UNIVERSIDAD DE LA FRONTERA
Facultad de Ingeniería y Ciencias
Depto. Ciencias de Computación e Informática

Introducción a JavaScript

¿Qué es JavaScript?

JavaScript (abreviado JS) es un lenguaje de programación interpretado. Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario de páginas web, lo que puede conllevar a la implementación de efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Añadir JS a nuestro HTML

Hay dos formas de vincular código JS a nuestro HTML.

La primera es escribir directamente el código JS dentro de etiquetas <script> **(…)** </script> en nuestro archivo HTML.

La segunda forma y la más recomendada es llamar a un archivo ".js" por medio de la etiqueta <script> especificando en el atributo src la ruta del archivo ".js":

```
<script src="codigo.js"></script>
```

Sintaxis

Comentarios

Para agregar comentarios en nuestro código JS, podemos hacerlo de dos formas.

- Para una línea:

```
// comentario
```

- Para una o más líneas:

```
/*comentario
de una o
más  líneas.
*/
```

Variables y concatenación

Para declarar una variable y poder guardar datos en memoria, debemos utilizar la palabra reservada "var" y luego entregar un nombre a la variable. Por último debemos asignar un dato a la variable creada por medio del símbolo igual (=), el dato asignado puede ser de cualquier tipo (no es necesario especificar el tipo de dato como en otros lenguajes).

Ejemplos:

```
var edad = 21;

var precio = 2.99;

var pagado = true;

var nombre = "Pedro";
```

Si queremos mostrar los datos, podemos hacerlo principalmente de dos formas:

- Por medio de una ventana en el navegador:

```
alert(nombre);
```

- Por medio de una consola:

```
console.log(nombre);
```

Concatenación

Recordemos que podemos concatenar datos si es necesario.

```
console.log("Mi nombre es " + nombre + " y mi edad es " + edad);
```

Operadores aritméticos

En JavaScript al igual que en otros lenguajes, tenemos símbolos que nos permiten realizar operaciones aritméticas.

- Suma: +
- Resta: -
- Multiplicación: *
- División: /
- Módulo: %

Ejemplos:

```
var edad = 21;
var dobleEdad = edad * 2;
console.log(dobleEdad); // resultado = 42

var edad = 24;
var mitadEdad = edad / 2;
console.log(mitadEdad); // resultado = 12
```

Incremento y decremento

Existen dos operadores los cuales solamente son válidos para las variables numéricas y se utilizan para incrementar o decrementar en una unidad el valor de una variable.

El operador de incremento se indica mediante el prefijo "++" y el operador de decremento se indica mediante el prefijo "--".

Ejemplos:

```
var numero = 5;
++numero;
console.log(numero);  // resultado = 6
```

```
var numero = 5;
--numero;
console.log(numero);  // resultado = 4
```

Operadores lógicos

Operador AND

La operación lógica AND se aplica al combinar dos o más valores booleanos. El operador se indica mediante el símbolo "&&" y su resultado solamente es true si **todos** los operandos son true.

Ejemplos:

```
var valor1 = true;
var valor2 = false;
resultado = valor1 && valor2; // resultado = false
```

```
var valor1 = true;
var valor2 = true;
var valor3 = true;
resultado = valor1 && valor2 && valor3; // resultado = true
```

Operador OR

La operación lógica OR se aplica al combinar dos o más valores booleanos. El operador se indica mediante el símbolo "||" y su resultado solamente es true si **alguno** de los operandos es true.

```
var valor1 = true;
var valor2 = false;
resultado = valor1 || valor2; // resultado = true
```

```
var valor1 = false;
var valor2 = false;
resultado = valor1 || valor2; // resultado = false
```

Operadores de relación

Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas:

Operador	Descripción
==	Es igual
!=	Es distinto
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual

Estos operadores generalmente son útiles en las condiciones, las cuales se describen más adelante.

Operador de negación

La negación lógica se utiliza para obtener el valor contrario al valor de una variable de tipo booleana y se obtiene prefijando el símbolo "!".

Ejemplo:

```
var visible = true;
alert(!visible);    // resultado = false
```

El funcionamiento de este operador se resume en la siguiente tabla:

variable	!variable
true	false
false	true

Condiciones

Permiten controlar qué procesos tienen lugar en función del valor de una o varias variables, alterando el flujo de un programa.

