



Curso de Front-End

Clase 13



Agenda de la clase



Agenda

- Repaso.
- `while` Loops.
- Loop infinito.
- Eventos.
- Ejercicios.



Repaso



¿Cómo se **declara** una función? – Ejemplo

```
function validarNombre(nom) {  
    if (typeof nom !== "string") {  
        return false;  
    }  
    if (nom === "") {  
        return false;  
    }  
    return true;  
}
```

```
function maximoDeDos(num1, num2) {  
    var resultado;  
    if (num1 <= num2) {  
        resultado = num2;  
    } else {  
        resultado = num1;  
    }  
    return resultado;  
}
```

Esta forma de definir una función se llama "Function [Declaration](#)".



¿Cómo se **llama** a una función? – Ejemplos

```
validarNombre("María"); // Retorna true.  
validarNombre("aaaaa"); // Retorna true.  
validarNombre(""); // Retorna false.  
validarNombre(30000); // Retorna false.  
validarNombre(true); // Retorna false.  
validarNombre(); // Retorna false.
```

```
maximoDeDos(34, 5); // Retorna 34.  
maximoDeDos(5, 34); // Retorna 34.  
maximoDeDos(5, 5); // Retorna 5.  
maximoDeDos(-67, -67); // Retorna -67.
```



Creación de un *Array*

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

Acceder a un elemento de un *Array*

```
marcas[2];
```

Algunas propiedades y métodos (funciones) de un *Array*

```
marcas.length; // Retorna el largo del array "marcas".  
marcas.push("Fiat"); // Agrega "Fiat" al final del array (sin especificar el índice).  
marcas.pop(); // Elimina el último elemento del array.  
marcas.splice(1,2); // Elimina 2 elementos del array, desde la posición 1 inclusive.
```

Documentación: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#Methods_2



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`.



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`.



while *Loops*



“A veces necesitamos ejecutar un bloque de código repetidamente, pero no sabemos cuántas veces”



while Loop – Ejemplo (1)



La alarma de un auto suena indefinidamente, **hasta** que el dueño la desactiva.
Dicho de otra manera, la alarma sonará **mientras** nadie la desactiva.



while *Loop* – Ejemplo (2)



Un equipo de A.C. tirará aire frío **hasta** que la temperatura del salón llegue a 23°C. Dicho de otra manera, el A.C. tirará aire frío **mientras** la temperatura del salón sea mayor a 23°C.



while *Loop* (1/3)

El `while` *loop* se parece a una sentencia `if`.

En ambos casos se ejecuta un bloque de código (definido por unas llaves `{ }`) dependiendo de una condición. La diferencia está en que el `while` ejecuta el código **repetidamente hasta que la condición deje de cumplirse**.

```
if (CONDICIÓN) {  
    // Bloque de código que se ejecuta  
    // una vez si la condición se  
    // cumple.  
}
```

```
while (CONDICIÓN) {  
    // Bloque de código que se ejecuta  
    // repetidamente mientras  
    // la condición se cumpla.  
}
```

El `while` se ejecuta mientras la condición se cumpla, o dicho de otro modo, hasta que la condición sea `false` (falsy).



while *Loop* (2/3)

Ejemplo:

```
while (tempSalon > 23) {  
    console.log("Aire acondicionado prendido.");  
    // Largar aire frío.  
    // Medir temperatura nuevamente.  
}
```



while *Loop* (3/3)

Otro ejemplo:

```
var contador = 0;
while (contador < 50) {
    console.log(contador);
    contador = contador + 2;
}
```

⚠ Si bien este ejemplo es correcto, hubiese sido una mejor práctica resolverlo con un `for`.

¿Qué pasaría si se omitiese la línea `contador = contador + 2;` ?



Loop infinito



Loop infinito (1/2)

El *loop* infinito sucede cuando se ejecuta un bloque de código infinitamente.

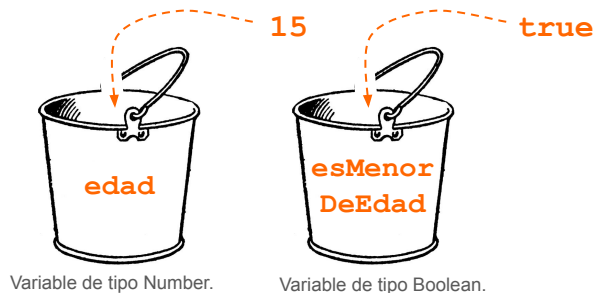
Deben evitarlo para no “**trancar**” el navegador y/o la computadora.

```
while ( 10 > 5 ) {  
    // El código aquí adentro se ejecutará infinitas veces.  
}
```

Para evitarlo es muy importante que la condición del loop sea falsa en algún momento.

