



A thin horizontal bar consisting of two segments: a teal segment on the left and an orange segment on the right.

# GIT

# Introducción al desarrollo web

Lic. Faure Analia

Sistema de control de versiones

Sistema distribuido

Velocidad de actualización, es casi instantánea

Diseño sencillo

Gran soporte para desarrollo



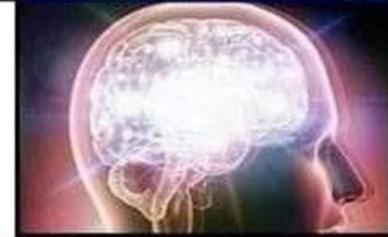
Name

- v1.0
- v2.0
- v2.1
- v2.2



Name

- project
- project-revised
- project-final
- project-final-for-real



Name

- project
- projecttt
- projectttttttt
- .....



- 
- 1. Instalación
  - 2. Configuración
  - 3. Primera parte
  - 4. Segunda parte
  - 5. Tercera parte

## Instalación

<https://git-scm.com/>

Descargar la versión del sistema operativo que tengamos.

Verificar el número de versión de Git que hemos instalado: **git --version**

```
PS C:\Users\admin> git --version
git version 2.43.0.windows.1
PS C:\Users\admin>
```

---

## Configuración de datos iniciales de Git

Establecemos el nombre de usuario y dirección de correo electrónico que luego cada vez que confirmamos los cambios en el repositorio de Git se usarán esta información que se introduce.

Debemos ejecutar los siguientes comandos:

**git config --global user.name [ nombre ]**

**git config --global user.email [ correo electronico ]**

Ver datos de configuración:

**git config --list**

---

## Primera parte.

`git init`

`git clone`

`git pull`

`git checkout -b rama-nueva`

`git remote add origin https://gitlab.com/usuario/nombre-del-repositorio.git`

---

## Segunda parte.

**git status (ver como esta mi ámbito de trabajo)**

**git pull origin main**

**git add ./\* (el punto agrega todos los archivos, se puede especificar qué archivo subir)**

**git commit -m “comentario”**

**git push origin rama-nueva**



## Tercera parte.

Fusionar las ramas:

Dentro de git crear un pull request de la rama-nueva contra la rama main

---

¿ Preguntas ?

Gracias por su atención!



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

**Unidad 4 – Diseño Web Adaptativo y Frameworks CSS**  
RWD y Bootstrap

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

# Objetivos de la clase

- **Objetivos**

- Diseñar documentos Web optimizados para diversos dispositivos.
- Utilizar Frameworks CSS

- **Temas a desarrollar:**

- Diseño Web Responsivo. Media Queries
- Frameworks CSS. Instalación. Clases. Componentes.

# Responsive Web Design (Diseño adaptativo)

- Los usuarios de hoy día esperan poder acceder a los sitios web no solo en sus PCs de escritorio o laptops sino también desde sus tablets o teléfonos móviles.
- Los navegadores de los dispositivos móviles soportan todas las características que hemos examinado, sin embargo, no siempre es fácil estructurar **documentos HTML** y definir **estilos CSS** que provean una experiencia óptima en todos los dispositivos por las diferencias obvias que existen en el tamaño de las pantallas.
- *Las técnicas y tecnologías que se utilizan para crear sitios realmente multidispositivos se conocen con el nombre de **Responsive Web Design** por sus siglas **RWD**. En español Diseño adaptativo.*
  - **RWD** alienta a los diseñadores a crear un único conjunto de recursos para todos los dispositivos, más que crear sitios web especializados para los diferentes dispositivos.
  - Esto se vuelve cada vez más importante ya que las dimensiones de las pantallas difieren incluso en dispositivos del mismo tipo.

# Responsive Web Design (Diseño adaptativo)

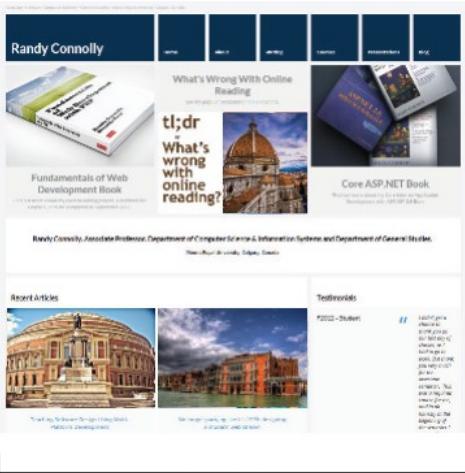
- Hay cuatro componentes clave que hacen que nuestros diseños sean responsivos:
  - Crear un diseño que utilice **Flexbox** o **Grid**.
  - Establecer **viewports** a través del tag **<meta>**.
  - Definir **reglas CSS** específicas para diferentes resoluciones utilizando **Media Queries**.
  - **Escalar imágenes** al tamaño de la pantalla.
- Si bien no es necesario en todos los casos, podemos utilizar también:
  - Utilizar **tipografías** que se escalen adecuadamente en distintos tamaños de pantalla.

# Viewports

- *El **viewport** de un navegador es la parte de la ventana del navegador donde se muestra el contenido web.*
- Los navegadores móviles **reducen la escala** de la página web de forma que coincide con el ancho de la pantalla.
  - Esto es así desde que existen los primeros smartphones dado que muchos sitios no tenían una versión mobile.
- Para solucionar de una mejor forma este problema el **navegador móvil Safari** introdujo el tag **meta name="viewport"** como una manera para que los programadores podamos controlar **el tamaño inicial del viewport**.

# Viewports (2)

- 1 Mobile browser renders web page on its viewport



960px  
Mobile browser viewport

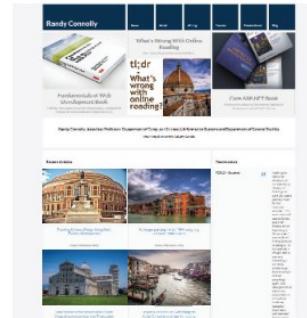
- 2 It then scales the viewport to fit within its actual physical screen



320px  
Mobile browser screen

```
<meta name="viewport"  
      content="width=device-width, initial-scale=1" />
```

- 1 Mobile browser renders web page on its viewport and because of the <meta> setting, makes the viewport the same size as the pixel size of screen.



320px  
Mobile browser viewport

- 2 It then displays it on its physical screen with no scaling.



# Viewports (3)

```
<html ...>  
  <head>  
    ...  
    <meta name="viewport" content="width=device-width, initial-scale=1"/>
```

- En el código el atributo **width** controla el tamaño del viewport mientras que **initial-scale** establece el nivel de zoom.
  - El elemento **meta** le dice al navegador que no se necesita un escalado adicional para hacer el **viewport** de tantos píxeles como tiene el ancho del dispositivo.
  - Si el dispositivo tiene **320px** de ancho, entonces el ancho del **viewport** será de **320px**, si la pantalla es de **480px** (por ejemplo si tenemos la pantalla apaisada) entonces el **viewport** será de **480px**.
  - Si no se especificada un **viewport** alternativo a través del elemento **meta**, entonces el navegador móvil no intentará escalar la versión de escritorio del sitio.

# Media Queries

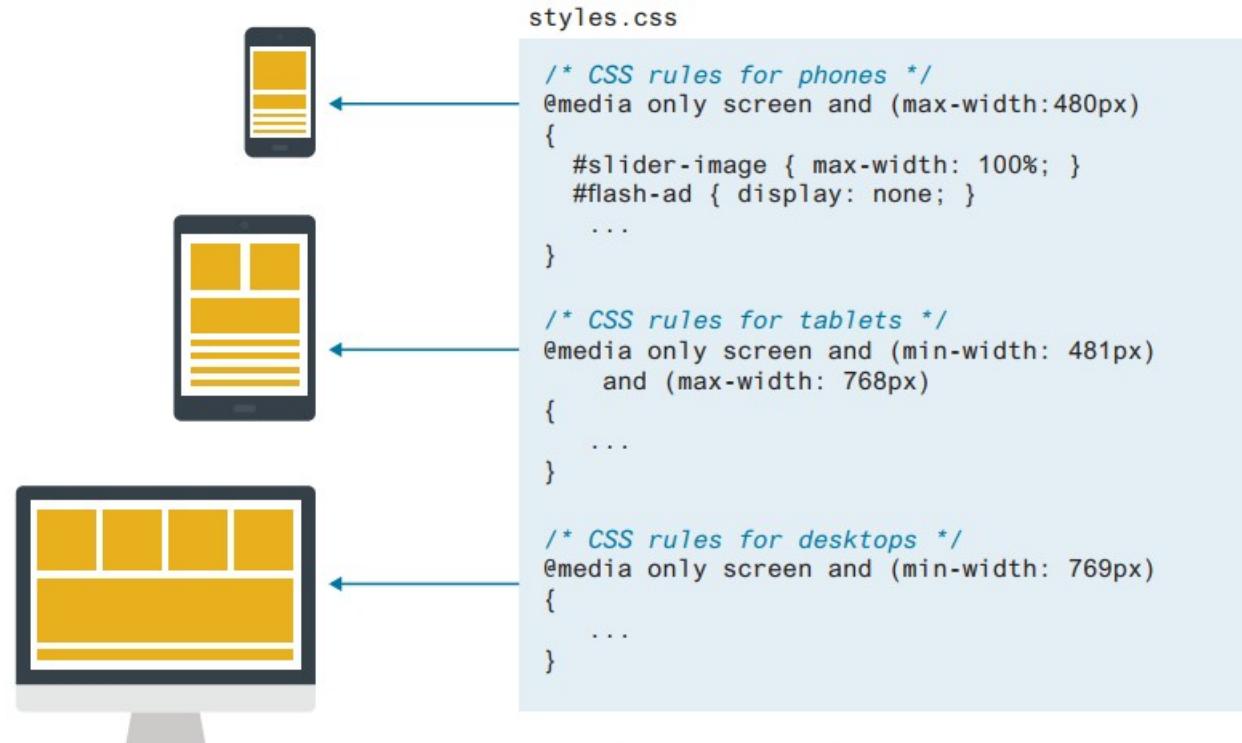
- Una **media query** es una forma de aplicar reglas de estilo según el medio en el que se mostrará la página web.
- Pueden usarse **media queries** para determinar las capacidades del dispositivo y definir **reglas CSS** específicas para ese dispositivo.
- Las **queries son expresiones booleanas** que pueden ser agregadas en los **archivos CSS**.

```
@media (min-width: 768px) { /* Pantallas medianas */
  body {
    font-size: 18px;
    background-color: #f5f5f5;
  }
}

@media (min-width: 992px) { /* Pantallas grandes */
  body {
    font-size: 20px;
    background-color: #e0e0e0;
  }
}
```

Feature	Description
width	Width of the viewport
height	Height of the viewport
device-width	Width of the device
device-height	Height of the device
orientation	Whether the device is portrait or landscape
color	The number of bits per color

# Media Queries (2)



Instead of having all the rules in a single file, we can put them in separate files and add media queries to `<link>` elements.

# Condicionado de aplicación de hojas de estilo

- En vez de tener todas las reglas en un único archivo es posible ponerlas en diferentes archivos y agregar una media query para condicionar su aplicación.

```
<!-- Estilo general, mobile-first -->
<link rel="stylesheet" href="estilo-base.css">

<!-- Estilo para pantallas mayores a 768px -->
<link rel="stylesheet" href="estilo-tablet.css" media="(min-width: 768px)">

<!-- Estilo para pantallas mayores a 992px -->
<link rel="stylesheet" href="estilo-escritorio.css" media="(min-width: 992px)">
```

- Es sobre todas las cosas una técnica utilizada cuando nuestro sitio no fue diseñado originalmente con un enfoque mobile-first.

# Escalado de imágenes

- Hacer escalado de imágenes es bastante simple. Definimos la siguiente regla:
- `img { max-width: 100% }`
- Sin embargo, ***esto no cambia el tamaño (en bytes) de la imagen a descargar***, simplemente la achica o expande (nunca más allá de su tamaño original) para que se adapte al tamaño del elemento contenedor (o la ventana del navegador si no hay elemento padre).
- Los sitios responsivos más sofisticados sirven imágenes de diferente tamaño de acuerdo a las dimensiones del **viewport**.
  - Este enfoque permite que los usuarios de móviles con pantallas más pequeñas recibirán ***archivos más livianos, mejorando así los tiempos de carga***.
- **HTML 5.1** define el elemento **picture** que es una forma elegante de lograr este objetivo a través de markup.
- Con **picture** podemos especificar múltiples elementos **img**, siendo el browser quien determinará cuál utilizar de acuerdo a las dimensiones del **viewport**.

# Enfoque Mobile-first

- Uno de los enfoques más influyentes para el diseño web a veces se denomina diseño mobile-first.
  - Como su nombre lo indica, este enfoque propone comenzar el diseño de nuestro sitio web desde su **versión móvil** en lugar de una adaptación posterior, como suele ser el caso.
- El fundamento del enfoque mobile-first radica no solo en la audiencia cada vez más grande cuya tecnología principal para acceder a sitios web es un dispositivo más pequeño, como un teléfono o una tableta.
  - Centrarse primero en la plataforma móvil también obliga a los diseñadores y arquitectos del sitio a centrarse en el componente más importante de cualquier sitio, **el contenido**.
- Debido a los tamaños limitados de estos dispositivos, **el contenido clave debe resaltarse** sobre los muchos elementos que a menudo ensucian la página de los sitios diseñados para pantallas más grandes.

# Bootstrap

- Bootstrap fue creado en 2011 por Mark Otto y Jacob Thornton, del equipo de desarrollo de la red social Twitter, con el fin de crear un estándar de desarrollo interno a través de un framework que utilice JavaScript y CSS3.
- **Bootstrap** es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web.
- Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS complementado con extensiones de JavaScript. A diferencia de muchos frameworks Web, se enfoca exclusivamente en el desarrollo del lado del cliente (front-end).
- En agosto de 2011, se publicó la primera versión como código abierto en GitHub, convirtiéndose en febrero de 2012 en el proyecto más popular de desarrollo web.

- Bootstrap 5 es la última versión de Bootstrap.
- Esta versión trae consigo nuevos componentes y mejor soporte para las últimas versiones de navegadores.
- Los cambios más sustanciales incluyen:
  - No se requiere jQuery (sin embargo se necesita Popper.js)
  - Abandonado el soporte de IE 10 y 11 (Bootstrap 4 abandonó el soporte de IE8, IE9, y iOS 6).
  - Los atributos **data-** ahora son **data-bs**.
  - Cambios en **badges**. Por ejemplo: **badge-pill** es ahora **rounded-pill**.
  - Fuentes responsivas por defecto.
  - Nuevo breakpoint **xxl**.
  - Nuevos componentes: accordion, offcanvas, floating labels, placeholders.

## El sistema de rejillas (12 columnas)

- Para hacer que un sitio web se adapte a todo tipo de pantalla, **Bootstrap** utiliza un sistema de 12 columnas (rejillas o grillas) para distribuir y adaptar el contenido de la página web.
- Si diseñamos una página web basada en cuatro columnas, cuando accedemos a ella desde una tablet, el contenido puede reorganizarse en tres columnas, o a una sola si lo hacemos desde un smartphone.
- En **Bootstrap**, ese sistema de columnas lleva un prefijo de clase compuesto por cuatro tamaños diferentes de acuerdo al dispositivo para el cual se esté programando.

# El sistema de rejillas (12 columnas) cont.

- La configuración por defecto es para dispositivos cuyo tamaño de pantalla no supera los **576px** de ancho.
- Si la resolución es mayor o igual a **768px**, el prefijo de clase es **.col-md-**.
- Si la resolución es mayor o igual a **992px**, el prefijo es **.col-lg-**.
- Si la resolución es mayor o igual a **1200px**, el prefijo es **.col-xl-**.
- Si la resolución es mayor o igual a **1400px**, el prefijo es **.col-xxl-**.

Breakpoint	Class infix	Dimensions
X-Small	None	<576px
Small	sm	≥576px
Medium	md	≥768px
Large	lg	≥992px
Extra large	xl	≥1200px
Extra extra large	xxl	≥1400px



# Bibliografía

- **Libro:** Randy Connolly, Ricardo Hoar. ***“Fundamentals of Web Development, Global Edition”***. 3era Edition. Ed. Pearson. 2022.
- **Libro:** Lenny Burdette. ***“The JavaScript Pocket Guide”***. 1st Ed. Peachpit Press. 2010.
- **Libro:** John Pollock. ***“JavaScript: A Beginner’s Guide”***. Ed. McGrall Hill. 2013.



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

**Unidad 1 – Fundamentos de la WWW**

Conceptos y Componentes

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

# Unidad 1 – Fundamentos de la WWW

- **Objetivos**

- Comprender el modelo de funcionamiento de la Web.
- Identificar los componentes involucrados y sus responsabilidades.
- Conocer productos disponibles y sus principales características.

- **Temas a desarrollar:**

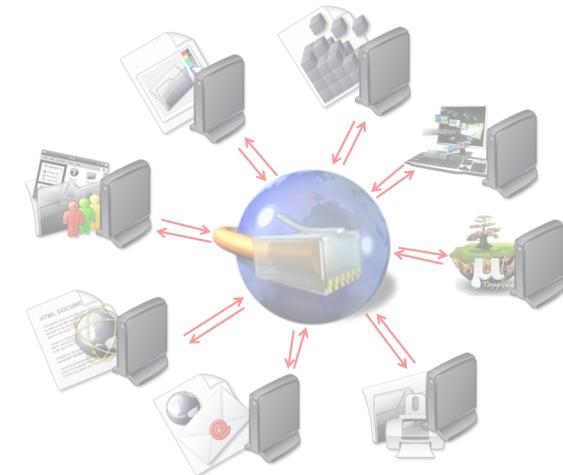
- **Internet y World Wide Web:**
  - Concepto.
  - Evolución.
  - Modelo Cliente/Servidor en la Web.
- **Componentes de la Web:**
  - Navegadores Web, Servidores Web, Servidores de Nombre de Dominio (DNS), URI y URL.
- **Fundamentos del protocolo HTTP:**
  - Patrón de comunicación, concepto de mensajes, métodos.

- Según la Real Academia Española Internet es:

- red informática mundial
- descentralizada,
- formada por la conexión directa entre computadoras
- mediante un protocolo especial de comunicación.

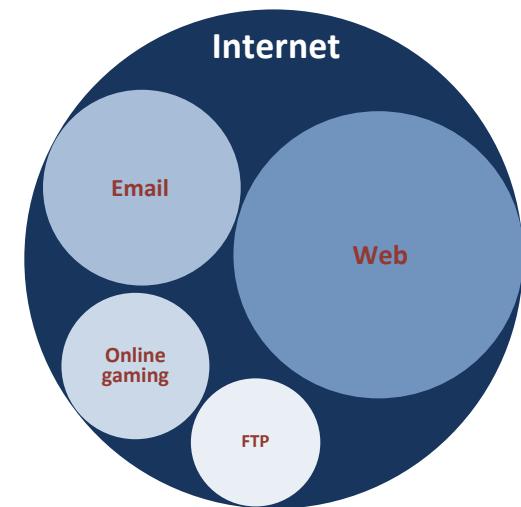
- Si analizamos estos puntos:

- Red informática mundial: prácticamente desde cualquier lugar del mundo puede conectarse a Internet.
- Descentralizada: es un conglomerado de tecnologías que se solapan y refuerzan mutuamente.
- Conexión directa entre computadoras (o dispositivos): a través de diferentes medios. Por ejemplo: cables de fibra óptica, UTP, líneas telefónicas convencionales, redes inalámbricas de computadoras, red celular, comunicaciones satelitales, etc.
- Protocolo especial de comunicación: el protocolo TCP/IP.



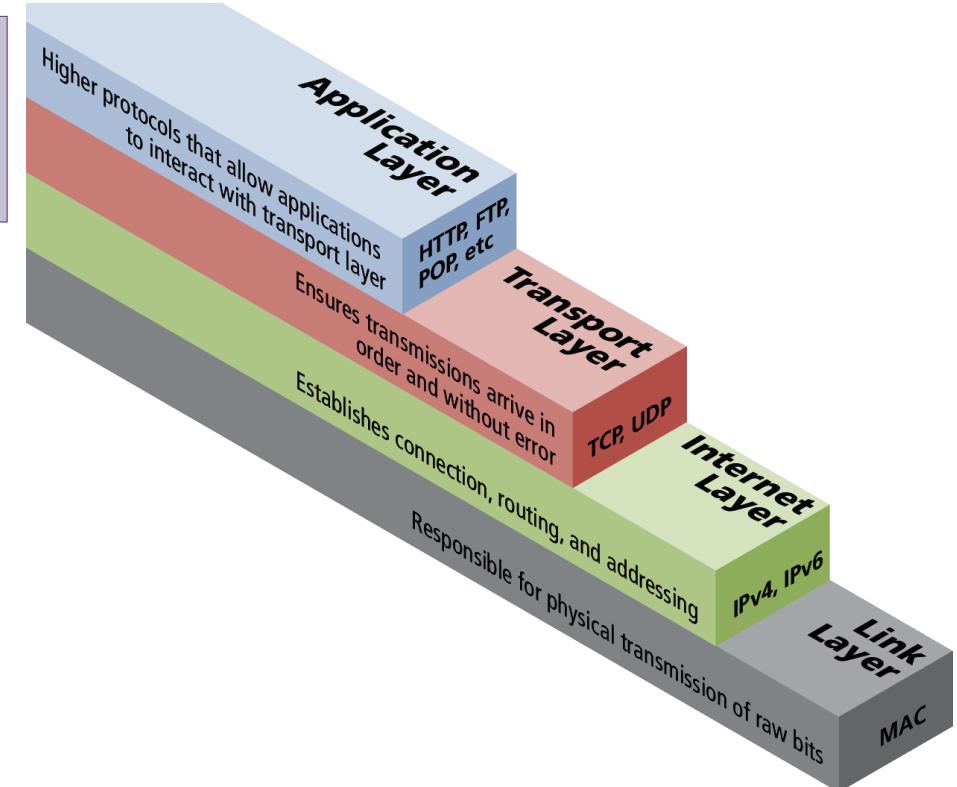
# Internet y la World Wide Web

- **Internet** no es sinónimo de **World Wide Web**:
  - La **WWW** es un servicio de **Internet**.
  - La **Web** es un sistema de distribución de documentos de hipertexto interconectados y accesibles vía **Internet**.
- **Internet** ofrece múltiples servicios:
  - **WWW** (Sitios Web, Buscadores, Blogs, .com).
  - Acceso Remoto (SSH y telnet).
  - Transferencia de Archivos (FTP, SFTP y P2P).
  - Correo electrónico (SMTP).
  - Boletines electrónicos o grupos de noticias (NNTP).
  - Conversaciones en línea (IRC y chats).
  - Telefonía IP (VoIP), Televisión IP (IPTV).
  - Mensajería Instantánea (Google Meet, Skype, Whatsapp, Telegram).
  - Procedimientos remotos (RPC) y Servicios Web (SOAP y RESTful).



# Protocolos de Internet

- **Internet** existe gracias a los protocolos de comunicación.
- *Un protocolo es un conjunto de reglas que deben cumplirse para establecer una comunicación.*
- La pila de protocolos **TCP/IP** es una pila compuesta por cuatro capas de abstracción. Cada capa se comunica con las que se encuentran inmediatamente arriba o abajo sin preocuparse por las más lejanas.
- Las capas de nivel inferior se encargan de los aspectos más fundamentales de la transmisión de señales lo que permite a los niveles superiores encargarse de la interacción entre clientes y servidores.

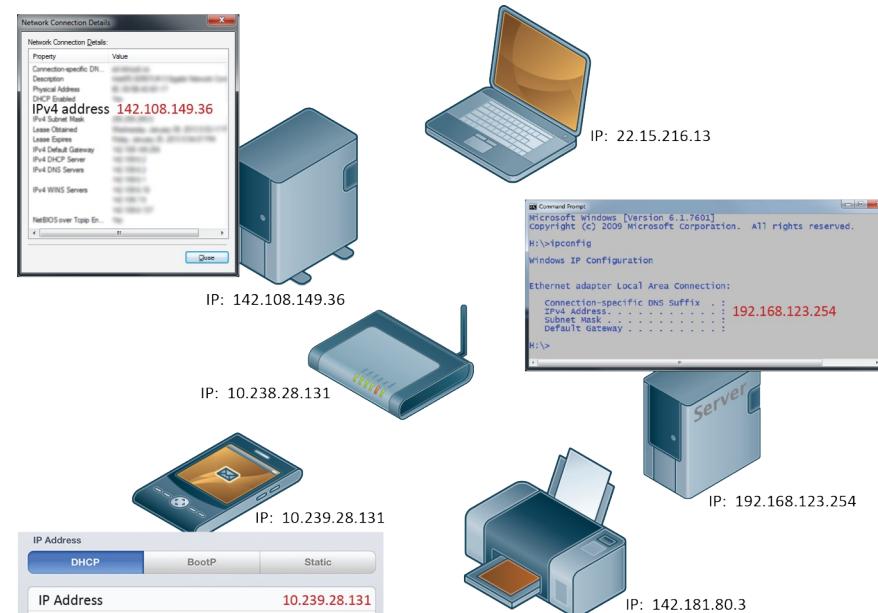


# Protocolos de Internet – Capa de Enlace

- Está a cargo de las transmisiones físicas a través de distintos medios (cableado o wireless) y de establecer enlaces lógicos
- Resuelve problemas como la creación, la transmisión y recepción de paquetes, la detección de errores, colisiones y uso compartido del medio, entre otros.
- Un término de esta capa que a veces se usa en el contexto de Internet es el de las direcciones **MAC (Media Access Control)**:
  - Se utiliza para identificar de manera única cada dispositivo en una red de datos
  - La dirección **MAC** está compuesta por una serie de números y letras, usualmente expresados en un formato hexadecimal.

