users"
    v-if="user.isActive"
    :key="user.id"
  >
    {{ user.name }}
  </li>
</ul>
```

**HTML  
(MAL)**

```
<ul>
  <li
    v-for="user in activeUsers"
    :key="user.id"
  >
    {{ user.name }}
  </li>
</ul>
```

**HTML  
(BIEN)**



# Vue.js: Directivas Condicionales + Bucle



- También se puede evitar la anidación entre **v-if** y **v-for** moviendo **v-if** a un elemento contenedor de nivel superior.

```
<ul>
  <li
    v-for="user in users"
    v-if="shouldShowUsers"
    :key="user.id"
  >
    {{ user.name }}
  </li>
</ul>
```

**HTML  
(MAL)**

```
<ul v-if="shouldShowUsers">
  <li
    v-for="user in users"
    :key="user.id"
  >
    {{ user.name }}
  </li>
</ul>
```

**HTML  
(BIEN)**



# Vue.js: Limitaciones en la Reactividad



- Vue no detecta cambios en un vector accediendo mediante el índice, o cambiando su tamaño:

```
var vm = new Vue({  
  data: {  
    items: ['a', 'b', 'c']  
  }  
})  
vm.items[1] = 'x' // NO es reactivo  
vm.items.length = 2 // NO es reactivo
```

**JS**

- En su lugar se puede usar `Vue.set(vm.items, indexOfItem, newValue)`
- Otros métodos que actualizan la vista: `push()`, `pop()`, `shift()`, `unshift()`, `splice()`, `sort()`, `reverse()`, `set()`



# Vue.js: Directivas para Eventos



- Con **v-on** se pueden asociar eventos a métodos definidos en las instancias de Vue.
- Algunas directivas (como v-on) pueden tomar un “argumento”, denotado por dos puntos después del nombre de la directiva. En este caso sirve para especificar el evento concreto que se asocia.

```
<div id="app-5">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>
```

**HTML**

```
var app5 = new Vue({
  el: '#app-5',
  data: {
    message: 'Hello Vue.js!'
  },
  methods: {
    reverseMessage: function () {
      this.message = this.message.split('').reverse().join('')
    }
  }
})
```

**JS**



# Vue.js: Directivas para Eventos



- Se puede utilizar la variable **\$event** que hace referencia al objeto evento que se ha producido y que se puede manipular luego en los métodos de la instancia.

```
<button v-on:click="Alertar('El Form no se puede enviar aun.', $event)">  
    Enviar  
</button>
```

**HTML**

```
// ...  
methods: {  
    Alertar: function (message, event) {  
        // ahora tenemos acceso al evento nativo.  
        if (event) event.preventDefault()  
        alert(message)  
    }  
}
```

**JS**



# Vue.js: Modificadores de eventos



- Vue proporciona modificadores de eventos postfijos para evitar tener que llamar a métodos comunes dentro de los manejadores.
- Los modificadores proporcionados son: .stop, .prevent, .capture, .self, .once, .passive

```
<!-- El evento de enviar ya no volverá a cargar la página. -->
<form v-on:submit.prevent="onSubmit"></form>
```

**HTML**

- Existen también modificadores de teclas para comprobar códigos de teclas comunes al escuchar eventos de teclado

```
<!-- solo llame a `vm.submit ()` cuando el `keyCode` es 13 -->
<input v-on:keyup.13="submit">

<!-- También funciona como abreviación. -->
<input @keyup.enter="submit">
```

**HTML**

- **Lista de alias:** .enter, .tab, .delete, .esc, .space, .up, .down, .left, .right



# *Vue.js: Modificadores de eventos*



- Existen modificadores de teclas del sistema que permiten capturar combinaciones de teclas: .ctrl, .alt, .shift, .meta

```
<!-- Alt + C -->
<input @keyup.alt.67="clear">

<!-- Ctrl + Click -->
<div @click.ctrl="doSomething">Do something</div>
```

**HTML**

- Modificadores de eventos del ratón: .left, .right, .middle



# Vue.js: Directivas para enlace bidireccional



- Con **v-model** se pueden asociar eventos a métodos definidos en las instancias de Vue.

```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>
```

**HTML**

```
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

**JS**

- **v-model** se puede utilizar con los tipos básicos de entrada en HTML (text, textarea, number, radio, checkbox, select) y se puede ver como una combinación de v-bind para renderizar un valor seguido de v-on:input para actualizar el objeto JS correspondiente.
- La interpolación en áreas de texto no funciona (`<textarea>{{ text }}</textarea>`). En su lugar debe usarse **v-model**.