Loop infinito (2/2)

Ejemplo:



```
var edad = 15;
var esMenorDeEdad = (edad < 18);

while (esMenorDeEdad) {
  console.log("Tu edad es: " + edad);
  console.log("Lo sentimos, no podemos venderle alcohol a menores de 18 años.");
  edad++; // Sentencia que "simula" el paso de un año.
}
```

Este código, ¿entra en *loop* infinito? En caso afirmativo, ¿qué se puede hacer para evitarlo?





for **vs.** while



¿Cuándo usar `while` y cuándo `for`?

- Todo lo que pueden hacer con un `while` lo pueden hacer con un `for`, y viceversa.
- Si conocen de antemano cuántas veces se ejecutará el *loop*, suele ser mejor opción usar un `for`. Es decir, si quieren que un bloque de código se ejecute “n” veces, es más prolijo usar un `for`.
- Si no conocen de antemano la cantidad de iteraciones que tendrán el loop, probablemente sea mejor opción usar un `while`.

En su día a día, probablemente usen mucho más los `for` que los `while`.



¿Cuándo usar `while` y cuándo `for`? (cont)

```
var contador = 0;
while (contador <= 100) {
  console.log(contador);
  contador++;
}
```

Aquí se ve claramente cómo un `while` y un `for` pueden ser usados de forma equivalente.

```
for (var contador = 0; contador <= 100; contador++) {
  console.log(contador);
}
```



¿Cuándo usar `if`, `else`, `for` y `while`?



¿Cuándo usar `if`, `else`, `for` y `while`?

Texto en español	Generalmente se traduce a este código
Si pasa A, entonces hacer B1...	<code>if (A) { B1; }</code>
...sino, hacer B2.	<code>else { B2; }</code>
Repetir B, X cantidad de veces.	<code>for (var i = 1; i <= X; i++) { B; }</code>
Repetir B, mientras se cumpla A.	<code>while (A) { B; }</code>

A es una condición.

B es un bloque de código de una o más sentencias.

Nota: Las indentaciones están mal hechas con tal de que entrara el código en la pantalla.



Eventos



¿Qué es un evento?

- Es “algo” que sucede mientras un usuario interactúa con una página web.
- Ejemplos de eventos:
 - El usuario presiona una tecla.
 - El usuario hace click en un botón.
 - El usuario agranda o achica una ventana.
 - Una página se terminó de cargar.
- Los eventos se detectan con la función `addEventListener` de JavaScript.
- Documentación: <https://developer.mozilla.org/en-US/docs/Web/Events>.

Tipos de eventos (1/3)

- Generados con el mouse (MouseEvent):

- `click` ←
- `dblclick`
- `mouseenter`
- `mouseleave`
- `mouseover`
- `mousemove` (https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onmousemove)
- **etc.**

Probablemente el
evento más usado.



Tipos de eventos (2/3)

- Generados con el teclado (KeyboardEvent):

- `keydown`
- `keypress` (produce un caracter) → Está deprecado (mejor usar `keydown`).
- `keyup`

- Generados con el browser/ventana (UIEvent):

- `load`
- `resize`
- `scroll`
- `etc.`



Otro evento bastante usado.

Tipos de eventos (3/3)

- Generados en un formulario:

- `change`
- `reset`
- `submit`
- `input`



Otros eventos muy usados.

- Otros:

- `focusin`
- `focusout`



Detectar un evento con JavaScript (1/2)

La función [addEventListener](#) permite **detectar eventos** que suceden sobre un elemento cualquiera de la página.

```
const unElemento = document.querySelector("#elem");  
  
unElemento.addEventListener("nombreDeUnEvento", nombreDeUnaFunción);
```

👉 El segundo parámetro de la función `addEventListener` (llamada *callback*) también se podría haber declarado como una **función anónima**.



Detectar un evento con JavaScript (2/2)

En caso de ser necesario, es posible “capturar” el **objeto Event** que se crea (automáticamente) cuando se genera el evento. Luego se puede analizar dicho objeto.

```
document.body.addEventListener("mousemove", function (event) {  
  
    console.log(event);  
  
});
```

Notar que el argumento `event` lo crea y lo pasa el *browser* automáticamente. Además, podría tener otro nombre, como por ejemplo: `e`.