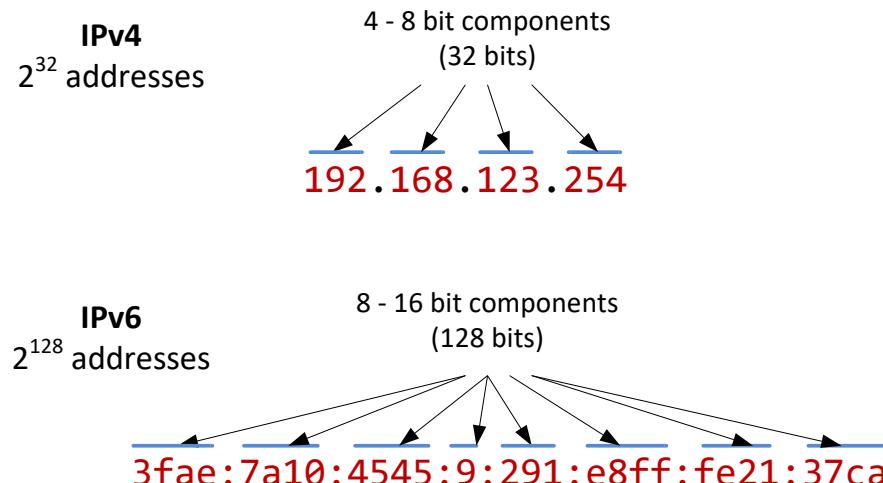
# Protocolos de Internet – Capa de Internet

- También llamada la **Capa IP**. Enruta los paquetes a través de la red para que arriben a destino.
- Provee comunicación del tipo "*el mejor esfuerzo*": envía el mensaje al destino sin esperar respuesta. No garantiza que el mensaje llegue o llegue intacto.
- **Internet** usa las direcciones del **Protocolo IP** para identificar los dispositivos en la red. Todo dispositivo conectado a **Internet** tiene una **Dirección IP** un código numérico que lo identifica únicamente.



# Protocolos de Internet – Direcciones IP

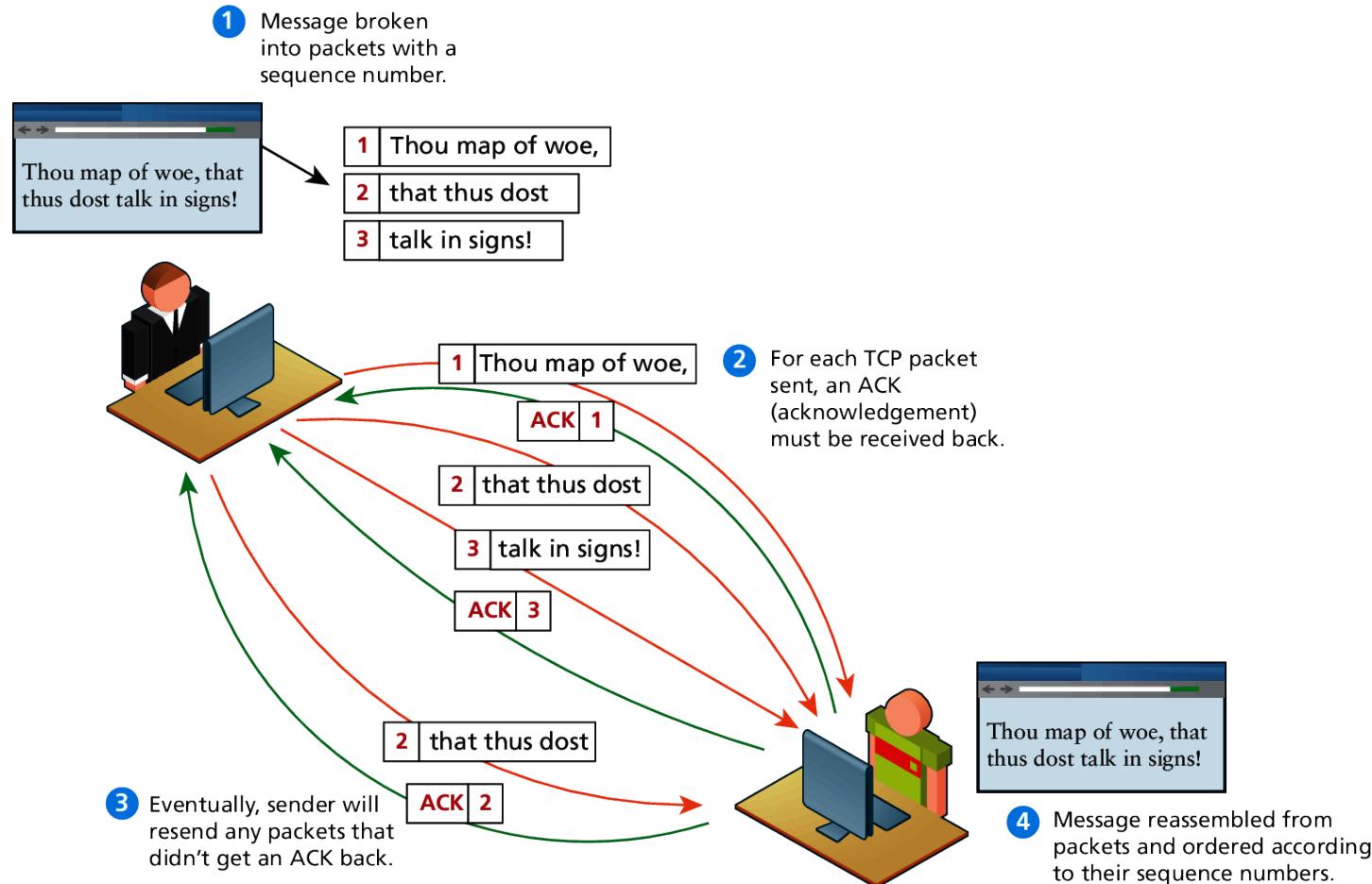
- Las direcciones **IPv4** son direcciones las originales del **Protocolo TCP/IP**.
- En **IPv4**, se usan 12 números (implementados como 4 enteros de 8-bits) que se escriben con un punto entre cada entero.
- Como el mayor valor que podemos representar con un entero no signado de 8-bits es 255, los cuatro enteros juntos pueden codificar aproximadamente 4,2 billones de direcciones IP únicas.
- **IPv6** fue creado para superar este límite, usando 8 enteros de 16-bits.
  - Los números generalmente se escriben en notación hexadecimal separados por dos puntos.



# Protocolos de Internet – Capa de Transporte

- A través de diversos mecanismos, asegura que las transmisiones se lleven a cabo, en orden y sin errores.
- Primero los datos se dividen en paquetes con el formato establecido por el protocolo **Transmission Control Protocol (TCP)**.
- Segundo, cada paquete es confirmado desde el receptor al emisor.
- En caso que se pierda un paquete el transmisor se va a dar cuenta que falta porque no hay un **ACK** (o confirmación) para ese paquete.
- El paquete es retransmitido, a pesar de estar fuera de orden y vuelto a ordenar en el destino.

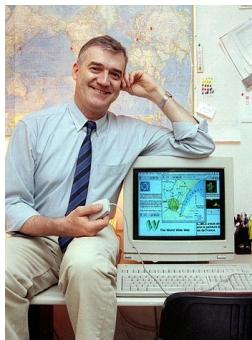
# Protocolos de Internet – Capa de Transporte (2)



# Protocolos de Internet – Capa de Aplicación

- La **capa de Aplicación** es con la cual la mayoría de los desarrolladores Web se encuentra familiarizado.
- Los protocolos de esta capa implementan una comunicación a nivel de procesos y constituyen un mayor grado de abstracción en comparación a los paquetes de bajo nivel, direcciones IP y protocolos de capas inferiores.
- Son ejemplos de esta capa los protocolos: HTTP, SSH, FTP, DNS, POP, SMTP.

# World Wide Web



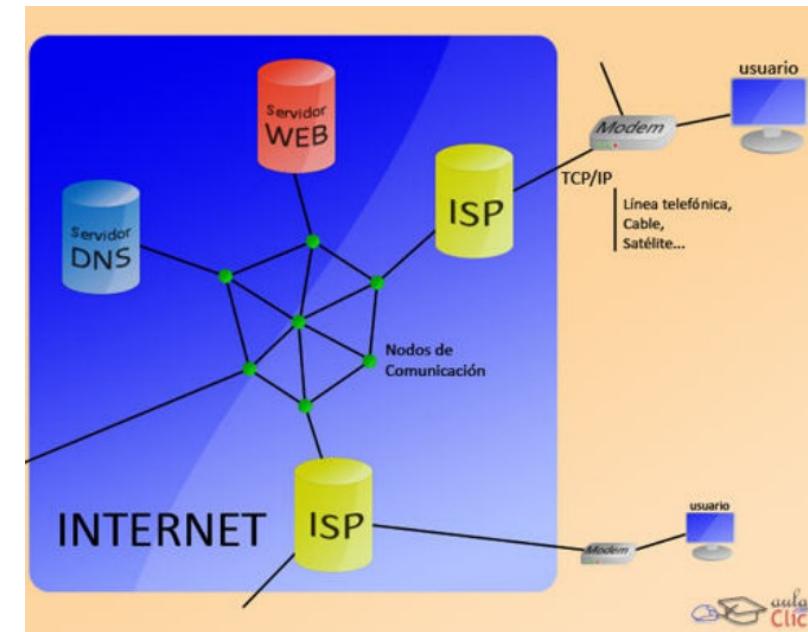
- La Web fue desarrollada por **Sir Tim Berners-Lee** con la ayuda de **Robert Cailliau** a finales de los 80's mientras trabajaban en el **CERN** (Consejo Europeo para la Investigación Nuclear).
- Fue concebida con la idea de “*vincular y acceder a información de diversos tipos como una red de nodos en los que el usuario puede navegar a voluntad*”.
- **Berners-Lee** no solo ideó la **Web**:
  - Desarrolló un esquema de acceso para que las páginas Web puedan ser localizadas, recuperadas y renderizadas (**URL**).
  - Programó el primer **navegador Web con GUI** (bajo el nombre de **WorldWideWeb**).
  - Es el creador del protocolo **HTTP (HyperText Transfer Protocol)** por el cual el contenido de una página Web es enviado desde el servidor Web hasta el navegador.
  - Es el creador del primer **servidor Web (HyperText Transfer Protocol daemon)**.
  - Desarrolló la primer versión de **HTML (HyperText Markup Language)**.

# ¿Qué es la WWW?

- La **World Wide Web** (literalmente “Red de Alcance Mundial”) es una colección de documentos electrónicos que están vinculados entre sí, como una telaraña.
- La **Web**, es una enorme colección de documentos, imágenes, sonidos y programas que se almacenan en los **servidores Web** y se puede acceder desde los **navegadores Web** a través de **Internet**.
- **Filosofía de la Web:**
  - Es una parte de **Internet**, ubicada en su capa superior.
  - Es un **servicio de Internet** que a su vez constituye un soporte para otros servicios.
  - La **Web** está basada en estándares, de código abierto.
  - Los mayores desarrolladores/vendedores de software apoyan su desarrollo.
  - Regulada por la **W3C** y otras organizaciones internacionales.
  - Es un medio de comunicación, de computación y plataforma de negocios.

# Componentes de la Web

- Computadoras y redes.
- Routers (Enrutadores o commutadores) para interconectar.
- Proveedores de Internet (ISP).
- Servidores de Nombres de Dominio (DNS)
- Protocolos (TCP/IP, HTTP, URI/URL).
- Programas Clientes: navegadores Web o Web browsers.
- Sitios Web.
- Servidores Web (servidores de Hosting): para alojar sitios Web.



# ¿Cómo funciona la Web?

- Las **páginas Web** están almacenadas en **servidores Web** situados por todo el mundo.
- Al introducir en el navegador la **URL** de una **página Web**, o al hacer click en un **vínculo** (texto o imagen), una petición es enviada al servidor que alberga la página.
- El servidor envía la **página Web** nuestra computadora y el **navegador Web** la muestra en su pantalla.



# Páginas Web

- Una **página Web** es un documento electrónico escrito en **HTML: HyperText Markup Language**.
- **HTML** es un lenguaje para construir documentos de **hipertexto** que se basa en el demarcado de elementos o entidades dentro del documento.
  - **Hipertexto**: se trata de texto con referencias (hiperenlaces) a través de los cuales el lector puede acceder inmediatamente a otros textos.
- Las **páginas Web** pueden contener texto, gráficos, video, animaciones, y sonido, así como elementos interactivos, como formularios de entrada de datos.
- Cada página tiene una dirección única o localizador único de recurso o **URL (Uniform Resource Locator)**.
- Las **páginas Web** contienen **hipervínculos** que son textos e imágenes que hacen referencia a **URLs** de otras **páginas Web**.
- Se estima que solo el 20% de las páginas Web son de acceso libre vía Internet.



# Sitios Web

- Un **sitio Web** se compone de una o más **páginas Web** referidas a un tema determinado, como a una persona, un negocio, una organización o a una temática, tal como el deporte.
  - Un sitio Web no puede existir sin una manera de vincular las **páginas Web** que lo componen.
  - Estas páginas deben ser vinculadas con algún tipo de lógica que permita el acceso rápido e intuitivo a cualquiera de las demás desde cualquier locación dentro del sitio.
- La primera página de un **sitio Web** se llama la **página de inicio (homepage) o portada**, y hace las funciones de presentación del sitio.
- Un **sitio Web** puede tener contenido estático o dinámico.
- En la actualidad, la mayoría de los sitios generan sus documentos dinámicamente según la petición del usuario.
- Un **sitio Web** conectado a otros sistemas para acceder a datos en tiempo real es una **Aplicación Web**.

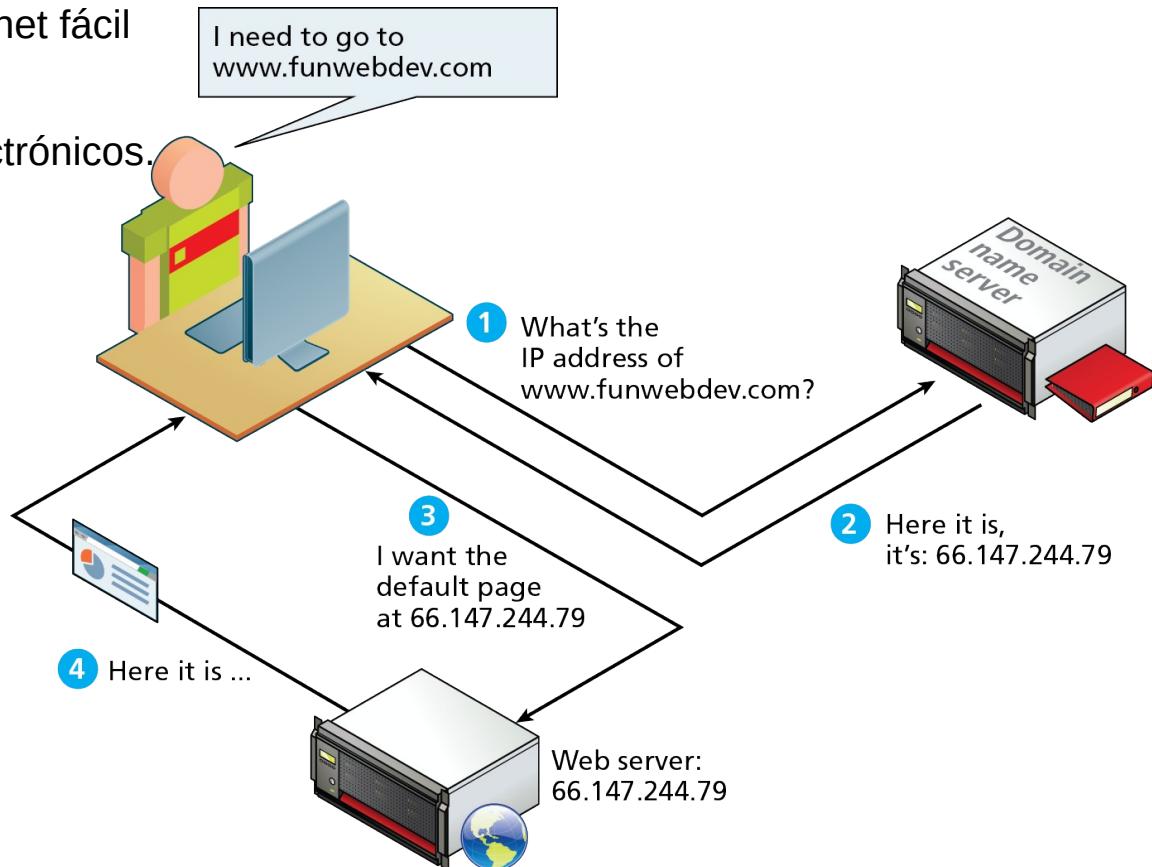


Existen 5 tipos básicos de sitios en la Web:

- Personal.
- Informativo.
- Organizacional.
- Político.
- Comercial.

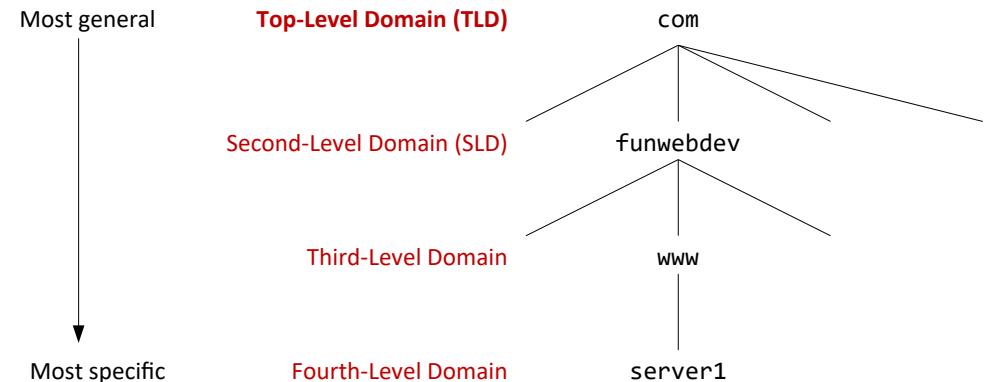
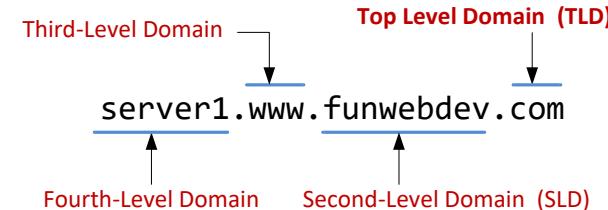
# DNS – Sistema de Nombres de Dominio

- Es más fácil recordar nombres que números. Por eso usamos el **Domain Name System (DNS)** o Sistema de Nombre de Dominio.
- Es escalable, distribuido y hace Internet fácil de usar.
- También se usa para los correos electrónicos.

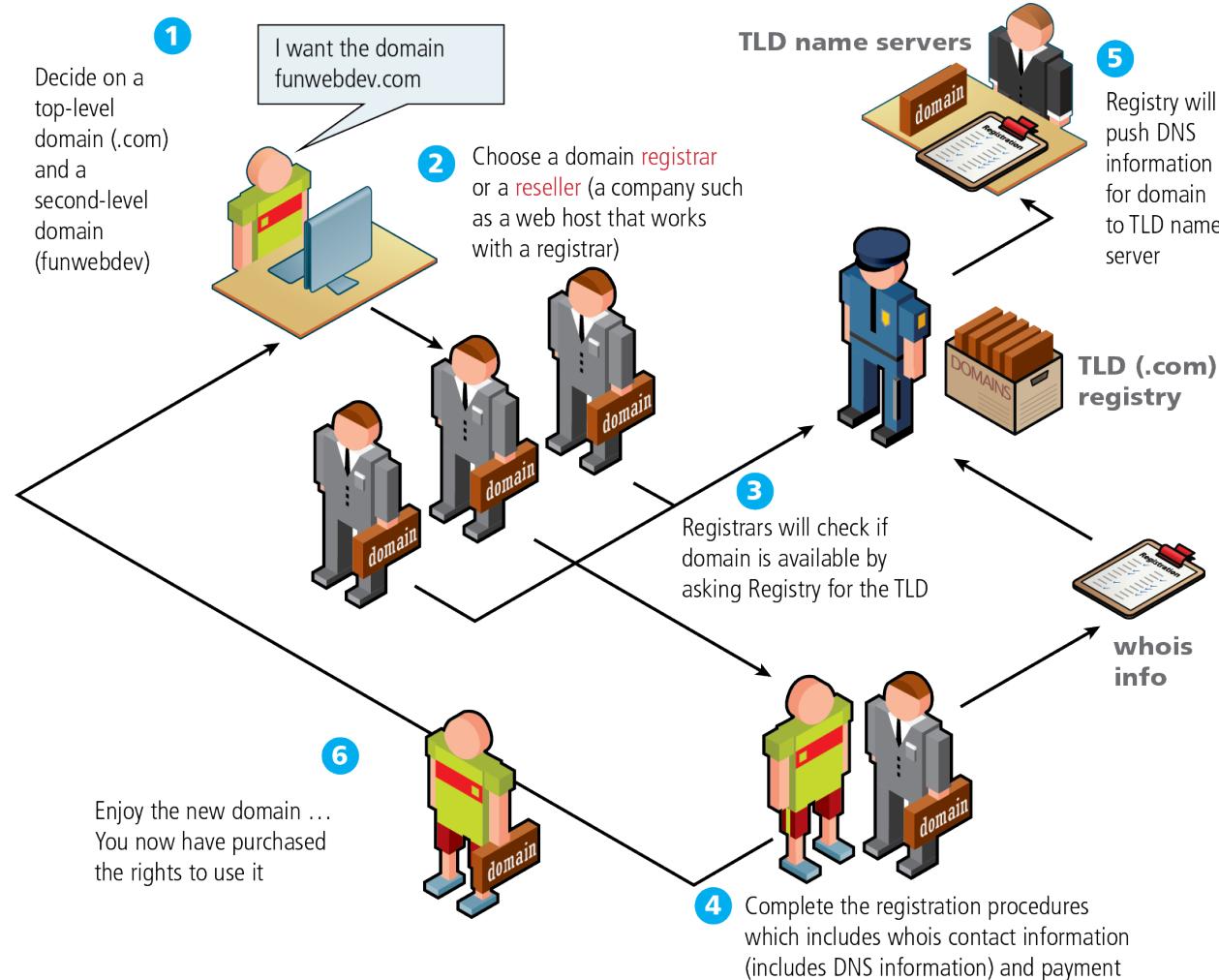


# DNS – Niveles de Nombres

- Un nombre de dominio puede dividirse en varias partes. Éstas representan una herencia.
- Las partes ubicadas más a la derecha son las más cercanas a la raíz de la **jerarquía de nombres de Internet**.
- Todos los sitios tienen al menos dos niveles de nombres: **TLD (Top Level Domain)** y **Second Level Domain**.



# DNS – Proceso de Registro de Nombres

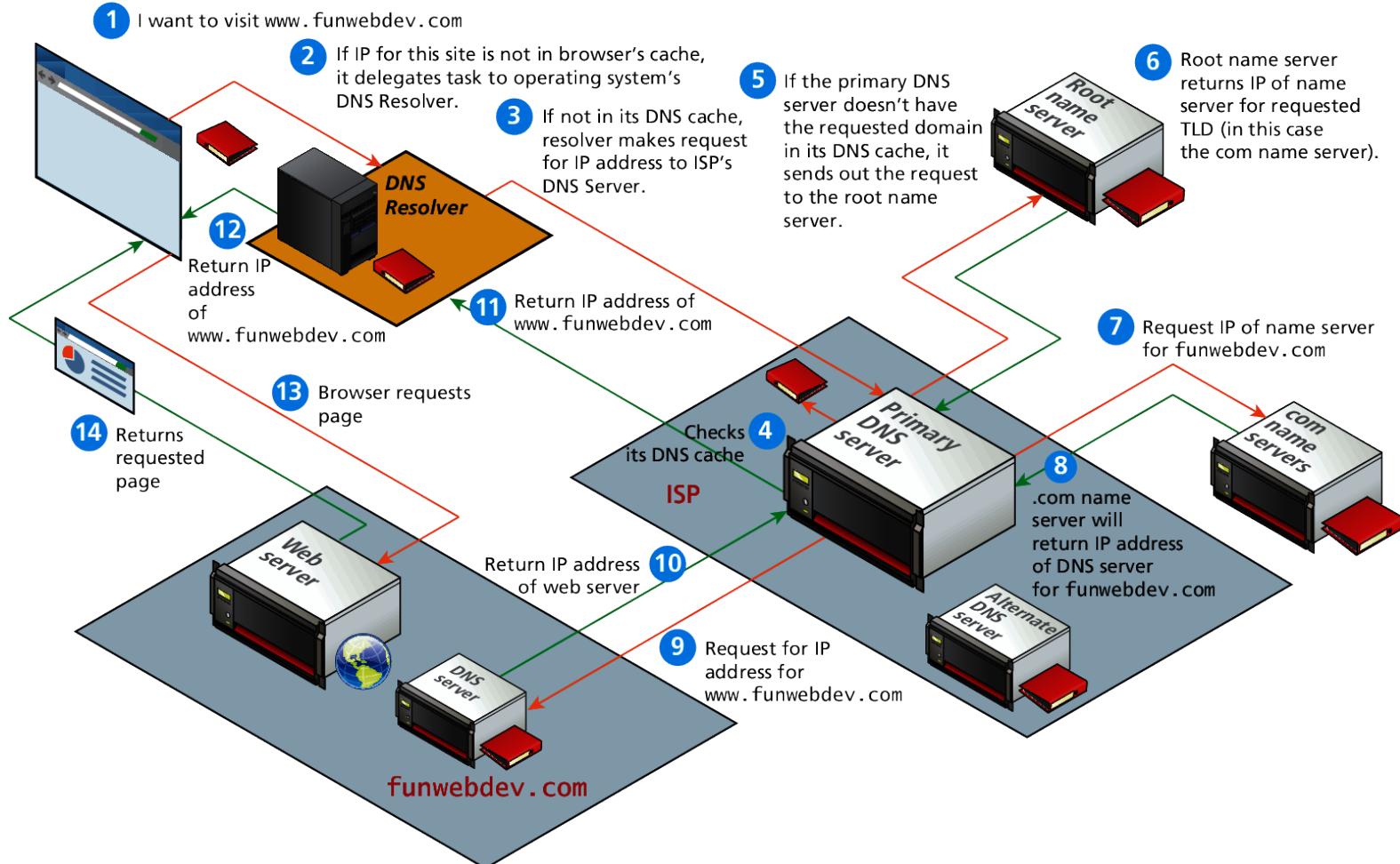


- NIC.AR significa Network Information Center Argentina.
- Administra el Registro de nombres de dominio y asegura el funcionamiento del DNS (Sistema de Nombres de Dominio) para el Country Code Top Level Domain o el Dominio de Nivel Superior Geográfico (ccTLD) '**.ar**'.
- En 1999, la Secretaría de Comunicaciones asignó al Correo Oficial de la República Argentina la administración, altas y bajas del dominio de Internet Argentina.
- En 2011 NIC Argentina, que hasta ese momento operaba dentro de la órbita de la Cancillería, inició un proceso de renovación. Se creó la Dirección Nacional del Registro de Dominios de Internet, dentro de la Secretaría Legal y Técnica de la Presidencia de la Nación.
- El 5 de marzo de 2014, NIC Argentina incorporó el cobro por ciertos trámites. Argentina era, hasta ese momento, uno de los pocos países que todavía mantenía el servicio en forma gratuita. Actualmente, las operaciones aranceladas son el registro, la renovación, la transferencia y la disputa de dominios.