# *Vue.js: Modificadores*



- Los modificadores son sufijos especiales indicados por un punto, que indican que una directiva debe estar vinculada de alguna manera especial.
- Por ejemplo, el modificador `.prevent` le dice a la directiva `v-on` que llame a `event.preventDefault()` en el evento activado:

```
<form v-on:submit.prevent="onSubmit"> ... </form>
```

**HTML**



# *Vue.js: Abreviaturas*



- Vue.js proporciona abreviaturas especiales para dos de las directivas más utilizadas, **v-bind** y **v-on**

```
<!-- full syntax -->
<a v-bind:href="url"> ... </a>

<!-- abreviado -->
<a :href="url"> ... </a>
```

**HTML**

```
<!-- full syntax -->
<a v-on:click="doSomething"> ... </a>

<!-- abreviado -->
<a @click="doSomething"> ... </a>
```

**HTML**



# Vue.js: Propiedades Calculadas



- Cuando se debe manipular un dato de forma compleja es preferible no hacerlo directamente en la plantilla por legibilidad
- La manipulación se realiza dentro de un método definido como una propiedad calculada (**computed**) en la instancia correspondiente

```
<div id="example">
  <p>Mensaje original: "{{ message }}"</p>
  <p>Mensaje invertido computado: "{{ reversedMessage }}"</p>
</div>
```

**HTML**

```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hola'
  },
  computed: {
    // un getter computado
    reversedMessage: function () {
      // `this` apunta a la instancia vm
      return this.message.split('').reverse().join('')
    }
  }
})
```

**JS**



- Se puede lograr lo mismo incluyendo un método en la instancia específico para invertir la cadena original
- El uso de propiedades calculadas es más eficiente porque éstas se almacenan en caché según sus dependencias, y sólo se vuelven a evaluar si alguna dependencia ha cambiado



# Vue.js: Propiedades Watched (Observadores)



- Forma genérica de reaccionar a cambios en una instancia
- Es preferible utilizar propiedades calculadas cuando sea posible

```
<div id="demo">{{ fullName }}</div>
```

**HTML**

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar',
    fullName: 'Foo Bar'
  },
  watch: {
    firstName: function (val) {
      this.fullName = val + ' ' + this.lastName
    },
    lastName: function (val) {
      this.fullName = this.firstName + ' ' + val
    }
  }
})
```

**JS**



- Vue permite la definición de componentes personalizados.
- Relacionado con los **custom elements** de HTML5, pero permite más flexibilidad y mejor compatibilidad con navegadores, además de:
  - Flujo de datos de componentes cruzados
  - Comunicación de eventos personalizados
  - Integraciones de herramientas de construcción
- Se puede establecer en la instancia Vue que un componente acepte propiedades (**props**) de forma que se puedan pasar datos entre distintos elementos



# Vue.js: Componentes



```
<div id="app-7">
<ol>
<todo-item
  v-for="item in groceryList"
  v-bind:todo="item"
  v-bind:key="item.id">
</todo-item>
</ol>
</div>
```

**HTML**

```
Vue.component('todo-item', {
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})

var app7 = new Vue({
  el: '#app-7',
  data: {
    groceryList: [
      { id: 0, text: 'Vegetales' },
      { id: 1, text: 'Queso' },
      { id: 2, text: 'Cualquier otra cosa que se supone que los humanos coman' }
    ]
  }
})
```

**JS**



# Vue.js: Componentes



- Los componentes son instancias de Vue y por tanto utilizan las mismas opciones (data, computed, watch, methods, hooks).
- El atributo *data* de los componentes, en lugar de ser un objeto como se ha visto hasta ahora, debe ser una *función que devuelva* un objeto. Esto es así para que cada instancia del componente pueda tener sus datos independientes.

```
// Definir un nuevo componente llamado button-counter
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button v-on:click="count++">Me ha pulsado {{ count }} veces.</button>'
})
```

**JS**

```
<div id="components-demo">
  <button-counter></button-counter>
</div>
```

**HTML**

```
new Vue({ el: '#components-demo' })
```

**JS**



# *Vue.js: Componentes, Organización y Registro*



- Los componentes pueden ser registrados global (como en el ejemplo anterior) o localmente.
- Los componentes globales pueden ser usados en la plantilla de cualquier instancia de Vue raíz creada posteriormente, e incluso dentro de todos los subcomponentes del árbol de componentes de esa instancia de Vue.
- Para utilizar componentes locales se pueden definir como objetos JS y utilizar la opción components:

```
var ComponentA = { /* ... */ }
var ComponentB = { /* ... */ }
var ComponentC = { /* ... */ }
```

**JS**

```
new Vue({
  el: '#app',
  components: {
    'component-a': ComponentA,
    'component-b': ComponentB
  }
})
```

**JS**



# *Vue.js: Componentes, Props*



- Las props son atributos personalizados que se pueden registrar en un componente. Cuando se pasa un valor a un atributo prop, se convierte en una propiedad en esa instancia de componente.
- Podemos usar *v-bind* para pasar propiedades dinámicamente a elementos de arrays cuando no se conoce el contenido exacto que se va a renderizar con anticipación.
- Se pueden utilizar plantillas que evitan tener que enlazar con v-bind individualmente cada campo del objeto a considerar:

```
Vue.component('blog-post', {  
  props: ['post'],  
  template: `  
    <div class="blog-post">  
      <h3>{{ post.title }}</h3>  
      <div v-html="post.content"></div>  
    </div>  
  `,  
})
```

**JS**

