

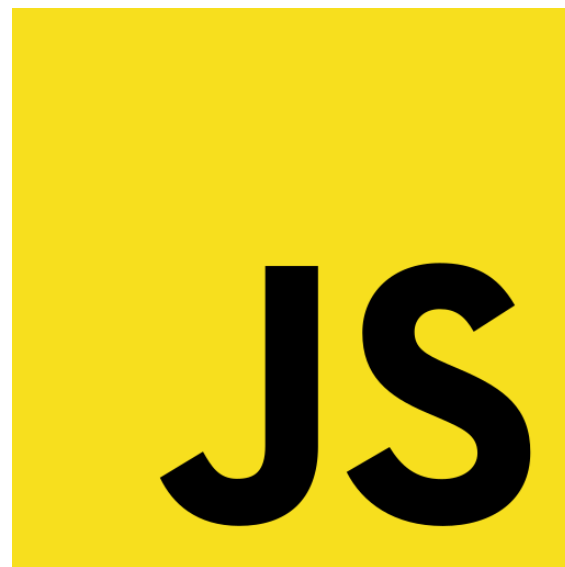
Fundamentos de la Web

Bloque II: Tecnologías de cliente web

Tema 7.1 – Introducción a JavaScript

JavaScript

- **Introducción**
- El lenguaje JavaScript
- HTML interactivo



- **Introducción**

- JavaScript
- Librerías JavaScript
- Frameworks de alto nivel
- JavaScript fuera del navegador
- Referencias

- El lenguaje JavaScript

- HTML Interactivo



- **Introducción**
 - JavaScript
 - Librerías JavaScript
 - Frameworks de alto nivel
 - JavaScript fuera del navegador
 - Referencias
- El lenguaje JavaScript
- HTML Interactivo



- Las páginas web pueden incorporar interactividad con el lenguaje **JavaScript**
- Con JavaScript se puede modificar la página ejecutando código JS (a través del modelo de objetos del documento **DOM**)
- También se pueden hacer peticiones al servidor web en segundo plano y actualizar el contenido de la web con los resultados (**AJAX**)

- Es un lenguaje de programación basado en el estándar **ECMAScript** de **ECMA** (otra organización diferente al **W3C**)
- Hay ligeras **diferencias** en la implementación de JS de los navegadores, aunque actualmente todos son bastante **compatibles** entre sí (**en el pasado no fue así**)

<http://www.ecma-international.org/>

- **Versiones de JS**

- JavaScript es un lenguaje que mejora cada año
- Al principio las versiones se numeraban con un número: ECMAScript 5 (**ES5**), ECMAScript 6 (**ES6**)
- Pero cuando se publicó ES6 decidieron llamarlo con el año: **ES2015**
- Se publica una actualización cada año

- **Navegadores web y versiones de JS**

- Los navegadores web tardan un tiempo en soportar las últimas versiones de JS
- En esta web se indica qué características soporta cada versión de cada navegador

<http://kangax.github.io/compat-table/es2016plus/>

- En clase veremos características soportadas por los navegadores más usados (Chrome, Firefox, Edge...)

- JavaScript no es Java

- Aunque algunos elementos de la sintaxis recuerden a **Java**, son lenguajes **completamente diferentes**
- El **nombre JavaScript** se eligió al publicar el lenguaje en una época en la que **Java estaba en auge** y fue principalmente por marketing (inicialmente se llamó **LiveScript**)



JavaScript

- **Características principales de JavaScript**
 - **Scripting:** No necesita compilador. Se ejecuta directamente desde el código fuente usando un **Runtime** integrado en el navegador
 - **Tipado dinámico:** habitual en los lenguajes de script
 - **Funcional:** Las funciones son elementos de primer orden
 - **Orientado a objetos:** De forma más dinámica que Java

- **Introducción**

- JavaScript
- **Librerías JavaScript**
- Frameworks de alto nivel
- JavaScript fuera del navegador
- Referencias

- El lenguaje JavaScript

- HTML Interactivo

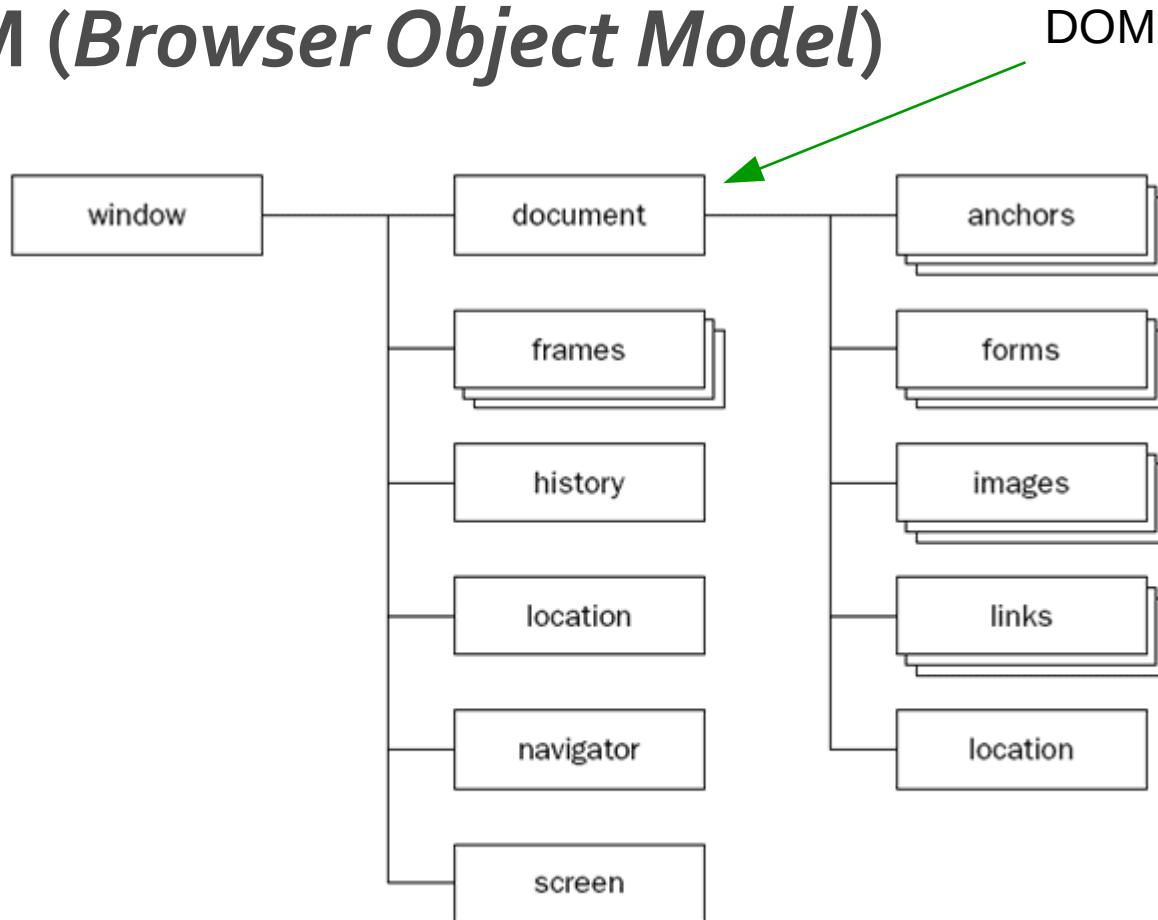


Librerías JavaScript

- **DOM (*Document Object Model*)**
 - Biblioteca usada para manipular el documento HTML cargado en el navegador
 - Permite la gestión de eventos, insertar y eliminar elementos, cambiar propiedades..
- **BOM (*Browser Object Model*)**
 - Biblioteca usada para usar otros elementos del browser: historial, peticiones de red AJAX, etc...
 - El BOM incluye al DOM como uno de sus elementos

Librerías JavaScript

- BOM (*Browser Object Model*)



Librerías JavaScript

- Existen multitud de **bibliotecas** (APIs) JavaScript para el desarrollo de aplicaciones



- Algunas de las más usadas son:

UNDERSCORE.JS

- **jQuery**: es un recubrimiento de la API DOM que aporta facilidad de uso, potencia y compatibilidad entre navegadores. Se usa para gestionar el interfaz de usuario (la página web) y para peticiones ajax.
- **underscore.js**: Librería para trabajar con estructuras de datos con un enfoque funcional. También permite gestionar plantillas (*templates*) para generar HTML partiendo de datos

- **Introducción**

- JavaScript
- Librerías JavaScript
- **Frameworks de alto nivel**
- JavaScript fuera del navegador
- Referencias

- El lenguaje JavaScript

- HTML Interactivo



Frameworks de alto nivel

- Además de bibliotecas, también existen **frameworks del alto nivel** que estructuran una aplicación de forma completa. Especialmente en **aplicaciones SPA**