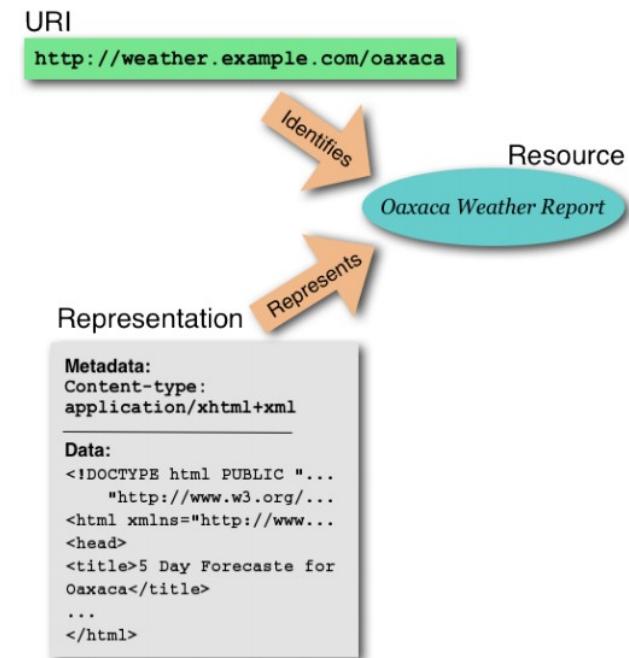
<https://nic.ar>

# DNS – Resolución de Direcciones



# URIs y URLs

- Cada documento que se intercambia según el protocolo **HTTP**, es identificado mediante un **Identificador Uniforme de Recurso URI (Uniform Resource Identifier)**. Un **URI** es un identificador que consiste en una secuencia de caracteres que sigue una sintaxis determinada.
- Los **URIs** están caracterizados por:
  - **Uniformidad**: Permite que diferentes tipos de recursos sean usados en el mismo contexto.
  - **Recurso**: no existe límite para definir que es un recurso. Un "recurso" es aquello que puede ser identificado por un URI.
  - **Identificador**: un identificador encierra la información requerida para distinguir un determinado recurso.
- Los URIs son **identificadores**, es decir, **nombres**.
- Identifican **recursos** y, generalmente (como los URLs) permiten el acceso a las **representaciones** de esos recursos.



# Fundamentos de los URIs

- Los **URIs** tienen un alcance global y son interpretados de manera consistente independiente del contexto.
- Sin embargo el resultado de la interpretación puede estar en relación con el contexto del usuario. Por ejemplo `http://localhost/` tiene la misma interpretación para cada usuario, sin embargo la máquina de referencia `localhost` es diferente.
- Esto muestra que los **URIs** permiten la independencia entre interpretación y acceso.
- Un **URI** puede ser clasificado como un **localizador (URL)**, o un **nombre (URN)**, o ambos.

# Definición de URL

- El término **Localizador Uniforme de Recurso URL (Uniform Resource Locator)** se refiere a un conjunto de **URIs** que, además de identificar un nombre, también provee medios para localizar el recurso describiendo su mecanismo de acceso, por ejemplo su dirección de red.
- Por lo tanto, podemos inferir que los **URLs** son tipos particulares de **URIs**.

# Anatomía de una URL

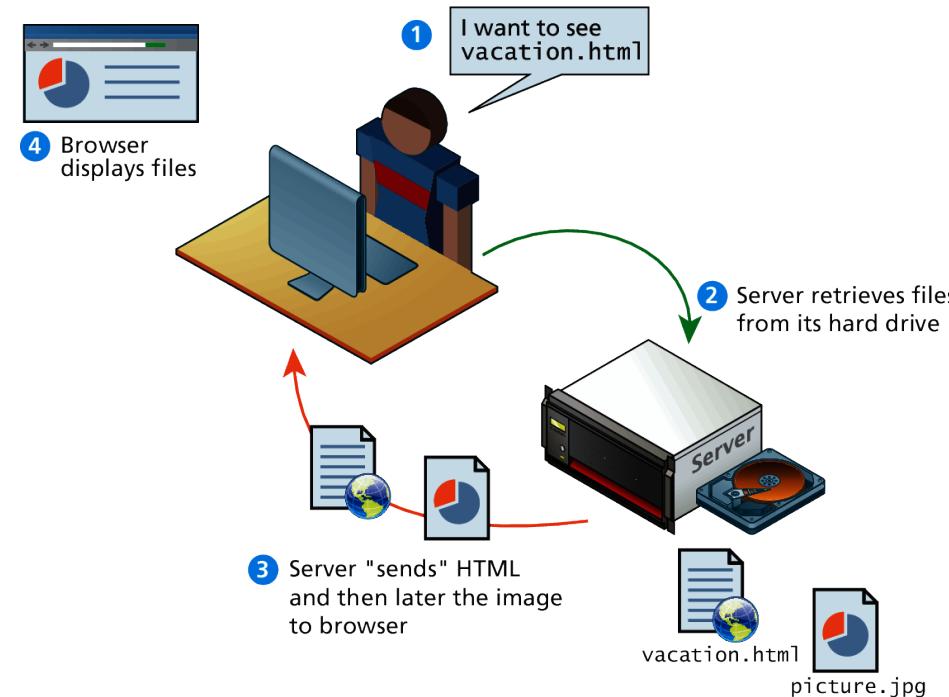
- Los **URL**, deben seguir una sintaxis general, esto les permite representar la dirección de un recurso, independientemente de la forma original de los componentes de la dirección.
- Una **URL** tiene la siguiente sintaxis

**esquema://autoridad/ruta?consulta#fragmento**

- El **esquema/protocolo** es el nombre del lenguaje utilizado para transportar los datos a través de la red desde la dirección de origen a la de destino.
  - La **autoridad** consiste usualmente en el **nombre** o **Dirección IP** de un servidor o (host), seguido a veces de ":" y un número de **Puerto TCP**. Puede incluir **nombre de usuario** y **password**.
  - La **ruta** es la especificación de una ubicación en alguna estructura jerárquica, usando "/" como delimitador entre componentes.
  - La **consulta** habitualmente indica parámetros de una consulta dinámica a alguna base de datos o proceso residente en el servidor.
  - El **fragmento** identifica a una porción de un recurso, habitualmente una ubicación en un documento.
- Ejemplo: <http://www.fcad.uner.edu.ar>

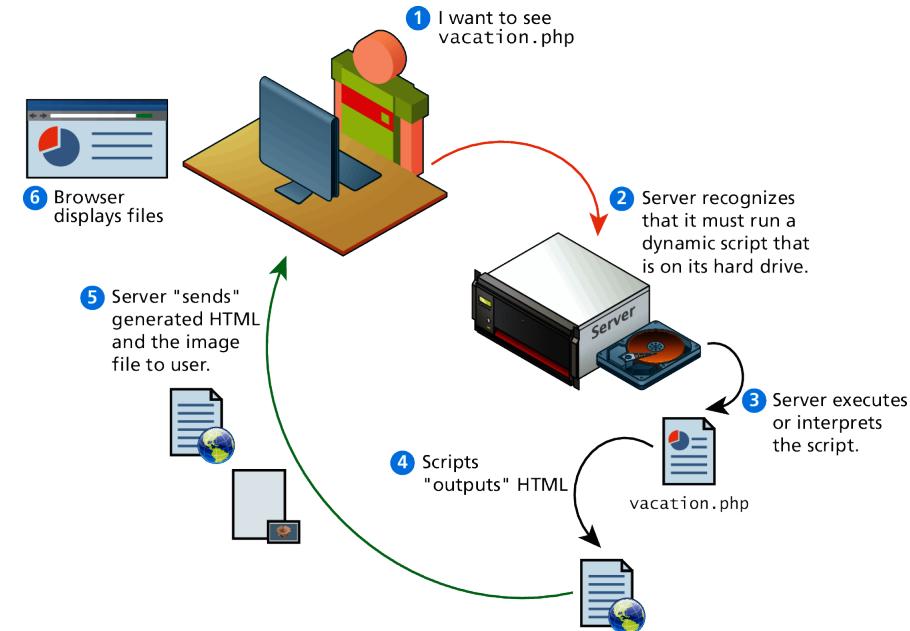
# WWW – Las distintas eras

- La **Web 1.0** es considerada la primera etapa de la Web, abarca la década de los 90s. Es una Web de una sola vía donde el usuario es espectador del contenido publicado por los administradores del sitio.
- El **webmaster** era el encargado de crear, actualizar y mantener el sitio Web.
- Se trataban de **sitios estáticos**: siempre iguales para todos los usuarios.



# WWW – Las distintas eras (cont.)

- Pasados algunos años, los sitios comenzaron a volverse más complicados y se empezaron a usar programas que corrían en los servidores Web para generar el contenido dinámicamente.
- Los **programas del lado del servidor** se encargaban de leer datos desde la bases de datos o de algún sistema y producir una **salida HTML** que era enviada al navegador del usuario.
- Los llamaremos **sitios dinámicos** porque el contenido de la página es creado en tiempo de ejecución por un programa y el contenido varía de usuario a usuario.
- A mediados de los 2000s se puso de moda el término **Web 2.0**, con significados diferentes para **usuarios** y **desarrolladores**:
  - Para **usuarios**: se refería a una experiencia interactiva donde podían contribuir y consumir contenido
  - Para **desarrolladores**: implicaba un cambio de paradigma. La lógica de programa que solo existía en el servidor, ahora migraba al navegador. Esto exigió aprender **JavaScript**, y **comunicación asíncrona**.



- La **Web 3.0** (conocida también como **Web3**) es la tercera generación de servicios de Internet que se centrará en la comprensión y análisis de los datos para proporcionar una web semántica.
- Hoy día, no existe una definición sólida de la **Web 3.0**, pues aún no se ha implementado de forma completa, y se basa es un ideario más que en hechos sólidos que se puedan analizar.
- El objetivo de la **Web 3.0** es el de crear webs más inteligentes, conectadas, abiertas y adaptadas a cada usuario. En este contexto, los usuarios pueden crear contenido mientras lo poseen, controlan y monetizan a través de la implementación de tecnología blockchain y criptomonedas o NFTs. Por ello encontramos una gran cooperación entre la **Web 3.0**, la tecnología de cadena de bloques y el metaverso.

# Bibliografía

- **Web:** Real Academia Española. “**Definición de Internet**”. <http://lema.rae.es/drae/?val=internet>
- **Web:** Learn the Net. [www.learnthenet.com](http://www.learnthenet.com/spanish/web/010www.htm).  
<http://www.learnthenet.com/spanish/web/010www.htm>
- **Libro:** Philip Crowder, David A. Crowder. “**Creating Web Sites Bible**”. 3rd Ed. Wiley Publishing Inc. 2008.
- **Web:** Web Directions. “**World Wide Web Timeline**”.  
<http://www.webdirections.org/history/>
- **Web:** “**Architecture of the World Wide Web**”, Volume One. Ian Jacobs, Norman Walsh  
<http://www.w3.org/TR/webarch/>
- **Libro:** Randy Connolly, Ricardo Hoar: “**Fundamentals of Web Development**”. 3ra Edición. Ed. Pearson. 2022.
- **Web:** NIC.AR. URL: <https://nic.ar/>
- **Web:** “**The Evolution of the Internet Web 3.0 explained**”. Binance Academy. URL:  
<https://academy.binance.com/es/articles/the-evolution-of-the-internet-web-3-0-explained>
- **Web:** Plain Concepts. “**What is Web 3.0**”. URL: <https://www.plainconcepts.com/es/que-es-web-3/>
- **Web:** GitHub. “**How the Web Works**”. URL: <https://github.com/vasanthk/how-web-works>



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

**Unidad 1 – Fundamentos de la WWW**

Orígenes de Internet y la WWW

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

# Unidad 1 – Fundamentos de la WWW

- **Objetivos**

- Comprender el modelo de funcionamiento de la Web.
- Identificar los componentes involucrados y sus responsabilidades.
- Conocer productos disponibles y sus principales características.

- **Temas a desarrollar:**

- **Internet y World Wide Web:**
  - Concepto.
  - Evolución.
  - Modelo Cliente/Servidor en la Web.
- **Componentes de la Web:**
  - Navegadores Web, Servidores Web, Servidores de Nombre de Dominio (DNS), URI y URL.
- **Fundamentos del protocolo HTTP:**
  - Patrón de comunicación, concepto de mensajes, métodos.

# Internet - Orígenes

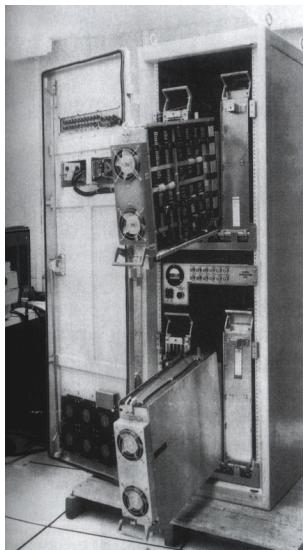
- No es una creación monolítica que surgió en un único lugar en tiempo y locación.
- Es un conglomerado de tecnologías provenientes de las ciencias de la computación, ciencias de almacenamiento y recuperación de datos y comunicaciones que han sido desarrollados (y continúan siendo) a partir de los últimos 60 años.

## ● Intercambio de Paquetes y ARPANET

- Antes las comunicaciones telefónicas se realizaban por intercambio de circuitos.
- Surge el **intercambio de paquetes**, de esta forma una única línea de comunicación puede ser utilizada para soportar múltiples conversaciones al mismo tiempo.
- A mediados de 1960, las redes de intercambio de paquetes permitieron acceder a computadoras remotas en múltiples redes pero solo se podían conectar computadoras con igual arquitectura y sistema operativo.
- En 1964, se crea ARPANET a partir de los trabajos realizados por DARPA (Defense Advanced Research Project Agency) parte del Departamento de Defensa de EE.UU. con el objetivo de investigar formas de conectar en red diferentes sistemas.



# Internet - Orígenes (2)



- En 1969 **ARPANET** consistía en 4 nodos interconectados a través de líneas telefónicas cableadas.
- Los nodos ubicados en el Standford Research Institute (SRI) y las universidades de Los Angeles, Santa Barbara y Utah se conectaban a través de **IMPs (Interface Message Processors)** que fueron los primeros **routers**.
- **IMP** permitió, a través de una interfaz común, comunicaciones entre diferentes máquinas con diferentes sistemas operativos a través de un canal común.
- El protocolo de comunicación era **NCP (Network Control Protocol)**.
- Con el paso del tiempo **NCP** se volvió obsoleto y fue reemplazado en 1983 por **TCP/IP (Transmission Control Protocol/Internet Protocol)** .
- A principios de los 70s **ARPANET** ya estaba formado por 30 instituciones y sobre él se desarrollaron los protocolos **FTP, Telnet, SMTP**.
- Estos desarrollos junto con el crecimiento de las redes y su interconectividad permitieron que **ARPANET** sea utilizado por instituciones gubernamentales, académicas, militares y privadas.

## Internet - Orígenes (3)

- El explosivo crecimiento del número de redes llevó a que se repita el mismo problema de la década anterior. Fue necesario entonces contar con un protocolo que permita conectar a todos los nodos en una única red.
- En 1973, Both Robert Kahn y Vint Cerf desarrollaron **TCP/IP**, protocolo que permitió conectar redes diferentes.
- Al mismo tiempo **Ethernet** fue inventado por Robert Metcalfe y David Boggs.
- **Ethernet** es un estándar para redes LAN que define las características de cableado y señalización de nivel físico y los formatos de tramas de datos del nivel de enlace de datos del **modelo OSI**.
- **Ethernet** es un protocolo flexible que ha sufrido modificaciones sutiles a lo largo del tiempo, incluso habiendo evolucionado de manera significativa los medios físicos que lo usan (cable coaxil, UTP, fibra óptica, routers, switches, etc.).



# Internet - Orígenes (4)

- La década de los 80's fue testigo de un inmenso crecimiento y aceptación de las nuevas tecnologías tanto para empresas como para particulares.
- Las tecnologías subyacentes continuaron evolucionando y mientras se desarrollaba la infraestructura principal de lo que hoy es **Internet**.
- En EE.UU. se funda el **National Science Foundation** para construir **CSNET**, una red que interconectaría universidades y entes gubernamentales.
- En 1981 IBM introducen en el mercado las primeras PC con DOS
  - Se funda Sun Microsystems, se presenta la primer Notebook, el mouse con trackball.
- Debido al crecimiento exponencial de Internet se crea el **Sistema de Nombres de Dominio (DNS)**. Surgen las clasificaciones (.com, .net, .org, .gov y .edu)
- En 1984, para asegurar el acceso de todas las universidades a **Internet** se establecen centros de supercómputo a lo largo de todo EE. UU.
- Para fines de los 80's existían 30.000 redes de las cuales 10.000 estaban conectadas a **Internet** y más de 50 naciones ya se encontraban vinculadas convirtiendo **Internet** en una revolución internacional.

# Historia de la WWW

- En 1941 se publica el libro “*El jardín de senderos que se bifurcan*” de Jorge Luis Borges. Se trata de un cuento corto que muchos entienden como el **pre-concepto del hipertexto**.
- El Dr. Vannevar Bush propone en 1945 la idea de una máquina denominada **Memex**, en el artículo “*As we may think*”.
  - Esta máquina tendría la capacidad de almacenar información gráfica y de texto, de una forma tal que cualquier pieza de información puede ser vinculada con otra(s) pieza(s).
- En 1965 Ted Nelson introdujo el término **hipertexto** e **hipermedia** en un paper para la ACM (Asociación de los Sistemas Informáticos).
- El primer sistema de hipertexto se implementó en 1967 en la Universidad Brown, corriendo en una IBM/360.
- En 1970, Charles Goldfarb, co-inventor del primer lenguaje de marcado **GML** y diseñador de **SGML**, acuña el término "markup language" o "lenguaje de marcas".
- En 1973, es desarrollado **GML (Generalized Markup Language)** es considerado el primer lenguaje de marcas moderno y precursor de SGML, XML y HTML.
- En 1974, Charles Goldfarb comienza a trabajar en **SGML**. El estándar se publica en 1986.

## Historia de la WWW (2)

- En 1990, Berners-Lee desarrolla el primer **navegador Web** bajo el nombre de **WorldWideWeb** (posteriormente renombrado como **Nexus**).
  - El contenido era mostrado en escala de grises
  - Textos e imágenes debían ser mostradas en ventanas diferentes.
- Casi al mismo tiempo **Pei Wei**, (estudiante de la Universidad de Berkley, California) desarrolló **violaWWW** un navegador Web que en muchos aspectos era idéntico al de Berners-Lee agregando características más avanzadas como gráficos en línea, hojas de estilo embebidas, tablas y scripts.
- El primer navegador Web comercial exitoso fue **Mosaic**. Marc Andreessen y Eric Bina del **NCSA** fueron sus creadores. Superaba los demás en muchos sentidos:
  - Estaba centrado en el usuario, por lo que era relativamente sencillo de instalar y usar.
  - Multiplataforma.
  - Soportaba numerosos formatos de imagen y permitía su disposición en línea.
  - Podía renderizar algunos formatos de video y audio.
  - Introdujo algunas características como add-ons: Marcadores e Historial.
- La versión mejorada por Bina de **Mosaic** se conoció como **Netscape** (lanzado en 1994).

## Historia de la WWW (3)

- NCSA cedió los derechos sobre **Mosaic** a Microsoft y otras compañías. Los primeros desarrollos de **Internet Explorer** fueron sobre el núcleo de **Mosaic**. El desarrollo sobre **Mosaic** oficialmente terminó en 1997.
- En 1994, Berners-Lee funda el **World Wide Web Consortium (W3C)** en el **MIT**, con apoyo de **DARPA**.
- La idea central era asegurar la compatibilidad por medio de la definición de estándares denominados **W3C Recommendations**.
- Algunos estándares **W3C**: **CSS, CGI, DOM, HTML, RDF, SVG, SOAP, VRML, XHTML, XML**.

# Bibliografía

- **Web:** Real Academia Española. “**Definición de Internet**”. <http://lema.rae.es/drae/?val=internet>
- **Web:** Learn the Net. [www.learnthenet.com](http://www.learnthenet.com/spanish/web/010www.htm).  
<http://www.learnthenet.com/spanish/web/010www.htm>
- **Libro:** Philip Crowder, David A. Crowder. “**Creating Web Sites Bible**”. 3rd Ed. Wiley Publishing Inc. 2008.
- **Web:** Web Directions. “**World Wide Web Timeline**”.  
<http://www.webdirections.org/history/>
- **Web:** “**Architecture of the World Wide Web**”, Volume One. Ian Jacobs, Norman Walsh  
<http://www.w3.org/TR/webarch/>
- **Libro:** Randy Connolly, Ricardo Hoar. “**Fundamentals of Web Development**”, Global Edition. Pearson. 2015.
- **Web:** NIC.AR. URL: <https://nic.ar/>
- **Web:** “**The Evolution of the Internet Web 3.0 explained**”. Binance Academy. URL:  
<https://academy.binance.com/es/articles/the-evolution-of-the-internet-web-3-0-explained>
- **Web:** Plain Concepts. “**What is Web 3.0**”. URL: <https://www.plainconcepts.com/es/que-es-web-3/>
- **Web:** GitHub. “**How the Web Works**”. URL: <https://github.com/vasanthk/how-web-works>



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

**Unidad 1 – Fundamentos de la WWW**

Modelo Cliente-Servidor y Protocolo HTTP

Tecnicatura Universitaria en Desarrollo Web

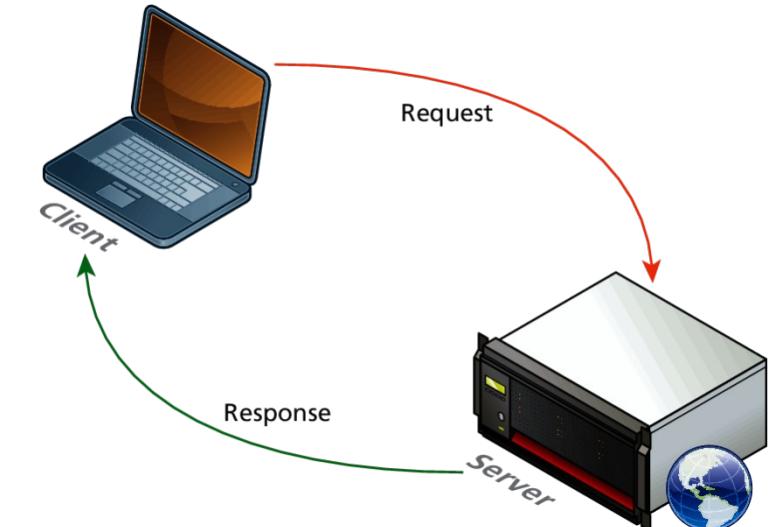
Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

# Unidad 1 – Fundamentos de la WWW

- **Objetivos**
  - Comprender el modelo de funcionamiento de la Web.
  - Identificar los componentes involucrados y sus responsabilidades.
  - Conocer productos disponibles y sus principales características.
- **Temas a desarrollar:**
  - **Internet y World Wide Web:**
    - Concepto.
    - Evolución.
    - Modelo Cliente/Servidor en la Web.
  - **Componentes de la Web:**
    - Navegadores Web, Servidores Web, Servidores de Nombre de Dominio (DNS), URI y URL.
  - **Fundamentos del protocolo HTTP:**
    - Patrón de comunicación, concepto de mensajes, métodos.

# Modelo Cliente - Servidor

- El escenario de trabajo durante todo el curso será el **modelo cliente-servidor**.
- El **modelo cliente-servidor** describe la relación entre:
  - Un **cliente**: proceso corriendo en un sistema que **requiere** un servicio y
  - Un **servidor**: proceso corriendo en un sistema, que **provee** un servicio.
- La interacción (comunicación) ocurre usualmente por medio de una red.
  - Sin embargo, puede encontrarse esta relación en un escenario más acotado.
- En este curso nos centraremos principalmente en el escenario web, por lo que la comunicación será sobre **Internet (TCP/IP)**.



Las computadoras son ilustrativas.  
Un sistema puede alojar más de un servidor o cliente.

# Modelo Cliente – Servidor: Servidores y Clientes

- El **servidor** normalmente está activo las 24 horas del día, todos los días.
- **Espera pasivamente** por los pedidos de los clientes. Al recibir un pedido, realiza las **computaciones necesarias** y **retorna el resultado**.
- En muchas aplicaciones el **servidor** puede atender a varios clientes simultáneamente. Tal es el caso de los **servidores web**.
- El cliente envía un **pedido** y **recibe** respuestas del **servidor** en forma de códigos, texto, imagen y otros formatos. Para que esto suceda debe conocer al servidor, comprender su servicio y forma de comunicación (protocolo).
- La comunicación entre el cliente y el servidor puede presentarse de varias formas:
  - Una aplicación interactúa con un sólo servidor.
  - Una aplicación es cliente de un servicio y luego se vuelve cliente de otro servicio.
  - Un servidor que provee un servicio puede volverse cliente de otro.

## Modelo Cliente – Servidor: Servidores y Clientes (2)

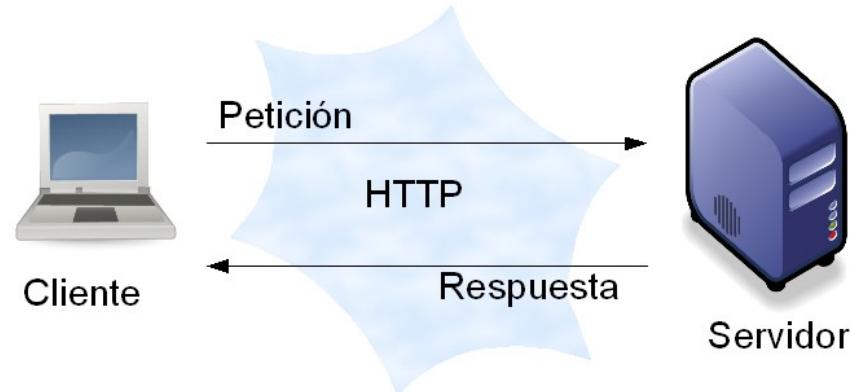
- En la Web, el cliente debe conocer dónde está alojado el programa servidor ¿cómo **identifica** al servidor entre todas las computadoras de Internet?
  - Con la **dirección IP** o el **nombre de dominio** equivalente.
- Siendo que una computadora tiene usualmente un solo canal de comunicación con la red, y que puede ofrecer más de un servicio... ¿cómo se **identifica** cada servicio?
  - Con el número de puerto.
- Por ejemplo:
  - [www.fcad.uner.edu.ar:80](http://www.fcad.uner.edu.ar:80) servidor Web de la facultad
  - [www.fcad.uner.edu.ar:3306](http://www.fcad.uner.edu.ar:3306) servidor de Base de Datos de la facultad.

