PRIMEROS PASOS

PRIMEROS PASOS 1
1.- CONSEJOS DE PROGRAMACIÓN PARA PROGRAMAR EN AJAX. 1
2.- MOOTOOLS: INTRODUCCIÓN. 4
3.- COMO FUNCIONA MOOTOOLS 5

1.- CONSEJOS DE PROGRAMACIÓN PARA PROGRAMAR EN AJAX.

https://blueprints.dev.java.net/bpcatalog/ee5/ajax/javascript-recommendations.html

1) Usar una guia de programación.
Esto es muy importante ya que siguiendo una metodología de programación estandar podremos permitir que en el futuro sean más fáciles de mantener nuestras aplicaciones y que cualquier otra persona que conozca estas convenciones, tendrá más fácil el trabajo de mantener estas aplicaciones.

2) Usar la orientación de objetos de Javascript.
Pese a que Javascript no parece un lenguaje de programación orientado a objetos, dispone de alguna similitudes que nos permiten trabajar de forma más cómoda y eficiente. La propiedad de poder crear nuestros propios objetos e interactuar con ellos nos permite separar funcionalidades en objetos independientes que no tienen por que saber como funcionan cada uno.

3) Usa jerarquias para organizar tus objetos Javascript
Javascript está expuesto a problemas de colisión de nombres de variables, y esto puede ser un problema ya que podemos sobreescribir una variable con el contenido de otra y perder toda la funcionalidad que la anterior nos ofrecía, por este motivo hemos de curarnos en salud y comprobar la existencia de esta variable antes de crearla, si esta existe continuaremos añadiendole funcionalidades a la ya existente.

```
if (!miObj) { var miObj = new Object();}
miObj.Saluda = function () { alert("Hola"); }
```

4) Usa la propiedad prototype para extender los objetos
La propiedad prototype, está definida en el lenguaje y nos permite extender objetos con funcionalidades propias. Esto nos permite ampliar el abanico de funcionalidades de los objetos integrados en el lenguaje, como los creados por nosotros mismo con una simple línea de código. Hemos de aprovecharnos de esta propiedad para armar nuestras aplicaciones.

Ejemplos:
1.-Ampliar funcionalidades de los objetos del lenguaje:

```
//Amplia funcionalidad de Array
Array.prototype.sonrie = function(){alert(":-))");}
var datos = new Array("zz","yy");
datos.sonrie();
```

2.- Ampliar nuestros propios objetos

```
//Ampliar nuestros propios objetos (incorrecto)
kk = function(){};
kk.prototype.propiedad =new Array("hola","hello");
kk.prototype.metodo = function(){alert(this.propiedad)};

var kk1 = new kk();
var kk2 = new kk();

kk1.propiedad.push("adios");
kk1.metodo();       //hola, hello, adios
kk2.metodo();       //hola, hello, adios (mal!! Debería ser hola, hello)

//Ampliar nuestros propios objetos (corregido)
nokk = function(){this.propiedad = new Array("hola","hello");};
//nokk.prototype.propiedad =new Array("hola","hello");
nokk.prototype.metodo = function(){alert(this.propiedad)};

var nokk1 = new nokk();
var nokk2 = new nokk();

nokk1.propiedad.push("adios");
nokk1.metodo();           //hola, hello, adios
nokk2.metodo();           //hola, hello (bien!!)
```

(+ en http://javis.wordpress.com/2006/10/23/javascript-orientado-a-objetos/)

5) Escribir un javascript reusable

Intenta hacer funciones o métodos pequeños que puedan ser llamados desde cualquier sitio, sin tener que ser dependientes de otras secciones de código que no necesitas.

6) Usa objetos como parametros flexibles para funciones.

Este punto va muy ligado con el anterior, ya que usando objetos podemos flexibilizar nuestras aplicaciones todo lo que queramos. Permitiendonos una interacción mucho mayor.

```
function doSearch(serviceURL, srcElement, targetDiv) {
 var params = {service: serviceURL, method: "get", type: "text/xml"};
 makeAJAXCall(params);
 }

 function makeAJAXCall(params) {
 var serviceURL = params.service;
 ...
}
```

7) Comprime tu Javascript

Este punto choca con el hacer un código fácilmente mantenible, ya que comprimir nuestro javascript nos asegura que no vamos a entender el código una vez comprimido pero eso se arregla con tener 2 ficheros, uno comprimido y otro descomprimido.

8) Considera cargar el javascript cuando lo necesites
Una forma de ahorrarnos unos KB en la carga inicial es ir cargando el javascript a medida que lo vás necesitando, de esta forma no estarás cargando nada más de lo que necesitas y en el momento que lo necesitas.

9) Considera JSON como modelo de transporte de datos
Planteate cambiar la forma de transportar datos. Frente al universal XML surge JSON con mucha fuerza y sobretodo facilidad.

10) Separa claramente el contenido, CSS y Javascript.
Usa un código no obstructivo, esto te asegurará aplicaciones sin Javascript puedan trabajar con tu aplicación sin ningún tipo de problema. Para conseguir esto la separación del diseño, el contenido y el javascript debe ser clara y bien delímitada, de esta forma en un futuro podremos cambiar cualquiera de las capas anteriores sin alterar el funcionamiento de las otras.

11) Usa element.innerHTML con cuidado
Siempre que puedas debes evitar el uso de innerHTML y reemplazarlo por las alternativas DOM que los navegadores nos ofrecen.