- **Introducción**

- JavaScript
- Librerías JavaScript
- Frameworks de alto nivel
- **JavaScript fuera del navegador**
- Referencias

- El lenguaje JavaScript

- HTML Interactivo



JavaScript fuera del browser



- **Node.js** permite desarrollar aplicaciones con JS fuera del browser
 - Aplicaciones web
 - Aplicaciones por línea de comandos
 - Aplicaciones gráficas

<https://nodejs.org>

- **Introducción**

- JavaScript
- Librerías JavaScript
- Frameworks de alto nivel
- JavaScript fuera del navegador
- **Referencias**

- El lenguaje JavaScript

- HTML Interactivo



Referencias

- **Documentación**

- <https://developer.mozilla.org/es/docs/Web/JavaScript>
- <https://es.javascript.info/>

- **Tutoriales interactivos**

- <https://www.codecademy.com/>
- <https://www.pluralsight.com/courses/javascript-getting-started>

JavaScript

- Introducción
- **El lenguaje JavaScript**
 - Características del lenguaje
 - Integración con HTML
 - Sintaxis básica
 - Arrays
 - Sentencias de control de flujo
 - Funciones
 - Excepciones
 - Orientación a objetos
- HTML Interactivo



JavaScript

- Introducción
- **El lenguaje JavaScript**
 - **Características del lenguaje**
 - Integración con HTML
 - Sintaxis básica
 - Arrays
 - Sentencias de control de flujo
 - Funciones
 - Excepciones
 - Orientación a objetos
- HTML Interactivo



Características del lenguaje JS

- Vamos a estudiar las características **más usadas** actualmente para desarrollar aplicaciones
- Hay ciertas características que ya **no se aconsejan y no veremos** (aunque te las puedes encontrar si ves código antiguo)
- Como estáis aprendiendo **Java**, haremos una comparativa entre ambos

Características del lenguaje JS

- **Lenguaje de script (Java es compilado)**
 - No existe compilador
 - El navegador carga el código, lo analiza y lo ejecuta
 - El navegador indica tanto **errores de sintaxis** como errores de **ejecución**
- **Tipado dinámico (Java tiene tipado estático)**
 - Al declarar una variable no se indica su tipo
 - A lo largo de la ejecución del programa una misma variable puede tener **valores de diferentes tipos**

Características del lenguaje JS

- Imperativo y estructurado (como Java)
 - Se declaran **variables**
 - Se ejecutan las sentencias **en orden**
 - Dispone de **sentencias de control** de flujo de ejecución (if, while, for...)
 - La **sintaxis** imperativa/estructurada es muy parecida a Java y C

Características del lenguaje JS

- **Orientado a objetos**

- Todos los **valores son objetos** (no como en Java, que existen tipos primitivos)
- Existe **recolector de basura** para liberar la memoria de los objetos que no se utilizan (como en Java)
- La **orientación a objetos** puede basarse en clases o en prototipos (Java se basa en clases)
- En tiempo de ejecución se pueden **crear objetos**, cambiar el valor de los **atributos** e invocar a **métodos** (como en Java)
- En tiempo de ejecución se pueden **añadir y borrar atributos y métodos** (en Java no se puede)

Características del lenguaje JS

• Funciones

- Aunque sea orientado a objetos, también permite declarar **funciones independientes** (en Java todas las funciones son métodos de clases)
- Las funciones se pueden **declarar en cualquier sitio**, asignarse a variables y pasarse como parámetro
- Existen **funciones anónimas** (como las **expresiones lambda** de Java)
- En JavaScript se puede implementar código inspirado en el **paradigma funcional**

JavaScript

- Introducción
- **El lenguaje JavaScript**
 - Características del lenguaje
 - **Integración con HTML**
 - Sintaxis básica
 - Arrays
 - Sentencias de control de flujo
 - Funciones
 - Excepciones
 - Orientación a objetos básica
- HTML Interactivo



Integración con HTML

- El código **JavaScript** se incluye en etiquetas **<script>** en el **<body>** o **<head>**
- Se puede incluir en el fichero HTML o en ficheros .js
- Se puede añadir en varios lugares (pero no es necesario)

index.html

Ejemplo1

```
<html>
  <head>
    <script>console.log("Head1");</script>
    <script src="app1.js"></script>
  </head>
  <body>
    <script>console.log("Body1");</script>
    <script src="app2.js"></script>
  </body>
</html>
```

app1.js

```
console.log("Head2");
```

app2.js

```
console.log("Body2");
```

Integración con HTML

- En la asignatura escribiremos código JavaScript en un fichero independiente y lo insertaremos en el <head>

Ejemplo2

index.html

```
<html>
  <head>
    <script src="app.js"></script>
  </head>
  <body>
    ...
  </body>
</html>
```

app.js

```
console.log("Hola Mundo");
```

JavaScript

- Introducción
- **El lenguaje JavaScript**
 - Características del lenguaje
 - Integración con HTML
 - **Sintaxis básica**
 - Arrays
 - Sentencias de control de flujo
 - Funciones
 - Excepciones
 - Orientación a objetos
- HTML Interactivo



Sintaxis básica

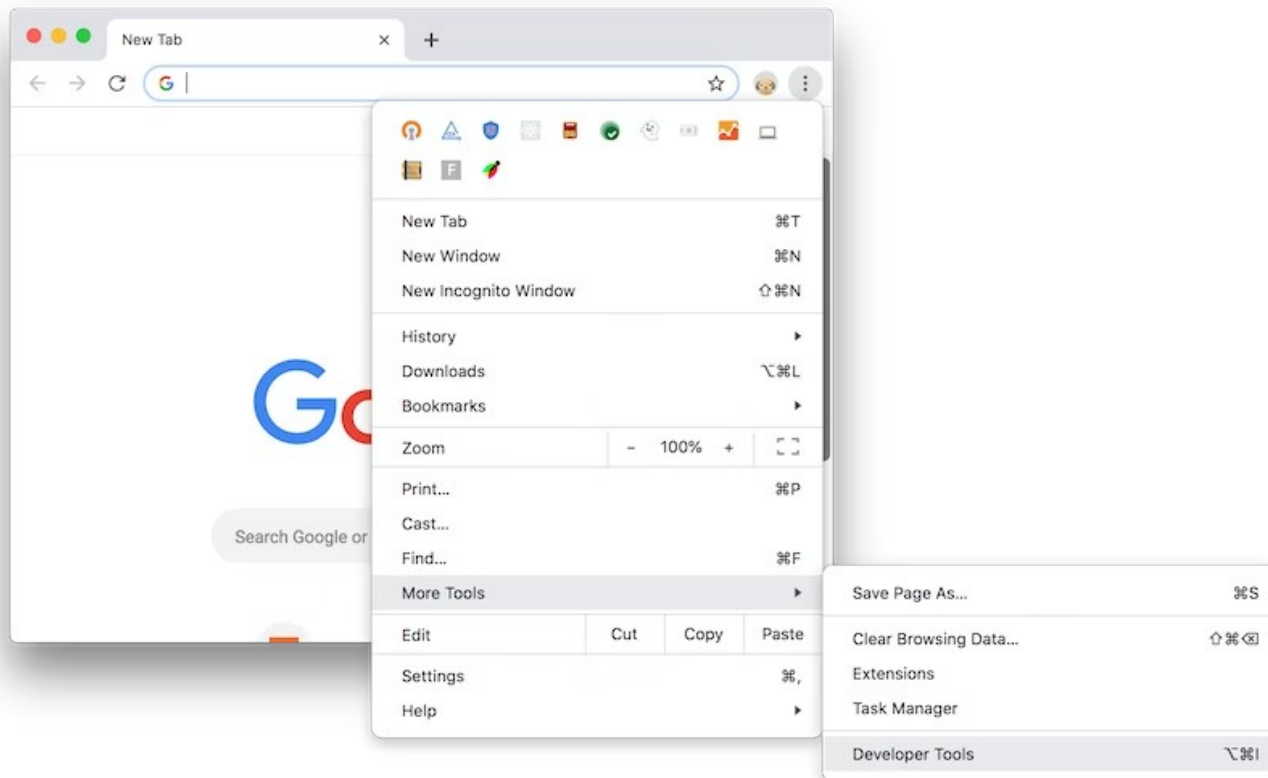
- **Mostrar información desde JavaScript**
 - Escribiendo en la consola JavaScript del navegador

```
console.log('Texto');
```

- Más adelante veremos cómo mostrar información en la página web

Sintaxis básica

- **Mostrar información desde JavaScript**
 - Mostrar la consola de Google Chrome



Ejercicio 1

- Crea una página web que muestre en la consola "Hola Mundo!" al cargarse en un navegador web