```
<blog-post  
  v-for="post in posts"  
  v-bind:key="post.id"  
  v-bind:post="post"  
></blog-post>
```

**HTML**



# Vue.js: Componentes, Props



- Las instancias de Vue proporcionan un sistema de eventos personalizados para enviar mensajes a componentes padre. Para emitir un evento a los padres, podemos llamar al método `$emit`, pasando el nombre del evento:

```
<button v-on:click="$emit('enlarge-text')">  
  Agrandar texto  
</button>
```

**HTML (hijo)**

```
<blog-post  
  ...  
  v-on:enlarge-text="postFontSize += 0.1"  
></blog-post>
```

**HTML (padre)**

- Para emitir un valor específico con un evento podemos usar el segundo parámetro de `$emit` para proporcionar este valor:

```
<button v-on:click="$emit('enlarge-text', 0.1)">  
  Agrandar texto  
</button>
```

**HTML (hijo)**



# Vue.js: Componentes, Props



- Podemos acceder al valor del evento emitido con `$event` en el padre:

```
<blog-post
  ...
  v-on:enlarge-text="postFontSize += $event"
></blog-post>
```

**HTML (padre)**

- O si se maneja el evento con un método se toma el primer argumento del mismo para acceder al valor pasado al evento:

```
<blog-post
  ...
  v-on:enlarge-text="onEnlargeText"
></blog-post>
```

**HTML (padre)**

```
methods: {
  onEnlargeText: function (enlargeAmount) {
    this.postFontSize += enlargeAmount
  }
}
```

**JS**



# *Vue.js: Componentes, Slots*



- Al igual que con los elementos HTML, a menudo es útil poder pasar contenido a un componente, como este:

```
<alert-box>
  Algo ha ocurrido mal.
</alert-box>
```

**HTML**

- Esto se consigue con el elemento `<slot>` de Vue. Simplemente hay que colocar la etiqueta donde queramos que se ubique el contenido que se pase al componente.

```
Vue.component('alert-box', {
  template: `
    <div class="demo-alert-box">
      <strong>Error!</strong>
      <slot></slot>
    </div>
  `
})
```

**JS**



# Vue.js: Componentes dinámicos



- Es posible cambiar de componente de forma dinámica, como por ejemplo para cambiar de pestaña cuando lo seleccione el usuario
- Lo anterior es posible gracias al elemento `<component>` de Vue con el atributo especial `is`:

```
<!-- El componente cambia cuando currentTabComponent cambia -->
<component v-bind:is="currentTabComponent"></component>
```

**HTML**

- Donde el componente a seleccionar (`currentTabComponent` en el ejemplo) debe ser el nombre de un componente registrado, o un objeto de opciones de un componente.



## *Vue.js: Peticiones HTTP (Ajax)*



- Vue.js no propone un procedimiento estándar para la realización de peticiones al servidor.
- Los métodos más popularmente usados consisten en usar la **API fetch** de JS o la librería de cliente **HTTP Axios**.
  - Ambos están basados en **promesas** (objetos con distintos estados para comunicaciones asíncronas que pueden estar listos ahora, en el futuro o nunca)



# *Vue.js: Peticiones HTTP con Axios*



- Instalación: npm install axios
- Se suelen realizar peticiones desde el método created de la instancia

```
<template>
  <div v-if="result" class="content">
    <p>User ID: {{result.userId}}</p>
    <p>Title: {{result.title}}</p>
  </div>
</template>
```

**HTML**

```
import axios from "axios";
...
  data: () => ({
    result: null
  }),
  async created() {
    let response = await axios.get("https://url_del_servidor.com");
    this.result = response.data;
  }
}
```

**JS**



# Vue.js: Peticiones HTTP con Axios



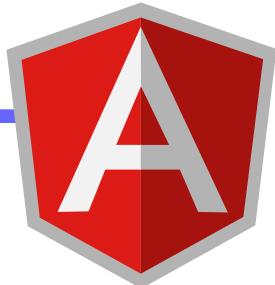
- Peticiones POST o PUT: Como GET, pero pasando un segundo argumento con los datos a enviar

