# Práctica 4
# Javascript: Formularios y Asíncrono

# Formularios

- Cuando se carga una página web, el navegador crea automáticamente un array llamado forms y que contiene la referencia a todos los formularios de la página.
  - document.forms[0]; //Primer formulario
- Además se crea automáticamente un array llamado elements por cada uno de los formularios de la página con todos sus elementos (cuadros de texto, botones, listas desplegables, etc.)
  - document.forms[0].elements[0];
  - document.forms[0].elements[document.forms[0].elements.length-1]; //Acceder al último elemento

# Formularios

- Por cada elemento se dispone de las siguientes propiedades:
  - type: indica el tipo de elemento que se trata.
    - ✓ Para los elementos de tipo <input> (text, button, checkbox, etc.) coincide con el valor de su atributo type.
    - ✓ Para las listas desplegables (elemento <select>)
      - su valor es select-one y select-multiple.
  - form: es una referencia directa al formulario Así, para acceder al formulario de un elemento, se puede utilizar
    - ✓ document.getElementById("id_del_elemento").form
  - name: obtiene el valor del atributo name de XHTML. Solamente se puede leer su valor, por lo que no se puede modificar.
  - value: permite leer y modificar el valor del atributo value de XHTML.
    - ✓ (<input type="text"> y <textarea>) obtiene el texto que ha escrito el usuario.
    - ✓ Para los botones obtiene el texto que se muestra en el botón.

# Formularios

- Obtener el valor de un cuadro de texto y textarea

```
<input type="text" id="texto" />
var valor = document.getElementById("texto").value;

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

- Obtener valor de un radioButton

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


 var elementos = document.getElementsByName("pregunta");

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

# Formularios

- Obtener valor de un checkbok

```
<input type="checkbox" value="condiciones" name="condiciones"
id="condiciones"/> He leído y acepto las condiciones
<input type="checkbox" value="privacidad" name="privacidad" id="privacidad"/>
He leído la política de privacidad
```

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

# Formularios

- Obtener valor de un select.
  - options, es un array para cada lista desplegable y que contiene la referencia a todas las opciones de esa lista.
    - ✓ document.getElementById("id_de_la_lista").options[0].
  - SelectedIndex devuelve la opción seleccionada

```
// Obtener el objeto select
var lista = document.getElementById("opciones");

// Obtener el valor de la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].value;

// Obtener el texto que muestra la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].text;
```

  - Recordatorio: Estos componentes tienen un valor y un texto asociado (que pueden ser diferente).

# Formularios

- Obtener valor de un select: Múltiple.

```
function multiple()
{
  var selectedArray = new Array();
  var selObj = document.getElementById('multiple');
  var i;
  var count = 0;
  for (i=0; i<selObj.options.length; i++) {
    if (selObj.options[i].selected) {
      selectedArray[count] = selObj.options[i].value;
      count++;
    }
  }
}
```

# Formularios

- Evitar el envio duplicado
  - Una buena práctica en el diseño de aplicaciones web suele ser la de deshabilitar el botón de envío después de la primera pulsación.

```
<form id="formulario" action="#">
  ...
  <input type="button" value="Enviar"
         onclick="this.disabled=true;
         this.value='Enviando...';
         this.form.submit()" />
</form>
```

# Formularios

- Limitar el texto de un textarea
  - Imposibilidad de limitar el máximo número de caracteres que se pueden introducir, de forma similar al atributo maxlength de los cuadros de texto normales.

```
function limita(maximoCaracteres) {
  var elemento = document.getElementById("texto");
  if(elemento.value.length >= maximoCaracteres ) {
    return false;
  }
  else {
    return true;
  }
}
```

```
<textarea id="texto" onkeypress="return
limita(100);"></textarea>
```

# Formularios

- Validación de un formulario
  - La principal utilidad de JavaScript en el manejo de los formularios es la validación de los datos introducidos por los usuarios.

```
<form action="" method="" id="" name="" onsubmit="return validacion()">
  ...
</form>
```

```
function validacion() {
  if (condicion que debe cumplir el primer campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }
  else if (condicion que debe cumplir el segundo campo del formulario)
{
    return false;
  }
  // Si el script ha llegado a este punto, es válido
  return true;
}
```

## Ejemplos de control: Teclas que se pueden introducir (Ej1)

```
function permite(elEvento, permitidos) {
  var numeros = "0123456789";
  var caracteres = " abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
  var numeros_caracteres = numeros + caracteres;
  var teclas_especiales = [8, 37, 39, 46];
  // 8 = BackSpace, 46 = Supr, 37 = flecha izquierda, 39 = flecha derecha
  // Seleccionar los caracteres a partir del parámetro de la función
  switch(permitidos) {
    case 'num':
          permitidos = numeros;break;
    case 'car':
          permitidos = caracteres;break;
    case 'num_car':
          permitidos = numeros_caracteres;break;
  }
  // Obtener la tecla pulsada
  var evento = elEvento || window.event;
  var codigoCaracter = evento.charCode || evento.keyCode;
  var caracter = String.fromCharCode(codigoCaracter);