Sintaxis básica

- **Comentarios (como en Java)**
 - Una línea //
 - Multilínea /* */
- **Delimitadores (como en Java)**
 - De bloque { }
 - De sentencia ; (opcionales)
- **Palabras reservadas**

```
abstract boolean break byte case catch char class const continue debugger default
delete do double else enum export extends false final finally float for function goto
if implements import in instanceof int interface long native new null package private
protected public return short static super switch synchronized this throw throws
transient true try typeof var volatile void while with
```

Sintaxis básica

• Variables

- Es un lenguaje dinámico (con tipado **dinámico**)
- Las variables se tienen que declarar pero **no se indica el tipo**

```
//La variable edad tendrá un valor de 34
let edad = 34;
let encontrado = false;
```

- La variable está disponible en el bloque en el que se declara (como en Java)

Sintaxis básica

• Variables

- Una misma variable puede tener un valor numérico en un punto de ejecución y una cadena de caracteres tiempo después

```
let id;
...
id = 34;
...
id = "XDFS"
```

- **Variables**

- Si las variables **no se inicializan** tienen el valor **undefined** (en vez de cero o null como en Java)

```
let hola;  
console.log("Hola: " + hola);
```



```
Hola: undefined
```

- Si las variables **no se declaran** (por una equivocación)
 - ▮ Si se intenta **leer su valor** salta un **error** (que finaliza la ejecución)
 - ▮ Si se **asigna un valor**, se crea un nuevo atributo en el objeto global (**no dan error las siguientes lecturas**)

Sintaxis básica

- Tipos de datos básicos
 - Son objetos (no existe distinción como en Java entre objetos y tipos primitivos)
 - **Number:** Números enteros y reales de cualquier precisión
 - **Boolean:** true o false (como en Java)

Sintaxis básica

- Tipos de datos básicos

- String

- ▮ Cadenas de caracteres
- ▮ Comillas simples o dobles
- ▮ No pueden modificarse. Inmutables (como en Java)
- ▮ Concatenación con + (como en Java)
- ▮ Como no hay tipo character, se usa un string con un único character

Sintaxis básica

- Operadores en expresiones
 - Similares a Java
 - Aritméticos: + - * / % (la división es siempre real)
 - Comparación números: < > <= >=
 - Lógicos: && || !
 - Comparativo: ?: (Elvis operator)
 - Modificación: ++ --
 - Asignación: = += -= *= /= %=

Sintaxis básica

Ejemplo3B

- Operadores en expresiones
 - Comparación (diferente a Java)
 - ▮ Igual: === Distinto: !==
 - ▮ En strings se comporta como el equals(...) en Java
 - ▮ En arrays se comporta como == en Java

```
let uniName = "URJC";

let otherName = "UR" + "JC";

console.log(uniName === otherName) // true
```

- Introducción
- **El lenguaje JavaScript**
 - Características del lenguaje
 - Integración con HTML
 - Sintaxis básica
 - **Arrays**
 - Sentencias de control de flujo
 - Funciones
 - Excepciones
 - Orientación a objetos
- HTML Interactivo



Arrays

- Los arrays de JavaScript son parecidos a los de Java
 - El acceso para lectura o escritura es con []
 - Se inicializan con [] (en Java con { })
 - Tienen la propiedad **length**
 - Empiezan por **cero**

Ejemplo3C

```
let numbers = [3, 4, 4, 2];

console.log(numbers[0]); // 3

console.log(numbers.length) // 4
```

- Los arrays de JavaScript son parecidos a los de Java
 - La **asignación** de una variable con un array a otra variable, no copia. Ambas variables apuntan al **mismo objeto array**
 - El operador `===` compara si son el **mismo objeto** (no que tengan el contenido igual)

Ejemplo3D

```
let nums = [1, 2, 3, 4];  
let numsAux = nums;  
  
numsAux[0] = -1;  
  
console.log(nums[0] === -1) // true  
console.log(numsAux === nums) // true
```

- Los arrays de JavaScript son parecidos a los de Java
 - Los **arrays de varias dimensiones** son arrays de **arrays**. Hay que crear de forma explícita los niveles.
 - Se pueden recorrer de forma similar

Ejemplo3E

```
let numbers = [3, 4, 4, 2];  
  
console.log(numbers[0] + ',' +  
             numbers[1] + ',' +  
             numbers[2] + ',' +  
             numbers[3]);
```



3,4,4,2,

Arrays

- Pero se diferencian en algunos detalles

- Los arrays literales con `[]` en vez de `{ }` y sin `new`

```
let empty = [];  
let numbers = ['zero', 'one', 'two', 'three']
```

- Un array en JS puede tener en cada posición valores de diferentes tipos mezclados (tipado dinámico)

Arrays

- Errores de acceso en arrays
 - La lectura de un elemento fuera de los límites devuelve **undefined**
 - ▮ En Java sería un `ArrayIndexOutOfBoundsException`
 - La escritura de un elemento fuera de los límites del array se permite, haciendo más grande el array y rellenando con valores **undefined** los huecos

Arrays

Ejemplo3F

```
let numbers = [3, 4];  
console.log(numbers[2]);  
  
numbers[4] = 5;  
console.log(numbers[4]);  
  
console.log(numbers[0] + ',' +  
               numbers[1] + ',' +  
               numbers[2] + ',' +  
               numbers[3] + ',' +  
               numbers[4]);
```



```
undefined  
5  
3,4,undefined,undefined,5
```

Arrays

Ejemplo3G

- Errores de acceso en arrays
 - Si una variable tiene el valor **undefined** y se intenta acceder a un elemento como si tuviera un array, genera un error “**Uncaught TypeError: Cannot read properties of undefined (reading '5')**”

```
let numbers;

console.log(numbers[5]);
```

□ En Java sería un `NullPointerException`

Arrays

- **Modificación del array**
 - Se pueden establecer elementos en posiciones no existentes y el array crece dinámicamente.
 - El método **push** añade un elemento al final del array (como el método **add** del **ArrayList** en Java)
 - Modificando la propiedad **length** se puede cambiar el tamaño del array (si se amplía, se rellena con **undefined**) (en Java es de sólo lectura)

Arrays

- **Modificación del array**

- El **operador delete** borra un elemento dejando el hueco con valor **undefined**

```
delete numbers[2];
```

- Para borrar y no dejar el hueco se usa el método **splice** indicando el índice desde el que hay que borrar y el número de elementos que borrar

```
numbers.splice(2, 1);
```

- Introducción
- **El lenguaje JavaScript**
 - Características del lenguaje
 - Integración con HTML
 - Sintaxis básica
 - Arrays
 - **Sentencias de control de flujo**
 - Funciones
 - Excepciones
 - Orientación a objetos
- HTML Interactivo



Sentencias de control de flujo

- **Sentencia if**

- Sintaxis como en Java
- No es obligatorio que la expresión devuelva un boolean
- Se consideran como falsos:
 - `false`, `null`, `undefined`, `""` (cadena vacía), `0`, `NaN`

Sentencias de control de flujo

Ejemplo4A

```
let nota = 7;
...
if(nota < 5){
  console.log('Suspendido');
} else {
  console.log('Aprobado');
}
```

```
let nombre = '';

if(!nombre){
  nombre = 'Sin nombre';
}
```

Sentencias de control de flujo

Ejemplo4B

- Sentencias switch, while, do while, for
 - Sintaxis y semántica como en Java y C

```
let animal = 'Jirafa';

switch (animal) {
  case 'Vaca':
  case 'Jirafa':
  case 'Cerdo':
    console.log('Sube al arca');
    break;
  case 'Dinosaurio':
  default:
    console.log('No sube');
}
```

```
let array = [3,4,5,6,4,5]

for(let i=0; i<array.length; i++){
  console.log(array[i]);
}
```


- Introducción
- **El lenguaje JavaScript**
 - Características del lenguaje
 - Integración con HTML
 - Sintaxis básica
 - Arrays
 - Sentencias de control de flujo
 - **Funciones**
 - Excepciones
 - Orientación a objetos
- HTML Interactivo



Funciones

- **JavaScript** es un lenguaje **funcional** en el sentido de que las **funciones** son **ciudadanos de primera clase**
- Se pueden declarar con nombre

Ejemplo5A

```
function imprime(param){
    console.log(param);
}

imprime(4);
```

Funciones

- Se pueden declarar **sin nombre (anónimas)** y asignarse a una **variable**

Ejemplo5B

```
let imprime = function(param){
    console.log(param);
}

imprime(4);
```

- **Sentencia return**

- Cuando se ejecuta fuerza la terminación de una función (Como en Java)
- Puede tener un valor asociado que será devuelto por la función en la que se ejecute

Ejemplo5C

```
function myFunction(a, b) {  
    return a * b;  
}  
  
console.log(myFunction(4, 3));
```

