POLITÉCNICO COLOMBIANO
JAIME ISAZA CADAVID

Educación para vivir mejor

Construcción de elementos de software web 1

Semana 8

Fundamentos de javascript

Media Técnica en Programación de Sistemas de Información
Politécnico Jaime Isaza Cadavid - 2020

¿Qué es JavaScript?

JavaScript es un lenguaje de programación que nos permite dar instrucciones al ordenador, en este caso **al navegador web**, para explicarle cómo debe mostrar nuestra página y que debe hacer en qué situación (si se pulsa un botón, si se rellena un campo, o si pulsamos un enlace)

JavaScript en su principio fue creado para realizar validaciones sobre datos en un formulario, **pero ese tiempo quedó ya muy atrás**. Hoy en día es uno de los lenguajes más populares y gran parte de ese mérito se debe a que **es el lenguaje de la web**, es decir, es el único lenguaje de programación que entienden los navegadores (debemos recordar que HTML y CSS no se consideran, lenguajes de programación, para profundizar más sobre este tema recomendamos que investigues que significa que un lenguaje sea **Turing Completo**). Desde su inicio se ha expandido y sus fronteras han ido más allá de la web hasta llegar al punto en el que se utiliza JavaScript para programar aplicaciones para ordenador, servidores, robots e incluso proyectos espaciales llevados a cabo por la NASA.

En lo que a nosotros nos respecta y enfocándonos en la web, JavaScript va a ser la herramienta **que nos permita hacer páginas dinámicas.** Nos va a permitir realizar tareas como cambiar el contenido de una web en función de eventos (al hacer clic en el ratón, con el paso del tiempo, pulsando una tecla), obtener datos de un servidor para mostrarlos en la página o mostrar una información u otra en función de una serie de datos.

Nuestro primer código en JavaScript

Vamos a dejar a un lado la explicación y vamos a ponernos manos a la obra. Para ello vamos a crear nuestro primer código JavaScript y este lo que hará será mostrar en la pantalla el mensaje *"¡Hola mundo!"*.

Lo primero que debemos hacer es crear un archivo HTML. Al igual que pasa con CSS, JavaScript solo funciona en un navegador si lo incluimos en nuestro HTML. Como en este caso no queremos ningún estilo, crearemos un html simple:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Mi primer código JavaScript</title>
  </head>
  <body>
    <h1></h1>
  </body>
</html>
```

Hasta aquí todo normal, ahora viene la parte interesante. **¿Cómo utilizamos un código JavaScript dentro de esta página?** Por norma general, de momento enlazaremos JavaScript al final de la web, antes del cierre de la **etiqueta body**. Más adelante veremos el por qué, al igual que sucedía con CSS, podemos introducir JavaScript de dos formas en nuestra página, escribiendo el código **directamente dentro de una etiqueta o escribiendo código en un archivo distinto y enlazando este con nuestro HTML**. Para hacerlo de la primera manera, simplemente creamos una etiqueta <script> y metemos el código JavaScript dentro de ella:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Mi primer código JavaScript</title>
  </head>
  <body>
    <h1></h1>
    <!-- De momento, colocaremos la etiqueta script justo antes de cerrar la etiqueta
        body -->
    <script type="text/javascript">
      // Aquí iría el código JavaScript
    </script>
  </body>
</html>
```

Nota: Como puedes ver, dentro del archivo de JavaScript hemos escrito un mensaje precedido del texto //. Esta combinación escrita al principio de una línea, **marca esa línea como un comentario de JavaScript,** esto funciona igual que los comentarios en CSS y HTML. **De esta forma podemos ponernos anotaciones sin que se ejecuten o produzca un error en el código.** En JavaScript existen también comentarios multilínea, estos son mensajes envueltos entre **/* (al comienzo) y */ (al final) (ejemplo: /* Este es un comentario */).** Este tipo de comentarios se utilizan cuando queremos escribir mensajes que ocupen más de una línea dentro de nuestro código.