If

En JS como en muchos lenguajes de programación se utiliza la palabra reservada "if", seguida de una condición encerrada por medio de paréntesis la cual puede manejar operadores lógicos, el código a ejecutar si se cumple la condición se especifica entre llaves. También existen las palabras reservadas "else if" para agregar más condiciones si no se cumplen las anteriores y "else" la cual se ejecuta cuando no se cumple ninguna condición.

Ejemplos:

```
var edad = 18;
if (edad >= 18) {
    alert("Eres mayor de edad");
}
else {
    alert("Eres menor de edad");
```

```
}
// resultado = Eres mayor de edad

edad = 5;
if (edad >= 0 && edad < 12) {
      alert("Eres un niño");
} else if (edad >= 12 && edad < 18) {
      alert("Eres un  adolescente");
} else {
      alert("Eres mayor de edad");
}
// resultado = Eres un niño
```

Switch

Si sólo queremos decidir entre unos valores y ejecutar código dependiendo de los posibles valores de una variable o devueltos por una función, podemos usar un condicional Switch.

Ejemplo:

```
var mes = 2;
switch (mes) {
      case 1:
            alert("El mes es enero");
           break;
      case 2:
            alert("El mes es febrero");
            break;
      default:
            alert("El mes no es enero ni febrero);
            break;
      }
// resultado = El mes es febrero
```

Arreglos y sus métodos básicos

Los arreglos nos permiten agrupar un conjunto de datos de una manera ordenada, accediendo a cada uno según su posición.

Podemos declararlos de dos formas.

- Forma 1 (Aquí podemos especificar un entero con la cantidad de elementos del arreglo):

```
//Arreglo de 4 elementos
var cantElementos = 4;
var arreglo = new Array(cantElementos);
```

Podríamos llenarlo de la siguiente manera:

```
arreglo[0] = "dato1";
arreglo[1] = "dato2";
arreglo[2] = "dato3";
arreglo[3] = "dato4";
```

- Forma 2 (Especificando inmediatamente el contenido del arreglo):

```
var arreglo = [1,2,3];
```

En JS los arreglos no tienen tipo, por lo tanto pueden almacenar todo tipo de valores.

```
var arreglo = [1,"hola",3.3];
```

Podemos acceder a un valor del arreglo por medio de su posición.

Para el arreglo anterior:

```
console.log(arreglo[0]); // resultado = 1
console.log(arreglo[1]); // resultado = hola
console.log(arreglo[2]); // resultado = 3.3
```

En JS los arreglos son objetos, por lo tanto podemos aplicar ciertos métodos sobre ellos. Algunos de ellos son:

Método	Descripción
length()	Devuelve la cantidad de elementos que hay en el arreglo.
push()	Permite añadir un elemento al final del arreglo el cual recibe por parámetro.
pop()	Permite eliminar el elemento que se encuentra al final del arreglo.
unshift()	Permite añadir un elemento al principio del arreglo el cual recibe por parámetro.
shift()	Permite eliminar el elemento que se encuentra al principio del arreglo.

Ciclos

Ciclo FOR

Usado cuando se conoce cuantas veces debe iterar el ciclo.

Ejemplo:

```
for (var i = 10; i > 0; i--) {
  console.log("El número es " + i);
};

/* Resultado
```

```
El número es 10
El número es 9
El número es 8
El número es 7
El número es 6
El número es 5
El número es 4
El número es 3
El número es 2
El número es 1
*/
```

Ciclo WHILE

Usado cuando se desea que un ciclo se ejecute mientras que una condición especificada sea cierta.

Ejemplo:

```
var n = 0;
while (n < 3) {
     console.log("El número es " + n++);
  n++;
}
/* Resultado
El número es 0
El número es 1
El número es 2
*/
```

Ciclo DO...WHILE

Usado cuando deseamos que un bloque de código se ejecute al menos una vez, y entonces se repetirá siempre y cuando la condición especificada sea cierta.