  // Comprobar si la tecla pulsada es alguna de las teclas especiales
  var tecla_especial = false;
  for (var i in teclas_especiales) {
     if (codigoCaracter == teclas_especiales[i]) {
          tecla_especial = true;
          break;
      }
   }
  // Si la tecla pulsada son caracteres permitidos o si es una tecla especial
  return permitidos.indexOf(caracter) != -1 || tecla_especial;
}
```

```
<input type="text" name="postal"
size="5" maxlength="5"
onkeypress="return permite(event,
'num')"/>
```

# Formularios

- Ejemplos de control: Validar un campo de texto obligatorio
  - El valor introducido sea válido
  - Que el número de caracteres introducido sea mayor que cero
  - Que no se hayan introducido sólo espacios en blanco.

```
function validacion(){
  valor = document.getElementById("campo").value;
  if (valor == null || valor.length == 0 || /^\s+$/.test(valor)) {
     return false;
  }
  return true;
}
```

- Ejemplos de control: Validar un campo de texto numéricos
  - Uso de la función isNaN

```
function validacion() {
  valor = document.getElementById("campo").value;
  if (isNaN(valor)) {
        return false;
  }
  return true;
}
```

# Formularios

- Ejemplos de control: Validar que se ha seleccionado una opción de una lista
  - Comprobar el valor de selectedIndex

```
function validacion() {
  indice = document.getElementById("opciones").selectedIndex;
  if (indice == null || indice == 0) {
        return false;
  }
  return true;
}
```

- Ejemplos de control: Validar una dirección de email
  - Se comprueba es que la dirección parezca válida pues no se puede comprobar realmente si existe.

```
function validacion() {
  valor = document.getElementById("campo").value;
  if (!(/\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)/.test(valor))) {
        return false;
  }
  return true;
}
```

# Formularios

- Ejemplos de control: Validar un dni
  - Si tiene valor el campo de texto

```
function validacion() {
  valor = document.getElementById("campo").value;
  var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X',
'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];

  if (!(/^\d{8}[A-Z]$/.test(valor))) {
        return false;
  }
  if (valor.charAt(8) != letras[(valor.substring(0, 8)) % 23]) {
        return false;
  }
  return true;
}
```

# Formularios

- Ejemplos de control: Validar un número de teléfono

```
function validacion() {
  valor = document.getElementById("campo").value;
  if (!(/^\d{9}$/.test(valor))) {
        return false;
  }
  if (!(/^\d{9}$/)) {
        return false;
  }
  return true;
}
```

| Número | Expresión regular | Formato |
|---|---|---|
| 900900900 | /^\d{9}$/ | 9 cifras seguidas |
| 900-900-900 | /^\d{3}-\d{3}-\d{3}$/ | 9 cifras agrupadas de 3 en 3 y separadas por guiones |
| 900 900900 | /^\d{3}\s\d{6}$/ | 9 cifras, las 3 primeras separadas por un espacio |
| 900 90 09 00 | /^\d{3}\s\d{2}\s\d{2}\s\d{2}$/ | 9 cifras, las 3 primeras separadas por un espacio, las siguientes agrupadas de 2 en 2 |
| (900) 900900 | /^\(\d{3}\)\s\d{6}$/ | 9 cifras, las 3 primeras encerradas por paréntesis y un espacio de separación respecto del resto |
| +34 900900900 | /^\+\d{2,3}\s\d{9}$/ | Prefijo internacional (+ seguido de 2 o 3 cifras), espacio en blanco y 9 cifras consecutivas |

# Formularios

- Ejemplos de control: Validar que un checkbox ha sido seleccionado

```
function validacion() {
  indice = document.getElementById("campo");
  if (!elemento.checked)
        return false;
  return true;
}
```

- Ejemplos de control: Validar que un radiobutton ha sido seleccionado

