



# Seminario de tecnología aplicada al diseño

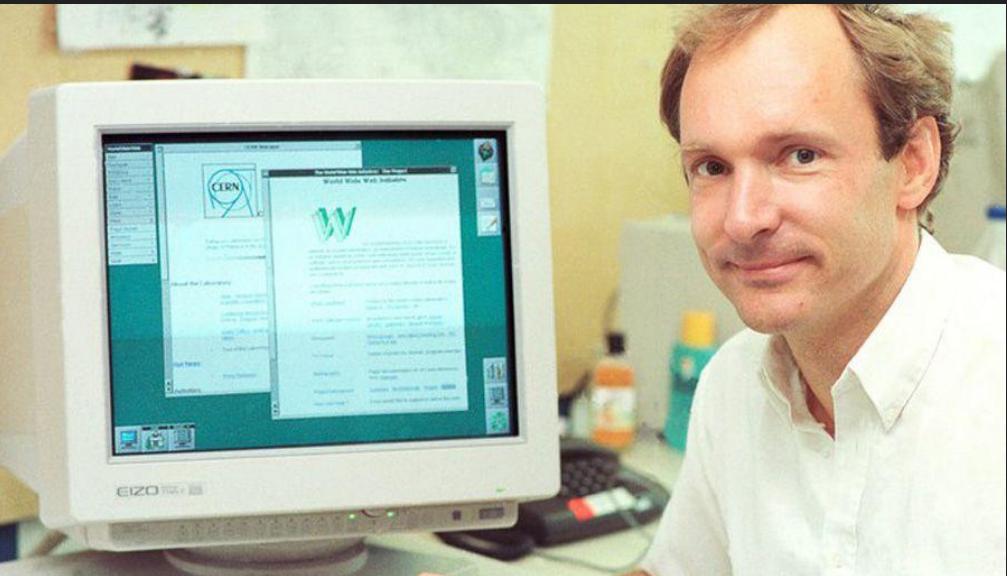
Cátedra Diseño de sistemas

# HTML + CSS

<HTML>



# Sir Timothy Berners Lee.



- Científico de computación Británico.
- Conocido por ser el padre de la World Wide Web, estableció la primera comunicación entre un Cliente y un Servidor utilizando el protocolo HTTP.
- Ante la necesidad de distribuir e intercambiar información acerca de sus investigaciones de forma más efectiva desarrolló el lenguaje HTML *Hyper Text Markup Language* o *Lenguaje de Marcado de HiperTexto*.

# HTML ¿Cómo estructurar documentos?

OPEN  ACCESS Freely available online

PLOS BIOLOGY

## A Role for Parasites in Stabilising the Fig-Pollinator Mutualism

Derek W. Dunn<sup>1,2,3</sup>, Simon T. Segar<sup>1,2</sup>, Jo Ridley<sup>3</sup>, Ruth Chan<sup>1</sup>, Ross H. Crozier<sup>4</sup>, Douglas W. Yu<sup>3</sup>, James M. Cook<sup>1,2,5\*</sup>

**1** Division of Biology, Imperial College London, Ascot, United Kingdom, **2** School of Biological Sciences, University of Reading, Reading, United Kingdom, **3** School of Biological Sciences, University of East Anglia, Norwich, United Kingdom, **4** School of Marine and Tropical Biology, James Cook University, Townsville, Queensland, Australia, **5** Natural Environment Research Council (NERC) Centre for Population Biology, Imperial College London, Ascot, United Kingdom

Mutualisms are interspecific interactions in which both players benefit. Explaining their maintenance is problematic, because cheaters should outcompete cooperative conspecifics, leading to mutualism instability. Monoecious figs (*Ficus*) are pollinated by host-specific wasps (Agaonidae), whose larvae gall ovules in their “fruits” (syconia). Female pollinating wasps oviposit directly into *Ficus* ovules from inside the receptive syconium. Across *Ficus* species, there is a widely documented segregation of pollinator galls in inner ovules and seeds in outer ovules. This pattern suggests that wasps avoid, or are prevented from ovipositing into, outer ovules, and this results in mutualism stability. However, the mechanisms preventing wasps from exploiting outer ovules remain unknown. We report that in *Ficus rubiginosa*, offspring in outer ovules are vulnerable to attack by parasitic wasps that oviposit from outside the syconium. Parasitism risk decreases towards the centre of the syconium, where inner ovules provide enemy-free space for pollinator offspring. We suggest that the resulting gradient in offspring viability is likely to contribute to selection on pollinators to avoid outer ovules, and by forcing wasps to focus on a subset of ovules, reduces their galling rates. This previously unidentified mechanism may therefore contribute to mutualism persistence independent of additional factors that invoke plant defences against pollinator oviposition, or physiological constraints on pollinators that prevent oviposition in all available ovules.

Citation: Dunn DW, Segar ST, Ridley J, Chan R, Crozier RH, et al. (2008) A role for parasites in stabilising the fig-pollinator mutualism. PLoS Biol 6(3): e59. doi:10.1371/journal.pbio.0060059

### Introduction

In a biosphere driven by selection at the level of the individual gene [1], explaining the existence of cooperation, such as mutualism, is a major scientific challenge. Mutualisms are interspecific ecological interactions characterised by reciprocal benefits to both partners [2] that usually involve costly investments by each. What factors thus prevent one partner from imposing unsustainable costs onto the other to enable mutualism stability [3–7]? In some mutualisms, the larger, more sessile partner, manipulates the other by directing benefits to cooperative individuals and costs to cheaters [4–7]. However, a general consensus on mutualism persistence has only recently been formulated, and this clearly shows that a high benefit-to-cost ratio of cooperating is one important factor [8,9].