Ejemplo:

```
var n = 2;
do {
    console.log("El número es" + n++);
} while (n < 5);
/* Resultado
El número es 2
El número es 3
El número es 4
*/
```

Funciones

Las funciones en JS se definen mediante la palabra reservada "function", seguida del nombre de la función. El código que ejecuta la función se especifica entre llaves "{}". Su definición formal es la siguiente:

```
function nombreFuncion() {
   ...
}
```

El nombre de la función se utiliza para llamar a esa función cuando sea necesario.

```
nombreFuncion();
```

Las funciónes pueden poseer argumentos o parámetros de cualquier tipo, para proporcionarlos se debe indicar cuántos argumentos necesita y cuál es el nombre de cada argumento. Los argumentos se indican dentro de los paréntesis de la función y se separan con una coma (,).

```
function sumaMuestra(primerNumero, segundoNumero){
...
}
```

Además, al invocar la función, se deben incluir los valores que se le van a pasar a la función.

```
sumaMuestra(2, 4);
```

Ejemplo:

Dentro de las llaves de la función se realiza el proceso de suma y muestra de datos.

```
function sumaMuestra(primerNumero, segundoNumero){
    var resultado = primerNumero + segundoNumero;
    console.log("El resultado es " + resultado);
}

sumaMuestra(2,4); // Resultado = 6
```

Al igual que en otros lenguajes, las funciones no solamente pueden recibir variables y datos, sino que también pueden devolver los valores, para realizar esto se utiliza la palabra reservada "return".

Algunos aspectos importantes:

- Las funciones pueden devolver valores de cualquier tipo.
- Las funciones solamente pueden devolver un valor cada vez que se ejecutan.

- Si la función llega a una instrucción de tipo return, se devuelve el valor indicado y finaliza la ejecución.

Ejemplo:

```
function sumaMuestra(primerNumero, segundoNumero){
    var resultado = primerNumero + segundoNumero;
    return resultado;
}

console.log(sumaMuestra(2,4)); // Resultado = 6
```

Como característica de las funciones en JS, podemos destacar que en el caso de no pasar parámetros a una función que estamos llamando y que requiere de estos, no se produce un error.

Objetos por medio de funciones

En JS las funciones son objetos y se tratan como tal.

Vamos a crear una clase "Persona" con sus atributos y métodos. La

misma función cumple con ser un constructor.

```
function Persona(nombre, edad) {
    // Atributos
    // Para declararlos usamos "this"
    this.nombre = nombre;
    this.edad = edad;
    // Métodos
    // Para declararlos usamos "this"
    this.saludar = function(){ console.log("Hola
        soy " + this.nombre);
    }
}
```

En base a la clase creada, podemos crear objetos de ella.

```
var pedro = new Persona("Pedro",19);

pedro.saludar(); // resultado = hola soy Pedro var

rodrigo = new Persona("Rodrigo",24);

rodrigo.saludar(); // resultado = Hola soy  Rodrigo
```

Como se puede observar, la palabra "this" en JS tiene un comportamiento diferente al de otros lenguajes, pero por lo general, su valor hace referencia al propietario de la

función que la está invocando o en su defecto, al objeto donde dicha función es un método. En nuestro caso, la referencia pertenece a la función "Persona".

JSON

Otra forma de declarar objetos en JS, es utilizando JSON (JavaScript Object Notation), el cual se define como un formato de texto ligero para el intercambio de datos. Leer y escribir un JSON es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Hoy en día este formato es ampliamente usado para intercambiar datos incluso entre diferentes lenguajes de programación en arquitecturas cliente-servidor.

Se declaran utilizando llaves "{}" y deben ser almacenados en una variable. La estructura de un JSON es del tipo "clave: valor" y cada conjunto "clave: valor" es separado por medio de una coma (,). Los valores pueden ser de cualquier tipo.

Ejemplo objeto "Persona":

```
var persona = {
    nombre: "Pedro",
    edad: 19,
    sexo: "M"
}
```

Nótese que para el último conjunto "clave: valor" no es necesario antecederlo con una coma.

Para acceder a un elemento podemos utilizar el nombre de la clave en forma similar a como accedemos a un arreglo.

```
console.log(persona["nombre"]); // resultado = Pedro

console.log(persona["edad"]); // resultado = 19
```

También podemos acceder a un elemento utilizando el nombre del objeto, seguido de un punto y la clave.

```
console.log(persona.nombre); // resultado = Pedro

console.log(persona.edad); // resultado = 19
```

A un objeto JSON también le podemos pasar funciones.

```
var persona = {
    nombre: "Pedro",
    edad: 19,
    sexo: "M",
    saludar: function(){
        console.log("Hola soy " + this.nombre);
    }
}

persona.saludar(); // resultado = Hola soy Pedro
```

EN el caso de un JSON la palabra reservada "this" nos permite acceder a los demás datos (atributos y métodos) del mismo JSON siempre que los utilicemos dentro de un contexto, en este caso el contexto sería la función declarada, si no es así, se van a interpretar como referencia externa por fuera del objeto, en el contexto global.