```
function validacion() {
  opciones = document.getElementsByName("opciones");
  var seleccionado = false;
  for (var i = 0; i < opciones.length; i++) {
        if (opciones[i].checked) {
              seleccionado = true;
              break;
        }
  }
  if (!seleccionado) { return false; }
  return true;
}
```

# Formularios

- Verificación formulario (Formulario1.html):
  - Añade el código Javascript necesario para que se valide el formulario. Para que los datos se puedan enviar es necesario:
    - ✓ Que se haya rellenado un nombre
    - ✓ Que se haya rellenado dos apellidos.
    - ✓ Que se haya rellenado una dirección de correo. Verifica que la dirección contiene una @, y que despues de esta haya un "." (punto).
    - ✓ Que el teléfono contenga 9 cifras
    - ✓ Que se haya rellenado la empresa
    - ✓ Se haya seleccionado un cargo

# Formularios

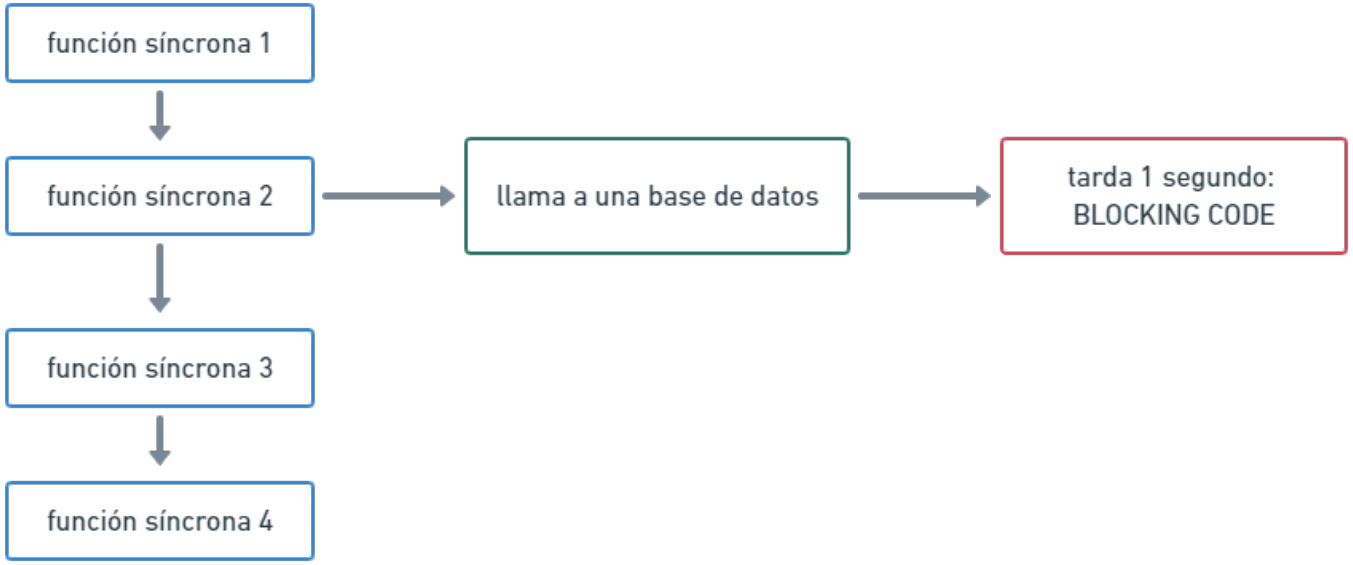
- Verificación formulario: Añade el código Javascript necesario para que se valide este formulario (Formulario2.html).

# Formularios. Calendarios

- Con la aparición de HTML5 han aparecido otra serie de controles:
  - Campo de dirección de correo electrónico
    - ✓ <input type="email" id="email" name="email">
  - Campo de búsqueda: Ccrear cajas de búsqueda en páginas y aplicaciones.
    - ✓ <input type="search" id="search" name="search">
  - Campo número de teléfono:
    - ✓ <input type="tel" id="tel" name="tel">
  - Campo URL
    - ✓ <input type="url" id="url" name="url">
  - Campo numérico
    - ✓ <input type="number" name="age" id="age" min="1" max="10">
  - Campo fecha y hora
    - ✓ <input type="datetime-local" name="datetime" id="datetime">

# JavaScript Asíncrono

- JS es por definición un lenguaje de código síncrono. Es decir, lee una línea de código detrás de otra, de arriba a abajo
- Pero en ocasiones se necesita ejecutar una función que requiere cierto tiempo, y si tuviésemos que esperar a que JS resolviese ese bloque de código para poder pasar al siguiente, el resultado no sería nada eficiente.



# JavaScript Asíncrono

- Para solucionar el problema anterior necesitamos utilizar código asíncrono –> JS viene con funciones async.

```
<body>
<script src="ej1.js">
</script>
</body>
```

**index.html**

**ej1.js**