Fig trees (*Ficus*) and their host-specific agaonid pollinator wasps are a classic example of an obligate mutualism [10,11]. The wasps pollinate the trees, and the trees provide resources for wasp offspring. In monoecious *Ficus*, female wasps push their way through a specialised entrance into receptive syconia (colloquially, “figs”), which are enclosed inflorescences. The wasps then pollinate the tree while depositing their eggs individually into ovules. Thus, each egg laid costs the tree one seed, but upon emergence, the female wasp offspring disperse that tree’s pollen. Trees need to produce both wasps and seeds for the mutualism to persist, but natural selection should favour wasps that exploit the maximum number of fig ovules in the short term, resulting in a conflict of interest between wasp and tree. However, the mutualism has persisted for at least 60 million years and has radiated into more than 750 species pairs [12]. The mechanisms preventing wasps

from overexploiting figs remain unknown, despite intensive study over four decades.

Within receptive syconia, the lengths of floral styles are highly variable [13,14], and ovipositing pollinators (foundresses) favour flowers with shorter styles for their offspring [15–18]. Style and pedicel lengths of flowers are negatively correlated. Short-styled ovules develop into seeds or galls (when a wasp is present) near the syconium inner cavity, while most long-styled ovules develop into seeds near the outer wall [19,20] (Figure 1). These patterns have been shown to reflect the oviposition preferences of foundresses, and are unlikely to be the result of greater elongation of pedicels containing eggs during syconial maturation, because in receptive syconia, pollinators’ eggs are mainly present in short-styled inner ovules [16]. These widespread observations have been tied to four, not necessarily mutually exclusive, mechanisms that have been proposed to stabilise the fig-pollinator mutualism: (1) Unbeatable seeds—outer ovules may be defended biochemically or physically against oviposition or larval development [21]. However, no mechanism has yet been identified. (2) Short ovipositors—pollinators’ ovipositors may be too short to fully penetrate the long styles of

Academic Editor: Anurag A. Agrawal, Cornell University, United States of America

Received September 14, 2007; Accepted January 22, 2008; Published March 11, 2008

**Copyright:** © 2008 Dunn et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

**Abbreviation:** s.e., standard error

\* To whom correspondence should be addressed. E-mail: james.cook@reading.ac.uk



# HTML - HyperText Markup Language.

Formado por **ETIQUETAS** y **ATRIBUTOS** que en conjunto conforman **ELEMENTOS**.

<h1> ..... </h1>



Etiqueta de Apertura



Etiqueta de Clausura

# HTML - HyperText Markup Language.

Sintaxis de una etiqueta.



# Etiquetas

Definición

- Son la parte del código que permite generar un elemento visual en el buscador.
- Sirven como “Esqueleto” de nuestra página web.

# Atributos y Valores

Definiciones

- Atributo: Característica que deseamos modificar de una etiqueta.
- Valor: definición de la característica que vamos a modificar.

# Estructura básica de un documento HTML.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <title> Título de mi página web </title>
  </head>
  <body>
    Hola mundo!
  </body>
</html>
```

El charset permite definir la codificación de caracteres a utilizar. Se Utiliza para evitar la visualización incorrecta en algunos navegadores.

Títulos.

# Tags HTML

<h1> Título Principal </h1>

<h2> Subtítulo </h2>

<h3> Otro subtítulo </h3>

<h4> ... </h4>

<h5> ... </h5>

<h6> ... </h6>

Nota: Este tag es útil para el posicionamiento en buscadores.  
El H1 debería utilizarse una sola vez en la página.

# Tags HTML

Párrafos.

<p>

  Lorem ipsum dolor sit amet,  
  consectetur adipisicing elit, sed  
  do eiusmod tempor incididunt ut  
  labore et dolore magna aliqua.

</p>

<p>

  Excepteur sint occaecat  
  cupidatat non proident, sunt in  
  culpa qui officia deserunt mollit  
  anim id est laborum.

</p>

# Tags HTML

Listas Ordenadas.

```
<ol>  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
</ol>
```

Tag para listar ítems de manera consecutiva y numerada.

En el navegador se vería:

1. Item 1
2. Item 2
3. Item 3

# Tags HTML

Listas  
Desordenadas.

```
<ul>  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
</ul>
```

Tag para listar ítems de manera consecutiva, sin numerar.

En el navegador se vería:

- Item 1
- Item 2
- Item 3

# Tags HTML

Tags para división  
del html.

Además de los tags para definir Párrafos, Títulos, Listas, etc. También existen tags que utilizaremos para **dividir** nuestro documento html en diferentes **secciones**.

```
<body>...</body>
<header> .... </header>
<footer> ... </footer>
<article>...</article>