Funciones

- Las funciones se pueden pasar como **parámetro** a otras funciones

```
function showItems(array, hasToShow) {
    for(let i=0; i<array.length; i++){
        if(hasToShow(array[i])){
            console.log(array[i]);
        }
    }
}

function isEven(number){
    return number % 2 == 0;
}

showItems([1,2,3,4,5,6], isEven);
```

Ejemplo5D



2
4
6

- Se pueden declarar funciones en el cuerpo de otras funciones

Ejemplo5E

```
function showNonZeros(numbers) {  
  
    let isNonZero = function(num) {  
        return num !== 0;  
    }  
  
    showItems(numbers, isNonZero);  
}
```

- Se pueden declarar funciones en la llamada a otras funciones

```
function showNonZeros(numbers) {  
  
    let isNonZero = function(num) {  
        return num != 0;  
    }  
  
    showItems(numbers, isNonZero);  
}
```

Ejemplo5F



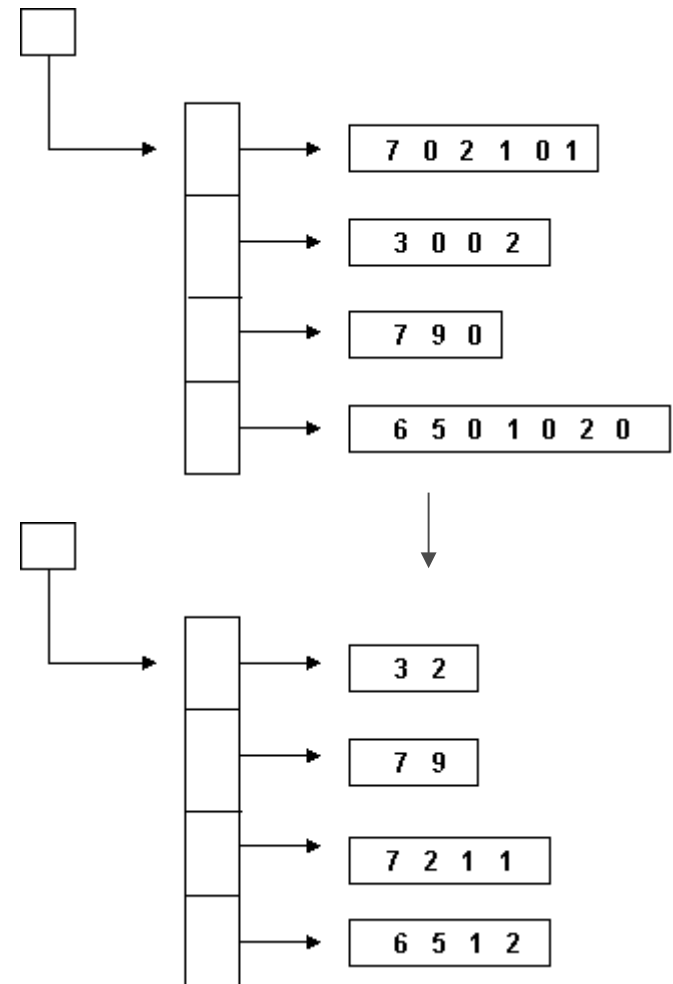
```
function showNonZeros(numbers) {  
  
    showItems(numbers, function(num) {  
        return num != 0;  
    });  
}
```

Ejercicio 2

- Crear una función que **reciba un array como parámetro** y devuelva un array de dos elementos
- El primer elemento apuntará a un **array con los números pares** del array que se pasa como parámetro
- El segundo elemento apuntará a un **array con los números impares** del array que se pasa como parámetro
- Para probar la función se implementarán varias llamadas con **diferentes arrays** y el resultado se mostrará en la **consola del navegador**

Ejercicio 3

- Crear una función que **reciba un array bidimensional, le quite los ceros y ordene las filas de menor a mayor longitud**
- Para probar la función se implementarán varias llamadas con **diferentes arrays** y el resultado se mostrará en la **consola del navegador**



Ejercicio 3

- Para ordenar los arrays por tamaño se puede usar el **algoritmo de la burbuja** (*bubble sort*)
- Ejemplo: Ordenación de un array de números (tiene que adaptarse para ordenar arrays por su tamaño)

```
function sortNumbers(numbers) {
  for (let i = 0; i < numbers.length; i++) {
    for (let j = 0; j < numbers.length-1; j++) {
      if (numbers[j] > numbers[j + 1]) {
        let temp = numbers[j];
        numbers[j] = numbers[j + 1];
        numbers[j + 1] = temp;
      }
    }
  }
  return numbers;
}
```

- **Parámetros de una función**
 - Si se pasan menos parámetros de los que están en la cabecera, los que faltan toman el valor **undefined (No hay error)**

Ejemplo5G

```
function imprime(param){  
    console.log(param); //undefined  
}  
  
imprime();
```

Funciones

- Como pasar **menos parámetros** no da error, se puede usar para implementar parámetros **opcionales**

```
function arrayToString(array, separador){
  if(!separador){
    separador = ",";
  }
  let result = "";
  for(let i=0; i<array.length; i++){
    result += array[i] + separador;
  }
  return result;
}

console.log(arrayToString([1,2,3,4]));
console.log(arrayToString([1,2,3,4],"-"));
```

Ejemplo5H

1,2,3,4,

1-2-3-4-

- **Parámetros de una función**
 - Si se pasan más parámetros de los que están en la cabecera, los que sobran se **ignoran (No hay error)**

Ejemplo51

```
function imprime(param){  
    console.log(param); //5  
}  
  
imprime(5, 7);
```

- Información accesible desde la función
 - Parámetros y variables declaradas en la función

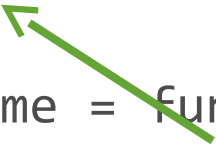
Ejemplo5J

```
let imprime = function (param){  
  let local = 0;  
  console.log("param:"+param+" local:"+local);  
}  
  
imprime(4); // param:4 local:0
```

- **Información accesible desde la función**
 - Variables accesibles en el punto en que se declara la función (ámbito léxico)

Ejemplo5K

```
let texto = "Hola";  
let imprime = function () {  
    console.log(texto);  
}  
  
imprime(); // Hola
```



- **Información accesible desde la función**
 - Cuando se referencia a una variable no sólo se accede a su valor, se accede a la **propia variable en sí**
 - Si se **cambia el valor** de la variable, la **función podrá leer** el nuevo valor
 - En Java las variables a las que accede una expresión lambda **no pueden cambiar de valor**

- **Información accesible desde la función**
 - El conjunto de variables a las que tiene acceso la función se llama **cerradura o cierre (*closure*)**

```
let texto = 'Hola'
function imprime(){
    console.log(texto)
}

imprime(); // Hola

texto = 'Adios'

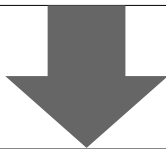
imprime(); // Adios
```

Ejemplo5L

Funciones flecha (arrow function)

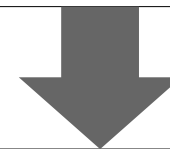
- Una forma más compacta de escribir funciones anónimas

```
let f = function (v) {  
  return v + 1;  
}
```



```
let f = v => v + 1;
```

```
var f = function (p1,p2) {  
  console.log('Hola');  
  console.log('Adios');  
  return 3;  
}
```



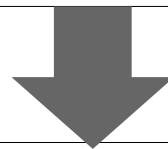
```
var f = (p1,p2) => {  
  console.log('Hola');  
  console.log('Adios');  
  return 3;  
}
```

Funciones flecha (arrow function)

- Muy usado cuando la función se declara en la lista de parámetros de otra función