Ejercicios






Ejercicio 1

1. Crear una carpeta en el Escritorio (o donde prefieran) con el nombre `HA_Clase13_Ejercicio1`, a partir del zip **Proyecto Base**.
2. Abrir dicha carpeta en VSC (como un **proyecto**). Esto se puede hacer yendo al menú: `File > Open Folder`.
3. Crear una página que contenga un formulario tal como se muestra en las diapositivas a continuación.
4. En caso de que el usuario escriba un email válido, el `input` se debe poner de color verde. En caso contrario, de color rojo. Los cambios de color deben suceder **a medida que el usuario ingrese caracteres** en el `input`.
5. Se compartirá una función llamada `validarEmail` que recibe un string y retorna `true` en caso de que se trate de un email válido. En caso contrario, retorna `false`.

Ejercicio 1 (cont)





 index.html 

Suscribite a nuestro newsletter

Ingresar E-mail

Ejercicio 1 (cont)



 index.html Ir

Suscribite a nuestro newsletter




Ingresar E-mail

Sugerencia: crear una clase `valid` que contenga los estilos del `input` cuando es válido. Como alternativa se podría usar alguna clase ya existente en Bootstrap.

Obtener el valor utilizando `.value`.
Ej:
`document.querySelector("#email").value;`

Ejercicio 1 (cont)



 index.html 

Suscribite a nuestro newsletter

Ingresar E-mail

Sugerencia: crear una clase `not-valid` que contenga los estilos del `input` cuando es inválido. Como alternativa se podría usar alguna clase ya existente en Bootstrap.

Ejercicio 1 (cont)

En este caso no es necesario entender completamente cómo es el funcionamiento interno de esta función. Se la puede ver como una "caja negra" que recibe cierto *input* (un string) y retorna cierto *output*. El código también está disponible para [copiar aquí](#).



```
/**
 * Función que valida si un email es válido.
 * Recibe un string y retorna true si corresponde a un email válido.
 * En caso contrario, retorna false.
 */

function validarEmail(email) {
    var re =
    /^((([^<>()\\[\]\\. ,;: \s@"]+)(\.[^<>()\\[\]\\. ,;: \s@"]+)*|("[.]+"))@((\[[0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\]|([a-zA-Z\-\0-9]+\. )+[a-zA-Z]{2,})))$/;
    return re.test(String(email).toLowerCase());
}
```

Fuente: <https://stackoverflow.com/questions/46155/how-to-validate-an-email-address-in-javascript>.