```
console.log(1);
console.log(2);
console.log(3);
console.log(4);
console.log(5);
```

**Asíncrono**

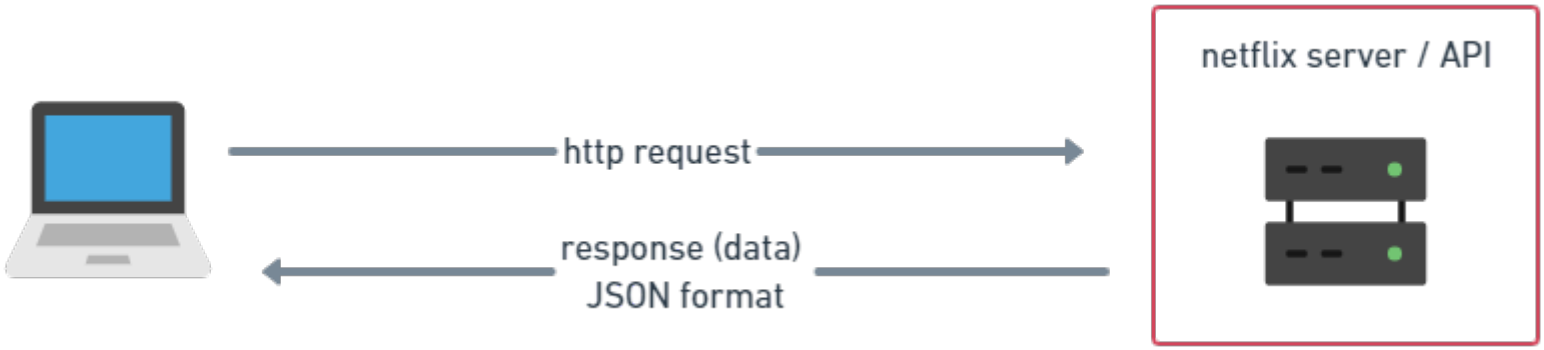
```
console.log(1);
console.log(2);
console.log(3);

setTimeout(() => {
   console.log('Después de todo!');
}, 2000);

console.log(4);
console.log(5);
```

# JavaScript Asíncrono

- SetTimeout permite simular que estamos solicitando información de algún servidor o una base de datos.
- A continuación usaremos javascript asíncrono para interactuar con bases de datos y servidores, pudiéndose usar:
  - Para recuperar información de form asíncrona en base a algún evento (selecciono una provincia y se cargan sus ciudades)
  - Cargar una página web desde distintas fuentes de forma asíncrona
  - ….



# Javascript Asíncrono

- 3 maneras de llamar a una API Rest con JavaScript
  - XMLHttpRequest (AJAX) es un objeto de JavaScript que fue diseñado por Microsoft y adoptado por Mozilla, Apple y Google. Es un estándar de la W3C.
  - Fetch: es una nueva implementación de JavaScript, nos permite hacer lo mismo que XMLHttpRequest pero de manera más sencilla.
  - Mediante librerías externas: Axios, Jquery ,Prototype, ….

# Javascript Asíncrono

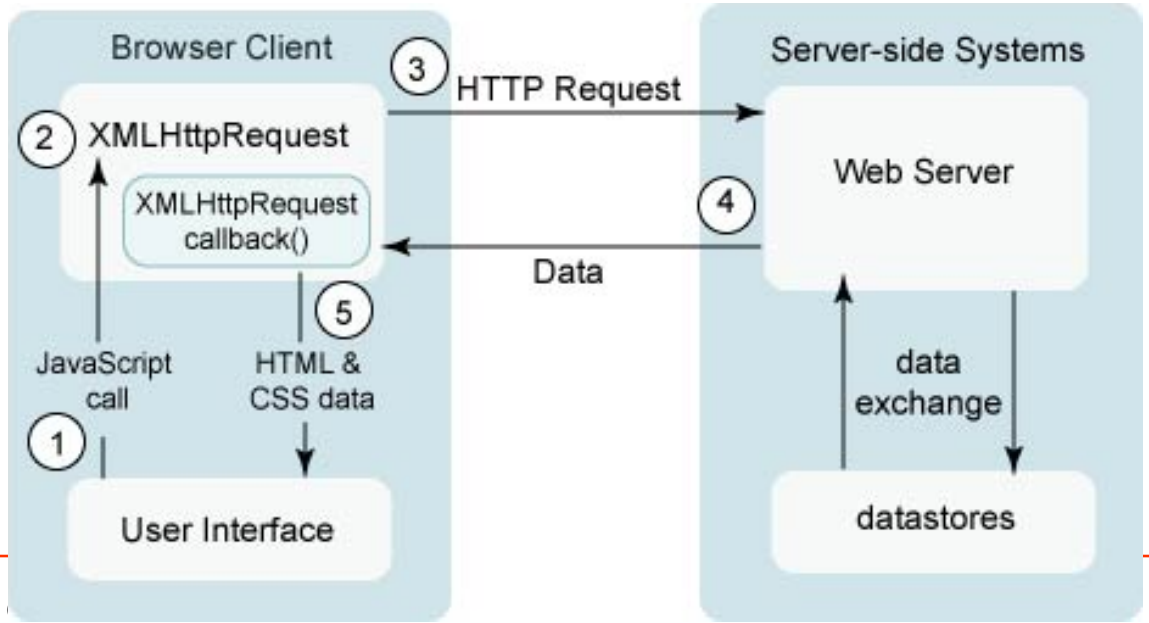
- Se puede utilizar Ajax de 4 formas distintas:
  1) synchronous, text-only (SJAT?)
  2) asynchronous, text-only (AJAT?)
  3) asynchronous w/ Prototype (AJAP?)
  4) asynchronous w/ XML data (real Ajax)

- En este sesión sólo se utilizará la solución asíncrona usando texto, XML y JSON.

# Javascript Asíncrono

## XMLHttpRequest

- Los clientes web podrán recuperar XML o json, en background → ASÍNCRONAMENTE.
- Usando DOM se podrá leer el contenido devuelto y modificar el documento a presentar en el navegador
- Es un API que se puede utilizar desde JavaScript, Jscript, VBScript y otros lenguajes de script



# Javascript Asíncrono

## Pasos XMLHttpRequest

- Se crea el objeto XMLHttpRequest
- Se envía la petición en función de la interacción del usuario
- Se procesa los datos de retorno