Ejemplo5M

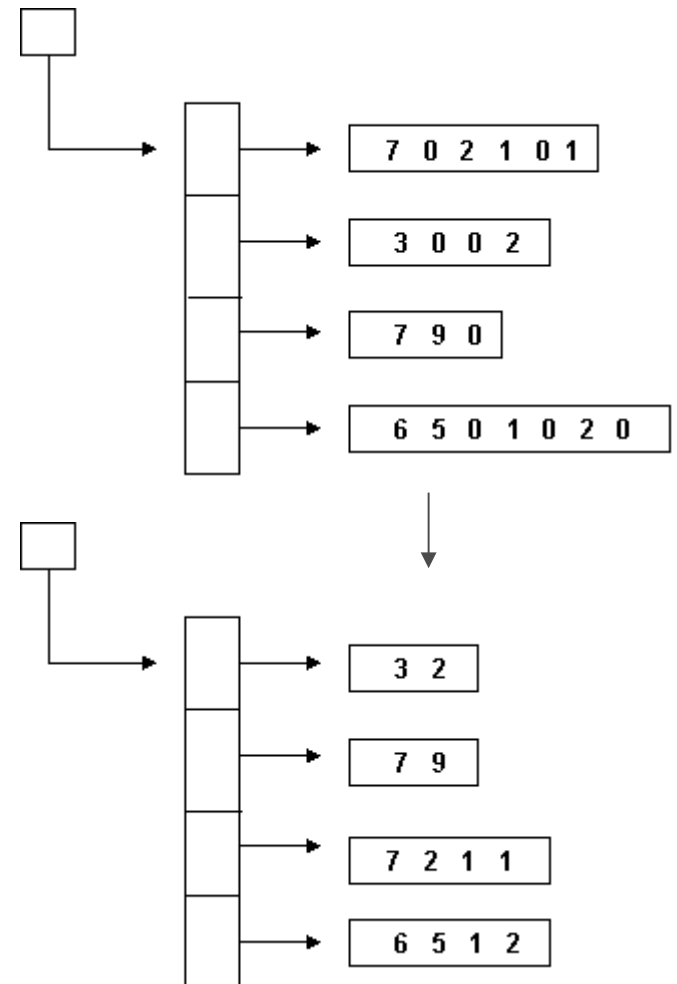
```
function showNonZeros(numbers) {
    showItems(numbers, function(num) {
        return num != 0;
    });
}
```



```
function showNonZeros(numbers) {
    showItems(numbers, num => num != 0);
}
```

Ejercicio 4

- Haz que el ejercicio 3 sea configurable
- En vez de eliminar los ceros de los arrays, que elimine los números que decida el usuario pasando una función como parámetro
- Usa arrow functions en las pruebas que llaman a `quitaNumerosYOrdena(...)`



JavaScript

- Introducción
- **El lenguaje JavaScript**
 - Características del lenguaje
 - Integración con HTML
 - Sintaxis básica
 - Arrays
 - Sentencias de control de flujo
 - Funciones
 - **Excepciones**
 - Orientación a objetos
- HTML Interactivo



Excepciones

- Funcionan igual que en **Java**
- Existe un bloque con **try catch finally**
- El operador **throw** eleva la excepción
- A diferencia de Java, se puede lanzar cualquier objeto como excepción, aunque lo recomendable es elevar un objeto con propiedades **name** y **message**

Excepciones

```
try {
    let error = ...;
    if (error) {
        throw "An error";
    }
    return true;
} catch (e) {

    alert (e);
    return false;

} finally {
    //do cleanup
}
```

JavaScript

- Introducción
- **El lenguaje JavaScript**
 - Características del lenguaje
 - Integración con HTML
 - Sintaxis básica
 - Arrays
 - Sentencias de control de flujo
 - Funciones
 - Excepciones
 - **Orientación a objetos**
- HTML Interactivo



Orientación a Objetos

- Hasta ahora hemos visto que **JavaScript** es un lenguaje bastante familiar en cuanto a **sintaxis** y **características**:
 - Imperativo y estructurado
 - Tipado dinámico
 - Tres tipos básico: Number, Boolean y String
 - Arrays dinámicos
 - Recolector de basura
 - Funciones

Orientación a Objetos

- La orientación a objetos en **JavaScript** a primera vista parece **similar** a la de **Java**, pero en esencia es **muy diferente**
- La gran **mayoría de los lenguajes** de programación implementan la Orientación a Objetos con **clases** (Java, C#, C++, Python, Ruby...)
- En JavaScript se añadió el soporte de **clases** en ES2015. Hasta ese momento se usaban los **prototipos** (que no veremos nosotros)

Orientación a Objetos

- Creación de objetos

- En cualquier momento se puede **crear un objeto**, **definir sus atributos** y asignarles **valor**

```
let empleado = {
  nombre: "Pepe",
  salario: 700
}
```

- **Atributos de un objeto**

Ejemplo6A

```
let empleado = {  
  nombre: "Pepe",  
  salario: 700  
}  
  
console.log("S:"+empleado.salario);  
  
empleado.salario = 800;  
  
empleado.telefono = "663232539";  
  
console.log("T:"+empleado.telefono);
```

Orientación a Objetos

- **Métodos en un objeto**

- Los objetos pueden tener **métodos**
- **Los métodos son en realidad** funciones asignadas a una propiedad

```
let empleado = {
  nombre: "Pepe",
  salario: 700,
  toString: function(){
    return "N:" + this.nombre + " S:" + this.salario;
  }
}
```

Para acceder a los atributos del objeto es necesario usar this (en Java es opcional)

Orientación a Objetos

- **Métodos en un objeto**

- Los métodos se **invocan** con la notación punto (como en Java)

```
//Devuelve 'Nombre:Pepe, Salario:700'
let texto = empleado.toString();
```

- Se pueden **añadir métodos** a un objeto en cualquier momento

```
empleado.getCategoria = function(){
    return this.salario > 800 ? 'Superior':'Normal';
}
```

- Métodos en un objeto

Ejemplo6B

```
let empleado = {
  nombre: 'Pepe',
  salario: 700,
  toString: function(){
    return 'N:'+this.nombre+' S:'+this.salario;
  }
}

//Muestra N:Pepe S:700
console.log(empleado.toString());

empleado.getCategoria = function(){
  return this.salario > 800 ? 'Superior':'Normal';
}

console.log('C:'+empleado.getCategoria());
```

Orientación a Objetos

- Como se puede ver, los **objetos son muy flexibles** porque **no necesitan** un **molde** (una clase)
- Esta forma de trabajar es muy útil cuando **sólo hay un objeto** con una estructura determinada (*singleton*)
- Se usa bastante en JavaScript para crear **objetos sin métodos** que guardan información (como los **registros** o **struts**)

Ejercicio 5

- Se quiere implementar un programa en JavaScript que calcule el área y perímetro de un rectángulo
- El rectángulo se representará como un objeto con los atributos:
 - color
 - alto
 - ancho
- Y métodos:
 - `area()`: $\text{largo} * \text{ancho}$
 - `perimetro()`: $2 * \text{ancho} + 2 * \text{largo}$

Orientación a Objetos

- Crear varios objetos con los mismos atributos y métodos

Ejemplo6C

```
function nuevoEmpleado(nombre, salario){

    var empleado = {
        nombre: nombre,
        salario: salario,
        toString: function(){
            return "N:"+this.nombre+" S:"+this.salario;
        }
    };

    return empleado;
}
```

Orientación a Objetos

- Crear varios objetos con los mismos atributos y métodos

Ejemplo6C

```
let empleado = nuevoEmpleado("Pepe",700);

//Devuelve 'Nombre:Pepe, Salario:700'
console.log(empleado.toString());

//Devuelve 700
console.log(empleado.salario);
```

Orientación a Objetos

- **Objetos, null y undefined**

- En JavaScript también se puede usar **null** como un valor válido para las variables
- Las variables que no están inicializadas tienen el valor **undefined**
- Usar una variable sólo si apunta a un objeto

```
let obj = null;
if(obj){
    obj.doSomething();
}
```

Devuelve true si
obj no es **null** ni es
undefined

Ejercicio 5b

- Se quiere implementar un programa en JavaScript que permita analizar un array de rectángulos
- De cada rectángulo se debe conocer:
 - Color
 - Alto
 - Ancho
 - Área: $\text{largo} * \text{ancho}$
 - Perímetro: $2 * \text{ancho} + 2 * \text{largo}$

Ejercicio 5b

- Los **análisis** serán:
 - Suma total de áreas de los rectángulos
 - Suma total de perímetros de los rectángulos
 - Área media
 - Perímetro medio
- Prueba del **correcto funcionamiento**:
 - Construir un array con diferentes rectángulos
 - Ejecutar todos los análisis
 - Mostrar el resultado en la consola

Orientación a Objetos con clases

- JavaScript también permite implementar clases (similares a las de Java)
- Se pueden crear objetos como instancia de una clase (eso hace que tenga sus métodos y atributos)
- Al igual que en Java se permite:
 - Constructor y métodos
 - Herencia de clases
 - Métodos/Atributos estáticos

Orientación a Objetos con clases

Clase en JavaScript

```
class Empleado {

    constructor(nombre, salario){

        this.nombre = nombre;
        this.salario = salario;
    }

    getNombre(){
        return this.nombre;
    }

    toString(){
        return "Nombre:"+this.nombre+
            ", Salario:"+this.salario;
    }
}
```

Clase en Java

```
public class Empleado {

    private String nombre;
    private double salario;

    public Empleado(String nombre, double salario){

        this.nombre = nombre;
        this.salario = salario;
    }

    public String getNombre(){
        return nombre;
    }

    public String toString(){
        return "Nombre:"+nombre+
            ", Salario:"+salario;
    }
}
```