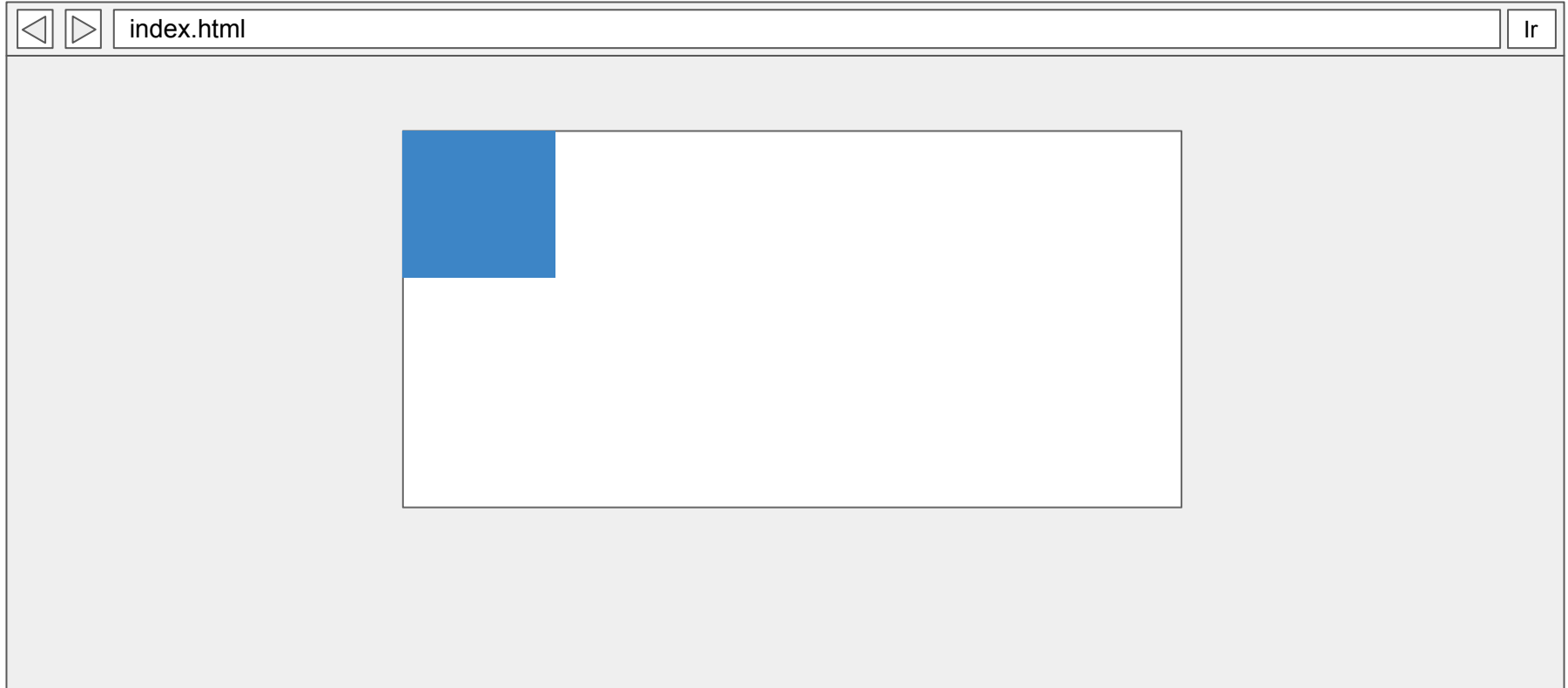
Ejercicio 2

1. Crear una carpeta en el Escritorio (o donde prefieran) con el nombre `HA_Clase13_Ejercicio2`, a partir del zip **Proyecto Base**.
2. Abrir dicha carpeta en VSC (como un **proyecto**). Esto se puede hacer yendo al menú: `File > Open Folder`.
3. Crear una página que contenga dos `<div>` (uno adentro del otro) como la figura que se muestra a continuación (uno blanco y otro azul).
4. Al usar las flechas del teclado, el usuario debe poder mover el `<div>` interior de una esquina a otra.

Ayuda: <http://stackoverflow.com/questions/1402698/binding-arrow-keys-in-js-jquery>. Está resuelto usando un `switch` pero se puede lograr lo mismo usando `if`'s.

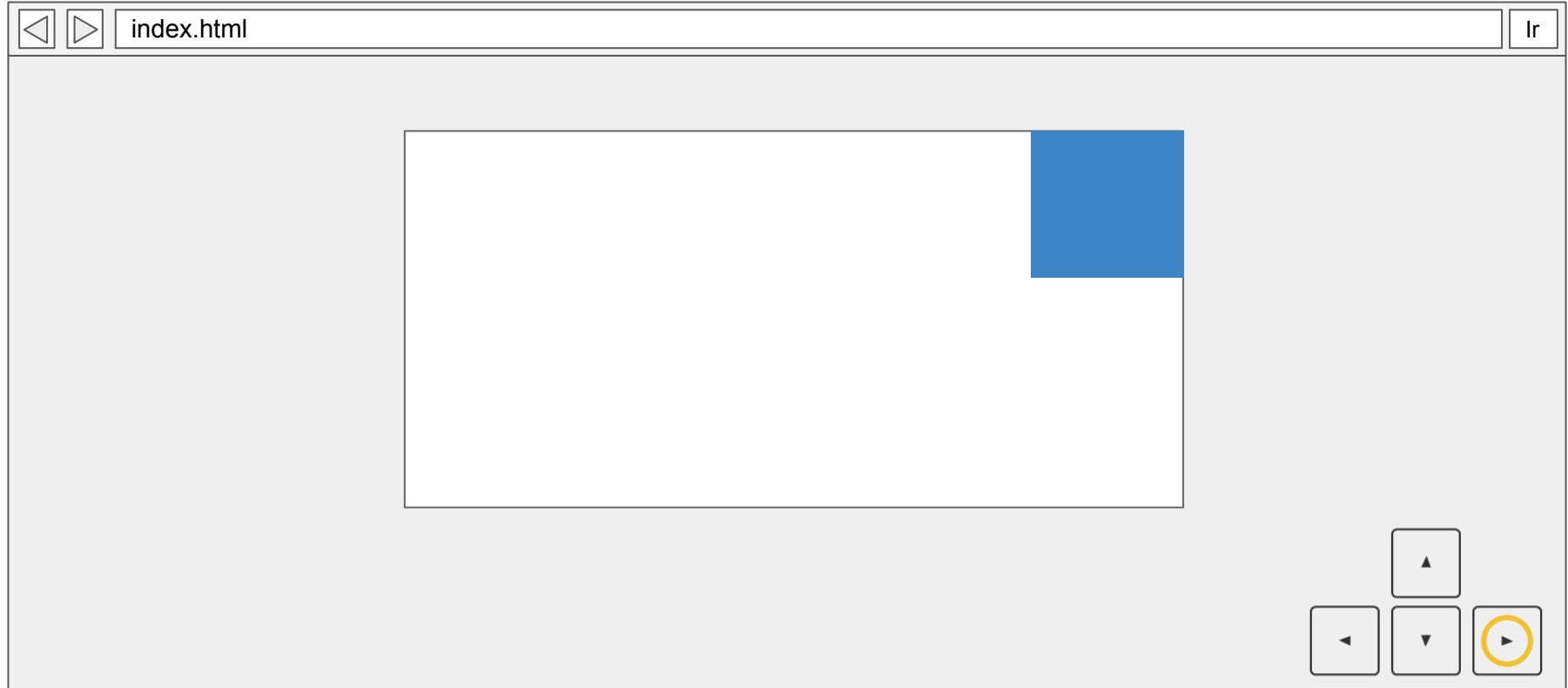
Ejercicio 2 (cont)