# Modelo Cliente – Servidor: Servidores

- No es suficiente conocer la ubicación del servidor. Ejemplo: IP + puerto.
  - Es necesario saber la forma de comunicación que debe seguirse: las reglas del diálogo, la estructura de los mensajes, etc.
  - Esto requiere una combinación de protocolos de bajo y alto nivel.
- Algunos aspectos relacionados con el rol del servidor:
  - Autenticación.
  - Autorización.
  - Seguridad de los datos.
  - Privacidad.
  - Conurrencia.
  - Eficiencia.

# El protocolo HTTP (1)

- **HTTP (Hypertext Transfer Protocol)** es el protocolo de red para la Web.
- Establece como se realiza la entrega de archivos de diverso tipo (texto, imágenes, etc.), usualmente denominados recursos que son ubicables por medio del **URL**.
- Se basa en el **modelo cliente-servidor**. Un navegador o browser es un **cliente HTTP** pues envía peticiones al **servidor HTTP** o **servidor web**. El puerto estándar para este servicio es el **80**.
- Como protocolo de comunicación, define el tipo y estructura de los mensajes que se envían y las reglas del diálogo entre los dos participantes.



- **HTTP** es un **protocolo sin estado** es decir que no mantiene información sobre la conexión entre transacciones.

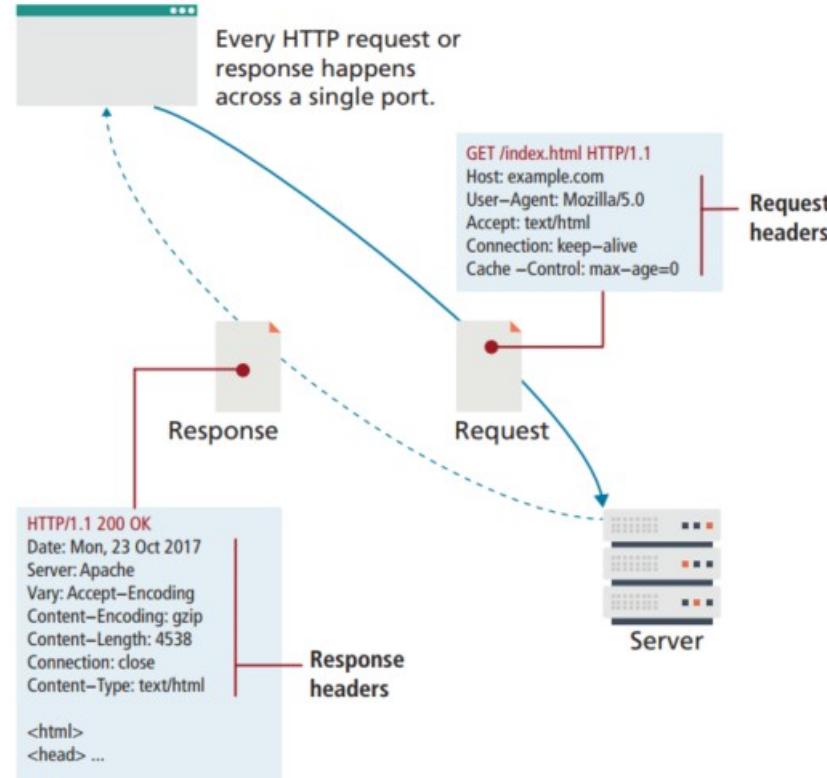
## El protocolo HTTP (2)

- HTTP fue desarrollado por el **World Wide Web Consortium (W3C)** y la **Internet Engineering Task Force (IETF)**.
- Esta colaboración culminó en 1999 con la publicación de una serie de **RFC**, el más importante de ellos es el **RFC 2616** que especifica la versión 1.1 de HTTP.
- Los **mensajes HTTP** intercambiados entre clientes y servidores son en **formato de texto plano**. Es por esto que **HTTP** es un protocolo inseguro y está sujeto a ataques del tipo **man-in-the-middle** y **eavesdropping**.
- Estos ataques pueden permitir a quienes lo perpetran obtener acceso a cuentas de un sitio web e información confidencial.
- **HTTPS (Hypertext Transfer Protocol Secure)** está diseñado para resistir esos ataques y ser más seguro.
  - **HTTPS** es un protocolo de aplicación basado **HTTP**, destinado a la transferencia segura de datos de hipertexto, es decir, es la **versión segura de HTTP**.
  - En un principio utilizado entidades bancarias, comercio en línea. Hoy día por cualquier tipo de servicio que requiera el envío de datos personales y/o contraseñas.

# HTTP versiones

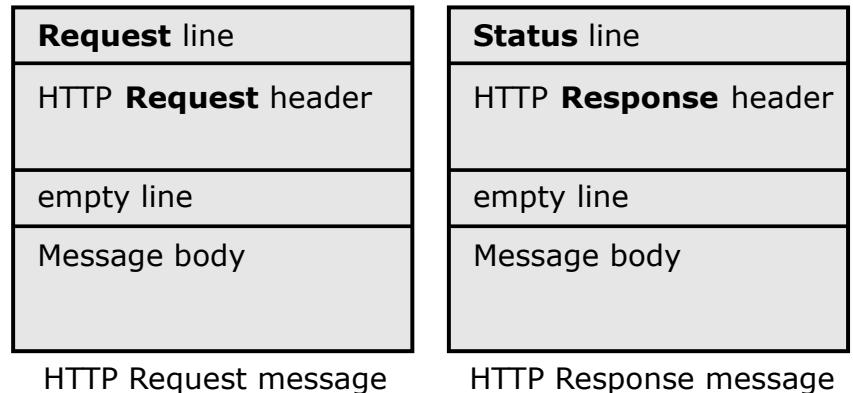
- **HTTP/0.9:** Introducida por **Tim Berners-Lee** en 1991. Era una versión muy básica que solo permitía la transferencia de datos de hipertexto sin encabezados de solicitud o respuesta.
- **HTTP/1.0:** En 1996 introdujo varios nuevos métodos de solicitud, encabezados de solicitud y respuesta, y soporte para distintos tipos de contenido. Sin embargo, cada solicitud abría una nueva conexión **TCP**, lo que resultaba en un rendimiento menos eficiente.
- **HTTP/1.1:** En 1997 mejoró significativamente el rendimiento al permitir la reutilización de conexiones TCP para varias solicitudes, lo que reducía la sobrecarga de establecer y cerrar conexiones repetidamente. También introdujo pipelining y soporte para encabezados de compresión (gzip) y autenticación más avanzados.
- **HTTP/2:** En 2015 ofrece una mayor eficiencia en la transferencia de datos. Introduce el multiplexado, la compresión de encabezados y otros mecanismos para reducir la latencia y mejorar el rendimiento de las aplicaciones web.
- **HTTP/3:** Es la versión más reciente del protocolo **HTTP** y se espera que reemplace a **HTTP/2** en el futuro.
  - Se basa en el protocolo desarrollado por **Google QUIC (Quick UDP Internet Connections)** y utiliza **UDP** en lugar de **TCP** para mejorar aún más el rendimiento y la seguridad de la transferencia de datos.
  - Integración de TLS 1.3 por defecto (cifrado obligatorio).

# El protocolo HTTP (1)

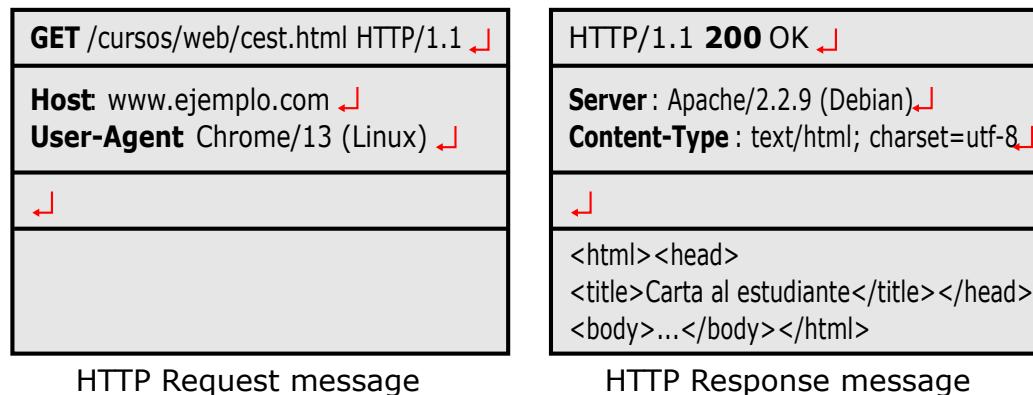


# El protocolo HTTP (2)

- La estructura de los mensajes es la misma para request y response.



- Dos ejemplos de mensajes **HTTP**. Los saltos de línea son importantes por lo que se muestran en rojo.



# El protocolo HTTP - Requerimientos y respuestas

- Una línea inicial de **request (solicitud)** tiene tres partes:
  - El nombre del método, en mayúsculas.
  - El path local del recurso solicitado.
  - La versión de **HTTP** utilizada.
  - **GET path/to/file/index.html HTTP/1.0**
- Una línea inicial de **response (respuesta)** tiene tres partes:
  - La versión **HTTP**.
  - El código de status de la respuesta.
  - Descripción del status.
  - **HTTP/1.0 200 OK**
  - **HTTP/1.0 404 Not Found**
- El primer dígito del código de status identifica la categoría:
  - 1xx indica un mensaje de información únicamente.
  - 2xx indica éxito en general.
  - 3xx redirecciona el cliente a otro URL.
  - 4xx indica un error en la parte del cliente.
  - 5xx indica un error en la parte del servidor.



# El protocolo HTTP – Códigos comunes

- Algunos códigos comunes son:
  - **200 OK**: El pedido tuvo éxito, y el recurso es returned en el cuerpo del mensaje.
  - **404 Not Found**:
  - El recurso solicitado no existe.
  - **301 Moved Permanently**
  - **302 Moved Temporarily**
  - **303 See another** (Solo HTTP/1.1) :
    - El recurso se movió a otro URL y debe ser pedido automáticamente por el cliente. Usualmente es utilizado por un script que redirecciona al visitante.
  - **500 Server Error**: Error inesperado. Típicamente un error en algún script alojado en el servidor que impide que se ejecute correctamente.
- El detalle completo de los códigos puede encontrarse en las especificaciones de la **W3C**. HTTP 1.0: RFC 1945 y HTTP 1.1: RFC 2616.



# El protocolo HTTP – Header lines

- Las líneas de encabezamiento proveen información sobre el pedido, la respuesta o el objeto que se está enviando en el cuerpo del mensaje.
- La estructura es **Header-Name: value**
  - HTTP 1.0 define 46 headers pero ninguno es requerido.
  - HTTP 1.1 define 46 headers y requiere uno, el denominado **Host**, en los pedidos realizados.
- Algunos headers importantes del cliente:
  - **User-agent:** es el programa que realiza el pedido, de la forma "Nombre/x.xx"
  - Ejemplo: **User-agent: Chrome/44.0.2403.157**
- Algunos headers importantes del servidor:
  - **Server:** identifica el software del servidor, en el mismo formato "Nombre/x.xx"
  - Ejemplo: **Server: Apache/2.4.2 (Win32)**
  - **Last modified:** es la fecha de modificación del recurso otorgado.

# El protocolo HTTP - Métodos

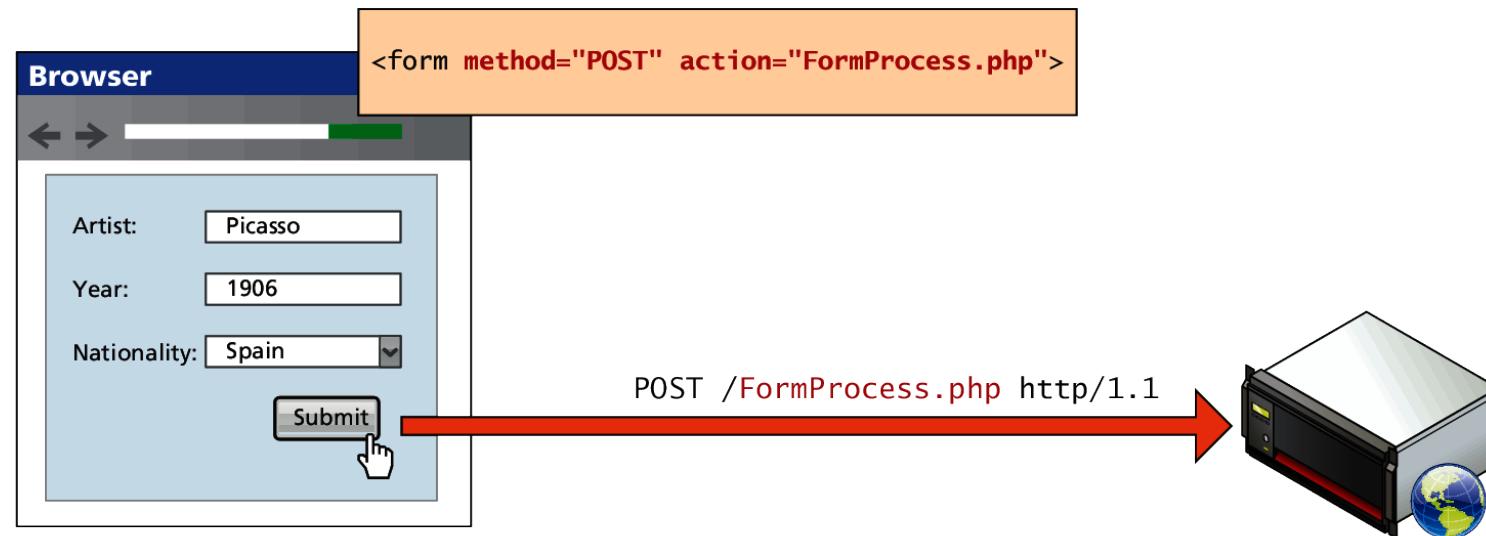
- Además de **GET**, **HTTP** define los siguientes métodos:
  - **HEAD**: solicita una respuesta idéntica a la que correspondería a una petición **GET**, pero sin el cuerpo de la respuesta.
  - **POST**: Envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas.
  - **PUT**: realiza un upload de un recurso especificado (archivo). Es la alternativa más eficiente para subir archivos a un servidor, esto es porque en **POST** utiliza un mensaje multiparte y el mensaje es decodificado por el servidor.
  - **PATCH**: Sirve para aplicarle modificaciones parciales a un recurso.
  - **DELETE**: Borra el recurso especificado.
  - **TRACE**: solicita al servidor que envíe de vuelta en un mensaje de respuesta, en la sección del cuerpo de entidad, toda la información que reciba del mensaje de solicitud. Se utiliza con fines de comprobación y diagnóstico.
  - **OPTIONS**: Devuelve los métodos **HTTP** que el servidor soporta para un URL específico. Esto puede ser utilizado para comprobar la funcionalidad de un servidor Web mediante petición en lugar de un recurso específico.

**CONNECT**: Se utiliza para saber si se tiene acceso a un host.

# El protocolo HTTP – El método POST

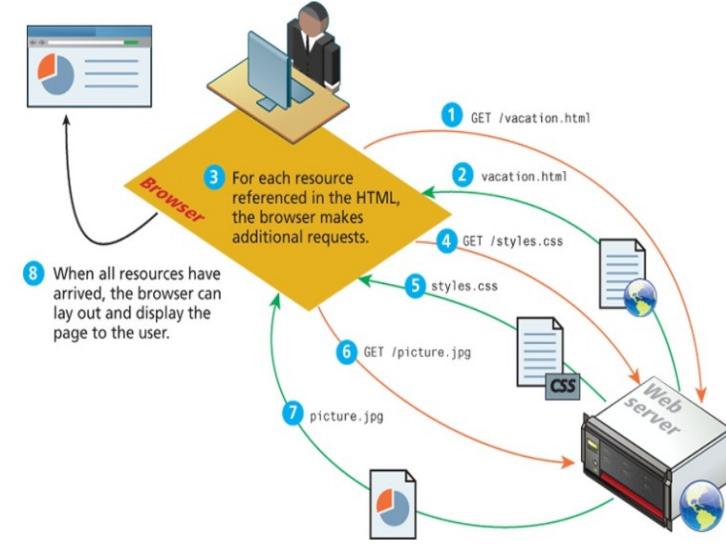
- El método **POST** es esencial para la interacción Web.
  - Este método envía información al servidor para ser procesada por quien corresponda, como un script CGI, PHP, Node JS, Java o cualquier tecnología del lado del servidor.
- Diferencias con el **GET**:
  - Se envía un bloque de datos adicional, en forma de headers.
  - El URI no es un recurso a transferir, sino el responsable del procesamiento.
  - El resultado a transferir es el output de ese procesamiento.
- El método **POST** se utiliza típicamente para enviar información desde un formulario.
- Sin embargo, el método **GET** también puede utilizarse para el mismo fin!
  - Esto es así dado que los datos pueden incluirse en el URL
- Solo debe utilizarse cuando los datos son reducidos.
- Caso contrario, deberá utilizarse el método **POST**:
- En ambos casos, el **URL** será **codificado (URL-encoded)** para transmitir los caracteres especiales (espacio, &, %, etc.)
- Al momento de implementar un cliente o un servidor Web deben tenerse en cuenta estos y muchos otros detalles para establecer una conversación HTTP.

# Solicitudes GET y POST



# Navegadores web

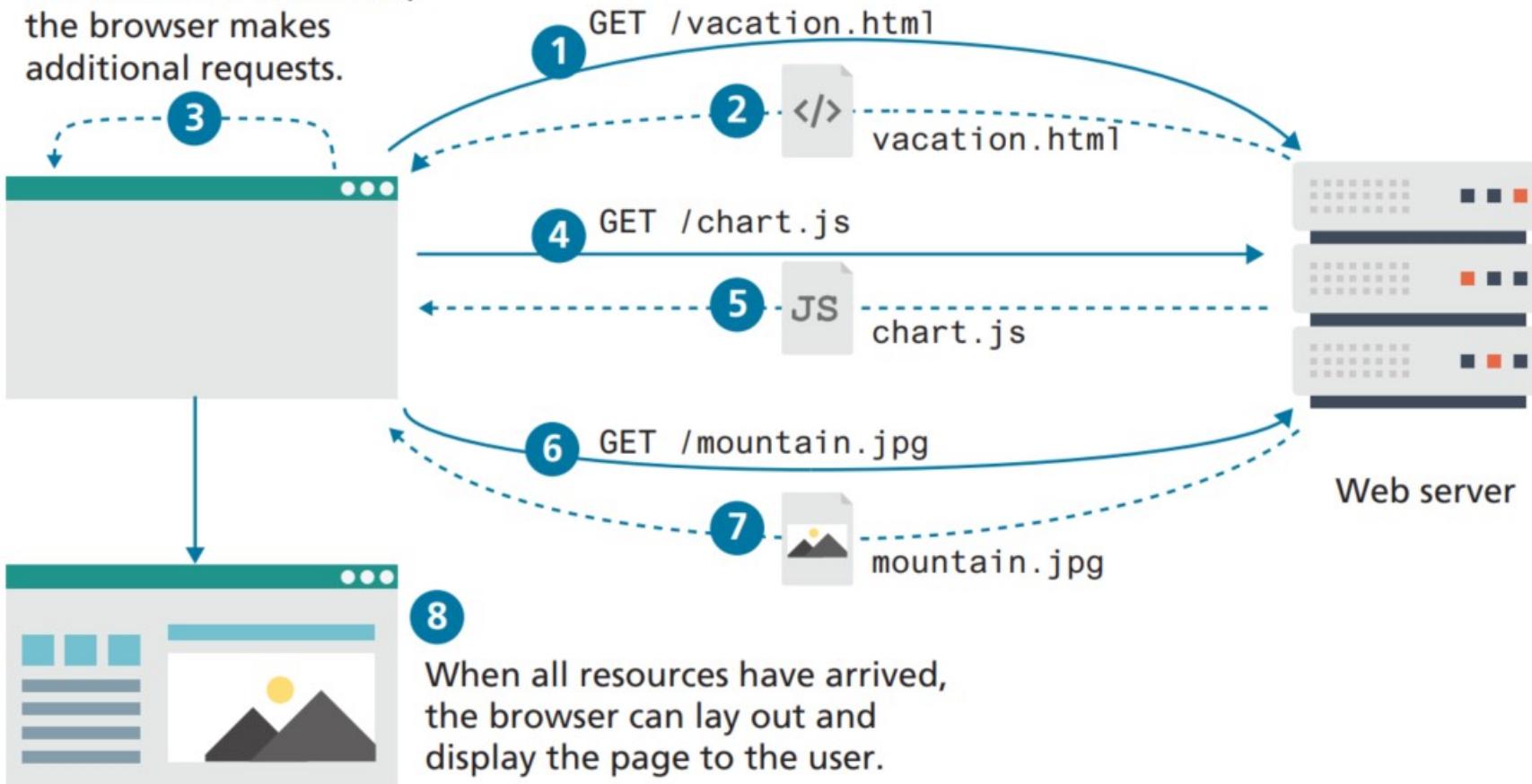
- Se puede creer que cada página web es devuelta en una **única respuesta HTTP** desde el servidor web hacia nuestro **navegador** esto no es en realidad lo que sucede.
- El **navegador cliente**, que solicita la página HTML y una vez devuelta **analiza** el código para encontrar todos los recursos a los que se hace referencia desde dentro, como imágenes, hojas de estilo y scripts.
  - Solo cuando se han recuperado todos los archivos, la página está completamente cargada para el usuario.
  - Una sola página web puede hacer referencia a **cientos de archivos** y **requiere muchas solicitudes y respuestas HTTP**.



Podemos ver el tráfico HTTP que requiere una web a través de las **herramientas para desarrolladores** con las que cuentan los navegadores web.

# HTTP – Request y parsing del navegador

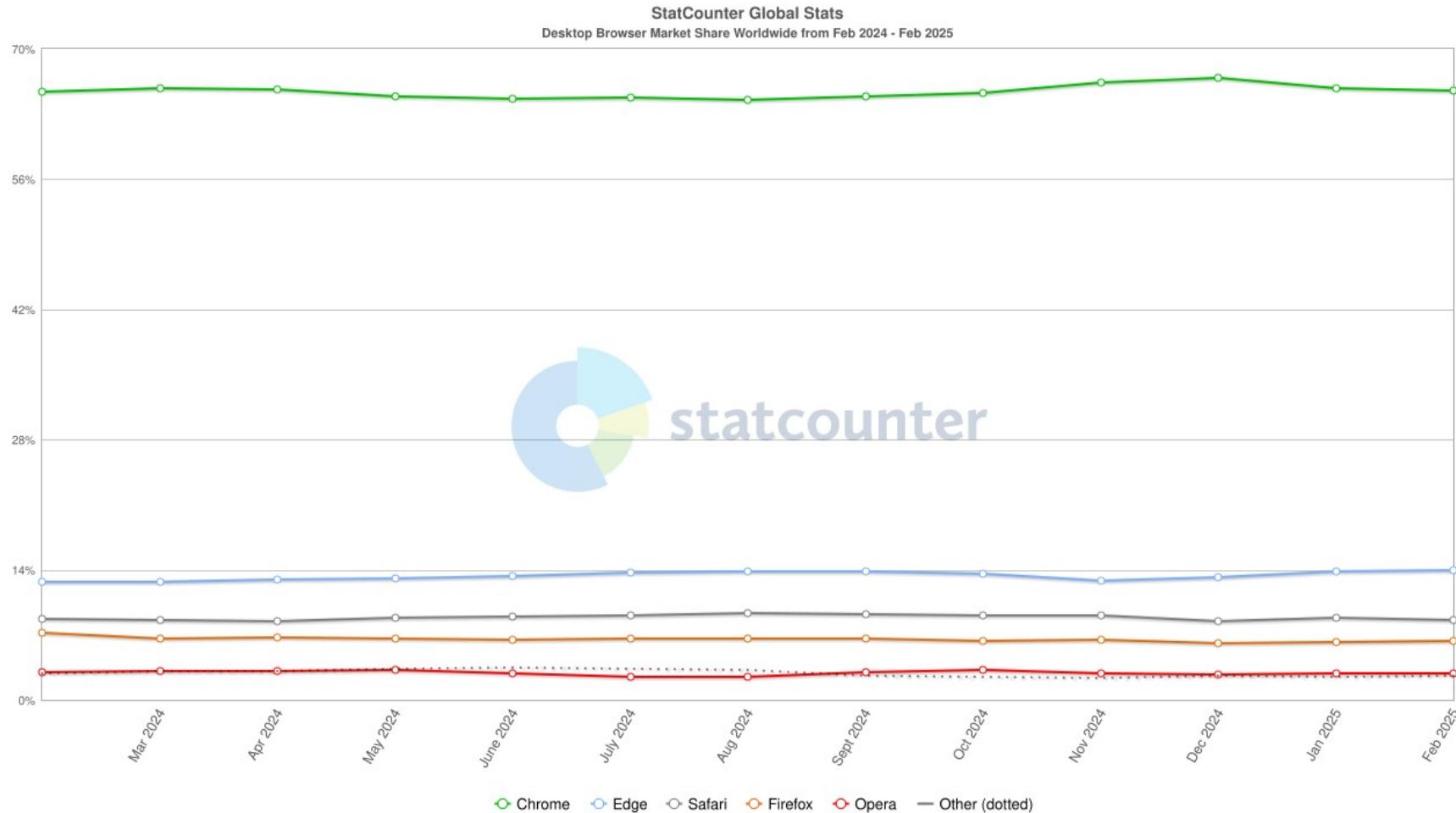
For each resource referenced in the HTML, the browser makes additional requests.