Es importante mencionar que dentro de un objeto JSON, el valor de una clave puede ser otro objeto JSON o bien un arreglo de datos.

Manipular el DOM

Una de las características más usadas de JS, es manipular el árbol DOM (Document Object Model), lo cual se refiere a la estructura lógica de un documento HTML, incluyendo todos sus elementos o etiquetas (document, head, body, h1, p, etc.). Con esto podemos seleccionar elementos de un HTML y almacenarlos en memoria con JS, con el fin de aplicar diferentes funciones sobre ellos obteniendo páginas web más dinámicas.

Obtener elementos por su id

Si tenemos en nuestro HTML un div con un atributo id con valor "caja", como se muestra a continuación:

```
<div id="caja"></div>
```

Podemos obtenerlo en JS utilizando el método del objeto document "getElementById()", el cual recibe como parámetro el valor del id del div a obtener.

```
var div = document.getElementById("caja");
```

Con esto, en nuestra variable "div" vamos a contener el objeto div que obtuvimos del HTML. Si imprimimos la variable por consola veríamos lo siguiente:

(...)

Como vemos en la imagen, se imprime un objeto "div" que posee un id especifico, además obtenemos una serie de datos que se pueden entender como los atributos del elemento, tales como el contenido que posee o sus propiedades de estilos.

Obtener elementos por su clase

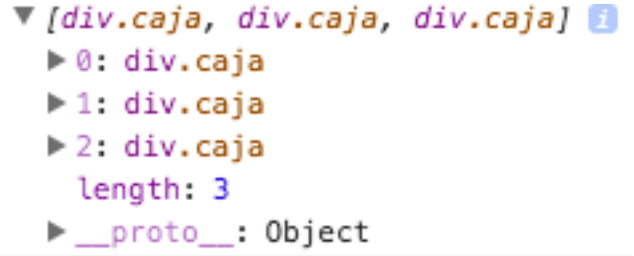
Si tenemos en nuestro HTML varios div con el mismo atributo class con valor "caja", como se muestra a continuación:

```
<div class="caja"></div>
<div class="caja"></div>
<div class="caja"></div>
```

Podemos obtenerlos en JS utilizando el método del objeto document "getElementsByClassName()", el cual recibe como parámetro el valor de la clase del div a obtener.

```
var clase = document.getElementsByClassName("caja");
```

Recordemos que en los atributos de los elementos HTML, el atributo "class" a diferencia del atributo "id" puede ser aplicado a uno o más elementos del HTML. Por lo tanto si ahora imprimimos por consola el valor que posee la variable "clase" vamos a obtener un arreglo de objetos de todos los elementos que poseen la clase "caja".



Como se ve en la imagen, se imprime un arreglo que contiene todos los elementos encontrados en el HTML con la clase indicada, estos elementos se representan como objetos que a su vez poseen atributos tales como se observó en el ejemplo anterior.

Obtener elementos por su tag

Para obtener elementos según el nombre del propio elemento ("div", "p", "span", etc), utilizamos el método del objeto document "getElementsByTagName()", el cual recibe como parámetro el nombre del elemento a obtener.

```
var elementos = document.getElementsByTagName("div");
```

Por el hecho de que puede existir uno o más elementos iguales, la variable "elementos" va a contener un arreglo con cada uno de los objetos encontrados, al igual que sucede si obtenemos un elemento por su clase.

Modificar texto de un elemento

Si deseamos modificar el texto de un elemento HTML por medio de JS, debemos usar la propiedad "innerHTML", la cual se aplica a un objeto que obtengamos del HTML.

Ejemplo:

```
document.getElementById("demo").innerHTML = "He cambiado el texto!";
```

Añadir y quitar estilos

Desde JS podemos añadir o cambiar estilos CSS de cualquier elemento HTML que podamos obtener.

Una forma de hacerlo es acceder directamente a los atributos específicos de un objeto.

```
document.getElementById("id").style.fontSize = "24px"
```

Otra manera es añadir o quitar clases que tengamos directamente en CSS.

```
var div = document.getElementById("caja");
// Añadir una  clase
div.classList.add("clasecss");
// Quitar una  clase
div.classList.remove("clasecss");
```

Eventos

Los eventos nos sirven para enlazar código de JS por medio de una acción que puede suceder en un HTML.

Por ejemplo, vamos a usar el evento click aplicado a un elemento button en HTML con id "btn", lo cual va a producir que se muestre un alert con un mensaje en el navegador.