<div>...</div>
```

# Tags HTML

Tags para división  
del html.

BODY

ENCABEZADO

ARTÍCULO

TÍTULO

PÁRRAFO

DIVISIÓN ESPECIAL

PIE DE PÁGINA

# Tags HTML

Tags para división  
del html.

¿Quién se anima a escribirlo en  
código?

# Tags HTML

Otros tags útiles.

- <table>...</table>
- <nav>...</nav>
- <form>...</form>
- <br>

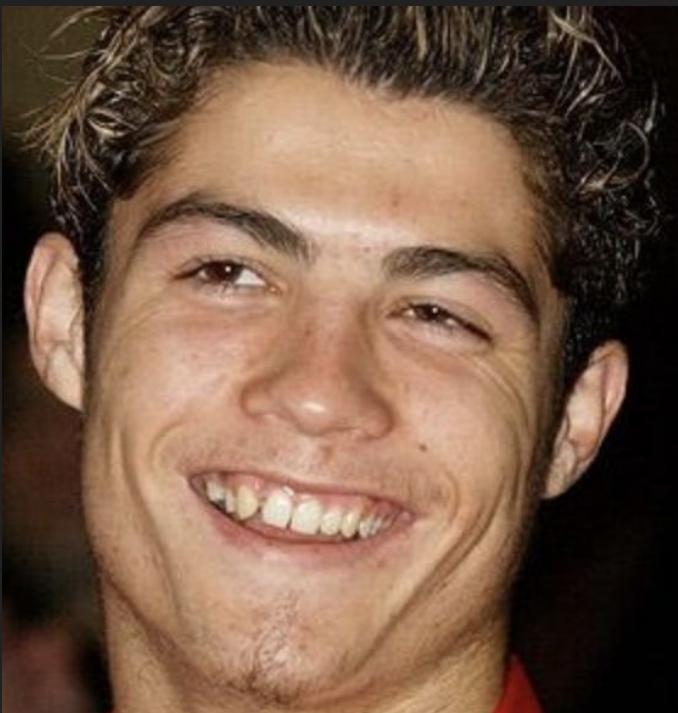
Y muchos más...

# **CSS**

# **Cascading Style Sheets**

# CSS: Cascading Style Sheets

Sin CSS



Con CSS



# CSS: Cascading Style Sheets

Las hojas de estilo sirven para “estilizar” nuestro contenido HTML. Con CSS podemos cambiar colores, fondos, tipografías, anchos, altos, posiciones, etc.

# Un poco de historia

El comienzo de los estilos...

## En línea.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <title> Título de mi página web </title>
  </head>
  <body>
    <h1 style="color: red;"> Mi nombre es Rodrigo </h1>
  </body>
</html>
```

# CSS

## SIN ESTILOS



## CON ESTILOS



# En línea

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <title> Título de mi página web </title>
  </head>
  <body>
    <h1 style="color: red; font-size:24px; text-align:center; text-decoration:underline;....." > Mi nombre es Rodrigo </h1>
  </body>
</html>
```

# CSS Interno

```
<style>
  h1{
    color:red;
    font-size:28px;
    text-decoration:underline;
    text-align: center;
  }
</style>

<!DOCTYPE html>
<html>
  <head>
    <meta charset = “utf-8”>
    <title> Título de mi página web </title>
  </head>
  <body>
    <h1> Mi nombre es Rodrigo </h1>
  </body>
</html>
```

# CSS

Mi nombre es Rodrigo.

# CSS Externo, el definitivo.

.HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = “utf-8”>
    <title> Título de mi página web </title>
  </head>
  <body>
    <h1> Mi nombre es Rodrigo</h1>
    <p> Estoy haciendo mi primer página
web con estilos </p>
  </body>
</html>
```

.CSS

```
body{
  background-color: black;
}

h1{
  color:blue;
  font-size:28px;
  text-decoration:underline;
  text-align: center;
}

p{
  color:white;
}
```

# ¿Como relacionar el html con el css?

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <link rel="stylesheet" href="estilos.css">
    <title> Título de mi página web </title>
  </head>
  <body>
    <h1> Mi nombre es Rodrigo</h1>
    <p> Estoy haciendo mi primer página web con estilos </p>
  </body>
</html>
```

# CSS.

## Tamaño de fuente

font-size: 22px;

## Alineación del texto:

text-align: center/left/right.

## Color de fuente:

color: red;

## Color de fondo:

background-color: rgb(142,98,132)

## Subrayado:

text-decoration: underline;

## Ancho y alto:

width: 500px;  
height: 700px;

## Márgenes exteriores:

margin: 10px;  
margin-top: 40px;

## Márgenes Interiores:

padding:5px;

## Bordes:

border: 1px solid rgb(142, 98, 132);  
border-radius: 5px;

# Practiquemos un poco antes de continuar...

Crear un documento html que se vea así:

## Taller de FrontEnd.

El FrontEnd es la parte de un sitio web que interactúa con los usuarios, por eso decimos que está del lado del cliente.

Con las hojas de estilo podemos:

- Modificar Colores.
- Cambiar fuentes y tamaños.
- Agregar Sombreados, subrayados, etc.
- Cambiar tamaños de contenedores.

- Estilos para todos.
- Buuuuuuh!!!!
- De acuerdo, no hay estilos para nadie.
- Buuuuuuuuuuuuh!!!



# Estilos para unos... Banderitas estadounidenses para otros.

En nuestra página web vamos a tener tags repetidos.  
¿Cómo le aplico estilos a ciertas etiquetas en particular?

## Vamos a definir CLASES:

Desde nuestra hoja de estilos, definimos las propiedades de estilo que nuestra clase tendrá.  
Se definen comenzando con un Punto.

```
.titulo{  
    color: green;  
    font-size: 60px;  
}
```

# En el html

Una vez definida la clase y sus propiedades de estilos, debemos informarle a QUÉ elemento le aplicamos dicha clase.

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="estilos.css">
    <meta charset = "utf-8">
    <title> Título de mi página web </title>
  </head>
  <body>
    <h1 class = "titulo"> Mi nombre es Rodrigo </h1>
  </body>
</html>
```

# Entonces...

```
.css   .Selector{ atributo:valor; }
```

```
.html   class = “nombreClase”
```

## Más selectores...

.Class

.Selector{ atributo: valor;}

#id

#Selector{ atributo:valor;}

tag

Selector { atributo:valor;}

Nota: No puede haber Ids repetidos.

# Más selectores...

## Descendente

```
.Selector elemento{ atributo: valor;}
```

# Modelo de cajas

¿Cómo  
localizamos los  
elementos en  
nuestro HTML ?

## This is a heading (for testing purposes)

Paragraphs are the bread and butter of HTML.

They can include **strong** tags.