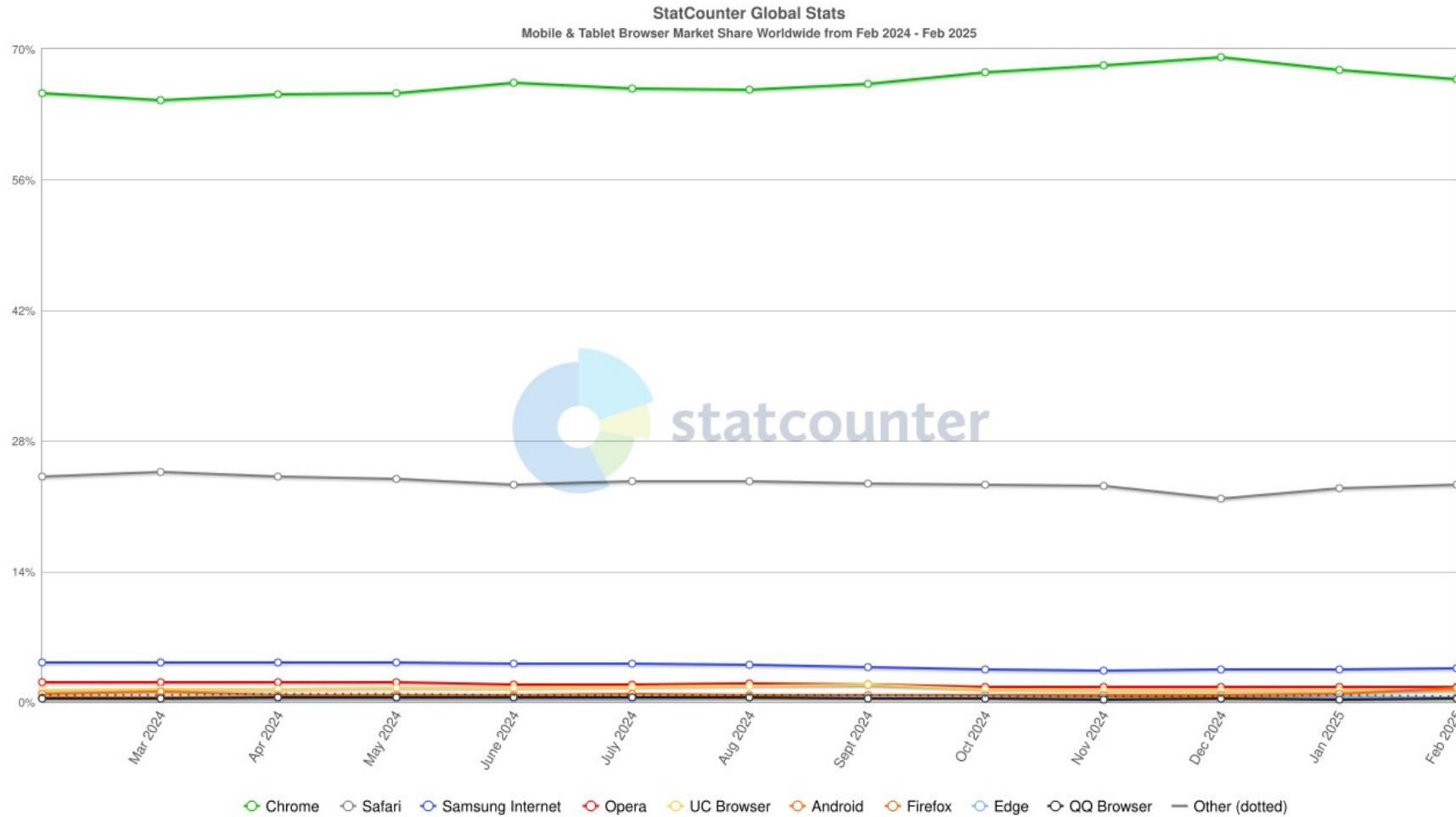
# Navegadores web

- **Renderizado:**
  - La acción de interpretar todo el **documento HTML** junto con las imágenes y otros recursos para mostrarlos dentro de la ventana del navegador se denomina **renderizado de la página web**.
  - Este proceso increíblemente complejo se implementa de manera diferente para cada navegador (Firefox, Chrome, Safari, Explorer y Opera) y es la causa de que los sitios se vean diferentes en diferentes navegadores.
- **Caché de navegación:**
  - Una vez descargada una página Web del servidor, es posible que el usuario, poco tiempo después, desee ver la misma página Web y actualice el navegador o vuelva a solicitar la URL.
  - Si bien parte del contenido puede haber cambiado (por ejemplo, una nueva publicación de blog), **es probable que la mayoría de los archivos a los que se hace referencia no hayan cambiado**, por lo que **no es necesario volver a descargarlos**.
  - El **almacenamiento en caché** del navegador tiene un impacto significativo en la reducción del tráfico de red.

# Navegadores de escritorio – Market Share



# Navegadores para dispositivos móviles

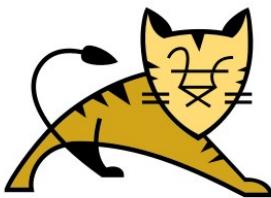


# El lado del servidor



- Las tecnologías que se encargan de trabajar del lado del servidor devolverán al cliente un resultado que pueda ser comprendido por el navegador. Algunos servidores Web (HTTP) son:
  - **Apache HTTP Server Project**: proyecto colaborativo que tiene por finalidad el desarrollo de un servidor Web libre, robusto con capacidades de uso comercial. Tiene su origen en NCSA HTTPd desarrollado originalmente por Rob McCool. Puede ejecutar scripts de lenguajes como PHP, Perl, Python y Ruby.
  - A un año de su salida se trataba del servidor Web más utilizado en el mundo.
- **Nginx**: Es un servidor Web/proxy inverso, mail proxy y proxy TCP genérico programado originalmente por Igor Sysoev. Es software libre y de código abierto, licenciado bajo BSD simplificada. Es multiplataforma.
  - El sistema es usado por una larga lista de sitios Web conocidos como: Yandex, VK, WordPress, Netflix, GitHub, SourceForge, TorrentReactor y partes de Facebook (como el servidor de descarga de archivos pesados).

# El lado del servidor (2)

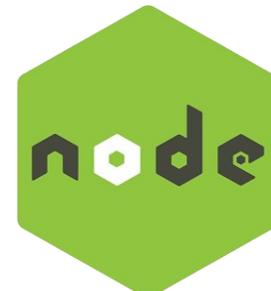


- **Internet Information Services (IIS)**: es un servidor Web y un conjunto de servicios para sistemas operativos Windows. Se encuentra integrado a los sistemas operativos para servidores Windows y las ediciones profesionales o superiores. Ofrece servicios como: FTP, SMTP, NNTP y HTTP/HTTPS.
  - Ejecuta scripts en ASP, ASP.net, PHP y Perl.
- **Apache Tomcat**: es un servidor Web y un contenedor de servlets open source desarrollado por la ASF. Implementa varias especificaciones de Java EE entre los que se incluyen servlets, JSP, EL y Web Sockets.
- **GlassFish/Payara**: es un proyecto de desarrollo de servidor de aplicaciones open source iniciado por Sun Microsystems para la plataforma JavaEE. Se trata de la implementación de referencia de JavaEE soportando: EJB, JPA, JSF, JMS, RMI, JSP y servlets.
- **WEBrick**: servidor HTTP que permite ejecutar scripts Ruby.

# El lado del servidor (3)

- **Node.JS**: si bien es un entorno de ejecución basado en el lenguaje de programación **ECMAScript** y por tanto, no se trata de un servidor Web se destaca la facilidad de crear uno utilizando el módulo http.

```
const http = require('http');
//Crea el objeto servidor:
http.createServer(function (req, res) {
    res.write('Hello World!'); //Escribe la respuesta al cliente.
    res.end(); //Se cierra la respuesta.
}).listen(3000); //Indica que el servidor atiende en el puerto 3000
```



# El lado del Servidor - Tecnologías

- Cualquiera sean las características físicas del servidor, se debe elegir por una pila de aplicaciones para que el sitio Web funcione.
- Esta pila incluirá:
  - Sistema Operativo.
  - Servidor Web.
  - Software de Base de Datos.
  - Lenguaje de Programación.
- La decisión depende del tipo de proyecto y de los conocimientos que tengamos.

# Bibliografía

- **Web:** James Marshall. URL: “***HTTP Made Really Easy***”. Año: 1997  
<http://www.jmarshall.com/easy/http/>
- **Web:** Desarrollo de aplicaciones Web, 2013  
<http://inciart.com/teach/ucr/appweb/2011b/libro/>
- **Libro:** Philip Crowder, David A. Crowder. “***Creating Web Sites Bible***”. 3rd Ed. Wiley Publishing Inc. Año: 2008.
- **Web:** wikiHow - “***¿Cómo activar telnet en Windows 7?***” . URL:  
<http://es.wikihow.com/activar-Telnet-en-Windows-7>
- **Web:** StatCounter. “***Desktop Browser Market Share***”. URL:  
<https://gs.statcounter.com/browser-market-share/desktop/worldwide>
- **Web:** StatCounter. “***Mobile Browser Market Share***”. URL:  
<https://gs.statcounter.com/browser-market-share/mobile/worldwide>



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

Unidad 2 – HTML

HTML primeros pasos

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

- **Objetivos**

- Comprender cómo estructurar el contenido de una página web.
- Identificar los principales elementos de HTML.
- Crear un sitio Web sencillo que relacione a través de enlaces múltiples páginas web.

- **Temas a desarrollar:**

- **Lenguaje de hipertexto:**
  - Definición.
  - Estructura.
- **Elementos:**
  - Básicos.
  - Semánticos.
- **Etiquetas y Atributos:**
  - Textos, enlaces, listas, imágenes, multimedia, tablas y elementos de bloque.



- El **W3C** es un consorcio internacional que genera recomendaciones y estándares que aseguran el crecimiento de la **World Wide Web** a largo plazo.
- Fue fundado por **Tim Berners-Lee** el 1 de octubre de 1994 tras dejar el **CERN**.
- Tiene actualmente sede en el Instituto Tecnológico de Massachusetts (MIT).
- Las especificaciones de la **W3C** siguen un proceso de maduración hasta convertirse en recomendaciones (o especificaciones).
- Las especificaciones muy grandes se dividen en módulos.
- Está compuesto por más de 400 miembros incluyendo:
  - Empresas,
  - Agencias gubernamentales,
  - Universidades e
  - Individuos.



# W3C Maduración de especificaciones

- Las especificaciones de la **W3C** siguen los siguientes pasos:
  - **Borrador de trabajo (WD)**: Un documento **WD** es la primera forma de un estándar que está disponible públicamente.
  - **Recomendación candidata (CR)**: Es una versión de una norma más madura que el **WD**. En este punto, el grupo responsable de la norma está convencido de que la norma cumple con su objetivo. El propósito de la **CR** es obtener ayuda de la comunidad de desarrollo en cuanto a la implementación de la norma.
  - **Recomendación propuesta (PR)**: pasados los dos niveles anteriores, en esta etapa el documento se presenta al **Consejo Asesor del W3C** para su aprobación final.
  - **Recomendación de W3C (REC)**: En este punto, la norma ha sido objeto de amplia revisión y pruebas, tanto en condiciones teóricas y prácticas. La norma está respaldada por el **W3C**, lo que indica su disposición para su despliegue al público, y fomentar un apoyo más generalizado entre los ejecutores y los autores.
  - **Revisiones posteriores**: Una recomendación puede ser actualizada o ampliada por separado con erratas publicadas o editor de borradores no técnicos, hasta que haya suficientes cambios sustanciales se acumulan para producir una nueva edición o nivel de la recomendación.

# HTML – Lenguaje y estándar

- **HTML (*HyperText Markup Language*)** es un lenguaje utilizado para crear documentos web.
  - La primer versión del estándar fue desarrollada por Tim Berners-Lee a finales de los 80's.
- Se trataba de un subconjunto simplificado de **SGML (*Simple Generalized Markup Language*)**.
- Así como otros estándares relacionados con la Web es desarrollado bajo la autoría del **W3C (*World Wide Web Consortium*)**.
- Tiene varias versiones. En cada una de ellas se han incorporado y suprimido características, con el fin de hacerlo más eficiente y facilitar el desarrollo de páginas web compatibles con distintos navegadores y plataformas.
  - La última especificación es **HTML 5.0**

# Elementos

- Un **documento HTML** está compuesto por **elementos** que definen su estructura.
- Cada **elemento** tiene su propósito particular. Como mostrar un encabezamiento, una imagen o información que debe ser utilizada por el navegador.
- Los **elementos**:
  - Nos posibilitan determinar cómo se conforma la página y sus secciones.
  - Se definen utilizando instrucciones especiales que se conocen con el nombre de **etiquetas** o **tags**. Por lo general, se componen de **dos tags** (uno de apertura y otro de cierre).
  - Dentro de los elementos podemos ubicar contenido, tales como texto o más elementos.

# Nuestra primera página web

- A continuación construiremos nuestra primer página web utilizando **HTML**.

```
<!DOCTYPE html>

<html lang="es">
    <!-- Esto es un comentario. lang="es" se trata de un código de lenguaje.
        Los códigos de lenguajes están definidos en la ISO 639-1.-->
    <head>
        <title>Mi primer página web</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <p>Hola Mundo!!!</p>
    </body>
</html>
```

## Nuestra primera página web (2)

- Todo documento **HTML** comienza con la etiqueta `<html>` y finaliza con `</html>`.
- Se divide en dos secciones: el encabezado o cabecera (**head**) y el cuerpo (**body**).
  - **Head**: Contiene información sobre el documento, tales como el título y metainformación para describir el contenido.
  - **Body**: Almacena el contenido del documento. Es decir, lo que será mostrado en la ventana del navegador.
- Cada **tag** se compone de un nombre seguido (opcionalmente) de una lista de **atributos**. Todo esto dentro de los símbolos `<` y `>`.
- Esta información **no** se muestra en el browser.
- Generalmente, el nombre de un **tag** es una abreviatura de su funcionalidad.
- Los nombres de los **tags** ***no son case sensitive***. O sea, es lo mismo `<BODY>` que `<body>`.
  - No obstante algunas versiones del estándar **exigen** que los nombres de las etiquetas se escriban en **minúsculas**.

## Elementos contenedores

- Los **elementos contenedores** o **elementos no vacíos**, son aquellos que tienen un **tag de inicio** o apertura y un **tag de cierre**.
- El texto comprendido dentro de los **tags** seguirá las instrucciones del mismo.
- Un **tag de cierre** tiene el mismo nombre que su correspondiente **tag de apertura** precedido por “/”.
- Los **tags de cierre** no llevan atributos.
- Por ejemplo:

Es una tarde **<strong>lluviosa</strong>**.

Resultará en:

Es una tarde **lluviosa**.

# Elementos independientes

- Algunos pocos **tags** no tienen su **tag** de cierre.
  - Esto se debe a que su función es ubicar elementos individuales en la página.
  - Por ejemplo: el **tag** de imagen `<img>` hace referencia a un gráfico que se debe mostrar en la página.
  - Otros **tags independientes** son `<br/>` **line break** (salto de línea) y `<hr/>` de **horizontal rule** (regla horizontal).
- Las versiones más restrictivas de **HTML** exigen que los **tags** independientes lleven el símbolo “/” antes de finalizar. Es decir: `<br>` se escribiría `<br/>`.
- Los **elementos independientes** son conocidos también como **elementos vacíos**.

# Atributos

- Los **atributos** son agregados a un **elemento** para extender o modificar la acción que este realiza.
- Se pueden agregar múltiples atributos a un único **tag**. Los **atributos** (*si existen*) se ubican a continuación del nombre del **tag** de apertura separados por espacios. El orden de aparición no es incidente.
- La mayoría de los **atributos** toman valores, que se especifican ubicando el símbolo “**=**” inmediatamente después de su nombre.
- Los **valores** están limitados a **1024** caracteres y en **ocasiones** son **case sensitive**.
- Es buena práctica que los valores aparezcan encerrados entre comillas dobles. Ejemplo: ``

# Análisis de documentos HTML

```
<html>

<head>

    <title>Page title</title>

</head>

<body>

    <h1>This is a heading</h1>

    <p>This is a paragraph.</p>

    <p>This is another paragraph.</p>

</body>

</html>
```

- El estándar **HTML** exige que el documento esté comprendido en el contenedor **<html>**.
- A pesar de ello, la mayoría de los **browsers** pueden desplegar los contenidos de un documento incluso si estos **tags** se omiten.
- Todos los documentos **HTML** están compuestos por dos estructuras principales: **head** y **body**.
  - La excepción ocurre cuando el documento contiene un frameset en lugar del **body**.
- No obstante la primera línea de nuestro documento no es **<html>** es **<!DOCTYPE html>**

# DOCTYPE

- La declaración **DOCTYPE** no es un tag **HTML**.
- Se trata de una instrucción que informa al **browser** la **versión** del lenguaje de marcado en el cual la página está escrita.
- **DOCTYPE** debería aparecer al principio de todo documento **HTML**. Antes del tag **<html>**.
- **DOCTYPE** se refiere a un **Document Type Definition (DTD)**.
  - Un **DTD** especifica las reglas para el lenguaje de marcado, para que los browsers puedan mostrar el contenido correctamente.
- Cuando abre una página web, nuestro navegador:
  - Buscará la declaración **DOCTYPE**.
  - La examinará para determinar qué estándares usa la página web.
  - Esta información determinará cómo el **código HTML** es interpretado y cómo la página será mostrada en la pantalla.

## DOCTYPE (2)

- Con algunas de estas declaraciones **DOCTYPE** podemos encontrarnos al analizar un documento **HTML**.
  - **HTML 4.01:**
    - `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">`
  - **XHTML 1.0:**
    - `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
- Sin embargo, durante el curso trabajaremos siempre con **HTML5**:
  - `<!DOCTYPE html>`

- El encabezado de un documento **HTML** está definido por el tag **<head>**, contiene información que describe un documento **HTML**.
- **<head>** no tiene atributos, sirve como **contenedor** de otros tags que ayudan a **definir** y **administrar** los contenidos del documento. Algunos de esos tags son:
  - **<title>**: provee una descripción del contenido de la página. Es importante incluirlo (Se muestra en **Favoritos** y **Motores de búsqueda**).
  - **<link>**: define una relación entre el documento actual y otro. Principalmente se lo utiliza para vincular el documento con **Hojas de Estilo en Cascada (CSS)**.
  - **<script>**: Si bien **NO SE RECOMIENDA**, scripts **JavaScript** y **VBScript** pueden ser agregados al documento dentro de su encabezado.
  - **<meta>**: proveen información sobre el documento. Por ejemplo: palabras claves y descripciones que usan los motores de búsqueda.

- El cuerpo de un **documento HTML** está delimitado por los tags **<body>** y **</body>**.
- Incluye los contenidos de un documento. Es decir lo que se quiere mostrar en la ventana del **browser**.
- El contenedor **<body>** puede consistir en unos pocos párrafos de texto, una imagen simple o complejas combinaciones de texto, imágenes, tablas y objetos multimedia.

# Etiquetas de Texto

- Los **Párrafos** y **Encabezamientos (Headings)** son considerados junto con las **listas** y **enumeraciones** elementos de **nivel de bloque** o **Block-Level Elements**.
- Son siempre formateados por el **browser** con un salto de línea antes y después, agregando espacios adicionales arriba y debajo del elemento.
- Las etiquetas de texto son: **párrafos**, los **encabezados** y las **citas (blockquotes)**.
- Las etiquetas de texto se encuentran relacionadas con fuentes, diferentes formas de representación de texto (tamaño, negritas, subrayados, etcétera) y también en lo referente a acrónimos y abreviaturas entre otras características de **HTML**.
- Sin embargo, muchas etiquetas han sido reemplazadas por propiedades **CSS** y su uso es desaconsejado.

## Etiquetas de Texto (2)

- **Párrafos**: Los tags `<p>` y `</p>` denotan el principio y final de un párrafo cuando es usado como contenedor.
  - Muchos **browsers** permiten que el tag `<p>` sea usado sin su correspondiente tag de cierre abriendo un nuevo párrafo.
  - El método de usar contenedores es recomendado, sobre todo cuando se usan **Hojas de Estilo**.
- **Headings**: Son mostrados en negrita, saltos de línea automáticos y un espacio extra abajo y arriba.
  - Existen seis niveles de encabezados que van desde `h1` hasta `h6`.
  - Los **browsers** muestran los encabezados con un tamaño de fuente descendente. Es decir, `h1` tiene fuente más grande que `h2`, `h2` que `h3`, etc.

## Etiquetas de Texto (3)

- Otras etiquetas de texto:

- **<b>...</b>**: permite definir un texto en negrita. **No usar.**
- **<i>...</i>**: define un texto en cursiva. **No usar.**
- **<u>...</u>**: define un texto en subrayado. **No usar.**
- **<ins>...</ins>**: define un texto insertado. **No usar.**
- **<del>...</del>**: define un texto eliminado. **No usar.**
- **<sub>...</sub>**: permite definir un subíndice. **No usar.**
- **<sup>...</sup>**: permite definir un superíndice. **No usar.**
- **<em>...</em>**: permite destacar con énfasis un texto.
- **<strong>...</strong>**: permite destacar con énfasis fuerte un texto.
- **<acronym>...</acronym>**: se emplea para definir acrónimos.
- **<pre>...</pre>**: Las líneas contenidas entre los tags se muestran como están en el archivo **HTML** fuente (**incluye espaciados multiples y saltos de línea**). Generalmente es mostrado en una fuente **monoespaciada** como por ejemplo **Courier**.

# Listas y Enumeraciones

- La especificación original de **HTML** incluyó tags para definir 5 elementos del tipo de listas: **Listas ordenadas**, **listas no ordenadas (viñetas)**, listas de definiciones, menús, listas de directorios.
- Las listas de directorio (**<dir>**) y de menú (**<menu>**) se han declarado **obsoletas** en las nuevas versiones del estándar.
- Las **listas** y los **ítems** que contienen son **Block-Level Elements**, esto quiere decir que espacios de línea serán agregados automáticamente antes y después de ellos.

## Listas no ordenadas

- Una **lista no ordenada** se utiliza para mostrar una colección de ítems relacionados que **no** deben aparecer en un orden particular.
- Las listas son dibujadas con una identación precedidas de una **viñeta** por cada ítem (**Lo hace el browser**).
- El tipo de **viñeta** se puede cambiar a través del atributo **type**.
- Los ítems se especifican a través de los tags **<li>** y **</li>**.

HTML	Browser
<code>&lt;ul&gt;   &lt;li&gt;Item 1&lt;/li&gt;   &lt;li&gt;Item 2&lt;/li&gt; &lt;/ul&gt;</code>	<ul style="list-style-type: none"><li>• Item 1</li><li>• Item 2</li></ul>

# Listas ordenadas (Enumeraciones)

- Las **listas ordenadas** son usadas cuando mantener la secuencia de los ítems es importante.
- Son mostradas en una identación con un número (*automáticamente insertado por el browser*) precediendo a cada ítem de la lista.
- Este tipo de listas siguen la misma estructura básica que las listas desordenadas: la lista completa está contenida por los tags **<ol>** y **</ol>** y cada ítem individual se indica usando el tag **<li>** y **</li>**.
- El esquema de números puede ser modificado.

HTML	Browser
<pre>&lt;ol&gt;   &lt;li&gt;Item 1&lt;/li&gt;   &lt;li&gt;Item 2&lt;/li&gt; &lt;/ol&gt;</pre>	1)Item 1 2)Item 2

- Clasifican como enlaces los elementos que nos permiten enlazar un documento con otro. Este conjunto está comprendido por: **Tag anchor, Area, Map.**
- El **tag anchor** es el de uso más común.
- Se define mediante los tags `<a>...</a>`.
- Además de permitir vincular un documento con otro, un **anchor tag** sirve también para rotular un fragmento (también llamado **anchor nominado**), que es usado como una referencia para vincular un punto en particular de un documento **HTML**.

## Enlaces – Links (3)

- Los atributos más importantes del tag anchor.
  - **href="url"**: Especifica la URL destino.
  - **name="texto"**: Ubica un identificador de fragmento en un documento HTML.
  - **title="texto"**: Título para el documento destino. Se ve como un tooltip.
  - **target="texto"**: Especifica el nombre de una ventana o frame en el cual el documento destino debe ser mostrado.
  - **accesskey="character"**: Asigna una tecla (atajo de teclado) al enlace.
  - **tabindex="number"**: Especifica la posición del elemento actual en el orden de tabulación del documento actual. Debe tener un valor entre **0** y **32767**.

- Ejemplo:

```
<a href="http://www.fcad.uner.edu.ar" title="Inicio"  
target="_blank">FCAD</a>
```

# URLs Absolutas

- Una URL absoluta está formada por:
  - Un esquema (opcional).
  - El nombre del host.
  - Una ruta.
  - Un nombre de archivo específico (opcional).
- Cuando se vincula un documento con otros servidores **se necesita usar una URL absoluta.**
- El siguiente es un ejemplo de un link con una URL absoluta:  
`<a href="http://www.littlechair.com/web/index.html">...</a>`

## URLs Relativas

- Un **URL** relativo provee un puntero a otro documento relativo a la localización del documento actual.
- La sintaxis está basada en estructuras de rutas relativas del sistema operativo Unix.
- Ejemplo: `<a href="index.html">...</a>`
- Ejemplo: `<a href="../tudw/index.html">...</a>`

- A pesar de que no existe una clasificación las **tablas** pueden usarse de las siguientes formas:
  - **Tabla de Datos**: este tipo de tabla es la más básica y es el propósito con el cual fue diseñado el tag por los creadores de **HTML**. Contiene filas y columnas con datos textuales.
  - **Alineamiento del Texto**: Las tablas son frecuentemente usadas para crear efectos como columnas identación y espacio libre. **No recomendable**.
  - **Plantilla de página**: Muchos diseñadores Web usan grandes estructuras de tablas como contenedor para darle estructura a su página. **No recomendable**.
  - **Contenedor multiparte de imágenes**: Las tablas pueden ser usadas para mantener un gran gráfico que ha sido dividido en secciones para generar animaciones y efectos. **No recomendable**.

## Tablas - Estructura básica

- En su forma más básica las **tablas** están formadas por celdas, organizadas en filas y columnas.
- Se pueden controlar las características de toda la tabla a distintos niveles (de tabla, filas y celdas individuales).
  - Actualmente no existe forma de controlar las columnas como grupo.
- Las etiquetas que se utilizan para describir una **tabla** son **<table>**, **<thead>**, **<tbody>**, **<tfoot>**, **<tr>**, **<th>** y **<td>**.

# Bibliografía

- **Libro:** Web Design in a Nutshell (1ra Edición) (Caps: 1, 5, 6, 7, 8, 9, 10).
  - Jennifer Niederst. O'Reilly. 2008
- **Libro:** Creating Web Sites Bible 3rd Ed.
  - Philip Crowder, David A. Crowder. Wiley Publishing Inc. 2008.
- **Web:** W3schools
  - <http://www.w3schools.com>.
- **Libro:** HTML5 Foundations, 2013
  - Matt West. Wiley. 2013
- **Libro:** Randy Connolly, Ricardo Hoar. *Fundamentals of Web Development, Global Edition*. Pearson. 2015.



