

# MF0491\_3

# Programación web en el entorno cliente



01

# **UF1841. Elaboración de documentos web mediante lenguajes de marca**

# Objetivos del curso



Elaborar documentos utilizando lenguajes de marcas y estándares de desarrollo software:

- Determinar las diferentes partes de un documento creado con lenguaje de marcas utilizado para su implementación.
- Reconocer las diferentes técnicas de desarrollo de software existentes en el mercado para mejorar la integración en el sistema y elaboración de documentos según el diseño especificado.
- Utilizar marcas adecuadas para generar la documentación interna en el desarrollo según las especificaciones del diseño.
- Usar marcas para proporcionar diferentes estilos a los documentos desarrollados según el diseño especificado.
- Construir documentos utilizando lenguajes de marcas para permitir al usuario el uso de dispositivos móviles y medios específicos de accesibilidad.

1. Diseño web
2. Lenguajes de marcado generales
3. Lenguajes de marcado para presentación de páginas web
4. Hojas de Estilos en Cascada  
Bibliografía y webgrafía

01

# Elaboración de documentos web mediante lenguajes de marca

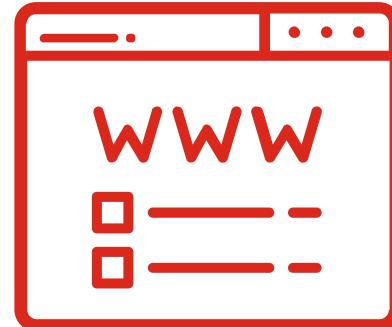
## 1. Diseño web

- Principios de diseño web
- El proceso de diseño web

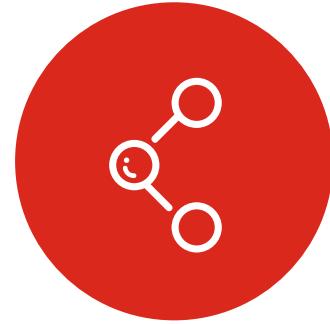
# Principios estéticos del diseño web



Contraste



Repetición



Alineación



Proximidad

# Principios estéticos del diseño web: **Contraste**

- Destacar elementos de la página para proporcionar aspecto visual atractivo.

The image shows two versions of a newsletter snippet side-by-side, demonstrating the principle of contrast in web design through color and font weight.

**Newsletter nº 46**

**NEWSLETTER Nº 46**

**LOREM IPSUM DOLOR SIT AMET**

LoREM iPSUM dOLOR sIT aMET, cONSECTETUR adIPISCING elIT. ViVAMUS VOLUPTAT MATTIS sagITTIS. MORBI sed SOLlicitUDIN nISL. IN non nISL nISI. IN tINCIDUNT ViVERRA ViVERRA.

**IN PELLentesque MOLLIS FEUGIAT**

IN pELLentesque MOLLIS FEUGIAT. CRAS IN BLANDIT dolor. MAURIS SEMPER FELIS AUGUE, AC pELLentesque LECTUS PuLVINAR EU. QUISQUE ERAT dolor, GRAVIDA NEC ALIQUAM EU, ORNARE SIT AMET odio.

**DAPIBUS PORTTITOR PLACERAT**

IN TELLUS MASSA, DAPIBUS PORTTITOR PLACERAT AT, PRETIUM EU VELIT. AENEAN IN SEM AC odio TEMPUS PORTTITOR EU ET ANTE. IN PORTA MI ET FACILISIS BIBENDUM.

**LOREM IPSUM DOLOR SIT AMET**

LoREM iPSUM dOLOR sIT aMET, cONSECTETUR adIPISCING elIT. ViVAMUS VOLUPTAT MATTIS sagITTIS. MORBI sed SOLlicitUDIN nISL. IN non nISL nISI. IN tINCIDUNT ViVERRA ViVERRA.

**IN PELLentesque MOLLIS FEUGIAT**

IN pELLentesque MOLLIS FEUGIAT. CRAS IN BLANDIT dolor. MAURIS SEMPER FELIS AUGUE, AC pELLentesque LECTUS PuLVINAR EU. QUISQUE ERAT dolor, GRAVIDA NEC ALIQUAM EU, ORNARE SIT AMET odio.

**DAPIBUS PORTTITOR PLACERAT**

IN TELLUS MASSA, DAPIBUS PORTTITOR PLACERAT AT, PRETIUM EU VELIT. AENEAN IN SEM AC odio TEMPUS PORTTITOR EU ET ANTE. IN PORTA MI ET FACILISIS BIBENDUM.

# Principios estéticos del diseño web: Alineación

- Ningún elemento de la página debe estar puesto arbitrariamente.
- Organiza el contenido bajo un formato específico que facilite la lectura de la página.
- Mantener las proporciones entre los márgenes y el espacio de trabajo utilizado.



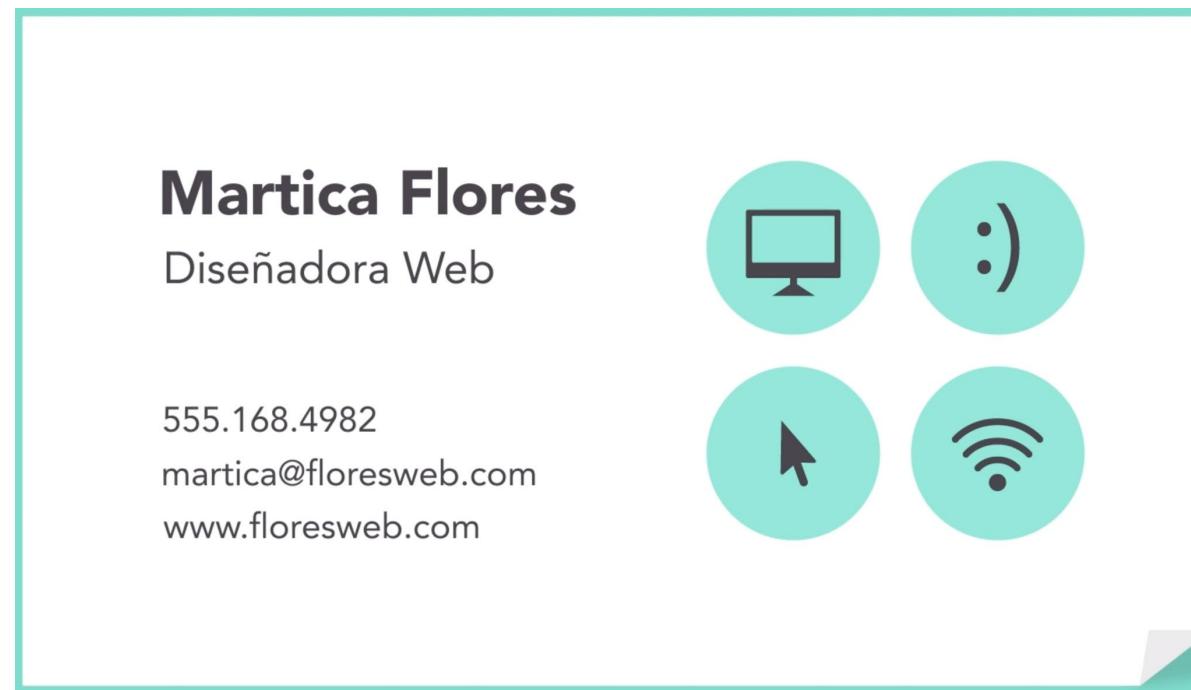
# Principios estéticos del diseño web: Repetición

- Repetir elementos de diseño para proporcionar uniformidad: colores, formas, texturas, fuentes, tamaños, conceptos gráficos, etc.
- Proporciona aspecto único, profesional y consistente permitiendo a los usuarios familiarizarse con la web.

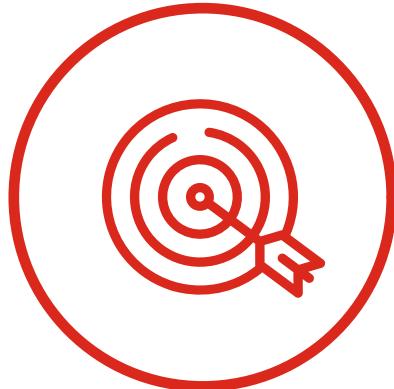


# Principios estéticos del diseño web: **Proximidad**

- Organización de los elementos de una página.
- Agrupar los elementos relacionados entre sí.
- Permite organizar la información, reduce el desorden y da al usuario un sentido de estructura.



# Principios de diseño web



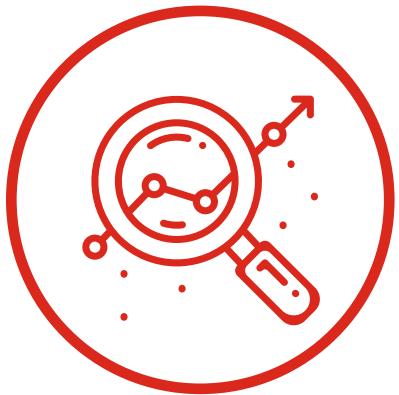
.....

1. Diseño orientado al **usuario (User Centered Design)**: todo el proceso conducido por el usuario, diseño alineado a sus necesidades, características y objetivos.
2. Diseño orientado a **objetivos**: definir y planificar los objetivos que se deseen alcanzar con el desarrollo de la web.
3. Diseño orientado a **implementación**: centrado en las posibilidades tecnológicas que están a la disposición del diseñador y que este es capaz de implementar.



# ¿Qué es **user-centered** design?

## Investigación



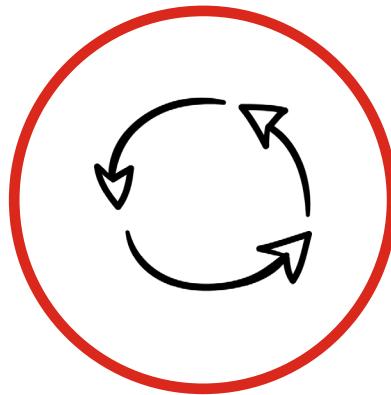
Comienza con un análisis de los usuarios, sus objetivos y contexto

## Empatía



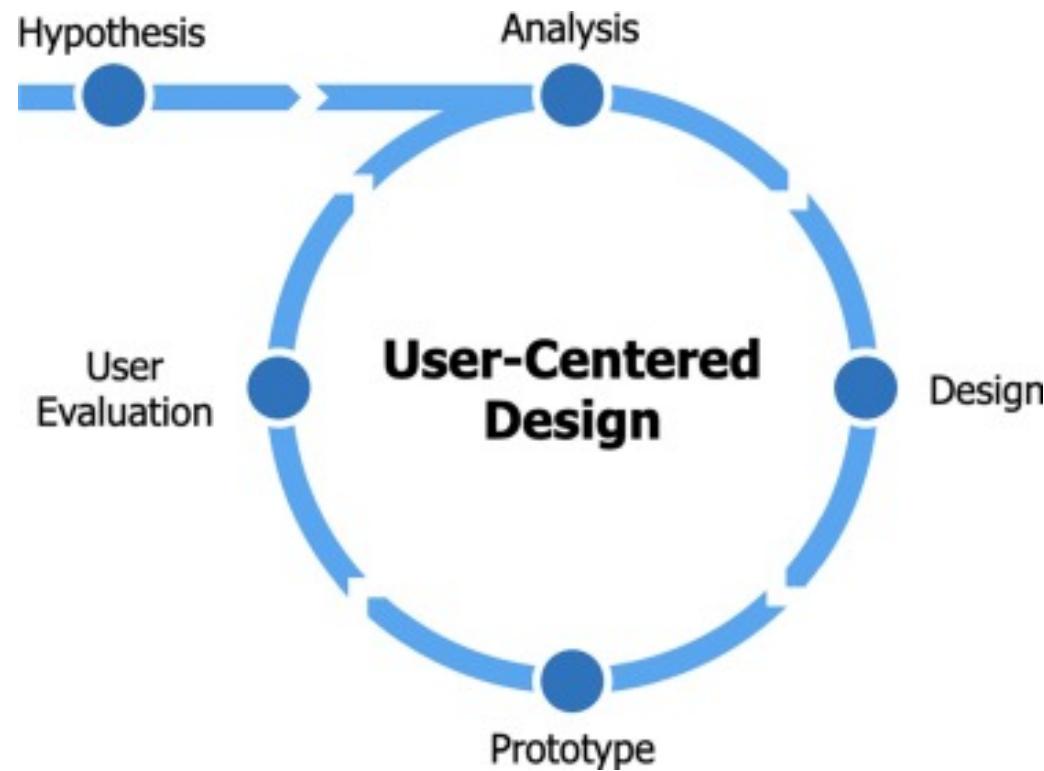
Ponerse en el lugar de los usuarios y entender sus necesidades

## Iteración

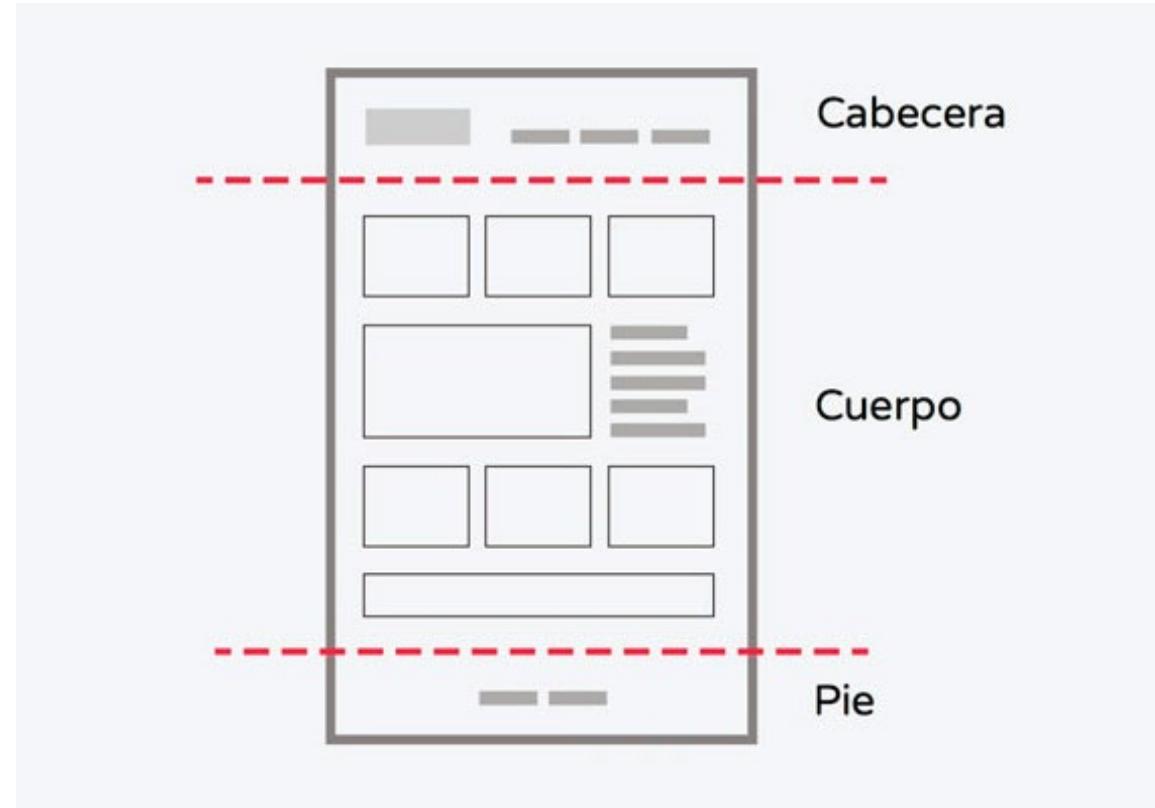


Proceso de iteración que permita la evaluación y la mejora de forma constante

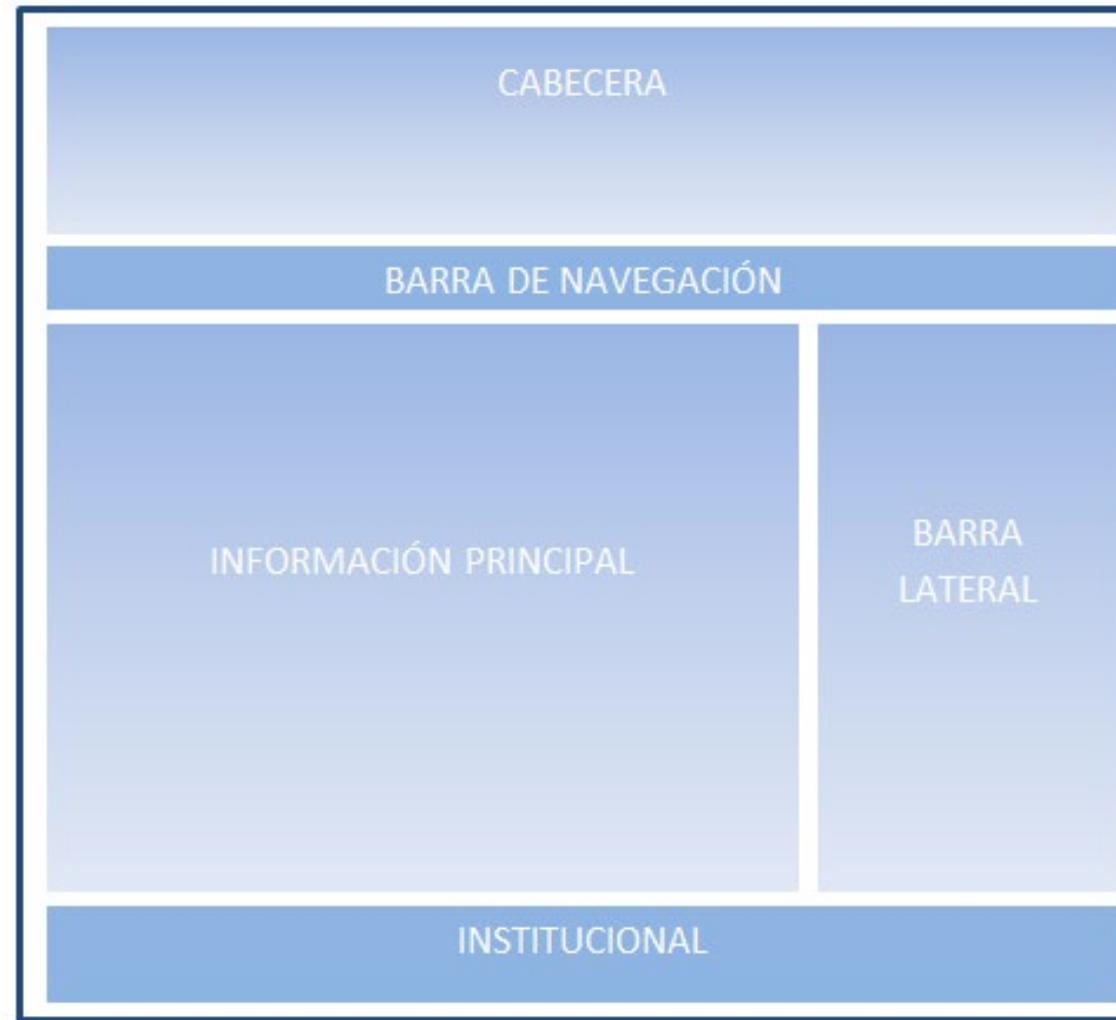
# User-centered design



# Proceso de diseño web: **Estructura** de una página web



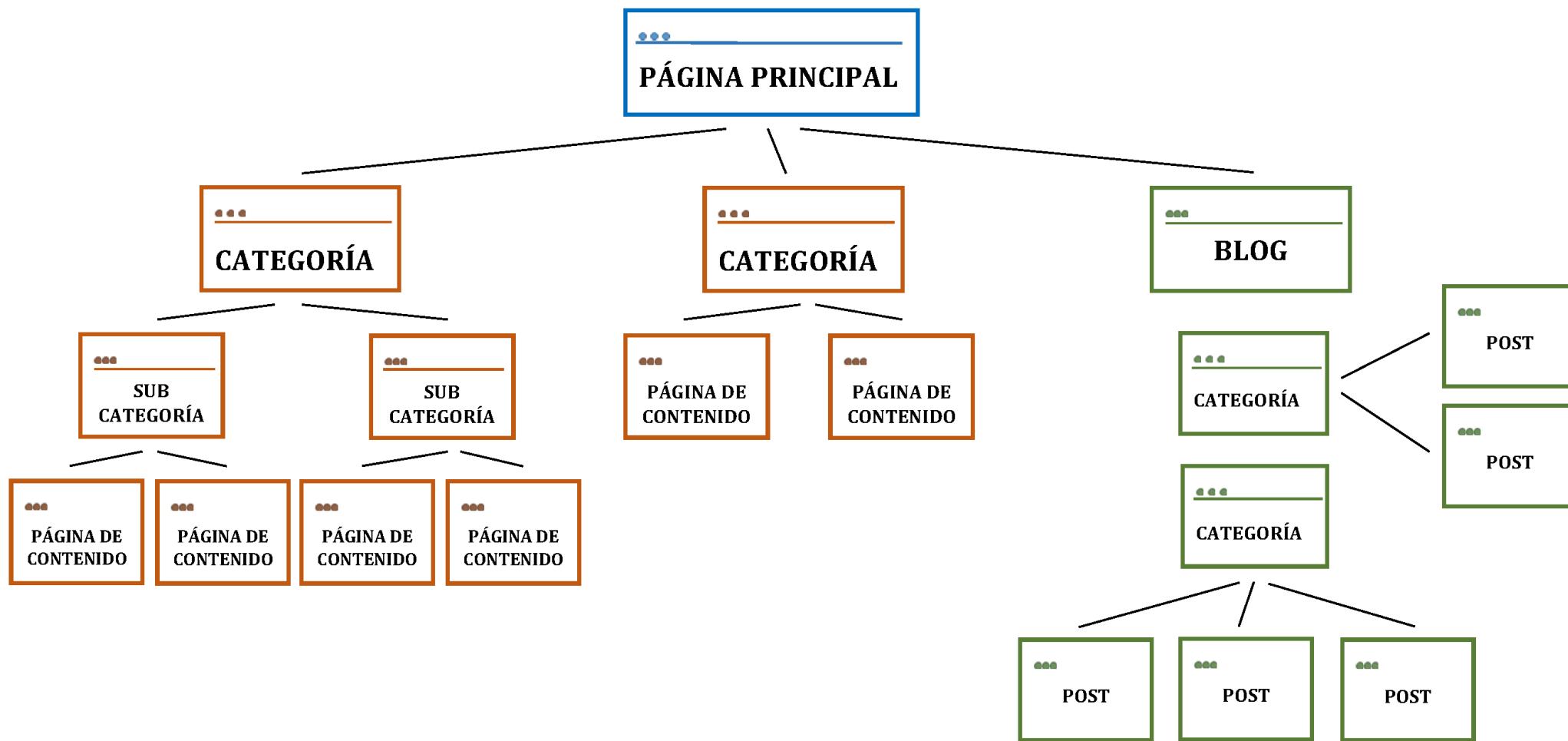
# Proceso de diseño web: **Organización y elementos** de la página



# Proceso de diseño web: Organización y elementos de la página (II)

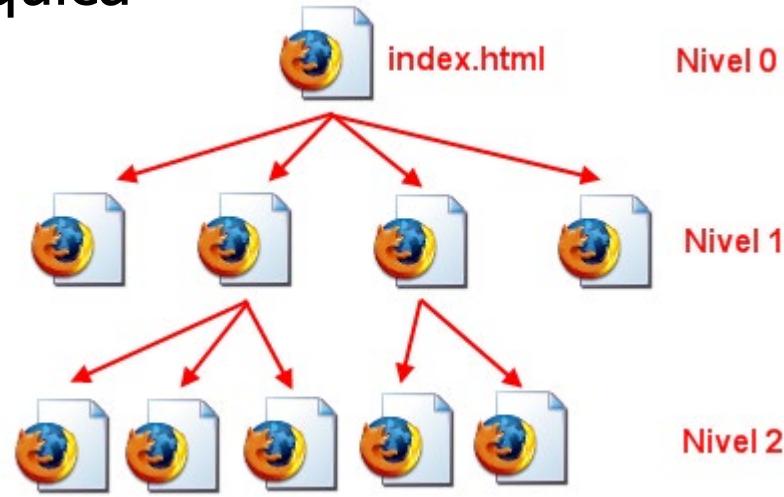


# Proceso de diseño web: Jerarquía y composición de páginas

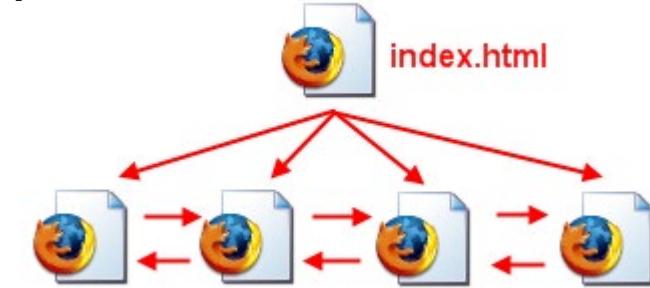


# Proceso de diseño web: Estructura de Navegación

Jerárquica



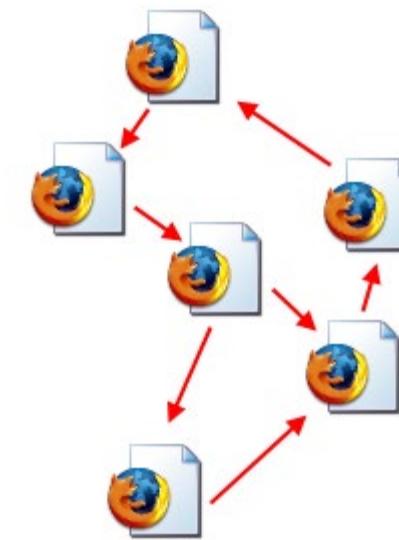
Lineal jerárquica



Lineal



Red



# Proceso de diseño web: **Navegabilidad web**

- Enfocado a mejorar la **experiencia del usuario (UX)**.
- Debe de ser **sencillo e intuitivo** localizar toda la información y los recursos disponibles, así como identificar las rutas y los atajos de navegación.
- Tiene en cuenta la organización y categorización del contenido de la web.
- El usuario debe poder saber:
  - En qué parte del sitio web se encuentra
  - De qué parte de la web viene
  - Cómo llegar a la información a la que quiere ir
- Cuando no hay buena navegabilidad:
  - Los usuarios tienden a buscar lo que necesitan en otro sitio
  - Aumenta la tasa de rebote
  - Afecta negativamente a los objetivos SEO

# Proceso de diseño web: **Navegabilidad web**

Cómo lograr una buena navegabilidad:

- **Menú de navegación visible** desde cualquier página del sitio y siempre en el **mismo lugar**.
- Incluir **breadcrumbs** que permitan al usuario identificar en qué parte del sitio se encuentra. Por ejemplo: Inicio > Equipaje > Mochilas > Mochilas de deporte.
- **URLs** organizadas por categorías y niveles jerárquicos.
- Incluir **mapa del sitio** que muestre todas las páginas y su organización.
- **Secciones** adecuadamente **categorizadas y jerarquizadas**.
- Las **rutas más utilizadas** sean más **accesibles** y **visibles**, e incluso tengan **atajos** especiales.
- Tener en cuenta diseño **responsive**.





# DINÁMICA

---

Diseño Responsive

## OBJETIVO

Familiarizarse con el diseño responsive mediante un caso real.

## INSTRUCCIONES

1. Abre una pestaña nueva del navegador y visita la siguiente URL:  
<https://www.workana.com/i/glosario/que-es-el-diseno-responsive/>
2. Modifica el tamaño de la ventana del navegador: hazla más estrecha, más bajita, más pequeña y más grande. Apreciarás cómo los elementos se ajustan de manera automática y la página se sigue viendo bien a pesar de los cambios de tamaño.
3. Probar en distintos navegadores y versiones.
4. Comentar lo observado con tu compañero/a.



10 min

# Proceso de diseño web: **Navegadores y Compatibilidad**

Que una web sea **compatible con todos los navegadores** significa **que se vea igual** (o muy similar) en todos (Chrome, Edge/IE, Firefox, Safari, Opera).

Cómo **mejorar la compatibilidad** con navegadores:

- Validar el código de tu web en base a los estándares (W3C Markup Validation Service)
- Resetear los estilos CSS
- Usar técnicas y componentes soportados por los principales navegadores
- Trabajar con navegadores que respeten los estándares web, seguros y con velocidad óptima. Buena opción por defecto: Chrome.

# Proceso de diseño web: **Herramientas online**

<b>HTML5 Test</b>	Verifica soporte para elementos HTML5 en un navegador concreto <a href="https://html5test.com">html5test.com</a>
<b>W3C Validator</b>	Markup Validation Service <a href="https://validator.w3.org">https://validator.w3.org</a>
<b>Modernizr</b>	Detección de funciones HTML5, CSS3 y JS <a href="https://modernizr.com">https://modernizr.com</a>
<b>Pollyfill</b>	Código JavaScript en la web que proporciona una funcionalidad moderna en navegadores antiguos que no lo admiten de forma nativa.
<b>Codepen</b> (SaaS)	Editor HTML, CSS y Javascript online <a href="https://codepen.io/">https://codepen.io/</a>
<b>Media Query</b>	Módulo CSS3 para adaptar la representación del contenido al dispositivo
<b>CSS3 Test</b>	Verifica soporte para elementos CSS3 en un navegador concreto <a href="https://css3test.com/">https://css3test.com/</a>



# ACTIVIDAD

Compatibilidad con navegadores

## OBJETIVO

Utilizar herramientas para comprobar la compatibilidad de HTML5 y CSS3 con los principales navegadores del mercado.

## INSTRUCCIONES

1. Probar los enlaces [www.html5test.com](http://www.html5test.com) y [www.css3test.com](http://www.css3test.com) en varios navegadores. Comparar los resultados obtenidos con el grupo.



15 min

01

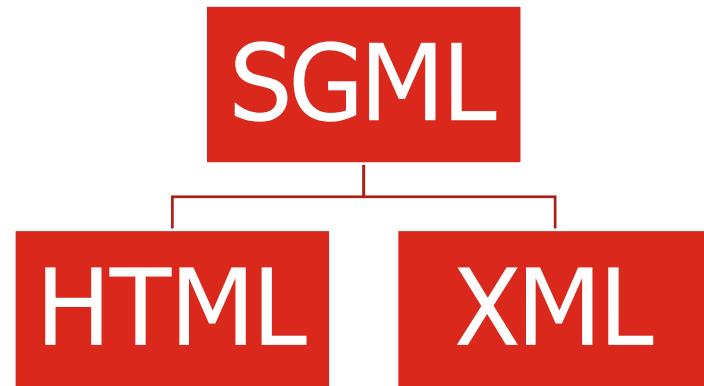
# Elaboración de documentos web mediante lenguajes de marca

## 2. Lenguajes de marcado generales

- Origen de los lenguajes de marcado generales: SGML y XML.
- Características generales de los lenguajes de marcado.
- Estructura general de un documento con lenguaje de marcado (Metadatos e instrucciones de proceso, Codificación de caracteres, Caracteres especiales, Etiquetas o marcas, Elementos, Atributos, Comentarios).
- Documentos válidos y bien formados. Esquemas.

# Lenguajes de marcado generales

Origen y características de los lenguajes de marcado generales



- Un lenguaje de marcado (**markup language**) o lenguaje de marcas es una forma de codificar un documento mediante texto plano (UTF-8, ASCII...)
- Basado en **etiquetas o marcas** que especifican la estructura del texto y su presentación.
- Lenguaje interpretado por los navegadores.
- Independiente del dispositivo.
- No es un lenguaje de programación pero puede contener bloques de código de un lenguaje de programación.
- Principales lenguajes de marcado: SGML, HTML, XHTML, XML, MathML, TeX, LaTeX.

# Lenguajes de marcado generales

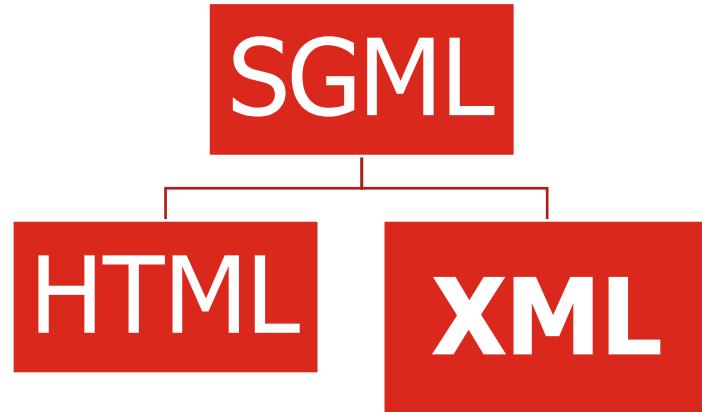
SGML - Standard Generalized Markup Language



- **SGML** es un sistema para la **organización y etiquetado/marcado de documentos** normalizado por la Organización Internacional de Estándares (ISO).
- Sirve para **especificar las reglas de etiquetado** de documentos y no impone ningún conjunto de etiquetas en concreto.
- HTML está basado en SGML.
- XML es un estándar de creación posterior a SGML más sencillo con una funcionalidad similar.

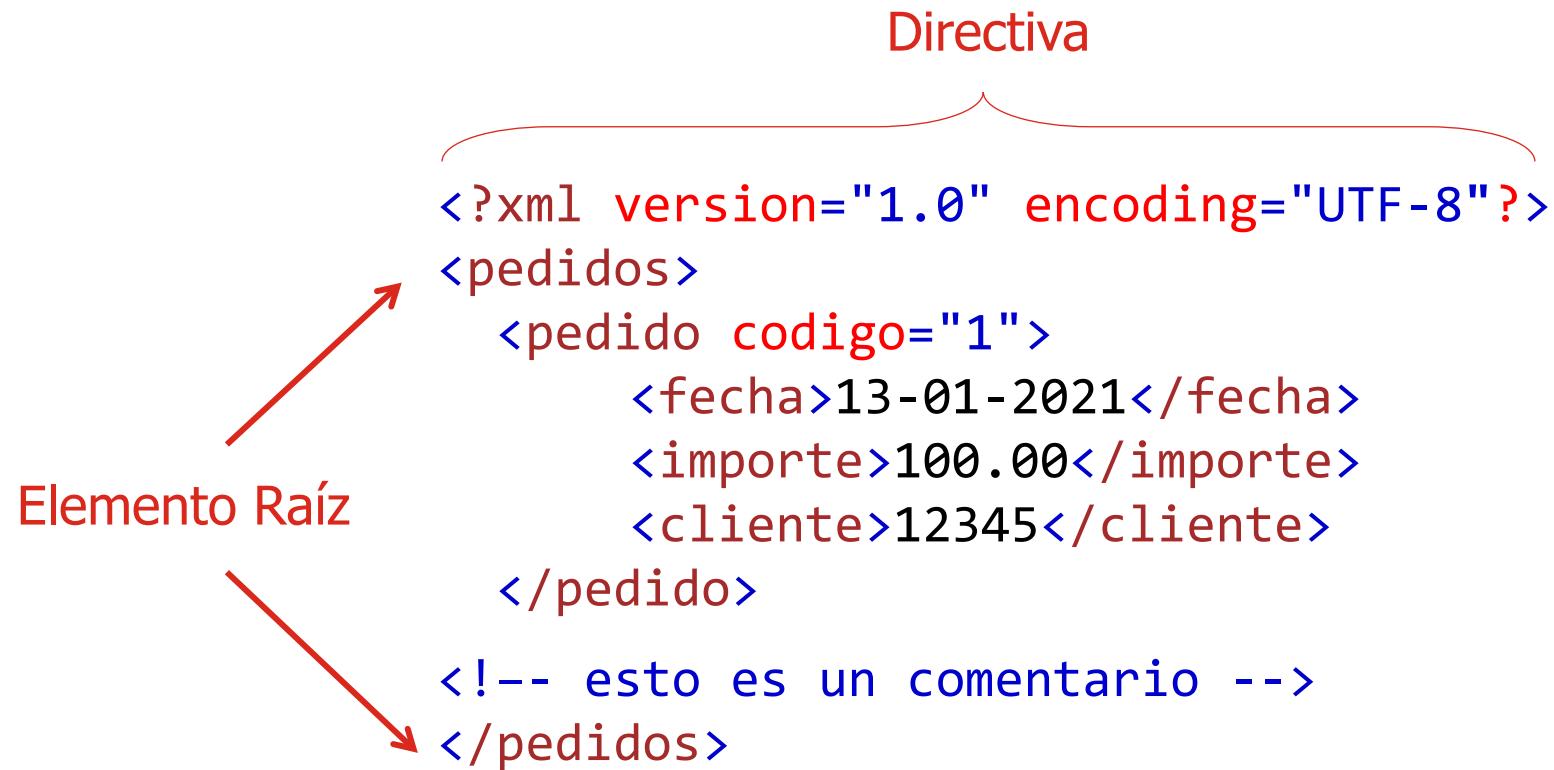
# Lenguajes de marcado generales

XML - eXtensible Markup Language

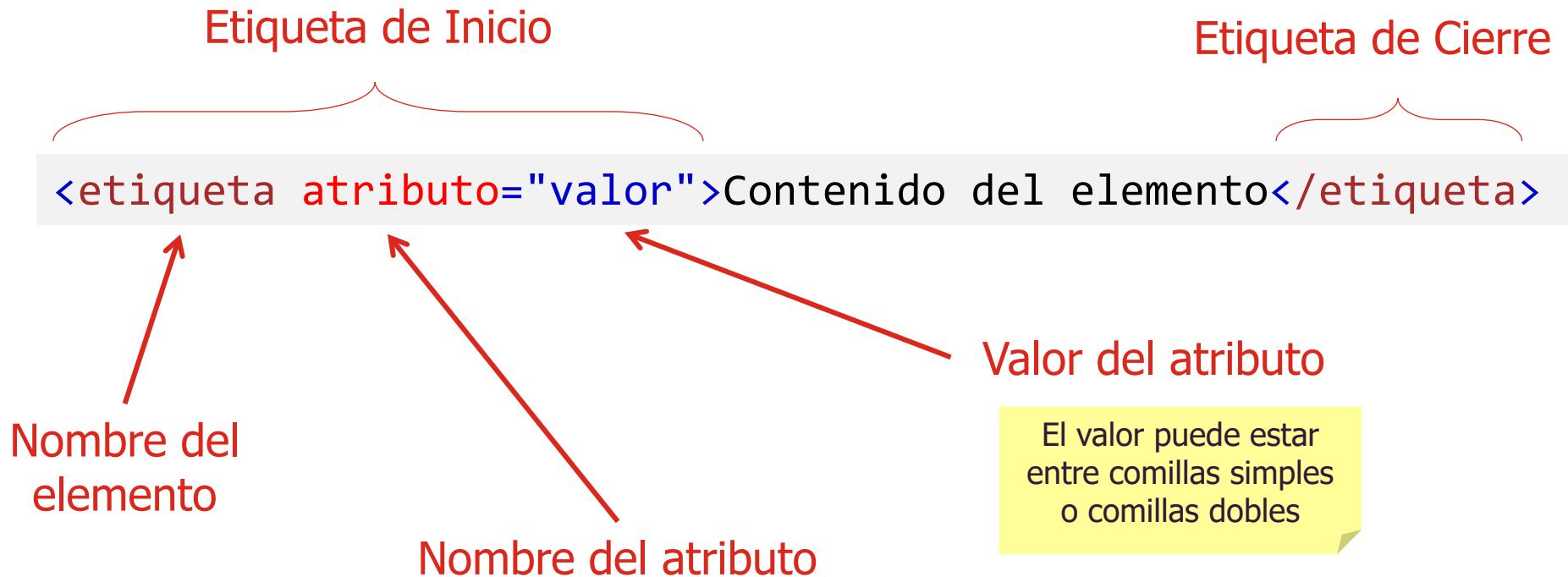


- Versión simplificada de SGML desarrollada por el World Wide Web Consortium (W3C).
- La estructura se puede entender fácilmente y permite diferenciar las partes de un documento.
- Permite añadir nuevas etiquetas.
- Estándar utilizado para el **intercambio de información** entre aplicaciones independientemente de la plataforma.
- Permite crear páginas web **más semánticas**.
- Archivo con extensión **.xml**
- Formato actualmente sustituido por **JSON**

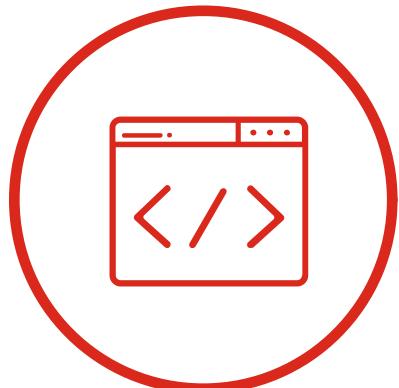
# Estructura general de un documento XML



# Elementos XML



# Sintaxis XML



- 1. Cada elemento tiene dos etiquetas: la de **inicio/apertura** y la de **final/cierre**.
- 2. **Único elemento raíz** y dentro de este cero o mas elementos correctamente **anidados**.
- 3. Los nombres de etiquetas son **case sensitive**.
- 4. Los nombres de los elementos deben cumplir con ciertas reglas.
- 5. Los elementos pueden contener **atributos** y sus valores entre comillas de manera obligatoria.
- 6. Los espacios en blanco se preservan.
- 7. **Comentarios** entre símbolos `<!--` y `-->`

# Ejemplo de código XML

```
<biblioteca>
  <libro categoria="infantil">
    <titulo>Harry Potter</titulo>
    <autor>J K. Rowling</autor>
    <año>2005</año>
    <precio>29.99</precio>
  </libro>
  <libro categoria="web">
    <titulo>Learning XML</titulo>
    <autor>Erik T. Ray</autor>
    <año>2003</año>
    <precio>39.95</precio>
  </libro>
</biblioteca>
```

# Documentos válidos y bien formados

- Un documento XML **bien formado** debe empezar con la directiva o línea de código `<?xml version="1.0"?>`
- Un documento XML **valido** es un documento bien formado y además debe tener una forma de validar las reglas para cada elemento XML a través de archivos DTD, XSD o esquemas XML.

# Lenguajes de Presentación

- Permiten **definir las características de visualización o apariencia** de la información contenida en el documento.
- Los lenguajes de marcado pretenden especificar sólo el contenido del documento.
- Algunos lenguajes de presentación:
  - **CSS** o **Cascading Style Sheets** define la apariencia de elementos HTML o XML que se muestran en el navegador web.
  - **XSL** o **Extensible Stylesheet Language** especifica la presentación de documentos XML.



# ACTIVIDAD

Conversión XML a JSON

## OBJETIVO

Interpretar correctamente código XML y hacer la conversión al formato de intercambio de datos JSON.

## INSTRUCCIONES

1. El formato JSON o JavaScript Object Notation es uno de los sistemas preferidos actualmente para el intercambio de datos en sustitución del formato XML. JSON es fácil de leer y escribir para las personas y fácil de parsear para las máquinas.
2. Tomando como referencia el ejemplo siguiente realiza la conversión del bloque de código XML 'Biblioteca' a su respectivo formato JSON con las dos opciones del ejemplo dado.