Orientación a Objetos con clases

```
class Shape {  
  
    constructor (id, x, y) {  
        this.id = id  
        this.move(x, y)  
    }  
  
    move (x, y) {  
        this.x = x  
        this.y = y  
    }  
  
    toString() {  
        return 'Shape('+this.id+')'  
    }  
}
```

Ejemplo6D

Orientación a Objetos con clases

Ejemplo6D

```
class Rectangle extends Shape {  
  
    constructor (id, x, y, width, height) {  
        super(id, x, y)  
        this.width = width  
        this.height = height  
    }  
  
    toString () {  
        return "Rectangle > " + super.toString()  
    }  
}
```

Orientación a Objetos con clases

Ejemplo6D

```
class Rectangle extends Shape {  
    ...  
    static defaultRectangle () {  
        return new Rectangle("default", 0, 0, 100, 100)  
    }  
}  
  
let defRectangle = Rectangle.defaultRectangle()
```

Orientación a Objetos con clases

Métodos y atributos privados

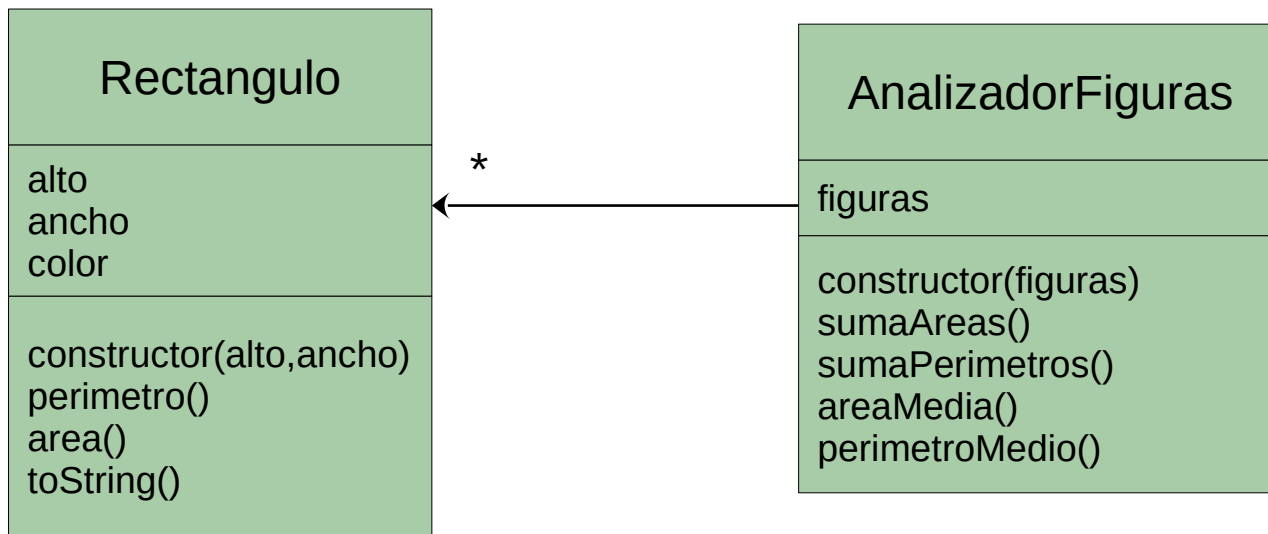
```
class ClassWithPrivate {  
    #privateField;  
    #privateFieldWithInitializer = 42;  
  
    #privateMethod() {  
        This.#privateField = "default";  
    }  
  
    static #privateStaticField;  
    static #privateStaticFieldWithInitializer = 42;  
  
    static #privateStaticMethod() {  
        // ...  
    }  
}
```

Orientación a Objetos con clases

- En JavaScript **no existe** algo equivalente a los **interfaces de Java**
- No son necesarios con **tipado dinámico**

Ejercicio 6

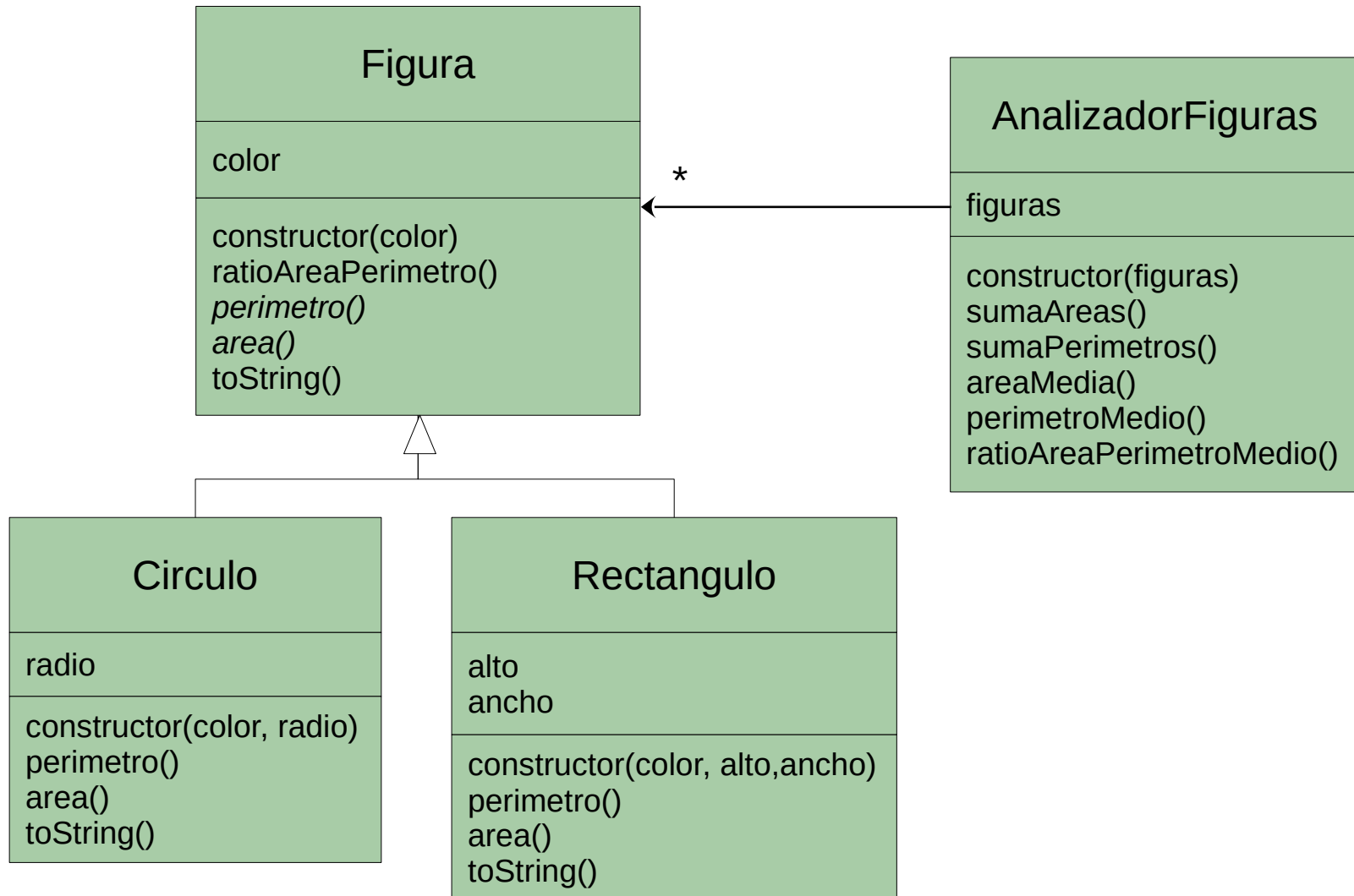
- Transforma el Ejercicio 5 para que use clases
- Agrupa las funciones de análisis en la clase AnalizadorFiguras



Ejercicio 7

- Amplía el Ejercicio 6 con las siguientes funcionalidades:
 - Añadir el círculo como nuevo tipo de figura
 - Área: $\text{Math.PI} * \text{radio} * \text{radio}$
 - Perímetro: $2 * \text{Math.PI} * \text{radio}$
 - Añadir un nuevo análisis:
 - Por cada figura saber el ratio area/perímetro
 - La media del ratio área/perímetro para todas las figuras

Ejercicio 7



JavaScript

- Introducción
- El lenguaje JavaScript
- **HTML Interactivo**

HTML Interactivo

- JavaScript se diseñó para dotar de **interactividad a las páginas HTML** cargadas en el navegador
- Se puede **modificar la página HTML** con JavaScript
 - **Cambio de CSS** de un elemento (ocultar, cambio de color...)
 - **Cambio del contenido:** Texto, imágenes, ...
- Se puede **ejecutar código** JavaScript cuando el usuario interactúa con la página (**click, hover, etc...**)