Código html:

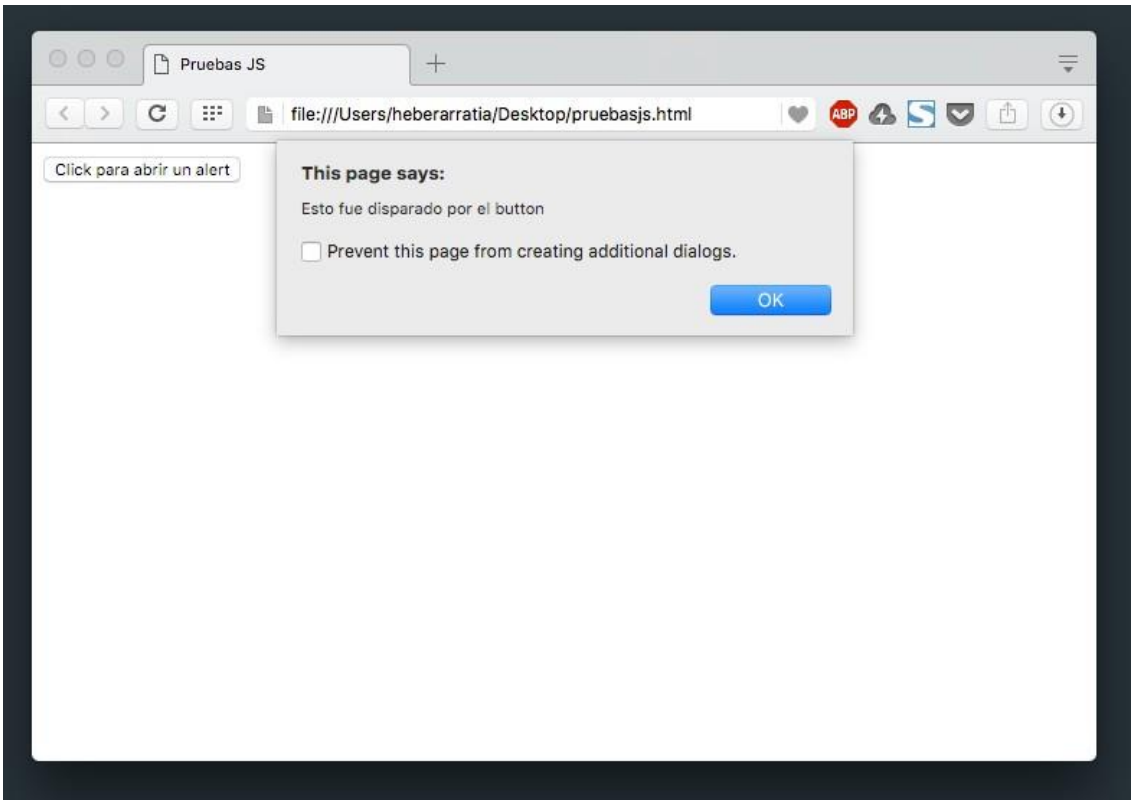
```
<button id="btn">Click para abrir un alert</button>
```

Código JS:

Primero obtenemos el elemento y luego le aplicamos la función "addEventListener()", la cual recibe dos parámetros, el primero es el evento que va a suceder y el segundo es la función que se va a ejecutar cuando el evento suceda.

```
document.getElementById("btn").addEventListener("click", function(){
    alert("Esto fue disparado por el button");
});
```

Entonces, al hacer click en el botón que muestra el HTML se podrá ver el alert disparado por el evento en JS.



Existen muchos tipos de eventos, como pueden ser "dblclick", "keydown", "keyup" o "load", y cada uno de ellos cumple con un proceso diferente. Cada evento se puede encontrar documentado en internet.

UNIVERSIDAD DE LA FRONTERA
Facultad de Ingeniería y Ciencias
Depto. Ciencias de Computación e Informática

Ejemplos prácticos

Ejemplo 1:

En nuestro primer ejemplo vamos a cambiar el color y el texto de un div en HTML.

