Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

JavaScript

Programación en el lado del cliente

Raúl Montes T.


PEOPLE SAY I DON'T CARE...
BUT I DO.


HEY I JUST CLEANED THE BUGS FROM YOUR WINDOWS
WHAT'S THAT NERF GUN FOR?
WWW.IMGHUT.COM
FEEL IT - CREATE IT - SHARE IT
imgHUT


TRIED TO HELP SOMEONE STUCK IN A CAGE
GOT SHOT WITH A SPEAR GUN
WeKnowMemes

i smell blood i should rescue the injured - Misunderstood Shark

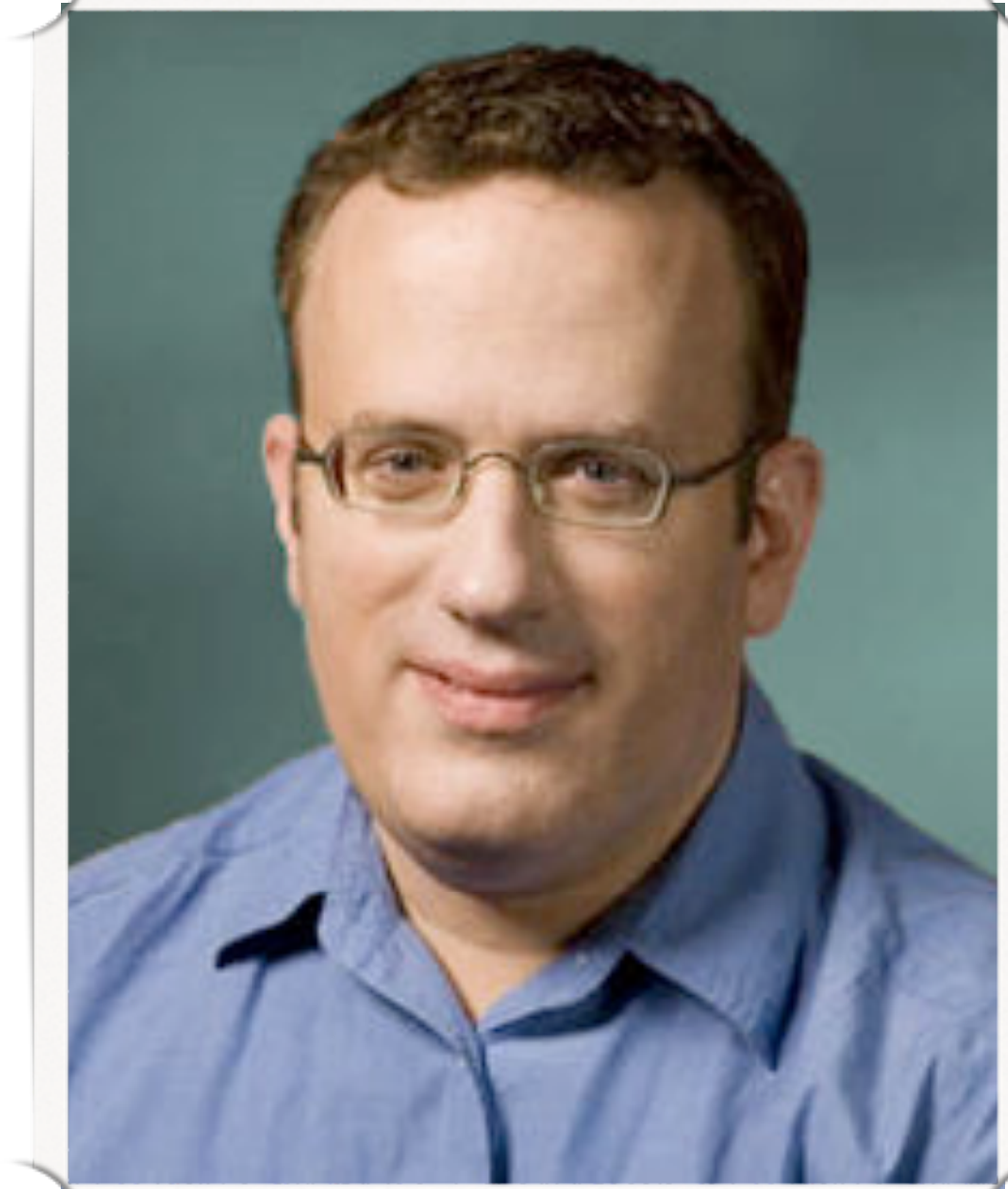

Everyone hates him during 7 movies
Turns out to be a good guy
MEMEBASE.com

JavaScript

I'm not BAD...
I'm just
MISUNDERSTOOD!