20 min



```
<users>
  <user>
    <id>1</id>
    <name>Ana</name>
    <email>ana@gmail.com</email>
  </user>
  <user>
    <id>2</id>
    <name>Jorge</name>
    <email>jorge@gmail.com</email>
  </user>
</users>
```

{  
**JSON**}

```
{
  "users": [
    "user": [
      {
        "id": "1",
        "name": "Ana",
        "email": "ana@gmail.com"
      },
      {
        "id": "2",
        "name": "Jorge",
        "email": "jorge@gmail.com"
      }
    ]
  }
}
```



20 min



```
<users>
  <user>
    <id>1</id>
    <name>Ana</name>
    <email>ana@gmail.com</email>
  </user>
  <user>
    <id>2</id>
    <name>Jorge</name>
    <email>jorge@gmail.com</email>
  </user>
</users>
```

{  
**JSON**}

```
[  
  {  
    "id": "1",  
    "name": "Ana",  
    "email": "ana@gmail.com"  
  },  
  {  
    "id": "2",  
    "name": "Jorge",  
    "email": "jorge@gmail.com"  
  }]  
]
```



20 min

```
<biblioteca>
  <libro categoria="infantil">
    <titulo>Harry Potter</titulo>
    <autor>J. K. Rowling</autor>
    <año>2005</año>
    <precio>29.99</precio>
  </libro>
  <libro categoria="web">
    <titulo>Learning XML</titulo>
    <autor>Erik T. Ray</autor>
    <año>2003</año>
    <precio>39.95</precio>
  </libro>
</biblioteca>
```



20 min

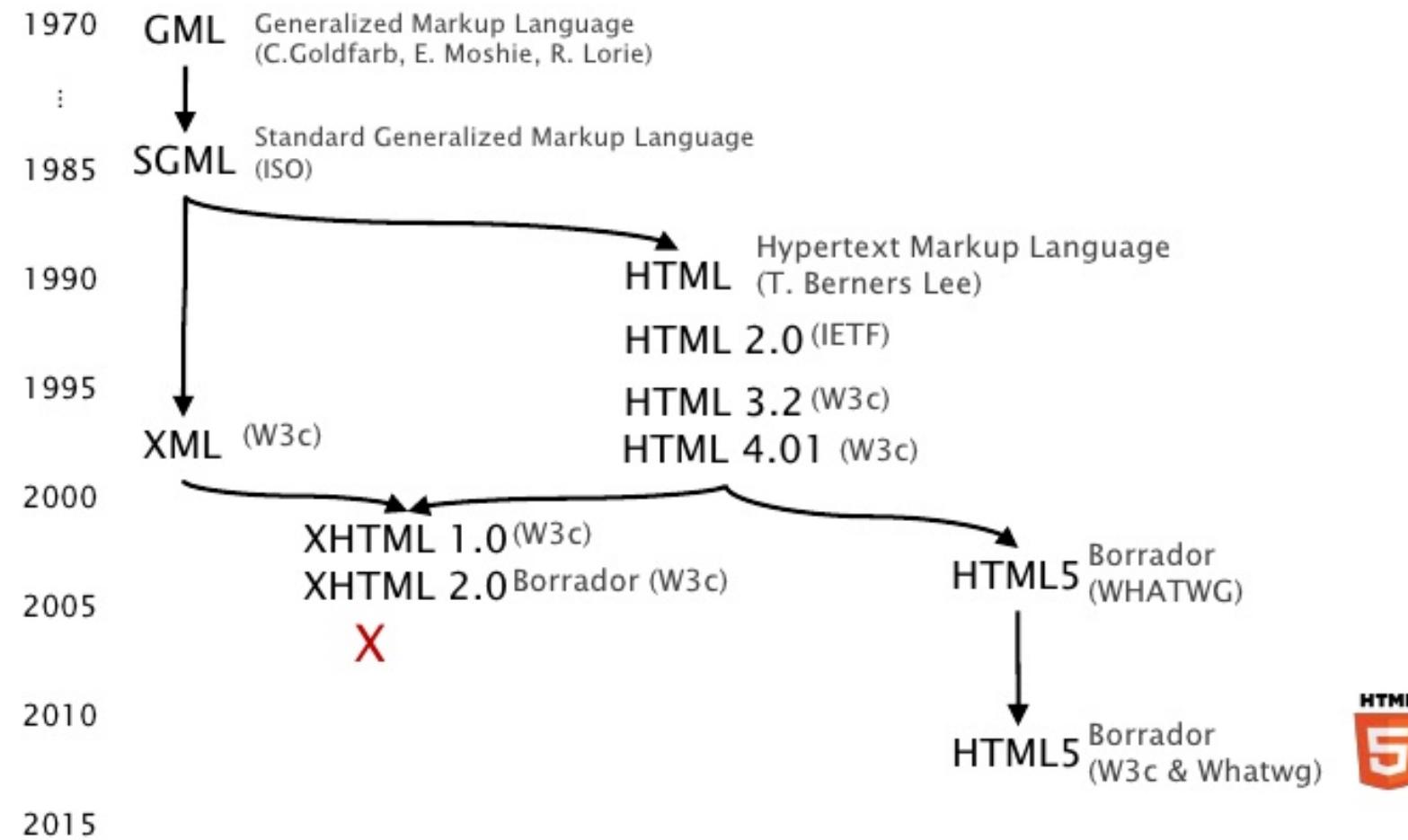
# 01

# Elaboración de documentos web mediante lenguajes de marca

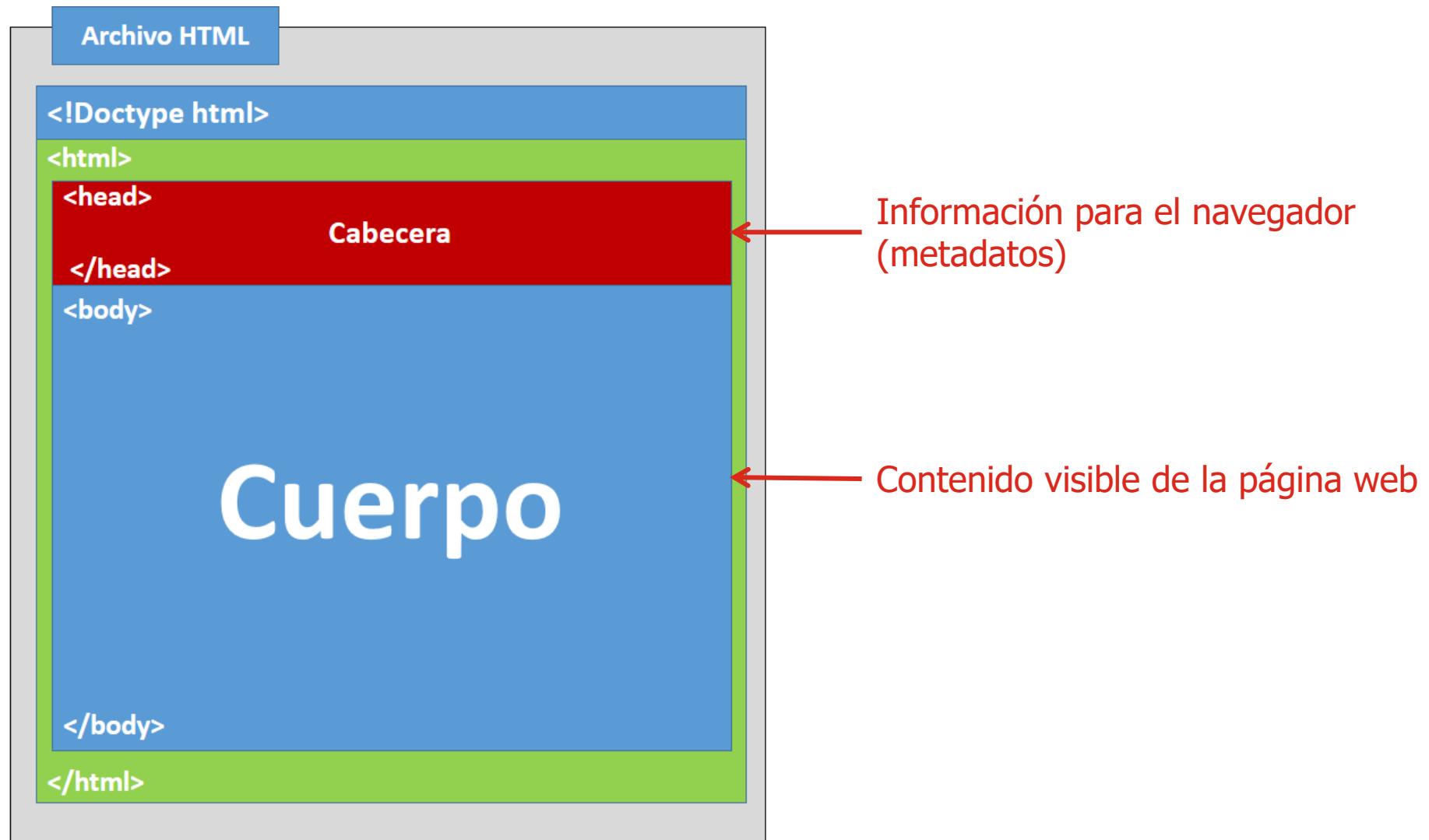
## 3. Lenguajes de marcado para presentación de páginas web

- Evolución de HTML
- Estructura de un documento HTML
- Color
- Texto
- Hipervínculos
- Imágenes
- Listas
- Tablas
- Formularios
- Frames
- Elementos específicos para tecnologías móviles
- Elementos en desuso (deprecated)
- HTML 5

# Evolución de HTML



# Estructura de un documento HTML



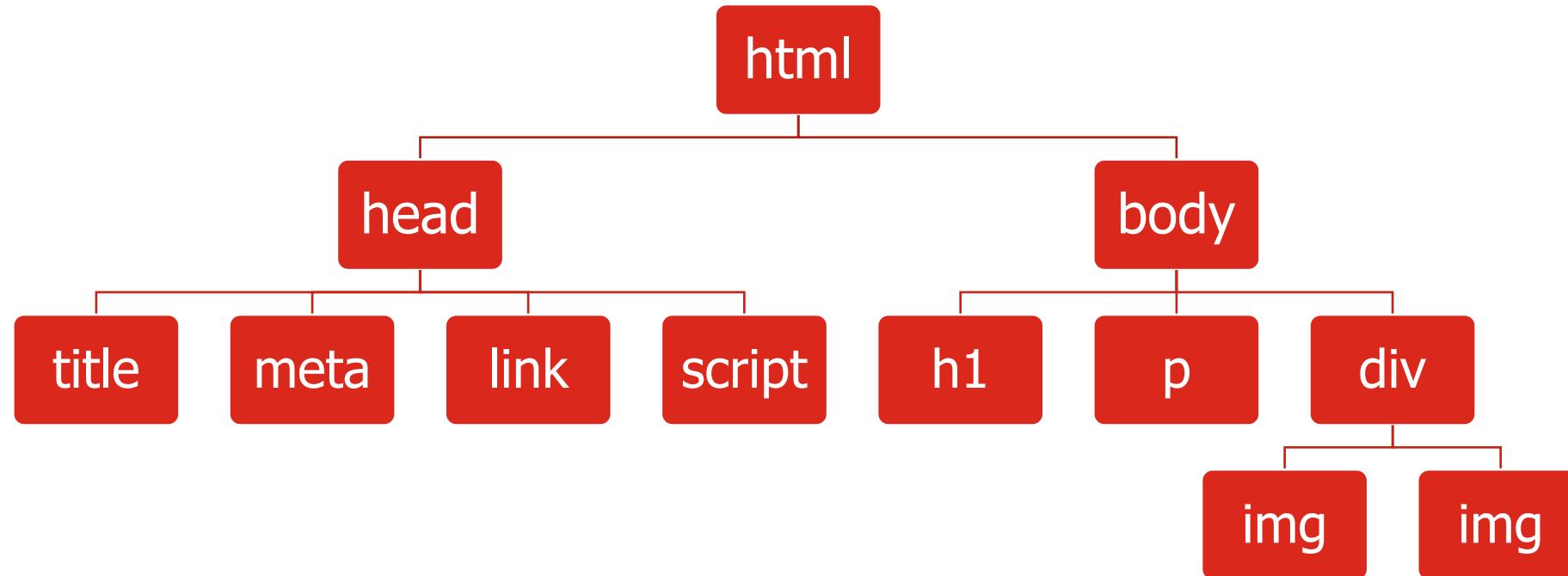
# Estructura de un documento HTML (II)

```
<!DOCTYPE html>           Especifica el tipo de documento (HTML 5)
<html lang="es">          Idioma predeterminado del texto de la página
  <head>
    <title>HTML Page Structure</title>   Título del documento que se muestra en la
    Define el                                     pestaña del navegador
    juego de
    caracteres
    <meta charset="utf-8">
    <link rel="stylesheet" href="styles.css">  Vinculo a hoja
    <script src="sample.js"></script>        de estilo externa
  </head>
  <body>
    <h1>My home page</h1>
    <p>Welcome to my Home Page!</p>
  </body>
</html>
```

Aquí van las etiquetas correspondientes al contenido que se mostrará en la página web

# Representación Jerárquica de un documento HTML

- Otra forma de representar la estructura de una página web es mediante un árbol en donde cada nodo representa un elemento o etiqueta HTML.



# Sintaxis de un elemento HTML

- HTML define su contenido a través de **etiquetas o tags**, que son los **elementos** del documento. La sintaxis de una etiqueta depende de si tiene contenido o no.
- En el caso de las que tienen contenido, la sintaxis es:

```
<nombre-elemento [atributo="valor"]>Contenido</nombre-elemento>
```

Mismo nombre al abrir y cerrar etiqueta

- En el caso de las que no tienen contenido, la sintaxis es:

```
1) <nombre-elemento [atributo="valor"]></nombre-elemento>
2) <nombre-elemento [atributo="valor"]>
```

- Los **atributos** son utilizados para especificar propiedades del contenido.

# Sintaxis HTML: Comentarios

- La forma de insertar comentarios en HTML es:

```
<!-- bloque de comentarios -->
```

- Permiten documentar y depurar el código fuente HTML.
- No se muestran en la página web.
- El bloque comentado puede tener una o más líneas y cualquier contenido.

```
<!--  
    Inicio sección de datos  
-->  
<h1> ... </h1>
```

# Elementos de <head>

# Elementos HTML básicos: head

- La etiqueta `<head>` (cabecera) define información del documento que no se muestra directamente al usuario. La sección head puede contener la siguiente información:
  - El título que se muestra en la barra o pestaña del navegador
  - Metadatos
  - Vínculos a hojas de estilo
  - Vínculos a scripts

```
<!DOCTYPE html>

<html>
  <head>
    </head>
  <body>
    </body>
</html>
```

# Elementos sección <head>: title

- La etiqueta <**title**> define el **título del documento** y se muestra en la barra de título del navegador o en la pestaña de la página.

```
<head>
  <title>Programas formación</title>
</head>
```

- Esta etiqueta es **obligatoria** en documentos HTML. Debe ser solo texto y proporciona un título para la página cuando se agrega a favoritos.
- Muestra el título en la página de resultados del motor de búsqueda. El contenido del título de una página es muy importante para la optimización de motores de búsqueda (SEO). Los algoritmos de los motores de búsqueda utilizan el título de la página para determinar el orden de aparición en las páginas de resultados.

# Elementos sección <head>: meta

- La etiqueta `<meta>` (metadatos) permite especificar información general del documento HTML.
- En general se utiliza para especificar el conjunto de caracteres:

```
<meta charset="utf-8">
```

- Para especificar el autor y la descripción del documento:

```
<meta name="author" content="Netmind">
<meta name="description" content="Detalle de programas">
```

- Configuración de la ventana gráfica para que el documento se ajuste a todos los dispositivos (diseño responsive):

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

# Elementos sección <head>: meta

- Información sobre actualización de la página, que permite que recargue otra página (o la misma) después de cierto tiempo:

```
<meta http-equiv="refresh" content="30">
```

Actualiza el documento  
cada 30 segundos.

- Definir palabras clave para los motores de búsqueda:

```
<meta name="keywords" content="HTML, CSS, JavaScript">
```

# Elementos sección <head>: Enlace

- La etiqueta **<link>** define la relación entre dos documentos vinculados. Se utiliza para enlazar páginas de estilo externas al documento HTML. Es un elemento vacío, solo contiene atributos.

```
<!-- HTML4 -->  
  
<link type="text/css" rel="stylesheet" href="styles.css">  
  
<!-- HTML5 -->  
  
<link rel="stylesheet" href="styles.css">
```

- Atributos:
- **rel**: Define la relación entre el documento enlazado con el actual. Valores: `stylesheet` para hojas de estilo CSS, y otros como `alternate`, `bookmark`, `chapter`, `help` y `glossary`.
- **type**: Especifica el tipo MIME (media type) del documento vinculado. Se utiliza `text/css` para hojas de estilo.
- **href**: La URL del documento vinculado.

# Elementos sección <head>: Script

- La etiqueta **<script>** permite vincular el documento con un script del lado del cliente (archivo Javascript).

```
<!-- HTML4 -->
<script type="text/javascript" src="javascript.js"></script>
```

```
<!-- HTML5 -->
<script src="javascript.js"></script>
```

- Atributos:
- **type**: Especifica el tipo de contenido del bloque de código, siendo `text/javascript` su valor por defecto.
- **src**: La URL del script a ser cargado.

# Elementos de <b>body</b>

# Elementos HTML básicos: Encabezado

- Se utilizan los tags `<h1>` a `<h6>` para insertar títulos o subtítulos (headings) dependiendo de la importancia .

```
<h1>Encabezado 1</h1>
<h2>Encabezado 2</h2>
<h3>Encabezado 3</h3>
<h4>Encabezado 4</h4>
<h5>Encabezado 5</h5>
<h6>Encabezado 6</h6>
```

- Son de tipo **block**, pues después de cada uno se agrega un salto de línea.

**Encabezado 1**

**Encabezado 2**

**Encabezado 3**

**Encabezado 4**

**Encabezado 5**

**Encabezado 6**

# Elementos HTML básicos: Párrafo

- Para especificar un párrafo, se utiliza la etiqueta **< p >**:

```
<h2>HTML, CSS y JavaScript</h2>  
  
<p>El contenido incluye  
HTML, CSS y JavaScript</p>  
<p>Primero se describe el  
lenguaje HTML</p>
```

## HTML, CSS y JavaScript

El contenido incluye HTML, CSS y JavaScript

Primero se describe el lenguaje HTML

El espacio entre párrafos  
es doble.

No se aplica salto de línea  
del código fuente.

- Es de tipo **block**, después de cada párrafo se añade un salto de línea.

# Elementos HTML básicos: Párrafo

- Atributo **align** para especificar la alineación de un párrafo:

```
<h3>Detalles</h3>
```

```
<p align="justify">El atributo  
align del párrafo define su  
alineación. Las opciones  
principales son left, center,  
right y justify (horizontal),  
y existen otras como top, middle  
y bottom para la alineación  
vertical.</p>
```

## Detalles

El atributo **align** del párrafo define su alineación. Las opciones principales son left, center, right y justify (horizontal), y existen otras como top, middle y bottom para la alineación vertical.

# Elementos HTML básicos: Salto de línea

El salto de línea del código fuente no tiene efecto sobre el HTML. Para insertar un salto de línea (break) se utiliza `<br>`:

```
<p>Las opciones de alineación horizontal son:<br>
left<br>center<br>right
</p>
```

Las opciones de alineación horizontal son:  
left  
center  
right

Saltos de línea

No se aplica salto de línea  
del código fuente HTML.

# Elementos HTML básicos: Formato

A un texto se le puede cambiar el aspecto, utilizando:

- **negrita:**

Conocer HTML es  
**muy importante**.

Conocer HTML es **muy importante**.

- *cursiva:*

El atributo *align* especifica alineación.

El atributo *align* especifica alineación.

- *énfasis:*

Este es el *contenido básico*.

Este es el *contenido básico*.

# Elementos HTML básicos: Imagen

- Añadir una imagen mediante la etiqueta **<img>**:

```

```

fuente: URL o ruta (relativa o absoluta) de la imagen.

# Elementos HTML básicos: Imagen

- Se puede especificar el **alto** y **ancho** de una imagen, la que se ajusta:

```

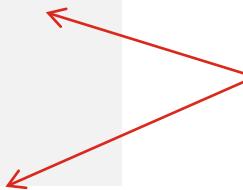
```

```

```

```

```



Si se omite uno de los dos atributos, se mantiene la relación de aspecto.

# Elementos HTML básicos: Imagen

## Imagen, atributos

- Una imagen puede tener una representación alternativa en caso que no logre cargarse, con el atributo **alt**:

```

```

- Si el archivo de la imagen no se encuentra, se ve lo siguiente en función del navegador:



tecnología



# Elementos HTML básicos: Imagen

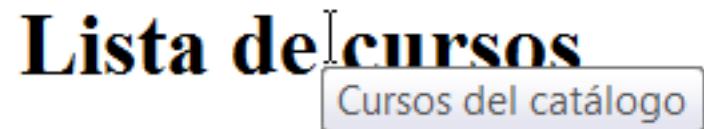
- En general, los elementos HTML tienen un atributo llamado **title**, que despliega un **tooltip** (texto flotante) cuando se pasa el puntero del mouse sobre éste.
- Ejemplos:

```

```



```
<h1 title="Cursos del catálogo">  
    Lista de cursos</h1>
```



# Formatos de imagen

Los navegadores soportan varios tipos de imagen:

- **GIF** (Graphics Interchange Format): Para imágenes de pocos colores, que no deben perder nitidez, como los logos, o textos en formato imagen. Soporta transparencia.
- **JPG** (de **JPEG**, Joint Photographic Experts Group): es un formato comprimido, que a cambio tiene una pérdida de calidad ajustable en función de cuánto se comprime. Es adecuado para fotografías grandes, donde la pérdida es poco perceptible. No soporta transparencia. Formato preferido para la web ya que ofrece un buen equilibrio entre calidad y tamaño (peso) de la imagen.
- **PNG** (Portable Network Graphics): Tiene tamaño intermedio entre GIF y PNG. Soporta transparencia. Permite utilizar más colores que GIF dependiendo del número de bits (PNG-8, 24 o 32). Se utiliza en los mismos casos que GIF, para imágenes que requieren nitidez como logos o textos en imágenes.

# Principales estilos: Lista

- Las listas HTML permiten agrupar elementos relacionados
- Dos tipos de listas: sin orden **<ul>** y ordenadas **<ol>**
- Cada elemento de la lista se define con la etiqueta **<li>**

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
  <li>Orange Juice</li>
</ul>
```

- Coffee
- Tea
- Milk
- Orange Juice

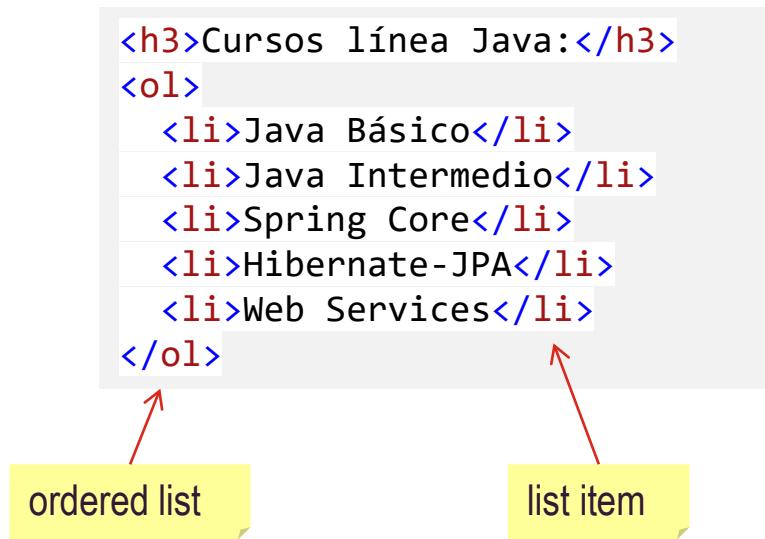
```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
  <li>Orange Juice</li>
</ol>
```

1. Coffee
2. Tea
3. Milk
4. Orange Juice

# Elementos HTML básicos: Lista

- Las listas con orden (opción por defecto) muestran un índice en cada ítem.
- Ejemplo de lista ordenada con números:

```
<h3>Cursos línea Java:</h3>
<ol>
  <li>Java Básico</li>
  <li>Java Intermedio</li>
  <li>Spring Core</li>
  <li>Hibernate-JPA</li>
  <li>Web Services</li>
</ol>
```



## Cursos línea Java:

1. Java Básico
2. Java Intermedio
3. Spring Core
4. Hibernate-JPA
5. Web Services

# Elementos HTML básicos: Lista ordenada

- Ordenada con **letras minúsculas**:

```
<ol type="a">
  <li>Java Básico</li>
  <li>Java Intermedio</li>
  <li>Spring Core</li>
  <li>Hibernate-JPA</li>
  <li>Web Services</li>
</ol>
```

- a. Java Básico
- b. Java Intermedio
- c. Spring Core
- d. Hibernate-JPA
- e. Web Services

- Ordenada con **números romanos** en mayúsculas:

```
<ol type="I">
  <li>Java Básico</li>
  <li>Java Intermedio</li>
  <li>Spring Core</li>
  <li>Hibernate-JPA</li>
  <li>Web Services</li>
</ol>
```

- I. Java Básico
- II. Java Intermedio
- III. Spring Core
- IV. Hibernate-JPA
- V. Web Services

Nota: el atributo type está en desuso.

# Elementos HTML básicos: Hipervínculo

- Permite vincular o la navegación entre documentos HTML:

```
<a href="catalogo.html">Ir a catálogo</a>
```

Ir a catálogo



- El principal atributo de <a> es **href** el cual indica el destino del enlace o hipervínculo.
- Un link, una vez visitado, por default cambia su aspecto gráfico: Ir a catálogo
- Para crear un link que abra el programa de email por defecto del usuario asignar el valor **mailto:** como valor del atributo **href**.

# Elementos HTML básicos: Hipervínculo

- Imagen como hipervínculo:

```
<a href="catalogo.html"></a>
```

Elemento `<img>` anidado dentro de `<a>`



# Elementos HTML básicos: Hipervínculo

- En el atributo **href** se puede incluir:
  - Una **URL** de una página de destino:

```
<a href="http://www.netmind.es">Netmind Home</a>
```

- Una ruta **relativa o absoluta**:

```
<a href="../pages/catalogo.html">Ir a catálogo</a>
```

```
<a href="C:/html/pages/catalogo.html">Ir a catálogo</a>
```

# Elementos HTML básicos: Hipervínculo

- En el atributo **target** permite definir como destino otra ventana o pestaña del navegador:

```
<a href="http://www.netmind.es" target="_blank">Netmind Home</a>
```

- Posibles valores del atributo `target`:
  - `_self`: Valor por defecto. Abre el documento en la misma ventana o pestaña
  - `_blank`: Abre el documento en una nueva ventana o pestaña
  - `_parent`: Abre el documento en el marco principal
  - `_top`: Abre el documento en el cuerpo de la ventana

# Elementos HTML básicos: Anchor

- La etiqueta `<a id="valor">` es un anchor (ancla), que permite definir posiciones de un documento para saltar a otra zona del mismo documento o a otro documento externo:

Sintaxis de un anchor: # seguido del id del anchor de destino

Anchor con id

```
<h1>Cursos Java:</h1>
<a href="#net">Ir a cursos .NET</a>
<ul>
  <li>Java Básico</li>
  <li>Java Intermedio</li>
  ...
</ul>
<a id="net"><h3>Cursos .NET:</h3></a>
<ul>
  <li>C# Básico</li>
  <li>C# Avanzado</li>
  ...
</ul>
<h1>Otros cursos:</h1>
...
```

# Elementos HTML básicos: **div**, **span**

- Una página puede ser dividida en secciones, las cuales pueden tener un tratamiento particular. Para ello, se utilizan las etiquetas **<div>** y **<span>**.

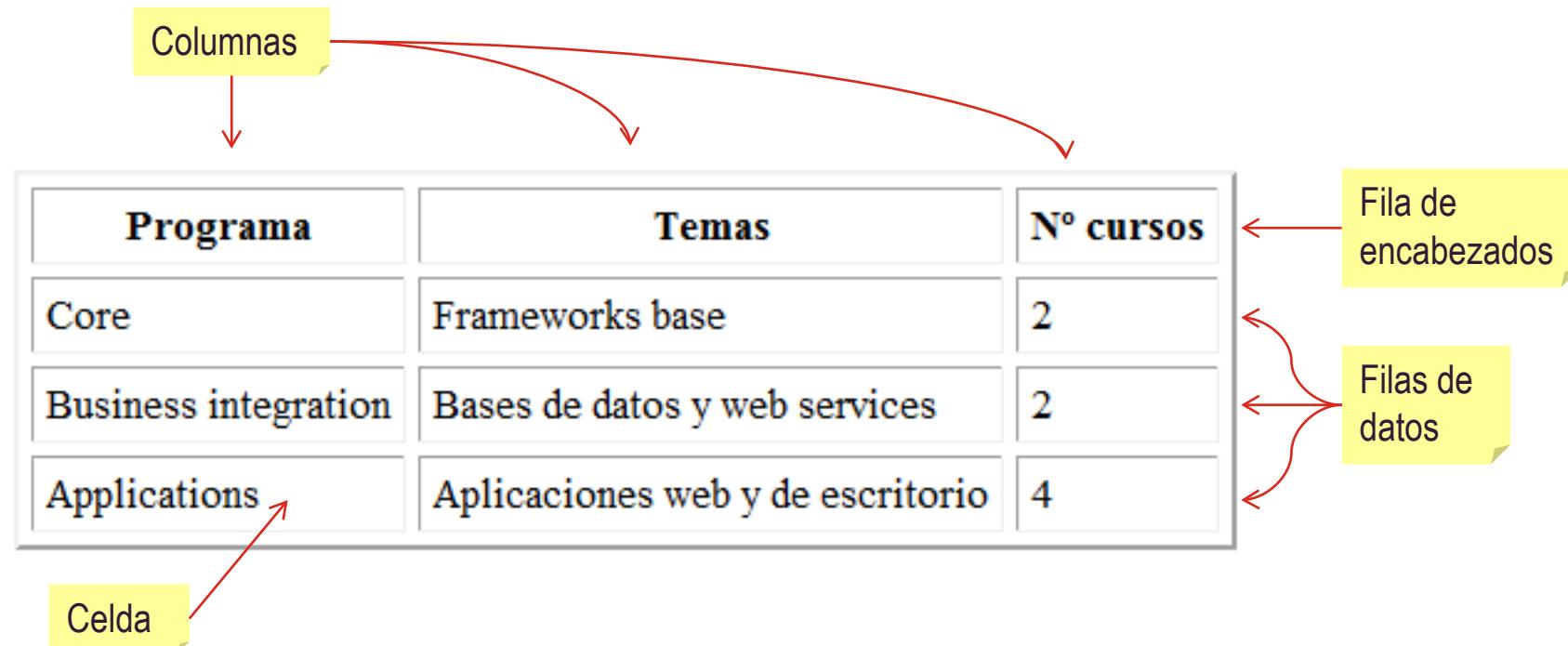
```
<div>
  <h3>Título</h3>
  <p>Contenido</p>
</div>
```

```
<p>Curso <span>(1)</span>: HTML</p>
```

- La diferencia entre **<div>** y **<span>** es que el primero es de tipo **block**, es decir, agrega un salto de línea, mientras que el segundo es **inline**, y no lo añade.
- Utilidad del manejo de secciones:
  - Definir características comunes, principalmente a través de reglas de estilo, como se ve más adelante.

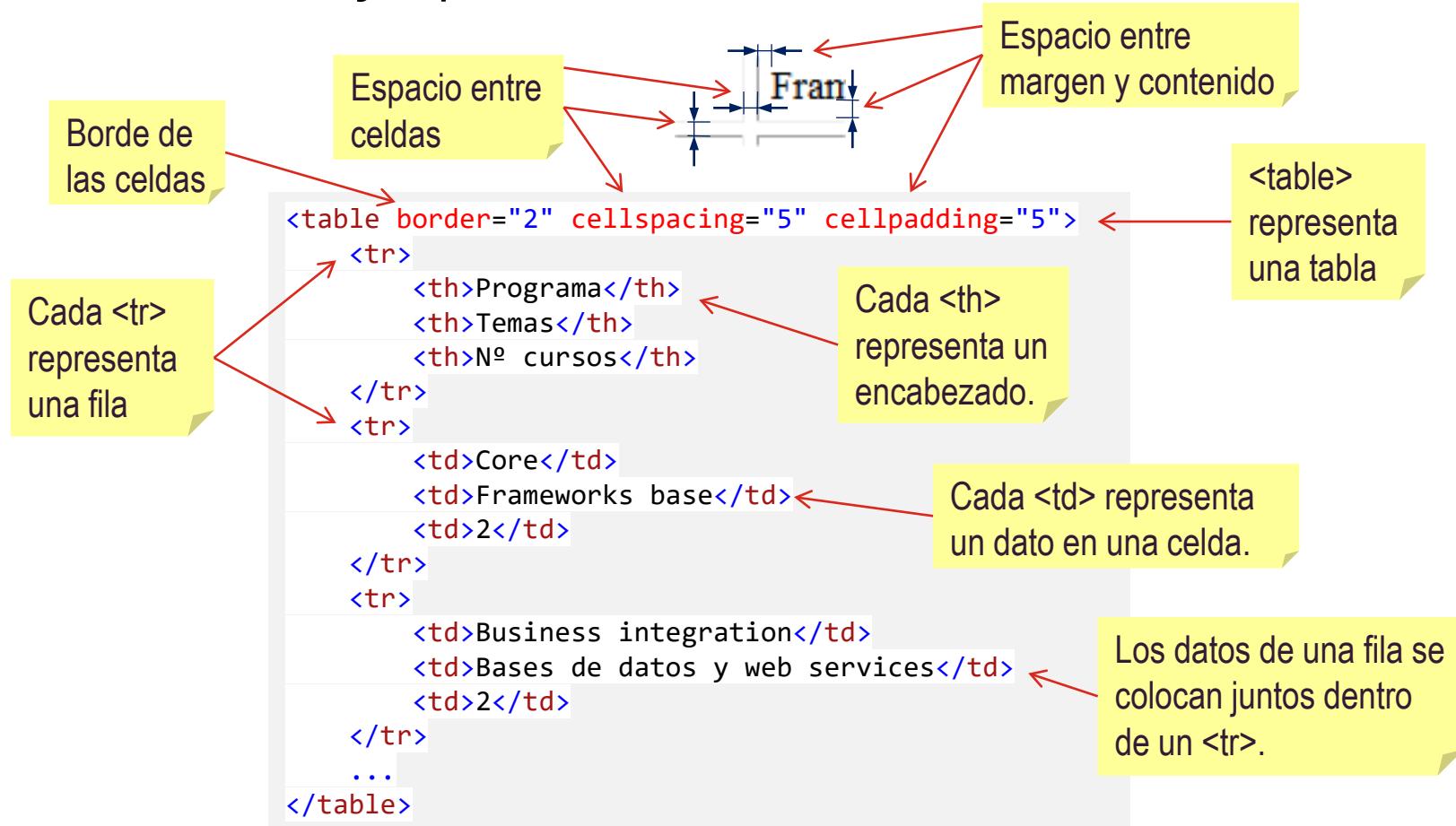
# Elementos HTML básicos: Tabla

- Estructura y elementos de una tabla:



# Elementos HTML básicos: Tabla

- La etiqueta **<table>** define una tabla junto **<tr>**, **<th>** y **<td>**
- Código HTML de la tabla ejemplo anterior:



# Elementos HTML básicos: Tabla

- Combinación de celdas con **rowspan** y **colspan**:

```
<table border="2" cellspacing="5" cellpadding="5">
    <caption>Cursos de formación</caption>
    <tr>
        <th rowspan="2">Programa</th>
        <th colspan="2">Curso</th>
    </tr>
    <tr>
        <th>Nombre</th>
        <th>Duración</th>
    </tr>
    <tr>
        <td rowspan="2">Business integration</td>
        <td>Persistencia</td>
        <td>24 horas</td>
    </tr>
    <tr>
        <td>Web services</td>
        <td>24 horas</td>
    </tr>
</table>
```

Cursos de formación		
Programa	Curso	
Nombre	Duración	
Business integration	Persistencia	24 horas
	Web services	24 horas

# Elementos HTML básicos: Formulario

- Un formulario `<form>` agrupa un conjunto de **campos** (controles de formulario) que permiten al usuario insertar datos e interactuar con sus elementos.

The diagram illustrates the mapping of an HTML form's structure to its visual representation. On the left, the HTML code is shown, and on the right, the corresponding form fields are displayed with red arrows indicating the mapping between them.

```
<form action="...">
  <input type="hidden" name="issueId" id="summ"
    name="summary" value="this is a summary">
  </p>
  <p>Secret: <input type="password" id="passwd"
    name="secret" size="15" value="mypassword">
  </p>
  <p>Type: <select name="typeId" id="typeId" >
    <option value="1" selected>
      Technology</option>
    <option value="2">Production</option>
  </select>
  </p>
  <p><input type="checkbox" name="hasAttachments">
  Has attachments</p>
  <p><input type="file" name="attachment">
  </p>
  ...

```

Visual representation:

- Campo oculto**: A dashed rectangular field labeled "Summary: this is a summary".
- Secret:** A password input field containing "\*\*\*\*\*".
- Type:** A dropdown menu set to "Technology".
- Has attachments**: A checkbox labeled "Has attachments".
- Campo para upload**: A file input field with a "Browse..." button.

# Elementos HTML básicos: Formulario

The diagram illustrates the mapping of an HTML form's structure to its rendered state. On the left, the HTML code is shown, and on the right, the resulting user interface elements are displayed. Red arrows point from specific code snippets to their corresponding UI components.

**HTML Code:**

```
...
<p>State:<br>
<input type="radio" name="state" value="P">Pending
<input type="radio" name="state" value="S" checked>Solved
<input type="radio" name="state" value="C">Closed
</p>
<p>Description:<br>
<textarea rows="3" cols="40" id="desc">
This is a description
</textarea></p>
<p><input type="reset" value="Clean">
</p>
<p><input type="submit" value="Send">
</p>
</form>
```

**UI Components:**

- State:** A group of three radio buttons labeled "Pending", "Solved", and "Closed". The "Solved" button is checked. A yellow callout notes: "El grupo de los radio button lo define el name común a todos."
- Description:** A text area containing the text "This is a description".
- Clean:** A reset button labeled "Clean".
- Send:** A submit button labeled "Send".

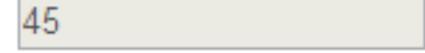
**Annotations:**

- A yellow callout near the bottom right points to the "Send" button with the text: "Envía el formulario al destino definido en "action" utilizando el "method"."

# Elementos HTML básicos: Formulario

- HTML provee ciertas **validaciones** o comportamientos de los formularios de forma nativa:
- **Habilitación**: se puede deshabilitar un campo de cualquier tipo con el atributo **disabled**. Tiene normalmente texto gris.

```
<input type="text" value="45"  
      name="identifier" disabled>
```



45

- **Sólo lectura**: se puede dejar un campo de texto como sólo lectura con el atributo **readonly**. Tiene el mismo estilo, pero no permite escribir.

```
<input type="text" value="resumen"  
      name="summary" readonly>
```



resumen

- **Largo máximo**: El atributo **maxlength** especifica el número máximo de caracteres permitidos en un campo de entrada, el cual es recomendable añadir por seguridad.

# Elementos HTML básicos: Formulario

- **Tamaño**: el ancho visible, en caracteres, de un campo de entrada se especifica con el atributo **size**.

```
<input type="text" name="description" size="45">
```

- **Requerido**: el atributo **required** obliga a introducir valores en un campo antes de enviar los datos del formulario.

```
<input type="text" name="id" value="Identifier" required>
```

# Elementos HTML básicos: Formulario

- Los controles de un formulario se pueden manipular como un objeto JavaScript, permitiendo aplicar HTML dinámico.

```
<form class="frm" >
  <div>
    <label for="idSumm">Summary:</label>
    <input type="text" size="30" id="idSumm" name="summary" value="this is a summary">
  </div>
  <div>
    <label for="secret">Secret:</label>
    <input type="password" id="secret" size="15">
  </div>
  <div>
    <label for="type">Type:</label>
    <select id="idType" name="type">
      <option value="1" selected="selected">
        Technology</option>
      <option value="2">Production</option>
    </select>
  </div>
</form>
```

# Elementos HTML básicos: iFrame

- La etiqueta `<iFrame>` especifica un marco en línea y permite incrustar otro documento HTML dentro del actual.

```
<iFrame src="https://www.netmind.es" title="Web de Netmind">
```

- Los atributos `height` y `width` definen el alto y ancho respectivamente del marco en línea.

```
<iFrame src="https://www.netmind.es" title="Web de Netmind" height="200" width="300">
```

# **Buenas prácticas**

# Buenas prácticas

## Espacios, tabulaciones y saltos de línea

- HTML sólo considera un espacio, aunque hayan más, e ignora las tabulaciones y saltos de línea:

```
<p>Los múltiples espacios, tabulaciones y saltos  
de línea se interpretan como un solo espacio</p>
```

Los múltiples espacios, tabulaciones y saltos de línea se interpretan como un solo espacio

- Para insertar más de un espacio, se debe utilizar "&nbsp;". Ejemplo:

```
<p>Múltiples&nbsp;&nbsp;&nbsp;espacios entre palabras</p>
```

Tres espacios

Múltiples espacios entre palabras

# Buenas prácticas

## Block vs inline

- En HTML existen etiquetas del tipo **block** y del tipo **inline**.
- Las de tipo **block** agregan un salto de línea después de su utilización:

```
<h2>Encabezado 2</h2>  
<h3>Encabezado 3</h3>
```

**Encabezado 2**

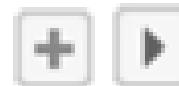
**Encabezado 3**

Saltos de  
línea

- Las de tipo **inline** no lo agregan:

```
  

```



No hay salto  
de línea

# Buenas prácticas

## Anidamiento de etiquetas

- En HTML, las etiquetas se pueden anidar, permitiendo contener elementos en otros:

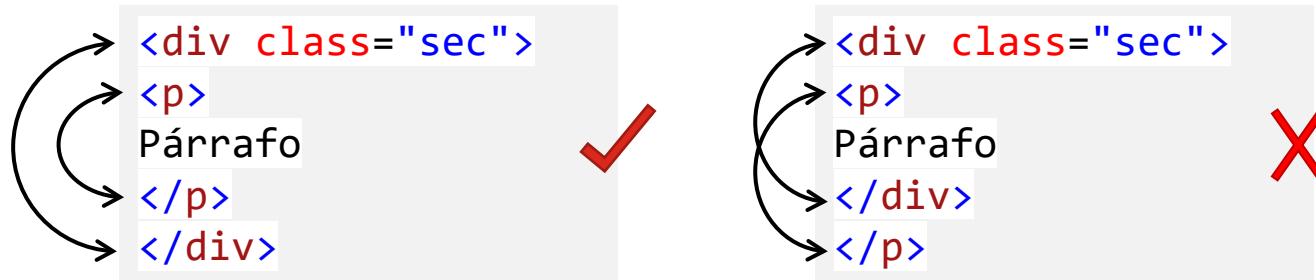
```
<html>
  <body>
    <div class="frame">
      <div class="brd">
        <div class="sec">Sección</div>
      </div>
    </div>
    <p>Párrafo</p>
  </body>
</html>
```

- Esto implica que un documento HTML tiene una estructura tipo árbol.

# Buenas prácticas

## Cierre de etiquetas

- Al estar anidadas, las etiquetas deben cerrarse de manera adecuada, respetando el orden de la jerarquía:



# Caracteres especiales

- En HTML, los caracteres especiales (acentos, eñes y otros) se pueden escribir directamente:

```
<p>la antigüedad en la compañía</p>
```

- No obstante, si el **juego de caracteres** de la página no corresponde con el del archivo, se pueden producir conflictos que hacen que no se vean correctamente. Por ejemplo, si el archivo HTML utiliza **utf-8** y la página declara **iso-8859-1**, se vería algo así:

la antigüedad en la compaÑa

# Caracteres especiales

- Para evitar los efectos por diferencias de juego de caracteres entre el archivo físico y el declarado en el contenido HTML, existe una nomenclatura para los caracteres especiales:

```
<p>la antig&uuml;edad en compa&ntilde;&iacute;a</p>
```

- Nomenclatura para las vocales acentuadas:

**&[letra][tipo modificador];**

a – e  
A – E

En algunos idiomas,  
otras letras no vocales.

- Distintos tipos usados en algunos idiomas:
- acute (agudo): á, Á
  - grave (grave): à, À
  - circ (cincunflejo): â, Â
  - tilde (tilde): ã, Ã
  - uml: ä, Ä
  - ring (anillo): å, Å

# Caracteres especiales

- Además de los acentos, hay otros caracteres especiales, algunos de los cuales se describen en la siguiente tabla:

Carácter	Nombre entidad	Descripción
&	&amp;	símbolo & (ampersand)
(espacio)	&ampnbsp	espacio en blanco (non breaking space)
"	&quot;	comillas
'	&apos;	apóstrofe (las llamadas comillas simples)
<	&lt;	menor que (less than)
>	&gt;	mayor que (greater than)
¿	&iquest;	abre signo de interrogación
¡	&iexcl;	abre signo de exclamación/admiración
©	&copy;	copyright

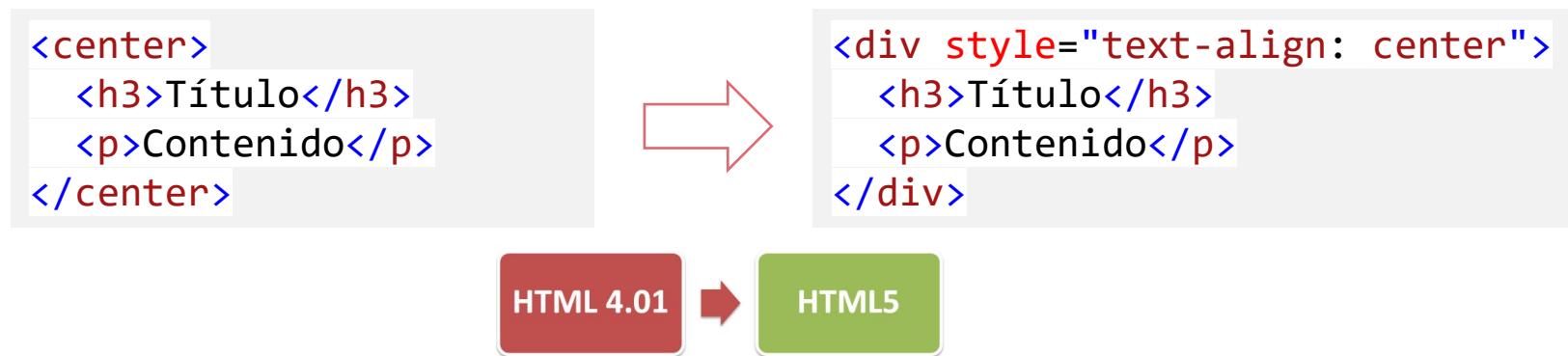
# Caracteres especiales

- También se utiliza la nomenclatura "&#[código decimal];" para los caracteres especiales.  
Ejemplos:

Carácter	Nombre entidad	Descripción
(espacio)	&#160;	Espacio en blanco
&	&#38;	símbolo &
"	&#34;	Comillas
'	&#39;	apóstrofe (las llamadas comillas simples)
<	&#60;	menor que
>	&#62;	mayor que
¿	&#191;	abre signo de interrogación
¡	&#161;	abre signo de exclamación/admiración

# Elementos en desuso (deprecated)

- Algunos elementos y etiquetas HTML en desuso:
  - Fuente `<font>`
  - Texto parpadeante `<blink>`
  - Marquesina `<marquee>`
  - Marco o Frame `<frame>`
  - Alineaciones `<center>`
  - ...



# **HTML 5**

# HTML 5

Novedades más importantes:

- Estructura semántica
- Etiquetas aportan estructura. Atributos aportan funcionalidad.
- Formularios mejorados: significado semántico
- Nuevos valores que puede adoptar la etiqueta `<input>`
- Accesibilidad
- Capacidad de crear atributos personalizados.
- Nuevos elementos multimedia (audio y video)

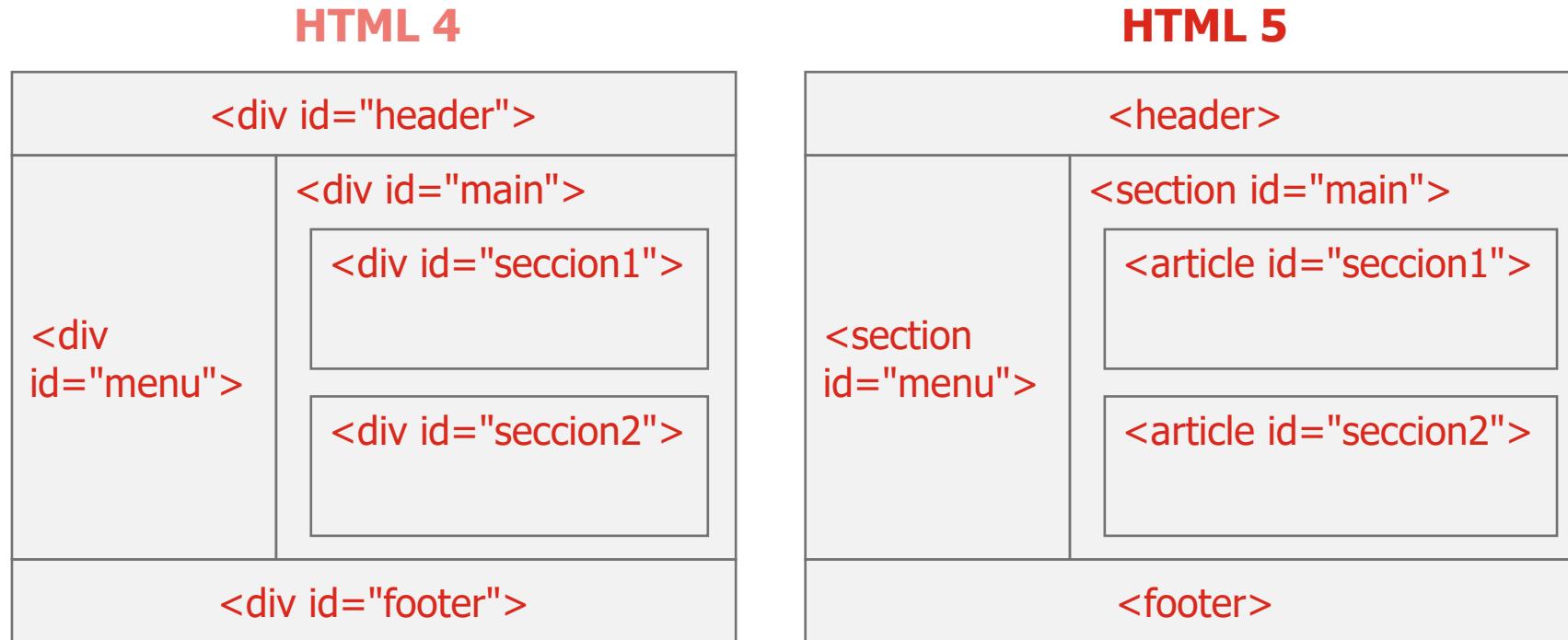


# Estructura de un documento HTML 5



# Estructura de un documento HTML 5

## Ejemplo estructura HTML4 vs HTML5



# HTML5

## Nuevos valores para <input>

```
<p>tel: <input type="tel" /></p>
<p>search: <input type="search" /></p>
<p>url: <input type="url" /></p>
<p>email: <input type="email" /></p>
<p>datetime: <input type="datetime" /></p>
<p>date: <input type="date" /></p>
<p>month: <input type="month" /></p>
<p>week: <input type="week" /></p>
<p>time: <input type="time" /></p>
<p>datetime-local: <input type="datetime-local" /></p>
<p>number: <input type="number" /></p>
<p>range: <input type="range" /></p>
<p>color: <input type="color" /></p>
```

tel:

search:

url:

email:

datetime:  2011-08-30 ▾ 23:55 ⌂ UTC

date:

month:  2011-08

week:  2011-W32

time:  23:27 ⌂

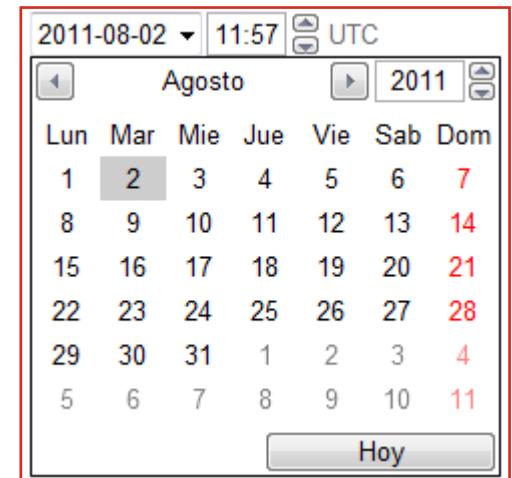
datetime-local:  2011-08-30 ▾ 23:53 ⌂

number:  26 ⌂

range:

color:  #000000 ⌂

Otros...



# HTML5

## Marcador de posición o **placeholder**

- Muestra un texto al usuario en un tono gris claro, indicando tipo de dato se espera que se introduzca en el cuadro del input.
- Se implementa como un atributo al que se asigna la cadena a mostrar y es común a todos los valores del atributo type donde tiene sentido: tel, search, url, email.
- Si el usuario pulsa dentro de la caja de texto, el texto gris claro desaparece.