Estructura de archivos:

```
ejemplo1
|-- ejemplo1.html
|-- estilo.css
|-- codigo.js
```

Archivo ejemplo1.html

```
<!DOCTYPE  html>
<html  lang="eS">
<head>
      <title>Ejemplo 1 JS</title>
      <meta  charset="UTF-8">
      <!-- Enlazamos los estilos CSS-->
      <link rel="stylesheet" href="estilo.css">
</head>
<body>
      <!-- Creamos un div con id "caja"  -->
      <div id="caja">
            <!-- Dentro del id creamos un span
            con id "text" y le añadimos un texto -->
            <span  id="texto">Hello!!</span>
      </div>
      <!-- Enlazamos el código JS -->
      <script src="codigo.js"></script>
</body>
</html>
```

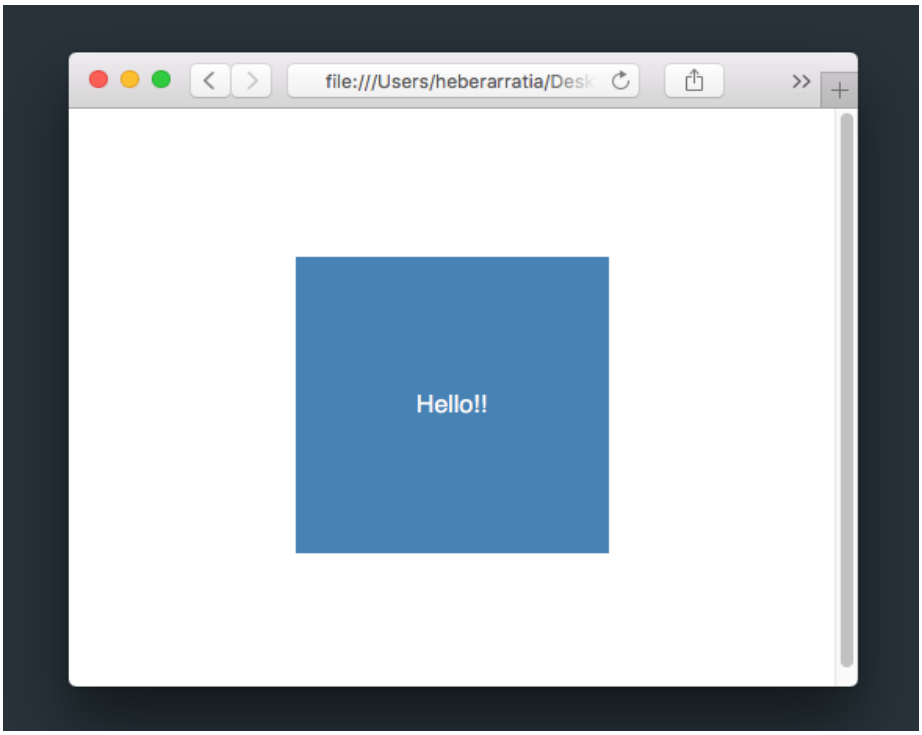
Archivo estilo.css

```css
body {
	background: #fff;
}

/*  Estilos  del  DIV  */
div {
	display:  table;
	width: 200px;
	height: 200px;
	background: steelblue;
	margin: 100px auto;
	cursor: pointer;
}

/* Estilos del span */
span {
	color: #fff;
	text-align: center;
	display: table-cell;
	vertical-align: middle;
	font-family: Helvetica;
}
```