Document Object Model (DOM)

- Desde JavaScript se puede acceder al documento y al navegador
 - El **Document Object Model (DOM)** es una librería para manipular documentos HTML

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

- El **Browser Object Model (BOM)** permite manipular otros elementos del navegador

<https://developer.mozilla.org/en-US/docs/Web/API/Window>

<https://developer.mozilla.org/en-US/docs/Web/API/Navigator>

<https://developer.mozilla.org/en-US/docs/Web/API/Location>

Seleccionar una parte del documento

- Existe una variable **document** que apunta al documento DOM cargado en el navegador

<https://developer.mozilla.org/en-US/docs/Web/API/Document>

- Se pueden seleccionar partes del documento usando su id, class, name o tag

```
let image1 = document.getElementById('image1');  
let citas = document.getElementsByClassName('cita');  
let img2 = document.getElementsByName('image2');  
let links = document.getElementsByTagName('a');
```

<https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById>

Modificar el documento con HTML

- Para cambiar el contenido del documento cargado en el browser se selecciona un elemento del documento y se cambia su contenido HTML con la propiedad **innerHTML**

```
let element = document.getElementById('txt');  
element.innerHTML = '<p>Nuevo texto</p>'
```

<https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML>

Modificar el documento con HTML

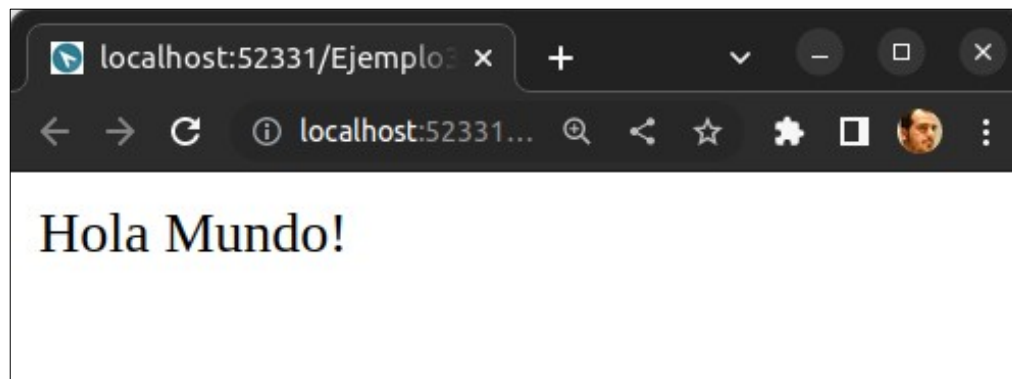
app.js

```
let content = document.getElementById('content');
content.innerHTML = '<p>Hola Mundo!</p>'
```

Ejemplo7A

index.html

```
<html>
  <body>
    <div id='content'></div>
    <script src='app.js'></script>
  </body>
</html>
```



Gestión de Eventos

- Se pueden ejecutar **funciones** cuando se interactúa con elementos del documento
- Ejemplo: Mostrar una alerta al pulsar un botón

Ejemplo7B

```
function alerta() {
    alert('El botón ha sido pulsado');
}
```

```
<html>
  <body>
    <script src='app.js'></script>
    <button onclick="alerta();">Botón</button>
  </body>
</html>
```

Gestión de Eventos

Ejemplo7C

app.js

```
function saluda() {  
    let content = document.getElementById('content');  
    content.innerHTML = '<p>Hola Mundo!</p>';  
}
```

index.html

```
<html>  
  <body>  
    <button onclick="saluda()">Saluda!</button>  
    <div id='content'></div>  
    <script src='app.js'></script>  
  </body>  
</html>
```


Gestión de Eventos

Ejemplo7C



Gestión de Eventos

- Documentación sobre Eventos en JavaScript y DOM

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

- Listado de eventos que se pueden producir en un navegador web

<https://developer.mozilla.org/en-US/docs/Web/Events>

- Evento click de un elemento HTML

https://developer.mozilla.org/en-US/docs/Web/API/Element/click_event

Ejercicio 8

- Implementa una web que muestre los títulos de unos libros
 - Cien años de soledad
 - El señor de los anillos
 - 1984
 - Un mundo feliz
- Los títulos de los libros estarán almacenados en un array de JavaScript

Gestión de Eventos

- En vez de configurar el evento en el HTML, también se puede configurar en el código JavaScript con el método **addEventListener**

Ejemplo7D

index.html

```
<html>
  <body>
    <button>Saluda!</button>
    <div id='content'></div>
    <script src='app.js'></script>
  </body>
</html>
```

Gestión de Eventos

- En vez de configurar el evento en el HTML, también se puede configurar en el código JavaScript con el método **addEventListener**

Ejemplo7D

app.js

```
function saluda() {
    let content = document.getElementById('content');
    content.innerHTML = '<p>Hola Mundo!</p>';
}

let buttons = document.getElementsByTagName('button');

buttons[0].addEventListener('click', saluda);
```

Gestión de Eventos

- La función que se ejecuta cuando se produce un evento (*handler*) puede recibir como parámetro información del evento

Ejemplo7E

app.js

```
let buttons = document.getElementsByTagName('button');

function greet(event) {
  console.log('Evento:', event);
  console.log('Elemento origen: ', event.target);
}

buttons[0].addEventListener('click', greet);
```

https://developer.mozilla.org/en-US/docs/Web/Events/Event_handlers

<https://developer.mozilla.org/en-US/docs/Web/API/Event>

Carga del documento DOM

- Si el script se configura al **final del html**, se ejecuta cuando el documento ya está **cargado** en memoria
- Si el script se configura en el **<head>**, se ejecuta **antes** de que el documento haya sido cargado

```
<html>
  <body>
    ...
    <script src='app.js'></script>
  </body>
</html>
```

```
<html>
  <head>
    <script src='app.js'></script>
  </head>
  <body>
    ...
  </body>
</html>
```

Carga del documento DOM

- Si ponemos el script en el **<head>** hay que esperar a que se haya cargado la página para insertar el contenido desde JavaScript

Ejemplo7F

index.html

```
<html>
  <head>
    <script src='app.js'></script>
  </head>
  <body>
    <div id='content'></div>
  </body>
</html>
```


Carga del documento DOM

- El documento tiene el evento **DOMContentLoaded** que se ejecuta cuando el documento se ha cargado
- Si no se espera a la carga del DOM, al buscar el elemento "content" no se encontrará

app.js

Ejemplo7F

```
document.addEventListener("DOMContentLoaded", function(){

    let content = document.getElementById('content');
    content.innerHTML = '<p>Hola Mundo!</p>';

});
```

Modificar los estilos

- Se puede cambiar cualquier estilo de un elemento con su propiedad **style**

```
let element = document.getElementById('img1');
element.style.borderWidth = width + 'px';
```

<https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style>

- Ocultar y mostrar un elemento con estilos

```
let element = document.getElementById('text');
element.style.display = 'none';
...
element.style.display = 'block';
```

Ejercicio 9

- Crea una página con un botón que oculte el texto “Hola Mundo” cuando se pulse
- Si se pulsa de nuevo, se volverá a mostrar el texto “Hola Mundo”

Modificar los estilos

- Una forma común de cambiar los estilos es añadiendo o eliminando clases CSS a un elemento HTML

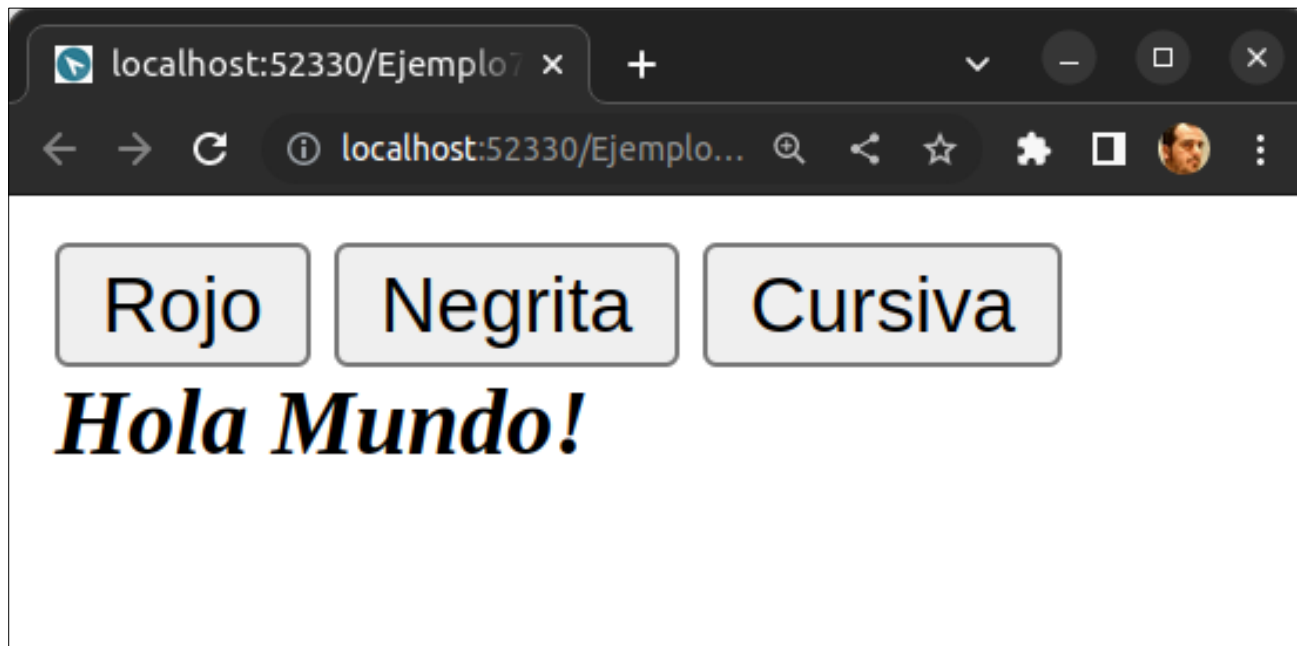
```
let content = document.getElementById('content');
...
content.classList.remove('rojo');
...
content.classList.add('negrita');
...
content.classList.toggle('cursiva');
```