JavaScript ≠ Java

Se parecen tanto como casilla a silla...



Brendan Eich

1995

Netscape

Nació en medio de las...

Mocha (codename)

LiveScript (1995) - Netscape 2.0 betas

JavaScript (1996)

JScript (Microsoft)

ECMAScript (ECMA-262, 1997)

ECMAScript (ECMA-262, 1997)

ECMAScript 2nd Edition, 1999

ECMAScript 3rd Edition, 1999

ECMAScript 5th Edition, 2009

ECMAScript 5.1, 2011

ECMAScript 6th Edition, Junio 2015

JavaScript is a **dynamically** typed language

JavaScript is a **weakly** typed language

JavaScript es **multiparadigma**

object oriented

imperative

functional

JavaScript is an Object Oriented language

JavaScript doesn't have classes

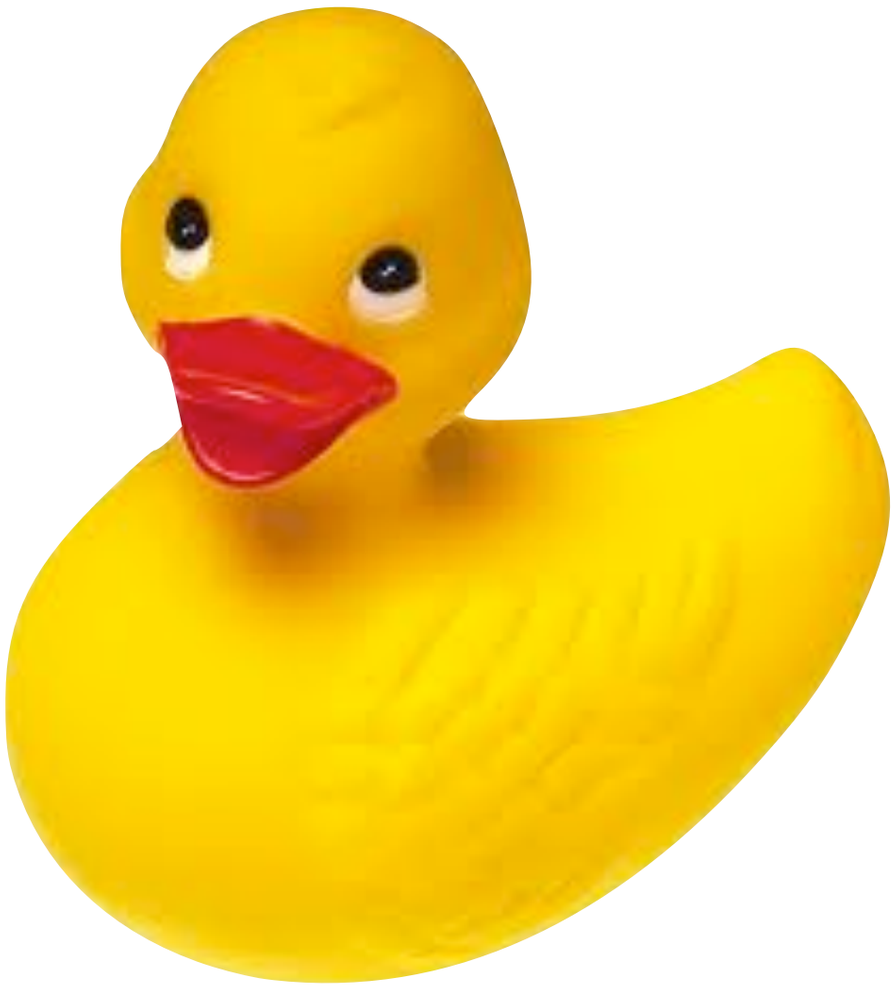
JavaScript has prototypes

JavaScript has First class functions

... y con closures

Las variables se declaran con **var**

Sino, pertenecen al Global Object



Duck Typing

El “Hello World!”

```
document.write("Hello World!");
alert("Hello World!");
console.log("Hello World!");
```

Dinamically Typed...

```
var foo = 1;
foo = true;
foo = "Texto";
```

Weak Typing...

```
0 == ''                 // true
0 == '0'                // true
'' == '0'               // false
false == 'false'        // false
false == '0'            // true
false == undefined      // false
false == null           // false
null == undefined       // true
' \t\r\n ' == 0         // true
```