```
// Creamos un nuevo XMLHttpRequest
var xhttp = new XMLHttpRequest();

// Esta es la función que se ejecutará al finalizar la
llamada
xhttp.onreadystatechange = function() {
    // La respuesta, aunque sea JSON, viene en formato
texto, por lo que tendremos que hace run parse
    document.write(JSON.parse(this.responseText));

};

// Endpoint de la API y método http
xhttp.open("GET", "https://pokeapi.co/api/v2/pokemon",
true);
xhttp.setRequestHeader("Content-type",
"application/json");
// Si quisieramos mandar parámetros a nuestra API,
podríamos hacerlo desde el método send()
xhttp.send(null);
```

# Javascript Asíncrono

## Pasos Fetch

- Se hace la llamada a fetch
- Si se recibe la respuesta entones se establece la acción a realizar

```
fetch('https://pokeapi.co/api/v2/pokemon')
      .then(response => response.json())
      .then(json => console.log(json));
```

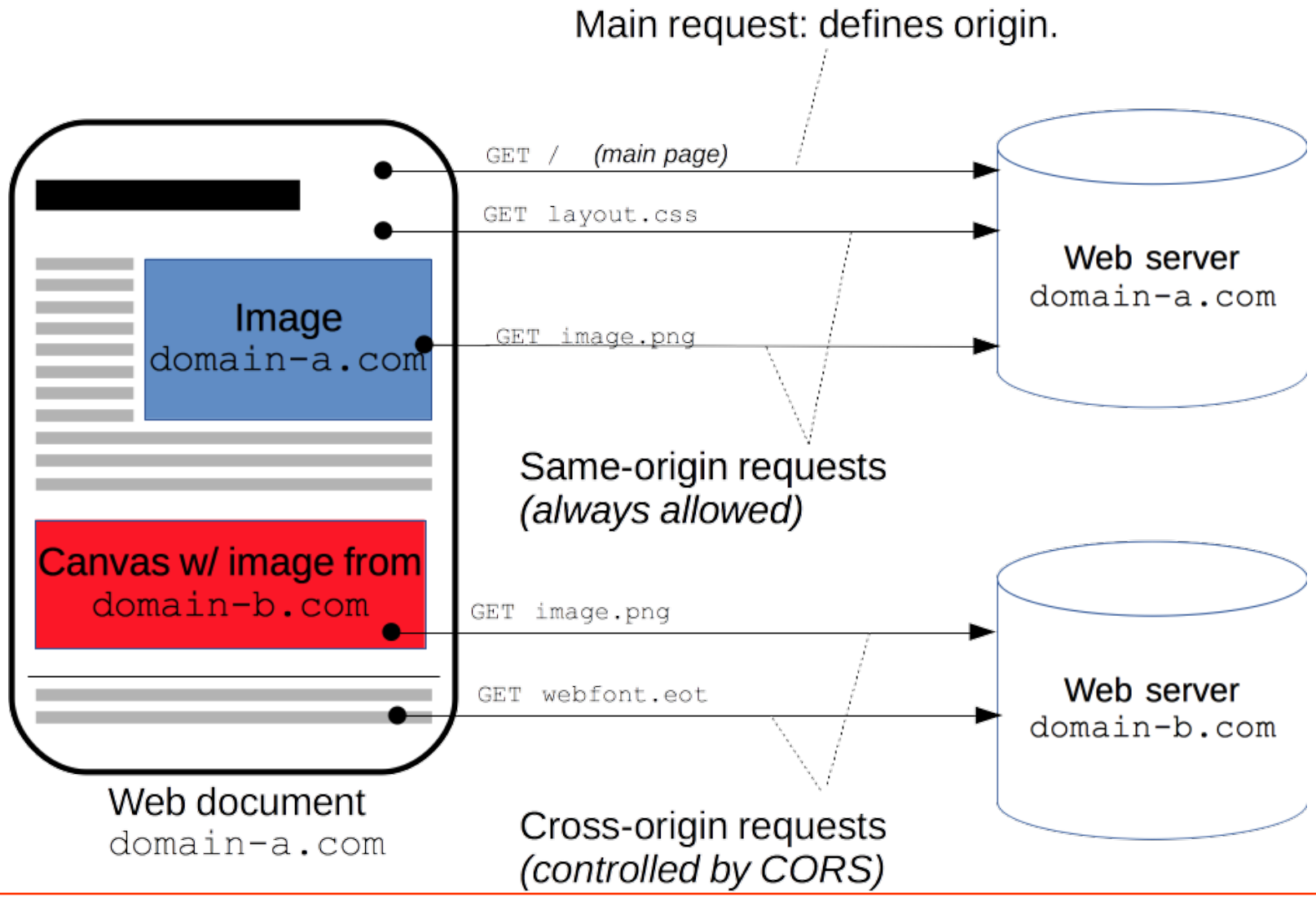
# Javascript Asíncrono

- **No ha funcionado –> CORS**
  - Cuando se abre una página web y se cargan datos de servidores ajenos, en teoría, está estrictamente prohibido.
  - Sin embargo, puede haber excepciones: si los administradores de ambas webs han acordado trabajar juntos, no hay por qué evitar el intercambio –> cross-origin resource sharing (**CORS**) regula la colaboración
  - La same-origin policy (SOP o política de seguridad del mismo origen) **prohíbe** que se carguen datos de servidores ajenos al acceder a una página web. **Todos los datos deben provenir de la misma fuente**, es decir, corresponder al mismo servidor. Se trata de una medida de seguridad, ya que JavaScript y CSS podrían cargar, sin que el usuario lo supiese, contenido de otros servidores (y, con este, también contenido malicioso).

# Javascript Asíncrono

- **No ha funcionado –> CORS**
  - El servidor solicitado (es decir, aquel del que se quiere cargar contenido) puede permitir entonces el acceso mediante cross-origin resource sharing (intercambio de recursos de origen cruzado).
  - El segundo servidor permite al primero un acceso exclusivo mediante una cabecera HTTP. En dicha cabecera de la respuesta HTTP está indicado específicamente qué servidores pueden cargar datos y ponerlos a disposición del usuario. El acceso generalizado a todos los clientes se permite únicamente mediante una “wildcard” o certificado comodín. Esta solución, sin embargo, solo es conveniente para servidores cuyo contenido debe estar a disposición del público general.

