Hack Academy®

Curso de Front-End

Clase 10

Agenda de la clase

Agenda

- A esta altura deberían poder...
- DOM.
  - Función `querySelector`.
  - Función `querySelectorAll`.
  - Agregar nuevos elementos con `createElement` e `insertAdjacentHTML`.
- Ejercicios.

Antes del repaso…

¿De qué deberían ser capaces a esta altura del curso?

A esta altura deberían poder...

- Usar las etiquetas HTML más importantes → para definir el QUÉ.
  `h1`, `h2`, `p`, `ul`, `li`, `div`, `span`, `form`, `input`, `select`, etc.
- Usar las propiedades CSS más importantes → para definir el CÓMO.
  `font-size`, `color`, `background-color`, `margin`, `padding`, `float`, `position`, `display`, etc.
- Usar Bootstrap.
- Usar, en JavaScript, variables, condicionales (`if/else`), funciones, *arrays* y *loops*.
- Usar Google y Stack Overflow para consultar dudas.
- Indentar su código y usar espacios de forma correcta.
  Ayúdense con herramientas como Prettier.
  Recuerden que un código mal indentado es mucho más difícil de leer, mantener y los hace parecer “amateurs”.
  Además, existen lenguajes en los que un código mal indentado ni siquiera funciona.
- Usar la consola para detectar errores en su código.
- Crear un proyecto en VSC y organizarlo en carpetas `img`, `css` y `js`.

A esta altura deberían saber que...

¡Practicar es fundamental!

Nadie aprende a tocar la guitarra mirando cómo otro la toca. Con la programación sucede lo mismo.

Repaso

Comparadores

Operador	Significado
==	Igual a
===	Estrictamente igual a
!=	Distinto a
!==	Estrictamente distinto a
>	Mayor a
<	Menor a
>=	Mayor o igual a
<=	Menor o igual a

La recomendación es evitar el uso del doble igual (“==”). Por defecto, siempre se sugiere usar el triple igual (“===”).

También se los llama operadores de comparación.

REPASO

Sentencia `if`

Una sentencia `if` se utiliza para especificar si un bloque de código se debe ejecutar o no dependiendo del valor de cierta condición.

```
if (CONDICIÓN) {

    // Bloque de código a ejecutar

    // si la condición se cumple...

}
```

```
if (LLUEVE) {

    // Buscar el

    // paraguas...

}
```

👉 La condición es algo que puede ser `true` o `false`. En realidad, *truthy* o *falsy*.

REPASO

Sentencia `else`

👉 Aprovechar **VSC** al máximo. Usar el auto-complete al crear un `if-else`.

```
if (CONDICIÓN) {

    // Bloque de código a ejecutar si la condición se cumple...

} else {

    // Bloque de código a ejecutar si la condición NO se cumple...

}
```

Nota: El `else` no necesita de una condición.

REPASO

Sentencia `else if`

Ejemplo:

Supongamos que cierto comercio está abierto entre las 09:00 y 18:00 horas. Fuera de dicho horario, está cerrado.

```
if (hora > 18) {
    alert("Está cerrado, es tarde.");
} else if (hora < 9) {
    alert("Está cerrado, es temprano.");
} else {
    alert("Está abierto");
}
```

REPASO

¿Qué es un *loop*?

Bloque de código que se ejecuta repetidamente mientras se da cierta condición.

¿Por qué usar *loops*?

Para no escribir código de más (repetido), y por lo tanto:

- Ahorrar tiempo.
- Tener un código más mantenible.

REPASO

`for` *Loops*

Forma general:

👉 Aprovechar **VSC** al máximo. Usar el auto-complete al crear un `for`.

```
for (SENTENCIA_INICIAL;  CONDICIÓN;  SENTENCIA_FINAL) {

    // Bloque de código que se ejecutará repetidamente...

}
```

- **Sentencia Inicial**: se ejecuta una sola vez, antes de la primer iteración.
- **Condición**: es una expresión que se evalúa antes de cada iteración del loop. El loop se ejecuta mientras la condición sea `true`.
- **Sentencia Final**: se ejecuta inmediatamente después de cada iteración.