12) Cuida los closures
Algo que generalmente olvidamos todos, o casi, los desarrolladores es el destruir las variables despues de usarlas. Esto en algunos casos puede darnos problemas considerables de memoria, ya que estamos dejando almacenado en ella una serie de datos que no nos son útiles. Sería conveniente igualar a null, las variables que no sean ya utiles.

13) UTF-8 te dará la libertad necesaria
Una forma de hacer llegar a todo el mundo tus aplicaciones, es usar una codificación de carácteres que te permita tener la flexibilidad necesaria para abarcar el máximo de idiomas posibles. Hoy por hoy, UTF-8 es la mejor opción.

2.- MOOTOOLS: INTRODUCCIÓN.

Mootools es un framework javascript orientado a objetos que agrupa gran cantidad de herramientas y plugins para facilitarle la vida a todos aquellos que a diario deben lidiar con efectos CSS, AJAX, JSON, etc sobre los diferentes navegadores e intentos de navegadores.

Entendemos por framework a una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Los Frameworks son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional. Quejas comunes acerca de que el uso de frameworks: añade código innecesario y que la preponderancia de frameworks competitivos y complementarios significa que el tiempo que se pasaba programando y diseñando ahora se gasta en aprender a usar frameworks.

Mootools va por su versión 1.2 y al día de hoy, acumula muchísimas herramientas y plugins realizados por una gran comunidad que se está formando.

La librería completa, incluyendo plugins y todo, pesa solo 42kb comprimida… pero dado que nos permiten descargar solo los componentes que vamos a utilizar, podemos hacerla adelgazar hasta 22kb sin perder sus herramientas mas importantes.

3.- COMO FUNCIONA MOOTOOLS

Desde http://mootools.net/ nos vamos a download.

En la versión 1.11 elegimos los componentes que nos van a hacer falta. En la nueva versión (junio) el sistema de descarga ha variado un poco, han dividido la libreria en dos partes: Core y More. Core es un archivo con los componentes mas básicos y More es otro archivo con los plugins y demas agregados.

Tanto el archivo Core como el archivo More lo pueden armar a la antigua, es decir, eligiendo los componentes que nos hacen falta, pero ahora ofrecen la posibilidad de descargar un archivo prearmado con las funcionalidades mas básicas.

Ahora con dos clicks y sin pensar mucho tienen la libreria armada y lista para usar. Al mismo tiempo no han quitado la posibilidad de descargar todo por separado asi que me parece un cambio bastante positivo.

En cada título de los capítulos que se tratan se pone el nombre del componente que se usa. Como en este curso vamos a usar muchos nos bajaremos la librería armada con Core y según nos haga falta montaremos el More.

Elegir compresión: La forma de comprimir javascript (se trata de un formato de texto plano) es eliminando espacios en blanco (incluye retornos de carro), eliminando los comentarios y reemplazando los nombres de las variables internas por otras mas cortas del tipo “a”. Elegimos YUI Compressor que es la que nos da por defecto.

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

<head>
<title>Hola mundo</title>

<script language="JavaScript" src=" mootools-1.2-core-yc.js" type=
"text/javascript"> </script>
<script>
    //Código javascript
</script>
</head>
<body>(…)</body>
</html>
```

Primer ejemplo:
Lo primero que vamos a aprender es crearnos una consola.

Para trabajar con mootools.

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

<head>
<title>Hola mundo</title>

<script language="JavaScript" src="../ mootools-1.2-core-yc.js" type=
```

```
"text/javascript"> </script>
```

Creamos al final del body un div con id “consola”. Para ello necesitamos new Element que creará un div. Luego usamos adpt para ponerlo al final del body.

```
<script>
function escribe(valor) {
    var elemento = new Element('div',{'id':'consola'});
    $$("body")[0].adopt(elemento);
}
```

Cada vez que llamemos a la función escribe para escribir insertará un nuevo div “consola” al body. Si queremos tener una capa div consola y luego varios divs por línea debemos preguntar primero si existe “consola”. Si es así que la cree, sino que no haga nada.

```
<script>
function escribe(valor) {
   if (!$('consola')) {
    var elemento = new Element('div',{'id':'consola'});
    $$("body")[0].adopt(elemento);
   }
}
```

Creamos una función que añada un div con texto cada vez. No podemos usar appendText directamento sobre “consola” ($(“consola”).appendText(“”)) porque no permite introducir etiquetas html y todo aparecería de corrido:

```
function escribe(valor){
   if (!$('consola')) {
    var elemento = new Element('div',{'id':'consola'});
    $$("body")[0].adopt(elemento);
   }

   var elemento = new Element('div');
   elemento.appendText(valor)
   $("consola").adopt(elemento);
}
</script>
</head>
```

Guardamos este código como consola.js y lo añadiremos en cada ejercicio para evitar tener que hacer alerts.

Para probar la consola:
Para no tener javascript incursivo como este:

```
<body>
  Escribe <a href='#'
  onClick="escribe('Escritura muy largaa');return false;">xxx</a>
</body>
</html>
```

Modificamos la función de onLoad:

```
<script>
window.onload = function(){
    var elemento = new Element('div',{'id':'consola'});
    $$("body")[0].adopt(elemento);

    // Aquí creamos un evento para <a>
    var mihref = $('escribe');
    mihref.onclick= function() {
        escribe('Escritura muy largaa');
        return false;
    };
}
```