En el caso de enlazar un JavaScript externo, utilizaremos también **la etiqueta <script> pero esta vez le añadiremos un atributo HTML src="" en el que escribiremos como valor la ruta del archivo JavaScript que hemos creado**. El resultado sería el siguiente (imaginando que el archivo main.js está en la misma carpeta que el archivo HTML que hemos creado):

main.js

```
// Aquí iría el código JavaScript
```

index.html

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Mi primer código JavaScript</title>
  </head>
  <body>
    <h1></h1>
    <script type="text/javascript" src="main.js"></script>
  </body>
</html>
```

Por el momento y como norma general, utilizaremos la opción de enlazar un JavaScript externo en todos los ejercicios que realicemos.

Bien, ya hemos visto cómo enlazar JavaScript en nuestra página, ahora es el momento de ver nuestro primer código JavaScript. Partiendo del código que mostramos en el ejemplo anterior con los archivo **main.js e index.html,** vamos a añadir el código que aparece a continuación en el archivo **main.js**. Una vez añadido, abre desde tu navegador web el archivo HTML donde has enlazado ese JavaScript y observa qué sucede.

```
1  'use strict';
2
3  document.querySelector('h1').innerHTML = '¡Hola mundo!';
```

Si has realizado los pasos anteriores y has copiado el código correctamente se mostrará una ventana en tu navegador con el mensaje **"¡Hola mundo!"**. Si es así, ¡enhorabuena! acabas de crear tu primer código JavaScript. **Fíjate que en el fichero index.html no hay nada dentro de la etiqueta h1,** pero el navegador nos muestra un mensaje.

En este momento estarás pensando «sí, lo he escrito pero no tengo ni idea de cómo funciona». No te preocupes, vamos a entender cómo funciona ahora mismo.

La primera línea del archivo JavaScript **('use strict';) sirve para mejorar la rapidez de ejecución del código y hará que el navegador nos muestre errore**s que, de no ponerla, no lo haría. Por lo tanto esta instrucción al inicio de nuestro código hace que sea más estable o, dicho de otra forma, menos propenso a fallos. Como norma general, escribiremos SIEMPRE esta linea al comienzo de todos nuestros archivos JavaScript y para que funcione correctamente deberá ser la primera línea del documento (sin contar los comentarios y las líneas en blanco).

La segunda línea **(document.querySelector('h1').innerHTML = '¡Hola mundo!';)** es una sentencia que describe una acción. En programación una sentencia (*statement*) es la unidad mínima que expresa una acción a llevar a cabo, en este caso, por el navegador. Básicamente le decimos **"Hey navegador, haz esto."**

De momento para la sintaxis utilizada en **document.querySelector('h1').innerHTML = '¡Hola mundo!';,** solo comentaremos que permite cambiar el texto de una etiqueta de HTML determinada, **en este caso el primer h1 de nuestro documento HTML.**

Otro aspecto a destacar es que escribimos cada orden en una línea **y ponemos un punto y coma al final de ésta**. En JavaScript se pueden escribir varias sentencias en una misma línea si se separan por un punto y coma (;), **por ejemplo 'use strict';document.querySelector('h1').innerHTML='¡Hola mundo!'**; sería válido. Esto es totalmente una mala práctica de programación y evitaremos hacerlo para que nuestro código sea más fácil de leer. Escribiremos como máximo una orden por línea y siempre añadiremos el punto y coma al final de esta para evitar posibles problemas.

Puede que en este punto aún sigas pérdida y no te haya quedado muy claro cómo usar realmente JavaScript pero no te preocupes, de momento sólo debes entender que programar no es otra cosa que pensar en los pasos para resolver un problema y traducirlo a órdenes con un lenguaje que entienda el navegador (JavaScript). Por tanto, lo que tenemos que hacer es practicar la lógica, familiarizarnos con la sintaxis de JavaScript y aprender a traducir pasos a este lenguaje para ir poco a poco mejorando y cogiendo soltura.

Variables

Uno de los problemas que no sabemos resolver aún es como guardar los datos de una operación o un texto para poder utilizarlo en otra operación posterior. **JavaScript tiene un recurso para poder hacer esto, las variables.**

Para tener una idea de qué es una variable, podríamos pensar en ella como si fuera una caja con una etiqueta que describe su contenido, en la que guardamos algo.

1. Para crearla, creamos la caja con la etiqueta.
2. Para guardar algo metemos esa información dentro de la caja.
3. Para obtener la información utilizamos la etiqueta de la caja.

Una cosa importante a saber es que las variables permiten guardar información de forma temporal y estarán disponibles hasta que la página web en donde se está ejecutando el código se cierre o se recargue.

Veamos cómo trabajar con variables en JavaScript.

Crear o declarar una variable

A la creación de una variable se le llama declaración.

El primer paso a la hora de utilizar variables es declararlas. Para declarar una variable, escribimos **let** seguido de un espacio y posteriormente del nombre que queremos dar a la variable. Vamos a declarar, por ejemplo una variable para la dirección postal de una oficina y la llamaremos **officeAddress:**