REPASO

Recorrer un *Array* – Usando un `for`

```
var marcas = ["BMW", "Peugeot", "Chevrolet", "Subaru", "Nissan"];
```

```
for (var i = 0; i < marcas.length; i++) {

    console.log(marcas[i]);

}
```

Esta es la forma más “tradicional” o “antigua” de recorrer un array.

Si bien la sintaxis del `for` es menos amigable con respecto a otros métodos más modernos utilizados para recorrer arrays (ej: `forEach`), hay algunos casos en los que puede ser útil debido a su flexibilidad.

Nota: El nombre de la variable `i` es arbitraria; por ejemplo, se le podría haber llamado `indice`.

REPASO

Recorrer un *Array* – Usando un `for...of`

```
var marcas = ["BMW", "Peugeot", "Chevrolet", "Subaru", "Nissan"];
```

```
for (var item of marcas) {

    console.log(item);

}
```

El `for...of` (ver docs) está disponible desde la versión ES6 de JavaScript (año 2015) y no funciona en ninguna versión de Internet Explorer. Posiblemente es la sintaxis más sencilla que existe para recorrer arrays.

Nota: El nombre de la variable `item` es arbitraria; por ejemplo, se le podría haber llamado `marca`.

REPASO

Ejercicios fundamentales de la última clase

- Definir una función llamada `mostrarArray` que recibe como argumento un *array* cualquiera y lo muestra (imprime) en la consola. Cada elemento debe ir en una nueva línea.

- Definir una función llamada `estaElemento` que recibe como argumentos un array cualquiera y un elemento cualquiera, y retorna `true` si dicho elemento se encuentra en el *array*. En caso contrario retorna `false`.

- Definir una función llamada `maximo` que recibe como argumento un *array* cualquiera y retorna el elemento más grande de la lista. Suponer que el *array* recibido es una lista de números.

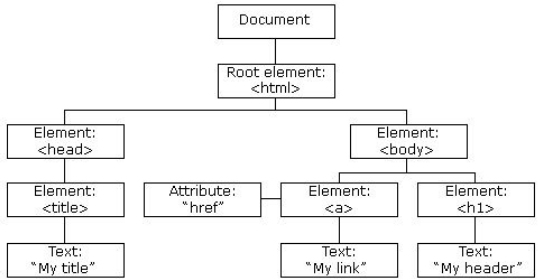
👉 Si quieren, los podemos hacer de vuelta, entre todos.

DOM

¿Qué es el DOM?

Más adelante se verá el concepto de **objeto**.

- DOM = Document Object Model.
  👉 Ver documentación.
- Cuando se carga una página, el *browser* crea un DOM de la página a partir del HTML. Es una representación del documento que tiene una estructura jerárquica con forma de árbol.
- DOM ≠ HTML.
- DOM ≈ El código que se ve en Dev Tools.
- DOM es una API (más adelante se verá una definición formal de API).
- JavaScript permite modificar el DOM (pero no modificar el HTML).



Objeto `window`

Escribir en la consola:

```
window;
```

El *objeto* `window` representa la ventana abierta del navegador. También se le llama BOM (*Browser Object Model*) y permite que JavaScript interactúe con toda la ventana, no sólo con el documento HTML.

Esto permite hacer consultas, como por ejemplo, obtener el ancho de la ventana (incluyendo la *scrollbar*) o información de la ubicación (URL):

```
window.innerWidth;
window.location;
```

Objeto `document`

Escribir en la consola:

```
document;
```

El *objeto* `document` es una referencia al documento HTML dentro de la ventana.

Es el objeto con el que interactuaremos más frecuentemente.

Es quien nos permite acceder y modificar el DOM.

Nota: También se puede acceder a `document` de esta forma:

```
window.document;
```

Manipular el DOM con JS (1/3)

El DOM cuenta con un montón de funciones (métodos) que permiten manipularlo.

Por ejemplo, la función `querySelector` permite seleccionar un elemento de la página web (usando selectores de CSS).

```
var titulo = document.querySelector("h1");
```

Luego es posible manipular dicho elemento.
Por ejemplo, es posible cambiarle el texto:

```
titulo.textContent = "Cursos de Programación";
```

Esto es manipulación de DOM usando JavaScript “puro”, sin uso de ninguna librería ni framework. Ej: jQuery.