1. Lists are cool!
2. The list things.
3. This one has an **i** inside.
4. This one contains a <ul>:
  - Item 1
  - **Marked item**
  - Final item
5. **What happens with an h2?**
6. Final order

A final paragraph with a dummy [link](#) inside it.

# ¿Qué es el atributo **display**?

Todos los elementos de nuestro HTML tienen el atributo “**display**” que define cómo se muestran y cómo se comportan respecto otros elementos.

Si desde nuestra hoja de estilos, le asignamos a un elemento el atributo “**display:none;**”, dicho elemento no será visible en nuestra página.

# **display:block; VS display:inline;**

## DISPLAY: BLOCK;

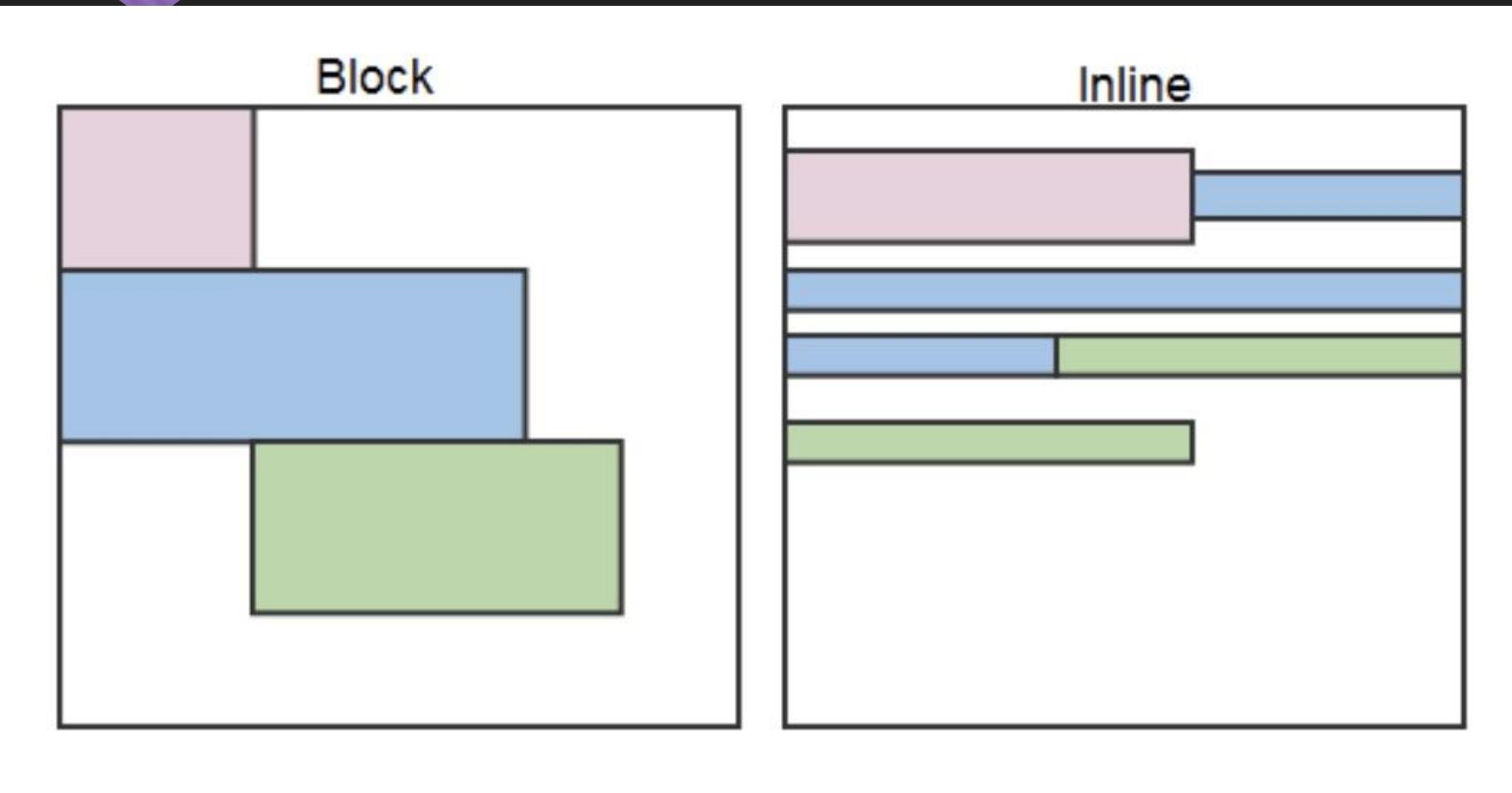
Define un elemento con comportamiento de **bloque**.

Recibe las propiedades del modelo de caja.

## DISPLAY:INLINE;

Define un elemento con comportamiento **en línea**. No recibe algunas propiedades del modelo de caja.

# display:block; VS display:inline;



# **display:block; VS display:inline;**

## **BLOCK:**

```
<div></div>  
<header></header>  
<h1></h1>  
<ul></ul>
```

## **INLINE:**

```
<a></a>  
<img>  
<button> </button>
```

Nota: Desde nuestra hoja de estilos (css) podemos modificar el “display” de los diferentes elementos.

# JavaScript



# JavaScript

JS

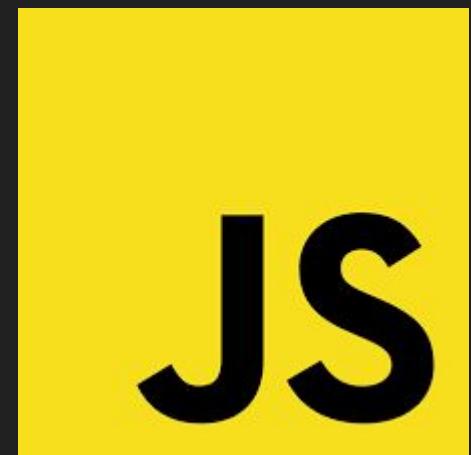
- JavaScript es un lenguaje de programación interpretado (no se compila)
- Creado en 1995 por Netscape
- No tiene nada que ver con Java (sólo parte de la sintaxis)
- La motivación inicial del lenguaje fue poder darle cierto dinamismo a los documentos HTML
- Todos los navegadores modernos interpretan JS.
- Hoy en día no se limita sólo al uso del lado del cliente.