```
<input type="tel" placeholder="format:(xxx)xx-xx-xxx">
```



A screenshot of a web browser showing a text input field. The placeholder text "format: (xxx) xx-xx-xxx" is displayed in a light gray font inside the input field. The input field has a thin black border and is currently empty of user input.

# HTML5

## Autoenfoque o **autofocus**

- Indica qué control del formulario debe tener el foco cuando se carga la página.
- Ejemplo:

```
<form>  
    <label for="fname">First name:</label><br>  
    <input type="text" id="fname" name="fname" autofocus><br>  
    <label for="lname">Last name:</label><br>  
    <input type="text" id="lname" name="lname"><br>  
    <input type="submit">  
</form>
```

The diagram shows a user interface with the following elements:

- A label "First name:" followed by a text input field.
- A label "Last name:" followed by a text input field.
- A "Submit" button.

A red arrow originates from the word "autofocus" in the explanatory text above and points directly to the first text input field.

# HTML5

## Elemento <datalist>

- El elemento <datalist> no tiene definido ningún tipo de presentación, pero cuando se utiliza conjuntamente con otro control, puede presentar un cuadro combinado o combobox.
- Se complementa con elementos <option> para indicar cada uno de los posibles valores de la selección.
- Ejemplo: Combinar un elemento <datalist> con una entrada <input> de tipo email.

```
<input list="contactos" name="contactmail">
<datalist id="contactos">
    <option value="davidcorrea@gmail.com" label="David">
    <option value="luisagomez@gmail.com" label="Luisa">
</datalist>
```

# HTML5

## Entrada de valores numéricos mediante el tipo **range**

```
<label>Years hired: </label>
<input type="range" max="10" min="0" value="5" onchange="funcionJS()">
<span id="showYears">7</span>

<script type="text/javascript">
    function funcionJS() {
        alert();
    }
</script>
```

Years hired:  7

# HTML5

## Acotación de valores con atributos **min** y **max**

- Podemos acotar aún más las posibilidades de entrada, facilitando la labor de validación y minimizando errores. Tal es el caso de **max** y **min** (valor máximo y mínimo), para el caso de los campos numéricos.

```
<label for="fyear">Years hired:</label>
<input type="number" max="10" min="0" value="5">
```



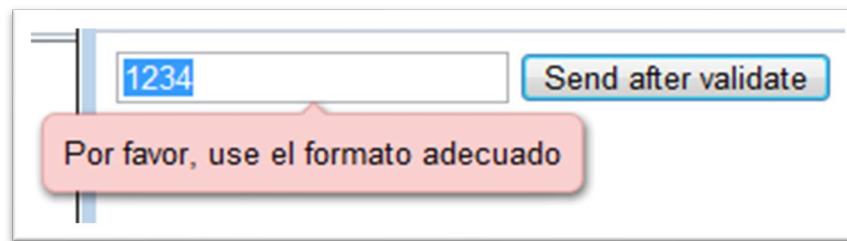
# HTML5

## Validación de campos de formulario

- A través del atributo **pattern** podemos utilizar un expresión regular como **patrón** para restringir la entrada de datos, obteniendo así un primer nivel de validación garantizando, en el ejemplo, que se introducen solamente números y un máximo de tres dígitos:

```
<input type="tel" pattern="\d{3}">
```

- Se recomienda acompañarlo del atributo **title** para avisar al usuario del patrón válido.
- Si el valor introducido no cumple con la expresión regular, el navegador puede mostrar una etiqueta flotante:





# ACTIVIDAD

---

## Formulario

## OBJETIVO

Diseñar un formulario para recopilar datos mediante los principales elementos de entrada.

## INSTRUCCIONES

1. Crear un formulario que incluya los elementos y propiedades de los siguientes puntos.
2. Añadir los campos para introducir los datos de un usuario: ID, Nombre de usuario, email, fecha de nacimiento y un campo que permita elegir el tipo de usuario de una lista de valores (Superusuario, Estandar, Lector).
3. El campo ID deberá introducirse obligatoriamente y debe ser el que reciba el foco cuando se cargue la página.
4. El ancho visible del campo de entrada para el nombre debe ser 50 y para el email de 30 caracteres.



30 min

5. La lista de tipos de usuario deberá realizarse con el elemento datalist.
6. Insertar un campo código postal que defina el patrón "`^[0-9]{5}$`". Mostrar un mensaje al usuario cuando el formato del CP introducido no cumpla con el patrón definido.
7. Incluir un área de texto que permita introducir un comentario de 500 caracteres.
8. Añadir un botón "Enviar" que permita el envío de los datos del formulario.
9. Añadir otro botón "Limpiar" para resetear o limpiar todos los campos.

ID:  
1

Nombre de usuario:

Email:

Fecha de nacimiento:  
dd/mm/aaaa

CP:

Tipo de usuario:

Superusuario  
Estandar  
Lector

Escriba aquí sus comentarios...

Enviar Limpiar



30 min

# 01

# Elaboración de documentos web mediante lenguajes de marca

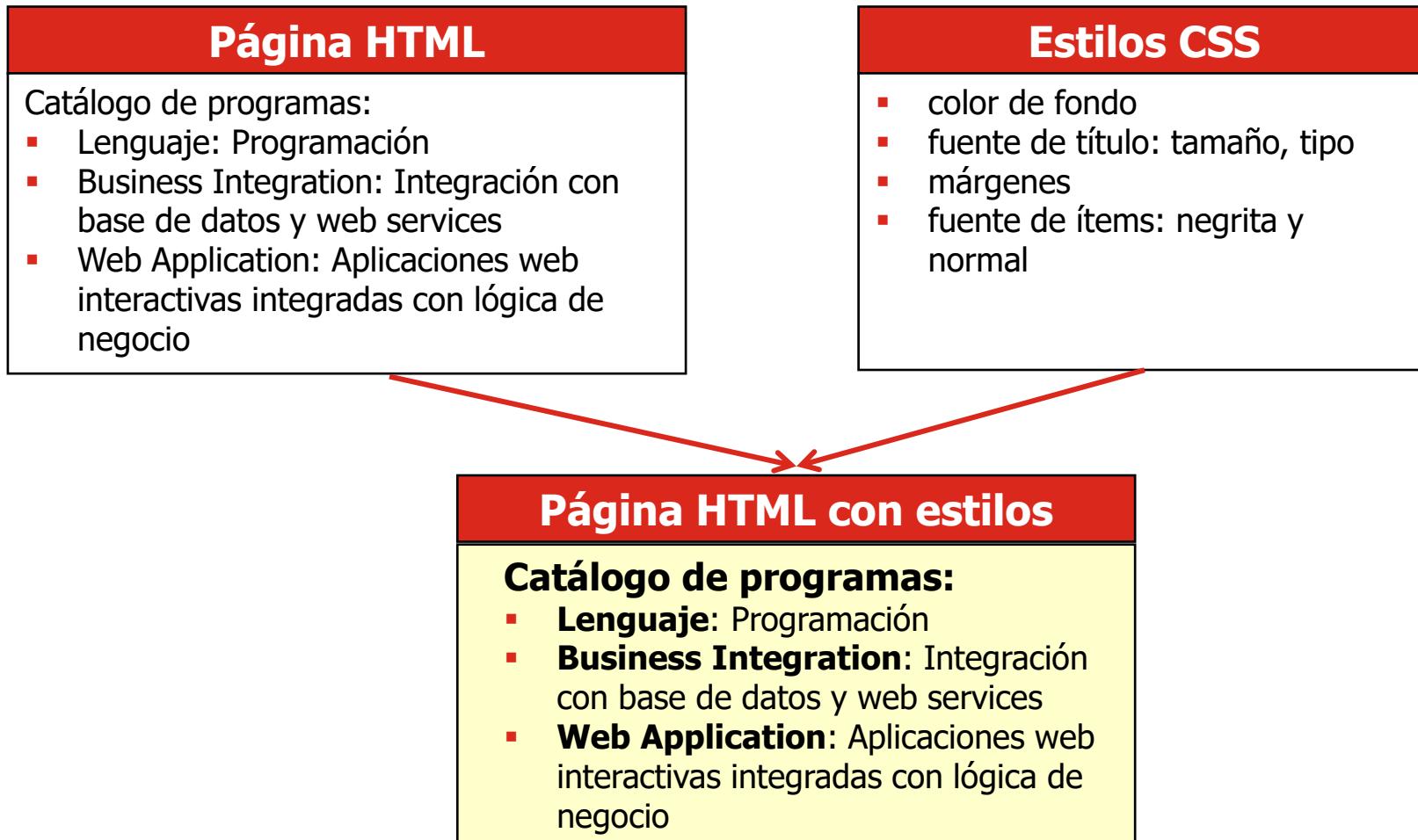
## 4. Hojas de Estilo en Cascada (CSS)

- Elementos y estructura de una hoja de estilo: Creación de hojas de estilo. Aplicación de estilos. Herencia de estilos y aplicación en cascada. Formateado de páginas mediante estilos. Estructura de páginas mediante estilos.
- Diseño de estilos para diferentes dispositivos.
- Buenas prácticas en el uso de hojas de estilo.

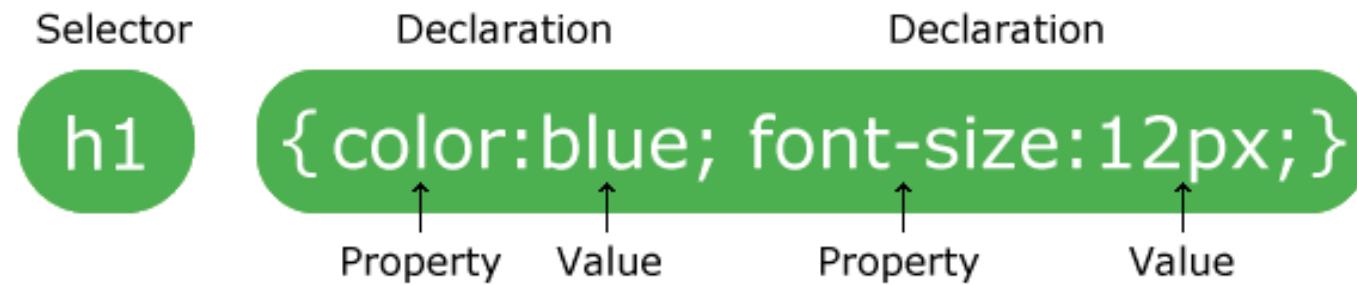
# Hojas de Estilo en Cascada - CSS

- Lenguaje usado para **definir la presentación de un documento** estructurado escrito en HTML o XML.
- Una indicación de estilo recibe el nombre de **regla de estilo**.
- Mediante un **selector** se especifica a qué elemento HTML se aplican los estilos.
- El **World Wide Web Consortium (W3C)** es el encargado de definir la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

# Aplicación de Estilos



# Regla de Estilo CSS



- Los **selectores CSS** se utilizan para especificar el/los elementos HTML a los que se desea aplicar estilo.

# CSS interno: Declaración de estilos con <style>

Utiliza la etiqueta <style>,  
a nivel de página.

```
<head>
  <style>
    h1 {
      color: blue;
      font-family: Verdana;
    }
  </style>
</head>
<body>
  <h1>Título con estilo</h1>
  <p style="color:red">En style se
    pueden definir estilos gráficos<br>
    para los elementos HTML</p>
</body>
```

La nomenclatura utiliza {llaves} y separación  
por ';' entre estilos, y ':' para la asignación.

Utiliza el atributo  
style localmente.

\*El uso de <style> en el <head> no es recomendado.

# CSS interno: Ejemplo

```
<head>
  <style>
    h1 {
      color: blue;
      font-family: Verdana;
    }
  </style>
</head>
<body>
  <h1>Título con estilo</h1>
  <p style="color:red">En style se
    pueden definir estilos gráficos<br>
    para los elementos HTML</p>
</body>
```

Este estilo se aplica a todos los elementos h1

Se define para los h1 el color azul y la familia (tipo de letra) Verdana para la fuente.

<p> tiene definido localmente el color de la fuente

## Título con estilo

En style se pueden definir estilos gráficos para los elementos HTML

# CSS externo: Declaración de estilos con <link>

- La declaración de estilos con la etiqueta <style> en el head, desventajas:
  - No es reutilizable en otras páginas.
  - Combina en un mismo archivo los estilos gráficos y el contenido HTML.
- Buenas prácticas: estilos especificados en un archivo separado llamado **Hoja de Estilo** o **Cascade Style Sheet (CSS)**.
- Ejemplo de uso en la sección <head>:

```
<link type="text/css" rel="stylesheet" href="estilos.css">
```

Referencia a archivo CSS con la hoja de estilos.  
Puede incluir ruta relativa, absoluta o URL.

# CSS externo: Ejemplo

```
<head>
  <link type="text/css"
        rel="stylesheet"
        href="styles.css">
</head>
<body>
  <h1>Título con estilo</h1>
  <p style="color:red">En style se
    pueden definir estilos gráficos<br>
    para los elementos HTML</p>
</body>
```

Se pueden utilizar estilos locales (**CSS en línea**) para casos concretos.

styles.css

```
h1 {
  color: blue;
  font-family: Verdana;
}
```

## Título con estilo

En style se pueden definir estilos gráficos para los elementos HTML

# Selectores CSS: universal

- El **selector universal** (\*) representa **todos** los elementos HTML:

```
* {  
    color: blue;  
    text-decoration: none;  
}
```

# Selectores CSS: elemento

```
p {  
    color: blue; ←  
}  
  
a {  
    text-decoration: none; ←  
}  
  
div {  
    padding: 20px; ←  
}
```

Todos los párrafos en color azul

Hyperlink sin texto subrayado

Espacio entre el borde y el contenido del elemento div

# Selectores CSS: id

Estilo asociado a id.  
Se marca con #.

```
#hjava {  
    color: red;  
}  
  
#hnet {  
    color: blue;  
}
```

id del elemento

```
<h3 id="hjava">Catálogo de cursos Java</h3>  
  
<h3 id="hnet">Catálogo de cursos .NET</h3>
```

**Catálogo de cursos Java**

**Catálogo de cursos .NET**

# Selectores CSS: clase

Nombre de clase. Se expresa con un punto

```
↓  
.hred {  
    color: red;  
}
```

```
.hblue {  
    color: blue;  
}
```

Nombre de clase.

```
<h3 class="hred">Catálogo de cursos Java</h3>  
  
<h3 class="hblue">Catálogo de cursos .NET</h3>
```

**Catálogo de cursos Java**

**Catálogo de cursos .NET**

# Selectores CSS: **body**

- Cuando se definen estilos para <body>, se aplican a todo el contenido HTML.

```
body {  
    background-color: #F8F8F8;  
    font-family: Arial;  
}
```

# Selectores CSS: Agrupación

- Seleccionar elementos HTML con las mismas definiciones de estilo.

```
h1, h2, p {  
    text-align: center;  
    color: red;  
}
```

# Comentarios en CSS

- Sintaxis:

```
/* bloque comentado */
```

- El bloque comentado puede tener una o más líneas.

```
/*
  Cuando un link es visitado recupera el color original
*/
a:visited { text-decoration: none; }
```

- No existen restricciones en el contenido de los comentarios en CSS.

# **Principales Estilos**

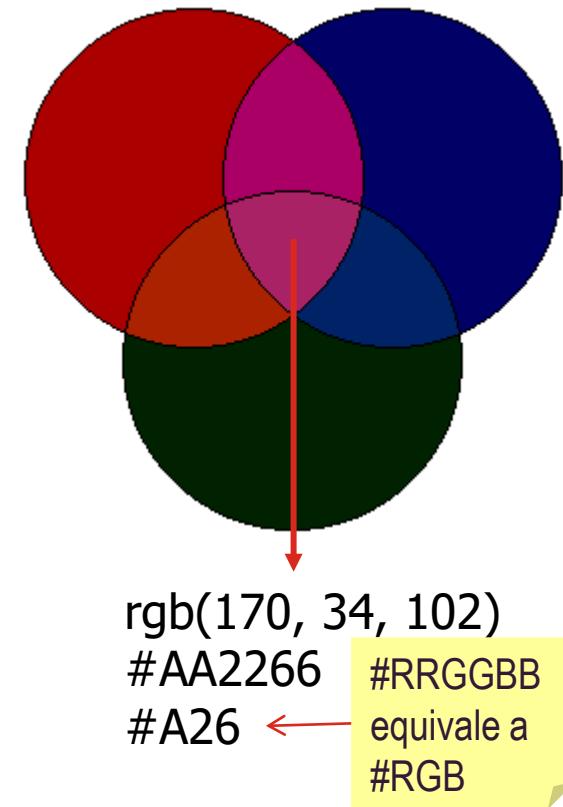
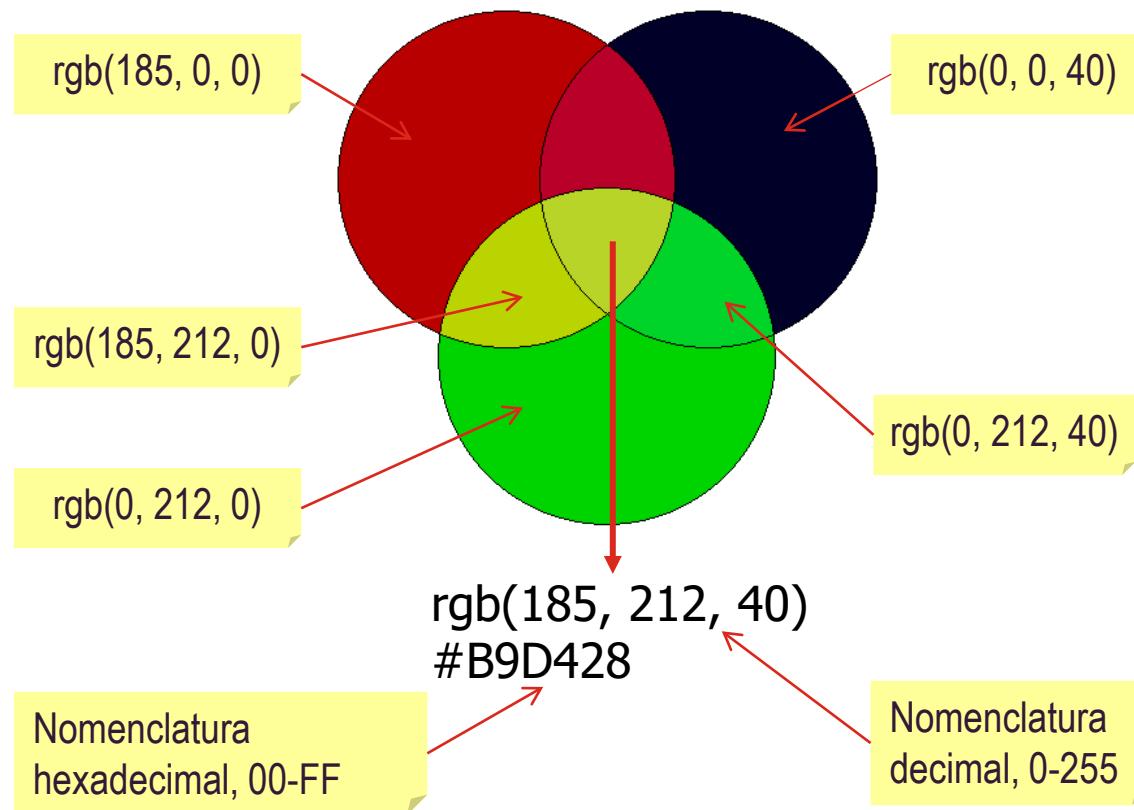
# Declaración de estilo: Color

- Permite definir colores utilizando **atributos** de estilo:
  - **color**: Color de fuente (texto)
  - **background-color**: Color de fondo
  - **border-color**: Color de bordes
- Especificación de un color mediante su nombre predefinido:



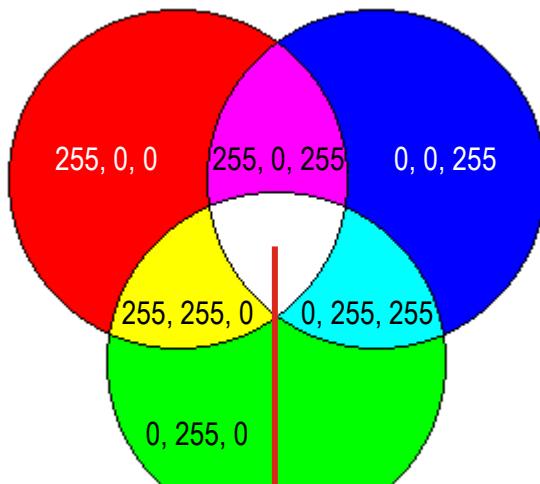
# Declaración de estilo: Color

- Modelo RGB (Red, Green, Blue):

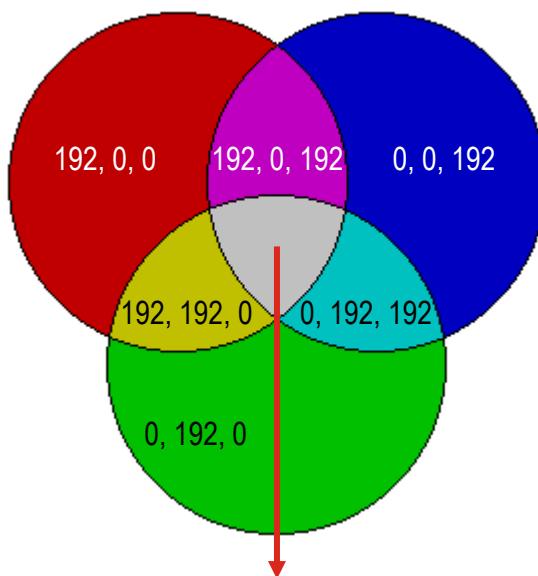


# Declaración de estilo: **Color**

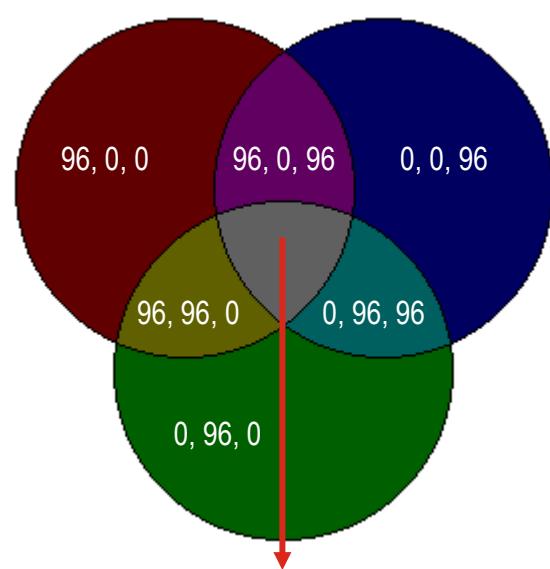
- Forma simplificada de RGB cuando el nivel es el mismo en los tres colores:



rgb(255, 255, 255)  
#FFFFFF  
#FFF



rgb(192, 192, 192)  
#C0C0C0



rgb(96, 96, 96)  
#606060

# Declaración de estilo: **Fuente**

- Principales atributos/propiedades de **tipografía** y sus posibles valores:

<b>font-family</b>	<b>font-size</b>	<b>font-weight</b>	<b>font-style</b>	<b>font-variant</b>
Arial	0.9em 9pt	400 normal	normal	normal
Verdana	1.0em 11pt	700 <b>bold</b>	<i>italic</i>	SMALL-CAPS
Sans-serif	1.4em 15pt	lighter	<i>oblique</i>	

# Declaración de estilo: Texto

- Estilos para **decorar, alinear o transformar** el texto:

text-align	text-decoration	text-indent	text-shadow	text-transform
left	<u>underline</u>	0px	0em 0.13em gray	none
center	blink	5px	0.13em 0em gray	lowercase
right	<u>line-through</u>	10px	0.12em 0.12em gray	UPPERCASE
A small text with align justify	<u>overline</u>	20px	0.12em 0.12em red	Capitalize First Letter

# Declaración de estilo: Link

- Para el hyperlink `<a>`, se pueden definir estilos en función de su estado:

```
a { cursor: pointer; text-decoration: none; } ← Estilos para el hyperlink.  
En este caso, se suprime el  
subrayado.  
  
a:link { } ← Estilos cuando aún no ha  
sido visitado.  
  
a:visited { text-decoration: none; } ← Estilos cuando ya ha sido visitado. En este  
caso, se elimina el subrayado y recupera el  
color original.  
  
a:hover { text-decoration: underline; } ← Estilos cuando se pasa el  
puntero sobre el hyperlink.  
  
a:focus, a:active {text-decoration: underline;} ← Estilos cuando tiene el foco (focus), o cuando se presiona (active).
```

# Principales estilos: Listas

- El atributo **style** permite especificar el estilo de los elementos <li>.

```
<ul style="list-style: square">
  <li>C# Básico</li>
  <li>C# Avanzado</li>
</ul>
```

- C# Básico
- C# Avanzado

```
<ul style="list-style: circle">
  <li>C# Básico</li>
  <li>C# Avanzado</li>
</ul>
```

- C# Básico
- C# Avanzado

```
<ul style="list-style: url(play.png)">
  <li>C# Básico</li>
  <li>C# Avanzado</li>
</ul>
```

- ▶ C# Básico
- ▶ C# Avanzado

# Principales estilos: Listas

- El atributo **style** permite especificar el estilo de los elementos <li>.

```
<ol style="list-style: upper-roman">
  <li>Java Básico</li>
  <li>Java Intermedio</li>
  <li>Spring Core</li>
  <li>Hibernate-JPA</li>
  <li>Web Services</li>
</ol>
```

- I. Java Básico
- II. Java Intermedio
- III. Spring Core
- IV. Hibernate-JPA
- V. Web Services

```
<ol style="list-style: lower-alpha">
  <li>Java Básico</li>
  <li>Java Intermedio</li>
  <li>Spring Core</li>
  <li>Hibernate-JPA</li>
  <li>Web Services</li>
</ol>
```

- a. Java Básico
- b. Java Intermedio
- c. Spring Core
- d. Hibernate-JPA
- e. Web Services

```
<ol style="list-style: decimal">
  <li>Java Básico</li>
  <li>Java Intermedio</li>
  <li>Spring Core</li>
  <li>Hibernate-JPA</li>
  <li>Web Services</li>
</ol>
```

- 1. Java Básico
- 2. Java Intermedio
- 3. Spring Core
- 4. Hibernate-JPA
- 5. Web Services

# Principales estilos: Contenedor o Sección (div)

- Si se requiere aplicar estilos a una sección en particular de una página, una opción bastante utilizada es a través de un <div> o <span>.
- Por ejemplo, una sección cuya fuente es de tipo Arial, y con color de fondo "wheat" (uno de los que no son principales):

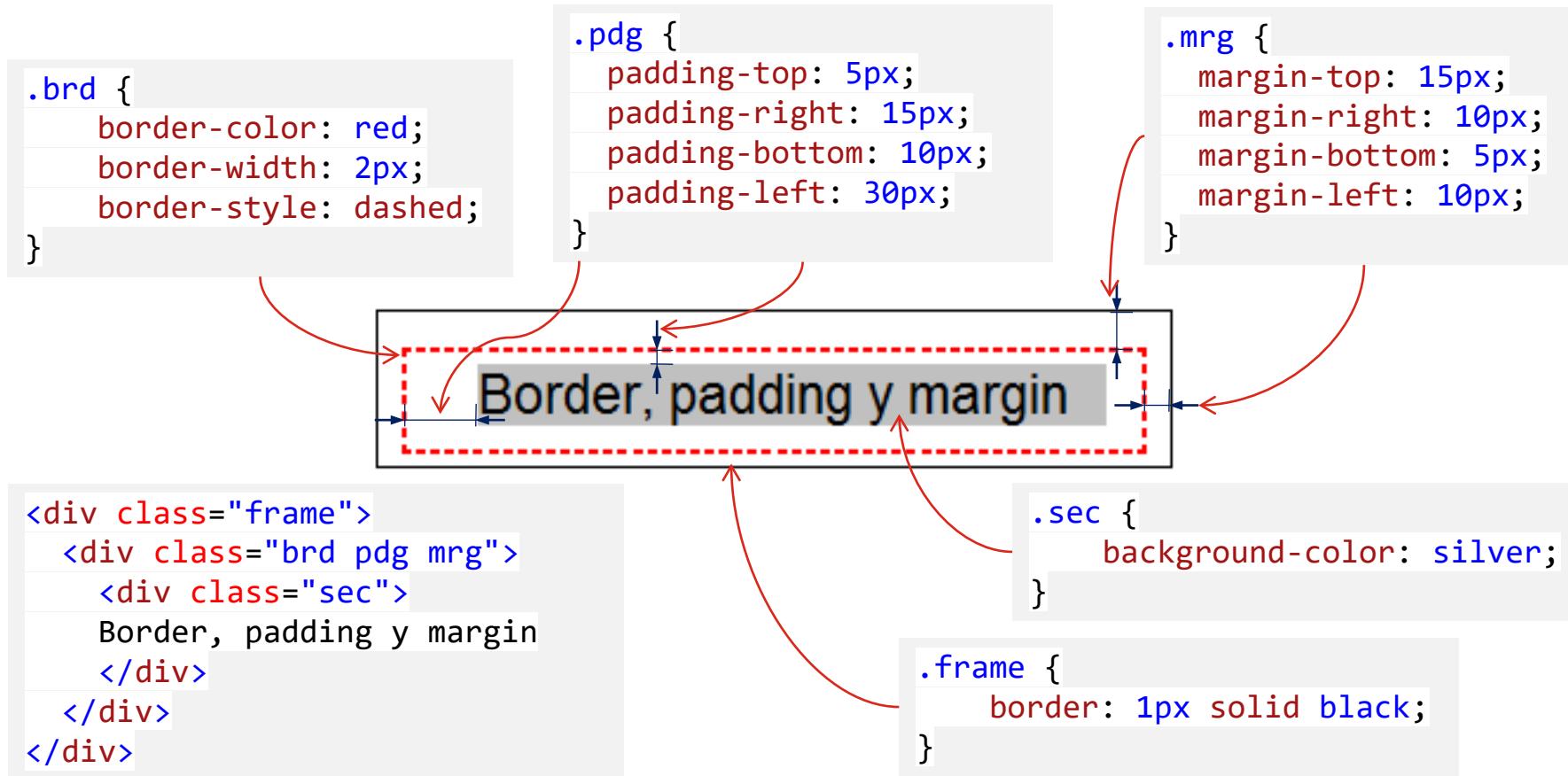
```
.sample {  
    font-family: Arial;  
    background-color: Wheat;  
}
```

```
<div class="sample">  
    <h3>Título</h3>  
    <p>Contenido</p>  
</div>
```

Título  
Contenido

# Principales estilos: margin, border, padding

- A un elemento HTML se le pueden definir los atributos gráficos border, padding y margin:



# Principales estilos: margin, border, padding

- Aplicando distintas combinaciones:

```
<div class="frame">
  <div class="brd">
    <div class="sec">Border</div>
  </div></div><br>
<div class="frame">
  <div class="brd pdg">
    <div class="sec">Border y padding</div>
  </div></div><br>
<div class="frame">
  <div class="brd pdg mrg">
    <div class="sec">Border, padding y margin</div>
  </div></div><br>
<div class="frame">
  <div class="pdg mrg">
    <div class="sec">Padding y margin</div>
  </div></div><br>
<div class="frame">
  <div class="mrg">
    <div class="sec">Margin</div>
  </div>
</div>
```

Border

Border y padding

Border, padding y margin

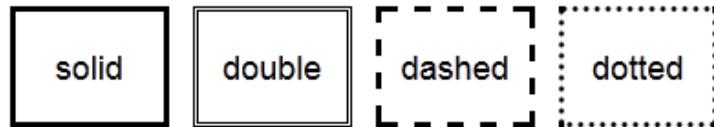
Padding y margin

Margin

# Principales estilos: **border-style**, **border-color**

Opciones para border:

- style:



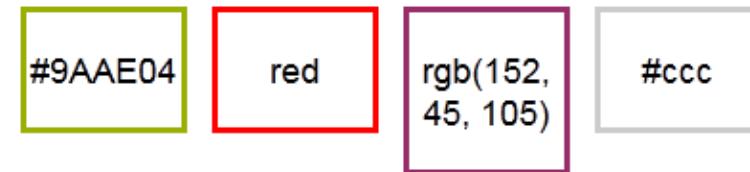
```
border-style: solid;
```

```
border-style: double;
```

```
border-style: dashed;
```

```
border-style: dotted;
```

- color:



```
border-color: #9AAE04;
```

```
border-color: red;
```

```
border-color: rgb(152, 45, 105);
```

```
border-color: #ccc;
```

# Principales estilos: **border-width**, **border-radius**

Opciones para border:

- width:



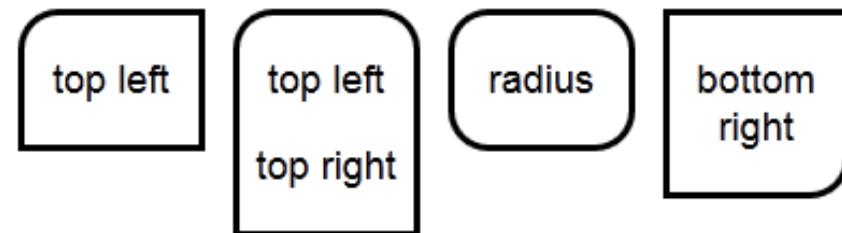
```
border-width: thin;
```

```
border-width: medium;
```

```
border-width: thick;
```

```
border-width: 2px;
```

- radius:



```
border-top-left-radius: 15px;
```

```
border-top-left-radius: 15px;  
border-top-right-radius: 15px;
```

```
border-radius: 15px;
```

```
border-bottom-right-radius: 15px;
```

# Ejemplo de aplicación de estilos a una Tabla

The diagram illustrates how CSS styles are applied to a table. On the left, a block of CSS code is shown with arrows pointing to specific elements in the table on the right. The table has a caption 'Programa' and a header row 'Curso'. The first column is labeled 'Nombre' and the second column is labeled 'Duración'. Two rows of data follow: 'Persistencia' (24 horas) and 'Web services' (24 horas). The table is styled with a dark border and thin dotted blue borders between cells.

Curso	
Nombre	Duración
Persistencia	24 horas
Web services	24 horas

```
table {  
    border: medium solid #6B002A;  
    caption-side: bottom;  
}  
td, th {  
    font-family: Arial;  
    border: thin dotted blue;  
    padding-left: 15px;  
}  
caption {  
    padding-top: 20px;  
    font-style: italic;  
}
```

# Simplificación de estilos

- Algunas definiciones de estilos se pueden escribir de forma simplificada. Por ejemplo, el padding:

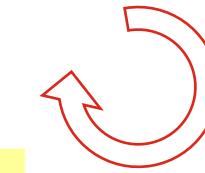
```
padding-top: 5px;  
padding-right: 15px;  
padding-bottom: 10px;  
padding-left: 30px;
```



```
padding: 5px 15px 10px 30px;
```

top right bottom left

orden "horario":



- El caso del margin es similar:

```
margin-top: 5px;  
margin-right: 15px;  
margin-bottom: 10px;  
margin-left: 30px;
```



```
margin: 5px 15px 10px 30px;
```

# Simplificación de estilos

- Si los cuatro valores son iguales, se puede colocar uno:

```
padding-top: 10px;  
padding-right: 10px;  
padding-bottom: 10px;  
padding-left: 10px;
```



```
padding: 10px;
```

- Similar es el caso de top-bottom y de left-right:

```
margin-top: 5px;  
margin-right: 15px;  
margin-bottom: 5px;  
margin-left: 15px;
```



```
margin: 5px 15px;
```

top-bottom

left-right

# Simplificación de estilos

- El caso del border también tiene un shorthand:

```
border-color: red;  
border-width: 2px;  
border-style: dashed;
```



```
border: red 2px dashed;
```

El orden no importa.

- En el caso de las fuentes, también:

```
font-size: 10pt;  
font-family: Arial, Helvetica, sans-serif;  
line-height: 1.2em;
```



```
font: 10pt/1.2em Arial, Helvetica, sans-serif;
```

font: font-style font-variant font-weight font-size/line-height font-family

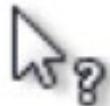
# Principales estilos: **cursor**

- A una sección del documento HTML se le puede cambiar la forma del cursor, utilizando estilos. Esto mejora la interacción con el usuario, indicando por ejemplo que debe esperar, o donde debe presionar:

`cursor: pointer;`



`cursor: help;`



`cursor: crosshair;`



`cursor: wait;`



`cursor: e-resize;`



`cursor: n-resize;`



`cursor: move;`



`cursor: progress;`



# **Combinación y herencia de estilos CSS**

# Combinación de estilos

- Si se quiere que un elemento HTML tenga los estilos gráficos definidos en varias clases, se pueden combinar declarándolas juntas:

```
<element class="class1 class2 class3">...</element>
```



Las clases de estilo se separan por espacio.

# Combinación de estilos

```
.fnt1 {  
    color: blue;  
}  
.fnt2 {  
    font-family: Arial, Helvetica, sans-serif;  
    font-size: 14pt;  
}
```

```
<p class="fnt1">Fuente 1</p>
```

```
<p class="fnt2">Fuente 2</p>
```

```
<p class="fnt1 fnt2">Fuente 1 2</p>
```

Fuente 1

Fuente 2

Fuente 1 2

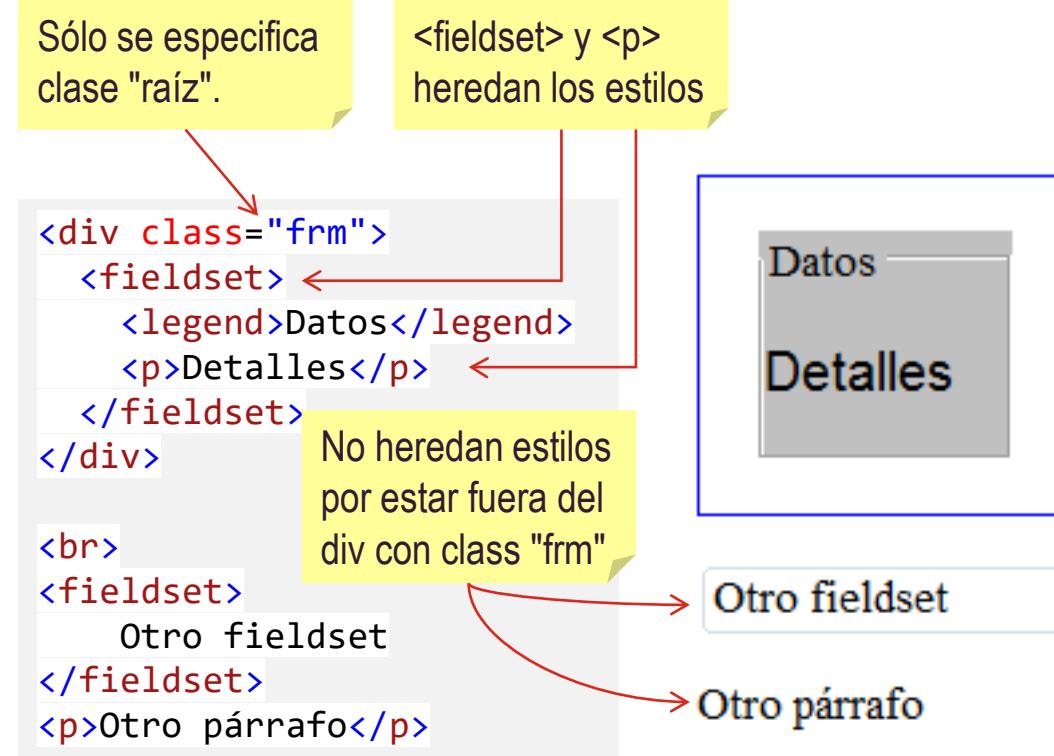
Combina atributos de  
las dos clases de estilo.

# Herencia de estilos

- En una estructura de etiquetas HTML, a través de CSS se pueden definir sus estilos, de modo que sólo se asigna la clase del elemento raíz. Ejemplo:

```
.frm {  
    border: 1px solid blue;  
    padding: 20px;  
}  
  
.frm fieldset {  
    background-color: silver;  
}  
  
.frm fieldset p {  
    font-family: Arial;  
    font-size: 1.2em;  
}
```

Estilo del <fieldset> dentro de un elemento con clase "frm".



# Herencia de estilos

- También se pueden especificar estilos de subelementos a través de su **id**, utilizando la nomenclatura con #:

```
.frm {  
    border: 1px solid blue;  
    padding: 20px;  
}  
.frm fieldset {  
    background-color: silver;  
}  
.frm fieldset #instr {  
    font-family: Verdana;  
    font-size: 1.2em;  
    color: navy;  
}  
.frm fieldset #details {  
    font-family: Arial;  
    font-size: 1.0em;  
    color: red;  
}
```

```
<div class="frm">  
  <fieldset>  
    <legend>Datos</legend>  
    <p id="instr">Instrucciones</p>  
    <p id="details">Detalles</p>  
  </fieldset>  
</div>
```



# Combinación y Herencia: Ejemplo

- Para las listas no ordenadas `<ul>`, se pueden definir estilos que permiten tener un aspecto gráfico similar a una barra de herramientas (toolbar). Aplicando herencia y los estilos vistos, resulta:

```
ul { list-style: none; } ← Elimina bullets
.tbar {
  width: 28%
}
.tbar ul li {
  float: left; ← Distribución horizontal de los <li>
  padding: 2%;
  border: 2px solid #fff;
}
.tbar ul li:hover {
  border: 2px solid #ccc; ← Mismo ancho
}
.tbar ul li a {
  padding: 2px; width: 100%; }
.tbar ul li a img {
  border: 0; ← Quita el borde (puesto por IE)
}
```

```
<div class="tbar">
  <ul>
    <li><a href="#" title="View"></a></li>
    <li><a href="#" title="Export xls"></a></li>
    <li><a href="#" title="New"></a></li>
  </ul>
</div>
```

Hyperlink a imagen.  
Simula un botón.



# Diseño Responsive

- Una página web responsive utiliza un despliegue visual dinámico, es decir, el contenido que se muestra **se ajusta a dispositivos** con diferentes capacidades, en especial, la manera en que se adapta a los diferentes tipos de pantalla.
- Puntos importantes:
  - Etiqueta viewport
  - Patrones de diseño responsive
  - Media queries
  - Breakpoints
  - Considerar si las imágenes son realmente necesarias





# ACTIVIDAD

Menú de navegación CSS

## OBJETIVO

Diseñar un menú de navegación mediante elementos de una lista e hipervínculos definiendo el aspecto utilizando reglas CSS.

## INSTRUCCIONES

1. Crear una lista sin orden con cuatro elementos: Inicio, Sobre nosotros, Servicios y Contacto. Esta lista será nuestro menú de navegación.