Manipular el DOM con JS (2/3)

El código de la diapositiva anterior, también se podría haber escrito en una sola línea de código. 👉 ¡Ingresar a esta página y probarlo!

```
document.querySelector("h1").textContent = "Cursos de Programación";
```

Probar también:

```
document.querySelector("p").style.color = "blue";
document.querySelector("p").style.fontWeight = "bold";
document.querySelector(".row").style.border = "10px solid red";
```

Manipular el DOM con JS (3/3)

¿Qué se puede hacer con JavaScript y el DOM?

- Modificar todos los elementos HTML en la página.
- Modificar todos los atributos HTML en la página.
- Modificar todos los estilos CSS en la página.
- Remover elementos HTML y atributos.
- Agregar nuevos elementos HTML y atributos.
- Reaccionar a eventos que suceden en la página.

Un poco de historia…

Un poco de historia...

Antiguamente, manipular el DOM era difícil. No existía una forma estandarizada de hacerlo, y que fuese compatible con todos los navegadores.

Por eso, en 2006 se creó una librería llamada jQuery con el fin de resolver dicho problema, y lo logró de forma muy satisfactoria. Gracias a su facilidad de uso, durante una década jQuery fue la forma más sencilla de manipular el DOM, y hasta el día de hoy sigue siendo librería sumamente popular, aunque de popularidad decreciente.

Sin embargo, los navegadores han mejorado notoriamente. Hoy en día es posible manipular el DOM usando JavaScript “puro”, de forma sencilla, tal como lo vimos en la última clase. 🎉

Además, en los últimos años han surgido otras librerías y frameworks (más modernos) como React, Vue.js y Angular que han hecho que jQuery vaya perdiendo terreno.

Función `querySelector`

Función `querySelector` (1/3)

La función `querySelector` retorna el primer elemento de la página que *matchee* con el selector especificado.

Si no hay ningún elemento en la página que *matchee* con dicho selector, la función retorna `null`.

Ejemplo:

```
var titulo = document.querySelector("h1");
```

Nota: Esta funcionalidad está disponible desde Internet Explorer 9.

Función `querySelector` (2/3)

Una vez que el elemento fue seleccionado, se le pueden aplicar transformaciones como, por ejemplo:

```
var titulo = document.querySelector("h1"); // Seleccionar el `h1` de la página.


titulo.textContent = "¡Hola alumnos de Hack Academy!"; // Asignarle un texto.
titulo.style.color = "blue"; // Cambiarle el color de letra.
titulo.style.backgroundColor = "red"; // Cambiarle el color de fondo. ¡Notar el camelCase!
titulo.style.display = "none"; // Oculta el elemento.
titulo.classList.add("importante"); // Agregarle la clase `importante`.
titulo.classList.remove("importante"); // Quitarle la clase `importante`.
```

Función `querySelector` (3/3)

Escribir el siguiente código en VSC y probarlo en el navegador. Actualizar la página varias veces. ¿Qué se nota?

En archivo HTML:

```
<p>
     Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor.
     Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus
     mus.
</p>
```

En archivo CSS:

```
.importante {
  color: red;
}
```

En archivo JS:

```
document.querySelector("p").classList.add("importante");
```

Ejercicio 1

Ejercicio 1

Escribir una **función** llamada `buscarTitulo` que al ejecutarse muestre una **alerta** con el mensaje *“Error: Falta título”* **en caso de** que la página no contenga ningún elemento `<h1>`. **En caso contrario**, **cambiarle el tipo de letra** al elemento `<h1>` a "Times New Roman".

Para ello será necesario utilizar la propiedad `style.fontFamily`.

Ver documentación (simple): http://www.w3schools.com/jsref/prop_style_fontfamily.asp.
Ver documentación (completa): https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style y https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference.

Probarlo en:

- https://ha.dev
- https://ha.dev/?withoutH1=yes

Ejercicio 2

Ejercicio 2

Escribir **una función** llamada `toggleMenu` que pueda ser usada en una página web hecha con Bootstrap y que tenga un Dropdown Menu. Por ejemplo, esta función se podrá probar en este sitio web, a través de la consola.

Si el elemento con clase `.dropdown-menu` está oculto, hacerlo aparecer seteándole `display:block`.

En **caso contrario**, hacerlo desaparecer seteándole `display:none`.