# JavaScript

JS

- Entre otras cosas, con JS se puede manipular el documento
- Acceder a elementos del documento HTML.
- Modificar atributos de elementos
- Crear elementos nuevos o eliminar elementos existentes.
- Realizar requests al servidor

# JavaScript



¿Para qué nos sirve JavaScript?

- Juegos.
- Mapas.
- Galerías.
- Buscadores.
- Formularios.
- etc...

# JavaScript

## Cómo injectar JS en nuestro documento HTML?

Mediante un archivo externo:

```
<head>
  <script src="miArchivo.js"></script>
</head>
```

Mediante código inline

```
<head>
  <script>
    function miFuncion() {
    }
  </script>
</head>
```

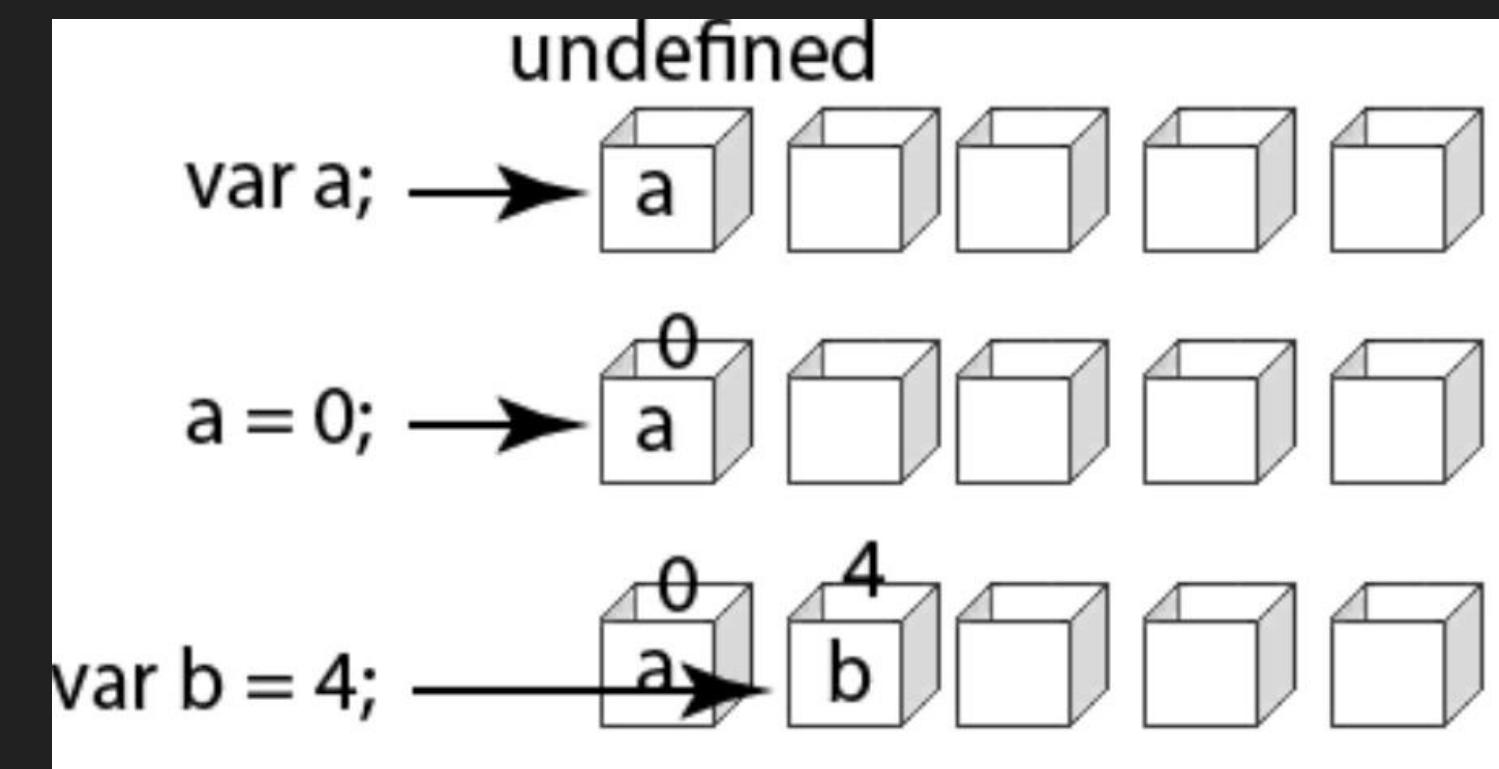
JS

**Sintaxis...**

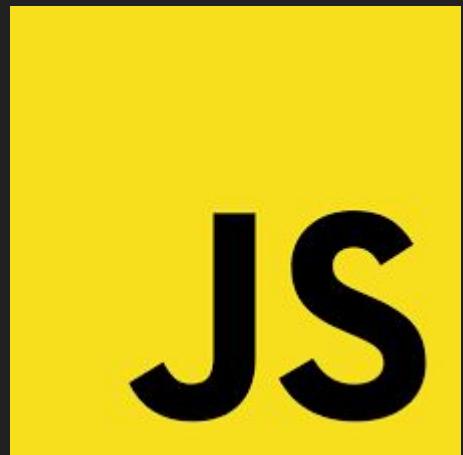
# JavaScript

JS

## DEFINICIÓN DE VARIABLES



# JavaScript



## Tipos base:

- String ---> “Hoy es sábado”
- Number ---> 10.4
- Boolean ---> true/false
- Object ---> {}
- Array---> []
- Null
- Undefined