```
<ul>
  <li><a href='#'>Inicio</a></li>
  <li><a href='#'>Sobre nosotros</a></li>
  <li><a href='#'>Servicios</a></li>
  <li><a href='#'>Contacto</a></li>
</ul>
```



25 min

2. Crear un estilo CSS para todos los elementos de la lista de manera que el menú se muestre de forma horizontal.

```
li {  
    display: inline;  
}
```

3. Definimos ahora algunos de los estilos del menú: color de fuente y tipo, tamaño del menú, color de fondo e indicamos que el texto de los hipervínculos no se muestren subrayados.

```
li a {  
    text-decoration: none;  
    font-family: arial;  
    color: #fff;  
    background-color: #2175bc;  
    padding: 10px;  
    float: left;  
}  
/* La propiedad float va posicionando cada elemento del menú a  
la izquierda de su contenedor. */
```



25 min

- Añadimos un estilo para que, al pasar con el puntero del ratón por encima de cada elemento del menú, cambie su color de fondo.

```
li a:hover {  
    background-color: #2586d7;  
}
```



25 min

# Bibliografía y webgrafía



HTML5

<http://www.w3.org/TR/html5/>

CSS3

<https://jigsaw.w3.org/css-validator/>

World Wide Web Consortium (W3C)

<https://www.w3.org/>

# Bibliografía y webgrafía



Webs de referencia

<http://www.w3schools.com/>

<http://stackoverflow.com/>

<http://www.codecademy.com/>

02

# **UF1842. Desarrollo y reutilización de componentes de software multimedia mediante lenguajes de guión**

# Objetivos del curso



- Crear componentes software mediante herramientas y lenguajes de guión utilizando técnicas de desarrollo estructurado.
- Crear y manipular componentes multimedia utilizando lenguajes de guión y herramientas específicas.
- Seleccionar componentes de software ya desarrollados según su funcionalidad para integrarlos en documentos.

1. Arquitecturas de aplicaciones web
2. Navegadores web
3. Creación de contenido web dinámico
4. Lenguajes de guión de uso general
5. Contenido multimedia
6. Single Page Application (SPA)

Bibliografía y webgrafía

02

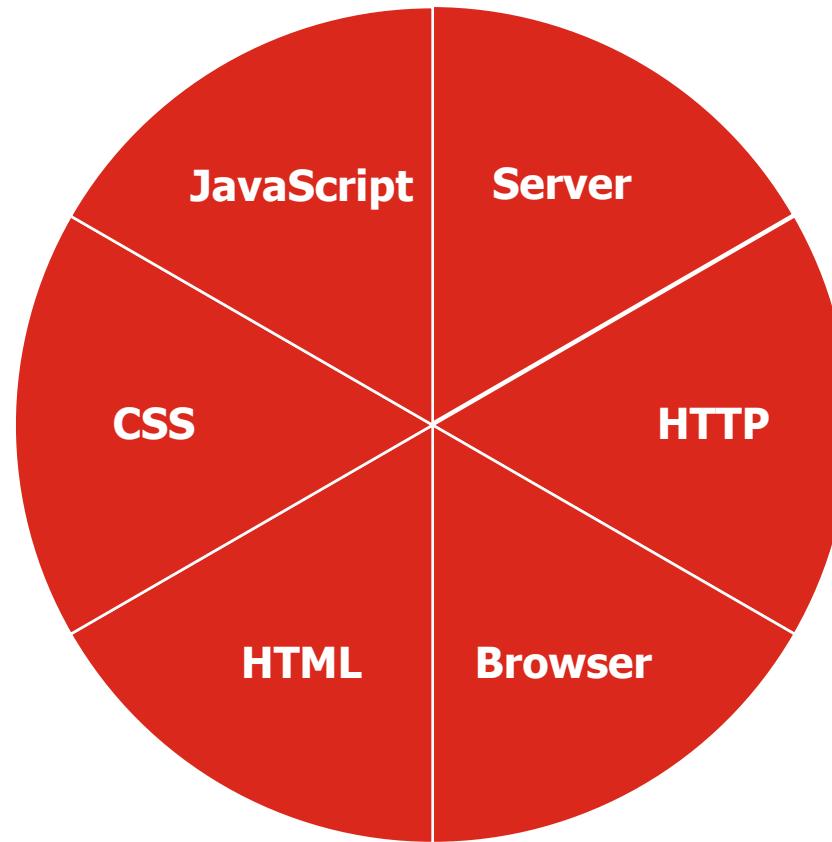
# Desarrollo y reutilización de componentes de software multimedia mediante lenguajes de guión

## 1. Arquitecturas de aplicaciones web

- Esquema general
- Arquitectura en capas
- Interacción entre las capas cliente y servidor
- Arquitectura de la capa cliente

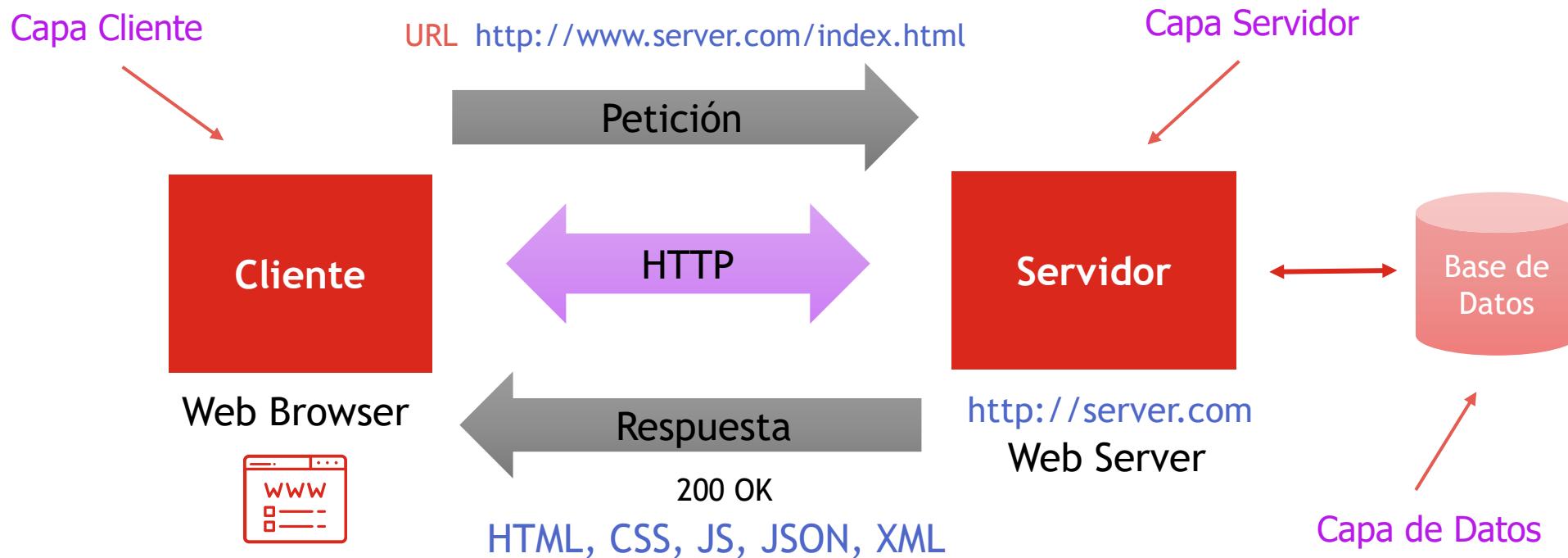
# Arquitectura básica de una aplicación web

- Para que una **aplicación web** esté funcionando en internet es necesaria una **arquitectura** que disponga como mínimo de los siguientes elementos:



# Modelo Cliente – Servidor

- Interacción entre capas cliente-servidor y servidor-datos: esquema general



# Principal protocolo de internet: HTTP

- **HTTP** (HyperText Transfer Protocol), protocolo de transferencia de hipertexto.
- Permite el intercambio de datos entre un cliente (navegador) y un servidor.

Ejemplo simplificado de **mensajes HTTP** que se intercambian cliente y servidor:

## Request

```
GET /app/pagina.html HTTP/1.1  
Host: www.netmind.es
```

## Response

```
HTTP/1.1 200 OK  
Server: ...  
Content-Type: ...  
  
<html>  
  <head>...</head>  
  <body>...</body>  
</html>
```

# Uniform Resource Locator: URL

La **URL** (Uniform Resource Locator) es una cadena alfanumérica que se corresponde con un recurso web.

- Formato de la URL  
protocol://path/to/resource
- URL que se refiere a un archivo local:  
file://path/to/resource

Ejemplo en Windows:

file:///C:/administracion/docu-basic/indice.html

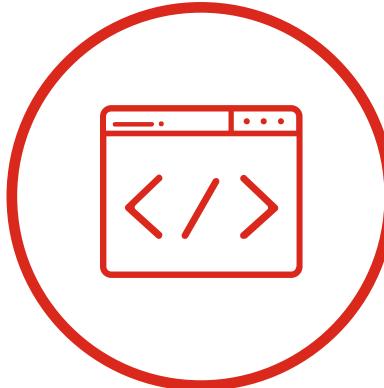
- Formato de URL remota, ubicación en un servidor basado en el protocolo HTTP:  
**http://servidor:puerto/application/path/to/resource**

Protocolo más  
utilizado en web.

El puerto permite que desde una misma máquina se puedan  
publicar varios servicios. Puerto http por defecto es el 80.

# Tecnologías de la capa Cliente

- Permiten crear e implementar una aplicación web del lado del cliente



- **HTML:** información estructurada en secciones, párrafos, título, imágenes, etc... y ofrece muchas librerías para añadir contenidos multimedia, canvas, comunicaciones y concurrencia.
- **CSS:** distribución de elementos y su estilo con colores, tipos de letra, fondos, efectos, etc...en documentos HTML, XML, SVG o incluso interfaces de usuario de otras tecnologías.
- **Scripting:** principalmente se utiliza JavaScript permitiendo comportamiento dinámico y funcionalmente orientado a objetos. Multitud de bibliotecas (APIS) para el desarrollo web y de aplicaciones.
- **DOM:** modelo de objetos del documento (Document Object Model) para manipular el documento HTML en tiempo real, permitiendo la gestión de eventos mediante de una librería (API).
- **AJAX:** Asynchronous JavaScript And XML), mejora la experiencia de usuario permitiendo la carga de nuevos elementos sin tener que cargar toda la página de nuevo.

# Tecnologías de la capa Servidor

- Permiten crear e implementar una aplicación web del lado del servidor



- **Java:** uno de los lenguajes de programación más utilizados para desarrollar el backend de una aplicación web.
- **Node JS:** entorno de ejecución multiplataforma de código abierto del lado del servidor basado en el lenguaje de programación JavaScript.
- **PHP:** lenguaje de programación con licencia libre, multiplataforma y se integra con Apache y MySQL. Actualmente ha perdido popularidad en su utilización.
- **Python:** lenguaje de programación multiparadigma para desarrollar software multiplataforma utilizado en Big Data, videojuegos, web scraping, Inteligencia Artificial y acceso a APIs.
- Django, Groovy, ASP .NET, Java EE, ...

# Tecnologías de la capa de Datos

- Sistemas de persistencia de datos e información manejada por la aplicación web



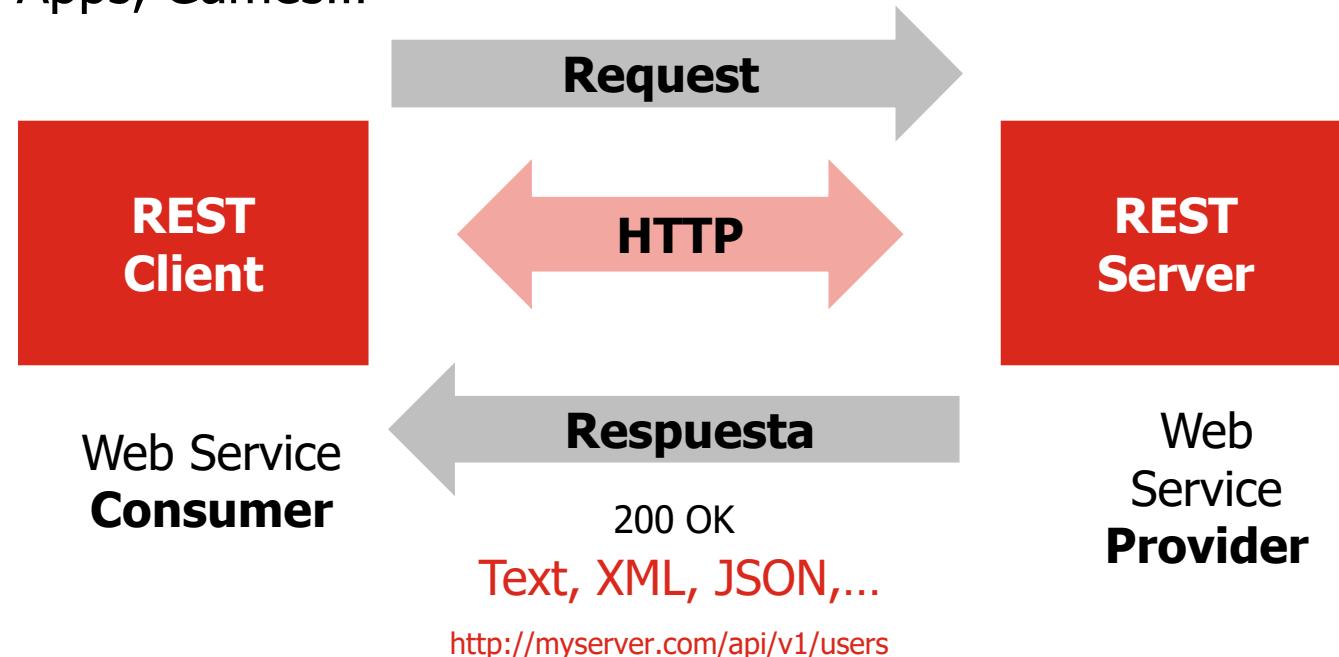
.....

- **Bases de datos relacionales:** MySQL, Oracle, MS SQL Server, PostgreSQL...
- **Bases de datos no relacionales:** Mongo DB, Cassandra, riak, redis...

# Arquitectura de Microservicios

**Servicio web:** API online a la que se puede acceder programáticamente

- Código distribuido en plataformas y servicios cloud
- Formatos de intercambio de información: JSON y XML
- Facebook y Twitter publican web services consumidos por otros desarrolladores desde sus códigos
- Aplicaciones: Apps, Games...



# Aplicación web vs Servicio web

<http://www.twitter.com>



HTML, CSS, JS



<http://api.twitter.com>



JSON / JSON

**JSON**

```
{  
  "siblings": [  
    {"firstName": "Anna", "lastName": "Clayton"},  
    {"lastName": "Alex", "lastName": "Clayton"}  
  ]  
}
```

**XML**

```
<siblings>  
  < sibling >  
    < firstName >Anna</firstName>  
    < lastName >Clayton</lastName>  
  </ sibling >  
  < sibling >  
    < firstName >Alex</firstName>  
    < lastName >Clayton</lastName>  
  </ sibling >  
</ siblings >
```

02

# Desarrollo y reutilización de componentes de software multimedia mediante lenguajes de guión

## 2. Navegadores web

- Arquitectura de un navegador
- Comparativa de navegadores más extendidos
- Seguridad en navegadores
- Integración de aplicaciones en navegadores
- Adaptadores o plugins
- Conformidad a estándares

# Navegadores web

## Características

- Un navegador web o browser es un software que se utiliza para buscar, recuperar y mostrar información, principalmente desde la Internet.
- Representa el concepto de **cliente** solicitando recursos a diferentes servidores web mediante peticiones HTTP.
- Los recursos se obtienen a través de un identificador, llamado genéricamente **URI** (Uniform Resource Identifier).

**Chrome**  
(Google)



**Firefox**  
(Mozilla)



**Opera**  
(Opera Software)



**Edge**  
(Microsoft)



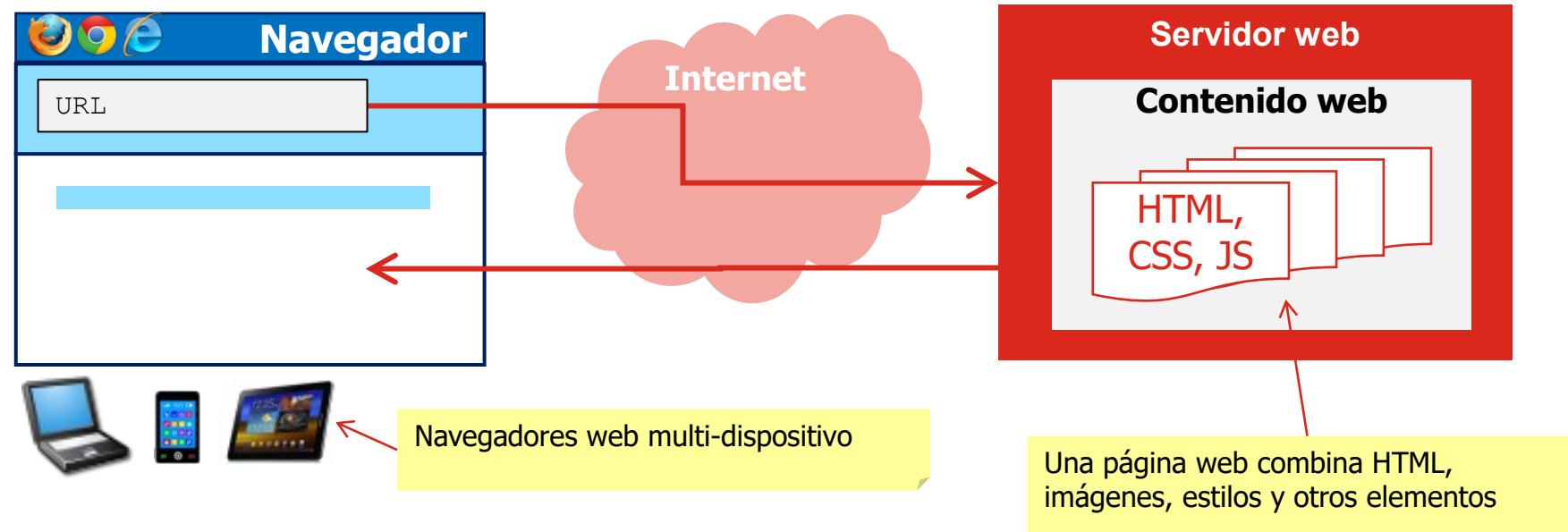
**Safari**  
(Apple)



# Navegadores web

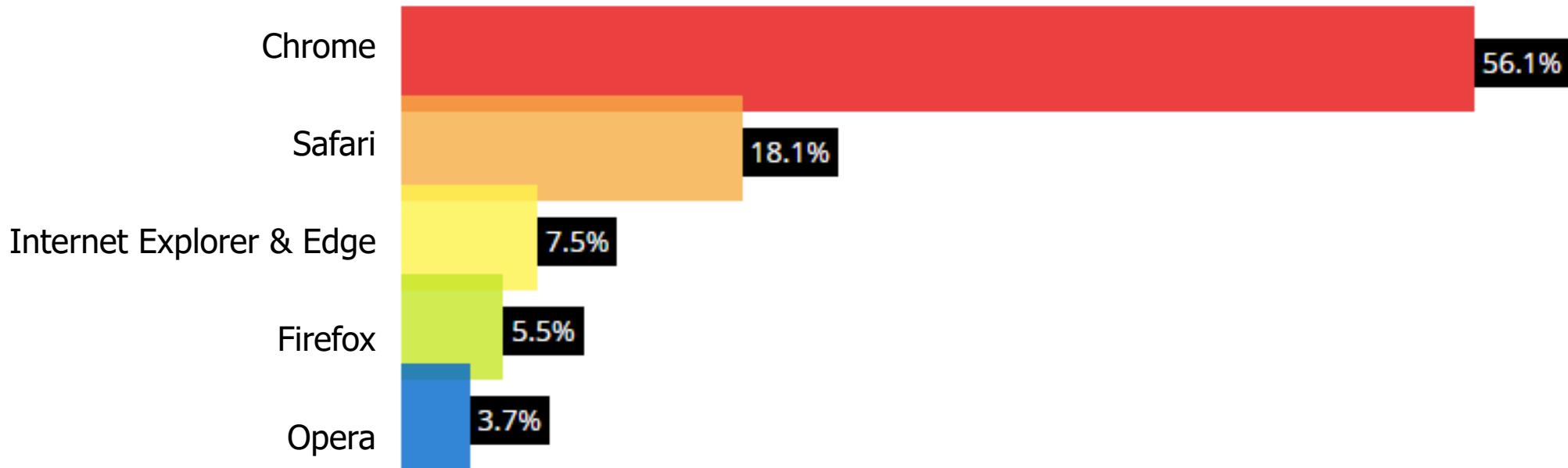
## Arquitectura y funcionamiento

- Desde un navegador web (lado cliente), se hace la petición de un recurso al servidor a través de su **URL**
- El servidor responde a la petición con el contenido web solicitado por el cliente, el cual es enviado al navegador, quien lo interpreta y muestra.



# Navegadores web

Preferencia de navegadores



Fuente: W3Counter, dic 2020

# Navegadores web

## Versiones más utilizadas

1	Chrome 78	22,37%
2	Chrome 79	15,65%
3	Safari 13	11,44%
4	Safari 12	5,02%
5	IE 18	3,07%
6	UC 12	2,62%
7	Firefox 71	2,37%
8	IE 11	2,30%
9	Samsung 10	2,26%
10	Chrome 77	1,86%

Fuente: W3Counter, dic 2020

# Navegadores web

## Comparativa: Detalles técnicos

DETALLES SOFTWARE	Google Chrome	Safari	Firefox	Internet Explorer	Microsoft Edge	Opera
Plataforma	WebKit	WebKit	Gecko	Trident	EdgeHTML	WebKit
Motor de JavaScript	V8	Nitro	TraceMonkey	Chakra	Chakra	Carakan
Código abierto	Sí	No	Sí	No	No	No
Página web	<a href="#">Google Chrome</a>	<a href="#">Safari</a>	<a href="#">Firefox</a>	<a href="#">Internet Explorer</a>	<a href="#">Microsoft Edge</a>	<a href="#">Opera</a>

Fuente: PCWorld – Enero 2021

# Navegadores web

## Comparativa: Funcionalidad

FUNCIONES	Google Chrome	Safari	Firefox	Internet Explorer	Microsoft Edge	Opera
Sincronización en la nube	Sí	Sí	Sí	No	Sí	Sí
Gestor de descargas	Sí	Sí	Sí	Sí	Sí	Sí
Navegación privada	Sí	Sí	Sí	Sí	Sí	Sí
Modo de pantalla completa	Sí	No	Sí	Sí	Sí	Sí
Pestañas verticales	Sí	No	Sí	No	No	Sí
Extensiones personalizadas	Sí	No	Sí	Sí	No	Sí

Fuente: PCWorld – Enero 2021

# Navegadores web

## Comparativa: Compatibilidad

COMPATIBILIDAD	Google Chrome	Safari	Firefox	Internet Explorer	Microsoft Edge	Opera
Windows	Sí	Sí	Sí	Sí	Sí	Sí
macOS	Sí	Sí	Sí	No	No	Sí
Linux	Sí	No	Sí	No	No	Sí
Android	Sí	No	Sí	No	No	Sí
iOS	Sí	Sí	Sí	No	No	Sí
Windows Phone	No	No	No	Sí	Sí	Sí

Fuente: PCWorld – Enero 2021

# Navegadores web

## Extensiones (plugins)

- Los complementos o **plugins** son pequeños programas que extienden las características y funcionalidades de los navegadores
- Gran variedad de complementos disponibles: funciones de diseño, mejora de la privacidad y de la seguridad, traducción automática del sitio al idioma que deseas, etc...
- Cuestiones importantes a tener en cuenta:
  - Descargar siempre versiones actualizadas
  - Revisa los permisos y accesos que te pide al instalarlo: por ejemplo, ¿qué datos personales solicita? ¿pide acceso a tu cámara?
  - Busca reseñas y revisa las calificaciones: ¿quiénes desarrollan y qué hacen con la información que extraen?

# Navegadores web

## Complementos (plugins) de Seguridad

<b>TunnelBear</b>  Es una VPN: <b>hace que tu información viaje de manera segura y cifrada</b> , y como en un túnel privado, oculta tu conexión y actividad en línea para que no pueda ser vista por alguien más.	<b>Disponible para</b>    	<b>Privacy Badger</b>  Evita que anunciantes y terceros sepan desde donde navegas y qué páginas visitas en la web, este tipo de plugin los bloquea y te hace invisible	<b>Disponible para</b>   
<b>ScriptSafe</b>  Ambos bloquean la ejecución de código Javascript, Java, Flash, Silverlight y otros tipos de scripts: esto <b>previene que se ejecuten acciones que no quieras en tu navegador</b> .	<b>Disponible para</b>  	<b>uBlock Origin</b>  Es un bloqueador de amplio espectro que además <b>bloquea sitios que han sido clasificados como malware</b> . Este plugin y Privacy Badger funcionan mejor juntos.	<b>Disponible para</b>   
<b>NoScript</b> 	<b>Disponible para</b>  	<b>HTTPS Everywhere</b>  Forza a que los sitios web utilicen https://, es decir, tu información viaja segura y cifrada, y te protege incluso si el sitio no lo hace.	<b>Disponible para</b>   

02

# Desarrollo y reutilización de componentes de software multimedia mediante lenguajes de guión

## 3. Creación de contenido web dinámico

- Fundamentos de programación
- Lenguajes para el desarrollo de contenido dinámico
- Typescript

# **Fundamentos de Programación**

# Instrucciones

- **Programa** = secuencia ordenada de **instrucciones** que se ejecutan para obtener un **resultado final**.
- Una **instrucción** o sentencia expresa una orden que se le da al programa.
- La instrucción como una **expresión**:

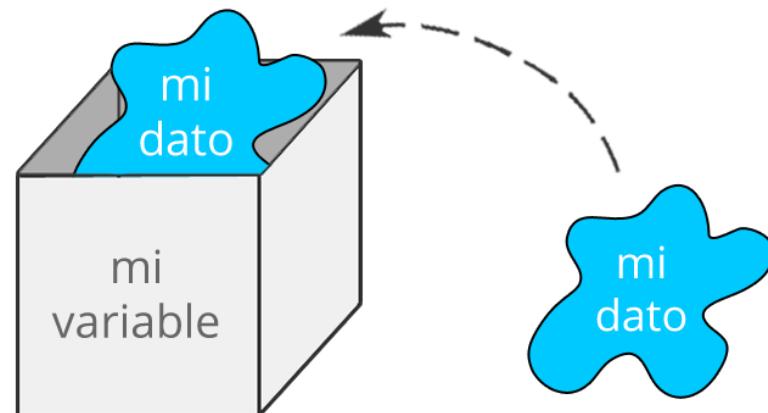
importeNeto = importeBruto - descuento

Resultado  
Operando o Variables



# Variables y Constantes

- Una variable es un **identificador que guarda o contiene un valor de un tipo de dato concreto** (número, cadena alfanumérica...).
- El valor de una variable va cambiando a lo largo de la ejecución del programa mientras que una **constante** no cambia nunca.
- Es de buenas prácticas definir o declarar las variables antes de utilizarlas o referenciarlas.
  - Declaración:
    - var ejemplo;
    - var contador, i;
  - Declaración y asignación:
    - var ejemplo = "Hola";
    - var contador = 1;
    - var i = 0;
  - Asignación:
    - ejemplo = "Hola";
    - contador = contador + 1;

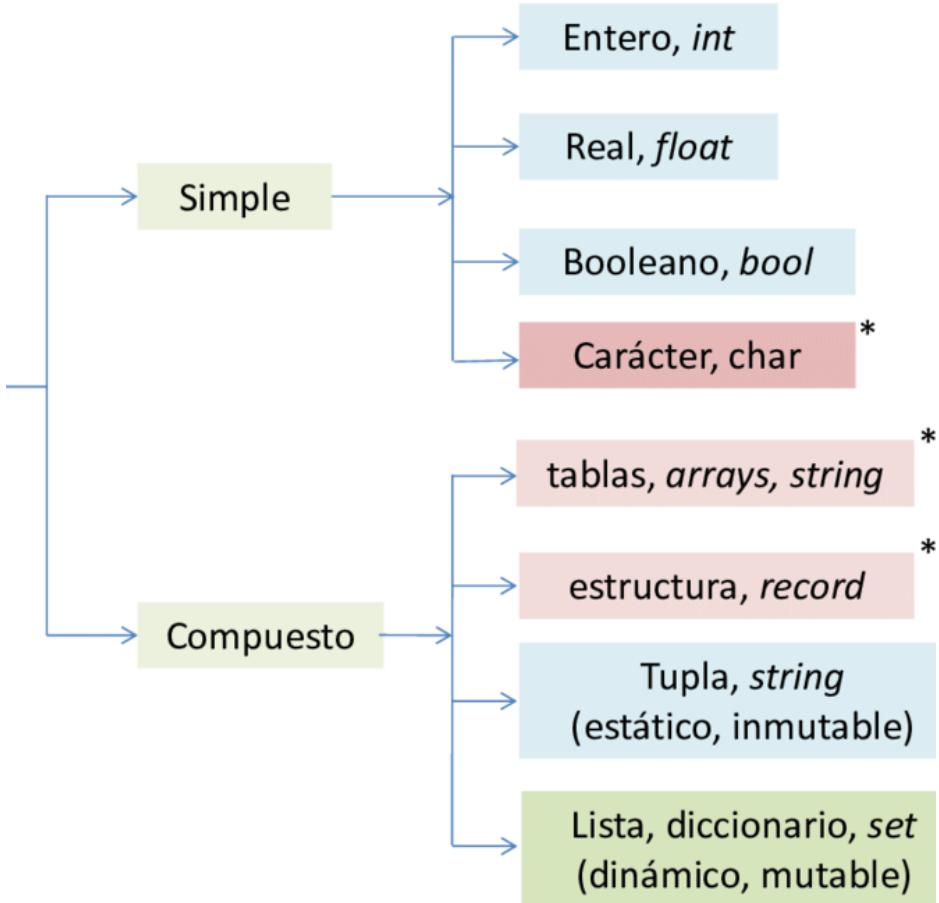


# Tipos de datos

- Para representar la información:

- Números
  - 17, -100
  - 5.6, 300.00
- Cadenas alfanuméricas
  - "¡Hola!"
  - 'Buenos días'
- Lógicos o Booleanos
  - True o Verdadero
  - False o Falso
- Arrays
  - [1, 6, 0, 3, 9, a, H, 3, Z, e, i]
  - {lunes, martes, miércoles, jueves, viernes}

Tipos  
de  
datos



# Expresiones

- Una **expresión** es un conjunto de **operandos** y **operadores** que devuelven un resultado.
- Tipos de expresiones: **aritméticas** y **lógicas**.

Expresiones  
Aritméticas

Expresión	Significado	Ejemplo	Resultado	Operador	Operandos
$a + b$	Suma	$8 + 1$	9	+	a, b
$a - b$	Resta	$0 - 6$	-6	-	a, b
$x * y$	Multiplicación	$7 * 2$	14	*	x, y
$x / y$	División	$3 / 2$	1.5	/	x, y

Expresiones  
Lógicas

Expresión	Significado	Ejemplo	Resultado	Operador	Operandos
$a > b$	Mayor que	$5 > 2$	Verdadero	>	a, b
$a < b$	Menor que	$8 < 1$	Falso	<	a, b
$a \geq b$	Mayor o igual que	$3 \geq 3$	Verdadero	$\geq$	a, b
$a \leq b$	Menor o igual que	$6 \leq 4$	Falso	$\leq$	a, b
$a = b$	Igual que	$7 = 9$	Falso	=	a, b
$a \neq b$	Distinto que	$1 \neq 0$	Verdadero	$\neq$	a, b

# Expresiones de asignación

**variable = expresión**

Evalúa la expresión, y el resultado se asigna a la variable

**variable += expresión**

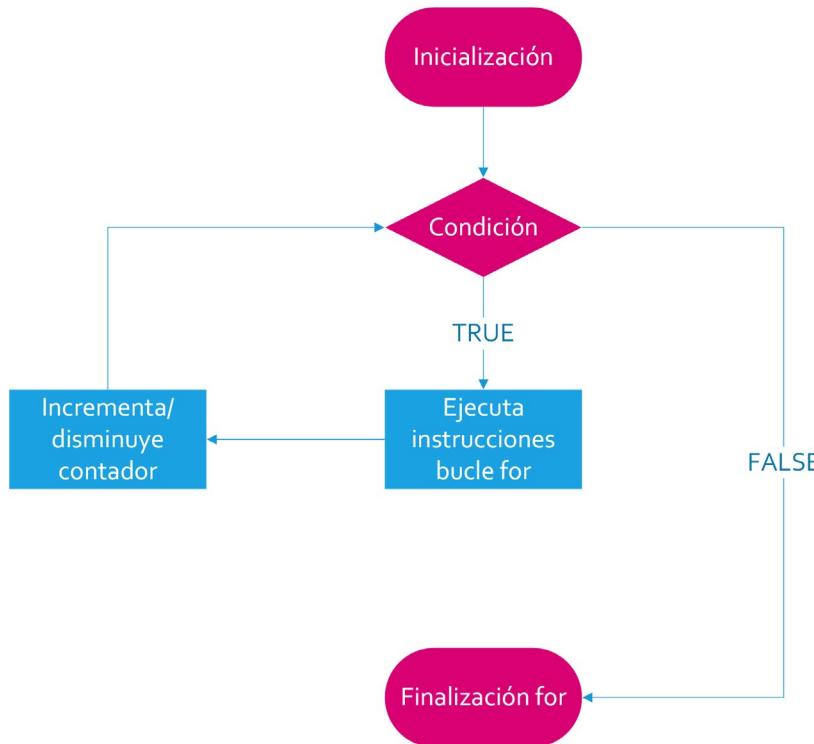
Evalúa la expresión, y el resultado más el valor de la variable se asigna a la variable

Operadores  
Lógicos

Operador	Significado	Ejemplo
AND	"y" lógico, devuelve verdadero cuando ambos operandos son verdaderos y falso en otro caso	V and F = F
OR	"o" lógico, devuelve verdadero cuando alguno de los operandos es verdadero	V or F = V
NOT	"no" lógico, devuelve verdadero si el operando es falso, y falso si el operando es verdadero	not V = F

# Control del flujo del programa

- Tipos de instrucciones para controlar el flujo de ejecución de un programa:
  - Instrucciones Secuenciales
  - Instrucciones Selectivas o Condicionales
  - Instrucciones Repetitivas o Iterativas (bucles o loops)



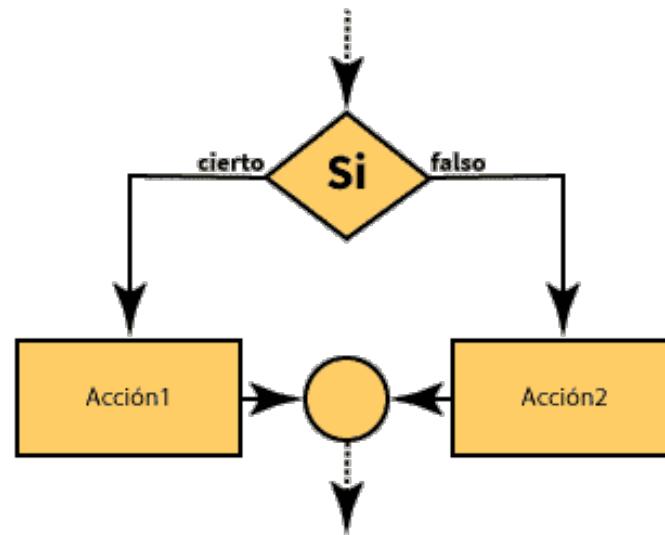
# Instrucciones Condicionales: If

- La instrucción condicional IF permite ejecutar un bloque instrucciones dependiendo de si cumple o no una determinada condición.

```
if (<condición>) {  
    <instrucciones>  
}
```

Ejemplo:

```
if (a == b) {  
    alert("a y b son iguales.");  
}
```



# Instrucciones Condicionales: If else

- El condicional IF ELSE considera el caso de que no se cumpla la condición.

```
if (<condición>) {  
    <instrucciones>  
} else {  
    <instrucciones>  
}
```

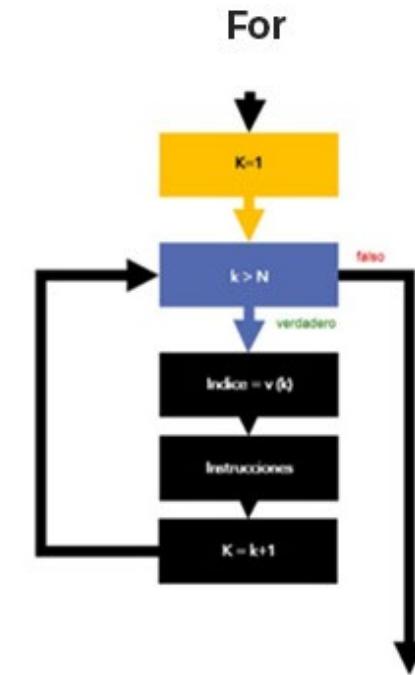
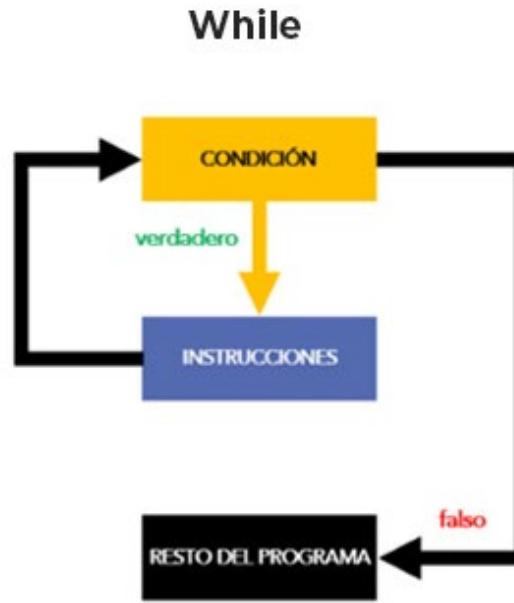
Ejemplo:

```
if (a > b) {  
    alert("A es mayor");  
} else {  
    alert("A no es mayor");  
}
```

```
if (<condición1>) {  
    <instrucciones>  
} else if (<condición2>) {  
    <instrucciones>  
}
```

```
// varias condiciones else if  
  
} else if (<condición>) {  
    <instrucciones>  
} else {  
    <instrucciones>  
}
```

# Instrucciones Iterativas o Bucles: While, Do While, For



# Instrucciones Iterativas: While, Do While

## Sintaxis en JavaScript

```
while (<condición>) {  
    <instrucciones>  
}
```

Ejemplo:

```
var a = 3;  
  
while (a > 0) {  
    alert(a);  
    a--;  
}
```

```
do {  
    <instrucciones>  
} while (<condición>);
```

Ejemplo:

```
var a = 0;  
  
do {  
    alert(a);  
    a--;  
} while (a > 0);
```

Se ejecuta al menos una vez

# Instrucciones Iterativas: For

Sintaxis en JavaScript

```
for (<declaración>; <condición>; <instrucción>) {  
    <instrucciones>  
}
```

Ejemplo:

```
for (var i = 0; i < 3; i++) {  
    alert(i);  
}
```

```
for (var <índice> in <iterable>) {  
    <instrucciones>  
}
```

Ejemplo:

```
var v = [4,7,9,1];  
  
for (var a in v){  
    alert(v[a]);  
}
```

# Funciones

- Una **función** agrupa un bloque de instrucciones bajo un nombre (el de la función) que realiza una tarea específica.
- Puede devolver o retornar un valor para ser usado en otra parte del programa.
- Una función puede recibir valores de entrada, llamados **parámetros**, que la función puede procesar internamente
- Las funciones fomentan la **división y reutilización** del código.

```
function nombre_funcion(param1,..., paramN) {  
    instrucciones;  
    return valor;  
}
```

Función sin  
parámetros que no  
devuelve ningún valor



```
function Saludar(){  
    alert("Hola a tod@s");  
}
```

# Funciones

- Ejemplo de función que recibe como parámetros dos números y realiza la suma de ellos.
- La función **retorna** o devuelve el valor de la suma.