## Funcionamiento de CORS




Obtenido de:https://developer.mozilla.org/es/docs/Web/HTTP/CORS

## Funcionamiento de CORS

- En CORS, Los métodos GET Y EHAD no suelen dar problemas pues no pueden alterar datos y, por lo tanto, no suelen considerarse como un riesgo para la seguridad.
- No se puede decir lo mismo de PATCH, PUT o DELETE: con ellos sí se puede llevar a cabo acciones maliciosas, por lo que en estos casos también hay que activar el cross-origin resource sharing
- Si se trata de métodos HTTP de seguridad, el cliente envía en primer lugar una solicitud preflight(preflight request) en la que solo se indica qué método HTTP se piensa transmitir al servidor a continuación y se pregunta si la solicitud es considerada segura. Para ello, se usa la cabecera OPTIONS (OPTIONS header). Una vez se haya recibido una respuesta positiva, ya se puede realizar la solicitud propiamente dicha.

# Javascript Asíncrono

- **Qué peticiones utiliza CORS?** Este estándar de intercambio de origen cruzado es utilizado para habilitar solicitudes HTTP de sitios cruzados para:
  - Invocaciones de las APIs XMLHttpRequest o Fetch en una manera de sitio cruzado.
  - Fuentes Web (para usos de fuente en dominios cruzados @font-face dentro de CSS).
  - Texturas WebGL.
  - Imágenes dibujadas en patrones usando drawImage.
  - Hojas de estilo (para acceso CSSOM).
  - Scripts (para excepciones inmutadas).

# Javascript Asíncrono

- **Existen diferentes cabeceras o CORS headers:**
  - Access-Control-Allow-Origin: ¿qué origen está permitido?
  - Access-Control-Allow-Credentials: ¿también se aceptan solicitudes cuando el modo de credenciales es incluir (include)?
  - Access-Control-Allow-Headers: ¿qué cabeceras pueden utilizarse?
  - Access-Control-Allow-Methods: ¿qué métodos de petición HTTP están permitidos?
  - Access-Control-Expose-Headers: ¿qué cabeceras pueden mostrarse?
  - Access-Control-Max-Age: ¿cuándo pierde su validez la solicitud preflight?
  - Access-Control-Request-Headers: ¿qué header HTTP se indica en la solicitud preflight?
  - Access-Control-Request-Method: ¿qué método de petición HTTP se indica en la solicitud preflight?
  - Origin: ¿de qué origen proviene la solicitud?
  - ….

# Ejemplo 02

- Llamada simple en Ajax con control de carga