Para ello será necesario utilizar la propiedad `style.display`.

Ver documentación (simple): http://www.w3schools.com/jsref/prop_style_display.asp.
Ver documentación (completa): https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style y https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference.

Función `querySelectorAll`

Función `querySelectorAll` (1/2)

La función `querySelectorAll` retorna un *array* (lista) con todos los elementos de la página que *matcheen* con el selector especificado.

Si no hay ningún elemento en la página que *matchee* con dicho selector, la función retorna una lista vacía.

Ejemplo:

```
var parrafos = document.querySelectorAll("#noticias p");
```

Nota: Esta funcionalidad está disponible desde Internet Explorer 9.

Función `querySelectorAll` (2/2)

Dado que `querySelectorAll` retorna una lista de elementos, es necesario recorrerla para poder aplicarles alguna transformación.

Ejemplo:

```
var subtitulos = document.querySelectorAll("h2");


for (var subtitulo of subtitulos) {
  subtitulo.style.color = "#FF6600";
}
```

Agregar un nuevo elemento al DOM

Agregar un nuevo elemento al DOM (1/2)

Además de manipular elementos existentes en el DOM, también es posible crear nuevos elementos.

Una forma de hacerlo es con la función `createElement` y, luego de creado, insertarlo en algún lugar de la página con la función `append`:

```
var nuevoParrafo = document.createElement("p"); <p></p>
nuevoParrafo.textContent = "Hola alumnos de Hack Academy.";


document.querySelector("#noticias").append(nuevoParrafo);
```

Nota: La función `append` no está disponible en ninguna versión de Internet Explorer.

Agregar un nuevo elemento al DOM (2/2)

Otra forma de agregar elementos a la página es usando la función `insertAdjacentHTML` que acepta dos *strings* como parámetros. El primero indica la posición donde se desea insertar el HTML (respecto del elemento seleccionado) y el segundo indica el HTML que se desea insertar. Ejemplo:

```
document.querySelector("#noticias").insertAdjacentHTML(
  "beforeend",
  `<div>
     <h2>Hack Academy</h2>
     <p class="importante">Institución educativa especializada en cursos de programación.</p>
     <a href="https://ha.dev">Link a Hack Academy</a>
   </div>`
);
```

Detectar un “click”

Detectar un “click”

Con JavaScript se pueden detectar ***eventos*** que suceden en una página web. 👉Se profundizará al respecto en una próxima clase.

La función `addEventListener` permite detectar un “click” que sucede sobre un elemento de la página. Por ejemplo, se puede detectar un “click” sobre un botón:

```
document.querySelector("#btn-guardar").addEventListener("click", function () {
  /**
   * Bloque de código que se ejecuta cuando el usuario
   * hace click en el elemento de id `btn-guardar`.
   */
});
```

La función `addEventListener` recibe dos parámetros (un *string* y una función). En este ejemplo:

- El *string* que se está pasando como primer argumento indica que se quiere “escuchar clicks” que sucedan sobre un botón de la página llamado `btn-guardar`.
- La función que se está pasando como segundo argumento es particular porque no tiene nombre. Es lo que se conoce como función anónima y es un caso particular de *Function Expression*. Esta función se ejecutará cada vez que un usuario haga click sobre el botón `btn-guardar`.

Ejercicio 3

Ejercicio 3

1. Crear una carpeta en el Escritorio (o donde prefieran) con el nombre `HA_Clase10_Ejercicio3`.
2. Abrir dicha carpeta en VSC (como un **proyecto**). Esto se puede hacer yendo al menú `File > Open Folder` y seleccionar la carpeta.
3. Desde VSC, crear un archivo llamado `index.html` dentro de la carpeta.
4. Desde VSC, crear una carpeta `js` y dentro de la misma el archivo `app.js`.
5. Linkear el HTML al JS.

Ejercicio 3 (cont)

⚠️ **Sugerencia**: Asuman que la página está inicialmente de color verde.