```
methods: {
    async peticion1() {
        let response = await axios.get("https://url_del_servidor/1");
        console.log(response.data);
    },
    async peticion2() {
        let post = {
            title: 'foo',
            body: 'bar',
            userId: 1
        };
        let response = await axios.put("https://url_del_servidor/2", post)
        console.log(response.data);
    }
},
created() {
    this.peticion1();
    this.peticion2();
}
```

**JS**



# Fundamentos ANGULAR



- Framework de cliente Web de código abierto
- Dirigido por Google <https://angular.io/tutorial/>
- **Angular == AngularJS 2+** (AngularJS == AngularJS 1.x)
- Paradigma basado en componentes
  - Angular 1.x se basa en el patrón MVC
- Contribuye a la generación de SPAs utilizando **TypeScript** y proporcionando herramientas
  - **NPM**: Gestión de paquetes de NodeJS
  - **Angular CLI**: Generación, compilación, ejecución, etc. de proyectos Angular
  - **Karma/Protractor**: Tests unitarios y extremo a extremo
  - **TSLint**: Análisis estático TrueScript
  - **Transpilador TypeScript**: convierte a JS en el momento de compilación de la aplicación



# ANGULAR: Primeros pasos con CLI



- Para la creación de una aplicación Web con Angular CLI:
  - **Instalar Node.js:** <https://nodejs.org>
  - Actualización de Node.js (recomendado):  
`npm install -g npm@latest`
  - **Instalar Angular CLI:**  
`npm install -g @angular/cli`
  - **Crear un nuevo proyecto:**  
`ng new`  
(Dentro del directorio correspondiente)
  - **Compilar/Transpilar el proyecto y servirlo a nivel local:**  
`ng serve`  
(Desde el directorio del proyecto)



# ANGULAR: Organización

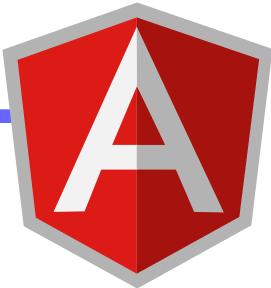


- Al ejecutar `ng new` o `ng new my-App` se crea un nuevo espacio de trabajo y un esqueleto de aplicación de bienvenida. La descripción detallada está en <https://angular.io/guide/file-structure>
- Los archivos fuente están en `src/`
- En `src/app/` están contenidos los archivos que definen los componentes y la lógica de la aplicación
- En `src/assets/` se incluyen imágenes y otros recursos que se copian al construir la aplicación
- La plantilla (página) principal está en `src/index.html`
- El punto de arranque de la aplicación está en `src/main.ts`, compila la aplicación y carga el módulo raíz `AppModule`:

```
import { AppModule } from './app/app.module';
```



# ANGULAR: Organización



- El archivo index.html de la aplicación generado por defecto simplemente inserta un elemento <app-root>, que consiste en la vista asociada al componente raíz

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Prueba</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

**HTML**



# ANGULAR: Organización



- El módulo raíz AppModule está en src/app/app.module.ts y carga al componente raíz. Aquí se declararán los demás módulos también.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

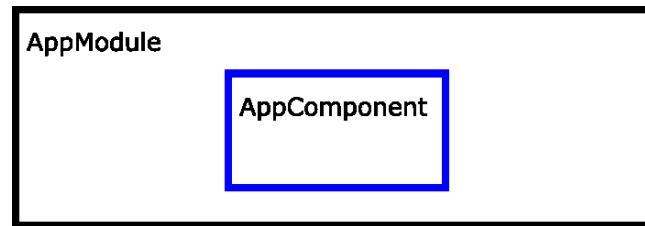
@NgModule({
  declarations: [ AppComponent ],
  imports: [ BrowserModule ],
  providers: [],
  bootstrap: [AppComponent]
})export class AppModule {}
```



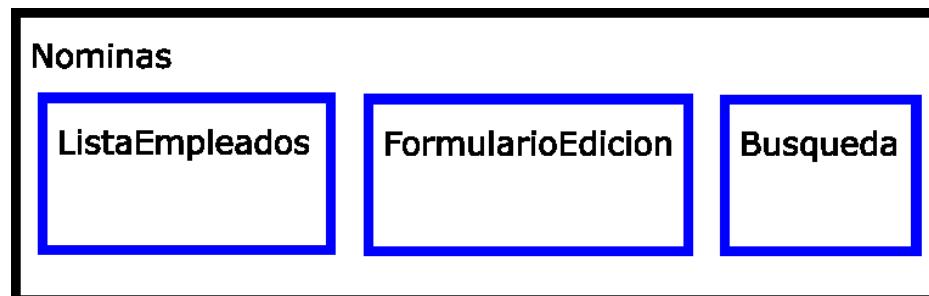
# ANGULAR: Organización



- Una aplicación con Angular se estructura en base a módulos y componentes
- Al menos debe existir un módulo raíz, que a su vez esté asociado al componente raíz



- Según aumente la complejidad de la aplicación se pueden definir nuevos módulos con funcionalidad relacionada con distintos componentes





# ANGULAR: Organización



- Creación de un módulo con CLI:

ng generate module miModulo

(o ng g module miModulo)

- El módulo creado se sitúa dentro de src/app en su propio directorio src/app/MiModulo

- Creación de componentes con CLI:

ng g component miComponente (**en el módulo raíz**)

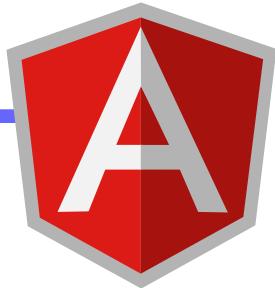
ng g component miModulo/miOtroComponente

- Al crear componentes dentro de módulos se actualiza automáticamente el archivo .ts del módulo, incluyendo a los nuevos componentes.

- Al usar ng generate también se generan los archivos de cada módulo/componente.