```
function sumar(param1, param2) {  
    var suma = param1 + param2;  
    return suma;  
}
```

Declaración  
de la función

```
var total = sumar(100, 50);  
// bloque de instrucciones del programa  
var total = sumar(500, 700);
```

Llamadas  
a la función

# Funciones

- Función asignada a una variable:

```
var suma = function(a, b) {  
    return a * b;  
}
```

Declaración  
de la función

```
alert(suma(7, 9));
```

Llamada  
a la  
función

# Objetos

## POO (Programación Orientada a Objetos):

Paradigma de programación en la que todo se representa como una entidad u objeto de la vida real o imaginario.

En su definición se especifican:

- **Propiedades:** características que distinguen a los objetos del mismo tipo o *Clase*.
- **Métodos:** funcionalidades del objeto. Tareas que se pueden realizar con las propiedades de un objeto.
  - Superclase: Teléfonos
  - Clase: Teléfono celular
  - Subclase: SmartPhone



### Propiedades o atributos

- Tipo de pantalla
- Espacio de memoria
- Cantidad de tonos
- Tipo de antena
- Cantidad de idiomas
- Otros

### Métodos o comportamientos

- Iniciar alarma
- Asignar tonos
- Registrar llamadas
- Iniciar juego
- Utilizar calculadora
- Enviar mensajes
- Otros

"La orientación a objetos será la más importante de las tecnologías que surjan en los años noventa"

Bill Gates

# Clase o Tipo

- Una **clase** especifica qué propiedades y métodos caracterizan a sus objetos, pero no asigna valores a sus miembros.

- Automóvil
  - Marca
  - Modelo
- Persona
  - Nombre
  - Edad
- Estudiante
  - Nivel
  - Curso
  - Especialidad

Ejemplos de clases  
y sus propiedades



## Persona

- Nombre (Nombre, apellido paterno, apellido materno)
- Edad
- Género
- Intereses

# Objeto o Instancia

- Un **objeto** es una instancia de clase definida, con valores en su propiedades o atributos.

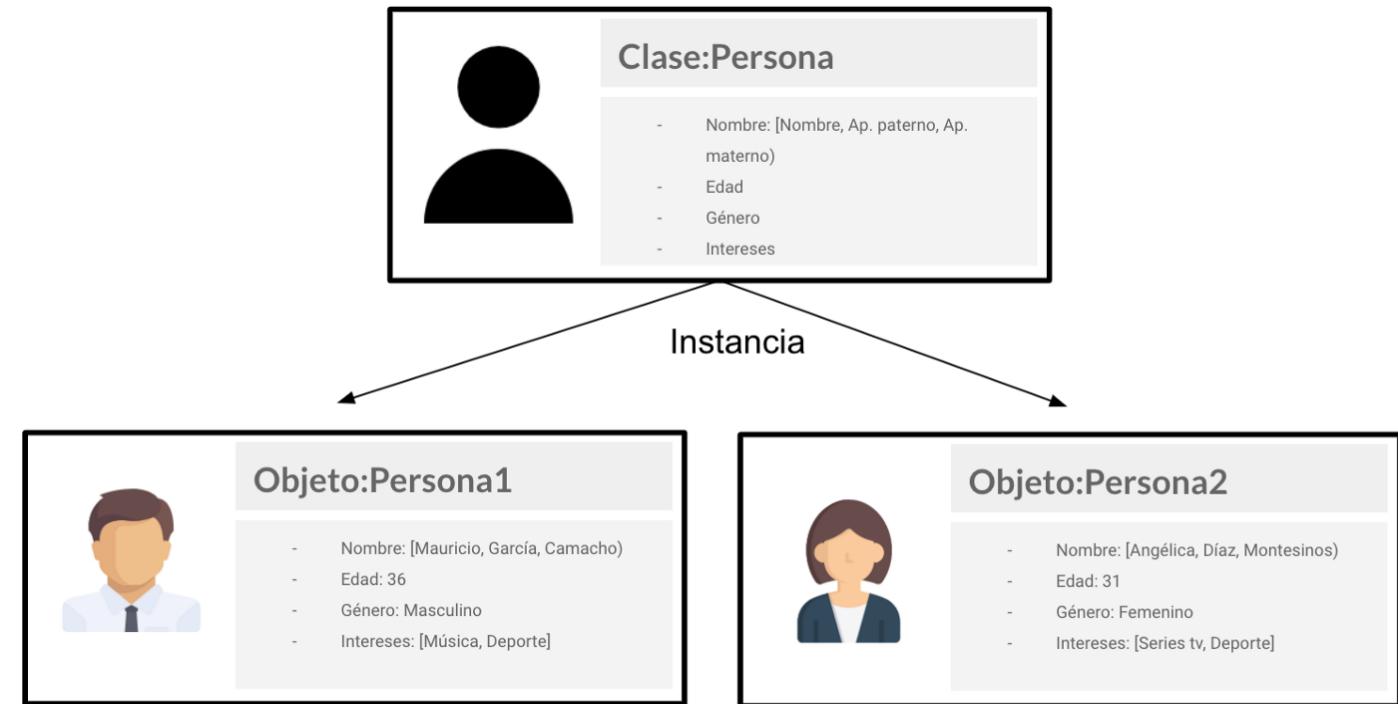
- Persona

- Nombre: Luis
- Edad: 36

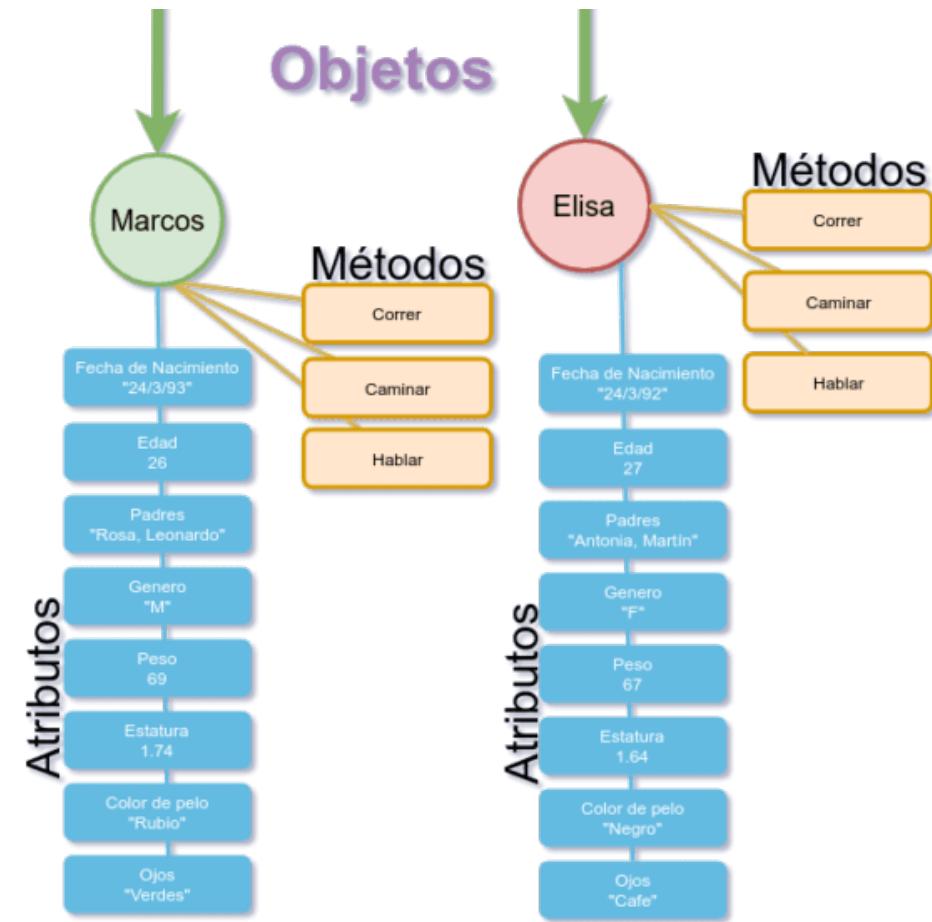
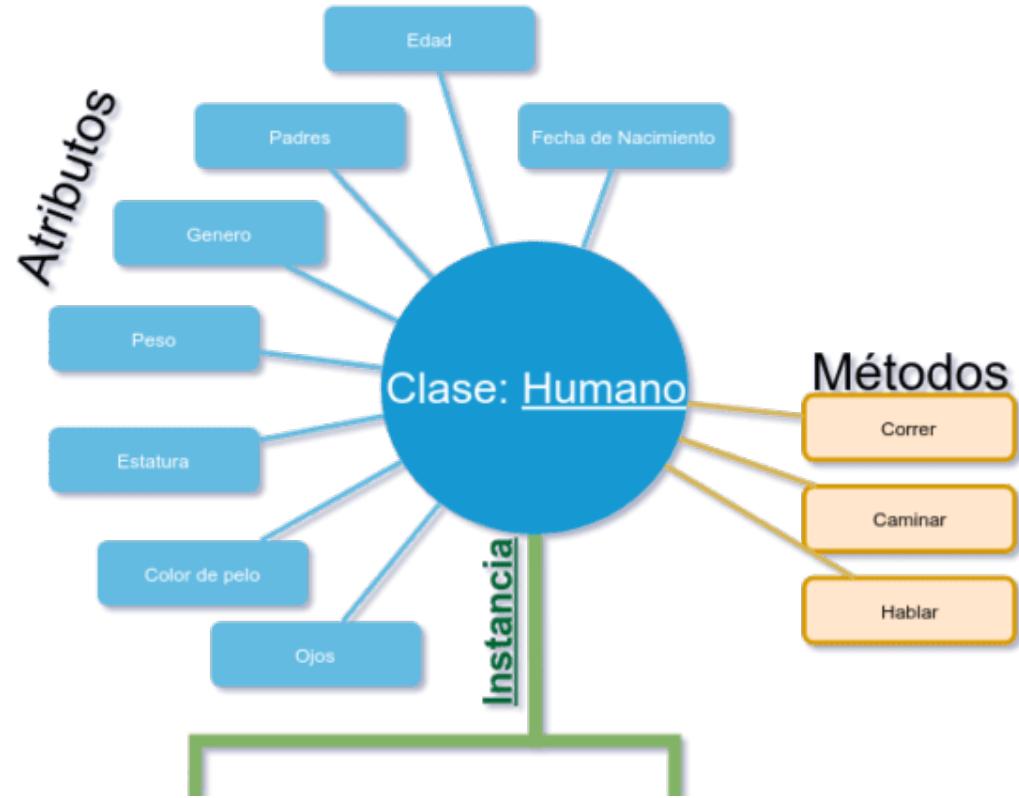
- Alumno

- Nombre: Juan
- Edad: 19
- Carrera: Diseño Gráfico

Ejemplos de objetos y sus estados  
(valores de sus propiedades)



# Clase e Instancia



# Herencia y Jerarquía de clases



# Metodologías de desarrollo de software

- Una metodología de programación es un **framework** (marco de trabajo) para **estructurar, planear y controlar** el ciclo de desarrollo de software.

## Tradicionales

Waterfall o En Cascada

Espiral

Prototipado

## Agile

Scrum

Kanban

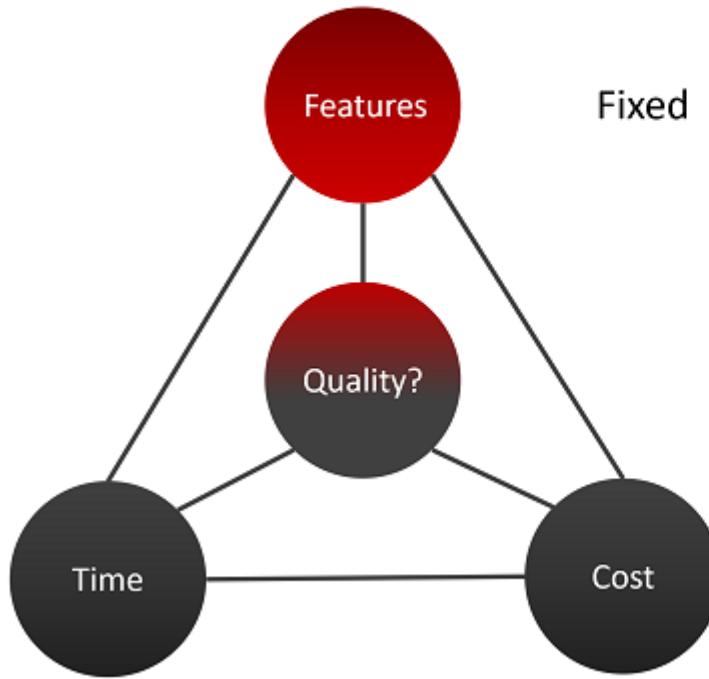
XP: eXtreme Programming

Lean

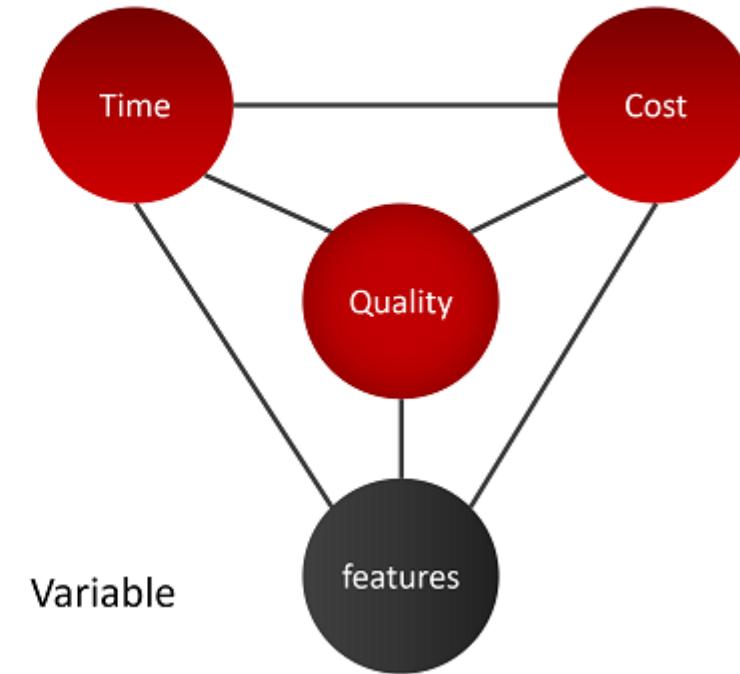
Scrumban

# Metodologías de desarrollo de software

Traditional Approach



Agile Approach



# Metodologías Ágiles vs Tradicionales

Metodologías Agiles	Metodologías Tradicionales
Más valor al equipo	Más valor al proceso
Proceso menos controlado	Etapas del proceso predefinidas
El cliente parte del equipo	Cliente solo en la etapa inicial y final
Equipos multidisciplinares	Equipos independientes
Pocos roles	Roles muy definidos
Mejora continua - etapas	Acciones orientadas a resultado final

# Principales M todolog as  giles

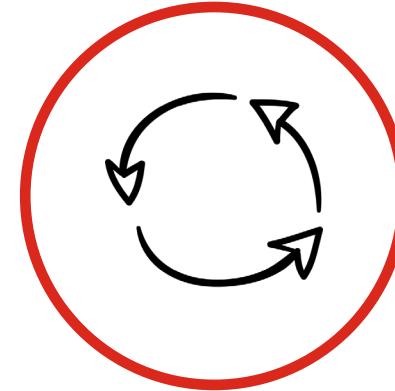
Kanban



Lean



Scrum



eXtreme  
programming



- Mejora el flujo de trabajo
- L mita el WIP (Work In Progress)
- Liderazgo en todos los niveles
- Funciona bien en flujos de trabajo definidos

- Revisi n de lo que no a ade valor (waste) al cliente
- Filosof a Kaizen
- Producto final testeado
- Retroalimentaci n del equipo

- Iteraciones, entregas, incrementos hasta terminado
- Roles m s definidos: Product Owner, Scrum Master, Developers Team
- Cliente siempre involucrado
- Equipo multidisc y auto organiz
- Artefactos: Product Backlog, Sprint Backlog

- Organizaci n
- Adaptable a las circunstancias
- Pair programming

Poco  
restrictiva

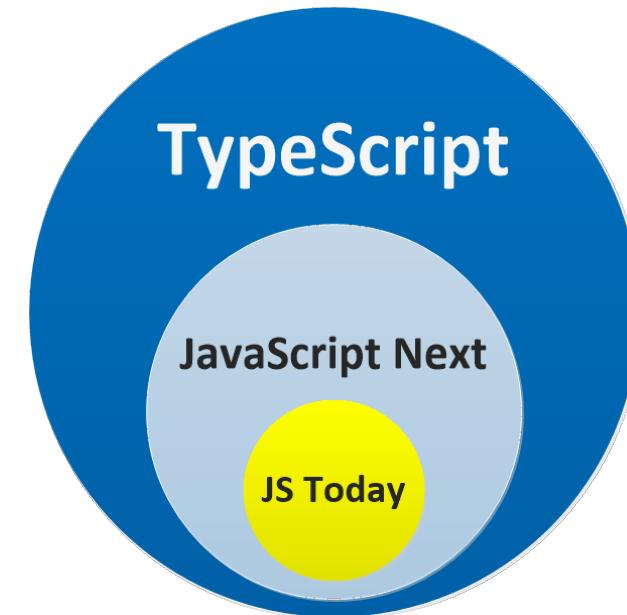
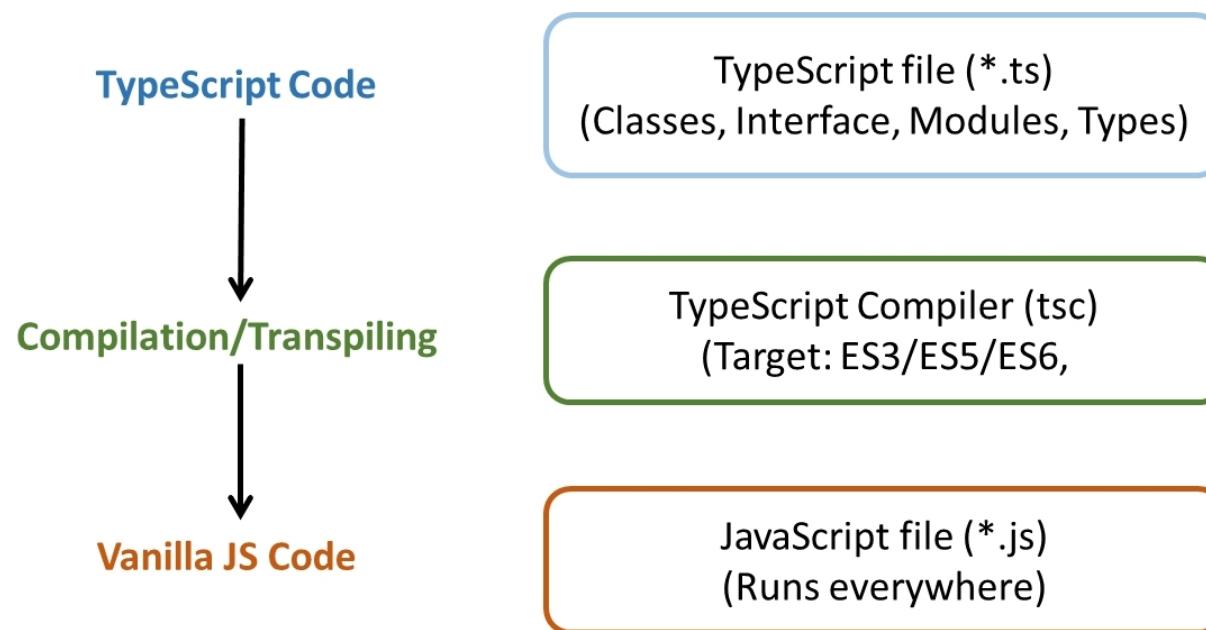


Muy  
restrictiva

# **Lenguajes para el desarrollo de contenido dinámico**

# Lenguajes de Scripts

- Un **script** en el lado del cliente es un programa que puede acompañar a un documento HTML o estar contenido dentro del mismo.
- Las instrucciones del script se ejecutan cuando se **carga en el navegador** o al producirse algún **evento de usuario** como p.e. el hacer clic en un enlace.
- Permiten actualizar dinámicamente el contenido del documento HTML, modificar el comportamiento predeterminado del navegador, validar formularios, cambiar el aspecto visual de los elementos HTML...



# Lenguajes de Scripts

Lenguajes de scripts	Lenguajes de programación
Plataforma específica	Plataforma agnóstica (multiplataforma)
Principalmente interpretado	Compilado
Más rápido en tiempo de ejecución	Más lento en el tiempo de ejecución
Más intensivo en codificación	Menos intensivo en codificación
Crea aplicaciones autónomas	Crea aplicaciones como parte de una pila

# Un ejemplo de script

```
<html>
<head><title>JavaScript sample</title>
</head>
<body>
    <!-- bloque de código de script -->
    <script language="JavaScript">
        document.writeln("Funciona!<br>");
    </script>
</body>
</html>
```

# Algunos lenguajes de scripts

Principales lenguajes de scripting que son actualizados regularmente y que también se utilizan en producción (ver características, casos y entorno de uso [aquí](#)):

- JavaScript / ECMAScript
- PHP
- Python
- Ruby
- Goovy
- Perl
- Lua
- Bash
- PowerShell
- R
- VBA
- Emacs Lisp
- GML

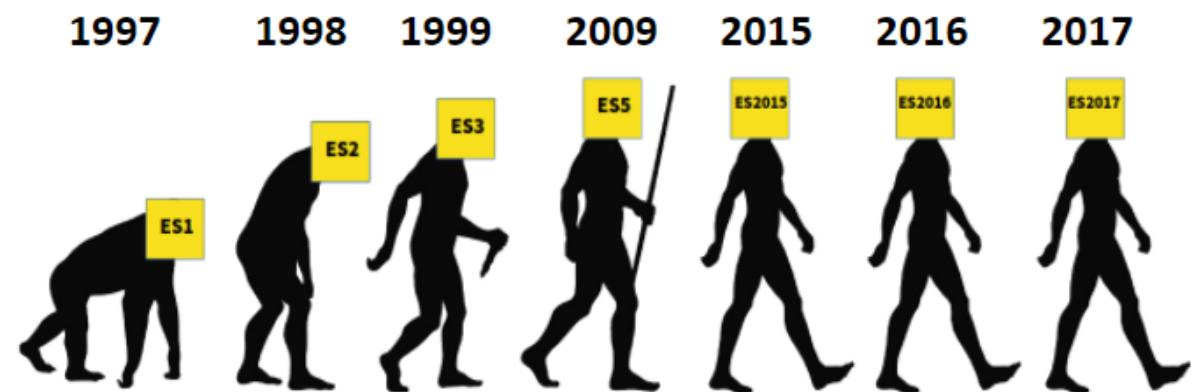


# TypeScript

# TypeScript

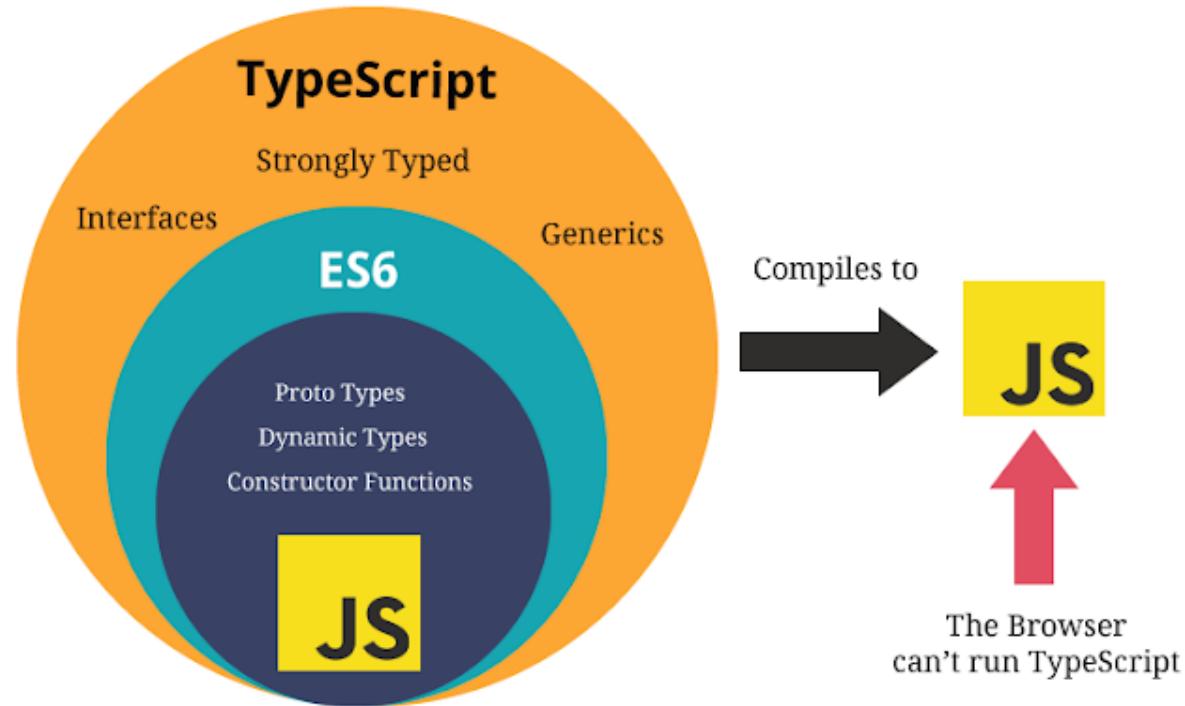
Superconjunto de JavaScript

- 85% JavaScript convencional
- **Open Source**
- Permite programar en una **sintaxis fuertemente tipada**
- **Orientación a objetos** soportada
- Lenguaje interpretado (se **transpila** a JavaScript)
- Incluye el propio estándar **ECMAScript**
- Extensión de los archivos **.ts**



# TypeScript

## Contextualización



# TypeScript

## Tipos básicos

- Boolean
- Number
- String
- Array
- Tupla
- Enum
- Any
- Void
- Null e indefinido
- Never
- Object
- Aserciones

```
let isDone: boolean = false;  
  
let decimal: number = 6;  
  
let color: string = "blue";  
  
let list: number[] = [1, 2, 3];  
let list: Array<number> = [1, 2, 3];  
  
let x: [string, number];  
x = ["hello", 10]; // OK  
  
enum Color {Red, Green, Blue};  
let c: Color = Color.Green;
```

# TypeScript

## Tipos básicos

```
let notSure: any = 4;
notSure = "maybe a string instead";

let unusable: void = undefined;

let u: undefined = undefined;
let n: null = null;

function error(message: string): never { throw new Error(message); }

declare function create(o: object | null): void;

let someValue: any = "this is a string";
let strLength: number = (<string>someValue).length;

let strLength: number = (someValue as string).length;
```

# TypeScript

## Declaraciones tipadas

- TypeScript permite la declaración de tipos en cualquier definición y reconoce el tipo correspondiente desde el IDE
- Decimos aquí que estamos usando una anotación de tipos:

The screenshot shows a code editor with three panels illustrating TypeScript's type inference and annotation features.

**Top Panel:** A code snippet with a green comment block:

```
// Type annotations
function process(x: string) {
    //x.name = "nombre";
    var v = x + x;
    alert(v);
}
```

**Middle Panel:** The same code in the editor. A red arrow points from the first panel to this one. The variable `x` is highlighted with a yellow selection bar. A tooltip below the cursor shows `(local var) v: string`.

```
// Type annotations
function process(x: string) {
    //x.name = "nombre";
    var v = x + x;
    alert(v);
}
```

**Bottom Panel:** The same code in the editor. A red arrow points from the middle panel to this one. A tooltip indicates an error: `La propiedad 'name' no existe en el tipo 'string'.` The variable `x` is highlighted with a yellow selection bar.

```
// Type annotations
function process(x: string) {
    x.name = "nombre";
    var v = x + x;
    alert(v);
}
```

# TypeScript

## Declaraciones tipadas

- Si cambiamos el tipo de declaración a **number** :

```
// Type annotations
function process(x: number) {
    //x.name = "nombre";
    var v = x + x;
    alert(v);
}
```



**var v = x + x;**  
    ⌚ (local var) v: number

---

**alert(v);**  
    ⌚ (function) alert(message?: any): void

# TypeScript

## Declaraciones tipadas

- Finalmente, si cambiamos el tipo de declaración a **boolean**:

```
// Type annotations
// También podemos declarar el valor
var x: string = "Saludos";

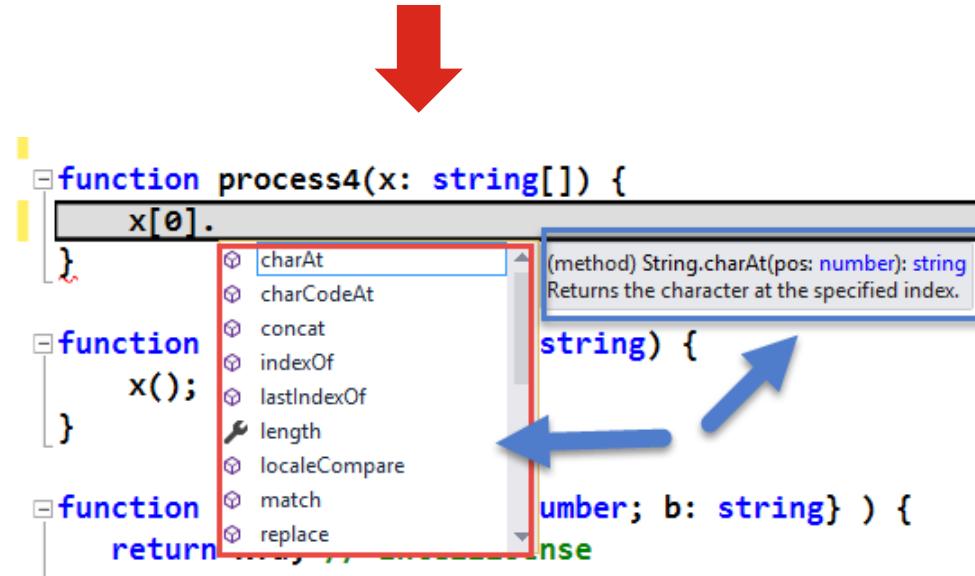
// Y se detectan operaciones incompatibles
function process(x: boolean) {
    var v = x + x;
}
```

# TypeScript

## Tipos complejos

- Los tipos complejos básicos (como **arrays**) se soportan de la misma forma: basta con indicar doble corchete al final.

```
function process4(x: string[]) {  
    // x[0]. -> Tiene "Intellisense"  
}
```



# Ejemplos de anotación de tipos

```
// array anotado
var nombres: string[] = ['Obdulio', 'Eufronio', 'Palmacio', 'Monglorio'];
// función con parámetro y retorno anotados
var decirHola: (nombre: string) => string;
// implementacion de la función decirHola
// (para el compilador la anotación anterior funciona como si fuera una interfaz)
decirHola = function (nombre: string) {
    return 'Hola, ' + nombre;
};
// objeto anotado
var persona: { nombre: string; alturaCM: number; };
// Implementacion del objeto persona
persona = {
    nombre: 'Monglorio',
    alturaCM: 183
};
```

# TypeScript

## Tipos complejos

- Los interfaces se prestan especialmente bien para este tipo de declaraciones:

```
interface Monumento {  
    nombre: string;  
    alturaEnMetros: number;  
}  
  
// Aquí se usa la interfaz Monumento  
var monumentos: Monumento[] = [];  
  
// Cada elemento añadido se comprueba en compatibilidad  
monumentos.push({  
    nombre: 'Estatua de la Libertad',  
    alturaEnMetros: 46,  
    ubicacion: 'EE.UU.'  
});  
...  
// Mas adelante podemos crear métodos de comparación
```

# Ejemplo: Comparador

```
monumentos.push({  
    nombre: 'Pedro el Grande',  
    alturaEnMetros: 96  
});  
monumentos.push({  
    nombre: 'Angel del Norte',  
    alturaEnMetros: 20  
});  
function compararAlturas(a: Monumento, b: Monumento) {  
    if (a.alturaEnMetros > b.alturaEnMetros) {  
        return -1;  
    }  
    if (a.alturaEnMetros < b.alturaEnMetros) {  
        return 1;  
    }  
    return 0;  
}  
// El método array.sort espera un comparador que acepte 2 argumentos  
var monumentosPorAltura = monumentos.sort(compararAlturas);  
// Lee el primer elemento del array que es el mas alto.  
var monumentoMasAlto = monumentosPorAltura[0];  
console.log(monumentoMasAlto.nombre); // Pedro el Grande
```

# TypeScript

## Expresiones Lambda

- Las declaraciones de funciones pasadas como argumento, se hacen mediante **expresiones lambda**.
- Se puede considerar que las expresiones lambda definen contratos.
- En el ejemplo, el contrato indica que proceso recibe un argumento de tipo **function**, que no recibe parámetros, y devuelve una cadena.

```
function proceso(x: () => string) {  
    x();  
}
```

# TypeScript

## Objetos

- Definición de una función que recibe un objeto cualquiera, y dispondremos de Intellisense.
- `proceso()` recibe como argumento un object cuya estructura doble consta de un número y una cadena.

```
function proceso( x: {a: number; b: string} ) {  
    return x.a; // Intellisense  
}
```

# TypeScript

## Interfaces

- Al igual que sucede en lenguajes OOP (Java, C#), podemos declarar interfaces: tipos personalizados cuya estructura define esos "contratos", que el contexto comprueba en tiempo de compilación.
- Esas interfaces permiten una declaración externa de contratos, reutilizable a través de la aplicación.

```
interface Cosa {  
    a: number;  
    b: string;  
    c?: boolean; //opcional  
}  
  
// el transpiler comprueba que x sea de tipo Cosa  
function process7(x: Cosa) {  
    x.b; // Intellisense  
}  
  
var llamar = process7({ a: 10, b: "Saludo" });
```

# TypeScript

## Interfaces

- Así pues, mediante las interfaces se establecen las expectativas sobre el comportamiento de los objetos. Esto incluye miembros obligatorios u opcionales, funciones (métodos), eventos, etc.
- Por ejemplo, podemos declarar la interfaz Amigo de la siguiente forma:

```
interface Amigo {  
    nombre: string;  
    colorFavorito?: string;  
}
```

- Si más adelante creamos una función que utilice esa interfaz en la declaración del argumento, la sintaxis será:

```
function nuevoAmigo(amigo: Amigo) {  
    var nombre = amigo.nombre;  
}
```

# TypeScript

## Interfaces

- En el momento de utilizarla, el entorno reconocerá las llamadas incorrectas, indicándolo al subrayar la segunda de estas llamadas, que no incluye el parámetro obligatorio nombre:



```
nuevoAmigo({ nombre: "Paco" }); // Ok
// Error, se requiere el nombre
nuevoAmigo({ colorFavorito: "Amarillo" });
nuevoAmigo({ nombre: "Cristina", colorFavorito: "Rojo" }); // Ok
```

- Como veremos al utilizar las clases, esto permite crear constructores personalizados que disponen de "Intellisense" desde el primer momento.

# TypeScript

## Interfaces

- Naturalmente, la interfaz, puede declarar métodos, y utilizar las técnicas habituales de sobrecarga:

```
interface Test {  
    a: number;  
    b: string;  
    hacerAlgo(s: string, n?: number): string; //funciones(métodos)  
    // Sobrecargas de métodos  
    hacerAlgo(n: number): number;  
}
```

# Clases en TypeScript

- TypeScript nos permite trabajar con una de las novedades más importantes de ES6: las clases.
- Hasta la versión 5, JavaScript ha funcionado mediante herencia de prototipos.
- A partir TypeScript (y ES6, cuando esté soportado en los navegadores), podemos definir clases al estilo de lenguajes como Java/C#:

```
class Greeter {  
    greeting: string;  
    constructor(message: string) {  
        this.greeting = message;  
    }  
    greet() {  
        return "Hello, " + this.greeting;  
    }  
}  
  
var greeter = new Greeter("world");
```

# Clases en TypeScript: Implementación

- La implementación de clases permite incluir miembros de datos (estado), así como una parte funcional (métodos).
- Toda clase contiene un método especial llamado igual que la clase: **constructor**.
- El constructor será el método invocado con la sintaxis de llamada **new Clase()...**
- La declaración de un método constructor debe seguir todas las restricciones habituales de la POO al objeto de potenciar la coherencia de la clase.
- La palabra reservada **this** siempre se refiere a la clase activa.

# Clases en TypeScript: Implementación

- Los métodos se definen como los de cualquier otra función:
  - Pueden recibir parámetros de cualquier clase
  - Se pueden definir parámetros que no son pasados si utilizamos las declaraciones opcionales.
- Adicionalmente, muchos teóricos afirman que los principios **SOLID** deberían aplicarse igualmente en la implementación de una clase, y en especial, el primero (Separation of Concerns).
- Una clase podrá heredar de otra, sobrescribir métodos, implementar interfaces predefinidas, y servir de base a otras clases, al objeto de implementar una jerarquía.

# TypeScript

## Métodos de acceso

- Los **getters y setters** son una manera de interceptar cómo se accede a un miembro de un objeto.
- Esto permite un control más preciso sobre cómo se accede a un miembro de un objeto.
- Vamos a convertir una clase simple para usar get y set. Empezamos con un ejemplo base:

```
class EmpleadoBase {  
    nombreCompleto: string;  
}  
  
var empleado = new EmpleadoBase();  
empleado.nombreCompleto = "Joaquín Tillizo";  
if (empleado.nombreCompleto) {  
    alert(empleado.nombreCompleto);  
}
```

# TypeScript

## Métodos acceso

- Este acceso podría permitir modificaciones no deseadas. Para evitarlo creamos una versión de la clase con **get/set** que requiere una contraseña.

```
class EmpleadoGS {  
    private _nombreCompleto: string;  
    get nombreCompleto(): string {  
        return this._nombreCompleto;  
    }  
    set nombreCompleto(nuevoNombre: string) {  
        if (contraseña == "contraseña") {  
            this._nombreCompleto = nuevoNombre;  
        }  
        else {  
            alert("Error: ¡Actualización no autorizada!");  
        }  
    }  
}
```

# Clases TypeScript: public, private, protected y static

- **public** es el por defecto

```
class Animal {  
    public name: string;  
}
```

- Los elementos **private** no pueden ser accedidos desde fuera de la clase.
  - Por convención los elementos private se suelen marcar con "\_" al inicio del nombre  
**private \_nombreCompleto: string;**

```
class Person {  
    protected name: string;  
}
```

- los elementos **protected** solo pueden accederse por derivados de una clase

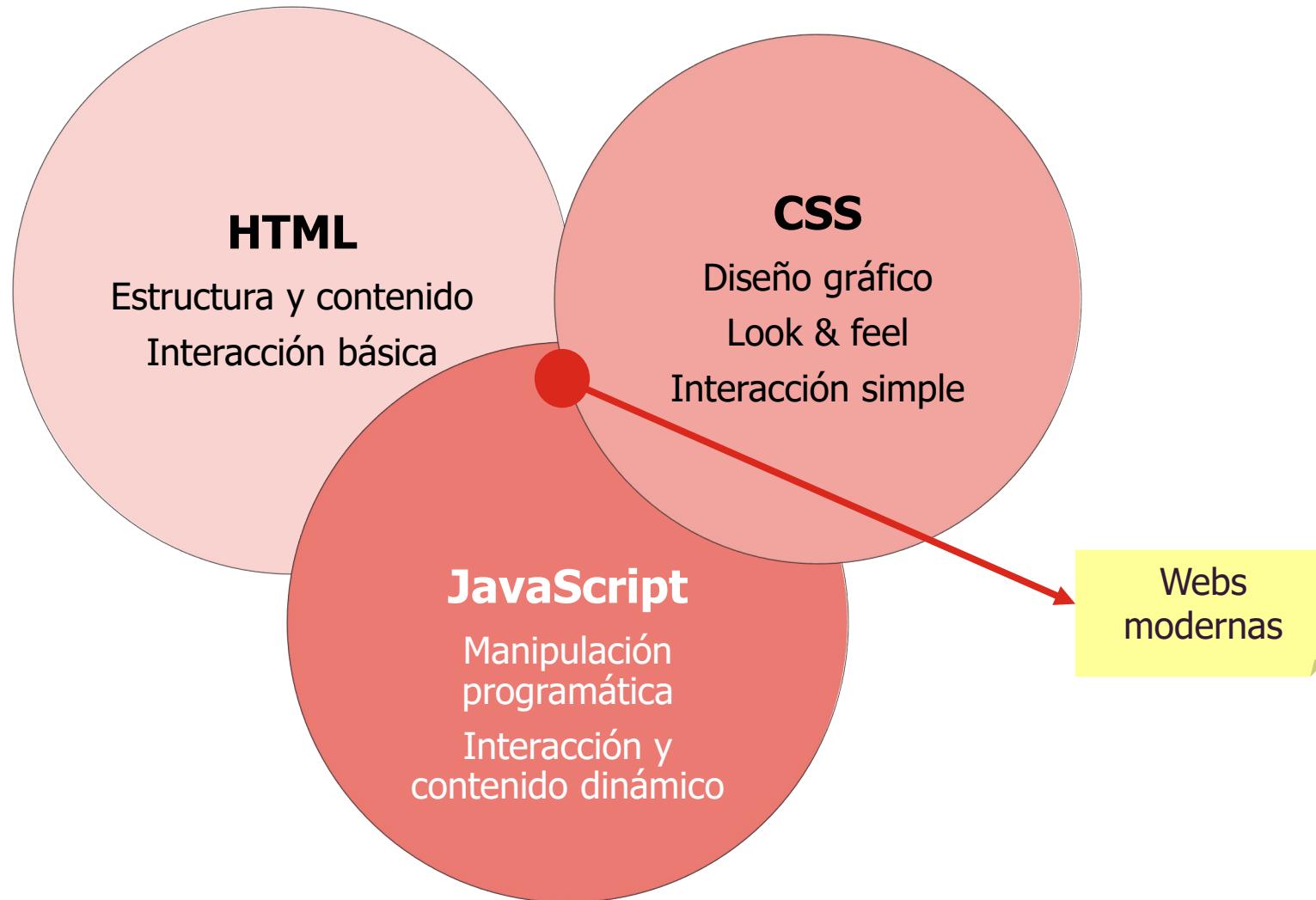
```
class Grid {  
    static origin = {x: 0, y: 0};  
}
```

# Desarrollo y reutilización de componentes de software multimedia mediante lenguajes de guión

## 4. Lenguajes de guión de uso general

- Integración de lenguajes de scripts en navegadores web
- Estructura general de un programa en un lenguaje de scripts
- Funciones
- Manipulación de texto
- Lista o arrays
- Formatos estándar de almacenamiento de datos en lenguajes de scripts
- Objetos
- El modelo de documento web. DOM
- Gestión de eventos
- Gestión de errores
- Usos específicos de lenguajes de scripts
- Entornos integrados o Frameworks para el desarrollo con lenguajes de scripts
- Extensiones útiles de navegadores

# Introducción



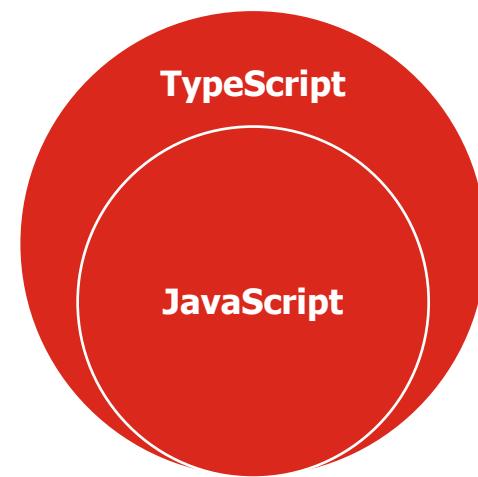
# JavaScript

## De contenido estático a contenido dinámico e interactivo

- A excepción de los elementos hipervínculos HTML (tag `<a>`), el contenido es estático, es decir, no es interactivo, pues no cambia frente a acciones del usuario.
- Utilizando reglas de estilo CSS, se enriquece el aspecto, y se crear hacer efectos interactivos limitados, como por ejemplo cambiar el aspecto de un hyperlink al pasar sobre este.
- JavaScript permite crear **contenido interactivo y dinámico**:
  - Modificación del aspecto visual del documento web
  - Validación avanzada de formularios
  - Manejo de eventos
  - Interacción con el servidor para descarga de datos (AJAX)
  - Soportado en todos los navegadores

# Características

- Originalmente llamado LiveScript
- Es **interpretado** y **orientado a objetos**
- Basado en la especificación del lenguaje, **ECMAScript (ES)**
- Su uso se ha incrementado significativamente por el enfoque **interactivo** y **dinámico** de los sitios web actuales.
- Desde 2009 podemos crear aplicaciones **fullstack** con un solo lenguaje de programación tanto en el **cliente** como en el **servidor**.



# JavaScript

## Historial de versiones

Versión	Fecha	Características principales
ES1	1997	Primera edición
ES2	1998	Alineación con ISO/IEC 16262
ES3	1999	Expresiones regulares, manejo de cadenas, try / catch, formatos
ES4	Nunca lanzada	
ES5	2009	strict mode, getters / setters, JSON, reflection
ES6	2015	Let y const, arrays tipados, promises
ECMAScript 2016	2016	ES2016, aislado del código, ** y Array.prototype.includes
ECMAScript 2017	2017	Concurrencia, await / async, sobrecarga de operadores, registros
ECMAScript 2018	2018	Propiedades de descanso/propagación, iteración asíncrona

# JavaScript

## Estructura general

- El tag **<script>** permite añadir un bloque de instrucciones JavaScript
- El bloque JS normalmente en el **<head>** del documento.

```
<script type="text/javascript">  
    //código JavaScript  
</script>
```

type: Define el tipo MIME del script incluido. Debe ser "text/javascript". Actualmente se puede omitir.

- Algunos navegadores **no soportan scripts**, por lo que ignorarán la marca **<script>**, pero no el contenido. La solución es escribir el bloque JS entre los símbolos de comentarios:

```
<script type="text/javascript">  
    <!-- ocultar script  
    //código JavaScript -->  
</script>
```

# JavaScript

## Código JS externo

- Código JavaScript contenido en archivo de extensión **.js** referenciado desde el documento:

```
<script type="text/javascript" src="myscript.js"></script>
```

src: URL que referencia al archivo que contiene el código JavaScript. Permite rutas relativas y absolutas.



- El contenido de `myscript.js` es directamente el código JavaScript, sin etiquetas.
- Esta opción permite **reutilizar el código** en otras páginas, por lo que es la **recomendable**.

# JavaScript

- “Hola Mundo” en JavaScript

```
<script>
    window.alert("Hello world!");
    document.write("<p>Hola Mundo!</p>");
</script>
```

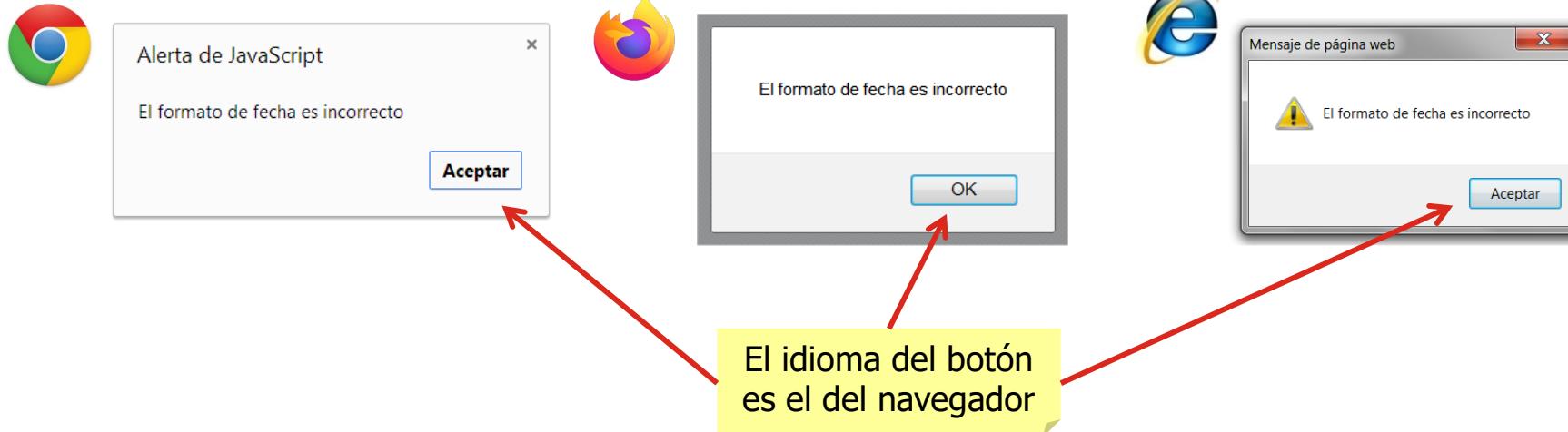


# Sintaxis básica JavaScript

## Mensajes interactivos

- JavaScript provee comandos para mostrar mensajes interactivos al usuario:
- **alert**: mensajes informativos o de advertencia.

```
alert("El formato de fecha es incorrecto");
```



# Sintaxis básica JavaScript

## Mensajes interactivos

- **confirm**: mensajes de decisión.