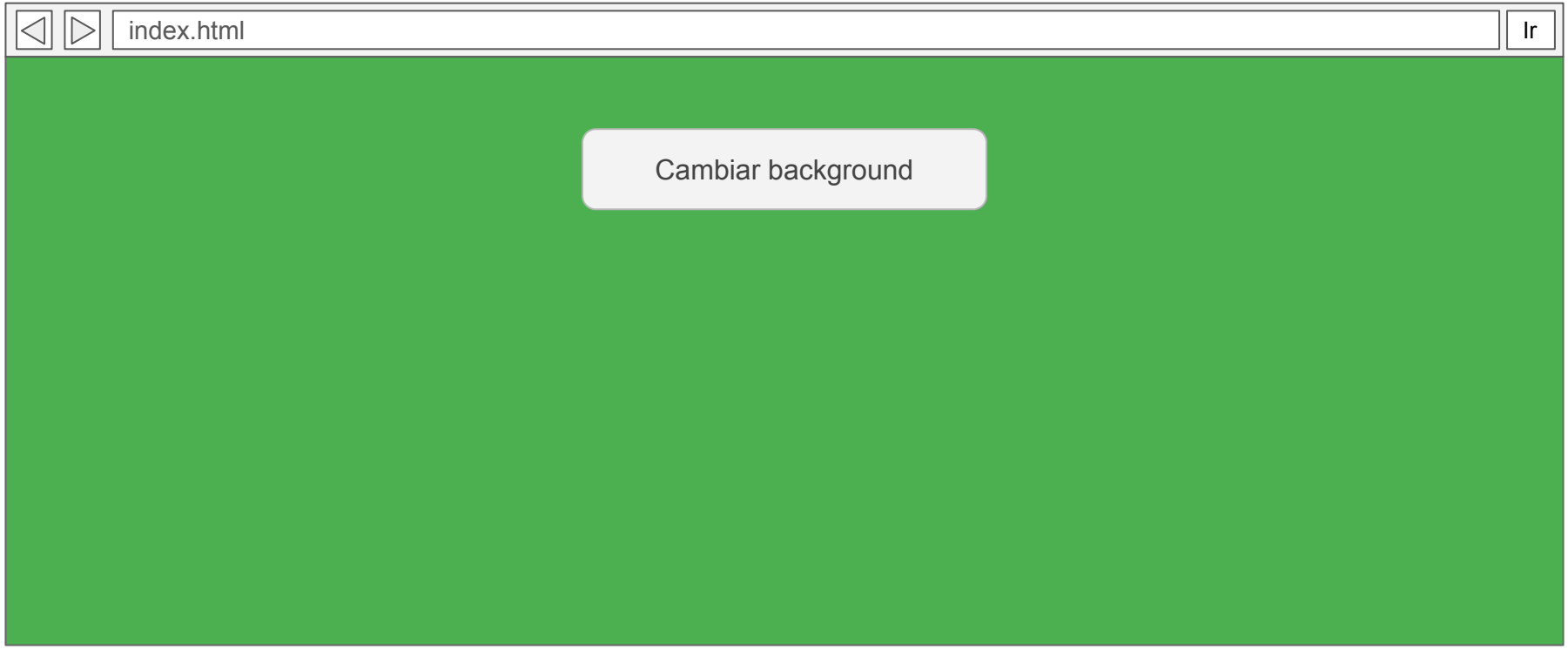
6. Crear en la página un botón con el texto: *“Cambiar Background”*.

    a. Si el color de fondo de la página es rojo, al hacer click en el botón, el color debe cambiar a verde.