# JavaScript

JS

IF

```
if (condición) {  
    //Hacer esto si se cumple la condición  
}
```

# JavaScript

JS

## SWITCH

```
switch(color) {  
    case "Rojo":  
        console.log("Usted eligió color Rojo");  
        break;  
    case "Azul":  
        console.log("Usted eligió color azul");  
        break;  
    default:  
        console.log("Es otro color");
```

# JavaScript

JS

## FOR

```
for (inicio; condicion; incremento){  
    //Hacer esto mientras la condicion sea verdadera  
}  
  
for(var i = 0; i < 10; i++){  
    console.log("El contador vale " + i);  
}
```

# JavaScript

JS

## WHILE

```
while (condicion){  
    // Realizar mientras sea verdadera la cond.  
}
```

# JavaScript

JS

## WHILE

```
while (condicion){  
    // Realizar mientras sea verdadera la cond.  
}
```

# JavaScript

JS

## ARRAYS

```
var profesores = ["Alejandro", "Ezequiel", "Rodrigo", "Lucas"];
```

0

1

2

3

# JavaScript

JS

## ARRAYS

Los objetos de tipo Array además se pueden iterar usando *forEach()*

```
1 ▶ let nombres = [
2     "Alejandro",
3     "Ezequiel",
4     "Rodrigo",
5     "Lucas"
6 ];
7
8 ▶ nombres.forEach(nombre => {
9     // hacer algo con el nombre
10    console.log(nombre);
11});
```

# JavaScript

## OBJETOS

JS

- Un objeto es una colección de propiedades.
- Una propiedad es una asociación entre un nombre (una clave) y un valor.
- El valor de una propiedad puede ser una función, en cuyo caso se denomina “Método”.

# JavaScript

JS

## OBJETO

```
var auto = {  
    marca = “Chevrolet”,  
    modelo = “Corsa”,  
    kilometraje = “120000”,  
    color = “Blanco”  
}
```

# JavaScript

JS

Podemos acceder a una propiedad de un objeto de varias formas:

auto.marca ---> “Chevrolet”

auto[“marca”] ---> “Chevrolet”