<https://developer.mozilla.org/en-US/docs/Web/API/Element/classList>

Modificar los estilos

- Ejemplo con botones que añaden o eliminan una clase (*toggle*) de un elemento

Ejemplo7G



Modificar los estilos

- Ejemplo con botones que añaden o eliminan una clase (*toggle*) de un elemento

Ejemplo7G

```
function rojo(){
  let content = document.getElementById( 'content' );
  content.classList.toggle( 'rojo' );
}
```

```
<body>
  <button onclick='rojo()'>Rojo</button>
  <div id='content'>Hola Mundo!</div>
  ...
</body>
```

```
.rojo {
  color: #F00;
}
```

Ejercicio 10

- Amplía el Ejercicio 8 para que al lado de cada libro aparezca un botón [Más info]
- Al pulsar el botón aparecerá debajo del título el autor del libro y su año de publicación
- Un elemento se puede ocultar con el estilo “display:none” y mostrarse de nuevo con “display:block”

Ejercicio 10

- Cuando el código HTML es complejo se pueden usar *template literals*
 - Se usan comilla ` (*backtick*) para multilínea
 - Se usa `${...}` para las variables

```
let html =
  `<div>
    <p>${libro.titulo}</p>
    <p> ${libro.autor} (${libro.año})</p>
  </div>`;
```


Ejercicio 10

- Información libros

- Cien años de soledad, de Gabriel García Márquez (1967)
- El señor de los anillos, de J. R. R. Tolkien (1954)
- 1984, de George Orwell (1949)
- Un mundo feliz, de Aldous Huxley (1932)

Generación de HTML

- Existen diversas formas de generar código HTML desde código JavaScript
 - **Cadena de caracteres (*String*)**
 - Creando los nodos del árbol DOM

Generación de HTML

- Cadena de caracteres (*String*)

```
let text = "Hola Mundo!";  
  
let content = document.getElementById("content");  
  
content.innerHTML = "<p>" + text + "</p>";
```

Generación de HTML

- Existen diversas formas de generar código HTML desde código JavaScript
 - Cadena de caracteres (*String*)
 - **Creando los nodos del árbol DOM**

Generación de HTML

- Creando los nodos del árbol DOM

```
let text = "Hola Mundo!";

let content = document.getElementById("content");

content.innerHTML = "<p>" + text + "</p>";
```



```
let text = "Hola Mundo!";

let content = document.getElementById("content");

let p = document.createElement("p");
p.textContent = text;
content.appendChild(p);
```

Generación de HTML

- Creando los nodos del árbol DOM

Ejemplo7H

```
let text = "Hola Mundo!";

let content = document.getElementById("content");

let p = document.createElement("p");
p.textContent = text;
content.appendChild(p);
```

<https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement>

<https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent>

<https://developer.mozilla.org/en-US/docs/Web/API/Node/appendChild>

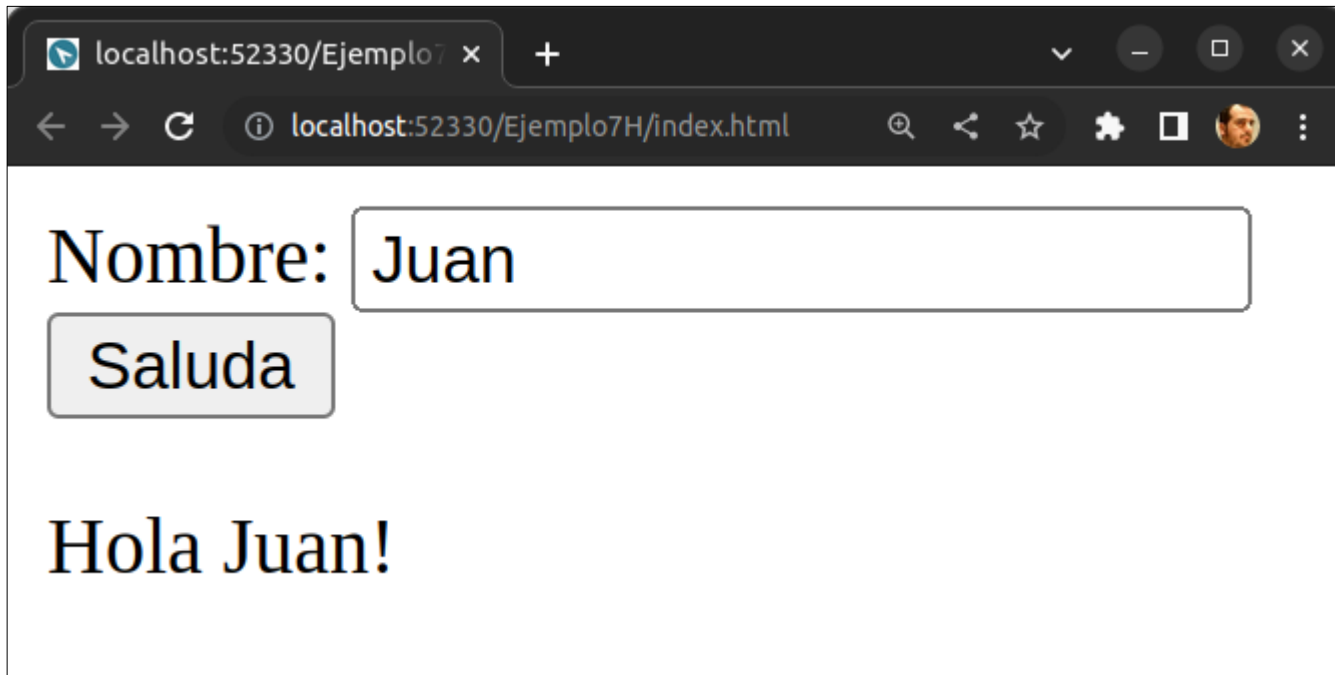
Ejercicio 11

- Actualiza el ejercicio 9 para no generar el HTML como texto y usar la API de nodos del DOM

Formularios

- Se puede obtener información de campos de formularios

Ejemplo7I



A screenshot of a web browser window. The address bar shows 'localhost:52330/Ejemplo7' and the page title is 'Ejemplo7H/index.html'. The main content area displays a form with the label 'Nombre:' followed by a text input field containing 'Juan'. Below the input field is a button labeled 'Saluda'. At the bottom of the form, the text 'Hola Juan!' is displayed.

- Se puede obtener información de campos de formularios

Ejemplo71

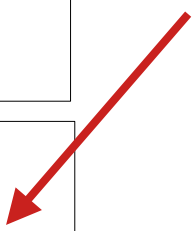
```
<div>
  Nombre: <input id='nombre' type='text'></input>
  <button onclick='saluda()'>Saluda</button>
</div>
<div id='content'></div>
```

```
function saluda() {

  let nombre = document.getElementById('nombre').value;

  let content = document.getElementById('content');
  content.innerHTML += '<p>Hola ' + nombre + '!</p>';

}
```



Ejercicio 12

- Amplía el ejercicio 11 para que se puedan dar de alta nuevos libros con un formulario

Ejercicio 13

- Crea una página que muestre el título de varios libros
- Al lado de cada título aparecerá el botón “Más info”
- Al pulsar el botón se borrará la lista de libros y aparecerá la información del libro junto con un botón “Volver a la lista”

Ejercicio 13