# ANGULAR: *Decoradores*



- La declaración de componentes de Angular se basa en la declaración de clases de TypeScript con **decoradores**.
- Un decorador es una característica de metaprogramación que permite añadir información adicional a una clase, método, etc.
- Ejemplo (definición componente Angular):

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

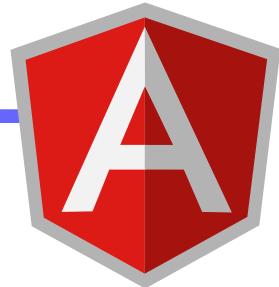
@Component({
  selector: 'product-card',
  template: `<div>
    <p>{{nombre}}</p>
    <p>{{precio}}€</p>
  </div>`
})
export class ProductComponent {
  nombre = "Leche";
  precio = 1.5;
}
```

Decorador para la definición de un selector CSS que define los elementos HTML a los que se aplica el componente, y de la plantilla HTML

Esto representa un módulo que hace uso del decorador **Component** (importado del framework). El módulo define la clase **ProductComponent** que puede ser importada por otros ya que ha sido marcada como “exportada”



# ANGULAR: Módulos

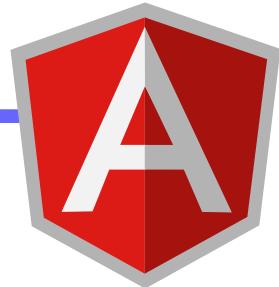


- Las aplicaciones se componen de módulos en su nivel superior.
- Un módulo es una clase que representa una serie de funcionalidades relacionadas entre sí.
- Cada aplicación tiene al menos el módulo raíz ( AppModule ).
- Los módulos se representan como clases anotadas con el decorador @NgModule
- Esquema general:

```
@NgModule({
    imports: [ /* Módulos importados. */ ],
    providers: [ /* Constructores de servicios. */ ],
    declarations: [ /* Componentes, directivas y tuberías. */ ],
    exports: [ /* Declaraciones visibles en las plantillas. */ ],
    bootstrap: [ /* Componente raíz del módulo. */ ]
})
export class Module { }
```



# ANGULAR: Módulos



`imports`: lista de módulos cuyas clases exportadas pueden ser utilizadas en los componentes del módulo actual.

`providers`: lista de constructores de servicios que son susceptibles de ser utilizados para la inyección de dependencias. Estos servicios serán accesibles globalmente, incluso desde otros módulos.

`declarations`: Clases correspondientes a componentes, directivas y tuberías que pertenecen al módulo. Los componentes, directivas y tuberías se conocen en su conjunto como **vistas**.

`exports`: Subconjunto de las clases especificadas en `declarations` que pueden ser utilizados en componentes de otros módulos (previa importación en dichos módulos).

`bootstrap`: Propiedad exclusiva del módulo raíz (`AppModule`). Representa al componente raíz, que aloja a las demás vistas.

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

main.ts



Código mínimo para arrancar el módulo raíz



# ANGULAR: Organización



- El componente raíz AppComponent está en src/app/app.component.ts y carga la vista (plantilla) principal de la aplicación, a partir de la que se van cargando el resto de los componentes.
- La plantilla del componente raíz está en src/app/app.component.html

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



# ANGULAR: Componentes y plantillas



Ejemplo: Definición de componente y uso de interpolación en la plantilla HTML

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'hello-comp',
  template: `
    <h1>Bienvenido</h1>
    <p>¡Hola, {{name}}!</p>
  `
})
export class HelloComponent implements OnInit {
  name: String;

  ngOnInit() {
    this.name = this.getNombre();
  }

  getNombre() {
    return "Laura";
  }
}
```

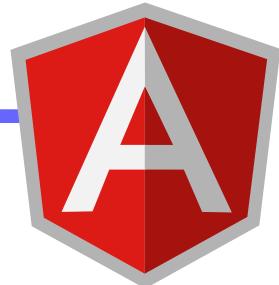
Con selector se define el componente en las vistas. El del componente raíz es **app-root**

La plantilla se puede definir en el componente (**template**) o en un archivo aparte, en casos de mayor complejidad (**templateUrl**)

Esto es un enlazado de datos unidireccional (**interpolación**) que permite la introducción de cadenas procesadas en el código del componente dentro de la plantilla



# ANGULAR: Directiva *ngModel*



- Permite establecer enlaces unidireccionales y bidireccionales entre propiedades de clases y valores mostrados en elementos de entrada HTML (input, textarea, etc).
- Se debe importar FormsModule en el módulo que contenga al componente que lo utilice:

```
import {FormsModule} from '@angular/forms';
```

- Enlace unidireccional: `<input type="text" [ngModel]="nombre">`, donde nombre es un atributo del componente que incluye la plantilla
- Enlace bidireccional: `<input type="text" [(ngModel)]="nombre">`, donde nombre es un atributo del componente que incluye la plantilla



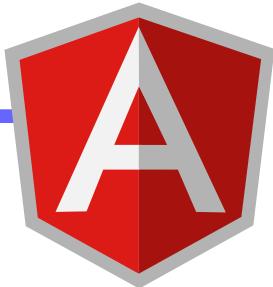
# ANGULAR: Enlaces



- La interpolación {{ }} es un tipo de enlace unidireccional de texto
- En Angular se pueden usar los corchetes [] para realizar un enlace unidireccional a una propiedad
- Para trabajar con plantillas y concatenando texto con contenido dinámico (Ej. src="http://{{url}}") se debe usar interpolación
- Enlace bidireccional: se consigue con [( )] (Ejemplo: <app-sizer [(size)]="fontSizePx"></app-sizer>)