EXTRA & AVANZADO



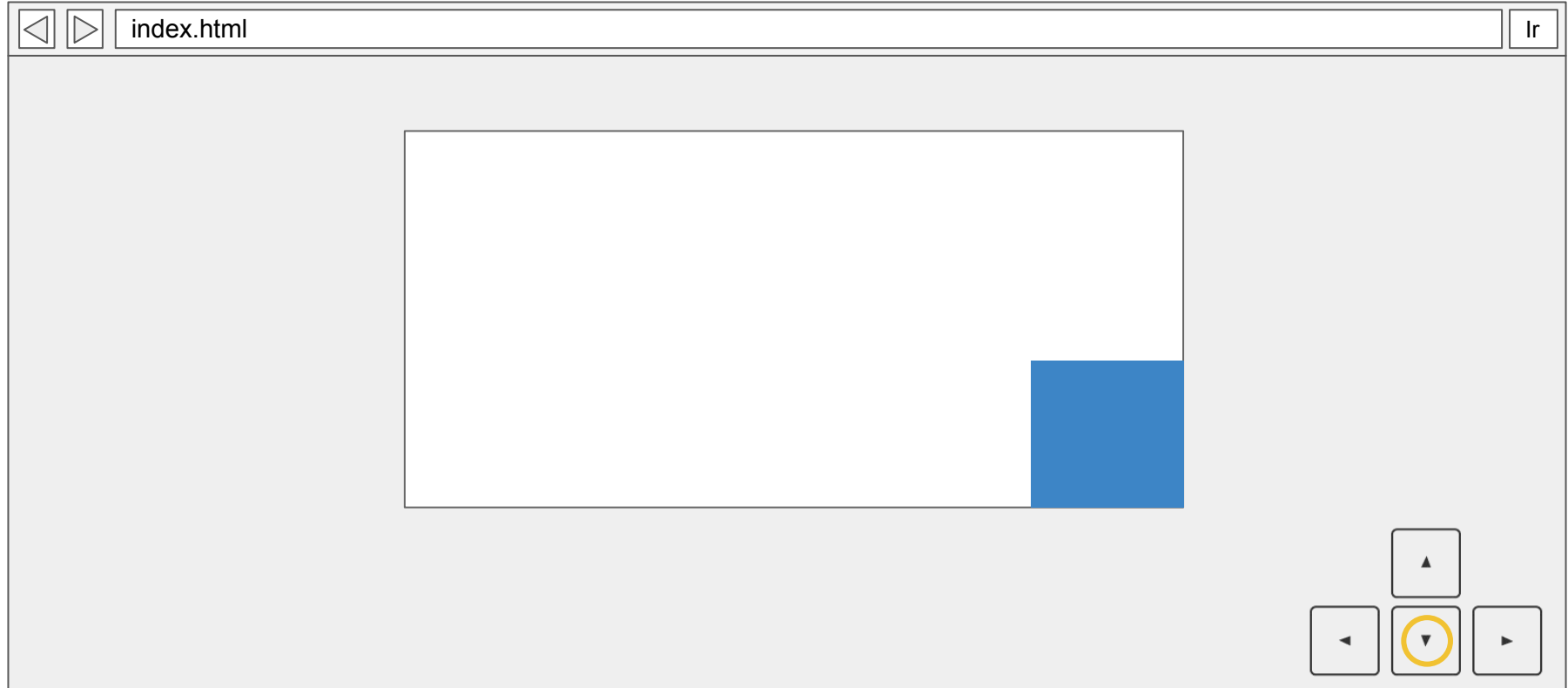
Ejercicio 2 (cont)

EXTRA & AVANZADO



Ejercicio 2 (cont)

EXTRA & AVANZADO



Ejercicio 2 (cont)

EXTRA & AVANZADO