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

Unidad 2 – HTML

Elementos Semánticos

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

- **Objetivos**

- Comprender cómo estructurar el contenido de una página Web.
- Identificar los principales elementos de HTML.
- Crear un sitio Web sencillo que relacione a través de enlaces múltiples páginas Web.

- **Temas a desarrollar:**

- **Lenguaje de hipertexto:**
  - Definición.
  - Estructura.
- **Elementos:**
  - Básicos.
  - Semánticos.
- **Etiquetas y Atributos:**
  - Textos, enlaces, listas, imágenes, multimedia, tablas y elementos de bloque.



# HTML - Elementos <div> y <span>

- **<div>**: Este elemento sirve como **contenedor genérico** de nivel de bloque.
  - A diferencia de los tags **<h1>**, **<p>** o **<strong>**, no tiene un significado semántico asociado.
  - Un elemento **<div>** por sí sólo no provoca cambios visuales en una página web. Es por ello que los desarrolladores comúnmente utilizan **<div>** para envolver bloques de contenido de sus **páginas HTML**, controlando con **CSS** su ancho, ubicación y otras características de presentación.
  - Una etiqueta **<div>** puede contener elementos de nivel de bloque y elementos en línea, incluso otros **divs**.
- **<span>**: El elemento **<span>** es un **contenedor genérico** en línea sin significado semántico.

- Sabemos que en nuestra lengua:
  - La **sintaxis** se refiere a la **gramática** y
  - La **semántica** al **significado** que tienen las estructuras y elementos lingüísticos que la componen.
- Cuando hablamos de un **lenguaje de marcas**, como **HTML** la semántica se refiere al **significado que tienen los elementos**.
  - Por ejemplo: La etiqueta **<h1>** define el título principal del contenido de la página.
- Los elementos semánticos:
  - Indican qué contienen en lugar de cómo se deben formatear.
  - Pueden no tener una representación asociada y la misma deben definirse utilizando reglas **CSS**.

## HTML5 – Elementos semánticos (2)

- Los **elementos semánticos** son una de las características más valoradas de **HTML5** ya que nos permiten definir secciones de nuestro documento web agregando metadatos semánticos
- De esta forma podremos ayudar a los motores de búsqueda como **Google** o cualquier otro mecanismo automático a interpretar e indexar más correctamente los contenidos de nuestro sitio web.
- Otras ventajas de desarrollar webs utilizando elementos semánticos son:
  - **Mantenibilidad**: más fáciles de actualizar y cambiar.
  - **Accesibilidad**: mejora la experiencia de usuario si se utiliza un software de lectura.



## HTML5 – Elementos semánticos (3)

- En lo que se refiere a la estructura del documento, en **HTML4** para definir las partes del documento debíamos hacer uso de la etiqueta **<div>**.
- El problema se planteaba en la imposibilidad de asignarles la semántica correspondiente a las diferentes partes.
  - Por ejemplo, si bien podíamos aplicar un **id** con el valor **nav** o **footer** (barra de navegación y pie), esto no era más que el valor de un atributo y no le daba significado semántico diferente al elemento.
- A partir de **HTML5** se definen etiquetas que nos permiten estructurar el cuerpo de una página con una semántica específica para cada elemento. Entre estas etiquetas encontramos:
- **<header>**, **<hgroup>**, **<nav>**, **<section>**, **<article>**, **<aside>**, **<footer>**.

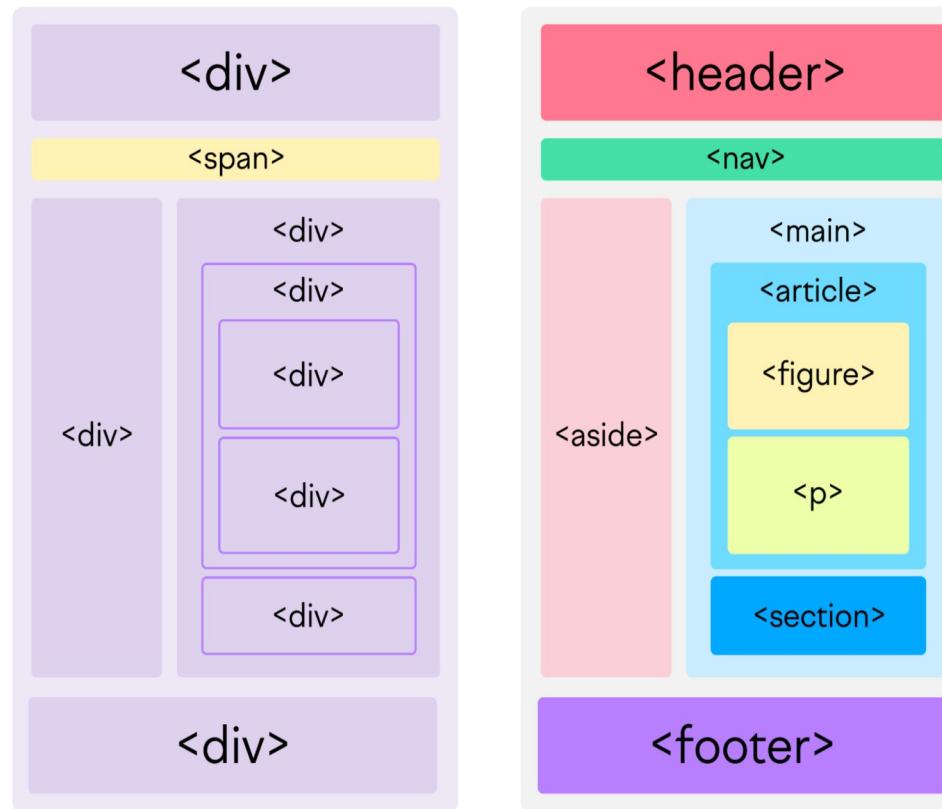
## HTML5 – Elementos semánticos (4)

- **<header>**: define la cabecera de la página. **No confundir con la etiqueta <head>** ya que **<header>** se emplea para elementos del cuerpo del documento.
- **<nav>**: establece una barra de navegación. Aquí podremos consignar enlaces internos y también hacia otros sitios.
- **<section>**: define una sección de contenido que se representa en la página.
- **<article>**: sirve para definir artículos o contenidos que podrían llegar a ser individualizados.
- **<aside>**: se emplea para el resto de la información que se incluye en el sitio y no está directamente relacionada con el contenido principal de dicha página. Puede utilizarse como sidebar en un sitio web o aparecer dentro de un **<article>** como contenido complementario.

# HTML5 – Elementos semánticos (5)

- El elemento `<main>` en **HTML** representa el contenido principal único de una página. Agrupa la información central directamente relacionada con el propósito del documento. Solo debe haber un `<main>` por página y mejora la accesibilidad al indicar el foco principal del sitio.
- `<figure>`: se utiliza para consignar elementos secundarios al flujo del texto que opcionalmente pueden ser anotados con `<figcaption>`. Pueden contener cualquier tipo de contenido multimedia, no solo imágenes (por ejemplo: gráficos, código, tablas).
- `<hgroup>`: agrupa títulos con subtítulos. Por ejemplo, en el caso que luego de un `<h1>`, ubiquemos un `<h2>` que se relaciona de manera directa con el primero. **Actualmente en desuso**.
- `<footer>`: se utiliza para el pie de la página.

# HTML5 – Elementos semánticos (5)



Fuente: <https://www.semrush.com/blog/semantic-html5-guide/>

# Bibliografía

- **Libro:** Luis Matos. “*Entrenamiento Práctico de CSS*”. Digerati. 2007
- **Libro:** Philip Crowder, David A. Crowder. “*Creating Web Sites Bible 3rd Ed*”. Wiley Publishing Inc. 2008.
- **Web:** W3schools. <http://www.w3schools.com>.
- **Libro:** Matt West. “*HTML5 Foundations*”. Ed. Wiley. 2013
- **Libro:** Randy Connolly, Ricardo Hoar. “*Fundamentals of Web Development, Global Edition*”. Pearson. 2015.



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

Unidad 3 – CSS

CSS primeros pasos

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

- **Objetivos**
  - Conectar documentos HTML con archivos CSS.
  - Comprender el funcionamiento de reglas, atributos y valores CSS.
  - Implementar diseños complejos a través de atributos de disposición.
- **Temas a desarrollar:**
  - **CSS:**
    - Definición.
    - Enlace de documentos HTML con hojas de estilo.
  - **Sintaxis de CSS:**
    - Selectores, Clases, Pseudoclases y Pseudoelementos.
    - Atributos y valores.
  - **Disposición de elementos:**
    - floats, position, display, box model, Flex Box y CSS Grid.



- **Cascading Style Sheet** u **Hojas de Estilo en Cascada (CSS)** es un lenguaje utilizado para describir la apariencia de los **elementos HTML**, controlando, por ejemplo: las fuentes, colores, márgenes, bordes, dimensiones, fondos y posicionamientos.
- Si bien vimos que podemos aplicar estilos usando elementos **HTML** entre las ventajas de usar **CSS** podemos citar:
  - Mejor nivel de control sobre el formato.
  - Simplificación del mantenimiento del sitio.
  - Incluye características de accesibilidad.
  - Mayor velocidad de descarga.
  - Flexibilidad de salida (monitores, celulares, tablets, televisores, impresoras, etc.).

- La función principal de **CSS** es permitir separación el **contenido y la estructura** que se define en un **documento HTML** de la **representación**, que queda a cargo de las hojas de estilos.
- Esta separación es importante para un proyecto web ya que:
  - Permite la definición de criterios que se deben respetar en el sitio web.
  - Centraliza el diseño de varios archivos **HTML** en un único archivo **CSS**.
  - Ofrece la posibilidad que se definan clases para evitar la necesidad de reescribir código.

# CSS – Un poco de historia

- **CSS** fue propuesto por Håkon Wium Lie en 1994. En ese tiempo Lie trabajaba junto a Tim Berners-Lee en el **CERN**.
- La primera versión de **CSS** se convierte en recomendación del **W3C** en el año 1996.
- Luego de bastante tiempo de trabajo **CSS 2** se convierte en recomendación del **W3C** durante el año 1998.
- La especificación de **CSS 3** está dividida en varios documentos separados, llamados "módulos". Cada módulo añade nuevas funcionalidades a las definidas en **CSS 2**, de manera que se preservan las anteriores para mantener la compatibilidad.
- Debido a la modularización del **CSS 3**, diferentes módulos pueden encontrarse en diferentes estados de su desarrollo. El estado actual del desarrollo de **CSS 3** puede ser consultado en <http://www.w3.org/Style/CSS/current-work>



- **Regla:** Es una declaración que indica cómo será aplicado un estilo a uno o más **elementos HTML**.
  - Está compuesta por tres partes: un **selector**, una **propiedad** y un **valor**.
  - **Selector:** En general, es el **elemento HTML** identificado por su **etiqueta**, por una **clase** o por un **id**. La regla será válida para el elemento indicado (ejemplo: **<p>**, **<h1>** y **.unaclasecualquiera**).
  - **Propiedad:** Es el atributo del **elemento HTML** al cual se aplicará la regla (ejemplo: **font**, **color** y **background**).
  - **Valor:** Es la característica que va a ser asumida por la **propiedad** (ejemplo: fuente Arial, fondo negro, color rojo).

## CSS - Sintaxis básica (2)

- En la sintaxis de una **regla CSS** es necesario escribir el **selector** y a continuación, entre llaves, la/s **propiedad/es** y su **valor** separados por dos puntos.
- Cuando se define más de una propiedad en la regla, es necesario utilizar “;” para separarlas.
- Ejemplos:

```
body {  
    background-color: #000000; /*Fondo negro*/  
    font-weight: bold; /*Fuentes en negrita*/  
}  
  
.estiloTexto {  
    font-family: "Roboto"  
}
```

## CSS - Sintaxis básica (3) – Selector de clase

- Para aplicar reglas de estilo **CSS** no es necesario limitarse sólo a los **elementos HTML** (etiquetas).
- Es posible crear un **selector de clase** con cualquier nombre, definiendo las **reglas CSS**.
- Las clases pueden aplicarse a cualquier **elemento HTML**.
- La sintaxis para una **clase de selectores** es:

```
[elementoHTML].nombreClase{  
    propiedad: valor;  
}
```

- El nombre de la clase **NO** debe poseer números y caracteres especiales, pues existen restricciones en cuanto al uso de esos elementos.

- Se distingue del **selector de clase** por ser único, y por tanto, aplicarse solamente a un **elemento HTML** dentro del documento.
- La **regla CSS** se aplicará a aquél **elemento HTML** cuyo valor para el atributo **id** coincida con el del selector.
- La sintaxis básica para el **selector ID** está compuesta por un nombre cualquiera precedido del símbolo **#**.
- Sintaxis:

```
[nombreElemento]#idElemento{  
    propiedad: valor;  
}
```

# Conectando CSS y documentos

- Las **reglas CSS** pueden conectarse a un documento de tres maneras diferentes: importadas, incorporadas o en línea (inline).
- **CSS importado:**
  - Las **reglas CSS** son importadas cuando están declaradas en un documento fuera del **HTML**.
  - La **hoja de estilo CSS** es entonces un archivo con extensión **.css** separado del **archivo HTML** con extensión **.html**.
  - Este tipo de procedimiento sirve para aplicarles **reglas CSS** a varias páginas de un **sitio Web**.
  - El archivo **.css** de la hoja de estilo externa deberá vincularse al **documento HTML**, dentro de la etiqueta **<head>**.
  - **<link rel="stylesheet" href="css/estilo.css" type="text/css" media="screen"/>**

## Conectando CSS y documentos (2)

- CSS incorporado (interno):

- Es posible incorporar **reglas CSS** declarándolas en el propio **documento HTML**.
- Con una hoja de estilo incorporada o interna, podemos cambiar la apariencia solamente de un documento, aquel en el cual la hoja de estilo está incorporada.
- Las **reglas CSS** internas deberán declararse en la sección **<head>** del documento con la etiqueta de estilo **<style>**.
- El navegador interpretará las **reglas CSS** en la propia página y formateará el documento de acuerdo con ellas.
- Para ocultar el contenido de las etiquetas en navegadores que no soportan estilos, utilizamos comentarios **HTML**: **<!--** y **-->**.

## Conectando CSS y documentos (3)

- CSS inline:
  - Son **reglas CSS** declaradas dentro de la etiqueta del elemento **HTML** (aplica reglas a un solo elemento).
  - Al utilizar ese procedimiento se pierden muchas de las ventajas de **CSS**, pues el contenido se mezcla con la presentación.
  - Este método debería utilizarse sólo **excepcionalmente** para aplicar un estilo a una única aparición de un elemento en la **página Web**.
  - No obstante cuando enviamos correos en formato **HTML** esta es la única opción para dar estilo a su contenido.
    - Los clientes de correo y las diferentes alternativas de Webmail por cuestiones de seguridad omiten las declaraciones dentro del elemento **<head>** y no permiten importar hojas de estilo o scripts externos

# Propiedades CSS - Font

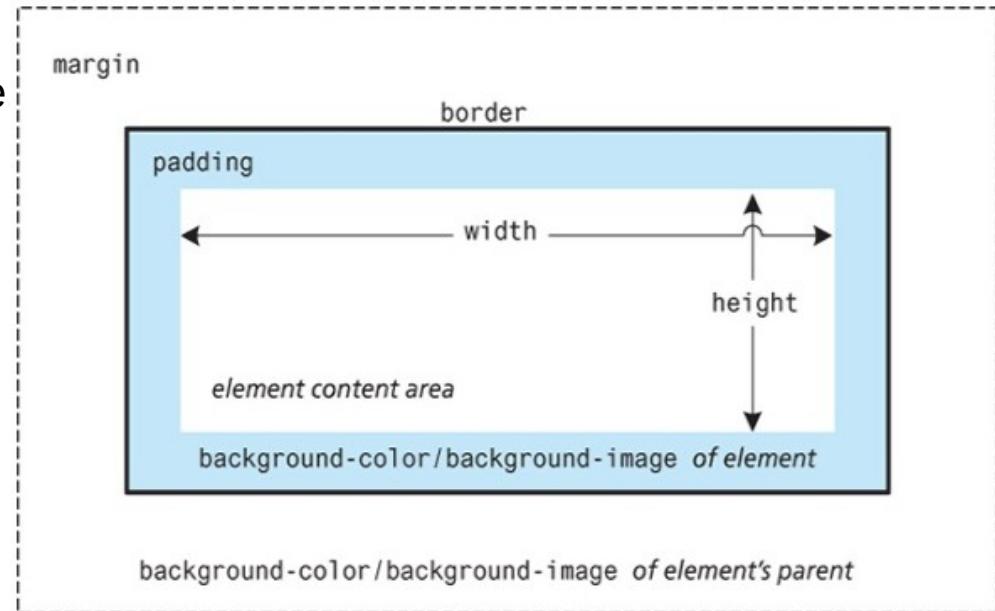
- Al utilizar las propiedades para fuentes en las **reglas CSS**, definimos las características de los caracteres que van a constituir los textos dentro de los **elementos HTML**.
- Las propiedades básicas para las fuentes son:
  - **color**: color de la fuente.
  - **font-family**: tipo de fuente.
  - **font-size**: tamaño de la fuente.
  - **font-style**: estilo de la fuente.
  - **font-variant**: fuentes mayúsculas de menor altura.
  - **font-weight**: oscurecimiento de la fuente (negrita).
  - **font**: manera abreviada para todas las propiedades.
  - **@font-face**: nos permite descargar fuentes y utilizarlas como si se trataran de safe-fonts.
  - **overflow-wrap**: (previamente **word-wrap**) para definir qué hacer cuando se corte una palabra.

# Propiedades CSS - Text

- A continuación se detallan las propiedades existentes para **textos HTML**:
  - **letter-spacing**: espacio entre letras.
  - **word-spacing**: espacio entre palabras.
  - **text-align**: alineamiento del texto.
  - **text-decoration**: decoración del texto.
  - **text-indent**: margen del texto.
  - **text-transform**: forma de las letras.
  - **direction**: dirección del texto.
  - **white-space**: cómo el navegador trata los espacios en blanco.
  - **text-shadow**: aplica una sombra al texto.
  - **text-overflow**: para definir qué hacer cuando el texto no entra en el contenedor.

# CSS – Box Model

- El **Box Model** de **CSS** esencialmente es un recuadro que envuelve cada **elemento HTML**.
- Consiste de márgenes, bordes, paddings (o espaciados interiores) y el contenido propiamente dicho del elemento.
  - **Contenido:** donde aparecen texto o imágenes.
  - **Padding:** espacio alrededor del contenido.
  - **Borde:** envuelve al padding y contenido.
  - **Margin:** espaciado exterior fuera del borde



# Propiedades CSS - Margin

- La propiedad **margin** permite el control del espacio alrededor de los **elementos HTML**. Son válidos los valores negativos para **margin**, con el objetivo de sobreponer elementos.
- A continuación se detallan las propiedades existentes para **margin**:
  - **margin-top**: define el margen superior.
  - **margin-right**: define el margen derecho.
  - **margin-bottom**: define el margen inferior.
  - **margin-left**: define el margen izquierdo.
  - **margin**: manera abreviada para todos los márgenes.

# Propiedades CSS - Border

- Las propiedades existentes para modificar los bordes son:
  - **border-width**: ancho del borde.
  - **border-style**: estilo del borde.
  - **border-color**: color del borde.
  - **border-top**: propiedades del borde superior.
  - **border-right**: propiedades del borde derecho.
  - **border-bottom**: propiedades del borde inferior.
  - **border-left**: propiedades del borde izquierdo.
  - **border**: manera abreviada para los cuatro bordes.
  - **border-radius**: para redondear las esquinas de un elemento.
  - **border-image**: para establecer un patrón de imágenes como borde de un elemento.

# Propiedades CSS - Padding

- La propiedad **padding** define un valor para los espacios entre el contenido y los bordes de los **elementos HTML**.
- Propiedades:
  - **padding-top**: define el espacio superior.
  - **padding-right**: define el espacio derecho.
  - **padding-bottom**: define el espacio inferior.
  - **padding-left**: define el espacio izquierdo.
  - **padding**: manera abreviada para todos los espacios.

# Propiedades CSS - Fondo

- Las propiedades **background** establecen el estilo del fondo de cualquier elemento y son:
  - **background-color**: color de fondo.
  - **background-image**: imagen de fondo.
  - **background-repeat**: manera como la imagen de fondo se posiciona.
  - **background-attachment**: si la imagen de fondo funciona o no con la pantalla.
  - **background-position**: cómo y dónde la imagen de fondo se posiciona.
  - **background**: manera abreviada para todas las propiedades.
  - **background-origin**: para definir la posición de la imagen de fondo.
  - **background-size**: para definir el tamaño de la imagen de fondo.

# Propiedades CSS - Listas

- Las propiedades **CSS** para modificar el estilo de las **listas HTML** son:
  - **list-style-image**: imagen como marcador de la lista.
  - **list-style-position**: donde el marcador de la lista se posiciona.
  - **list-style-type**: tipo del marcador de la lista.
  - **list-style**: manera abreviada para todas las propiedades.

- Las **unidades de medida de CSS** permiten definir las dimensiones de los elementos.
  - El formato para declarar el **valor** de una **unidad de medida CSS** es un número con o sin punto decimal inmediatamente seguido por una unidad identificadora (**px**, **em**, **deg**).
  - Por ejemplo:
    - **width: 70px;**
    - **height: 100%;**
    - **font-size: 1em;**
  - La unidad identificadora es opcional cuando el valor es **0**. Ejemplo: **border: 0px** es igual a **border: 0px**.
  - Algunas propiedades **CSS** permiten que sean declarados valores negativos para unidades de medida.
  - Son dos los tipos de unidad de medida de longitud **CSS**: relativa y absoluta.

## Medidas en CSS (2) – Unidades Relativas

- La **unidad relativa** es aquella que se toma usando como referencia otra medida.
- Las **reglas CSS** que usan unidades de longitud relativas son más apropiadas para efectuar ajustes de uso al transitar entre diferentes tipos de soportes como por ejemplo, de una pantalla de una PC de escritorio, de un dispositivo móvil o una impresora.
- Para cada unidad de medida el valor se toma con relación a:
  - **em**: el ancho de la letra M de la fuente (**font-size**) heredada. Por defecto 16px.
  - **rem**: ídem anterior pero no dependiente del elemento padre.
  - **ex**: la altura de la letra x (equis) de la fuente heredada.
  - **%**: una medida previamente definida.

## Medidas en CSS (3) – Unidades Absolutas

- Es aquella que no se refiere a ninguna otra unidad ni es heredada. Son unidades de medida de longitud indicadas para ser usadas cuando los medios de exhibición son perfectamente conocidos.
  - **pt**: puntos – 1/72 pulgadas.
  - **pc**: pica - 12 points o 1/6 pulgadas.
  - **mm**: milímetro – 1/10 cm.
  - **cm**: centímetro – 1/100 m.
  - **in**: pulgada – 2,54 cm.
  - **px**: (según el estándar moderno) aunque históricamente el **px** dependía del dispositivo, hoy en día **CSS** considera al **px** como una unidad absoluta relativa al sistema de referencia visual ( $1\text{px} \approx 1/96\text{in}$ ).

# Pseudelementos y Pseudoclases

- Los **pseudelementos** nos permiten hacer referencia a una parte del elemento y así poder asignarle una característica particular mediante las reglas correspondientes a **CSS**.
- Puede ser utilizado para:
  - Dar estilo a la primera letra o línea de un elemento de texto.
  - Insertar contenido antes o después de un elemento.
- Sintaxis:

```
selector::pseudelemento{  
    propiedad: valor;  
}
```

- Son pseudelementos: **::after**, **::before**, **::first-letter**, **::first-line**, **::selection**.

## Pseudelementos y Pseudoclases (2)

- Las **pseudoclases** nos brindan la posibilidad de referirnos a un estado particular del elemento.
- Puede ser utilizado para:
  - Dar estilo a un elemento cuando el usuario pasa el cursor por sobre encima de él.
  - Dar un estilo diferente a los links que aún no han sido accedidos.
- Sintaxis:

```
selector:pseudoclas{  
    propiedad: valor;  
}
```

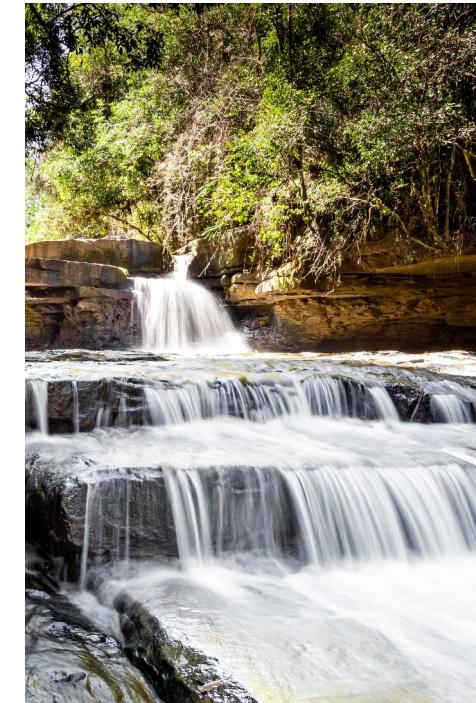
- Algunas pseudoclases son: **:active**, **:hover**, **:link**, **:visited**, **:checked**, **:first-child**, **:last-child**, **:focus**.

## Pseudelementos y Pseudoclases (3)

- Los efectos en los links son posibles a través de declaraciones de reglas de estilo para las pseudoclases del elemento `<a>` de **HTML**.
- Los links poseen cuatro **pseudoclases**:
  - **a:link**: define el estilo del link en el estado inicial.
  - **a:visited**: define el estilo del link visitado.
  - **a:hover**: define el estilo del link cuando se pasa el ratón sobre él.
  - **a:active**: define el estilo del link activo (aquel en el que ya se ha hecho click).

# Efecto cascada

- El uso de **reglas CSS** para controlar la apariencia de un **documento HTML** puede traer como resultado reglas distintas, aplicables a un mismo elemento en el documento.
- Al tener más de una regla atribuida a un mismo elemento, aumentamos la posibilidad de tener un **conflicto entre reglas**.
- La preocupación en relación al **efecto cascada** es saber cuál regla se aplicará cuando surja un conflicto de estilos especificados.
  - Por ejemplo, dos reglas de estilo para un mismo encabezado HTML, donde la primera determina que el encabezado tendrá una fuente de color amarillo y la otra que la fuente será de color naranja.
  - Aquí entra en juego el **efecto cascada**, que no es más que el establecimiento de una propiedad para la aplicación de la regla CSS al elemento.