Recomendación: usar **===** y **!==** en lugar de == y !=

Si el tipo no es el mismo, retornan false de inmediato, por lo que son menos confusos y propensos a errores

Conversión rápida con operadores

```
+"100" -> 100
!!"100" -> boolean true
!!0 -> boolean false
```

If/else if/else y while

```
if(condicion) {
  // sentencias...
} else if(condicion 2) {
  // sentencias...
} else {
  // sentencias...
}

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

For y for..in

```
for(var i = 0; i < object.length; i++) {
  // sentencias...
}
for(var prop in object) {
  // en prop tendremos cada una de las
  // propiedades/atributos de object
}
```

OR y AND
como Default y Guard

```
var foo = bar && bar.length;
// si bar es interpretable a false no se pedirá el largo

foo = bar || "default";
// si bar es interpretable a false se usará el valor default

function ejemplo(param1) {
  var valorParam1 = param1 || "valor default para param1"
}
```

Functions

```
// ambas son equivalentes
function ejemplo() {
  alert("Hola!");
}
var ejemplo = function() {
  alert("Hola!");
}
```

Function params

```
// - los params no identifican a la funcion
// - se pueden entregar más o menos params de los declarados
// - si uno no se entrega => valor undefined
// - si se entrega uno no declarado, se ignora
function ejemplo(prefijo, nombre) {
  alert("Hola, " + prefijo + " " + nombre + "!");
}

// se pueden obtener todos los params entregados con arguments
function ejemplo() {
  if (arguments.length > 1) {
    alert("Hola, " + arguments[0] + " " + arguments[1] + "!");
  } else {
    alert("Hola, " + arguments[0] + "!");
  }
  return arguments.length;
}
```

Arreglos

```
foo = []; // creamos un arreglo
foo[0] = 1;
foo[1] = 2;
foo[10] = "diez"; // se "rellenan" los espacios con undefined
foo.push("chao"); // agregado al final...pop saca del comienzo, como un stack
foo.splice(index, quantity); // elimina, desde index, quantity elementos
foo = new Array(); // otra forma de crear un arreglo
```

Objetos

```
foo = {}; // creamos un objeto
// son solo key/value pairs
// las keys pueden ser texto libre
foo = {uno: 1, dos: 2, "ciento ocho": 108}
// las keys se pueden acceder como en un arreglo asociativo
foo["uno"] -> 1
foo["ciento ocho"] -> 108
// o como "dot notation" para las keys simples
foo.uno -> 1
```

Métodos

```
// los métodos son sólo funciones asignadas
// como una de las propiedades de un objeto
foo.metodo = function() {
  alert("Método de instancia: " + this.uno);
}
// lo ejecutamos
// this será una referencia al objeto al cual se le llama
foo.metodo();
```

Otra forma de llamar funciones...

```
var f = function() {
  // cuerpo de la función
};

// "call" llama a la función y permite elegir el "this" que usará
// y además entregar los params como en una llamada normal
f.call(referenciaAThis, param1, param2, ...);
// "apply" es lo mismo, pero los params se entregan en un Array
f.apply(referenciaAThis, paramsArray);
```

Scope

```
// scope de función, no de bloque
function f() {
  var v1 = 1;
  if (v1 > 0) {
    var v2 = 2;
  }
  console.log(v2);
}
f();  // muestra 2, funciona
```

```
// el scope contiene el scope externo
function f1() {
  var v1 = 1;
  function f2() {
    console.log(v1);
  }
  f2();
}
f1();  // muestra 1
```

Closure

El **closure** de una función contiene todas las variables existentes en el scope en el cual fue declarada la función

```
function f1() {
  var v1 = 1;
  var f2 = function() {
    console.log(v1);
  };
  return f2;
}
var unaFuncion = f1();
unaFuncion(); -> muestra 1
```

La variable local v1 forma parte del closure de f2

Constructores

Cualquier función puede ser usada como **Constructor** de objetos. Tendrá a this como referencia al objeto que se está construyendo. Se crea el objeto con new

```
function Perro(nombre, raza) {
  this.nombre = nombre;
  this.raza = raza;
}
var perro = new Perro("Blacky", "Beagle");
```

Construir objetos con métodos

```
function Perro(nombre, raza) {
  this.nombre = nombre;
  this.raza = raza;
  this.habla = function(veces) {
    while (veces-- > 0) {
      console.log("Guau!");
    }
  }
}
var perro = new Perro("Blacky", "Beagle");
perro.habla(5);
```

Y esto?