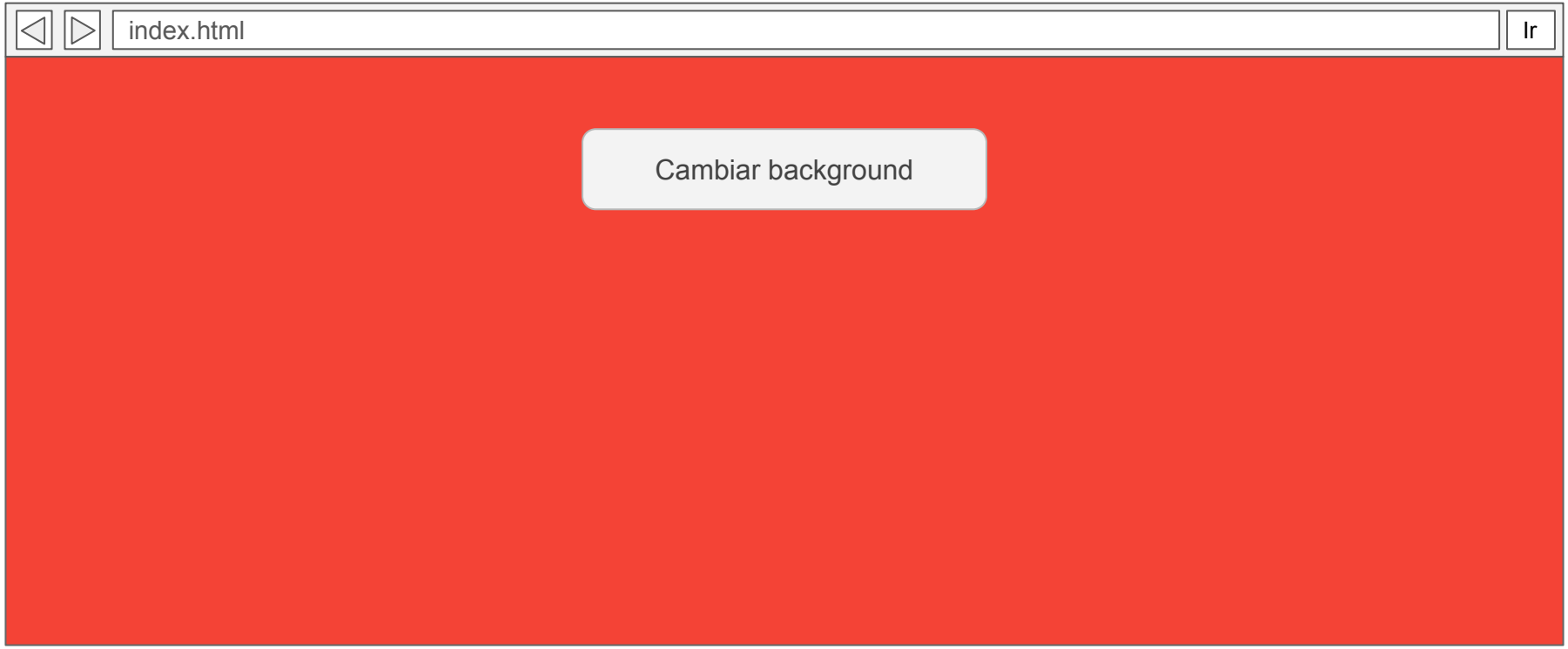
    b. Si el color de fondo de la página es verde, al hacer click en el botón, el color debe cambiar a rojo.

    c. Usar colores de rojo y verde disponibles en esta página: https://material.io/design/color/the-color-system.html#tools-for-picking-colors (sección *"2014 Material Design color palettes"*).
    Como alternativa se puede buscar alguna otra paleta de colores en Google.

Ejercicio 3 (cont)



Ejercicio 3 (cont)