# ANGULAR: Bindings



## Ejemplo: Enlazado de propiedad, unidireccional y bidireccional

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

@Component({
  selector: 'my-image',
  template: `
    <img [src]="imageUrl" alt="Imagen personalizada" width="200" />
    <div>
      <label for="input-url">URL de la imagen:</label>
      <input id="input-url" type="url" [(ngModel)]="imageUrl" />
      <button (click)="setDefault()">Imagen predeterminada</button>
    </div>
  `
})
export class ImageComponent {
  static DEFAULT_IMAGE =
    "https://angular.io/assets/images/logos/angular/angular.png";
  imageUrl: String;

  constructor() {
    this.imageUrl = ImageComponent.DEFAULT_IMAGE;
  }

  setDefault() {
    this.imageUrl = ImageComponent.DEFAULT_IMAGE;
  }
}
```

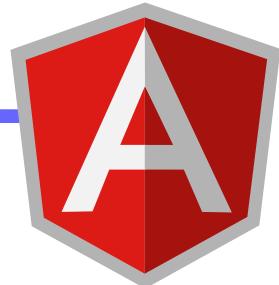
Enlace de una dirección a la propiedad src del elemento img

Enlace de dos direcciones en el elemento input. Se utiliza la directiva ngModel (módulo FormsModule)

Se define un manejador para el evento click del botón



# ANGULAR: Directivas



Las directivas son clases creadas con el decorador `@Directive`, que se utilizan para transformar el DOM del elemento donde se aplican.

- Estructurales (`*ngFor`, `*ngIf`, ...): Modifican el DOM de un componente
- De atributo (`ngModel`, `ngStyle`, ...): Modifican apariencia/comportamiento

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-fruits',
  template: `
    <h1>Listado de frutas</h1>
    <ul>
      <li *ngFor="let fruit of fruits" (click)="selectFruit(fruit)">
        {{fruit}}
      </li>
    </ul>
    <p *ngIf="selectedFruit">
      Has hecho click sobre la fruta: {{selectedFruit}}
    </p>
  `})

```

```
class FruitsComponent {
  selectedFruit: string;
  fruits: Array<string>;
  constructor() {
    this.fruits = ['Pera', 'Melón', 'Limón', 'Sandía'];
    this.selectedFruit = null;
  }
  selectFruit(fruit: string) {
    this.selectedFruit = fruit;
  }
}
```



# ANGULAR: Eventos



- En Angular los eventos se pueden registrar y capturar envolviéndolos entre paréntesis ()
- Esto es equivalente a llamar a `addEventListener()`

```
@Component({
  selector: 'app-root',
  template: `<button (click)="clickEvent()">Click
Me</button>`
})
export class AppComponent{
  clickEvent() {
    console.log('You clicked me')
  }
}
```

- Con más de un evento:

```
<button (click)="clickEvent()" (mouseenter)="mouseEnterEvent()">Click Me</button>
```



# ANGULAR: Comunicación entre Comps.



- En Angular se parte del componente raíz y se construye una jerarquía en la aplicación Web
- Al crear componentes hijos se pueden pasar valores a atributos desde el componente padre
- El decorador @Input() indica que un componente hijo puede recibir datos de su componente padre

componente-hijo.component.ts

```
import { Component, Input } from '@angular/core'; // Se importa
export class ComponenteHijo {
  @Input() item: string; // se decora con @Input()
}
```

```
<p> Mensaje: {{item}} </p>
```

componente-hijo.component.html

componente-padre.component.html

```
<app-componente-hijo [item] = "msgActual"></app-componente-hijo>
```

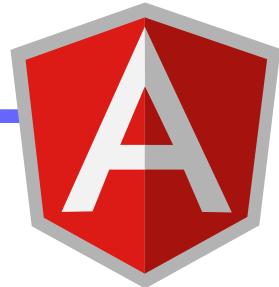
```
export class ComponentePadre { msgActual = 'Soy tu padre'; }
```

componente-padre.component.ts

(app-componente-hijo es el selector del componente hijo)



# ANGULAR: Comunicación entre Comps.



- Para pasar datos del componente hijo al padre se utiliza el decorador `@Output()` y eventos que son capturados por el padre.
- En el componente hijo se debe importar `Output` y `EventEmitter` (para emitir eventos), definir el nuevo evento con el decorador, y un método que lance el evento:

componente-hijo.component.ts

```
import { Output, EventEmitter } from '@angular/core';
...
export class ComponenteHijo {

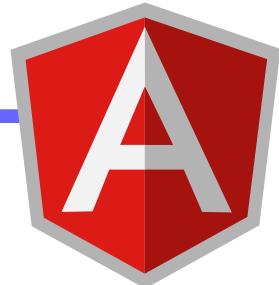
  @Output() unEvento = new EventEmitter<string>();

  lanzaEvento(valor: string) {
    this.unEvento.emit(valor);
  }

}
```