```
var perro1 = new Perro("Pluto", "QuienSabe");
var perro2 = new Perro("Snoopy", "Beagle"); // sí, es un beagle...

perro1.habla === perro2.habla -> false
// no son el mismo método... estamos creando la función cada vez
// un poco tonto o no?
```

Prototype

Todos los objetos (salvo el objeto base) tienen un **prototipo**.

El **prototipo** es... un objeto... que tiene prototipo...

Las propiedades de un objeto se buscarán en el objeto mismo y, si no están ahí, en su prototipo, y prototipo del prototipo y... => **herencia**!

Prototype

Todas las funciones, además de su prototipo como objeto, tienen un prototype para asignárselo a los objetos que construyen

```
var perro1 = new Perro("Pluto", "QuienSabe");
// para obtener el prototipo de un objeto
// sólo la segunda es 100% segura en browsers "no modernos" (sí, IE... :P)
perro.__proto__, perro.constructor.prototype, Object.getPrototypeOf(perro)
// para obtener el prototipo que una función asigna a sus objetos:
Perro.prototype
// este mismo prototipo es el que se asigna a los objetos
Perro.prototype === perro1.__proto__ -> true
```

Prototype

Entonces, si modificamos el prototype que se le asigna a los objetos...

```
Perro.prototype.habla = function(veces) {
  while(veces-- > 0) {
    console.log(this.nombre);
  }
}
perro1.habla === perro2.habla -> true
```

El objeto función es el mismo, pero dependiendo desde donde se llame, **this** será un diferente objeto :-)

Herencia con prototipos

```
function Mamifero(nombre) {
  this.nombre = nombre;
}
Mamifero.prototype.habla = function(veces) { console.log("..."); }
// subclase de Mamifero
function Perro(nombre, raza) {
  Mamifero.call(this, nombre); // llamamos al constructor de la super "clase"
  // (como llamar a un "super()"
  this.raza = raza;
}
// hacemos que el prototipo que asigne este constructor sea un objeto de la superclase
// así los Perros tendrán todos los métodos de los Mamíferos y además podremos
// agregar/sobreescribir otras propiedades
// Además asignamos Perro como el constructor del prototipo
Perro.prototype = new Mamifero();
Perro.prototype.constructor = Perro;
var perro = new Perro("Odie", "Teckel");
perro.habla === perro1.habla -> true
PerroEducado.prototype.habla = function(veces) {
  while(veces-- > 0) {
    console.log(this.nombre);
  }
};
perro.habla === mamifero.habla -> false
```

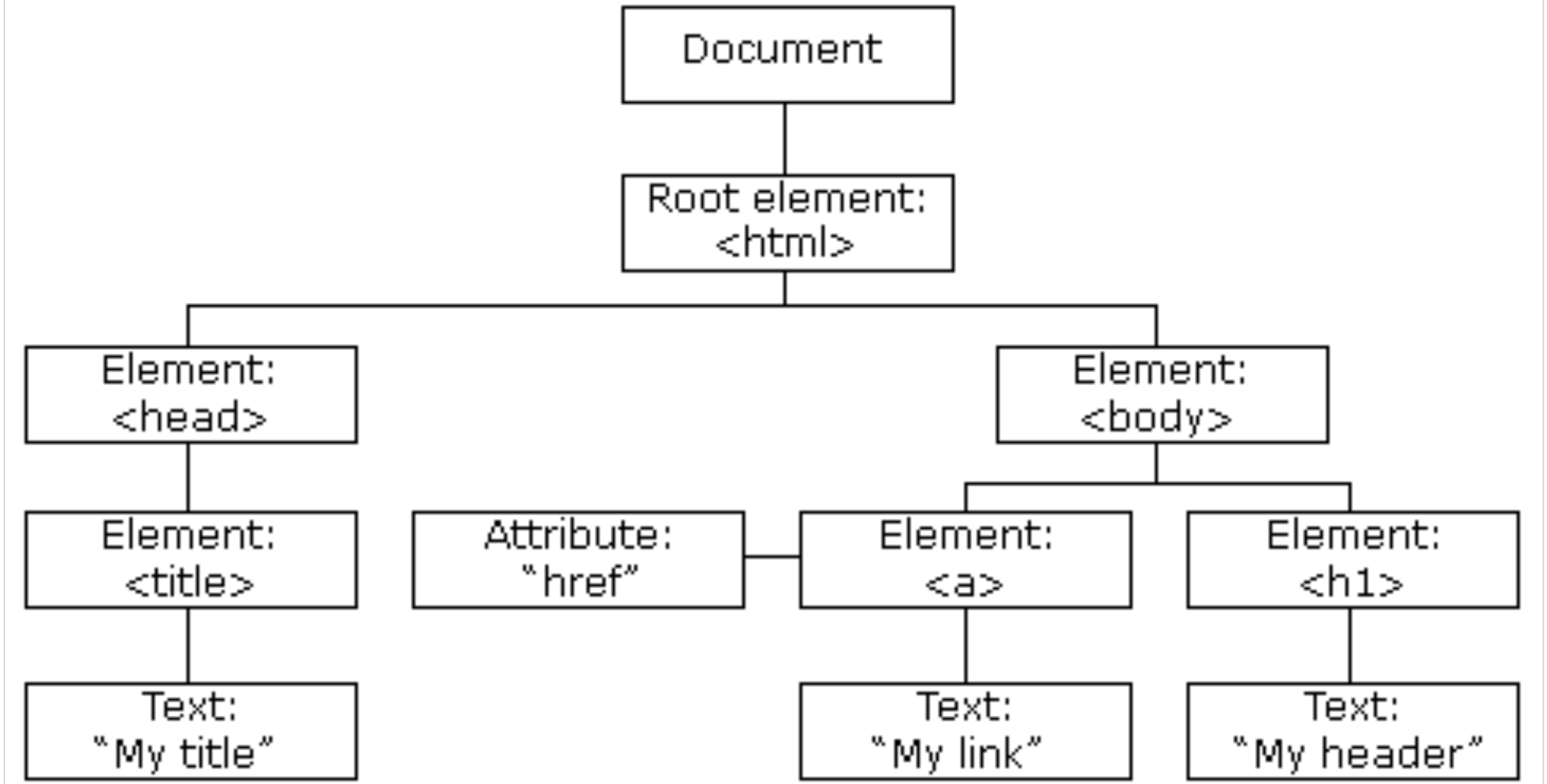
Atributos/Métodos privados, privilegiados y públicos