```
if (confirm("Confirme si desea continuar.")) {  
    //lógica si marca Aceptar/OK  
}
```

El retorno indica el botón presionado:

- Aceptar/OK: true
- Cancelar: false



JavaScript

Confirme si desea continuar.

Aceptar Cancelar



Confirme si desea continuar.

OK Cancel



Mensaje de página web

Confirme si desea continuar.

Aceptar Cancelar

- **prompt**: permite recoger valores alfanuméricos.



# ACTIVIDAD

**Editor de código online**

## OBJETIVO

Manejar herramientas online para edición y prueba de código JavaScript, HTML y CSS.

## INSTRUCCIONES

1. Probar las funciones alert, prompt y confirm en el editor de código online codepen.

<https://codepen.io/>



15 min

# Sintaxis básica JavaScript

## Comentarios

- Los comentarios pueden ser por línea o por bloque:

```
//Comentarios  
//de  
//línea
```

Ejemplo:

```
var a = 56; //asign. básica
```

```
/*  
Comentario de bloque  
*/
```

Ejemplo:

```
/*  
Con el siguiente comando se obtienen  
las propiedades del navegador  
*/  
var nav = window.navigator;
```

# Sintaxis básica JavaScript

- Un **identificador** es un nombre para referenciar a un elemento del código (clase, variable, método...)
- Es una secuencia alfanumérica cuyo primer carácter no es un número.
- Se pueden incluir vocales acentuadas, la ñ y letras con símbolos (ç, ...), aunque no es recomendable por problemas de portabilidad entre plataformas, ya que los juegos de caracteres pueden afectar el código.
- Se permiten incluir otros caracteres imprimibles (\$, \_) exceptuando aquellos que tienen un significado especial dentro del lenguaje (por ejemplo % y ?). De todas formas, no se recomienda.
- Ejemplos válidos pero no todos cumplen las convenciones:

Arriba, caña, C3P0, àëÎò, hola, \_barco\_

- Ejemplos no válidos:

3com, 123, 4D5v, C#, a!b, r..t, qwe`t



comienzan con número



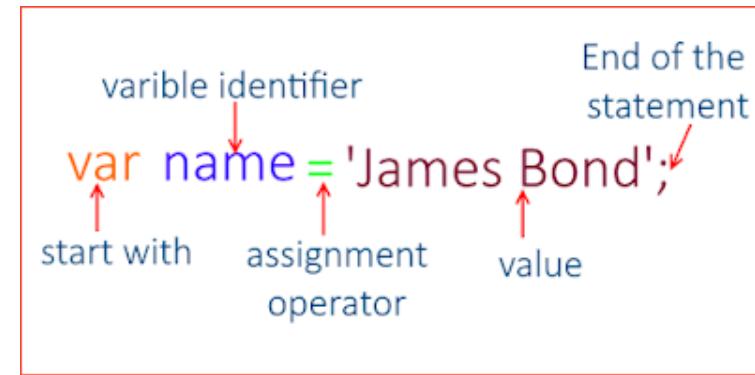
caracteres no válidos

# Sintaxis básica JavaScript

## Variables

- Una **variable** es una referencia a un valor u objeto de un determinado tipo. Para referirse al valor utilizamos su identificador.
- Declaración o definición de una variable con **var**:

```
var <identificador>;  
  
var x; var letra; var mensaje; var obj;
```



- Asimismo se puede inicializar una variable asignándole un valor u objeto en la declaración, utilizando el símbolo "=":

```
var x = 45;  
var mensaje = "Hola";  
var enabled = true;  
var today = new Date();
```

# Sintaxis básica JavaScript

## const

- A partir de la versión 2015 de JavaScript (ES6 - ECMAScript 2015) podemos usar **const** para declarar **una constante** (no se puede reasignar).

```
const A_STRING = 'I am a string';
const A_NUMBER = 100;
const AN_ARRAY = [ 'var1', 12, { foo: 'bar' } ];
const AN_OBJECT = { name: 'MyAPP', ver: '1.0' };
/*
Convención: escribir las constantes en mayúsculas para
distinguir las del resto de variables
*/
```

# Sintaxis básica JavaScript

## let

- A partir de la versión 2015 de JavaScript (ES6 - ECMAScript 2015) podemos usar **let** para definir una variable con **alcance restringido**.
- Su ámbito o scope es el bloque en donde está definida:

```
function pets() {  
    var dog = "Fido";  
  
    if (true) {  
        let cat = "Fluffy";  
    }  
}
```

Alcance de **var** es en cualquier parte dentro la función

Alcance de **let** es dentro del bloque {}

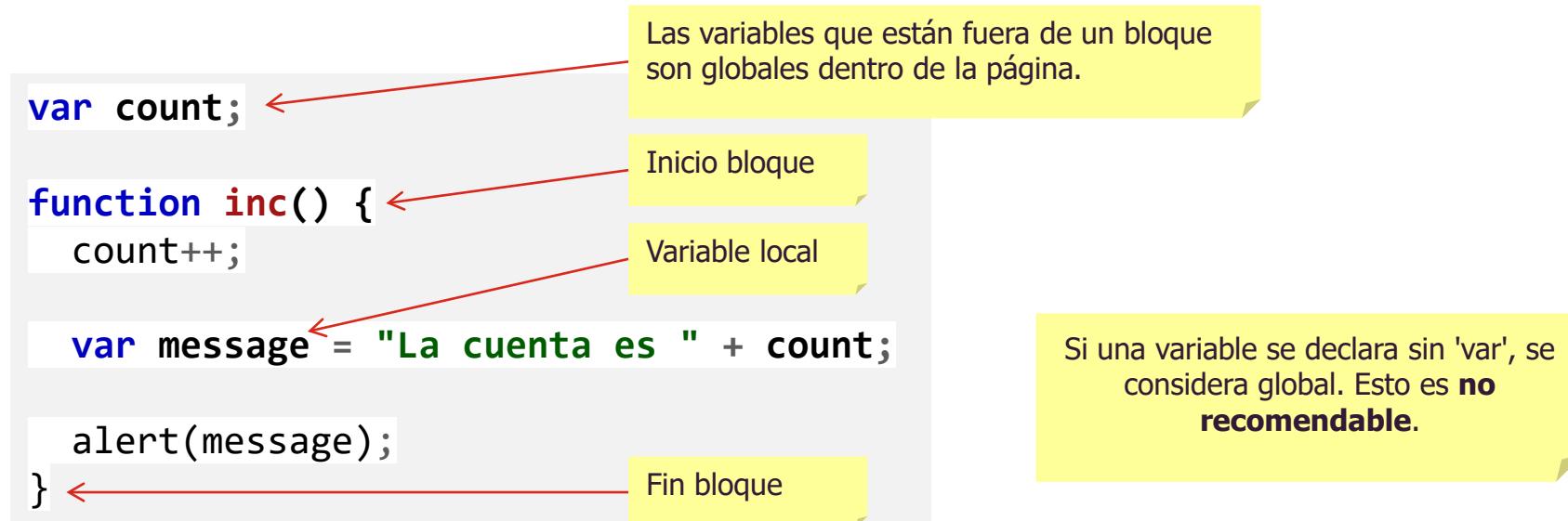
# const vs let vs var

keyword	const	let	var
<b>Alcance Global</b>	NO	NO	SÍ
<b>Alcance Función</b>	SÍ	SÍ	SÍ
<b>Alcance Bloque</b>	SÍ	SÍ	NO
<b>Re-asignable</b>	NO	SÍ	SÍ

# Sintaxis básica JavaScript

## Ciclo de vida de una variable

- Una **variable** se crea en el ámbito de un bloque que la contiene:



- Todas las variables creadas son **destruidas al salir de la página**.
- Si se vuelve a entrar a la página, las variables se crean nuevamente, no estando relacionadas con valores anteriores de las mismas.

# Sintaxis básica JavaScript

## Tipos de variables

- Una **variable** puede ser de varios tipos, aunque siempre se identifica como var.
- Los tipos utilizados son:
  - **texto (string)**: se asignan con una cadena, con comillas dobles o simples:

```
var mensaje = "El campo 'nombre' es requerido";
```

```
var mensaje = 'Comillas "dobles"';
```

Las comillas simples y dobles  
son intercambiables, siempre  
que se mantenga el orden de  
cierre.

- **número (number)**: se asignan con un número de cualquier tipo. No se hace distinción entre enteros o números con decimales.

```
var max = 10;
```

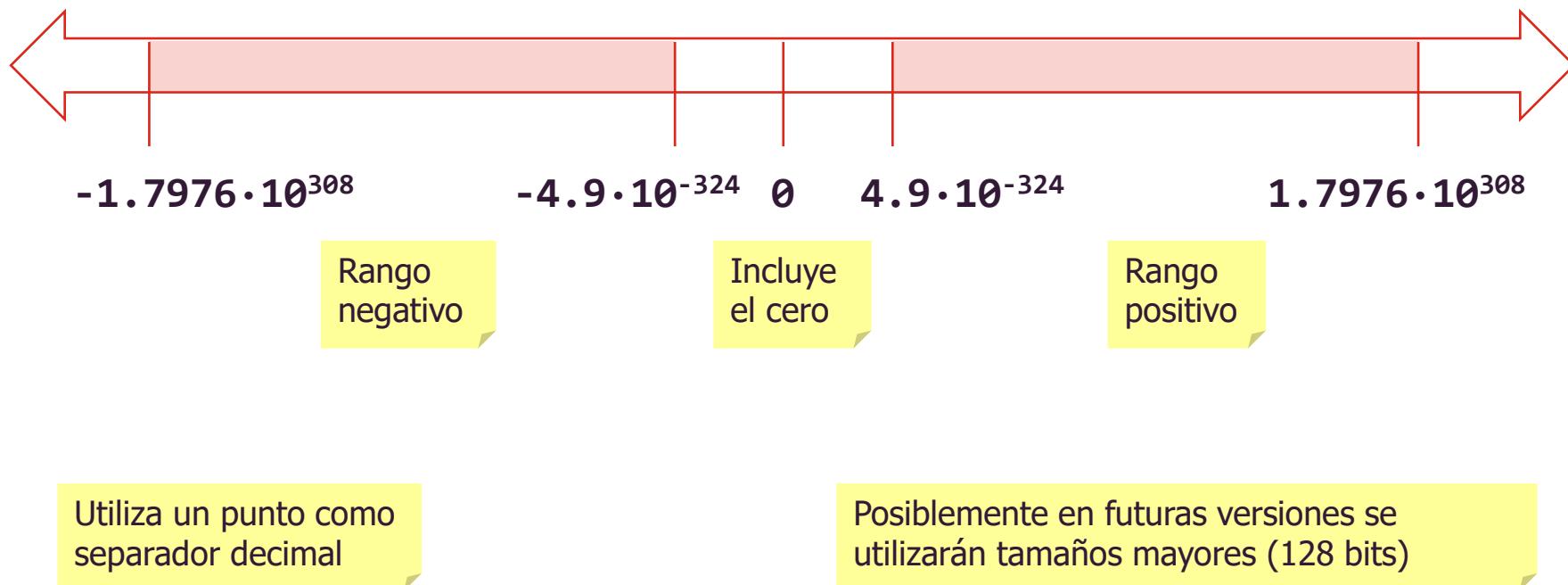
```
var distancia = 12.34;
```

Separador decimal: punto.

# Sintaxis básica JavaScript

## Número

- JavaScript maneja variables de tipo **numérico (number)**. Internamente se representan como un número de punto flotante de **64 bits**, lo que permite **doble precisión** y el rango siguiente:



# Sintaxis básica JavaScript

## String

- JavaScript maneja variables de tipo **texto (string)**. Se declaran como textos entre comillas simples o dobles. Internamente se almacenan como una cadena de caracteres, sin un largo máximo establecido. Ejemplo:

```
var mensaje = "Hola";
```

- Las cadenas son objetos que tienen asociadas algunas funciones y propiedades predefinidas útiles para su manejo.
- Para conocer la longitud de un string, es decir, el número de caracteres, se utiliza la propiedad **length**:

```
str = "12345";
```

```
str.length
```



5

# Sintaxis básica JavaScript

## Boolean

- **Lógico (boolean)**: se asignan con los valores true o false.

```
var enabled = true;  
var isValid = false;
```

# Sintaxis básica JavaScript

- JavaScript maneja variables de tipo **lógico (boolean)**. Internamente se representan como un valor equivalente a un bit.
- Ejemplos:

```
var enabled = true;  
  
var isFinished = count > 10;  
  
var hasValue = message != "";
```

Asignación

Comparación

Un boolean puede  
contener el resultado de  
un operador relacional

# Sintaxis básica JavaScript

## Conversión de Tipos

- Si se tiene un valor tipo texto, por ejemplo "1", y se quiere manejar como número, JavaScript provee funciones para **convertir** entre tipos de variables. Esto es útil porque al ser todas 'var', no es posible diferenciarlas por su tipo.
- Ejemplos:

```
var strOne = "1";
var numOne = Number(strOne);

var strEnabled = "true";
var enabled = Boolean(strEnabled);
```

- También es posible obtener el tipo interno de una variable, utilizando `typeof`. Ejemplo:

```
if (typeof numOne == "number") {
  alert("numOne es un number");
}
```

# Sintaxis básica JavaScript

## Operadores aritméticos y relacionales

- Un **operador** es un símbolo formado por uno o más caracteres, que permite realizar una determinada operación entre uno o más datos y produce un resultado.

### Operadores aritméticos:

- Unarios: negativo (-) y positivo (+)
- Binarios: suma (+), resta (-), producto (\*), división (/) y módulo (%)

### Operadores relacionales:

- igualdad (==). No confundir con =, que es para asignación.
- desigualdad (!=)
- mayor que (>)
- menor que (<)
- mayor o igual que (>=)
- menor o igual que (<=)

# Sintaxis básica JavaScript

## Operadores y asignación

- El operador de **asignación** básico es (**=**) que, a parte de asignar, devuelve el valor asignado. La asignación es asociativa por la derecha.
- Operadores **compuestos**: realizan una operación y asignan el resultado. Existen para la suma (**+=**), resta (**-=**), producto (**\*=**), división (**/=**), módulo (**%=**).
- Operadores **incrementales**: realizan un incremento (**++**) o decremento (**--**) de una unidad en una variable. Si el operador se coloca antes del nombre de la variable devuelve el valor de la variable antes de incrementarla; si se hace después, se incrementa y devuelve el valor ya incrementado.

# Sintaxis básica JavaScript

## Operadores y asignación

- Ejemplos de asignación (todos son `var` tipo `number`):

```
a = 56;          //asignación básica  
  
b = 78 + a;    //asignación básica  
  
c += 23;        //asignación compuesta. Equivale a c = c + 23  
  
i++;           //Incremento. Equivale a i = i + 1  
  
d = i++;       //Asigna e incrementa. Si i vale 3 al inicio, d vale 3.  
  
d = ++i;       //Incrementa y asigna. Si i vale 3 al inicio, d vale 4.
```

# Sintaxis básica JavaScript

## Concatenación

- Los textos se pueden **concatenar** utilizando el operador **“+”**:
- Como el operador **“+”** también se utiliza para sumar números, hay ciertos matices a tener en cuenta, los que se ilustran a continuación:

```
var str = "Esto es " + "un string";
```

```
s1 = 1 + 3 + "5" + "7";
```

457

Regla de izquierda a derecha.  
Suma 1 + 3, y luego concatena "5" y "7"

```
s2 = 1 + (3 + "5") + 7;
```

1357

Calcula el paréntesis primero.  
Concatena 1, el resultado, y 7

```
s3 = "1" + (3 + 5) + 7;
```

187

Como el primer operando es un string,  
comienza concatenando

```
s4 = 1 + "3" + 5 + 7;
```

1357

Como la primera operación incluye a un  
string, comienza concatenando

```
s5 = "" + 1 + 3 + 5 + 7;
```

1357

Si se quieren concatenar sólo números, se  
puede agregar un string vacío

```
s6 = "" + (1 + 3 + 5 + 7);
```

16

Si se quiere convertir una suma a string,  
similar a la anterior y también String(...)

# Sintaxis básica JavaScript

## Operadores lógicos

- Unarios: not (**!**)
- Binarios: and (**&&**) y or (**||**). Son perezosos (no evalúan el segundo operando cuando se deduce el resultado del primero). No perezosos: **&** y **|**, que son poco utilizados.

```
var a = false;  
var b = true;  
var c = true;
```

```
var j = a && (b || c);
```

Evalúa sólo a, y resulta false

```
var k = b || (b && c);
```

Evalúa sólo b, y resulta true

```
var m = !b & (b || c);
```

Evalúa !b y también el paréntesis, del cual sólo evalúa b. Resultado se convierte con Boolean(m)

```
var n = !a | (b && c);
```

Evalúa !a y también el paréntesis, del cual evalúa b y c. Resultado se convierte con Boolean(n)

# Sintaxis básica JavaScript

## Operador condicional

- Es el único de JavaScript con tres operandos, cuya sintaxis es:

<condición> ? <expresión1> : <expresión2>

- Se evalúa <condición>
- Si es verdadera se devuelve el resultado de evaluar <expresión1>
- Si es falsa, el resultado de evaluar <expresión2>
- Ambas expresiones deben ser del tipo al que se asigna el resultado.
- Ejemplo:

```
max = (a > b) ? a : b;
```

Si a es mayor que b, devuelve a. En caso contrario, devuelve b



# Sintaxis básica JavaScript

## Precedencia de operadores

Operadores	Prioridad
!, +, - (unarios)	mayor
*, /, %	
+, -	
<, <=, >=, >	
==, !=	
&&	
= (asignación)	menor

# Sintaxis básica JavaScript

## Caracteres especiales

- En JavaScript existe una representación específica para algunos caracteres especiales, que permite definirlos y utilizarlos:

Carácter	Significado	Utilización
\n	newline (Nueva línea)	Para saltos de línea en manejo de cadenas
\r	carriage return (Salto de carro)	Idem anterior, en Windows.
\t	tab (tabulación)	Detectar o insertar tabulaciones
\"	comillas (también existe \'')	Colocar comillas en un String. Ej: str = "Esto va \"entre comillas\".";
\\	escapa un \	Para escribir un "\" en una cadena. Ej: str = "Para newline se utiliza \\n";
\b	backspace	Poco usado
\f	form feed	Poco usado



# ACTIVIDAD

Debugging tools

## OBJETIVO

Inspeccionar y depurar código JavaScript a través de las herramientas integradas en el navegador: Chrome Devtools.

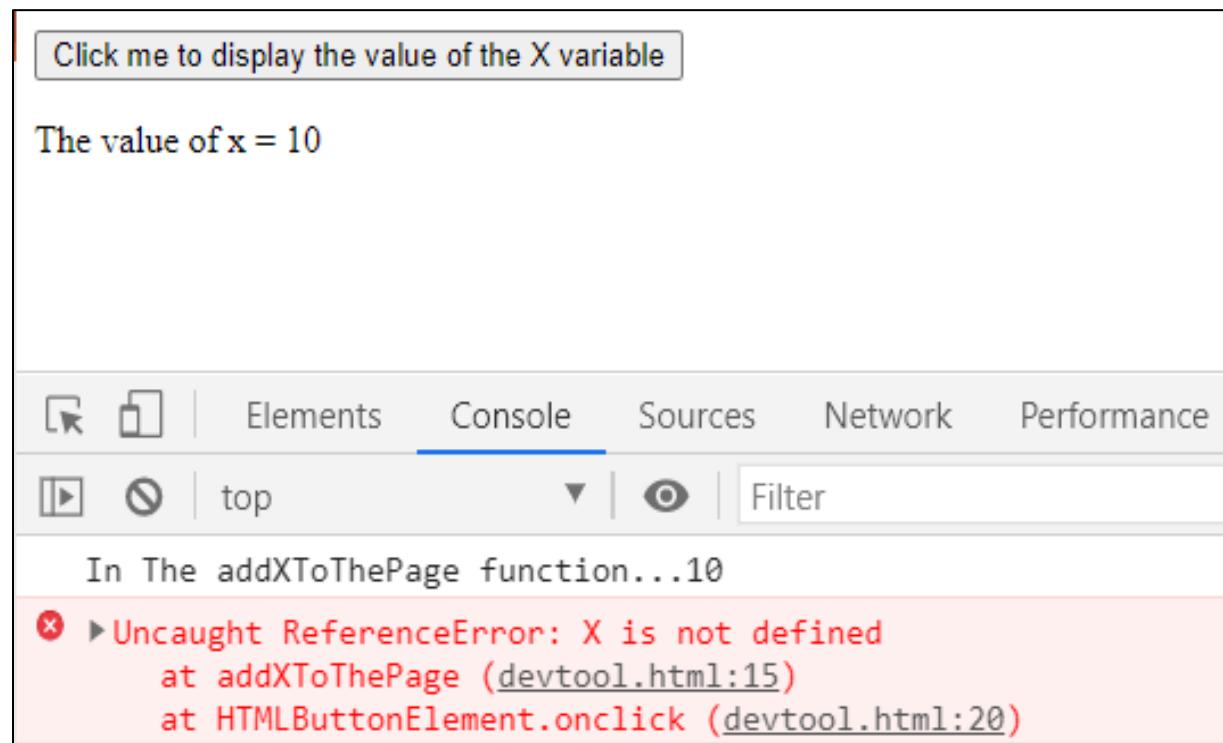
## INSTRUCCIONES

1. Abrir a la consola de depuración del navegador con F12. Seleccionar la pestaña Console y escribir la siguiente instrucción JS en la consola:  
`console.log("Probando la consola...").`
2. Copia el código que se adjunta tal cual está y guárdalo con formato HTML.



25 min

3. Abre el fichero anterior en el navegador y en la consola Devtool observa el error que se muestra (en rojo). Intenta encontrar el problema y modifica el código para solucionarlo.



25 min

```
<!DOCTYPE html>
<html>
<head>
    <title>Write messages to the devtool console</title>
    <script>
        var x = 10;
        function addXToThePage() {
            console.log("In The addXToThePage function..." + x);
            //This code is strangely never executed...
            document.body.innerHTML += "<p>The value of x = " + x + "</p>";
            x = x + 1;
        }
    </script>
</head>
<body>
    <button onclick="addXToThePage();">
        Click me to display the value of the X variable
    </button>
</body>
</html>
```



15 min

# Sintaxis básica JavaScript

## Control de flujo

- Sentencia condicional **if**

```
if (<condición>) {  
    <instrucciones>  
} else {  
    <instrucciones>  
}
```

Ejemplo:

```
if (a > b) {  
    alert("A gana");  
} else {  
    alert("B gana");  
}
```

```
if (<condición1>) {  
    <instrucciones>  
} else if (<condición2>) {  
    <instrucciones>  
}
```

//varias condiciones else if...

```
} else if (<condiciónN>) {  
    <instrucciones>  
} else {  
    <instrucciones>  
}
```

# Sintaxis básica JavaScript

## Control de flujo

- Sentencia condicional **switch-case**

```
switch (<expresión>){  
    case <literal1>:  
        <instrucciones>  
        break;  
    case <literal2>:  
        <instrucciones>  
        break;  
    //N valores...  
    case <literalN>:  
        <instrucciones>  
        break;  
    default:  
        <instrucciones>  
}
```

Cuando entra a un "**case**", continúa hasta el siguiente "**break**"

Ejemplo:

```
switch (estado){  
case 0:  
    alert("Opción 0");  
    break;  
case 1:  
    alert("Opción 1");  
case 2:  
    alert("Opción 2");  
    break;  
default:  
    alert("Ninguna");  
}  
si estado es 0:  
"Opción 0"  
si estado es 1:  
"Opción 1"  
"Opción 2"
```

Permite cualquier tipo de variable. Normalmente se utiliza Number y String

Ya que no hay break en case 1

# Sintaxis básica JavaScript

## Control de flujo

- Sentencia iterativas **while** y **do-while**

```
while (<condición>) {  
    <instrucciones>  
}
```

Ejemplo:

```
var a = 3;  
  
while (a > 0) {  
    alert(a);  
    a--;  
}
```

```
do {  
    <instrucciones>  
} while (<condición>);
```

Ejemplo:

```
var a = 0;  
  
do {  
    alert(a);<-----  
    a--;  
} while (a > 0);
```

Pasa al menos  
una vez

# Sintaxis básica JavaScript

## Control de flujo

- Sentencia iterativa **for**

```
for (<declaración>; <condición>; <instrucción>) {  
    <instrucciones>  
}
```

```
for (var i = 0; i < 3; i++) {  
    alert(i);  
}
```

```
for (var <índice> in <iterable>) {  
    <instrucciones>  
}
```

```
var v = [4, 7, 9, 1];  
  
for (var a in v){  
    alert(v[a]);  
}
```

Lista de números (array)

# Sintaxis básica JavaScript

## Break y continue

- Existen dos elementos de control para los ciclos:
  - **break**: detiene el ciclo, continuando con las instrucciones posteriores.
  - **continue**: detiene la iteración, continuando con la siguiente.

```
for (var i = 0; i < 5; i++) {
    if (i == 3) {
        break;
    }
    alert(i);
}
```

0  
1  
2

```
for (var i = 0; i < 5; i++) {
    if (i == 3) {
        continue;
    }
    alert(i);
}
```

0  
1  
2  
4

# Sintaxis básica JavaScript

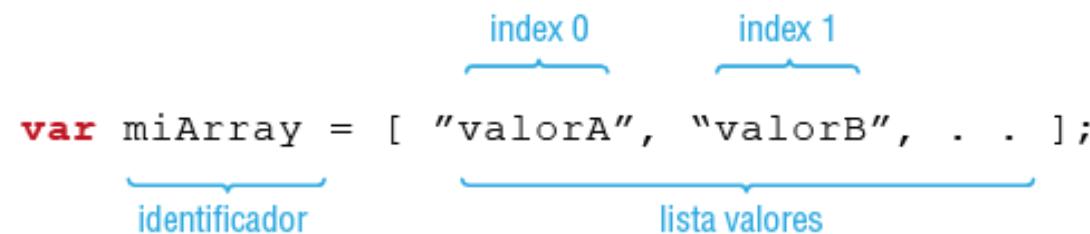
## Arrays

- Conjunto de valores accesibles mediante un índice que indica su posición dentro del array.

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

- Otra forma de declarar un array:

```
var nums = new Array();
nums[0] = 1;
nums[1] = 2;
nums[2] = 3;
```



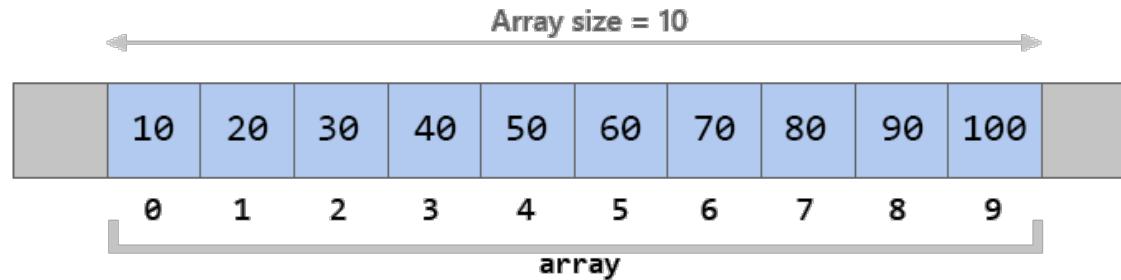
- Otros ejemplos de array:

```
var finDeSemana = ["sábado", "domingo"];
```

```
var random = ["sábado", 5, 7, "domingo",];
```

# Sintaxis básica JavaScript

## Atributo length de Arrays



```
var nums = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];
for (var i=0; i < nums.length; i++) {
    //se accede con nums[i]
}
```

# Sintaxis básica JavaScript

## Búsqueda de elementos en Arrays

- El método de los arrays `indexOf()` devuelve la posición del elemento pasado como parámetro, si no lo encuentra devuelve -1:

```
días = ["lunes", "martes", "miércoles", "jueves"]  
var posición = días.indexOf("jueves");
```

- Otra forma de buscar un elemento recorriendo el array a través del método `forEach()`:

```
<script>  
    var días = ["lunes", "martes", "miércoles", "jueves"];  
    días.forEach(function(dato, índice){  
        document.write("Hoy es " + dato + "índice: " + índice + "<br/>");}  
</script>
```

# Sintaxis básica JavaScript

## Métodos útiles de Arrays

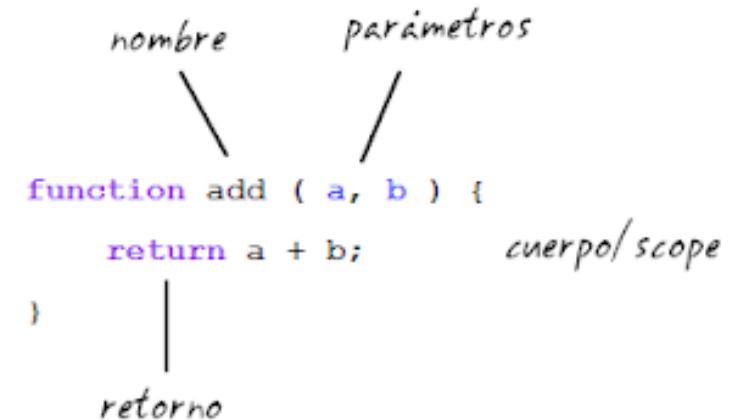
- push() añade un elemento al final del array
- pop() elimina el último elemento del array
- shift() elimina el primer elemento del array
- unshift() inserta un elemento al principio del array
- concat() fusiona dos arrays en uno
- split() convierte un string en array pasando como parámetro el carácter separador a utilizar
- join() convierte un array en un string
- sort() ordena los elementos del array en orden ascendente
- reverse() invierte el orden de un array
- splice() reemplaza elementos del array desde una posición concreta, si se omiten elementos a reemplazar se puede usar para eliminar elementos en cualquier posición del array

# Sintaxis básica JavaScript

## Funciones

- Declaración de una función en JavaScript:

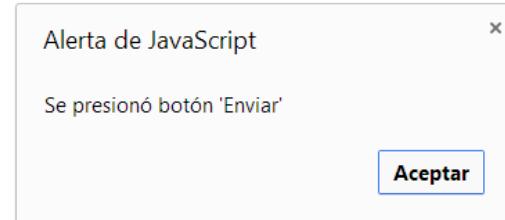
```
function mensaje() {  
    alert("Se ha presionado el botón 'Enviar'");  
}
```



- Llamada a la función (en el ejemplo asociamos la función a un evento de un botón):

```
<input type="button" value="Enviar" onclick="mensaje()" />
```

Enviar



Evento "click" del botón. Nomenclatura comienza con "on"

Función manejadora que se ejecuta cuando sucede el evento "click" del botón

# Sintaxis básica JavaScript

## Función asignada a una variable

```
var suma = function(a, b) {  
    return a * b;  
}
```

Declaración  
de la función

```
alert(suma(7, 9));
```

Llamada  
a la  
función

# Sintaxis básica JavaScript

## Funciones Flecha

- Definición de una función (ECMA5):

```
function suma(x, y) {  
    return x + y;  
}
```

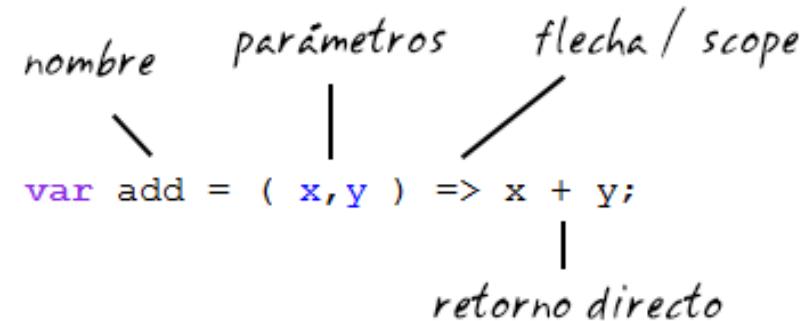
- ECMA6 permite declarar funciones de forma **abreviada** llamada **función flecha**:

```
var suma = (x, y) => { x + y };
```

- Sintaxis:

Las funciones flecha son siempre anónimas  
Sintaxis más limpia y simplificada

```
( param1, param2, ..., param n ) => { expression; }
```



# Sintaxis básica JavaScript

## Funciones Callback

- Una **callback** es una función anónima que se le pasa como argumento a otra función y se ejecuta en un determinado momento (**asíncrona**).
- Utilidad: cuando no sabemos lo que tarda en responder una función con el valor de retorno.
- Casos de uso: programación asíncrona y reactiva, funciones de escritura en ficheros y en peticiones HTTP en servicios web de tipo REST.

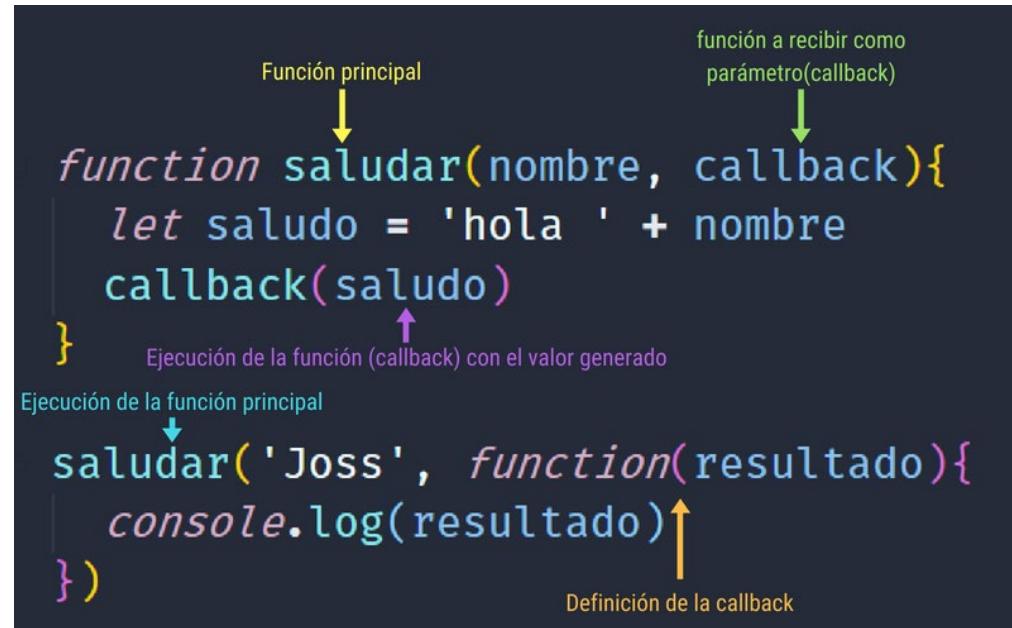


Diagrama que ilustra el flujo de ejecución de una función callback. Se muestra un código de JavaScript con anotaciones explicativas:

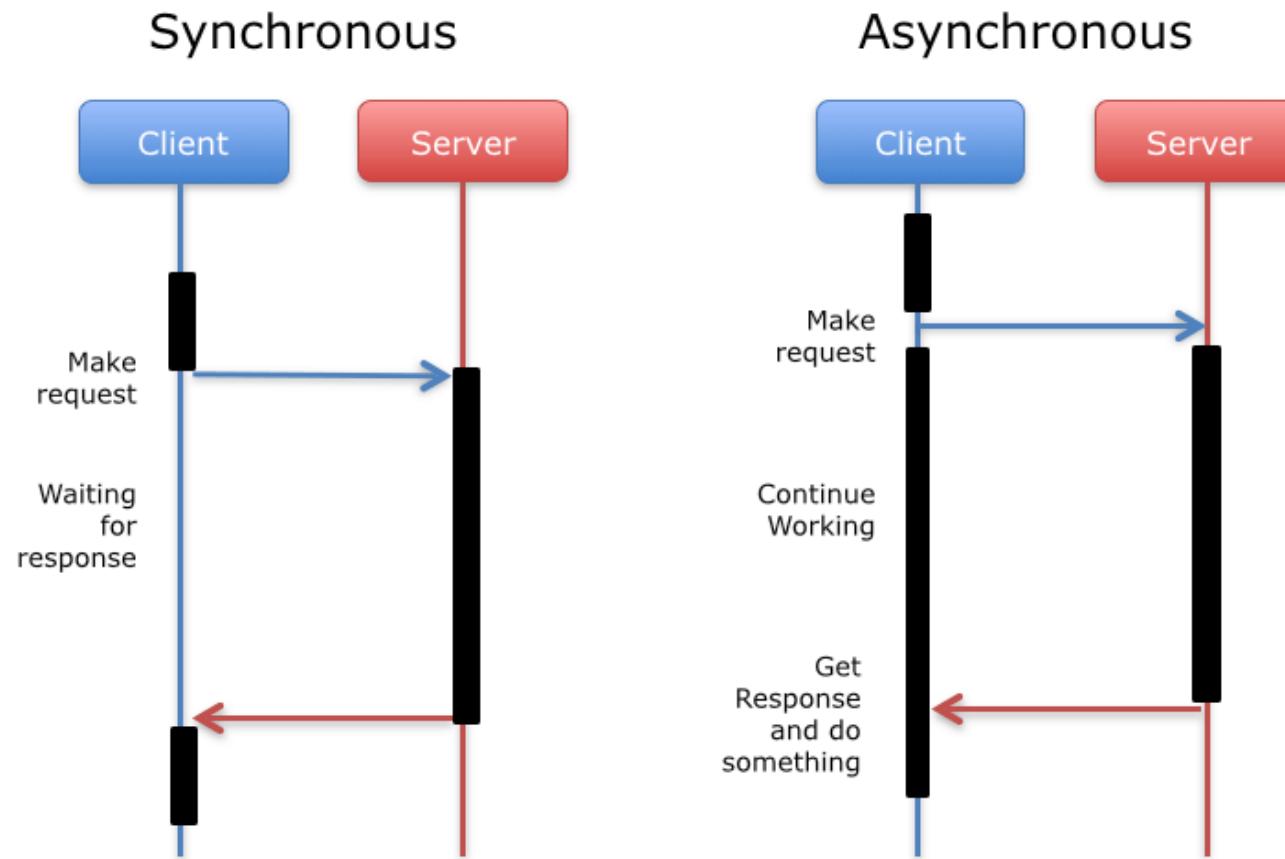
- Función principal:** `function saludar(nombre, callback){`
- función a recibir como parámetro(callback):** `callback(saludo)`
- Ejecución de la función (callback) con el valor generado:** `saludar('Joss', function(resultado){`
- Ejecución de la función principal:** `console.log(resultado)`
- Definición de la callback:** `})`

Las flechas amarillas indican el flujo de datos y el orden de ejecución entre las distintas partes del código.

```
function saludar(nombre, callback){
  let saludo = 'hola ' + nombre
  callback(saludo)
}

saludar('Joss', function(resultado){
  console.log(resultado)
})
```

# Programación Asíncrona



# Sintaxis básica JavaScript

## JavaScript ES5: Closures

- En JavaScript toda variable declarada fuera de una función, pertenece a este objeto.
- Esto puede entrar en conflicto con otros elementos declarados en librerías de soporte cargadas por la página.
- Pero una variable declarada dentro de una función es local a la función...
- La solución se presenta en forma de una construcción llamada **closure**.

```
var nombre_numero_c = function () {
    var numeros = ['cero', 'uno', 'dos', 'tres', 'cuatro'];
    return function (n) {
        return numeros[n];
    };
}();
alert(nombre_numero_c(1));
```

# Casos de uso JavaScript

## Programación Reactiva

- Programación **Asíncrona**
- Considera el procesamiento de datos de manera asíncrona
- Procesamiento de datos = **Flujo de datos asíncrono** (Stream)
- Establece los mecanismos para manipular streams (**Observable** y **Observer**)
- Implementación **ReactiveX**: API for asynchronous programming with observable streams
- Soporte para distintos lenguajes de programación
- **RxJS** Reactive API para lenguaje JavaScript

# Sintaxis básica JavaScript

## Objetos

- Definición de un objeto:

```
var person = {  
    firstName: "John",  
    lastName : "Denver",  
    id       : 12345,  
    fullName : function() {  
        return this.firstName + " " + this.lastName;  
    }  
};
```

Propiedades del objeto

Método del objeto

{ JSON }

```
{  
    "id" :1,  
    "name" :"Me",  
    "email" : me@gmail.com  
}
```

# Sintaxis básica JavaScript

## Objetos

- Formas de acceder a **propiedades** y **métodos** de un objeto:

```
// person.id  
nombreObjeto.nombrePropiedad;
```

```
// person["id"]  
nombreObjeto["nombrePropiedad"]
```

```
// person.fullName()  
nombreObjeto.nombreMetodo();
```

# Sintaxis básica JavaScript

## Objetos: Buenas prácticas

- literales de objeto {} en lugar de new Object()
- cadenas literales "" en lugar de new String()
- literales numéricos 12345 en lugar de new Number()
- literales booleanos true / false en lugar de new Boolean()
- literales de matriz [] en lugar de new Array()
- patrones literales ()/ en lugar de new RegExp()
- expresiones de función () {} en lugar de new Function()

# Sintaxis básica JavaScript

# Tipo String: funciones

- Para buscar la primera posición de un string dentro de otro, se utiliza **indexOf**, y la última, con **lastIndexOf**:

```
str = "ser o no ser, esa es la cuestión";
```

↑  
↑  
↑  
↑  
↑  
↑  
↑  
↑  
↑  
↑  
0123456789

18

26

```
str.indexOf("ser");
```

26

```
str.indexOf("ser", 3)
```

1

18

```
str.indexOf("hamlet")
```

1

-1 cuando no lo encuentra

# Sintaxis básica JavaScript

## Tipo String: funciones

- Continuando:
  - indexOf se utiliza también para saber si un string contiene una cadena:

```
str = "Cinema Paradiso";
```

```
str.indexOf("a P") != -1 → true
```

- Para extraer un carácter en una posición, se utiliza charAt:

```
str = "Cinema Paradiso";
```

```
str.charAt(2) → 'n'
```

0 a n-1

- Para extraer un trozo de una cadena, se utiliza substring:

Desde 2, largo 4

4 ←

```
str = "casablanca";
```

0123456789

Ubicar posición 2

```
str.substring(2, 6)
```

Restar:  $6 - 2 = 4$

"sabl"

# Sintaxis básica JavaScript

## Tipo String: funciones

- Continuando:
  - Para pasar el contenido todo a minúsculas, se utiliza `toLowerCase()`, y a mayúsculas es con `toUpperCase()`:

```
str = "Artificial Intelligence";  
str.toLowerCase() → "artificial intelligence"  
str.toUpperCase() → "ARTIFICIAL INTELLIGENCE"
```

- Para comparar dos textos, se utiliza `"=="`:

```
var str1 = "euro";  
var str2 = "\u0065\u0075\u0072\u006f";  
str1 == str2 → true
```

"euro" en unicode

# Sintaxis básica JavaScript

## Tipo String: funciones

- Para comparar dos textos, ignorando mayúsculas y minúsculas (case-insensitive), se deben pasar a mayúsculas o minúsculas y utilizar "==":

```
str1 = "lower or UPPER";
str2 = "Lower or Upper";
```

```
str1.toLowerCase() == str2.toLowerCase() → true
```

- Si se compara un texto y un número con el mismo valor, sucede lo siguiente:

```
str1 = "12";
str2 = 12;
```

```
str1 == str2 → true
```

Compara valores

```
str1 === str2 → false
```

Compara valores y tipos

```
str1 === String(str2) → true
```

```
Number(str1) === str2 → true
```

# Sintaxis básica JavaScript

## Tipo String: funciones

- Para reemplazar en un String las ocurrencias de una cadena de caracteres por otra, se utiliza el método replace:

```
name = "les aventures de tintin et milou";
```

Por default, sólo reemplaza la primera ocurrencia

```
name = name.replace("tin","tan"); → "les aventures de tantin et milou"
```

- El método replace admite expresiones regulares (se ven más adelante), lo que flexibiliza su uso:

```
name = "Déjà vu";
```

Encuentra las minúsculas de la "a" a la "z"

```
name.replace(/[a-z]/, "_");
```

Reemplaza la primera por un "\_"

```
"Dé_à vu"
```

```
name.replace(/[a-z]/g, "_");
```

Reemplaza todas las minúsculas por un "\_"

```
"Dé_à __"
```

```
name.replace(/[a-z]/gi, "_");
```

Reemplaza todas, ignorando mayúsculas y minúsculas

```
"_é_à vu"
```

# JavaScript

## Template Strings (ECMA6)

```
<script>
    var numUno = 10;
    var numDos = 20;
    document.write(`La suma de ${numUno} + ${numDos} =`, numUno + numDos);
</script>
```



Las variables numUno y numDos son sustituídas por sus valores cuando escribimos el nombre entre \${ }.

# Eventos

- Señales generadas cuando ocurren acciones específicas.
- Son la base para la **interacción con el usuario**.
- Los eventos se pueden asociar a una función que se ejecute cuando el evento se produzca: **funciones manejadoras**.

Función manejadora:

```
<input type="text" onChange="validarCampo(this)">
```

hace referencia  
al evento

Bloque de instrucciones:

```
<input type="text" onChange="  
    if (parseInt(this.value) <= 5)  
    { alert('Escriba un número mayor que 5.') ; }">
```

# Eventos

## Tipos de eventos

Evento	Descripción
<b>blur</b>	Cuando el usuario hace click fuera de un campo en un formulario
<b>click</b>	Cuando el usuario hace click en un botón, un link o un elemento de un formulario
<b>change</b>	Cuando el usuario cambia el valor de un campo
<b>focus</b>	Cuando se activa el foco en un campo de entrada
<b>load</b>	Cuando se carga una página en el navegador
<b>mousevoer</b>	Cuando el puntero del ratón pasa por encima de un hipervínculo
<b>select</b>	Cuando el usuario selecciona un campo de un elemento de formulario
<b>submit</b>	Cuando el usuario envía un formulario
<b>unload</b>	Cuando un usuario abandona una página (para cerrar la ventana o cambiar de página)

# Eventos

## Ejemplo eventos onLoad y onUnload

```
<html>
<head><title>Ejemplo Eventos </title></head>
<body  onLoad="alert('Bienvenido!');"
      onUnload="alert('Adios!');">
<h1>Página con eventos</h1>
</body>
</html>
```



# ACTIVIDAD

---

Evento onclick

## OBJETIVO

Procesar eventos de usuarios mediante funciones JS manejadoras.

## INSTRUCCIONES

1. Copia el ejemplo de formulario de autenticación que se acompaña:  
`login.html` y `login.css`
2. Modifica el formulario de `login.html` para que cuando se haga clic en el botón se muestre un mensaje con `alert()`.



**20 min**

```
<!doctype html>
<html>
<head>
    <title>Login Page</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="login.css"/>
</head>