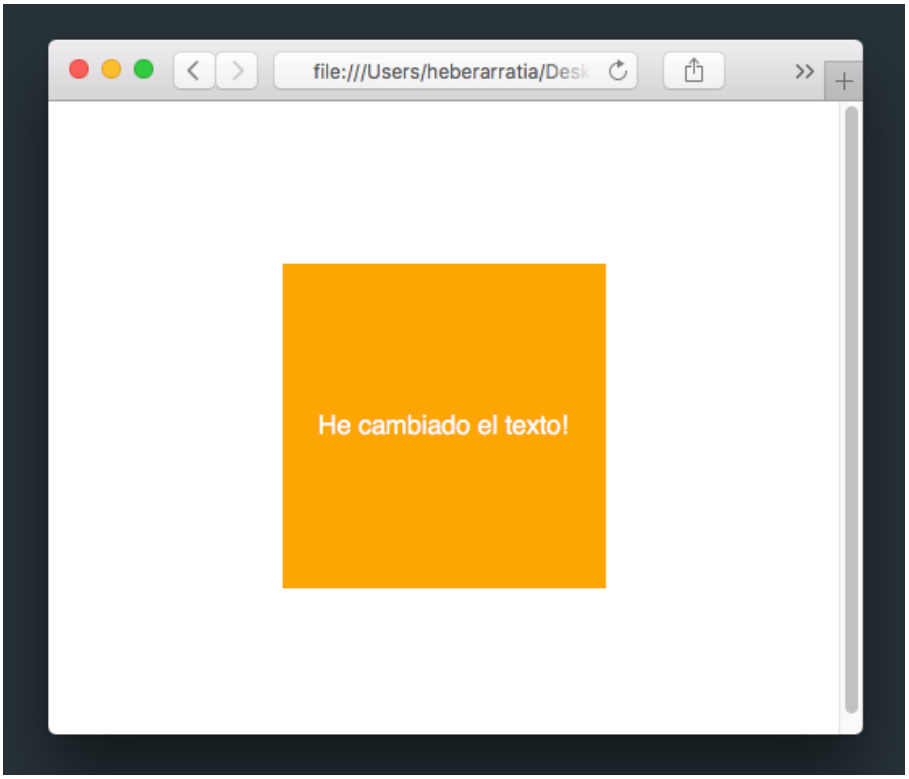
Archivo codigo.js

```javascript
// Obtenemos el DIV y lo almacenamos en una variable llamada "caja"
var caja = document.getElementById("caja");
// Obtenemos el span y lo almacenamos en una variable llamada "texto"
var texto =  document.getElementById("texto");

// Añadimos  el evento click a  la variable "caja"
caja.addEventListener("click",function(){
	// Cambiamos el texto del span que almacenamos en la variable "texto"
	texto.innerHTML = "He cambiado el texto!";
	// Cambiamos  el color del DIV  que  almacenamos  en  la variable "caja"
	caja.style.background = "orange";
});
```

Como resultado, al hacer click en el div.



Debería cambiar su color y su texto.



Ejemplo 2:

En este segundo ejemplo vamos a crear una pequeña aplicación que permita sumar dos números.

Estructura de archivos:

```
ejemplo2
|-- ejemplo2.html
|-- estilo.css
|-- codigo.js
```

Archivo ejemplo2.html

```
<!DOCTYPE html>
<html lang="eS">
<head>
    <title>Ejemplo 1 JS</title>
    <meta charset="UTF-8">
    <!-- Enlazamos los estilos CSS-->
    <link rel="stylesheet" href="estilo.css">
</head>
<body>
    <!-- Creamos un DIV para contener el formulario -->
    <div id="form">
        <span>Ingrese números a sumar:</span>
        <!-- Creamos dos input para recuperar los datos
        ingresados por el usuario -->
        <input id="input-uno" type="text">
        <input id="input-dos" type="text">
        <!-- Creamos un botón con el fin de invocar la suma -->
        <button id="btn">sumar</button>
    </div>
    <!-- Creamos in DIV que permita mostrar el resultado de la suma -->
    <div id="resultado"></div>
    <!-- Enlazamos el código JS -->
    <script src="codigo.js"></script>
</body>
</html>
```