```
var Perro = function(nombre, raza) {
  // variables locales como nombre o raza, independiente
  // de si almacenan una referencia a un objeto cualquiera
  // o una función, serán propiedades de visibilidad privada, sólo
  // accesibles por quien tenga estas variables en su scope/closure

  // funciones definidas dentro del constructor pero
  // asignadas a una propiedad del objeto resultante (this),
  // serán de visibilidad "privilegiada" (de acceso público,
  // pero con acceso a propiedades privadas)
  this.getNombre = function() {return nombre;}

}

// propiedades definidas al objeto o a su prototipo pero fuera de la
// función constructora, serán de visibilidad pública, pero no tendrán
// acceso a propiedades privadas (pero sí a privilegiadas)
Perro.prototype.ladra = function(veces) {
  while(veces-- > 0) {
    console.log(this.getNombre() + ": Guau!");
  }
}
```

¿Y qué hacemos con JavaScript?

Su interacción con el browser la realiza principalmente gracias al...

Document Object Model



... y sus eventos...

¿Y cómo hacemos interactuar JavaScript con un documento HTML?

Esta es la forma de **NO** usar JavaScript con HTML

```
<a href="ejemplo2.html" id=“button"
onclick="alert('Ehh! me hiciste click!!')">Click me!!!!</a>
```

Así como evitamos mezclar estilos con HTML, también debemos evitar mezclar comportamiento con la estructura/contenido

Esto se conoce como **unobtrusive** JavaScript

Incluimos el script en el documento HTML, dentro de <head></head>

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

Y en ejemplo.js

```
// escuchamos el evento de carga del DOM
// sino, aún no existirá el link en el DOM pues este script
// se ejecuta cuando se está cargando el HEAD
document.addEventListener('DOMContentLoaded', function() {
  var link = document.getElementById('button');
  // al handler del evento se le entrega el evento gatillado
  link.addEventListener('click', function(e) {
    alert("Ehh! me hiciste click!");
    // impedimos que se siga con la acción normal del evento
    // que en este caso es ir al href del anchor
    e.preventDefault();
  });
});
```

Mediante DOM se manipula TODO el documento