## Efecto cascada (2)

- Para determinar la prioridad son considerados diversos factores, entre ellos el tipo de regla, el posicionamiento de la regla en el todo, el posicionamiento de la regla en el conjunto de reglas y la especificidad de la regla de estilo.
- El efecto cascada da prioridad a los siguientes elementos (en orden ascendente):
  1. Declaraciones del usuario con **!important**.
  2. Declaraciones del diseñador con **!important**.
  3. Regla CSS del diseñador.
    - i. Estilo inline (dentro de un elemento HTML).
    - ii. Estilo incorporado (definido en la sección head del documento).
    - iii. Estilo externo (importado).
  4. Regla CSS del usuario.
  5. Regla CSS estándar del navegador del usuario.

# Variables

- Con lo visto hasta el momento, si aplicamos estilos con **CSS** a más de una página notaremos que nuestro código tiene duplicación de estilos.
- Esto es necesario porque queremos que el sitio tenga un estilo único pero también implica un problema... **¿Cómo mantenemos un diseño consistente sin copiar y pegar?**
  - Si bien la aplicación de estilos en cascada ayuda, hay muchos valores que no son heredados como por ejemplo fondos, espaciados internos, márgenes, entre otros.
  - Por muchos años la solución a esta problemática era utilizar un **pre-procesador CSS**.
- Sin embargo, las versiones más actuales de **CSS** soportan **variables CSS** (también conocidos como valores personalizados).
- Para definir una variable, dentro de un selector de pseudoclase **:root** al principio del **documento CSS** incluimos la definición de la variable anteponiendo doble guion medio.
- Luego usamos el valor en todas las propiedades que queramos usando la función **CSS var()**.

```
:root {  
    --color-principal: #530C9B;  
}  
.texto { color: var(--color-principal); }
```

## Bibliografía

- **Libro:** Luis Matos. “*Entrenamiento Práctico de CSS*”. Digerati. 2007
- **Libro:** Philip Crowder, David A. Crowder. “*Creating Web Sites Bible 3rd Ed*”. Wiley Publishing Inc. 2008.
- **Web:** W3schools. <http://www.w3schools.com>.
- **Libro:** Matt West. “*HTML5 Foundations*”. Ed. Wiley. 2013
- **Libro:** Randy Connolly, Ricardo Hoar. “*Fundamentals of Web Development, Global Edition*”. Pearson. 2015.



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

Unidad 3 – CSS

Disposición de elementos

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

- **Objetivos**
  - Conectar documentos HTML con archivos CSS.
  - Comprender el funcionamiento de reglas, atributos y valores CSS.
  - Implementar diseños complejos a través de atributos de disposición.
- **Temas a desarrollar:**
  - **CSS:**
    - Definición.
    - Enlace de documentos HTML con hojas de estilo.
  - **Sintaxis de CSS:**
    - Selectores, Clases, Pseudoclases y Pseudoelementos.
    - Atributos y valores.
  - **Disposición de elementos:**
    - floats, position, display, box model, Flex Box y CSS Grid.



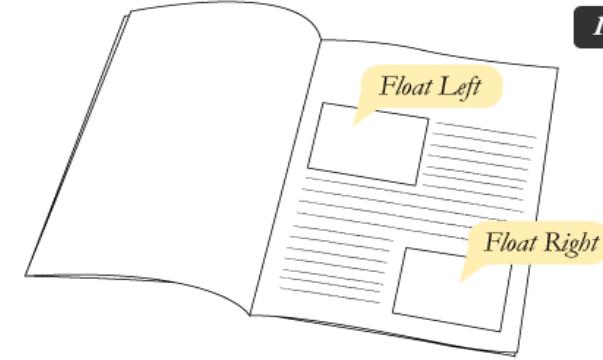
# Flujo de documentos Web

- Si no utilizamos **CSS** en nuestros documentos **HTML** los elementos se disponen:
  - En orden de arriba hacia abajo y de izquierda a derecha.
  - Los elementos de nivel de bloque toman el ancho completo del documento y se siguen unos a otros con su contenido en línea dentro de ellos.
- Mediante las propiedades **CSS margin** y **padding** podemos alterar la visualización y el espaciado de los elementos. Sin embargo, estas propiedades no permiten alterar el posicionamiento de los mismos.
- Existen varias formas de definir posicionamiento y establecer relación entre los elementos:
  - Forzar los elementos para que se comporten como si fueran de un tipo diferente (nivel de bloque o nivel de línea).
  - Quitar los elementos completamente fuera del flujo normal del documento.
  - Establecer que un ítem se ubique a un lado de su contenedor y que los demás lo rodeen.
  - Utilizar **flexbox** o **grid layout**.

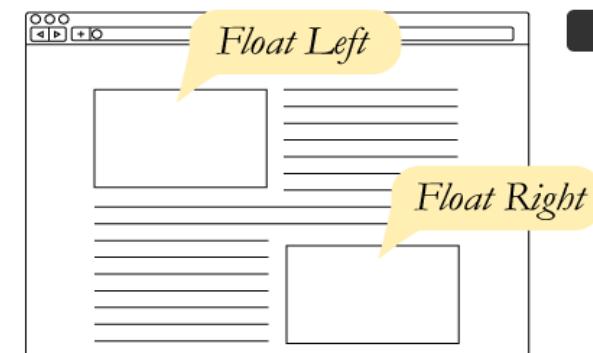
# CSS – Propiedad float y clear

- La propiedad **float** se utiliza para definir la posición de los elementos dentro de las páginas Web.
- A través de la propiedad podemos establecer que el texto fluya alrededor de las imágenes como sucede en el diseño de las publicaciones impresas.
- A diferencia de los elementos con posicionamiento absoluto, los elementos “flotantes” **continúan siendo parte del flujo de la página Web**.
- Los elementos con posicionamiento absoluto son quitados del flujo del documento y no son afectados ni afectan otros elementos.

Print Layout

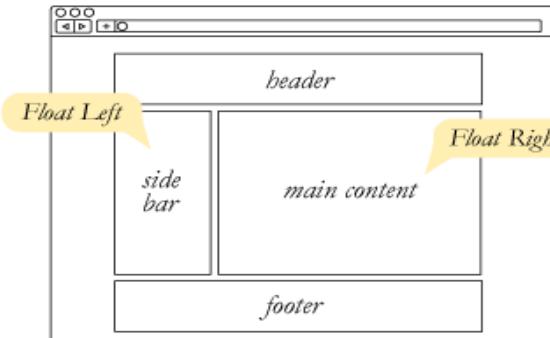


Web Layout

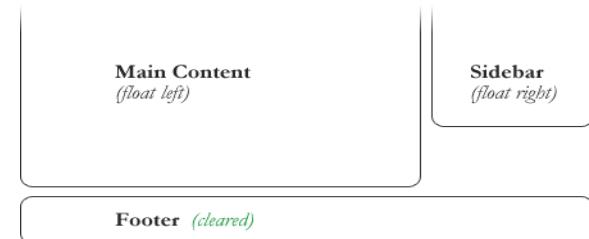


# CSS – Propiedad float y clear (2)

- Si bien el propósito para el cual fue diseñada la propiedad **float** es la que mostramos anteriormente, es utilizada también para establecer la disposición de los elementos en los documentos Web.



- Una propiedad estrechamente vinculada a **float** es **clear**. Un elemento que tiene la propiedad **clear** no se ubicará de manera adyacente al elemento flotado, sino que se ubicará debajo de este.



# CSS – Propiedad display

- La propiedad **display** es clave en lo que se refiere a posicionamiento y disposición de elementos en la Web.
- Algunos de sus valores posibles:
  - **block**: el elemento al que se aplica toma nivel de bloque (como la presentación por defecto de **<div>**, **<p>**, **<h1>**, etc.).
  - **inline**: establece que el elemento tiene nivel de línea (como la presentación por defecto de **<span>**, **<a>**, etc.).
  - **inline-block**: establece que el elemento si bien tiene nivel de bloque se comporta como si fuese un elemento de nivel de línea (como las imágenes).
  - **flex**: el elemento tiene nivel de bloque y es contenedor **flex**.
  - **grid**: el elemento tiene nivel de bloque y es contenedor **grid**.
  - **none**: quita el elemento de la presentación de manera completa. No se dibuja nada en pantalla. A diferencia de la propiedad **visibility** ni siquiera permanece el espacio donde estaba ubicado el elemento.

# CSS – Propiedad position

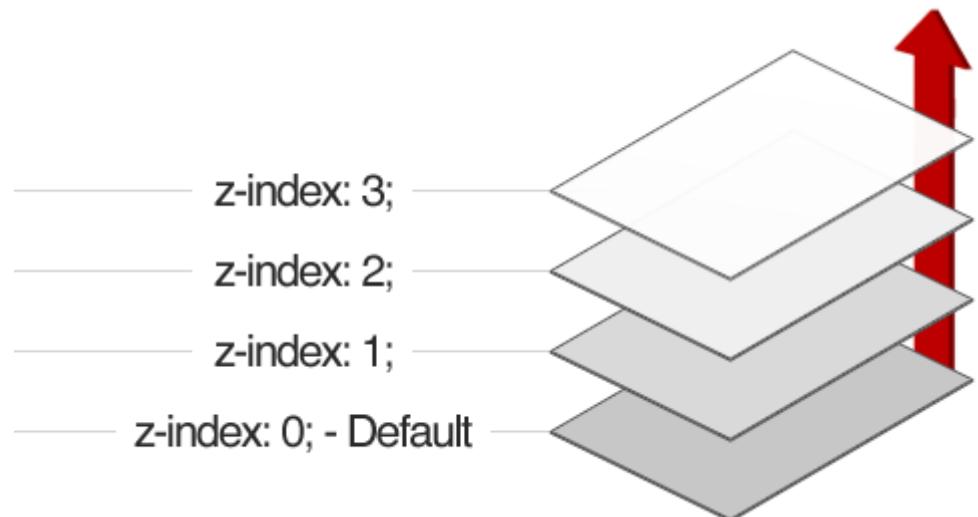
- La propiedad **position** nos permite establecer el tipo de posicionamiento que se utilizará para un elemento. Los valores posibles de esta propiedad son: **static**, **relative**, **absolute** y **fixed**.
- Si bien el elemento puede ser ubicado dentro del documento utilizando las propiedades offset (desplazamiento): **top**, **right**, **bottom** y **left**, esto solo funcionará dependiendo del tipo de posicionamiento utilizado.
- Posicionamiento estático (**static**) (valor por defecto):
  - Todos los **elementos HTML** utilizan por defecto posicionamiento estático.
  - Este tipo de posicionamiento ignora por las propiedades de offset.
  - Un elemento que utiliza posicionamiento **static** no es ubicado de manera especial. Siempre se ubica de acuerdo al flujo normal de la página.
- Posicionamiento relativo (**relative**):
  - Cuando **position** tiene este valor el elemento es ubicado de manera relativa a su ubicación normal.
  - Utilizando **top**, **right**, **bottom** y **left** ajustaremos el elemento fuera de su posición normal.

- Posicionamiento absoluto (**absolute**):
  - Un elemento con este tipo de posicionamiento es ubicado en relación a su contenedor más cercano.
  - Si no hay antecesores se utilizará el cuerpo del documento y el elemento se moverá cuando el usuario haga scroll.
- Posicionamiento fijo (**fixed**):
  - Se trata de una variante del posicionamiento absoluto.
  - La posición inicial del elemento se calcula de la misma manera que con la el valor **absolute**. Una vez posicionado, el elemento permanecerá fijo.
  - Aún cuando hagamos scroll el elemento siempre se mostrará en la misma posición de la pantalla.

# CSS – Propiedad z-index

- Cuando los elementos son posicionados se pueden superponer.
- La propiedad **z-index** establece el orden en que deben ser apilados (quien va en el frente, detrás, al fondo).
- Esta propiedad puede tener valores positivos o negativos.

```
div.contenedor{  
    position: absolute;  
    left: 0;  
    top: 0;  
    z-index: -1;  
}
```



- **Flexible boxes o flexbox (cajas flexibles)** es un módulo de **CSS3** que tiene como objetivo proporcionar una forma más eficiente de diseñar, alinear y distribuir el espacio entre los elementos en un contenedor, incluso cuando su tamaño es desconocido y/o dinámico.
- **Flexbox** le brinda al contenedor la capacidad de modificar el ancho/alto (y orden) de los elementos que contiene para llenar mejor el espacio disponible (adaptándose a diferentes tamaños de pantalla).
  - Un contenedor flexible expande los elementos para llenar el espacio libre disponible o los reduce para evitar el desbordamiento.
- Para muchas aplicaciones el modelo de **flexbox** es mejor que el modelo de bloques porque no utiliza la propiedad **float** ni hace que los márgenes del contenedor flexible interfieran con los márgenes de sus contenidos.

# CSS – Flexbox (3)

← float



float

## Fall in Calgary

Nunc nec fermentum dolor. Duis at iaculis tarpis. Sed rutrum elit ac egestas dapibus. Duis nec consequunt enim.

Mauris porta arcu id magna adipiscing lacinia at congue lacinia. Vivamus blandit quam quis tincidunt egestas. Etiam posuere lectus sed sapien malesuada molestie.

Phasellus vel felis purus. Aliquam consequat pellentesque dui, non mollis erat dictum sit amet. Curabitur non quam dictum, consectetur arcu in, vehicula justo.

margin-left

= image size + margin-right

```
.media-image {  
    float: left;  
    margin-right: 10px;  
}  
.media-body {  
    margin-left: 160px;  
}
```

```
<div class="media">
```

```
    
```

```
    <div class="media-body">
```

```
        <h2>Fall in Calgary</h2>
```

```
        <p>Nunc nec fermentum dolor...</p>
```

```
        <p>Mauris porta arcu id...</p>
```

```
        <p>Phasellus vel felis purus...</p>
```

```
    </div>
```

```
</div>
```

Prior to flexbox, one would create such a layout within a container using floats plus margins. The problem with this approach is that margins needed to be in pixels and had to exactly match image size. If image size changed (or you wanted same kind of style elsewhere), you had to modify the style.

```
.media {
```

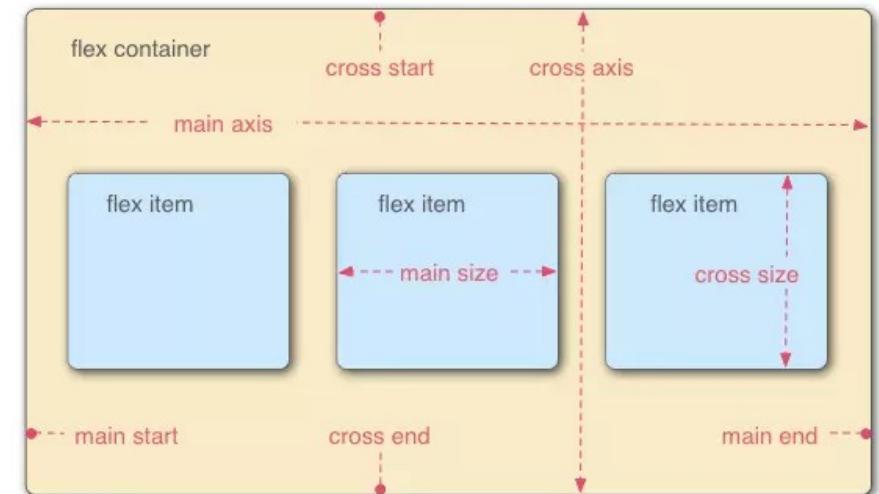
```
    display: flex;  
    align-items: flex-start;  
}
```

```
.media-image {  
    margin-right: 1em;  
}
```

Using flexbox, we now have a much more generalized (and thus reusable) style.

# CSS – Flexbox (2)

- **Flexbox** consiste de:
  - **Flex-containers**: se establece un elemento como contenedor flex cuando la propiedad **display** toma el valor **flex**.
  - **Flex-items**: puede haber uno o más **flex-items** dentro de un **contenedor flex**.
- Todo lo que está fuera de un contenedor **flex** o dentro de un **flex-item** se muestra como lo hacía usualmente.
- **Flexbox** define la manera en que los items son ubicados dentro de un **contenedor flex**.
- Los **flex-items** son mostrados en líneas flex (por defecto una por contenedor).



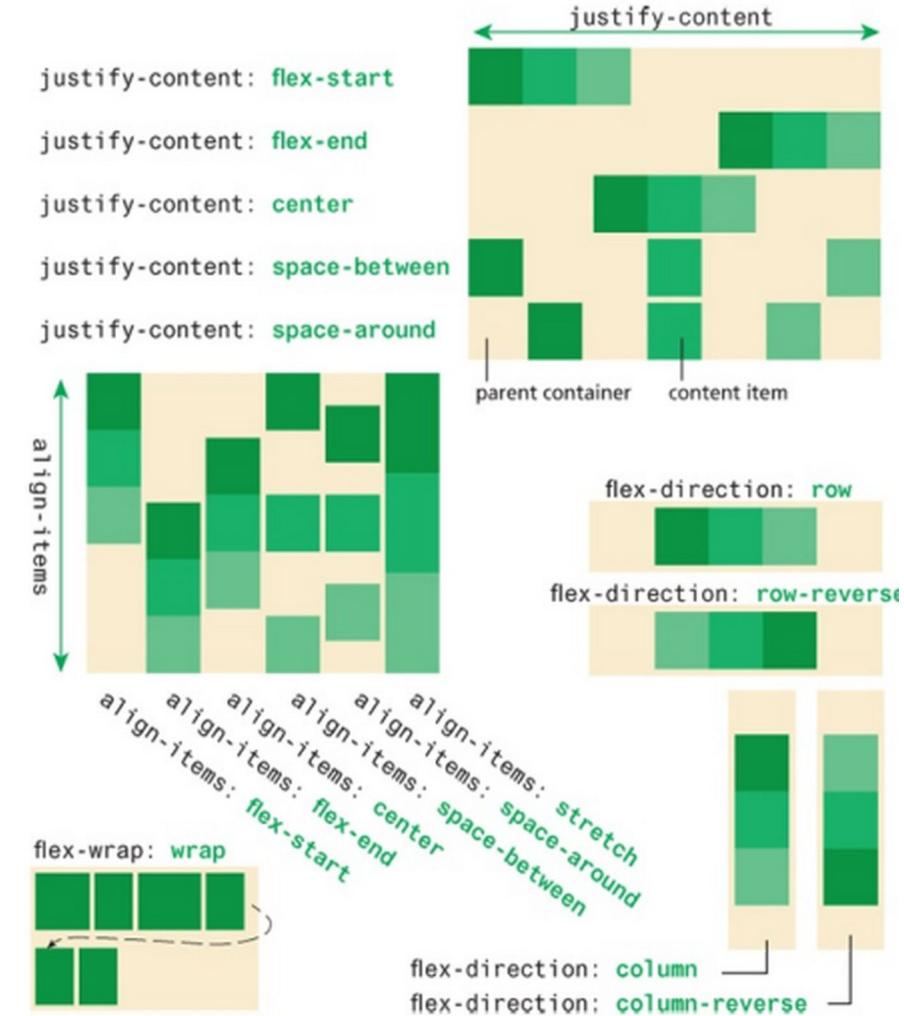
# CSS Flexbox (4) – Propiedades Flex

- Contenedor Flex:

- justify-content
- flex-direction
- flex-wrap
- flex-flow (shorthand de flex-wrap y flex-flow)
- align-items
- align-content
- gap, row-gap, column-gap

- Flex items:

- order
- flex-grow
- flex-shrink
- flex-basis
- flex (shorthand de los anteriores)
- align-self



- **CSS Grid Layout (o simplemente Grid)** es un sistema de diseño bidimensional basado en cuadrículas que con unas pocas líneas de **CSS** nos permite implementar diseños complejos.
- Para trabajar con **CSS Grid Layout** tenemos que definir:
  - Un contenedor grid utilizando **display: grid**.
  - Establecer los tamaños de columna y fila con **grid-template-columns** y **grid-template-rows**.
  - Colocar los elementos secundarios en la cuadrícula con **grid-column** y **grid-row**.
- Algunas consideraciones:
  - En los grid items no aplican las propiedades/valores: **float**, **display: inline-block**, **display: table-cell**, **vertical-align** y **column-\***
  - Excepto Firefox ningún navegador soporta subgrid.

## Bibliografía

- **Libro:** Luis Matos. “*Entrenamiento Práctico de CSS.*”. Ed. Digerati. 2007
- **Libro:** Philip Crowder, “*David A. Crowder. Creating Web Sites Bible*”. 3rd Ed. Wiley Publishing Inc. 2008.
- **Web:** W3schools. URL: <http://www.w3schools.com>
- **Libro:** Matt West. “*HTML5 Foundations*”. Ed. Wiley. 2013.
- **Libro:** Ian Lunn. “*CSS3 Foundations*”. Ed. Wiley. 2013.
- **Libro:** Randy Connolly, Ricardo Hoar. “*Fundamentals of Web Development, Global Edition*”. Ed. Pearson. 2015.

## Bibliografía (2)

- **Web:** DesarrolloWeb. “*Etiquetas semánticas del HTML5*”. URL: <http://www.desarrolloweb.com/articulos/etiquetas-semanticas-html5.html>
- **Web:** Mozilla Developer Network. “*Secciones y líneas generales de un documento HTML5*”. URL: [https://developer.mozilla.org/es/docs/Sections\\_and\\_Outlines\\_of\\_an\\_HTML5\\_document](https://developer.mozilla.org/es/docs/Sections_and_Outlines_of_an_HTML5_document)
- **Web:** LibrosWeb. “*Propiedad position (Referencia de CSS 2.1)*”. URL: <http://librosweb.es/referencia/css/position.html>.
- **Web:** Mozilla Developer Network. “*Usando Cajas flexibles*”. URL: [https://developer.mozilla.org/es/docs/Web/Guide/CSS/Cajas\\_flexibles](https://developer.mozilla.org/es/docs/Web/Guide/CSS/Cajas_flexibles).
- **Web:** CSS Tricks. “*A Complete Guide to Flexbox*”. URL: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>.
- **Web:** CSS Tricks. “*A complete Guide to CSS Grid*”. URL: <https://css-tricks.com/snippets/css/complete-guide-grid/>.



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

Unidad 5 – JavaScript

DOM y Eventos

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

# Objetivos de la clase

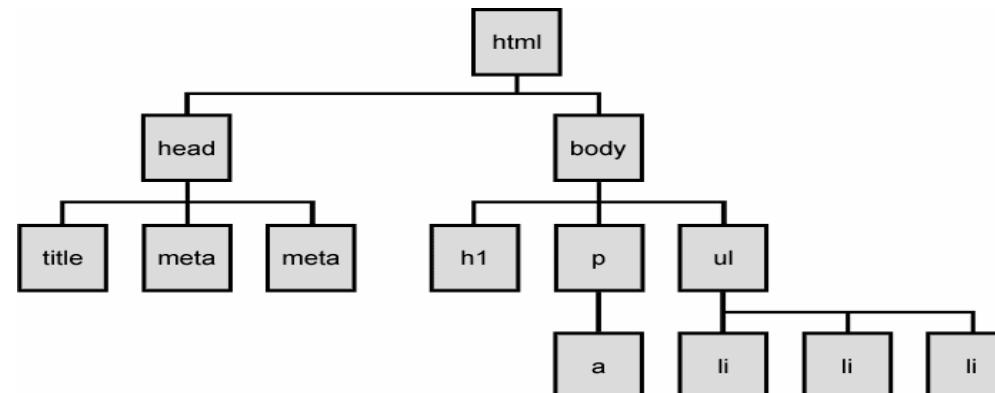
- **Objetivos**
  - Conocer el lenguaje de programación y sus características fundamentales.
  - Establecer asociaciones entre elementos y manejadores de eventos.
  - Acceder y modificar contenido y atributos a través de DOM.
- **Temas a desarrollar:**
  - Sintaxis y construcciones básicas: valores, variables y literales. Operaciones y sentencias.
  - Funciones y Manejo de Eventos.
  - DOM: Modelo de objetos del documento de hipertexto. Acceso y manipulación de elementos de un documento HTML.

# Objetivos de la clase

- **JavaScript** hoy día se usa en una variedad de contextos diferentes: aplicaciones de escritorio, aplicaciones móviles, videojuegos, etc. pero por lejos el más común es el **navegador web**.
- En este contexto, **JavaScript** necesita una forma de interactuar con el **documento HTML** en el que está contenido.
  - Como tal, debe haber alguna forma de acceder mediante programación a los elementos y atributos dentro del **HTML**.
- Esto se logra a través de una interfaz de programación de aplicaciones (API) llamada **Document Object Model (DOM)**.

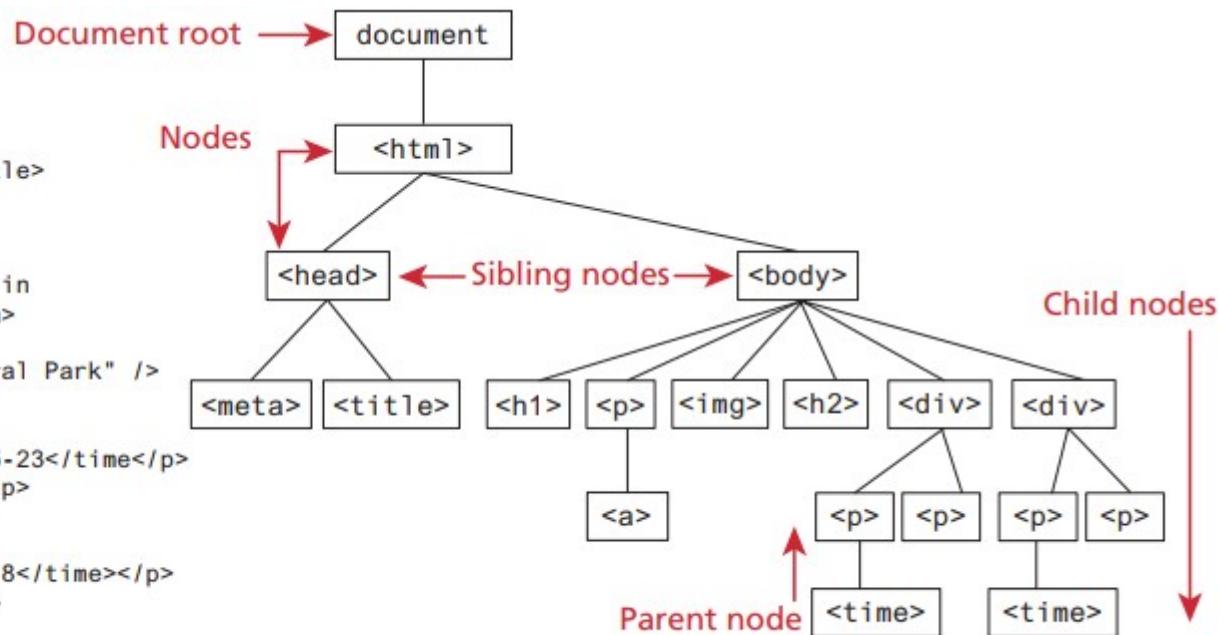
# DOM: Document Object Model

- DOM es el acrónimo de Document Object Model es el modelo (o representación) de los objetos en un documento HTML.
- Según el W3C, DOM es una plataforma e interfaz independiente de cualquier lenguaje de programación que permite a programas y scripts acceder dinámicamente y actualizar el contenido, estructura y estilo de los documentos.
- Ya conocimos el DOM pero con otro nombre, la estructura de árbol de los documentos HTML se llama formalmente Árbol DOM. La raíz es elemento que se encuentra más arriba en la jerarquía y se conoce con el nombre de Document Root.