```
let officeAddress;
```

Existen una serie de reglas importantes a la hora de establecer el nombre de una variable:

- Deberán empezar por una letra y podrán contener letras y números
- No podremos usar espacios ni puntos para separar los nombres de variables
- Utilizaremos el estilo *camel case* para nombrar las variables. Este estilo se basa en juntar varias palabras en una haciendo que cada palabra empiece por mayúscula excepto la primera de todas ellas (ejemplo: 'myCoolVariableName').

Asignar un valor a una variable

Una vez hemos declarado una variable, es el momento de asignarle un valor. Este sería el paso en el que guardamos algo dentro de la caja para poder luego cogerlo y utilizarlo cuantas veces queramos. Para asignar un valor a una variable, escribiremos el nombre de la variable seguido del símbolo = y finalmente el valor que queremos almacenar.

```
1  let officeAddress;
2  officeAddress = 'Calle Leganitos, 24';
```

Nota: Nunca debes asignar valor a una variable que no ha sido declarada.

Cosas importantes a tener en cuenta a la hora de asignar una variable:

- En vez de asignar un valor a una variable podemos asignar una operación. JavaScript lo que hace es que primero realiza las operaciones a la derecha del símbolo de igual y después el resultado de esas operaciones lo almacena en la variable.

```
let totalHours;
totalHours = 10 + 20 + 30;
```

- Los pasos para declarar una variable y asignarle un valor pueden combinarse y realizarse en una única línea:

```
let officeAddress = 'Calle Leganitos, 24';
```

Normalmente utilizaremos esta combinación de declaración y asignación para hacer más sencillo y más claro nuestro código y porque, no nos engañemos, a nadie le gusta escribir de más cuando es innecesario.

- Las variables pueden ser reasignadas, por lo que podemos cambiar el valor que almacenan las veces que queramos.

```
let officeAddress;
officeAddress = 'Calle Leganitos, 24';
officeAddress = 'Calle Mayor, 7';
officeAddress = 'Calle Embajadores, 7';
```

Utilizar variables

Hemos creado nuestra caja y hemos guardado la información en ella. Hasta ahí todo bien, pero nada de esto tiene sentido si no podemos utilizar posteriormente eso que hemos guardado en la caja. Para utilizar ese valor, lo que tenemos que hacer es escribir el nombre de la variable en el lugar en el que queramos utilizar su valor:

```
let earnings = 12020;
let expenses = 5342;
let benefits = earnings - expenses;
```

En el momento en el que se ejecuta el código, las variables se sustituyen por los valores que almacenan. En el ejemplo anterior, la línea final se convertiría en **let benefits = 12020 - 5342;.**

Constantes

Una "variable" cuyo valor nunca varía recibe el nombre de **constante**.

Las constantes funcionan de manera similar a las variables, pero su valor no puede ser reasignado. En JavaScript utilizamos const para declarar y asignar valor a una constante.

```
const officeAddress = 'Calle Leganitos, 24';
```

A veces no tendremos claro cuando utilizar una variable y cuando una constante, ya que puede ser que a priori no estemos seguras de si el valor que vamos a guardar va a cambiar en algún momento o no. Ante la duda, es una buena práctica usar const. Si en algún momento intentamos volver a asignar el valor de una constante, JavaScript no lo permitirá y nos mostrará un error. Esto nos permitirá: evitar errores si no era nuestra intención cambiar el valor, o cambiar ese const por un let si finalmente decidimos que nuestro valor no debería ser constante.

Con esto tendríamos la información necesaria para poder trabajar con variables y constantes sin problemas.

Para finalizar, una de las cualidades fundamentales que las variables y las constantes nos aportan es la capacidad de poder poner nombres descriptivos a nuestro código y poder dividir en varias partes las operaciones que realizamos. Un ejemplo:

```
(620 - 72 - 24) / 4
```

¿Sabrías decir para qué sirve el código anterior? Yo probablemente en el momento de crearlo sí pero pasada una semana estaría en la misma situación que tú. ¿Por qué? Porque el código no es descriptivo y no sabemos qué representa cada número. En ese caso estaríamos realizando un mal código, ya que si estamos trabajando en una empresa y cambiamos de proyecto, cuando le toque trabajar con esto a otra persona probablemente tarde mucho tiempo en poder entenderlo para modificarlo o incluso no pueda y la empresa se vaya a pique (cosa que ha pasado en varias ocasiones en la realidad). Tenemos que tener en cuenta que el código se escribe para que otras personas (o nosotras mismas dentro de un tiempo) puedan entenderlo. Veamos cómo las variables y las constantes nos pueden ayudar en esto:

```
const headerHeight = 72;
const subHeaderHeight = 24;
const screenHeight = 620;
const remainingSpace = screenHeight - headerHeight - subHeaderHeight;
const sections = 4;
const sectionSize = remainingSpace / sections;
```

En este caso el código es mucho más verboso y más largo pero se entiende mucho mejor para que sirve -> establecer la altura de una sección en función de la altura de la pantalla sin tener en cuenta la cabecera y la subcabecera. La idea es que nuestro código sea así, semántico y que se entienda perfectamente qué queremos hacer en cada momento. Por eso, a partir de ahora, todos nuestro ejercicios en JavaScript deberán intentar parecerse a este lo máximo posible para adquirir esta buena práctica muy bien valorada en las empresas

Programación en la web

Ahora que hemos visto cómo funcionan las constantes y las variables para trabajar con valores, vamos a continuar aprendiendo cómo utilizar JavaScript para modificar el contenido y los estilos de nuestras páginas de forma ordenada.

La operación más básica a la hora de trabajar con nuestra página web es obtener información acerca de una etiqueta bien sea para añadir algo a su contenido, modificarlo o eliminarlo directamente. En JavaScript nos referimos a las etiquetas de HTML como elementos (en futuras lecciones veremos el porqué de esto).

Como vimos al principio de la lección con document.querySelector('h1').innerHTML = '¡Hola Mundo!' cambiábamos el texto de a la etiqueta h1 del documento HTML. Vamos a desgranar esto un poco para entenderlo mejor.

Obtener una etiqueta o elemento de HTML

Con document.querySelector('h1') obtenemos el primer elemento h1 que hayamos escrito en nuestro HTML. Podemos usar esta sentencia para recoger un elemento de HTML y guardarlo en una constante.

Como sucedía en las hojas de estilo, acceder a las etiquetas por su nombre puede ser problemático y no es la mejor práctica. document.querySelector() nos permite acceder a los elementos de HTML utilizando los selectores de CSS:

selector de etiqueta

```
const mainTitle = document.querySelector('h1');
```

selector de id

```
const mainTitle = document.querySelector('#mainTitle');
```

selector de clase

```
const mainTitle = document.querySelector('.mainTitle');
```

Modificar el contenido de una etiqueta o elemento de HTML

Cuando hablamos de modificar el contenido de un elemento HTML, nos referimos a cambiar lo que hay entre la etiqueta de apertura y la de cierre. Para realizar este tipo de operaciones utilizaremos la propiedad innerHTML.

Imaginemos que tenemos el siguiente código HTML con un h1 para el título y este contiene el texto "Binvenida" (sí, con un error faltando la e).

```html
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <h1 class="title">Binvenida</h1>
  </body>
</html>
```

Imaginemos que queremos cambiar ese texto por otro sin la falta de la e que evidenciamos en la imagen anterior usando JavaScript. Lo haríamos de la siguiente manera:

```javascript
// Obtenemos el elemento con el que queremos trabajar
//usando document.querySelector()
const titleElement = document.querySelector('.title');

// Cambiamos su contenido con innerHTML
titleElement.innerHTML = 'Bienvenida';
```

De esta forma el contenido de h1 pasaría de ser *"Bienvenida"* a *"Bienvenida"* al cambiar el contenido de ese elemento (.title) el navegador automáticamente actualizará la vista de la página mostrando el texto nuevo. Esto sucede tan rápido que ni siquiera nos dará tiempo a ver la página con el texto anterior, sino que directamente aparecerá con el texto que hemos introducido mediante JavaScript.

NOTA: Debemos dejar claro que la utilidad de JavaScript no es la de solucionar erratas en nuestro código. Esto lo modificaremos directamente en el HTML para corregirlo, pero creemos que este ejercicio ilustra bien cómo funciona innerHTML y para qué sirve.

Ahora supongamos que tras cambiar con JavaScript el texto de *"Bienvenida"* a *"Bienvenida"* queremos hacerlo más personal y queremos que ponga *"Bienvenida, geek girl"* en vez de un soso *"Bienvenida"*.