# **DOM**

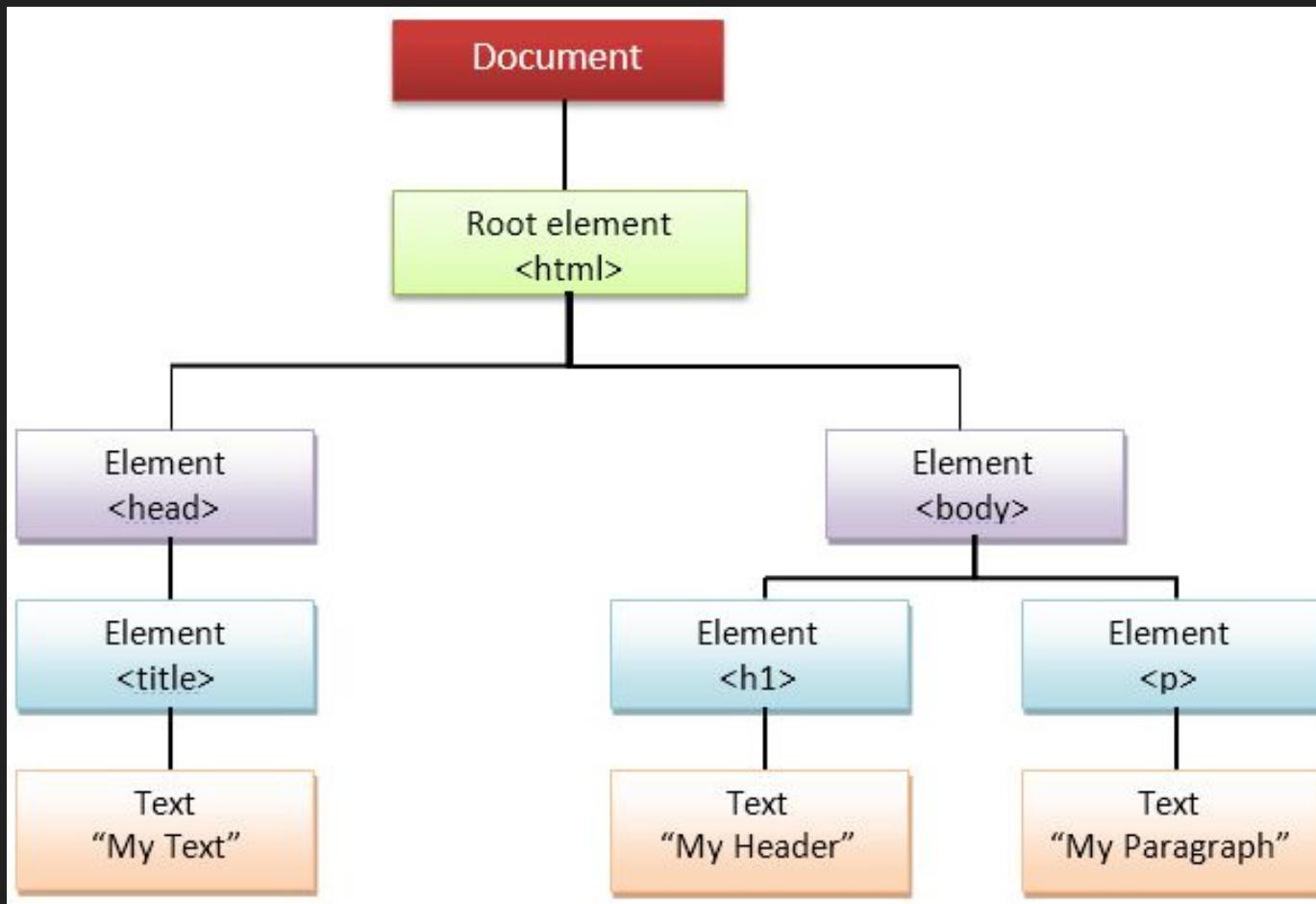
## **Domain Object Model**

# DOM - Domain Object Model

JS

- Es un modelo que representa en forma de árbol la estructura de un documento HTML.
- Mediante este modelo se puede "navegar" a través del documento para manipular los elementos

# DOM - Domain Object Model



# DOM API

Para manipular los nodos se utiliza la DOM API

[https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model)

A través de esta API se puede acceder a los nodos, iterarlos, modificar atributos, crear nuevos elementos, etc.

Algunos métodos y propiedades útiles se detallan a continuación.

JS

# DOM API

## Métodos para seleccionar

- `document.getElementById(<id>);`
- `document.querySelectorAll(<selector>);`

## Métodos para trabajar con Elementos

- `document.createElement(<tag>);`
- `<element>.setAttribute(<att>, <value>);`
- `<element>.getAttribute(<att>, <value>);`
- `<element>.removeAttribute(<att>);`
- `<element>.classList`
- `<element>.innerHTML = <value>`
- `<element>.appendChild(otro element)`

JS

# DOM API

## HTML + JS

HTML ▾

```
1 <div class="cuerpo"></div>
2 ▼ <button onclick="demo()">Agregar parrafo</button>
3 .
```

JavaScript + No-Library (pure JS) ▾

```
1 ▼ window.demo = function() {
2 ▼   document.querySelectorAll(".cuerpo").forEach(cuerpo => {
3     let element = document.createElement("p");
4     element.classList.add("azul");
5     element.setAttribute("align", "left");
6     element.innerHTML = "hola";
7     cuerpo.appendChild(element);
8   });
9 }
```

## OUTPUT

```
<div class="cuerpo">
  <p class="azul"
     align="left">hola
  </p>
</div>
```

JS

# DOM API - Ejercicio 1

## Objetivos:

- Leer el valor de un input
- Escuchar evento click de un botón
- Escribir en un elemento usando innerHTML

# Bienvenido anonimo

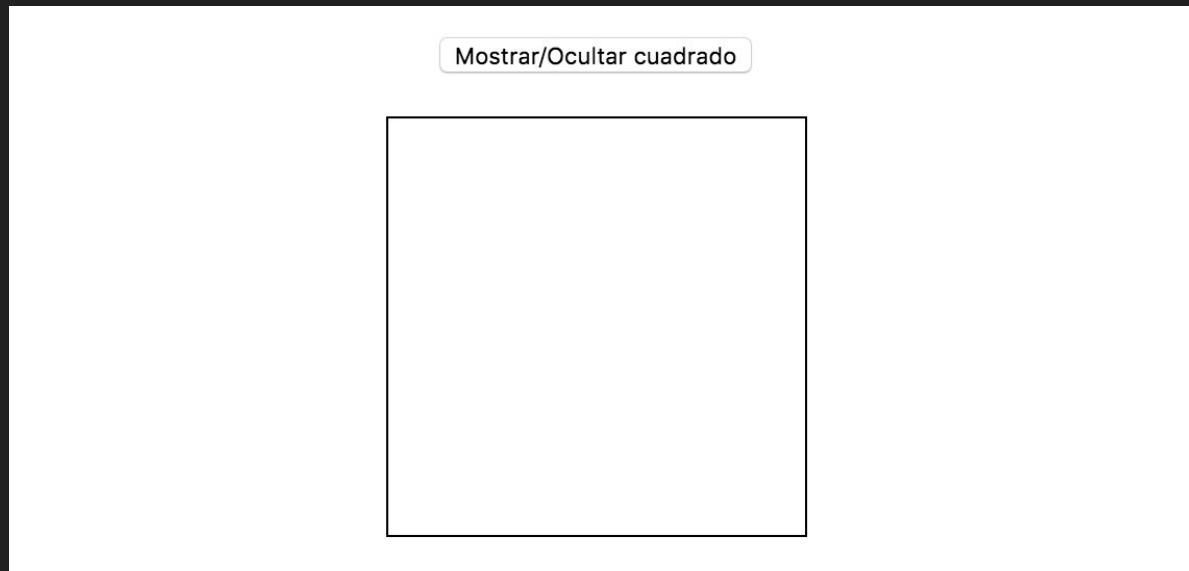
Ingrese nombre:  OK

JS

# DOM API - Ejercicio 2

## Objetivos:

- Escuchar evento click de un botón
- Familiarizarse con la propiedad *classList*
- Repasar CSS