```
document // referencia al documento
document.childNodes // nodos hijos, de cualquier tipo
document.childNodes[0] // Doctype
document.childNodes[1] // nodo <html>
// nodos hijos de <html> pero sólo de tipo Element
document.childNodes[1].children
// para encontrar un elemento por su atributo id
var elem = document.getElementById('idDelElemento')
// propiedades y métodos de un nodo
elem.nodeName // nombre del nodo. Si es un elemento -> DIV, A, etc.
elem.nodeType // número que representa el tipo (1 -> Elemento, 3 -> Texto...)
elem.parentNode // nodo padre
elem.nextSibling, elem.previousSibling // nodos hermano
document.createElement() // para crear nuevos nodos de tipo elemento
document.createTextNode() // crea nodos texto
elem.appendChild(child)  // agrega un nodo al final de los hijos
elem.cloneNode() // crea una copia de un elemento
elem.removeChild(child) // elimina el hijo indicado
elem.set/get/removeAttribute() // cambia/crea/obtiene/elimina atributos
// se puede acceder a todas las propiedades de estilo
elem.style.etc // guiones se cambian por camelCase
```

Pero hoy en día se suelen usar librerías para, además de extender el lenguaje, lidiar con las diferencias entre diferentes browsers.

Nosotros usaremos la más popular de todas...


jQuery
write less, do more.

Sólo agrega una función...

jQuery (o $, para los amigos)

```
$(funcion); // ejecuta la función cuando se carga el DOM (DOMContentLoaded)
$(selectorCSSConEsteroides); // entrega un objeto jQuery
var h1s = $('h1'); -> [primerH1, segundoH1, ...]
// el objeto jQuery tiene muuuuchos métodos útiles
h1s.hide();
h1s.append("contenido a agregar");
// ... y muuuuuuuuchos más... ver documentación
// además, casi siempre devuelven el mismo objeto jQuery, para chaining pattern
h1s.show().addClass('nuevaClase');
```

Así, lo anterior nos queda...

```
$(function() {
  $('#button').on('click', function(event) {
    alert('Ehhh!! me hiciste click!!!');
    event.preventDefault();
  });
});
```

Un uso común de JavaScript es client side validation

Por ejemplo...

```
var $form = $('#form');
$form.on('submit', function(e) {
  // revisar los valores de cada campo
  var $field = $form.find('#first_name');
  var value = $field.val();
  // realizar las validaciones
  // si hay un error, mostrarlo mediante manipulación de DOM
  // y además impedir el envío del formulario
  if (error) {
    e.preventDefault();
  }
});
```

¿Otro uso común? R: lo que, de hecho, popularizó enormemente este lenguaje... **AJAX**

Asynchronous JavaScript and XML

... aunque hoy en día XML está presente, generalmente, sólo en el nombre...

AJAX

Es un conjunto de tecnologías:

- HTML y CSS para la vista
- DOM para interactuar con la vista
- XML, JSON, HTML o JS para intercambio de info
- El XMLHTTPRequest para requests asíncronos
- JavaScript para unirlos a todos... ~~y atarlos en las tinieblas~~

AJAX

En lugar de cambiar el estado completo de la app con un request normal, hacemos un request especial de manera asíncrona, y cuando la respuesta llega, actualizamos sólo lo que debiera cambiar en la vista.

La actualización de la vista es más rápida y no se necesita esperar por el request (la app no se bloquea). Las apps se sienten más "responsivas"

Ajax en Rails

Helpers link_to y form_tag/form_for tienen opción :remote

```
<%= link_to 'Resources', resources_path, remote: true %>
```

Sólo con eso se generará un request AJAX

Y... ¿cómo se responde?

AJAX en Rails

Por default el request pedirá JS, así que en el controller...

```
respond_to do |format|
  format.html {
    #...
  }
  format.js {
    render :text => 'alert("respuesta no muy útil...");'
  }
end
```

Si se deja el formato vacío (format.js {}) se procesará el template nombreAccion.js o .js.erb (para instrumentar con código Ruby, como en los .html.erb)

AJAX en Rails

Por default el request pedirá JS, así que en el controller...

```
respond_to do |format|
  format.html {
    #...
  }
  format.js {
    render :text => 'alert("respuesta no muy útil...");'
  }
end
```

Todo el código JS que se envíe se ejecutará en el browser. Por ello, podremos actualizar la vista como nosotros queramos.

AJAX en Rails

Pero además se puede pedir JSON o HTML, agregando otra opción además de :remote

```
<%= link_to 'Photos', user_photos_path(user.id),
remote: true, 'data-type' => :html %>
```

Pero... tendremos que procesar la respuesta por nosotros mismos...

```
$(function() {
    $('a[data-type=html]').on('ajax:success', function(event, data, status, xhr) {
      var container = $('#photo_container');
      container.append(data);
    });
});
```