```
<html><head>
<script type="text/javascript" language="javascript">
var peticion_http=new XMLHttpRequest();

function cargaContenido(url, metodo, funcion) {
    peticion_http.onreadystatechange = funcion;
    peticion_http.open(metodo, url, true);
    peticion_http.setRequestHeader("Content-type", "application/json");
    peticion_http.send(null);
}

function muestraContenido() {
  if(peticion_http.readyState == 4 && peticion_http.status == 200) {
        document.getElementById("myDiv").innerHTML=peticion_http.responseText;
  }else {
        document.getElementById("myDiv").innerHTML="Problema en la carga";
  }
}

function descargaArchivo() {
  cargaContenido("https://pokeapi.co/api/v2/pokemon", "GET", muestraContenido);
}
</script>
</head><body>

<div id="myDiv"><h2>Let AJAX change this text</h2></div>
<button type="button" onclick="descargaArchivo()">Cambiar Contenido</button>
</body></html>
```

Función que se ejecuta cuando se ha completado

Si todo ha ido bien

Cuando se pincha en el boton

# Ejemplo 03

## Llamada simple en Ajax con control de estado

```
<script type="text/javascript" language="javascript">
  var READY_STATE_UNINITIALIZED=0;
  var READY_STATE_LOADING=1;
  var READY_STATE_LOADED=2;
  var READY_STATE_INTERACTIVE=3;
  var READY_STATE_COMPLETE=4;
  var peticion_http=new XMLHttpRequest();

  function cargaContenido(url, metodo, funcion) {
            peticion_http.onreadystatechange = funcion;
            peticion_http.open(metodo, url, true);
            peticion_http.send(null);
   }

   function muestraContenido() {
            if (peticion_http.readyState == READY_STATE_COMPLETE)  && peticion_http.status == 200) {
                        document.getElementById("myDiv").innerHTML = peticion_http.responseText;

            } else if (peticion_http.readyState == READY_STATE_LOADING) {
                        document.getElementById("myDiv").innerHTML = "Cargando";
            }
    }

   function descargaArchivo() {
        //  cargaContenido("https://pokeapi.co/api/v2/pokemon", "GET", muestraContenido);
        //  cargaContenido("http://api.openweathermap.org/data/2.5/weather?
q=Madrid&appid=14c293857fefc5bfc79ee7cbe354ee57", "GET", muestraContenido);
        cargaContenido("http://programacion-cum.unex.es/cargaProvinciasJSON.php", "GET", muestraContenido);
   }
</script>
```

Posibles estados

Resultado en un div

Mientras se carga

# Paso de Parámetros en Ajax

- Hasta ahora, el objeto XMLHttpRequest se ha empleado para realizar peticiones HTTP sencillas.
- Sin embargo, las posibilidades que ofrece el objeto XMLHttpRequest son muy superiores, ya que también permite el envío de parámetros junto con la petición HTTP.
- El objeto XMLHttpRequest puede enviar parámetros tanto con el método GET como con el método POST de HTTP.
- En ambos casos, los parámetros se envían como una serie de pares clave/valor concatenados por símbolos &.

http://localhost/aplicacion?parametro1=valor1&parametro2=valor2&parametro3=valor3

# Pasos de parámetros mediante XML

- El objeto XMLHttpRequest permite el envío de los parámetros por otros medios alternativos a la tradicional query string.
- De esta forma, es posible realizar una petición al servidor enviando los parámetros en formato XML o en formato JSON.

```
<resultado>
          <login>SI/NO</login>
          <username>------</username>
</resultado>
```

```
{
  "resultado": {
    "login": "SI/NO",
    "username": "------"
  }
}
```

# Pasos de parámetros mediante XML

- XML (eXtensible Markup Language) es un metalenguaje de marca, es decir, permite definir lenguajes de marcas.
- Cada uno puede hacer su propio lenguaje de marca con sus propias etiquetas para así añadir metainformación.
- Se utilizó muchísimo al inicio del 2000 pero ha decrecido su uso en la web por JSON: más sencillo y menos pesado.

```
<resultado>
        <login>SI/NO</login>
        <username>------</username>
</resultado>
```

# Pasos de parámetros mediante JSON

- JSON (acrónimo de JS Object Notation) es un formato de texto sencillo para el intercambio de datos. JSON es un formato mucho más compacto y ligero que XML.
- En JavaScript, un texto JSON se puede analizar fácilmente usando la función eval().

```
{
  "departamento":8,
  "nombredepto":"Ventas",
  "director": "Juan Rodríguez",
  "empleados":[
    {
      "nombre":"Pedro",
      "apellido":"Fernández"
    },{
      "nombre":"Jacinto",
      "apellido":"Benavente"
    }
  ]
}
```

# Pasos de parámetros mediante JSON

- Los tipos de datos disponibles con JSON son:
  - **Números**: Se permiten números negativos y decimal: 123.456
  - **Cadenas**: Representan secuencias de cero o más caracteres. Se ponen entre doble comilla: "Hola"
  - **Booleanos**: Representan valores booleanos y pueden tener dos valores: true y false
  - **null**: Representan el valor nulo.
  - **Array**: Representa una lista ordenada de cero o más valores los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector se mete entre corchetes. Ejemplo ["juan","pedro","jacinto"]
  - Objetos: Son colecciones no ordenadas de pares de la forma <nombre>:<valor> separados por comas y puestas entre llaves.