```
/*
Obtenemos el elemento con el que queremos trabajar usando
document.querySelector()
*/
var titleElement = document.querySelector('.title');

/* Cambiamos el contenido del elemento, indicando que sea igual al actual,
más una nueva palabra añadida*/
titleElement.innerHTML = titleElement.innerHTML + ', geek girl';
```

Bien, ya sabemos cómo obtener un elemento y cómo modificarlo. Vamos a hacer un par de ejercicios para practicarlo.

Obtener información sobre las clases y añadir o quitar clases

Hemos visto cómo añadir o modificar el contenido de un elemento HTML, pero aquí no se queda la cosa. Normalmente también queremos modificar otras cosas de este elemento y una de las que más se suele modificar es la clase para hacer que cambien los estilos de un elemento. Esto nos permitirá añadir clases para ocultar o mostrar un elemento, por ejemplo.

Para trabajar con clases, los elementos ofrecen una propiedad llamada classList. Esta contiene una serie de métodos que nos permitirán, añadir o eliminar una clase o comprobar si el elemento contiene una clase o no.

Imaginemos que tenemos en nuestro CSS una clase .hidden que sirve para ocultar elementos:

```
.hidden {
  display: none;
}
```

Y tenemos un HTML con dos divs .section-a y .section-b, cada uno para una sección diferente de nuestra página:

```
<div class="section-a"></div>
<div class="section-b"></div>
```

Si queremos ocultar la sección B, lo que haremos será añadir la clase **.hidden**, para que se apliquen sus estilos y por tanto ese div tenga display: none; y no se muestre. Para esto tendremos que hacer lo siguiente en JavaScript:

```
/*
Como siempre que queremos trabajar con un elemento de HTML lo obtenemos con
document.querySelector() y lo asignamos a una constante.
*/
const sectionB = document.querySelector('.section-b');

sectionB.classList.add('hidden');
```

Esto hará que el div con clase section-b pase a ser . Como se puede intuir classList.add() sirve para añadir una o más clases a un elemento. En el caso de que quisiéramos añadir más clases, solo tenemos que separarlas con comas:

```
sectionB.classList.add('hidden', 'otraClase', 'otraMas');
// Así hasta aburrirnos
```

Bien ya sabemos cómo añadir clases, veamos cómo eliminar. Supongamos que hemos aplicado añadido la clase .hidden a la sección B y actualmente está oculta y ahora queremos que se muestre de nuevo y, además, se oculte la sección A. Esto lo podemos hacer de la siguiente manera:

```
const sectionA = document.querySelector('.section-a');
const sectionB = document.querySelector('.section-b');

sectionA.classList.add('hidden');
sectionB.classList.remove('hidden');
```

Lo único nuevo de ese código sería classList.remove(). Como se puede deducir, este servirá para quitar una clase a un elemento. Al igual que con add(), podemos quitar varias clases pasando cada una por separado a remove():

```
sectionB.classList.remove('hidden', 'otraClase', 'otraMas');
```

Bonus

getElementById

Hemos visto cómo usar querySelector() para obtener un elemento de HTML usando selectores de CSS. Cuando el selector al que queremos acceder es un id también podemos utilizar getElementById(). Veamos ambas opciones:

```
const mainTitle = document.querySelector('#mainTitle');
```

```
const mainTitle = document.getElementById('mainTitle');
```

En esta última variante el nombre del id (mainTitle) no va precedido de #, no es necesario por que con getElementById ya estamos indicando que el selector será un id.

Este es un método muy usado, y lo encontramos en muchos ejemplos online.

var

En 2015 apareció una nueva versión de JavaScript (llamada ES2015 o ES6) con grandes actualizaciones al escribir nuestro JavaScript, uno de los grandes aportes y novedades fue la declaración de variabales.

Entonces, ¿cómo escribíamos variables y constantes antes de esta versión, cuando solo existía ES5?

Con la palabra clave var.

```
var pageBgColor;
pageBgColor = '#3d7e9a';
```

¿Pero y cómo diferenciamos si el valor podría variar o era constante? Con pequeños trucos de uso en buenas prácticas, como por ejemplo nombrar las constantes en mayúscula.