<body>
<div class="login-page">
    <div class="form">
        <form class="login-form">
            <input type="text" id="username" placeholder="Enter username..."/><br>
            <input type="password" id="password" placeholder="Type password..."/><br>
            <button type="button">Login</button>
        </form>
    </div>
</div>
</body>
</html>
```

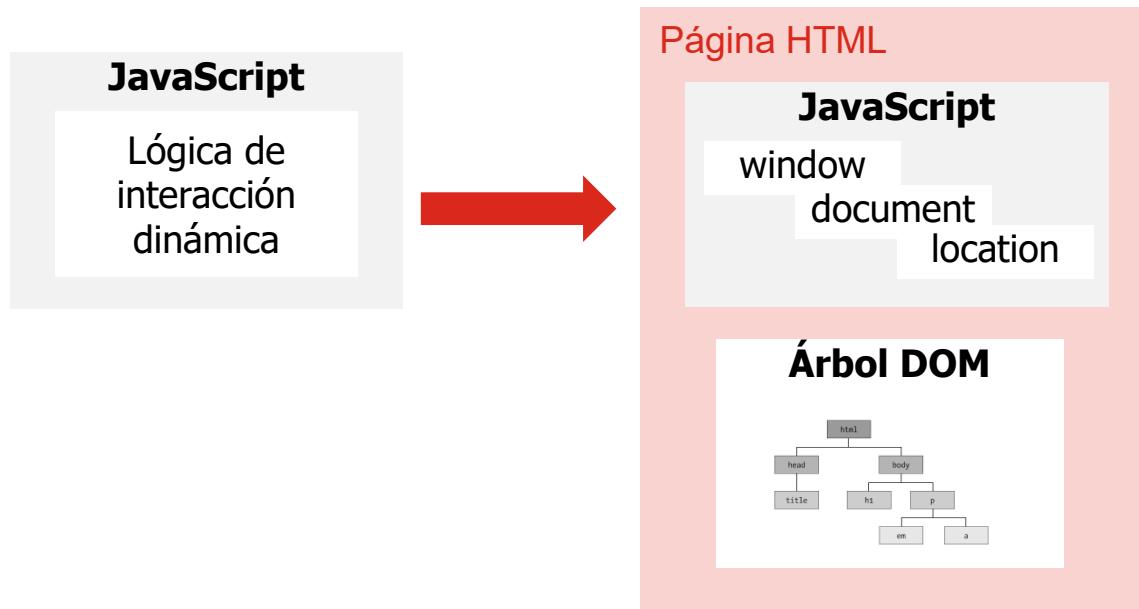


20 min

# DOM: Document Object Model

## Modelo de Objetos del Documento

- El DOM (Modelo de Objetos del Documento) es un estándar del W3C (World Wide Web Consortium) que nos **permite el acceso y la modificación** de los diferentes elementos de un documento HTML.



«El DOM es una (...) interface que permite a programas y scripts acceder dinámicamente y actualizar el contenido, estructura y estilo de un documento.»  
Definición del W3C

# DOM: Document Object Model

## Funcionamiento

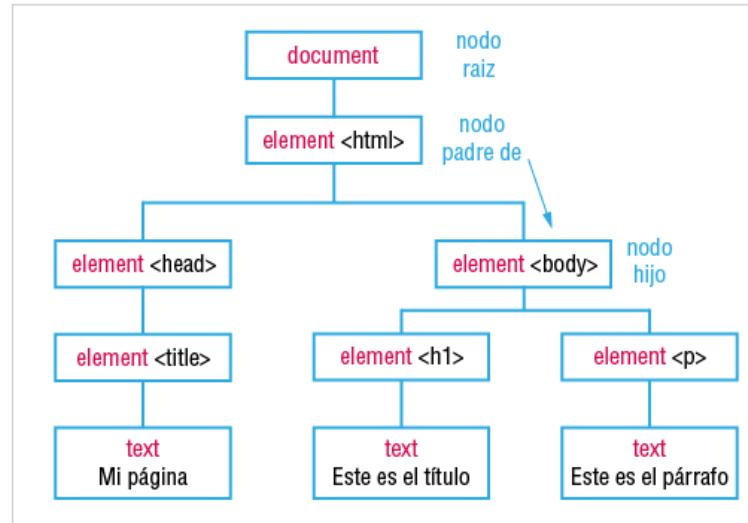
- Cuando creamos un documento HTML, este se compone de diferentes etiquetas que el navegador debe interpretar para poder mostrar los elementos visuales que queremos que el usuario vea en pantalla.
- El navegador realiza la tarea de interpretar las etiquetas y genera una estructura (el DOM) en la que organiza jerárquicamente todos los elementos que conforman el documento (<html>, <body>, <a>...).
- La estructura DOM o **diagrama en árbol** está compuesta por nodos que pueden ser padres, hijos o hermanos de otros nodos. Existen diferentes tipos de nodo según su contenido.

# DOM: Document Object Model

Documento HTML

```
<html>
  <head>
    <title> Mi página </title>
  </head>
  <body>
    <h1> Este es el título </h1>
    <p> Este es el párrafo </p>
  </body>
</html>
```

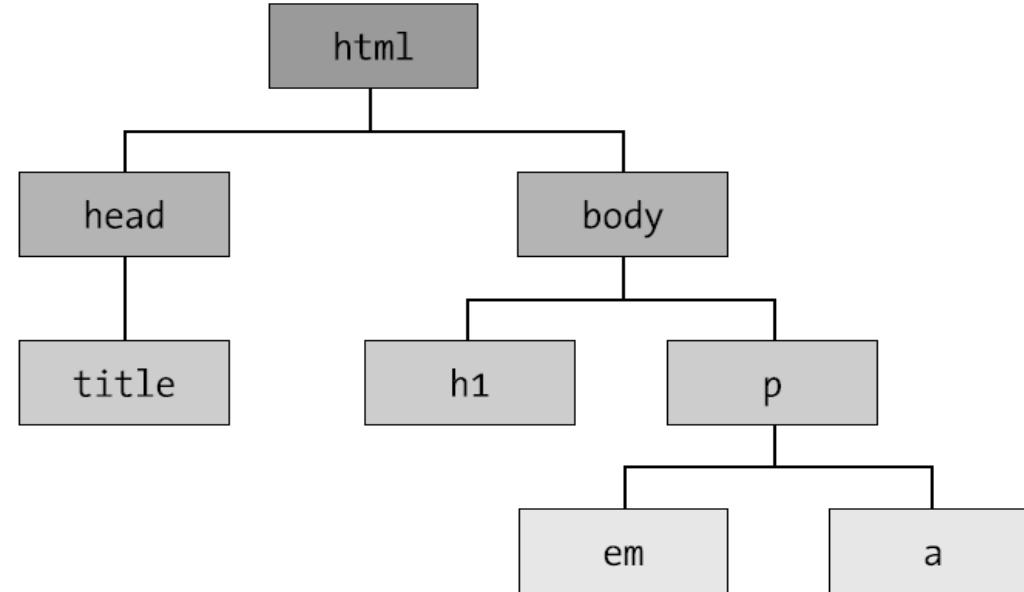
Presentación DOM



# DOM: Document Object Model

## Árbol DOM del documento

```
<html>
  <head>
    <title>DOM</title>
  </head>
  <body>
    <h1>DOM</h1>
    <p>
      Ejemplo <em>árbol DOM</em>
      <a href="#">Volver</a>
    </p>
  </body>
</html>
```



# DOM: Document Object Model

## Objetos predefinidos

- Un documento HTML tiene objetos JavaScript predefinidos, que pueden ser manipulados de forma programática con JavaScript y la **interfaz de programación de aplicaciones (DOM API)**.

**window**: referencia a la ventana propiamente tal. Todos los elementos pertenecen implícitamente al objeto window. Si se abre una nueva ventana o pestaña, corresponde a otro window.

Por ejemplo, el método alert(...) pertenece al objeto window

**document**: es la raíz del documento HTML, y el "padre" de todos los elementos. Provee atributos y métodos para acceder a los elementos de la página, lo que permite modificarlos dinámicamente, y de este modo realizar los efectos interactivos.

Por ejemplo, document.write() imprime texto

# DOM: Document Object Model

## Objetos predefinidos

- **navigator**: contiene información sobre el navegador. Permite particularizar la programación, lo que es útil cuando existen diferencias entre los navegadores. También permite restringir una aplicación a un tipo o versión de navegador.
- **location**: contiene información sobre la URL de la página. Si se modifica, se abre una nueva URL, lo que permite utilizarlo para navegar programáticamente.
- **screen**: contiene información de la pantalla, incluyendo alto y ancho. Permite ajustar una página en función de la resolución de pantalla.
- **history**: contiene el historial de URLs visitadas, lo que permite volver atrás programáticamente.

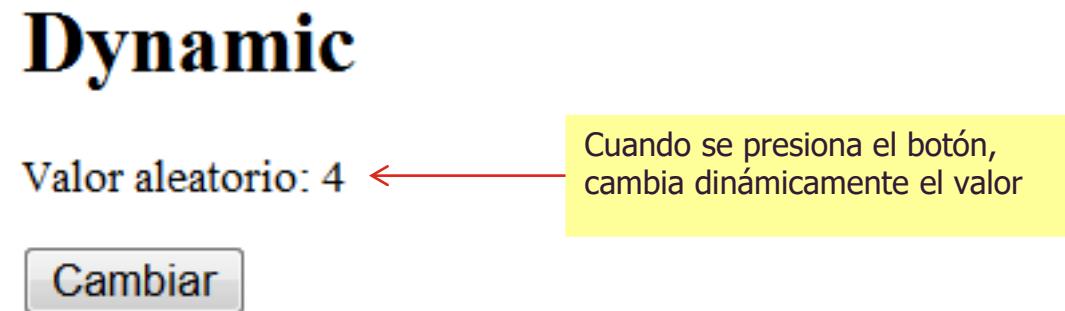
# Casos de usos del lenguaje JavaScript

- Modificar el contenido **HTML** y los estilos **CSS**, a través de la utilización del **API DOM**.
- Controlar la **visibilidad** de una sección de la página, lo que permite hacer efectos como pestañas, menús o acordeón.
- Cambio dinámico de **estilo css**, lo que permite por ejemplo mostrar selecciones, o efectos de habilitación y deshabilitación.
- Crear **controles personalizados** que no existen nativamente en HTML, como barras de progreso.
- **Actualizar** secciones del documento pantalla, como por ejemplo una de noticias, sin tener que recargar la página.

# Interfaz de programación de aplicaciones (DOM API)

## Cambio de contenido

- Para cambiar el contenido HTML de una parte de una pantalla en forma dinámica, se utiliza la propiedad `innerHTML`. Aunque no es parte del estándar, todos los navegadores conocidos manejan la propiedad. Por ejemplo, se puede modificar el contenido de un elemento.



# Interfaz de programación de aplicaciones (DOM API)

## Cambio de contenido

- Código HTML y JavaScript:

```
<h1>Dynamic</h1>
<p id="par">
    Valor aleatorio: 0
</p>
```

```
<input type="button" value="Cambiar"
       onclick="change()">
```

Al presionar el botón, se invoca la función change()

```
<script>
    function change() {
        var p = document.getElementById("par");
        var num = 1 + Math.floor((9*Math.random()));
        p.innerHTML = "Valor aleatorio: " + num;
    }
</script>
```

Obtiene el objeto asociado al párrafo, con el id="par"

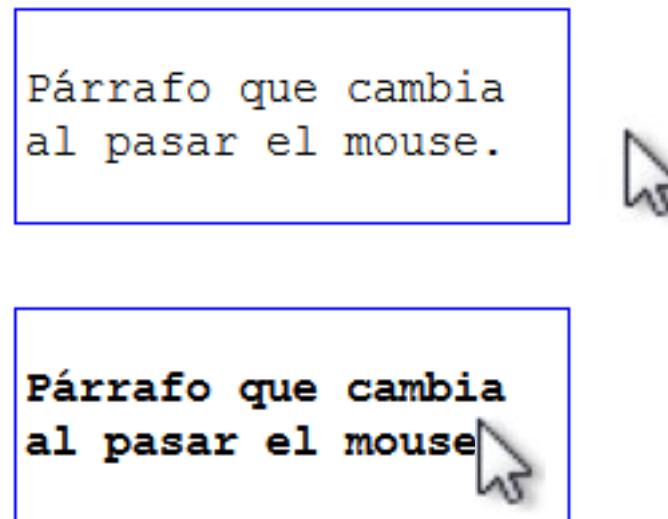
Genera un número aleatorio entre 1 y 9

Cambia el contenido del objeto p, que corresponde al párrafo

# Interfaz de programación de aplicaciones (DOM API)

## Cambio de regla de estilo

- Podemos cambiar dinámicamente la clase de estilo de una sección de pantalla. Por ejemplo, un párrafo tiene un estilo normal y uno destacado, y se quiere que al pasar el puntero sobre él, cambie.



```
.sec {  
    font-family: Courier;  
    border: 1px solid blue;  
    padding: 3px;  
    cursor: default;  
    width: 200px;  
}  
  
.pnormal {  
    font-weight: normal;  
}  
  
.phigh {  
    font-weight: bold;  
}
```

# Interfaz de programación de aplicaciones (DOM API)

## Cambio de regla de estilo

- Código HTML y JavaScript:

```
<div class="sec"
    onmouseover="change(true)"
    onmouseout="change(false)">
    <p id="par" class="pnorma...
        Párrafo que cambia al pasar el mouse.
    </p>
</div>
```

Cuando se pasa el mouse sobre el área del div, se invoca la función change(true)

Cuando se saca el mouse del área del div, se invoca la función change(false)

```
<script>
    function change(opt) {
        var p = document.getElementById("par");
        p.className = (opt) ? "phigh" : "pnorma...
    }
</script>
```

Obtiene el objeto asociado al párrafo, con el id="par"

Cambia dinámicamente la clase de estilo del párrafo, según el parámetro



# ACTIVIDAD

---

**HTML dinámico y API DOM**

## OBJETIVO

Modificar elementos del DOM dinámicamente usando el API DOM.

## INSTRUCCIONES

1. Copia el código de ejemplo anexo y guárdalo en un fichero HTML.
2. Abrelo en el navegador y pulsa sobre Listar, aparecerá una tabla que permite eliminar sus filas mediante el botón Delete.
3. Añade un botón a la derecha de Listar. Implementa el código JS necesario para que al hacer clic se añada una nueva fila al final de la tabla. El método `HTMLTableElement.insertRow()` te puede ayudar:  
<https://developer.mozilla.org/es/docs/Web/API/HTMLTableElement/insertRow>
4. No te olvides de ponerle un título al botón, por ejemplo, añadir ;)



30 min

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>JavaScript can change table content on the fly</title>
    <link rel="stylesheet" href="build-table.css">
</head>
<body>
    <h2>Listado de Contactos</h2>
    <button onclick="buildTable();">Listar</button>
    <table id="myTable">
        <thead><tr><th>Nombre</th><th>Apellido</th></tr></thead>
        <tbody id="tableContactBody"></tbody>
    </table>
    <footer>JavaScript</footer>
<script>
var persons = [
    {name:'Michel', age:51},
    {name:'Henri', age:20},
    {name:'Francois', age:29}
];
var tableBody = document.querySelector("#tableContactBody");
```



30 min

```
function buildTable() {  
    for(let j in persons) { // Get the body of the table using the selector API  
        addLineToHTMLTable(persons[j].name, persons[j].age); } }  
  
var pos;  
function removeRow(buttonCellParm) {  
    pos = buttonCellParm.parentNode.parentNode.rowIndex;  
    document.getElementById("myTable").deleteRow(pos); }  
function addLineToHTMLTable(name, age) { // Add a row to the HTML table  
    // Add a new row at the end of the table  
    var newRow = tableBody.insertRow();  
    // add new cells to the row  
    var firstNameCell = newRow.insertCell();  
    firstNameCell.innerHTML = name;  
    var lastNameCell = newRow.insertCell();  
    lastNameCell.innerHTML = age;  
    var buttonCell = newRow.insertCell();  
    //buttonCell.innerHTML = '<button onclick="removeRow(this)">Delete</button>';  
    var boton = document.createElement('input');  
    boton.setAttribute(name="type", value="button");  
    boton.setAttribute(name="value", value="Delete");  
    boton.setAttribute(name="onclick", value="removeRow(this)");  
    buttonCell.append(boton); }  
</script></body></html>
```



30 min

## Listado de Contactos

Listar

Nombre	Apellido	
Michel	51	<a href="#">Delete</a>
Henri	20	<a href="#">Delete</a>
Francois	29	<a href="#">Delete</a>



30 min

# Interfaz de programación de aplicaciones (DOM API)

## Control de visibilidad

- Para controlar la **visibilidad** de una parte de una página, se controla el atributo de estilo llamado **display** asociado a la rama del árbol DOM correspondiente.

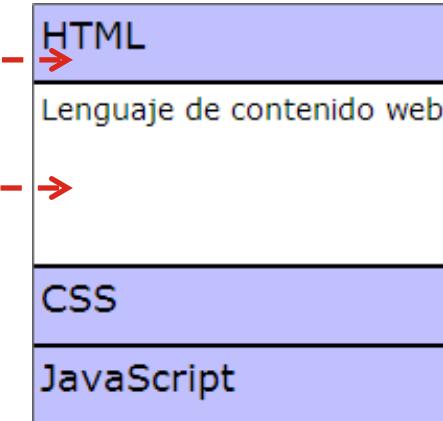


# Interfaz de programación de aplicaciones (DOM API)

## Control de visibilidad

- Estilos:

```
.header {  
    border: 1px solid black;  
    cursor: pointer;  
    width: 200px;  
    height: 30px;  
    padding: 3px;  
    background-color: #C0C0FF;  
    font-size: 1.1em;  
}  
  
.content {  
    border: 1px solid black;  
    font-size: 0.9em;  
    width: 200px;  
    height: 80px;  
    padding: 3px;  
}
```



# Interfaz de programación de aplicaciones (DOM API)

## Control de visibilidad

- Elementos HTML:

```
<div class="header" onclick="visib('ht')">
    HTML
</div>
<div class="content" id="ht">
    Lenguaje de contenido web
</div>
<div class="header" onclick="visib('cs')">
    CSS
</div>
<div class="content" id="cs" style="display: none">
    Estilos gráficos
</div>
<div class="header" onclick="visib('js')">
    JavaScript
</div>
<div class="content" id="js" style="display: none">
    Lenguaje para interacción dinámica
</div>
```

Al hacer click sobre el div, se ejecuta el método JavaScript "visib"



JavaScript

Inicialmente oculta el contenido del div y el espacio que utiliza

# Interfaz de programación de aplicaciones (DOM API)

## Control de visibilidad

- Función JavaScript:

```
function visib(divId) {  
    var ar = ["ht", "cs", "js"];  
    for (var i=0; i < ar.length; i++) {  
        var divElem = document.getElementById(ar[i]);  
        divElem.style.display =  
            (divId == ar[i]) ? "block": "none";  
    };  
}
```

array con ids de los contenidos

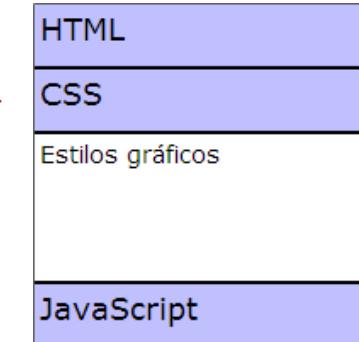
Accede al objeto del árbol DOM con el contenido, dado el id.

Cambia el atributo de estilos display, en función del parámetro:  
• block: mostrar contenido  
• none: ocultar contenido

divId = "ht"



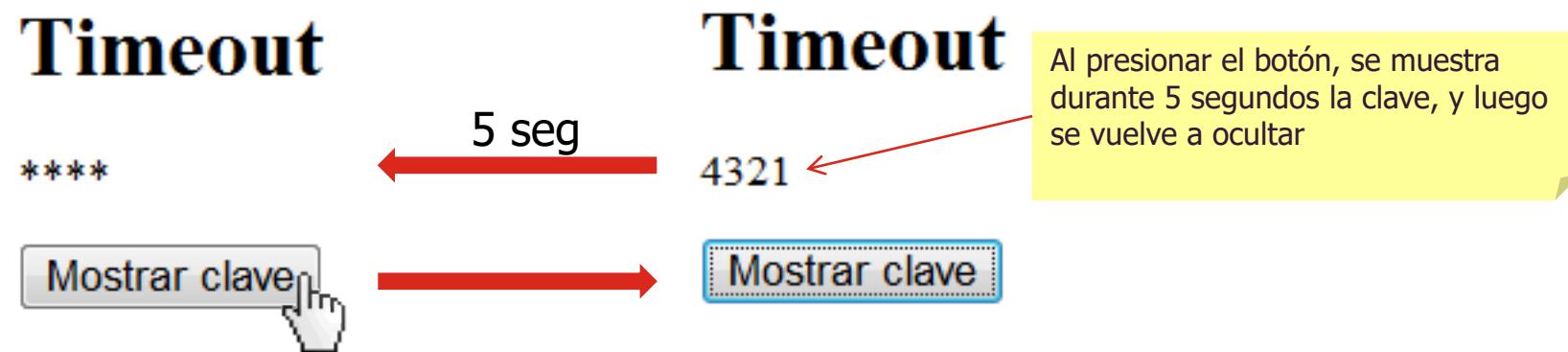
divId = "cs"



# Interfaz de programación de aplicaciones (DOM API)

## Timeout

- A través del método setTimeout del objeto implícito window, se pueden ejecutar funciones un tiempo después. Esto permite controlar efectos en el tiempo. Por ejemplo, una página que muestra una clave durante 5 segundos, y luego la oculta.
- Resultado:



# Interfaz de programación de aplicaciones (DOM API)

## Cambio de contenido

- Código HTML y JavaScript:

```
<h1>Timeout</h1>
<p id="pwd">****</p>
<input type="button" value="Mostrar clave" onclick="showPwd()">
```

Al presionar el botón se invoca la función manejadora showPwd()

```
<script>
  function showPwd() {
    var pwd = "4321";
    document.getElementById("pwd").innerHTML = pwd;
    setTimeout(hidePwd, 5000);
  }
```

Cambia el contenido del objeto con id="pwd", que corresponde al párrafo que muestra la clave

```
function hidePwd() {
  document.getElementById("pwd").innerHTML = "****";
```

5000 milisegundos después, invoca la función hidePwd. Se coloca la referencia, sin paréntesis

Vuelve a colocar los \*\*\*\* en el contenido del párrafo con la clave



# ACTIVIDAD

**Validación de formulario**

## OBJETIVO

Procesar eventos de usuarios y validarlos mediante una función JS manejadora.

## INSTRUCCIONES

1. Modifica el formulario de autenticación de la actividad previa, Evento onclick, `login.html` y asocia al evento onclick un función manejadora llamada `validar()`.
2. Para implementar el código de `validar()` crearemos un fichero Javascript `login.js`
3. La lógica de `validar()` será comparar los valores introducidos en los campos de usuario y password y validarlos con tu nombre y una contraseña fácil. Si los valores coinciden mostrar cuadro de éxito o fallo en caso contrario.



30 min

# Casos de uso JavaScript

## API de AJAX con soporte de datos JSON

- AJAX significa **Asynchronous JavaScript And XML**.
- La API de AJAX permite la ejecución de llamadas asíncronas al servidor, que devuelven datos para actualizar fragmentos de una página.
- Se basan en el objeto XMLHttpRequest.
- Existen dos versiones o niveles de llamada.
- El nivel 1, es prácticamente compatible con todos los navegadores, incluyendo los más antiguos.
- El nivel 2, ya disponible en todos los navegadores modernos, permite controlar el nivel de progreso de la llamada, con lo que se simplifica el código.
- Mediante este tipo de llamadas, podemos conseguir actualizaciones parciales de la página, logrando una experiencia de usuario mucho más agradable.

```
{ "id" : 1,  
  "name" : "Me",  
  "email" : "me@gmail.com" }
```

{JSON}

# Casos de uso JavaScript

## API de AJAX con soporte de datos JSON

- Hay que tener en cuenta que, por motivos de seguridad, este API genera un error en la respuesta cuando se intenta llamar a servicios de este tipo en un dominio distinto.
- En otros entornos, como Silverlight, los ficheros de servidor CrossDomain.xml, permiten establecer los permisos para este tipo de llamadas.
- Un objeto XMLHttpRequest puede devolver datos en formatos diversos, como XML, JSON, Texto plano, etc.
- La petición adopta un formato como el siguiente:

```
GET /recurso HTTP/1.1
Host: host:puerto
User-Agent: Mozilla/5.0
```

# Casos de uso JavaScript

## API de AJAX con soporte de datos JSON

```
function leerVentas() {  
    var url = "http://localhost:1565/DemosJS5/ventas.json";  
    var request = new XMLHttpRequest();  
    request.responseType = "application/json"; //Predeterminado  
    request.open("GET", url); //Solo configura la llamada  
    request.onload = function() {  
        if (request.status == 200) { //200 significa correcto.  
            actualizarIU(request.responseText);  
        }  
        request.send(null);  
    }  
}
```

# Casos de uso JavaScript

## API de AJAX con soporte de datos JSON

```
function actualizarIU(respuestaJSON) {
    var DivVentas = document.getElementById("DIV_ventas");
    var ventas = JSON.parse(respuestaJSON);
    for (var i = 0; i < ventas.length; i++) {
        var venta = ventas[i];
        var div = document.createElement("div");
        div.setAttribute("class", "FormatoVenta");
        div.innerHTML = "Total ventas de " + venta.name + ": " + venta.TVentas;
        DivVentas.appendChild(div);
    }
}
```

# Casos de uso JavaScript

## API de AJAX con soporte de datos JSON

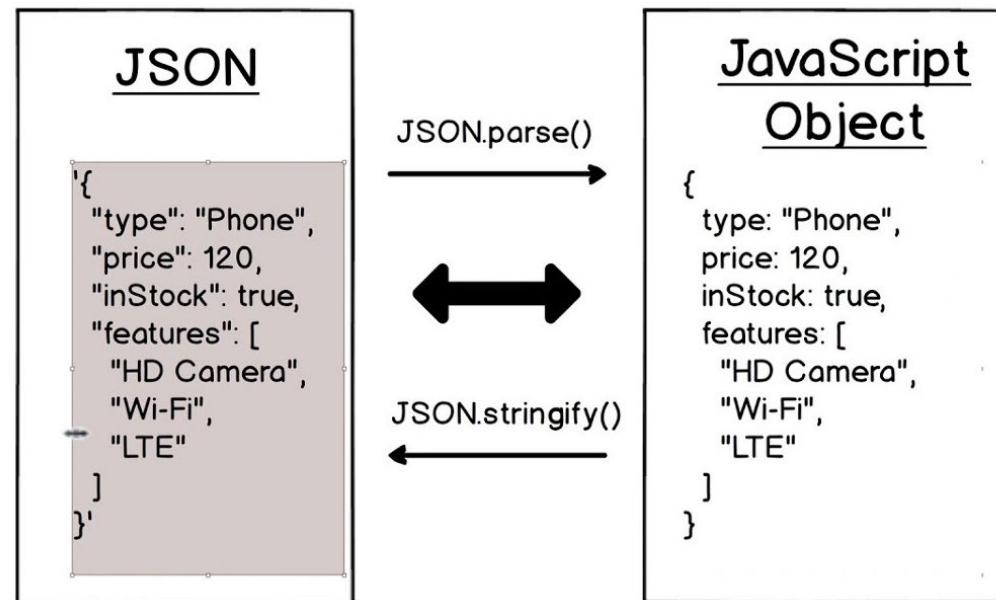
- En el caso de que tengamos que dar soporte a algunos navegadores muy antiguos deberíamos comprobar el evento `onreadystatechange`

```
function leerVentas_XHRv1() {  
    var url = "http://localhost:1565/DemosJS5/ventas.json";  
    var request = new XMLHttpRequest();  
    request.open("GET", url);  
    request.onreadystatechange = function() {  
        if (request.readyState == 4 && request.status == 200) {  
            actualizarIU(request.responseText);  
        }  
    };  
    request.send(null);  
}
```

# Casos de uso JavaScript

## API de AJAX con soporte de datos JSON

- Para la manipulación de datos JSON, las implementaciones de JavaScript de los navegadores actuales, incluyen un objeto del mismo nombre, que incorpora métodos útiles para facilitar su manejo.
  - stringify()** convierte un objeto JavaScript a cadena.
  - parse()** realiza la operación contraria.



# Casos de uso JavaScript

## API Storage: Almacenamiento local y de sesión

- El objetivo de estas API es la persistencia de información entre distintas peticiones Web.
- Almacenar datos entre peticiones, de una forma distinta y más adecuada a la que es posible hacer ahora mediante cookies.
- Estos servicios son accedidos mediante objetos vinculados con el objeto window:  
**window.sessionStorage** y **window.localStorage**
- El funcionamiento es a través de una "base de datos" manejada internamente por el navegador y accedida mediante estas API.
- El primer paso es la comprobación del soporte por parte del navegador utilizado.

# Casos de uso JavaScript

## Almacenamiento local y de sesión

```
function comprobarSoporte() {
    //sessionStorage
    if (window.sessionStorage) {
        alert('Este navegador soporta sessionStorage');
    } else {
        alert('Este navegador NO soporta sessionStorage');
    }
    //localStorage
    if (window.localStorage) {
        alert('Este navegador soporta localStorage');
    } else {
        alert('Este navegador NO soporta localStorage');
    }
}
```

# Casos de uso JavaScript

## Almacenamiento local y de sesión

- `sessionStorage` y `localStorage` disponen de varios métodos para asignación (escritura) y recuperación (lectura) de información.
- En la actualidad solo encontramos soporte para cadenas.
- La instrucción de escritura utilizada es `setItem(clave, valor)`.
- Recibe parejas clave/valor y las almacena para recuperación posterior en la base de datos asignada (al navegador).
- La lectura se realiza mediante el método `getItem(clave)` que permite recuperar cualquier valor guardado a partir de su clave y la devuelve en formato cadena.

# Casos de uso JavaScript

## Almacenamiento local y de sesión

```
<body>
  <form action="valida.html" id="Formulario" >
    <p>Almacenamiento de valores de sesión</p>
    <input type="text" id="claveini" />
    <input type="button" value="Guardar"
      onclick="escribirClave('Dato');" /><br />
    <input type="text" id="claveLeida" />
    <input type="button" value="Leer" onclick="leerClave('Dato');"
      />
  </form>
</body>
```

# Casos de uso JavaScript

## Almacenamiento local y de sesión

```
function escribirClave(datos) {
    var valor = document.getElementById("claveini").value;
    window.sessionStorage.setItem(datos, valor);
}

function leerClave(datos) {
    var valor = window.sessionStorage.getItem(datos);
    document.getElementById("claveLeida").value = valor;
}
```

# Casos de uso JavaScript

## Almacenamiento local y de sesión

- También existen limitaciones a la cantidad de información que puede guardarse en el apartado valor vinculado con una clave dada.
- Existen unos valores de cuota que deberán de ser configurables por el usuario:

sessionStorage	localStorage
La persistencia se limita a la página o solapa de navegación dentro del sitio	La persistencia se mantiene incluso después de cerrar el navegador
Los valores almacenados o recuperados solo son visibles desde la página o solapa que los creó.	Los valores se mantienen entre distintas ventanas o solapas ejecutando el mismo origen URL.

# localStorage vs sessionStorage

localStorage	sessionStorage
Stores data with <b>no expiration</b> date	Stores data <b>only for a session</b> (until the tab/browser is closed)
Gets cleared through JS or Browser cache / Locally Stored Data	Storage limit is larger than a cookie (at least 5 MB). Max 4KB
Changes available for all current and future visits to the site	Persists over page reloads and restores. Opening a new tab/window will initiate a new session.
	Data is never transferred to the server*
<ul style="list-style-type: none"><li>▪ Both extend <b>Storage</b></li><li>▪ Both can only be read on client-side</li><li>▪ Web storage is per origin (<b>per domain and protocol</b>)</li><li>▪ Both allow to storage JS primitive types (integers, strings, ...) but nor arrays neither objects → we must convert them to <b>JSON</b></li></ul>	

# localStorage methods

Method	Description
<code>setItem()</code>	Add key and value to local storage
<code>getItem()</code>	Retrieve a value by the key
<code>removeItem()</code>	Remove an item by key
<code>clear()</code>	Clear all storage
<code>localStorage.length</code>	Nº de elementos almacenados en el espacio local.

# HTML Web Storage Objects

**Check** browser **support** for localStorage and sessionStorage:

```
if (typeof(Storage) !== "undefined") {  
    // Code for LocalStorage/sessionStorage.  
} else {  
    // Sorry! No Web Storage support...  
}
```

# IndexedDB API

IndexedDB persistently allows to store objects indexed with a "key"

- It's an asynchronous API
- IndexedDB is object-oriented
- IndexedDB databases store key-value pairs
- It's build on a transactional database model
- It does not use SQL

# Casos de uso JavaScript

## API para acceso a archivos locales (File API)

- Podemos programar una sencilla interfaz que habilite la selección de uno o más archivos por parte del usuario y muestre posteriormente la información de detalles de esos archivos en la página.
- En la parte HTML 5 tendríamos:

```
<!DOCTYPE html>
<html>
<head>
    <title>API de ficheros</title>
</head>
<body>

    <h3>Seleccionar fichero(s)</h3>
    <p><input id="files-upload" type="file" multiple /></p>
    <h3>Ficheros subidos</h3>
    <ul id="file-list"><li>(no hay ficheros todavía)</li></ul>
```

# Casos de uso JavaScript

## API para acceso a archivos locales (File API)

```
<script>
  (function () {
    // Recuperamos las referencias a los dos elementos implicados
    var filesUpload = document.getElementById("files-upload"),
        fileList = document.getElementById("file-list");

    // Función para recorrer el conjunto de archivos
    // y actualizar la interfaz de usuario con la información
    // sobre ellos.
    function traverseFiles(files) {
      var li,file,fileInfo;
      fileList.innerHTML = "";
```

# Casos de uso JavaScript

## API para acceso a archivos locales (File API)

```
for (var i = 0, il = files.length; i < il; i++) {
    li = document.createElement("li");
    file = files[i];
    fileInfo = "<div><strong>Fichero:</strong> " + file.name + "</div>";
    fileInfo += "<div><strong>Tamaño:</strong> " + file.size + " bytes</div>";
    fileInfo += "<div><strong>Tipo:</strong> " + file.type + "</div>";
    li.innerHTML = fileInfo;
    fileList.appendChild(li);
};

filesUpload.onchange = function () {
    traverseFiles(this.files);
};

})();
```



# ACTIVIDAD

---

Webapp Agenda

## OBJETIVO

Leer y escribir objetos en el local storage usando JavaScript y el API DOM.

## INSTRUCCIONES

1. Analiza el código fuente facilitado (agenda.html, agenda.css y agenda.js) de una pequeña webapp de agenda de contactos. Los contactos de la agenda están modelados como objetos de JavaScript. Abre agenda.html en el navegador y juega con la webapp para probar su funcionamiento.
2. Realiza los cambios que consideres necesarios para que la webapp permita guardar contactos como un par {clave=nombre, valor=email} en el local storage.



60 min

3. A continuación añade funcionalidad para poder recuperar el email de un contacto después de haber escrito el nombre en un campo de entrada.
4. Por último, añade un botón que permita borrar un contacto por su nombre. Y otro que elimine todos los contactos previa confirmación mediante cuadro de diálogo.
5. Opcionalmente puedes dedicar tiempo a mejorar la apariencia visual de la tabla, que no le vendría mal ;)

Recursos: [ficheros de la webapp de contactos](#)

Personal informations

Name :  Email :

#### List of contacts

Angus Young	angus@acdc.com
Arnold Schwarzenegger	T2@terminator.com
Jimi Hendrix	jimi@rip.com
Robert Fripp	robert.fripp@kingcrimson.com



60 min

# Casos de uso JavaScript

## Geo-Localización

- Con `navigator.geolocation`, controlamos el soporte, a la vez que programamos las peticiones y respuestas.
- Una llamada a este objeto desde JavaScript, nos devolverá un valor falso si no existe.
- Para programar el funcionamiento para un navegador podríamos usar:

```
<head>
    <meta charset="utf-8" />
    <title>Geo-localización nativa con HTML 5</title>
</head>
<body onload="comprobarNavegador()">
    <h1>Ejemplo de Geo-localización nativa con HTML 5</h1>
    <p id="nivelSoporte">La Geo-localización nativa NO está soportada en este
navegador.</p>
    <h2>Ubicación actual:</h2>
    <h5>Latitud: <span id="latitud">n/c</span></h5>
    <h5>Longitud: <span id="longitud">n/c</span></h5>
    <h5>Precisión: <span id="precision">n/c</span></h5>
```

# Casos de uso JavaScript

## Geo-Localización

- Primero, comprobamos el soporte en el navegador llamando a la función de geolocalización:

```
<script>  
function comprobarNavegador() {  
    if(navigator.geolocation) {  
        document.getElementById("nivelSoporte").innerHTML =  
            "La Geo-Localización HTML5 está soportada en este navegador.";  
        navigator.geolocation.getCurrentPosition(updateLocation);  
    }  
}  
}
```

- A continuación, consultamos los datos geográficos.

# Casos de uso JavaScript

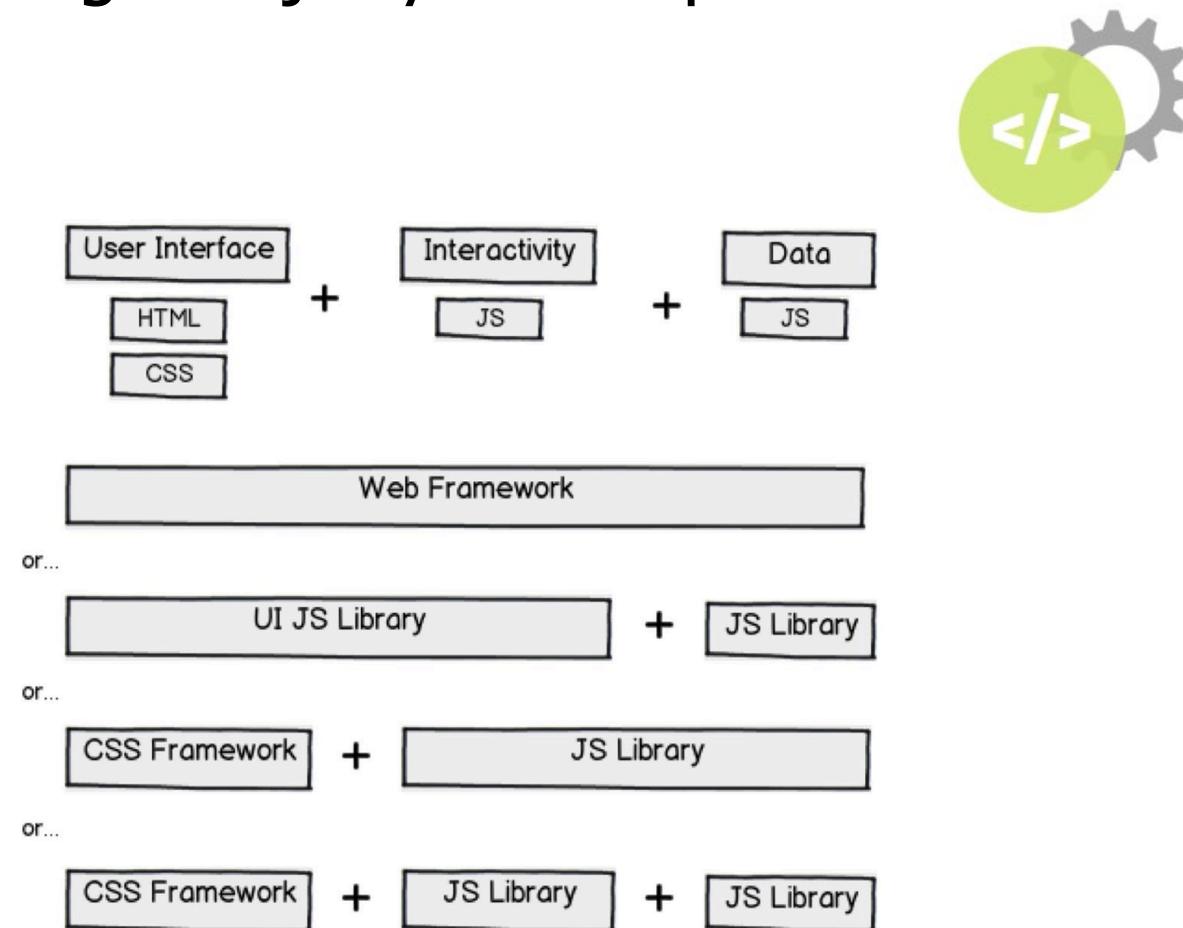
## Geo-Localización

```
function updateLocation(position) {
    var lat = position.coords.latitude;
    var lon = position.coords.longitude;
    var pre = position.coords.accuracy;
    if (!lat || !lon) {
        document.getElementById("nivelSoporte").innerHTML =
        "Geo-Localizacion HTML5 soportada, pero no en este momento.";
        return;
    }
    document.getElementById("latitud").innerHTML = lat;
    document.getElementById("longitud").innerHTML = lon;
    document.getElementById("precision").innerHTML= pre + " ms.";
}
</script>
</body>
</html>
```

# Bibliotecas y Frameworks

Código JavaScript para hacer algo mejor y más rápido.

- Bibliotecas
  - moment.js
  - jQuery
- Frameworks de componentes
  - Dojo
  - jQuery UI
  - jQueryMobile
  - Sencha
  - KendoUI
- Frameworks para SPA
  - Angular
  - React
  - Vue
  - Node JS
- Editores, IDEs: VS Code, IntelliJ, Sublime Text, Atom, Notepad++, Eclipse



# Desarrollo y reutilización de componentes de software multimedia mediante lenguajes de guión

## 5. Contenido multimedia

- Definición de multimedia. Tipos de recursos multimedia
- Inclusión de contenido multimedia en páginas web
- Gráficos, audio, vídeo, animaciones y elementos interactivos

# Multimedia: Tipos

- Multimedia hace referencia a cualquier objeto o sistema que utiliza **múltiples medios** de expresión digitales para presentar la información.
- Los medios pueden ser variados, desde **texto e imágenes, gráficos, animación, sonido, video, alto nivel de interactividad**.
- La realidad virtual es una extensión de multimedia que utiliza los elementos básicos de ésta como imágenes, sonido y animación.



# Multimedia: Audio

## Formatos y Compatibilidad con navegadores

Formato	Media Type
MP3	audio/mpeg
WAV	audio/wav
OGG	audio/ogg

Browser	MP3	WAV	OGG
Edge/IE	SI	NO	NO
Chrome	SI	SI	SI
Firefox	SI	SI	SI
Safari	SI	SI	NO
Opera	SI	SI	SI

# Multimedia: Audio

- Para incluir y reproducir un archivo de audio se usa el elemento HTML **<audio>**

```
<audio controls>
  <source src="opera.ogg" type="audio/ogg">
  <source src="opera.mp3" type="audio/mpeg">
Your browser does not support the audio tag.
</video>
```

- El atributo **controls** muestra controles de audio, como reproducción, pausa y volumen.
- El elemento **<source>** permite especificar archivos de audio alternativos entre los que el navegador puede elegir. El navegador utilizará el primer formato reconocido.
- El texto entre **<audio> </audio>** sólo se mostrará en los navegadores que no lo soportan. Navegadores compatibles: Chrome 4.0, Internet Explorer 9.0, Firefox 3.5, Opera 10.5 y Safari 4.0.

# Multimedia: Video

## Formatos y Compatibilidad con navegadores

Formato	Media Type
MP4	video/mp4
WebM	video/webm
Ogg	video/ogg

Browser	MP4	WebM	Ogg
Edge	SI	SI	SI
Chrome	SI	SI	SI
Firefox	SI	SI	SI
Safari	SI	SI	NO
Opera	SI	SI	SI

# Multimedia: Video

- Para incluir y reproducir un video se usa el elemento HTML **<video>**

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

- El atributo **controls** muestra controles de video, como reproducción, pausa y volumen. Podemos usar **autoplay** para que el video se inicie automáticamente.
- Es recomendable especificar los atributos **width** y **height** (ancho y alto).
- El elemento **<source>** permite especificar archivos de video alternativos entre los que el navegador puede elegir. El navegador utilizará el primer formato reconocido.
- El texto entre **<video> </video>** sólo se mostrará en los navegadores que no lo soportan. Navegadores compatibles: Chrome 4.0, Internet Explorer 9.0, Firefox 3.5, Opera 10.5 y Safari 4.0.

# Multimedia

Formato	Extensión	Descripción
MPEG	.mpg .mpeg	MPEG. Developed by the Moving Pictures Expert Group. The first popular video format on the web. Not supported anymore in HTML.
AVI	.avi	AVI (Audio Video Interleave). Developed by Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.
WMV	.wmv	WMV (Windows Media Video). Developed by Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.
QuickTime	.mov	QuickTime. Developed by Apple. Commonly used in video cameras and TV hardware. Plays well on Apple computers, but not in web browsers.
RealVideo	.rm .ram	RealVideo. Developed by Real Media to allow video streaming with low bandwidths. Does not play in web browsers.
Flash	.swf .flv	Flash. Developed by Macromedia. Often requires an extra component (plug-in) to play in web browsers.
Ogg	.ogg	Theora Ogg. Developed by the Xiph.Org Foundation. Supported by HTML.
WebM	.webm	WebM. Developed by Mozilla, Opera, Adobe, and Google. Supported by HTML.
MPEG-4 or MP4	.mp4	MP4. Developed by the Moving Pictures Expert Group. Commonly used in video cameras and TV hardware. Supported by all browsers and recommended by YouTube.