Ejercicio 4

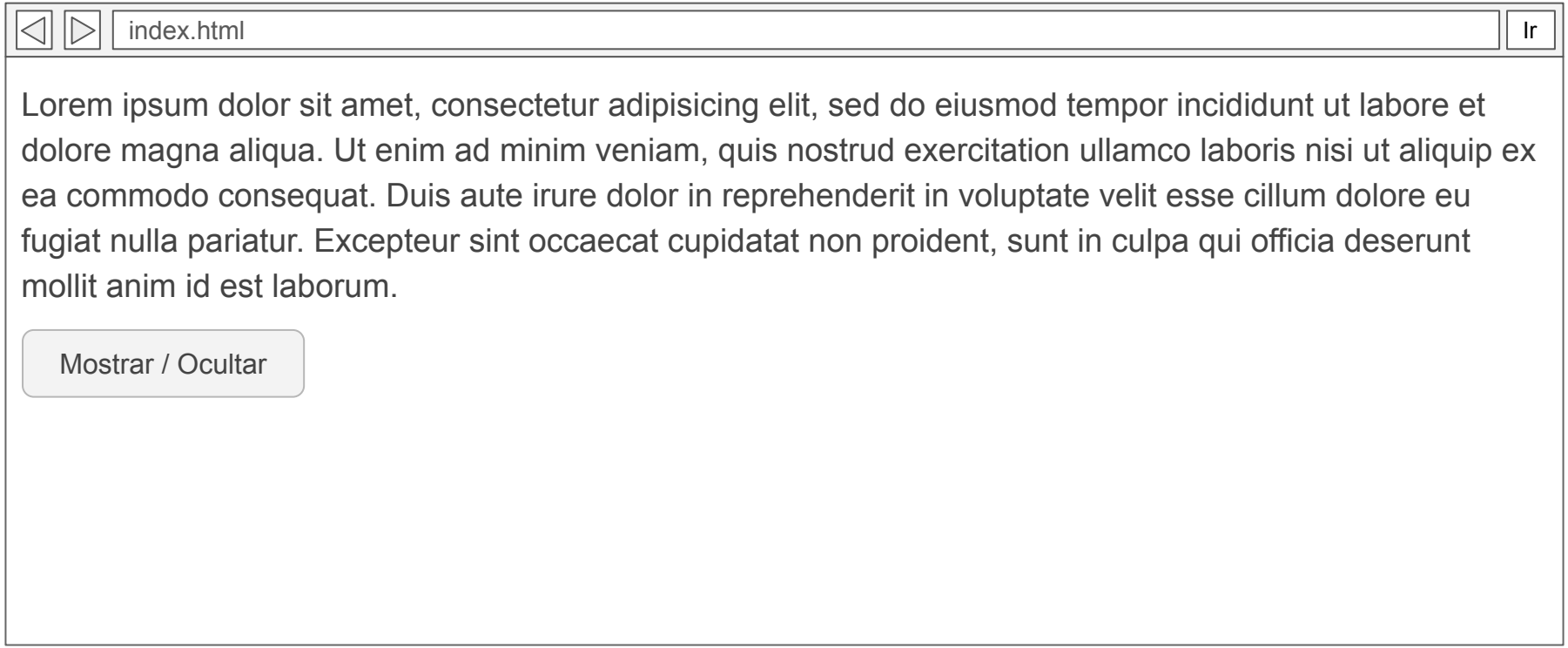
Ejercicio 4

1. Crear una carpeta en el Escritorio (o donde prefieran) con el nombre `HA_Clase10_Ejercicio4`.
2. Abrir dicha carpeta en VSC (como un **proyecto**). Esto se puede hacer yendo al menú `File > Open Folder` y seleccionar la carpeta.
3. Desde VSC, crear un archivo llamado `index.html` dentro de la carpeta.
4. Desde VSC, crear una carpeta `js` y dentro de la misma el archivo `app.js`.
5. Linkear el HTML al JS.

Ejercicio 4 (cont)

1. En `index.html` agregar un párrafo `<p>` con texto *“Lorem Ipsum…”*.
2. Crear un botón con el texto “Mostrar/Ocultar”.
    a. Si el párrafo está visible, al hacer click en el botón, el párrafo se debe ocultar.
    b. Si el párrafo está oculto, al hacer click en el botón, el párrafo se debe mostrar.
3. Probar de mostrar y ocultar el párrafo usando diferentes efectos.

Ejercicio 4 (cont)



Ejercicio 4 (cont)

EXTRA

Ejercicio 5

Algunos de estos ejercicios requieren conceptos teóricos adicionales a los vistos en clase.

EXTRA

Ejercicio 5

A. Definir una función llamada `formatP` que pinte de color verde y setee en `2rem` el tamaño de letra de todos los párrafos que haya en una página.

B. Definir una función llamada `lorem` que inserte el texto *"Lorem ipsum..."* en todos los párrafos que haya en una página. Probarla en distintas páginas a través de la consola.

C. Definir una función llamada `alertRepetido` que reciba como argumento un texto `S` y un número `N` y despliegue un `alert` conteniendo el texto `S` repetido `N` veces. Ej: `alertRepetido("hola",3)` produce un `alert` con *"hola hola hola"*.

D. Definir una función llamada `alarma` que haga que el `background-color` de una página se ponga de color rojo intermitentemente durante 5 segundos. Es decir, que durante 5 segundos el `background-color` debe cambiar varias veces de color entre rojo y blanco.
Sugerencia: usar las funciones de JS: `setInterval`, `setTimeout` y `clearInterval`.