Archivo estilo.css

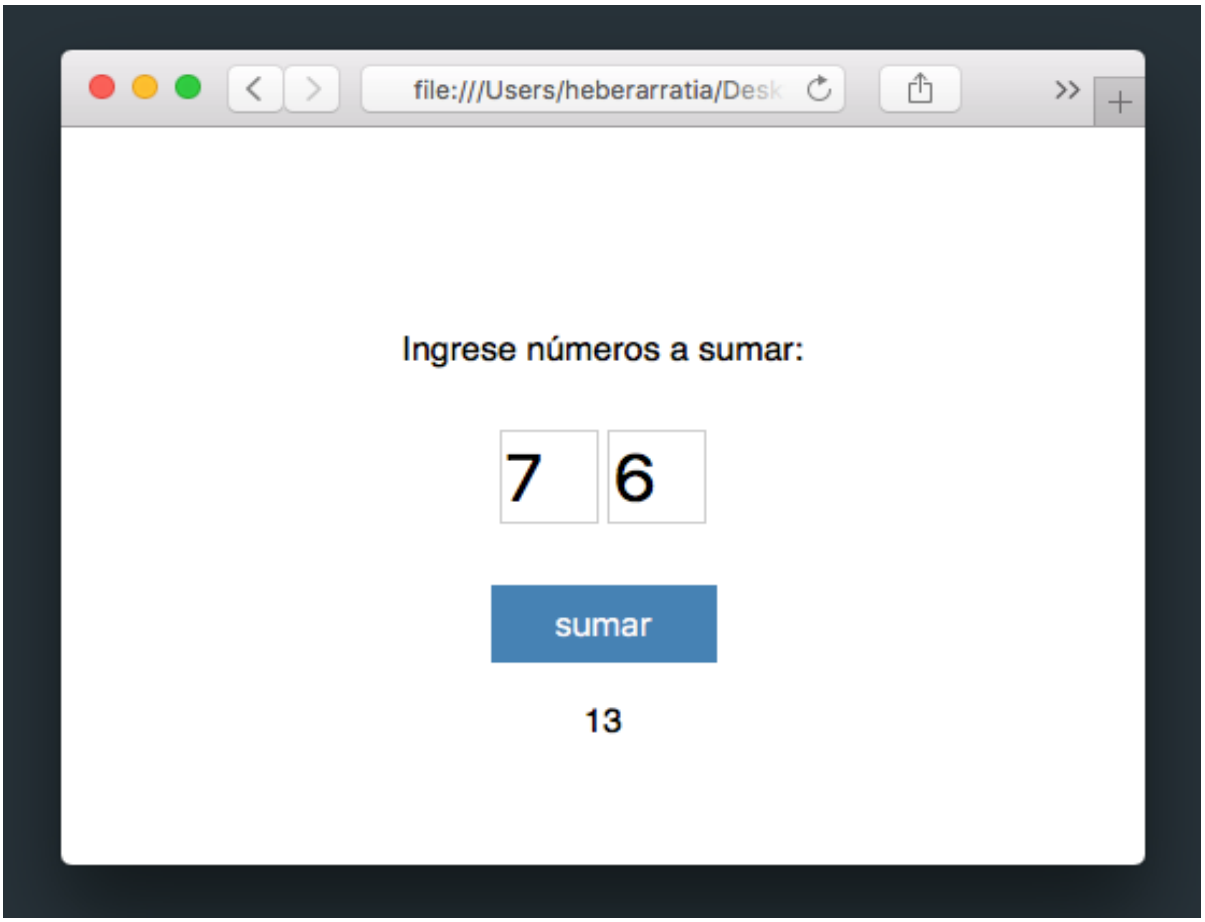
```
body {
	background: #fff;
	font-family: Helvetica;
}

#form{
	margin-top: 100px;
	text-align: center;
}

span{
	display: block;
	margin-bottom:  30px;
}

input{
	width: 40px;
	height: 40px;
	margin-bottom: 30px;
	font-size:  2em;
}

button{
	display: block;
	margin: 0 auto;
	padding: 10px 30px 10px 30px;
	font-size:  1em;
	border-style: none;
	background: steelblue;
	color:  #fff;
	cursor: pointer;
}

#resultado{
	margin:  20px;
	text-align: center;
}
```

UNIVERSIDAD DE LA FRONTERA
Facultad de Ingeniería y Ciencias
Depto. Ciencias de Computación e Informática

Archivo codigo.js

```
// Obtenemos el button y lo almacenamos en una variable llamada "btn"
var btn = document.getElementById("btn");
/* Obtenemos el div que muestra el resultado y lo
almacenamos en una variable llamada "resultado" */
var resultado = document.getElementById("resultado")
/* Obtenemos los dos input y los almacenamos en
variables "inputUno" y "inputDos" */
var inputUno = document.getElementById("input-uno");
var inputDos = document.getElementById("input-dos");

// Añadimos el evento click a la variable "btn"
btn.addEventListener("click",function(){
    /* Obtenemos el valor de cada input accediendo a
    su atributo "value" */
    var n1 = inputUno.value;
    var n2 = inputDos.value;
    /* Llamamos a una función que permite realizar la
    suma de los números y los mostramos al usuario" */
    resultado.innerHTML = suma(n1,n2);
});

/* Función que retorna la suma de dos números */
function suma(n1, n2){
    // Es necesario aplicar parseInt a cada número
    return parseInt(n1) + parseInt(n2);
}
```

Como resultado, deberíamos poder realizar la suma de dos números.



Actividades:

- Permitir que el usuario ingrese dos números que pertenezcan a un rango (min y max) en un formulario de HTML, luego al pulsar botón "Enviar", se debe mostrar en el mismo HTML un número random que pertenezca al rango ingresado.

- Mejorar calculadora del ejemplo 2, permitiendo que el usuario pueda sumar, restar, multiplicar y dividir dos números. Se debe validar que se ingresen ambos números, que sean enteros y la división por 0, en caso de no cumplir con estas validaciones, mostrar mensaje de error al usuario con texto en color rojo.