Formatos compatibles con el estándar HTML: MP4, WebM y Ogg

02

# Desarrollo y reutilización de componentes de software multimedia mediante lenguajes de guión

## 6. Single Page Application (SPA)

- Modelo SPA vs modelo tradicional
- Web component
- Desarrollo web basado en componentes web
- Frameworks

## SPA: Single Page Application

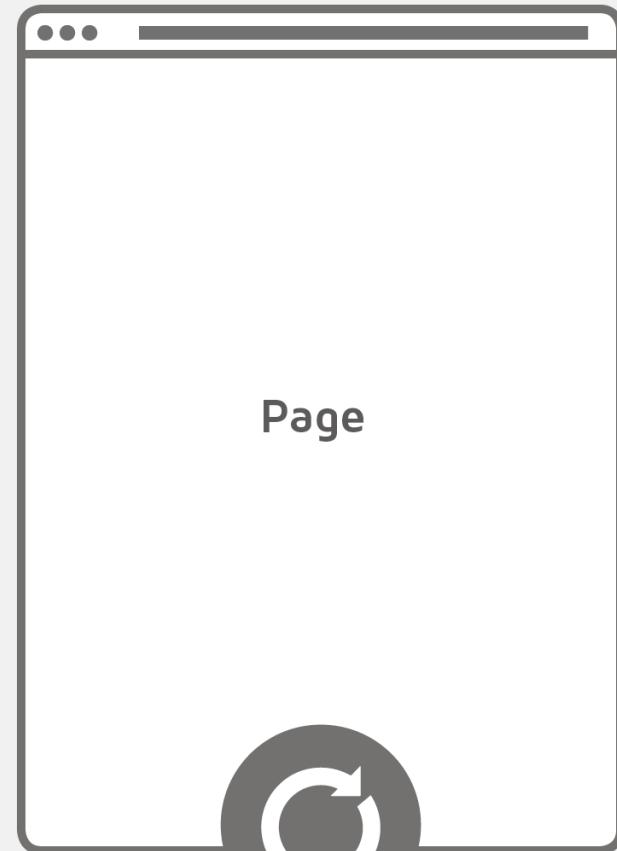
- **Una sola página** con todo el contenido de la web.
- Todo el código HTML, CSS y JavaScript en un **único archivo** que se carga una sola vez.
- Cada vista se corresponde con un **componente web** (cada vista es similar a una página tradicional).
- Se ejecutan en el lado del cliente (navegador).
- Lenguaje de programación: **JavaScript + Bibliotecas y Frameworks** (Angular, React, Polymer, Ember JS, Vue, ...).
- Modelo usado por Gmail, Google Maps, Facebook, Twitter, Google Drive...

## Single Page Application



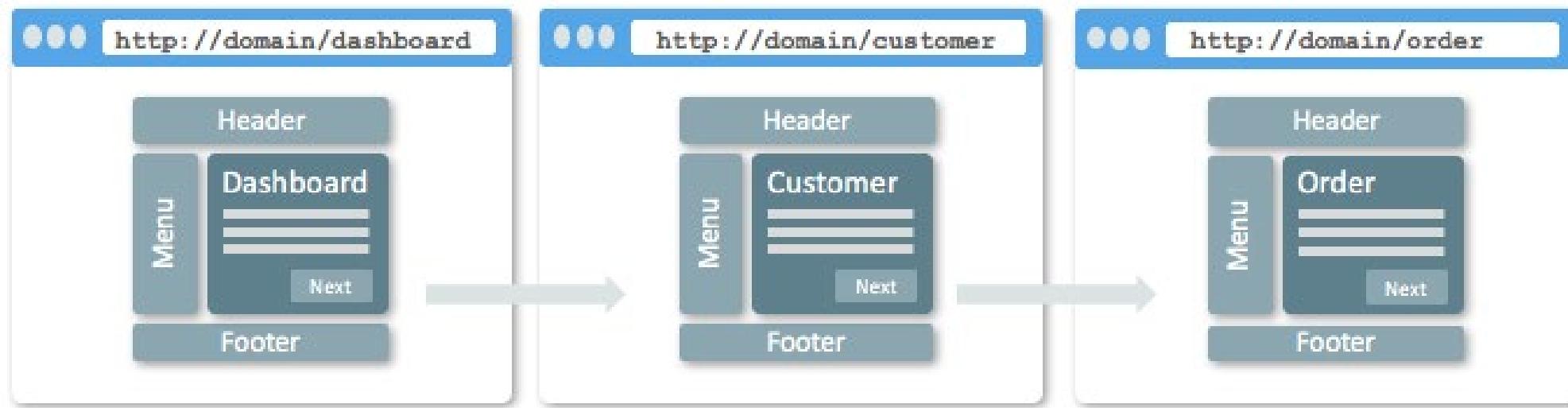
No page refresh on request

## Traditional Web Application



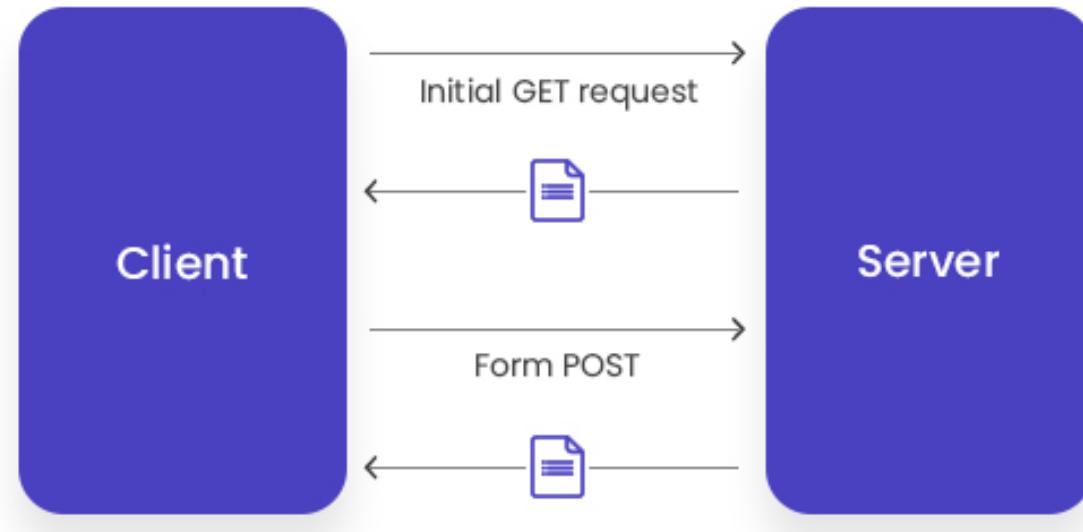
Whole page refresh on request

# SPA - Single Page Application



# SPA Funcionamiento

Traditional Page Lifecycle



SPA Lifecycle



# SPA Web Component

<custom-element>

Structure

```
<div>
  <input>
  <p>
    <span></span>
  </p>
</div>
```

Behavior

```
tag.verifyAccount();
```

Styles

```
<style>
  p { color: red; }
</style>
```

## SPA: Ventajas

- **Navegación más rápida:** todos los contenidos precargados.
- Nuevos contenidos se cargan **dinámicamente**.
- Como backend sólo requieren de una API que proporcione el contenido a mostrar.
- Backend y Frontend totalmente **separados e independientes**.

## SPA: Inconvenientes

- Compatibilidad con **SEO no es buena** (contenido se carga dinámicamente), posible solución: Server Side Rendering.

# Bibliografía y webgrafía



ECMAScript

<https://www.ecma-international.org/technical-committees/tc39/>

World Wide Web Consortium (W3C)

<https://www.w3.org/>

# Bibliografía y webgrafía



Webs de referencia

<http://www.w3schools.com/>

<http://stackoverflow.com/>

<http://www.codecademy.com/>

<https://www.w3counter.com/globalstats.php>

03

# **UF1843. Aplicación de técnicas de usabilidad y accesibilidad en el entorno cliente**

# Objetivos del curso



Aplicar técnicas de usabilidad y accesibilidad en el desarrollo de interfaces de usuario.

- Distinguir y explicar pautas de accesibilidad al contenido en los documentos elaborados para permitir una mejor navegación y comprensión de los usuarios.
- Distinguir y explicar pautas de usabilidad al contenido en los documentos elaborados para permitir una mejor calidad, efectividad y satisfacción de los usuarios.
- Crear y mantener componentes software y documentos aplicar normas de accesibilidad y usabilidad para mejorar su utilización

- 1. Accesibilidad web
- 2. Usabilidad web

Bibliografía y webgrafía

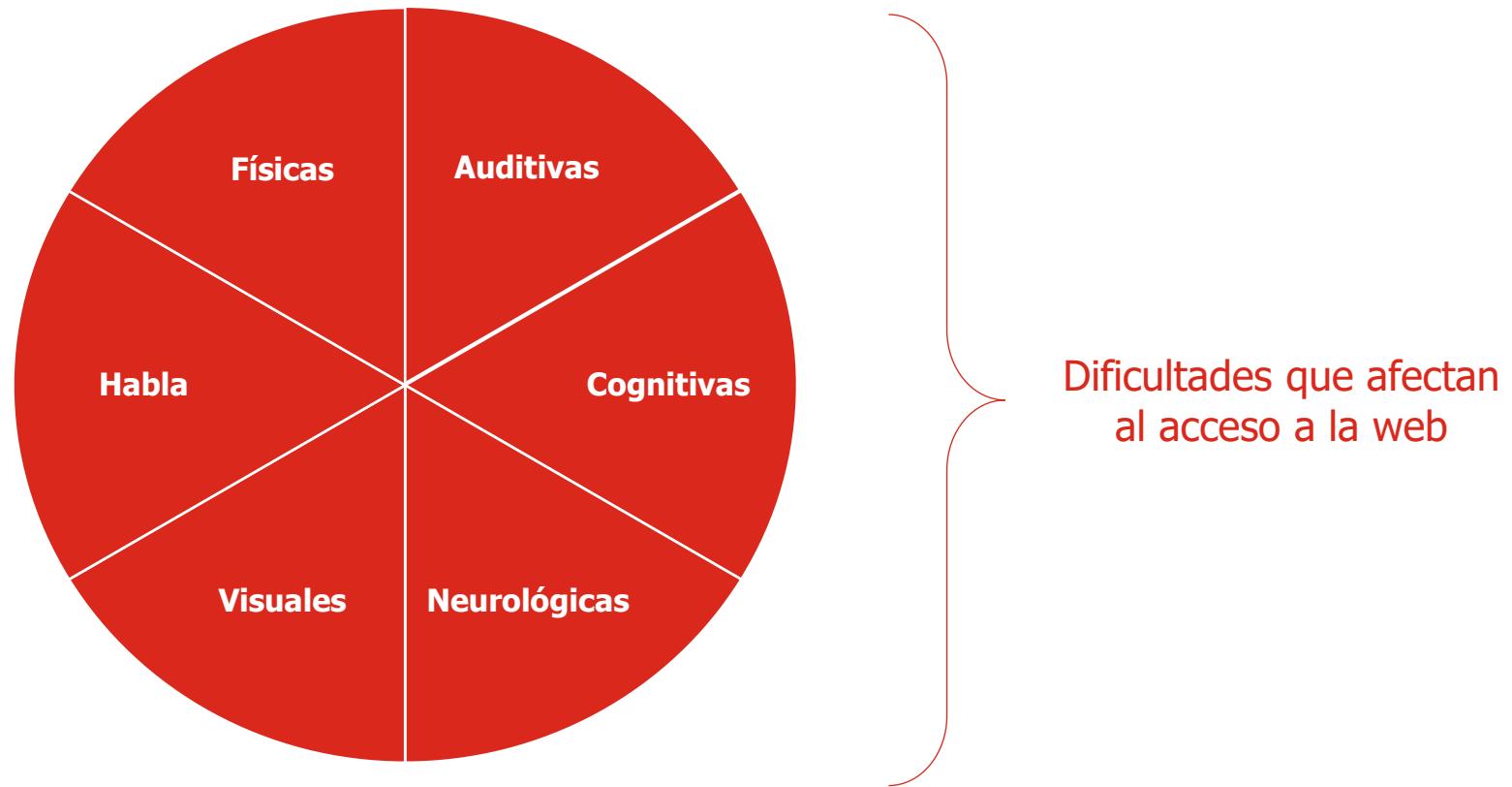
# Aplicación de técnicas de usabilidad y accesibilidad en el entorno cliente

## 1. Accesibilidad web

- Definición de accesibilidad web
- Ventajas y dificultades en la implantación de la accesibilidad web
- Normativa y estándares sobre accesibilidad web
- Guías para el cumplimiento de normativas y estándares
- Descripción del proceso de la conformidad en accesibilidad web
- Tecnologías donde la accesibilidad es aplicable
- Herramientas para la validación de la accesibilidad
- Evolución de la accesibilidad. Nuevas tendencias

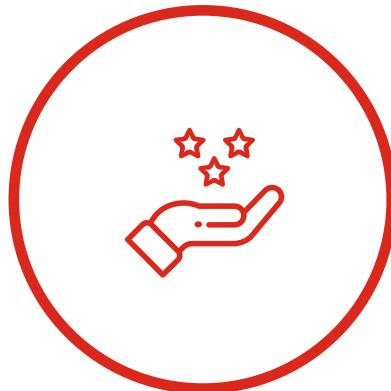
# Accesibilidad web

- Herramientas y tecnologías diseñadas y desarrolladas para que **cualquier persona pueda percibir, comprender, navegar, contribuir e interactuar** con la web sin dificultades.



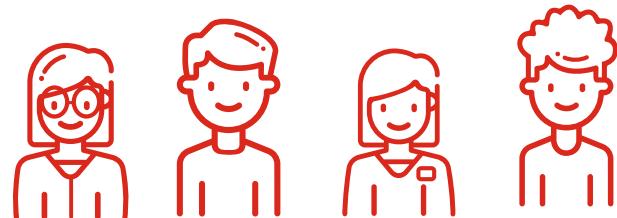
# Accesibilidad web

- Beneficios de la accesibilidad web:



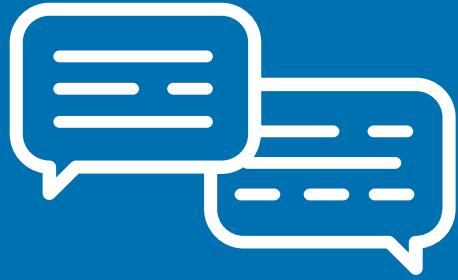
.....

- Personas utilizando **teléfonos móviles, relojes inteligentes, televisores inteligentes y otros dispositivos** con pantallas pequeñas, diferentes modos de entrada, etc.
- Personas **mayores** cuyas habilidades cambian con la edad
- Personas con **discapacidades temporales**, como puede ser un brazo roto o la pérdida de unas gafas
- Personas con **limitaciones por su ubicación**, como puede ser bajo la luz del sol o en un entorno donde no se puede escuchar audio
- Personas con **conexión lenta a Internet** o que tienen ancho de banda limitado o costoso.



# Accesibilidad web





# DEBATE

---

**Factores que afectan a la  
accesibilidad web**

## **OBJETIVO**

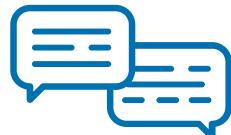
Tener conciencia de los principales factores que contribuyen a mejorar la accesibilidad web.

## **INSTRUCCIONES**

1. Ver el video “Perspectivas de accesibilidad web” (7:36 min de duración) que habla de la importancia que tiene para las personas con discapacidad y de la utilidad para todos. Además conocerás el impacto de la accesibilidad y los beneficios para todos en una variedad de situaciones reales y cotidianas: <https://youtu.be/3f31oufqFSM>
  
2. Para obtener más información sobre este tema puedes consultar:  
<https://www.w3.org/WAI/perspectives/>



**20 min**



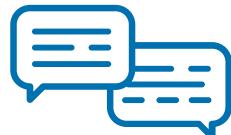
## **REFLEXIONES**

Después de ver el video, debate en grupo las siguientes cuestiones y otras que consideres oportunas y/o interesantes:

1. ¿Ya eras consciente o tenías conocimiento de todos los factores y situaciones vistos en el video?
2. ¿Cuál de ellos te ha sorprendido más debido a que nunca lo hubieras tenido en cuenta?
3. ¿Qué situación/es o factor/es crees que, a nivel técnico, es más difícil o complejo de implementar?



**20 min**



# Accesibilidad web

## Estándares internacionales para la accesibilidad web



- Iniciativa de Accesibilidad Web del W3C ([WAI](#))
- ISO: ISO/IEC 40500
- Pautas de Accesibilidad al Contenido Web ([WCAG](#))
- Pautas de Accesibilidad para Herramientas de Autor ([ATAG](#))
- Aplicaciones de Internet Enriquecidas Accesibles ([WAI-ARIA](#))
- Pautas de accesibilidad para el agente de usuario ([UAAG](#))
- Otros recursos importantes: Resumen de los estándares de accesibilidad de W3C

# Accesibilidad web

## W3C Web Accessibility Initiative (WAI)



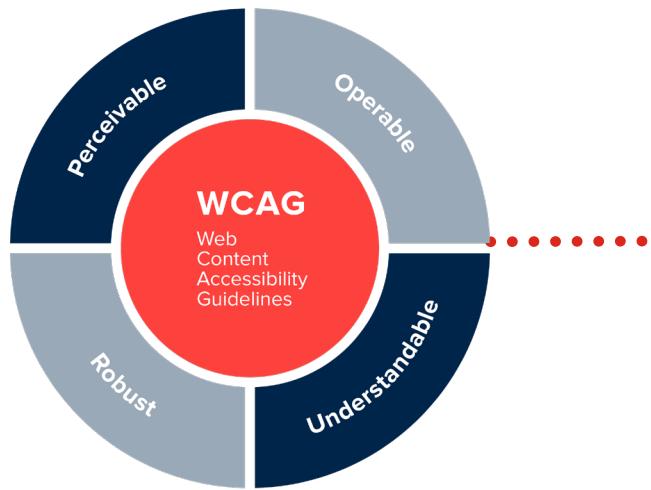
.....

- La Iniciativa de **Accesibilidad Web del W3C (WAI)** desarrolla especificaciones técnicas, pautas, técnicas y recursos que describen soluciones.
- Proporciona un foro internacional para la colaboración entre la industria, las organizaciones de personas con discapacidad, los investigadores de accesibilidad, el gobierno y otros interesados en la accesibilidad web.



# Accesibilidad web

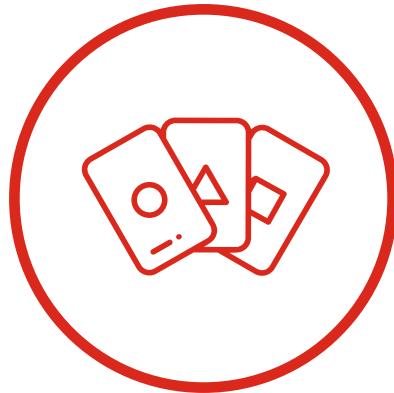
## Web Content Accessibility Guidelines (WCAG 2.0)



- Pautas a seguir, hasta que la legislación oficial cubra todo el contenido publicado en la web.
- ResponsiveVoice puede ayudar a que el sitio sea **WCAG 2.0 compatible**, especialmente para las personas discapacitadas, y aquellos que sufren de una variedad de obstáculos incluyendo la dislexia, trastorno de la visión, y cuadriplejia.
- La experiencia de navegación actual pone muchos obstáculos en frente de ellos, llegando a ser innecesariamente difícil y frustrante.
- El **estándar WCAG 2.0** se creó para resolver ese problema.

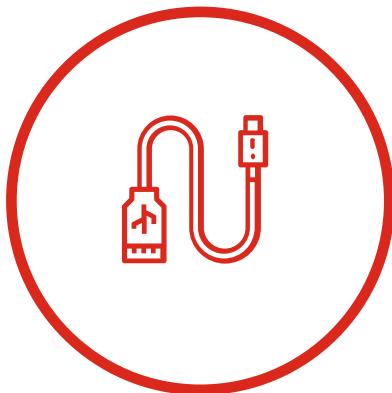
# Principios básicos de accesibilidad web

Perceptible



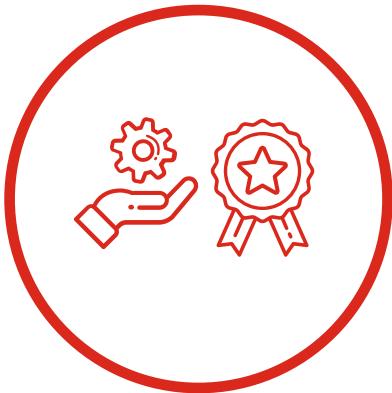
Disponible a través de la vista, el oído o el tacto

Operable



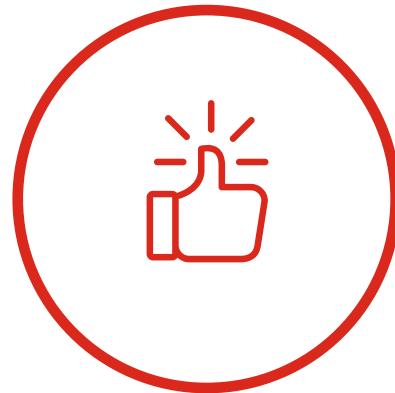
Compatible con teclado o mouse

Robusto



- Funciona en navegadores, tecnologías asistencia, disp móviles, disp / nav antiguos...
- Cumple con los estándares

Comprendible



Legible, claro y fácil de entender

# Pautas y técnicas de accesibilidad

## Texto alternativo

- › Atributo 'alt' descriptivo con detalle. Si una imagen es decorativa o redundante en el texto cercano, debe tener un texto alternativo vacío ( alt="" ).

## Legibilidad

- › Lenguaje más simple y apropiado para su contenido y audiencia.
- › Revisar la ortografía, la gramática y el nivel de grado de lectura.

## Navegación de contenido

- › Estructura semántica con regiones , encabezados y listas.
- › Enlace de salto para ayudar a los usuarios a acceder directamente al contenido de la página principal.

## Tablas de datos

- › Encabezados de tabla de datos con `<th scope="col">` para encabezados de columna y `<th scope="row">` para los encabezados de fila. Agregar `<caption;>` para la tabla datos.

## Color

- › No usar el color solo para transmitir información. Especial cuidado combinaciones de colores rojo-verde.

## Formularios

- › Colocar etiquetas de formulario junto a sus controles o cerca de ellos, de modo que las etiquetas se asocien visualmente.
- › `<label>` para asociar etiquetas con controles mediante programación.

## Enlaces y Botones

- › Enlaces y botones con texto descriptivo.
- › Evitar "haga clic aquí", "aquí", "más", "más información", "leer más", "continuar", etc.

## Multimedia

- › Videos y el audio en vivo deben tener subtítulos y una transcripción.
- › El audio archivado debe tener una transcripción.

## Documentos

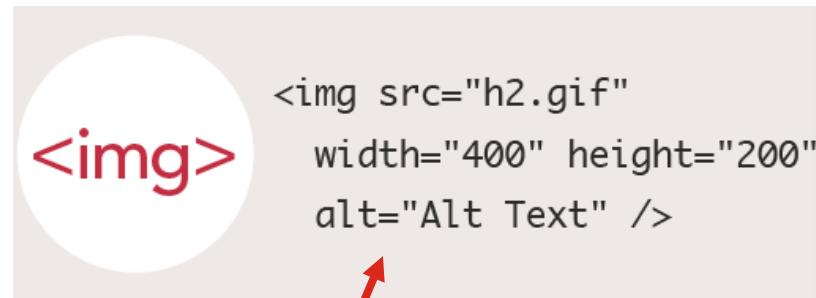
- › Para una accesibilidad más sólida usar HTML en lugar de formatos de documentos propietarios.

# Pautas y técnicas de accesibilidad

## Texto alternativo

- El texto alternativo conocido como **atributo alt** permite añadir la descripción y nos da información de la apariencia y función de una imagen y de otros elementos de una página web.
- Funciones del atributo alt:
  - Informar y contextualizar el contenido
  - Facilitar información ante problemas concretos
  - Servicio de ayuda a crawlers

"El texto alternativo mejora la accesibilidad porque ayuda a las personas que no pueden ver imágenes en las páginas web, como los usuarios que emplean lectores de pantalla o los que tienen conexiones con poco ancho de banda" –  
*Soporte de Google*



texto alternativo que se muestra en el caso que la imagen no pueda ser mostrada

# Pautas y técnicas de accesibilidad

## Opciones válidas y no recomendadas de texto alternativo

- alt muestra lo que es la imagen, pero no es realmente descriptivo.



```

```



- Mejor texto alternativo. No solo indica qué animal es, sino que describe y contextualiza la imagen:



```

```

- No recomendado:



```

```

# Pautas y técnicas de accesibilidad

## Enlace de salto

- Omitir la sección de navegación permite a usuarios de lectores de pantalla y a personas que no pueden usar el mouse omitir largas listas de enlaces, como la navegación principal de un sitio web.
- **Saltar navegación** es simplemente un **enlace** en la parte superior de la página web que, cuando se hace clic, lo posiciona directamente en la sección de contenido.

```
<a id="skipnav" href="#contenido" title="Jump to content"> Saltar navegación </a>
<ul id="nav">
    <li><a href="home.html" title=""> Página principal </a> </li>
    <li><a href="about.html" title=""> Acerca de </a> </li>
    <li><a href="blog.html" title=""> Blog </a> </li>
    <li><a href="portfolio.html" title=""> Portafolio </a> </li>
    <li><a href="contact.html" title=""> Contacto </a> </li>
</ul>
<div id = "leftCol"> <h1> Mi web</h1> <ul>
    <li><a href="http://blog.com/" title="">Blog de un amigo</a></li>
    <li> <a href="http://blog.com/" title="">Amigo de un blog</a> </li></ul>
</div>
<div id="#contenido">
<h1> Título de la página </h1>
...
</div>
```



# ACTIVIDAD

Texto Alternativo

## OBJETIVO

Aplicar buenas practices básicas de accesibilidad web.

## INSTRUCCIONES

1. Complementar elementos HTML como <a>,<img> y <input> con texto descriptivo y de ayuda mediante los atributos <alt> y <title> para dotar de accesibilidad mínima al contenido de la página.



15 min

# Accesibilidad web

## Guías y Herramientas para hacer sitios web accesibles

- Introducción a los requisitos de accesibilidad y estándares internacionales: [Principios de accesibilidad](#)
- Evaluación: [Pruebas sencillas - Una Primera Revisión \(en Inglés\)](#)
- Recursos para ayudar en la evaluación de la accesibilidad: [Visión general sobre la evaluación de la accesibilidad web](#)
- Consideraciones básicas de diseño, edición y desarrollo para accesibilidad: [Consejos Para Empezar \(en Inglés\)](#)
- Recursos útiles para saber más sobre desarrollo y diseño:
  - [Cómo cumplir las WCAG \(Referencia Rápida\)](#)
  - [Tutoriales de Accesibilidad Web](#)

# Aplicación de técnicas de usabilidad y accesibilidad en el entorno cliente

## 2. Usabilidad web

- Definición de usabilidad
- Importancia del diseño web centrado en el usuario
- Diferencias entre accesibilidad y usabilidad
- Ventajas y problemas en la combinación de accesibilidad y usabilidad
- Ventajas y dificultades en la implantación de sitios web usables
- Métodos de usabilidad
- Análisis de requerimientos de usuario
- Principios del diseño conceptual. Creación de prototipos orientados al usuario
- Pautas para la creación de sitios web usables
- Evaluación de la usabilidad

# Usabilidad web

## ¿Qué es la usabilidad?



.....

- Propiedad por la que una web resulta **fácil de usar**
- Es **eficiente**
- Aporta **experiencia satisfactoria** al usuario
- Siempre **enfocada al usuario**
- Tiene en cuenta la **psicología, destrezas y qué busca** hacer el usuario
- Un **diseño usable** hace que el usuario valore positivamente la web, permanezca más tiempo en ella y vuelva a visitarla

# Principios básicos de usabilidad

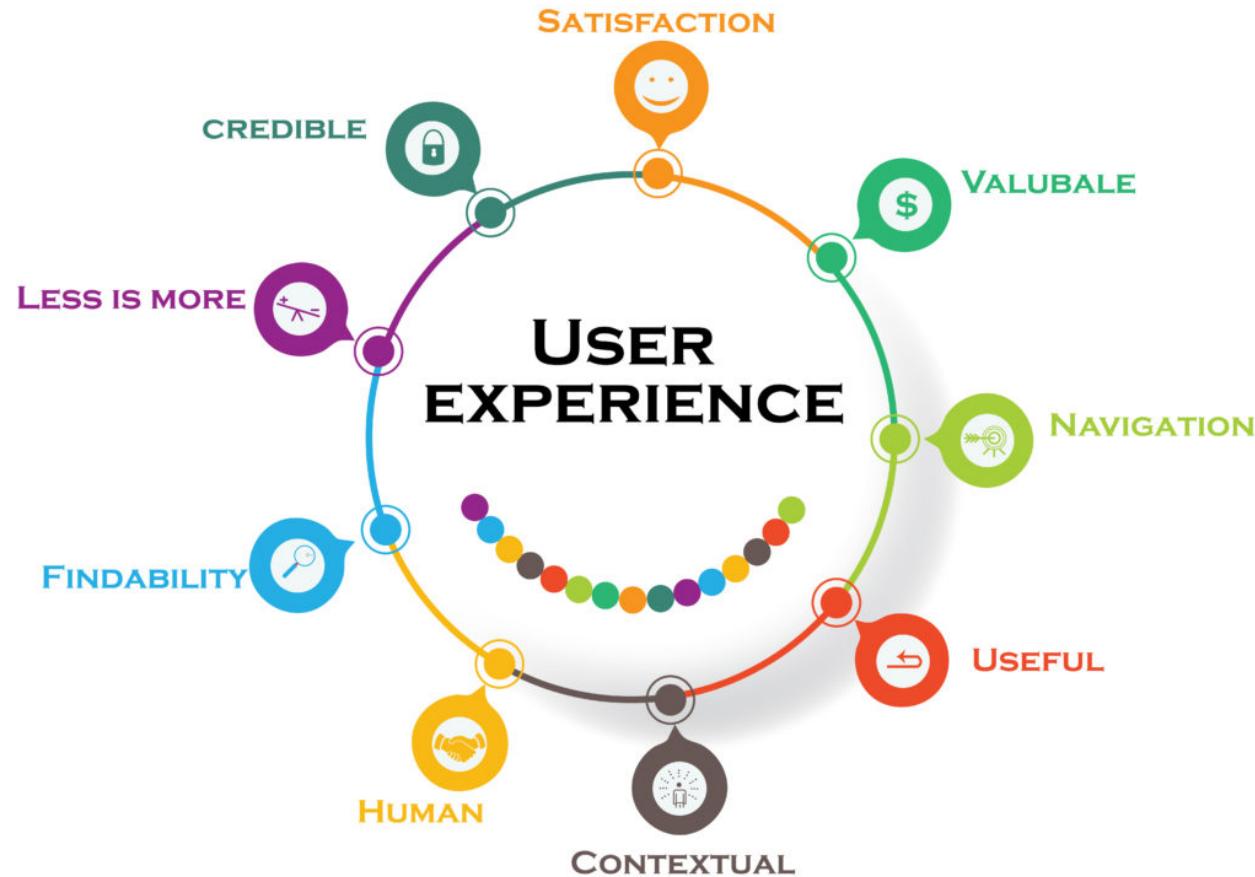


- 1. Conoce al usuario
- 2. Minimiza la memorización
- 3. Optimiza las operaciones
- 4. Gestiona los errores

En el fondo, la usabilidad no es más que aplicar el sentido común...

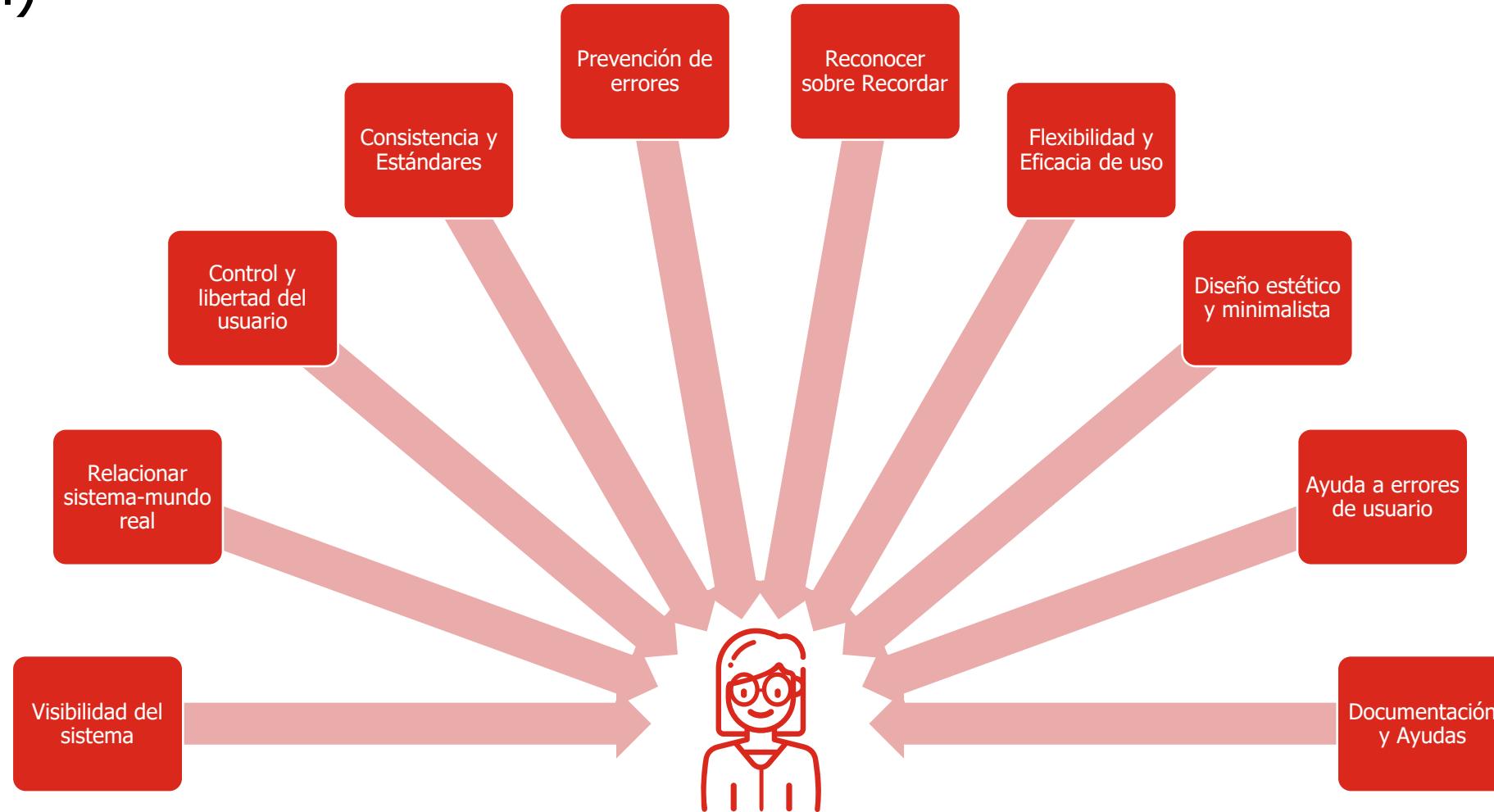
# Usabilidad web

## User Experience



# Usabilidad web

Principios fundamentales para hacer una página **user friendly** (Jakob Nielsen)

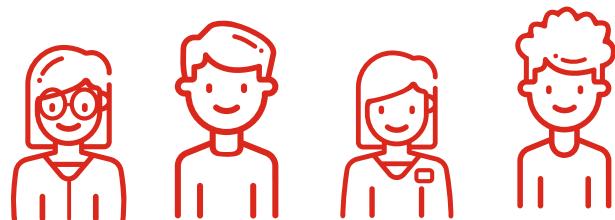


# Usabilidad web

- Atributo de **calidad** en las interfaces de usuario que indican que tan fácil son de utilizar.

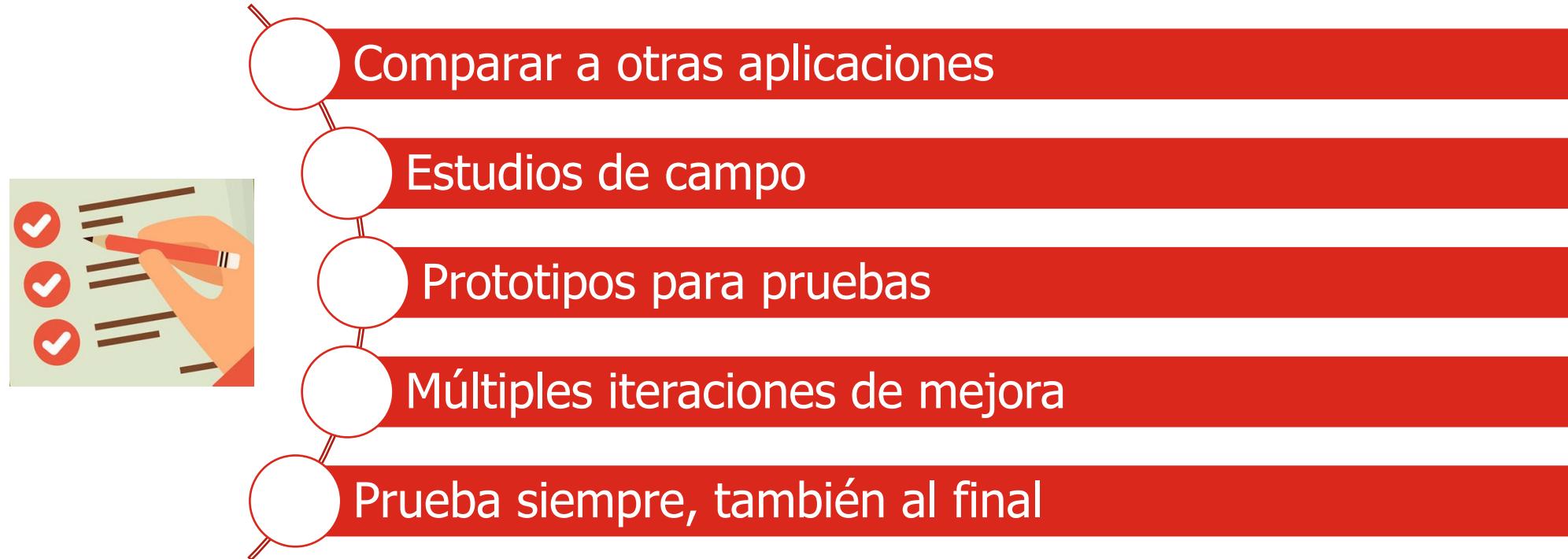


- **Aprendizaje**: que tan fácil es cumplir tareas básicas la primera vez que utilizan la interfaz.
- **Eficiencia**: una vez aprendido a usar la interfaz, que tan rápido se pueden llevar a cabo las tareas.
- **Memoria**: tras no usar un sistema después de un cierto tiempo, que tan sencillo es volver a retomarlo.
- **Errores**: ¿Cuántos errores se cometen? ¿Qué tan graves son? ¿Se pueden recuperar?
- **Satisfacción**: que tan agradable/amigable es de usar un sistema para el usuario, en cuanto a lo visual y uso.



# Usabilidad web

## ¿Cómo evaluar la usabilidad?



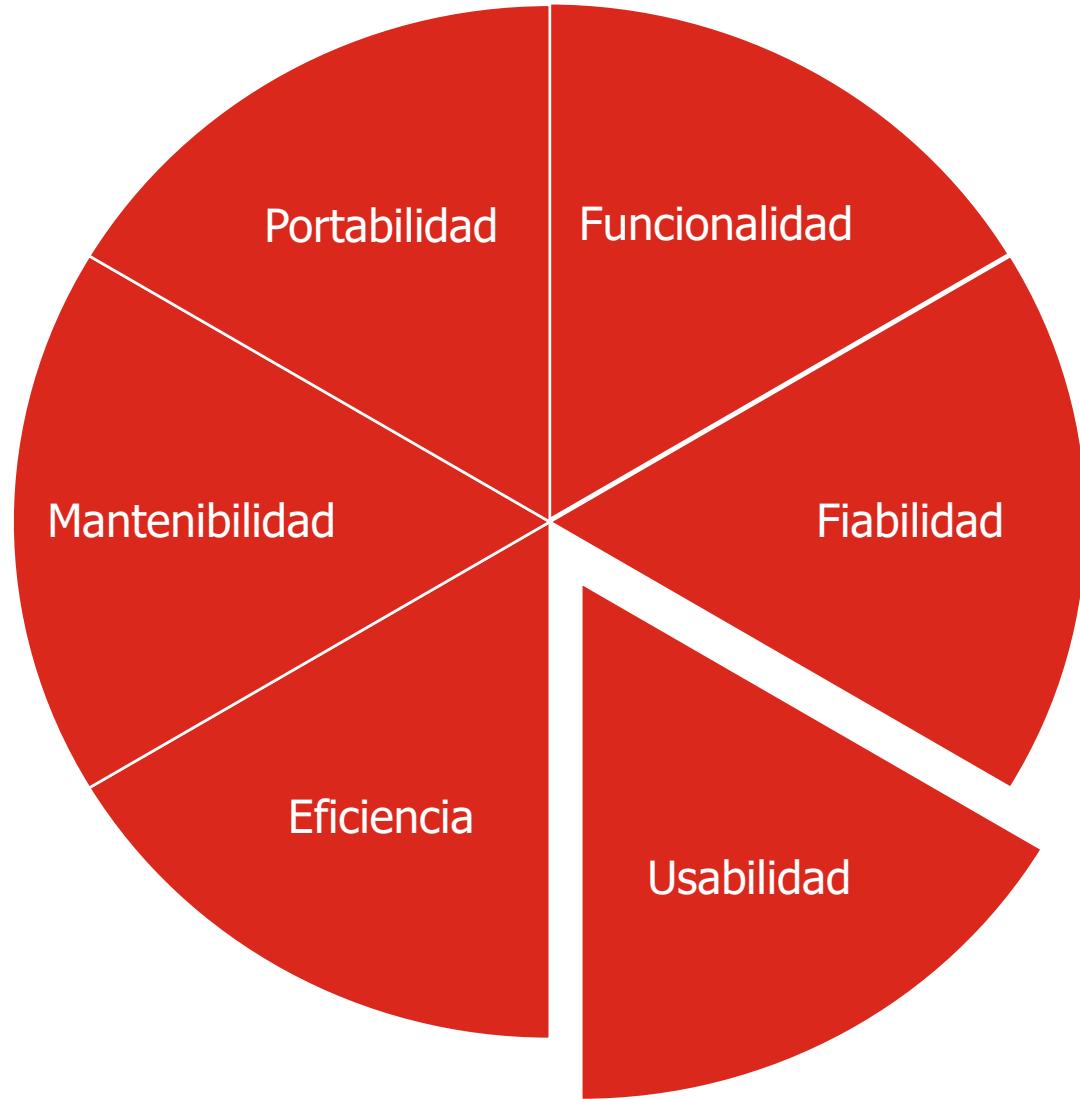
# Usabilidad web

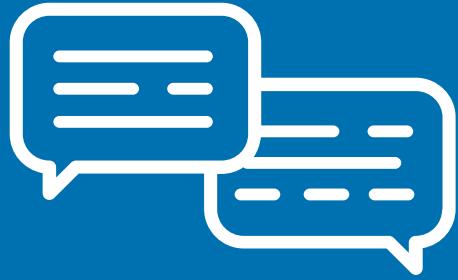
## Estándares internacionales



International Organization for Standardization (ISO)

ISO/IEC 9126 Software engineering product quality





# DEBATE

---

¿Usabilidad = Accesibilidad?

## OBJETIVO

Tener presente los principales factores que contribuyen a mejorar la usabilidad web y qué relación tiene con la accesibilidad.

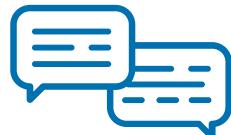
## INSTRUCCIONES

1. Ver el video “Usabilidad web: ¿Qué es y qué aporta al usuario?” (4:57 min de duración) que hace referencia a la comodidad con la que un usuario navega por la web y los diferentes aspectos a tener en cuenta.

<https://www.youtube.com/watch?v=Y7nLeu4cY38>



**30 min**



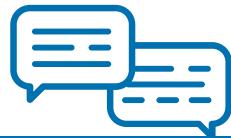
## **REFLEXIONES**

Tras ver el video de Accesibilidad web y este último sobre la Usabilidad web, debate en grupo las siguientes cuestiones y otras que consideres oportunas y/o interesantes:

1. ¿Tienen algún tipo de relación la usabilidad y la accesibilidad web?
2. ¿Qué diferencias hay entre accesibilidad y usabilidad?
3. ¿Conoces y/o has navegado alguna página que cumpla con la mayoría de los principios vistos en el video?
4. Por parejas, buscar una web de ejemplo que consideréis que cumple con las cuestiones más importantes sobre usabilidad y accesibilidad.
5. Comparte dicha web con todo el grupo.



**30 min**



# Bibliografía y webgrafía



- ECMA International

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

- World Wide Web Consortium (W3C)

<https://www.w3.org/>

- Principios de Accesibilidad (WAI)

<https://www.w3.org/WAI/fundamentals/accessibility-principles/es>

- Web Content Accessibility Guidelines (WCAG)

<https://www.w3.org/WAI/standards-guidelines/wcag/>

# Bibliografía y webgrafía



- Principios de accesibilidad web (WebAIM)

<https://webaim.org/resources/quickref/>

- International Organization for Standardization (ISO)

<https://www.iso.org/home.html>

- w3schools.com

<http://www.w3schools.com>

# BOOSTING YOUR DIGITAL SKILLS

[bit.es](http://bit.es)

## Barcelona

C. dels Almogàvers 123  
08018 Barcelona  
Tel. +34 933 041 720  
Fax. +34 933 041 722

## Madrid

Plaza Carlos Trías Bertrán,  
7 Edificio Sollube, 1<sup>a</sup> Planta  
28020 Madrid  
Tel. +34 914 427 703  
Fax +34 914 427 707