```
{
  "departamento":8,
  "nombredepto":"Ventas",
  "director": "Juan Rodríguez
  "empleados":[
    {
      "nombre":"Pedro",
      "apellido":"Fernández"
    },{
      "nombre":"Jacinto",
      "apellido":"Benavente"
    }
  ]
}
```

# Ejemplo 04

## Llamada simple en Ajax con JSON

```
<html><head>
<script type="text/javascript" language="javascript">
var peticion_http=new XMLHttpRequest();

function cargaContenido(url, metodo, funcion) {
    peticion_http.onreadystatechange = funcion;
    peticion_http.open(metodo, url, true);
    peticion_http.setRequestHeader("Content-type", "application/json");
    peticion_http.send(null);
}

function muestraContenido() {
   if (peticion_http.readyState == READY_STATE_COMPLETE  && peticion_http.status == 200) {
         var json_data = peticion_http.responseText;
         document.getElementById("myDiv").innerHTML = json_data;
   } else if (peticion_http.readyState == READY_STATE_LOADING) {
         document.getElementById("myDiv").innerHTML = "Cargando";
   }
}

function descargaArchivo() {
  cargaContenido("https://pokeapi.co/api/v2/pokemon", "GET", muestraConte
}
</script>
</head><body>

<div id="myDiv"><h2>Let AJAX change this text</h2></div>
<button type="button" onclick="descargaArchivo()">Cambiar Contenido</button>
```

Función que se ejecuta cuando se ha completado

Si todo ha ido bien

Cuando se pincha en el boton

# Pasos de parámetros mediante JSON

- La respuesta JSON se obtiene con la propiedad responseText:

```
var respuesta_json = http_request.responseText;
```

- Esta propiedad se encuentra en forma de cadena de texto. Para trabajar con el código JSON devuelto, se debe transformar esa cadena de texto en un objeto JSON → función eval() o JSON.parse()

```
var objeto_json = eval("("+respuesta_json+")");
var objeto_json = JSON.parse(respuesta_json);
```

- Ahora ya permite acceder a sus métodos y propiedades mediante la notación de puntos tradicional.

# Pasos de parámetros mediante JSON

- Ahora ya permite acceder a sus métodos y propiedades mediante la notación de puntos tradicional.

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
```

```
var objeto_json = JSON.parse(respuesta_json);
objeto_json.homeTown
objeto_json['active']
objeto_json['members'][1]['name’]
objeto_json['members'][1]['powers'][2]
```

# Ejemplo 04

## Llamada simple en Ajax con JSON

```
<html><head>
<script type="text/javascript" language="javascript">
var peticion_http=new XMLHttpRequest();

function cargaContenido(url, metodo, funcion) {
    peticion_http.onreadystatechange = funcion;
    peticion_http.open(metodo, url, true);
    peticion_http.setRequestHeader("Content-type", "application/json");
    peticion_http.send(null);
}

function muestraContenido() {
   if (peticion_http.readyState == READY_STATE_COMPLETE  && peticion_http.status == 200) {
         var json_data = peticion_http.responseText;
          var the_object = JSON.parse(json_data);
          document.getElementById("myDiv").innerHTML =the_object."15"; //15 es la propiedad
   } else if (peticion_http.readyState == READY_STATE_LOADING) {
         document.getElementById("myDiv").innerHTML = "Cargando";
   }
}

function descargaArchivo() {
  cargaContenido("https://pokeapi.co/api/v2/pokemon", "GET", muestraCon
}
</script>
</head><body>

<div id="myDiv"><h2>Let AJAX change this text</h2></div>
```

Función que se ejecuta cuando se ha completado

Si todo ha ido bien

Cuando se pincha en el boton

# AJAX: Listas Desplegables

- Es habitual, que en las web cuando se selecciona un elemento de una lista desplegable, en otra se carguen unos valores dependiendo del seleccionado.
- El mayor inconveniente de este tipo de listas se produce cuando existen un gran número de opciones posibles.
- Problemas:
  - Si todos los elementos de las listas desplegables se almacenan mediante arrays de JavaScript en la propia página, los tiempos de carga se pueden disparar y hacerlo completamente inviable.
  - Por otra parte, si se recarga la página cada vez que se selecciona un valor → aumenta la carga en el servidor y el tiempo de espera del usuario.
- Una posible solución intermedia consiste en actualizar las listas desplegables mediante AJAX.