# ANGULAR: Comunicación entre Comps.



- En la plantilla del componente hijo (como ejemplo) se captura el valor de un input para pasárselo al padre:

componente-hijo.component.html

```
<label>Escribe algo: <input #unDato></label> <button  
(click)="lanzaEvento(unDato.value)">Añadir</button>
```

(#unDato es una variable de plantilla, que permite utilizar ese contenido en otras partes de la plantilla)

- El padre debe capturar el nuevo evento lanzado por el hijo, y puede obtener el dato enviado del propio evento:

componente-padre.component.html

```
<app-componente-hijo (unEvento)="manejadorPadre($event)"></ app-componente-hijo>
```

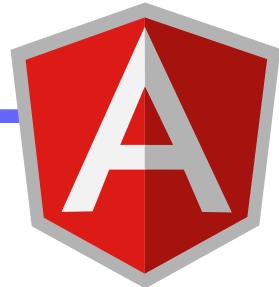
(app-componente-hijo es el selector del componente hijo)

componente-padre.component.ts

```
export class ComponentePadre {  
    ...  
    manejadorPadre(cadenaIn: string) {  
        ... // Procesa la cadena recibida del hijo  
    }  
}
```



# ANGULAR: Comunicación entre Comps.



- Un componente padre puede llamar a métodos de componentes hijos desde su plantilla utilizando variables de plantilla (identificadas comenzando por # )

componente-padre.component.html

```
<app-componente-hijo #miHijo></ app-componente-hijo>
<button (click)="miHijo.unMetodo()">Soy un boton</button>
```



# ANGULAR: Servicios



Un servicio es un componente de una aplicación que proporciona funciones auxiliares

- Acceso a fuentes de datos, registro de eventos, etc.

Funcionalidad compartida por varios componentes

Son clases en ficheros `nombreServicio.service.ts`

Si se decoran con `@Injectable()` se permite que pueda ser objeto de inyección de dependencias

- Proporcionar a los objetos las dependencias que necesiten para funcionar, abstrayendo a los objetos dependientes de su creación/obtención.



# ANGULAR: Servicios



- En los servicios se delegan las responsabilidades de acceso a datos (peticiones, envíos,...), así como parte de la lógica de negocio
- Por ejemplo: responsabilidad de cargar datos desde un servidor Web
- Creación de un servicio desde CLI:

```
ng generate service MiServicio
```

- Los módulos deben listar los servicios que utilizan en su campo providers, excepto los módulos que ya vengan en el campo providedIn del propio servicio

`mi-servicio.service.ts`

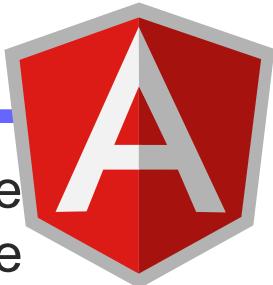
```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class MiServicio {

  constructor() { }
}
```



# ANGULAR: Servicios



- Pasando un atributo al constructor de un componente se puede injectar un servicio, de manera que se dispone de un atributo que no ha sido creado por el propio componente
- Modificación del servicio anteriormente creado:

**mi-servicio.service.ts**

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

@Injectable({
  providedIn: 'root'
})
export class MiServicio {
  miMetodo() {return "Mensaje"; }
  constructor() { }
}
```



# ANGULAR: Servicios



- Modificación del componente raíz para consumir el servicio MiServicio:

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { MiServicio } from './mi-servicio.service';

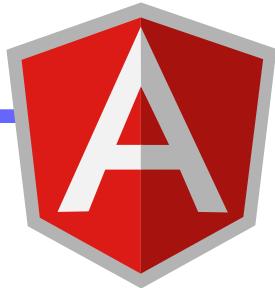
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit{
  texto = null;

  constructor(private atribServicio: MiServicio) {
  }

  ngOnInit() {
    this.texto=this.atribServicio.miMetodo();
  }
}
```



# ANGULAR: Servicios y peticiones



- Para realizar peticiones AJAX se puede utilizar el servicio HttpClient
- Se debe importar el módulo HttpClientModule

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import {HttpClientModule} from '@angular/common/http';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



# ANGULAR: Servicios y peticiones



- Se puede utilizar ahora este servicio directamente para realizar una petición a un servidor desde un componente (por ejemplo el raíz):

app.component.ts

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

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

  constructor(private http: HttpClient) { }

  ngOnInit() {
    this.http.get("http://urlDelServidor.com/obtenerDatos.php")
      .subscribe(
        result => {
          this.datos = result;
        },
        error => {
          console.log('Hay problemas');
        }
      );
  }
}
```



# ANGULAR: Servicios y peticiones



- Puede ser más conveniente mover el código donde se realizan las peticiones a un servicio propio:

mi-servicio.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class MiServicio {

  constructor(private http: HttpClient) { }

  retornar() {
    return this.http.get("http://urlDelServidor/ObtenerDatos.php");
  }
}
```



# ANGULAR: Servicios y peticiones



- ... y utilizar ahora este servicio:

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { MiServicio } from './mi-servicio.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit{

  datos = null;

  constructor(private datosServicio: MiServicio) {}

  ngOnInit() {
    this.datosServicio.retornar()
      .subscribe( result => this.datos = result)
  }
}
```



# ANGULAR: Tuberías (Pipes)



Las tuberías son tareas sencillas que toman una entrada y la transforman, dándole formato para mostrar los datos.

- Se pueden utilizar cuando se hace uso de enlazado de datos en un sentido.

Se pueden crear con el decorador `@Pipe`. Algunas predefinidas:

- `DatePipe`: Da formato a un objeto Date de JS, pudiendo elegir el formato concreto para mostrar.
- `JsonPipe`: Convierte un objeto JS a JSON.
- `UpperCasePipe` y `LowerCasePipe`: Convierten una cadena de entrada a mayúsculas y minúsculas, respectivamente.

<https://angular.io/guide/pipes>



# ANGULAR: Tuberías



- En este ejemplo se utiliza el operador | para transformar una entrada utilizando una tubería, a la cual se hace referencia mediante el nombre date
- date es el selector de la tubería DatePipe

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

@Component({
  selector: 'my-date',
  template: `
    <h1>Formato de fechas</h1>
    <div>
      <label for="day-input">Selecciona el día de agosto:</label>
      <input type="number" min="1" max="31" step="1" value="1"
        (change)="selectDay($event.target.value)" />
    </div>
    <p>Fecha con formato: {{ theDate | date:'fullDate' }}</p>
  `
})
```

```
export class DateComponent {
  theDay: number;
  theDate: Date;

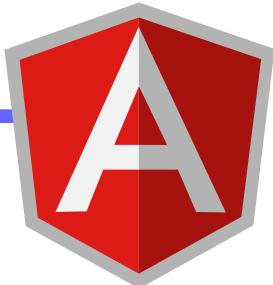
  constructor() {
    this.selectDay(1);
  }

  selectDay(day: number) {
    this.theDay = day;
    this.theDate = new Date(2017, 7, day);
  }
}
```

- Se pasan parámetros a tuberías separándolos con el carácter :
- En este caso el parámetro es la fecha en formato largo



# ANGULAR: Rutas (*Routing*)



- Angular proporciona un módulo especial que se puede utilizar si se desea incorporar rutas en una aplicación Web (por ejemplo: <https://miAplicacion.com/alumnos>, <https://miAplicacion.com/admin>, etc.)
- La vista dependerá de la ruta especificada
- Al crear el proyecto se debe especificar que se desea incluir routing, o bien hacer

```
ng new miProyecto --routing
```

- Se genera el archivo `app-routing.module.ts`, donde debemos especificar las rutas que deseamos definir y el componente a asociar a cada ruta



# ANGULAR: Rutas (Routing)



- Ejemplo con 2 rutas:

app-router.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { Componente1 } from './componente1/componente1.component';
import { Componente2 } from './componente2/componente2.component';

const routes: Routes = [
  {
    path:'ruta1',
    component:Componente1
  },
  {
    path:'ruta2',
    component:Componente2
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```



# ANGULAR: Rutas (*Routing*)



- En app.module.ts importamos AppRoutingModule:

app.module.ts

```
...
import { AppRoutingModule } from './app-routing.module';
import { Componente1 } from './componente1/componente1.component';
import { Componente2 } from './componente2/componente2.component';
...

@NgModule({
  declarations: [
    AppComponent,
    Componente1,
    Componente2,
    ...
  ],
  imports: [
    ...,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



# ANGULAR: Rutas (*Routing*)



- ... y en la plantilla del componente raíz se pueden definir hipervínculos a las rutas con la propiedad `routerLink`, además de un elemento `router-outlet` para indicar dónde mostrar la componente que se especifica en la ruta configurada en `app-routing.module.ts`

`app.component.html`

```
<div style="text-align:center">
  <a routerLink="/ruta1">Enlace a ruta 1</a> -
  <a routerLink="/ruta2">Enlace a ruta 2</a>
  <div>
    <router-outlet></router-outlet>
  </div>
</div>
```



# Usando librerías JS desde Angular



## - Opción 1:

- Añadir línea con la fuente del script en `index.html`

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js">
```

- Declarar librerías como tipo `any` donde se quieran usar
- Llamar a las funciones desde TypeScript

### **app.component.ts**

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

declare var jQuery:any;
declare var $:any;

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent{
  texto = 'Usando jQuery UI con Angular';

  public mostrarTexto(){
    $(".texto").slideToggle();
  }
}
```

### **app.component.html**

```
<h1 class="texto" style="display:none;">
  {{texto}}
</h1>

<button (click)="mostrarTexto()">Mostrar</button>
```



# Usando librerías JS desde Angular



## - Opción 2:

- Instalar la librería mediante NPM

```
npm install -save jquery
```

- Instalar el archivo de declaración de tipos

```
npm install -D @types/jquery
```

- Importar la librería

```
import * as $ from 'jquery';
```

- A partir de aquí se puede usar como en el ejemplo anterior. Este procedimiento proporciona una referencia tipada a jQuery, en lugar de sin tipar (`any`)



**Fin**