```
var DAYS_PER_WEEK = 7;
```

Es una convención entre programadores utilizar nombres en **UPPER_SNAKE_CASE** para valores constantes. Muchas programadoras continúan usando esta convención con const.

Esto implica que vamos a encontrar montones de ejemplos con var, e incluso puede ser que nos toque trabajar en algún proyecto más antiguo donde aún se use.

Resumen

Como hemos visto durante esta sesión, programar es simplemente dar órdenes al ordenador para conseguir un resultado a partir de unos datos previos.

Aparte de ver qué es programar también hemos visto:

- Usar variables y constantes en JavaScript para guardar valores y poder utilizarlos después y además que nuestro código sea legible por otras programadoras.
- Leer y modificar el contenido de las etiquetas de HTML.
- Añadir y quitar clases de las etiquetas de HTML.

Taller

EJERCICIO 1

Mensaje de navegador obsoleto

En este ejercicio y con lo poquito que hemos visto hasta ahora de JavaScript, vamos a crear un código que muestre un título con el mensaje "Esta página no es compatible con la versión actual de tu navegador. Por favor actualízalo a la versión más reciente.". Para ello utilizaremos **document.querySelector('h1').innerHTML** tal y como hemos visto en los ejemplos anteriores.

Una vez que lo hayáis realizado podéis enviárselo a algún amigo o familiar y decirle que os diga que le parecen los colores de vuestra nueva web para que pase un rato divertido intentando actualizar el navegador.

EJERCICIO 2

Arreglando errores

Vamos a declarar una constante y a asignarle como valor nuestra dirección. En la siguiente línea, tenemos que vuelva a asignar su valor con la dirección de nuestra compañera.

Ahora abrimos las herramientas para desarrolladores de Chrome (DevTools) y seleccionamos la pestaña Consola. Debería aparecer un mensaje en rojo similar a este: **Uncaught TypeError: <aquí explicación de que está causando el error>.**

Hay que leer la explicación y cambiar el código para que desaparezca el error.

EJERCICIO 3

Hola Mundo

Vamos a crear una página HTML con un párrafo en el que ponga *Hola* y, usando JavaScript, vamos a cambiar ese texto por *Hola Mundo*.

EJERCICIO 4

Seleccionando Geek Girl

Hay que crear una página HTML que contenga un listado con tu nombre y el de tu compañera, y un título que diga "La Geek Girl seleccionada es: ". Usando JavaScript, tenemos que cambiar el título añadiendo el nombre del primer li.

EJERCICIO 5

Protege ese password

A partir de una página HTML en la que ponga "Mi contraseña es: Geek2020" y, usando JavaScript, tenemos que cambiar el texto 'Geek2020' por '**'.

Una cosa importante de innerHTML es que nos permite cambiar el contenido de un elemento y como en JavaScript podemos meter HTML dentro de HTML, innerHTML también nos va a permitir hacer esto, por lo que, si tuviésemos un HTML con una lista vacía (<ul class="list">) y quisiéramos introducir dos lis cada uno con un enlace, podríamos hacerlo de la siguiente manera:

```
const listElement = document.querySelector('.list');
const content = '<li><a href="#">Home</a></li><li><a href="#">Contact</a></li>';
listElement.innerHTML = content;
```

Esto añadirá dos lis a la lista y la página los mostrará automáticamente.

Con esto ya podemos trabajar editando el contenido de nuestra página. A partir de ahora podremos añadir contenido a nuestra web o modificarlo y por tanto hacer nuestra web dinámica de verdad.

EJERCICIO 6

Lorem ipsum

Tenemos que crear una página HTML con un solo div, y usando JavaScript, añadir un h1 con el texto "Lorem ipsum", una imagen con el src http://via.placeholder.com/350x150 y un párrafo con el texto "Lorem ipsum dolor sit amet, consectetur adipisicing elit".

EJERCICIO 7

1, 2, 3, responda otra vez

Crear una página HTML con una lista ul vacía y, usando JavaScript, añadir al contenido de esa lista tres li, el primero tendrá como texto 1, el segundo 2 y el tercero 3.

EJERCICIO 8

Deshabilitando botones

Crear dos botones sencillos con los mismos estilos (padding, borde, color) y, usando JavaScript, al segundo añadirle una clase para que parezca que está desactivado (por ejemplo, aplicarle una opacidad menor).