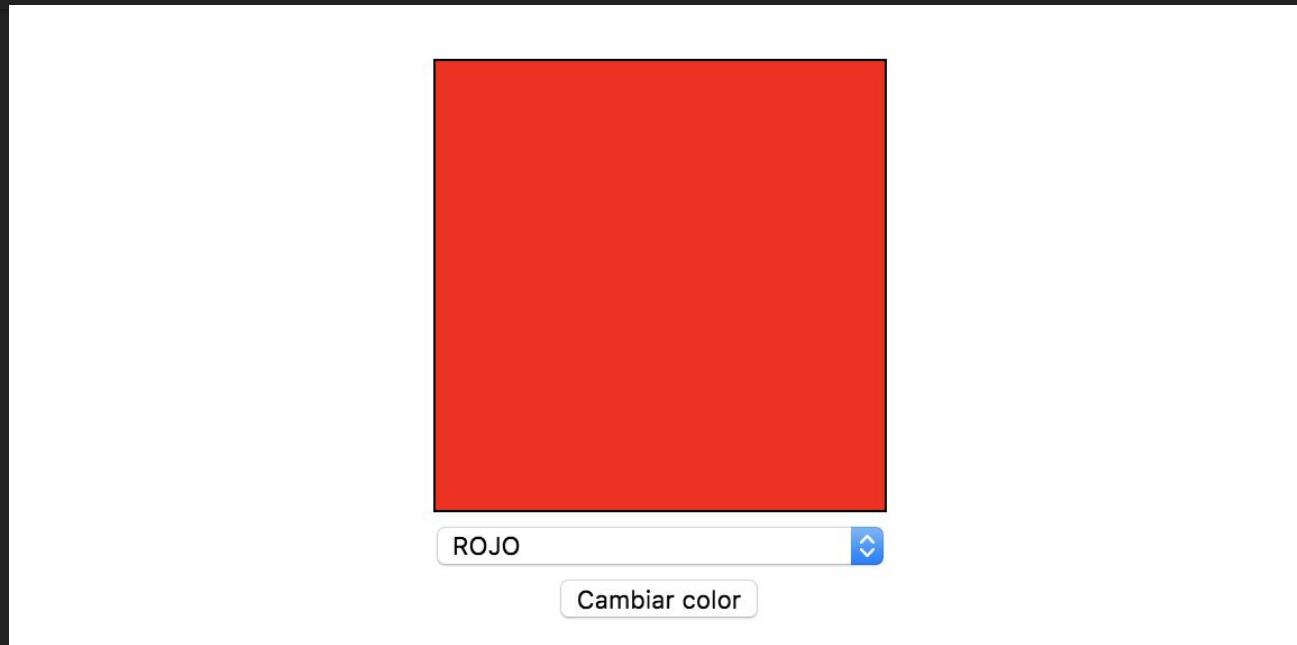
JS

Solución: <https://jsfiddle.net/magoleoz/hL13qo5s/7/>

# DOM API - Ejercicio 3

## Objetivos:

- Leer el valor de un SELECT
- Continuar con *classList*



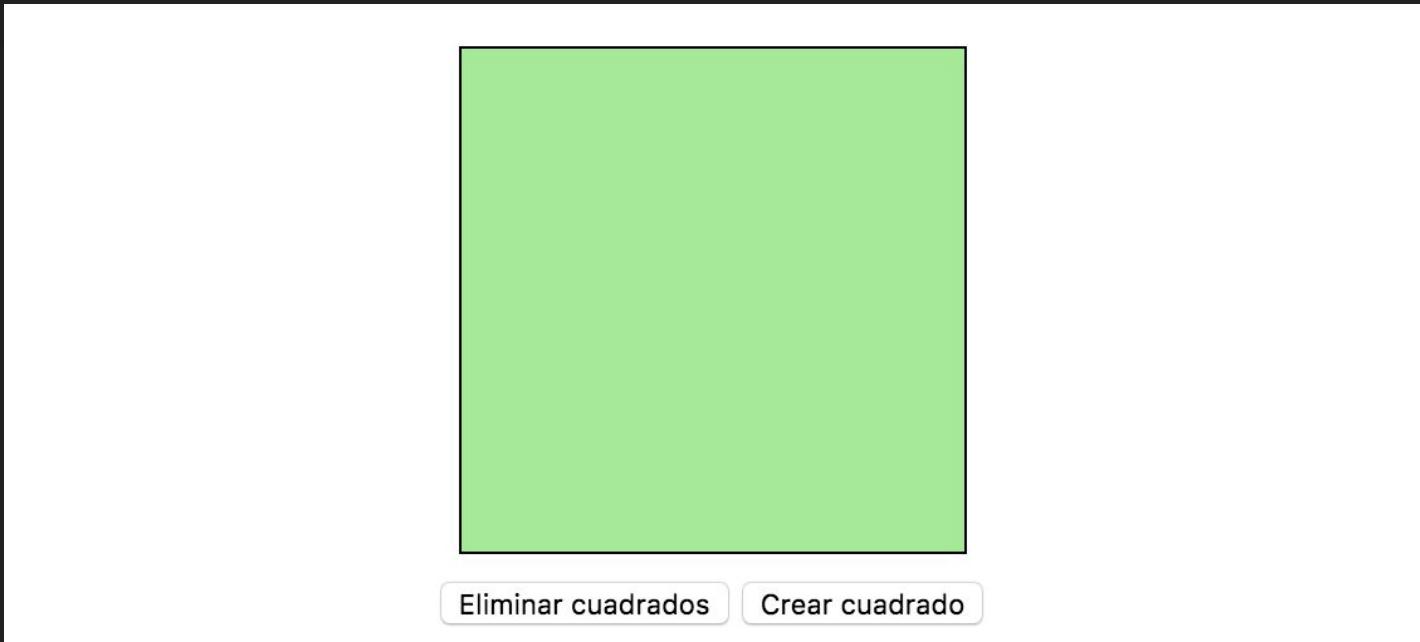
Solución: <https://jsfiddle.net/magoleoz/j3hqts5x/3/>

JS

# DOM API - Ejercicio 4

## Objetivo:

- Crear y eliminar elementos con DOM API



JS

# ¡Gracias!

Seminario de tecnología aplicada al  
diseño



UTN.BA

DPTO. INGENIERÍA EN SISTEMAS DE INFORMACIÓN  
CÁTEDRA DISEÑO DE SISTEMAS