# DOM: Document Object Model (2)

```
<html>
<head>
    <meta charset="utf-8">
    <title>Share Your Travels</title>
</head>
<body>
    <h1>Share Your Travels</h1>
    <p>Photo of Conservatory Pond in
        <a href="#">Central Park</a>
    </p>
    
    <h2>Reviews</h2>
    <div id="latestComment">
        <p>Ricardo on <time>2021-05-23</time></p>
        <p>Easy on the HDR buddy.</p>
    </div>
    <div>
        <p>Susan on <time>2021-11-18</time></p>
        <p>I love Central Park.</p>
    </div>
</body>
</html>
```



# Nodos

- El **árbol DOM** se organiza en **nodos** que son el tipo de dato general para los objetos en **DOM**. Tienen atributos y algunos nodos pueden contener otros.
  - Si el **DOM** es un árbol, cada **nodo** puede considerarse una rama.
- Hay diferentes tipos de nodos, los más comunes son:
  - Nodos de elementos
  - Nodos de texto.
- Todos los **nodos del DOM** comparten un conjunto común de propiedades y métodos que nos permiten recuperar información sobre el nodo, manipular sus propiedades e incluso crear contenido nuevo.
- Algunas de estas propiedades están disponibles para todos los nodos otras solo para el elemento específicos.

# Propiedades fundamentales de objetos nodo

- Si a un nodo del árbol **DOM** le aplicamos el operador punto “.” podemos acceder a las siguientes propiedades:
  - **childNodes**: un **NodeList** de los nodos hijo del nodo.
  - **firstChild**: Primer hijo del nodo.
  - **lastChild**: Último hijo del nodo.
  - **nextSibling**: siguiente hermano del nodo.
  - **nodeName**: nombre del nodo.
  - **nodeType**: tipo del nodo.
  - **nodeValue**: valor del nodo.
  - **parentNode**: padre del nodo.
  - **previousSibling**: anterior hermano del nodo.
  - **textContent**: representa el contenido del nodo quitando los tags.

## NodeList

- El DOM también define un objeto especializado llamado **NodeList** que representa una colección de nodos
- Funciona de manera muy similar a un **array** JavaScript pero no tiene las mismas propiedades y métodos que un **array**
  - **NodeList** y **Array** heredan de diferentes prototipos.
- A diferencia de los **arrays**, cuando el **DOM** es modificado, agregando, quitando o reordenados nodos, la colección **automáticamente se actualiza** para reflejar esos cambios.

# El objeto document

- El objeto documento **DOM** es el objeto **JavaScript** raíz de la representación del **documento HTML** completo.
- Contiene algunas propiedades y métodos que utilizaremos extensivamente en nuestros desarrollos y es accesible a través de la referencia **document**.
- Las propiedades del objeto **document** nos permite acceder o modificar información de la página:
  - Algunos de estas propiedades son de solo lectura, pero otros son modificables.
  - Como cualquier otro objeto de **JavaScript**, puede acceder a sus propiedades utilizando el símbolo punto “.” o la notación de corchetes:
    - `let a = document.URL; // devuelve la URL del documento actual.`
    - `let b = document["inputEncoding"]; // devuelve la codificación del documento. Por ejemplo: ISO-8859-1`
- Además de estas propiedades, existen varios métodos esenciales como por ejemplo el método **document.write()** que vimos durante la Introducción a **JavaScript**.
- Para familiarizarnos mejor con este objeto, agruparemos los métodos en estas tres categorías:
  - Métodos de selección.
  - Métodos de manipulación de parentezco.
  - Método de evento.

# Métodos de selección

- Los métodos **DOM** más importantes son aquellos que le permiten seleccionar uno o más elementos del documento.
- Detalle de métodos:
  - **getElementById("id")**: es quizás el más utilizado de los métodos de selección. Devuelve el nodo de un solo elemento cuyo atributo **id** coincide con el id pasado.
  - **getElementsByClassName("nombre")**: Devuelve una **lista de nodos** de elementos cuyo **nombre de clase** coincide con el nombre pasado.
  - **getElementsByTagName("nombre")**: Devuelve una **lista de nodos** de elementos cuyo **nombre de tag** coincide con el nombre pasado.
  - **querySelector("selector")**: Devuelve el **primer nodo** de elemento que coincide con el **selector CSS** pasado.
  - **querySelectorAll("selector")**: Devuelve una **lista de nodos** de elementos que hacer coincidir el **selector de CSS** pasado. Devuelve un **NodeList** estático.
- **getElementsByTagName()** y **getElementsByClassName()** devuelven una **NodeList** (en WebKit) o una **HTMLCollection** (en FireFox). Ambos tipos son casi idénticos.

# Objetos de tipo Element Node

- El tipo de objeto devuelto por los métodos `getElementById()` y `querySelector()` descritos en la sección anterior es un objeto **Element Node**. Esto representa un **elemento HTML** en la jerarquía, contenido entre las etiquetas de apertura `<>` y cierre `</>` para este elemento.
- Todo los nodos de elemento tienen las propiedades:
  - **classList**: Una lista de solo lectura de clases **CSS** asignadas a este elemento. Esta lista tiene una variedad de métodos auxiliares para manipular esta lista.
  - **className**: El valor actual para el atributo de clase de este elemento **HTML**.
  - **id**: El valor actual para el id de este elemento.
  - **innerHTML**: Representa todo el contenido (texto y etiquetas) del elemento.
  - **style**: El atributo de estilo de un elemento. Esto devuelve un objeto **CSSStyleDeclaration** que contiene subpropiedades que corresponden a las diversas propiedades **CSS**.
  - **tagName**: El nombre de la etiqueta para el elemento.
- Otras propiedades importantes:
  - Para **input** y **textarea value**.
  - Para **a**, **input** y **textarea name**.
  - Para **img**, **iframe** y **script src**
  - Para **a href**

# Modificando el DOM

- Dado que la mayoría de las propiedades que vimos anteriormente son de lectura y escritura, podemos cambiar sus valores programáticamente y modificar la forma en que se visualiza nuestro documento.
- El uso de `innerHTML` para agregar elementos está generalmente desaconsejado por cuestiones de **seguridad** (prevenir Cross Site Scripting), **performance** y porque son strings conteniendo HTML son **muy propensos a generar HTML mal formado**.
  - Un mejor enfoque es entonces aplicar Manipulación DOM.
- Métodos de manipulación DOM
  - Cada nodo DOM tiene varias propiedades y métodos que permiten acceder a otros nodos con los que tiene una relación de parentezco. Sin embargo, este mecanismo puede no resultar confiable en todos los casos.
- Algunos de los métodos más comunes son:
  - `appendChild`: agrega un nuevo nodo al final del nodo actual.
  - `createElement`: crea un nodo de elemento HTML.
  - `createTextNode`: crea un nodo de texto.
  - `removeChild`: elimina un hijo del nodo actual.
  - `replaceChild`: Reemplaza un nodo hijo con un hijo diferente.

# Atributos de estilo

- El atributo **class**:

- Las clases **CSS** se visualizan a la hora de programar en **Javascript** de la siguiente manera:

```
/*En CSS*/  
  
.rojo { color: red; }  
  
// En JavaScript  
  
elemento.className = "rojo";
```

- El atributo **style**:

- Las propiedades de estilo pueden ser accedidas y modificadas utilizando el objeto **style**.
  - **text.style.color = "red";**

# Funciones callback

- Dado que las funciones de **JavaScript** son objetos completos, se pasar una función como argumento a otra función.
- La función que recibe el argumento de la función puede llamar a la función pasada.
- Se dice que dicha función pasada es una **función callback (callback function)** y es una parte esencial de la programación de **JavaScript** del mundo real.
- Una **callback function** es simplemente una función que se pasa a otra función.

```
function funcion1(argFunction){  
    ...  
    argFunction();  
}  
  
const funcion2 = function () { alert("Hola! Soy una callback  
function!"); }  
  
funcion1(funcion2);
```

# Eventos

- Los **eventos** son una parte esencial de casi toda la programación **JavaScript** del mundo real.
- El procesamiento de eventos de **JavaScript** se sigan cuatro pasos:
- Dos en tiempo de programación y dos en ejecución.
  - 1) Comienza por la definición de un **controlador de eventos (event handler)**. Se trata simplemente de una **función de callback**.
  - 2) Luego, ese controlador se registra con un evento específico para un elemento específico.
  - 3) El evento específico se desencadena, generalmente por alguna acción del usuario,
  - 4) El controlador finalmente se ejecuta.



# Implementación de Event Handler

- El registro de un controlador de eventos requiere pasar una función callback al método **addEventListener()** de un objeto de tipo **Node Element**.
- A continuación, tres alternativas para hacerlo:

```
const btn = document.getElementById("btn");
btn.addEventListener("click", function () {
    alert("Utilizando una función anónima");
});

document.querySelector("#btn").addEventListener("click", function (){
    alert("Un enfoque diferente pero mismo resultado");
});

document.querySelector("#btn").addEventListener("click", () => {
    alert("Utilizando arrow functions");
});
```

- Es importante recordar que los objetos de tipo Node tienen el método **addEventListener()** pero lo **NodeList** no.

# El objeto event

- Los manejadores de las API estándar reciben por parámetro un **objeto event**.
- Este objeto contiene información del evento así como también método para controlar los resultados y el ciclo de vida de un evento.
- Propiedades:
  - Si bien cada tipo de evento tiene propiedades diferentes, algunas comunes a todos son: **type**, **target**, **currentTarget**, **eventPhase**, **timeStamp**.
- Métodos:
  - Los métodos permiten controlar el ciclo de vida de un evento.
  - Algunos ejemplos son: **preventDefault()**, **stopPropagation()**.

# Propagación de eventos

- Cuando un evento se dispara en un elemento que tiene elementos antepasados, el evento se **propaga** a esos antepasados.
- Hay dos enfoques distintos o fases de propagación: hay una **fase de captura (capture phase)** y una **fase de burbujeo (bubbling phase)**:
  - **En la fase de captura de eventos:**
    - El navegador verifica el ancestro más externo (elemento `<html>`) para ver si ese elemento tiene un controlador de eventos registrado para el evento desencadenado y, de ser así, se ejecuta.
    - Luego procede al siguiente ancestro y realiza los mismos pasos; esto continúa hasta que alcanza el elemento que desencadenó el evento (es decir, el objetivo del evento).
  - **En la fase de burbujeo del evento ocurre lo contrario:**
    - El navegador comprueba si el elemento que desencadenó el evento tiene un controlador de eventos registrado para ese evento y, de ser así, se ejecuta. Luego continúa hacia afuera hasta que alcanza el antepasado más externo.
- Por defecto todos **los eventos se registran en la fase de burbujeo**.
- Podemos cambiar este comportamiento agregando el argumento **capture** al método **addEventListener()**.
- Podemos detener la propagación utilizando el método **stopPropagation()** del argumento **event**.

# Tipos de eventos

- Hay muchos tipos diferentes de eventos que se pueden activar en el navegador.
- El evento más obvio es el evento de **click**, pero **JavaScript** y el **DOM** admiten varios otros.
- En realidad, existen varias clases de eventos, con varios tipos de eventos dentro de cada clase especificada por el **W3C**.
- Algunos de los tipos de eventos más utilizados son los eventos del mouse, los eventos del teclado, los eventos táctiles, los eventos de formulario:
  - **Eventos del mouse:** **click**, **dblclick**, **mousedown**, **mouseup**, **mouseover**, **mouseout**, **mousemove**.
  - **Eventos del teclado:** **keydown**, **keyup**.
  - **Eventos de formulario:** **blur**, **change**, **focus**, **reset**, **select**, **submit**.

# Bibliografía

- **Libro:** Randy Connolly, Ricardo Hoar. ***“Fundamentals of Web Development, Global Edition”***. 3era Edition. Ed. Pearson. 2022.
- **Libro:** Lenny Burdette. ***“The JavaScript Pocket Guide”***. 1st Ed. Peachpit Press. 2010.
- **Libro:** John Pollock. ***“JavaScript: A Beginner’s Guide”***. Ed. McGrall Hill. 2013.



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

Unidad 5 – JavaScript

Arrays, Prototipos, Clases y Módulos

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

# Objetivos de la clase

- **Objetivos**
  - Conocer el lenguaje de programación y sus características fundamentales.
  - Establecer asociaciones entre elementos y manejadores de eventos.
  - Acceder y modificar contenido y atributos a través de DOM.
- **Temas a desarrollar:**
  - Sintaxis y construcciones básicas: valores, variables y literales. Operaciones y sentencias.
  - Funciones y Manejo de Eventos.
  - DOM: Modelo de objetos del documento de hipertexto. Acceso y manipulación de elementos de un documento HTML.

# Introducción

- Ya conocemos las características fundamentales del JavaScript.
- En la clase anterior vimos como aplicar esos conocimientos básicos en los navegadores Web. Esto exigió que conozcamos como:
  - Manipular DOM y
  - Gestionar eventos con JavaScript.
- A continuación:
  - Veremos programación orientada a objetos en JavaScript.
  - Exploraremos nuevas características que aumentan nuestra productividad.

# Métodos de Array

- Sabemos que los **arrays** son un tipo especial de **objetos** en **JavaScript** que tienen varios atributos y métodos. En clases anteriores vimos como crear e iterar arrays.
- Es momento de revisar otros métodos poderosos que, por lo general, toman como parámetro ***una función que es invocada por cada elemento del array.***
  - **forEach**: vista en presentaciones anteriores, nos brinda una manera alternativa de iterar un array.
  - **find**: permite encontrar el primer elemento que cumple la condición pasada por parámetro.
  - **filter**: devuelve un array formado por los elementos que cumplen con la condición pasada por parámetro.

## Métodos de Array (2)

- **map**: opera de manera similar a los anteriores excepto que crea un **nuevo array** del mismo tamaño pero cuyos valores han sido **transformados** por la función pasada por parámetro.
- **reduce**: es utilizada para **reducir** el array a un **único valor**. La función pasada por parámetro puede que recibir 4 parámetros 2 de los cuales son obligatorios:
  - **Total**: el valor que vamos acumulando (obligatorio)
  - **Valor Actual**: el elemento actual del array que debe ser procesado (obligatorio).
  - **Índice actual**: Índice del elemento actual.
  - **Array**: array al que pertenece el elemento actual.
- **sort**:
  - Sin parámetros puede ordenar arrays unidimensionales de tipos primitivos.
  - Si necesitamos ordenar objetos entonces debemos pasar por parámetro una función que retorne **-1**, **0** ó **1**.

# Prototipos, Clases y Módulos

- En presentaciones anteriores vimos que podemos **definir objetos de manera literal** y a través de **funciones constructoras**. Sin embargo, este enfoque es **ineficiente** si tenemos que tratar con gran cantidad de objetos. Existen otras alternativas más performantes:

## Prototipos

- Son un mecanismo de sintaxis esencial de **JavaScript** utilizado para que el lenguaje sea más orientado a objetos.
  - Cada objeto función tiene la propiedad **prototype** (inicialmente vacía).
- Lo que hace a **prototype** poderosa es que se define una vez para todas las instancias de objetos cuando se crea un objeto a través de la palabra reservada **new** desde una función constructora.

```
function Persona(nombre, apellido) {  
    this.nombre = nombre; this.apellido = apellido;  
}  
  
Persona.prototype.saludar = () {  
    return `Hola ${ this.apellido }, ${ this.nombre }!`;  
}
```

# Clases

- Los **prototipos** son utilizados para extender funcionalidades de objetos existentes. Esto, en lenguajes de programación orientados a objetos tradicionales, como Java o C# se llama **herencia**.
- **ES6** agregó clases a **JavaScript** pero en realidad son una forma más amigable de crear **prototipos**.

```
class Persona {  
    constructor (nombre, apellido) {  
        this.nombre = nombre;  
        this.apellido = apellido;  
    }  
    saludar() {  
        return `Hola ${ this.apellido }, ${ this.nombre }!`;  
    }  
}
```

# Herencia en JavaScript

- Una de las características clave de los lenguajes de programación orientados a objetos es lograr la **reutilización de código** a través de **heredar** propiedades y métodos de una clase existente.
- Las clases **JavaScript** proveen una prestación similar a través de la palabra reservada **extends**.

```
class Alumno extends Persona {  
    constructor (nombre, apellido) {  
        super(nombre, apellido);  
        this.materiasAprobadas = 4;  
    }  
    saludar() {  
        return `Hola soy ${this.apellido}, ${this.nombre}.  
        Aprobé ${this.materiasAprobadas} materias!`;  
    }  
}
```

# Módulos

- Con lo visto hasta el momento en medida que crezca en **complejidad** nuestra aplicación tendremos más archivos con extensión **.js** con **múltiples variables de alcance global**.
- Lo expuesto anteriormente, puede traer aparejado **conflictos de nombres**. Es decir, variables o funciones son **sobrescritas** por otras definidas en un archivo diferente.
- Para solucionar esto **ES6** nos permite definir módulos:
  - Es simplemente un **archivo** que contiene código **JavaScript** cuyos literales tienen el **alcance restringido a ese archivo**. Digamos que las variables en un módulo son **privadas a ese módulo**.
- Si queremos que otros módulos accedan a las variables, funciones, objetos y clases que definimos en un módulo, debemos hacer lo siguiente:
  - Exponer las variables, funciones, objetos o clases explícitamente utilizando la palabra reservada **export**. Ejemplo: **export function saludar () { ... }**
  - En otro módulo (no un script que no sea un módulo) importar el recurso que necesitamos a través de la palabra reservada **import**. Ejemplos:
    - **import { saludar } from “./funciones.js”;**
    - **import \* as funciones from “./funciones.js”;**

# Bibliografía

- **Libro:** Randy Connolly, Ricardo Hoar. ***“Fundamentals of Web Development, Global Edition”***. 3era Edition. Ed. Pearson. 2022.
- **Libro:** Lenny Burdette. ***“The JavaScript Pocket Guide”***. 1st Ed. Peachpit Press. 2010.
- **Libro:** John Pollock. ***“JavaScript: A Beginner’s Guide”***. Ed. McGrall Hill. 2013.



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

Unidad 5 – JavaScript

Introducción al lenguaje

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

# Objetivos de la clase

- **Objetivos**
  - Conocer el lenguaje de programación y sus características fundamentales.
  - Establecer asociaciones entre elementos y manejadores de eventos.
  - Acceder y modificar contenido y atributos a través de DOM.
- **Temas a desarrollar:**
  - Sintaxis y construcciones básicas: valores, variables y literales. Operaciones y sentencias.
  - Funciones y Manejo de Eventos.
  - DOM: Modelo de objetos del documento de hipertexto. Acceso y manipulación de elementos de un documento HTML.



- JavaScript fue desarrollado en 1995 por Brendan Eich para funcionar en el navegador Netscape.
- Su nombre original era Mocha, luego fue renombrado como LiveScript para tomar finalmente el nombre por el que hoy es reconocido.
  - A pesar de estar su nombre formado por la palabra Java, las semejanzas con este lenguaje de programación son superficiales.
- En 1997, ECMA (European Computer Manufacturers Association) elaboró un estándar basado en el desarrollo de Netscape con el objetivo que otros navegadores lo implementen.
- El estándar se llama ECMAScript y es una de las implementaciones más conocidas es JavaScript.
- Ha pasado por varias versiones. La versión 5 se publicó en 2009. En 2015 la versión ECMAScript 6. En 2016 ES7. Las versiones siguientes llevan el número de año de publicación.
- Características: <https://kangax.github.io/compat-table/es2016plus/>

# JavaScript - Características

- JavaScript es un lenguaje de scripting, tipado dinámico y orientado a objetos.
- El entorno de ejecución puede ser una página web por tanto los resultados se visualizan en un navegador web.
- Podemos utilizar JavaScript para programar respuestas a eventos del mouse y teclado, crear y animar elementos en la página, comunicarse con servidores, entre otras operaciones.
- Algunas de sus características son:
  - Basado en prototipos: es orientado a objetos pero los objetos son creados a partir de otros objetos.
  - Lado del Cliente: los scripts corren en el software de visualización del usuario y no en el servidor donde el sitio está alojado.

## JavaScript – Características (2)

- **Lenguaje de scripting:** ellos programas escritos en **JavaScript** son interpretados y ejecutados por el navegador u otro intérprete.
- **Requerimientos mínimos Hardware-Software:** Podemos programar en **JavaScript** utilizando un editor de textos y un navegador web.
- **Fácil puesta en funcionamiento:** necesitamos subir nuestros códigos **JavaScript** al servidor e incluir el tag **<script>** en nuestras páginas web.
- **La comunidad:** Podemos encontrar gran cantidad de videos, manuales, tutoriales y ejemplos de código **JavaScript** disponibles gratis en la web.

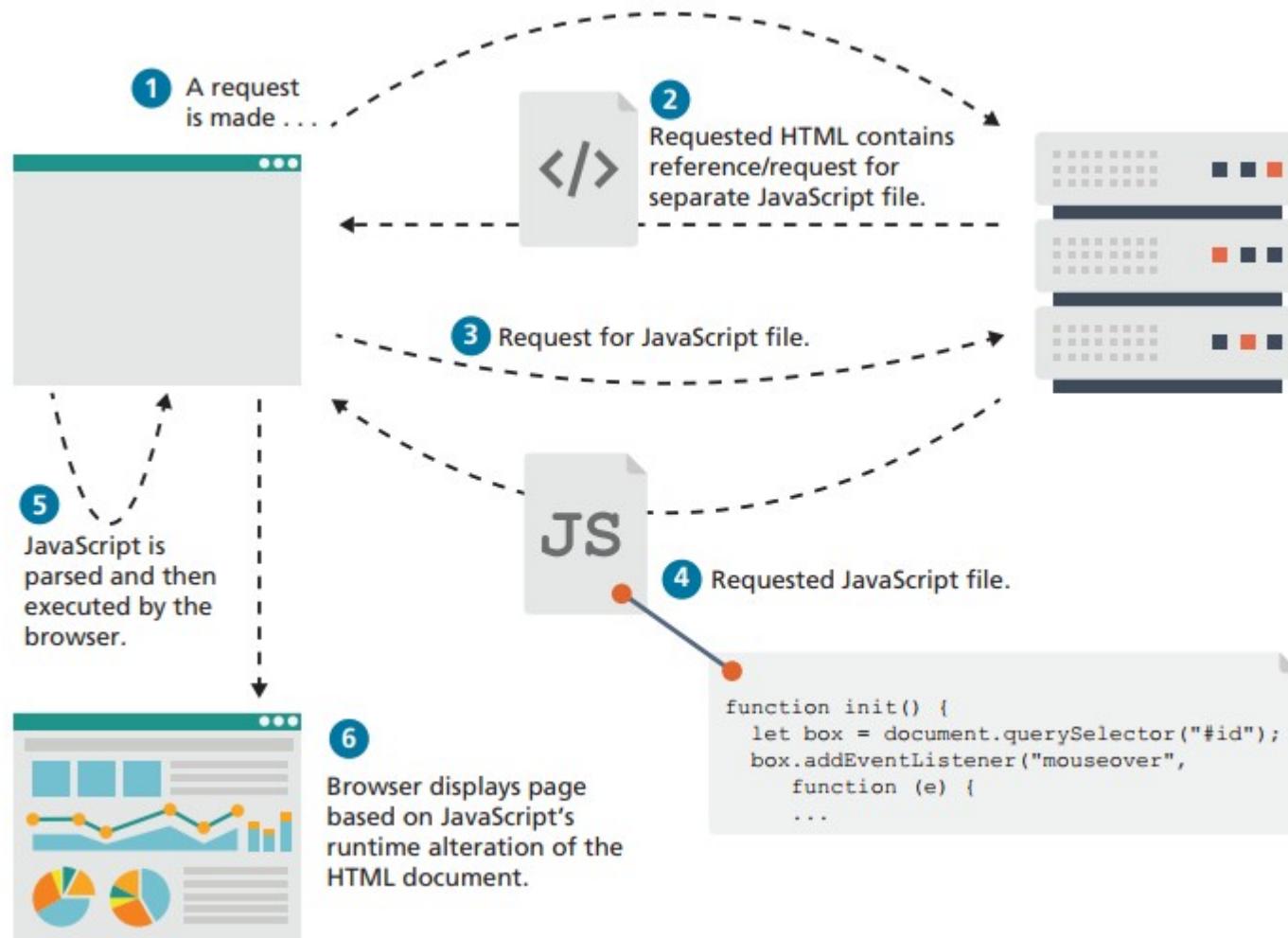


# JavaScript – Primer script

```
<!DOCTYPE html>

<html lang="es">
  <head>
    <title>Mi Primer JavaScript</title>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <script>
      document.write("Hola Mundo!!!");
    </script>
  </body>
</html>
```

# Programación del lado del cliente



# Sintaxis del lenguaje - Características Generales

- **Case sensitive:** debemos prestar atención a la hora de nombrar variables, funciones y objetos incorporados.
- **Comentarios:** De línea: `//` y de bloque: `/**/`
- **Punto y Coma:** es el delimitador de sentencias en **JavaScript**.
  - Técnicamente su inclusión es opcional pero sólo porque la mayoría de los intérpretes lo agregan automáticamente. *No incluirlos puede dar lugar a extraños efectos secundarios.*
- **Espacios en Blanco:** el lenguaje ignora el carácter de espaciado, las tabulaciones y las líneas en blanco, su uso tiene como finalidad aumentar la legibilidad del programa resultante.
  - En la producción de scripts a gran escala, se suelen quitar los espacios en blanco innecesarios para que los mismos puedan ser descargados más rápidamente.
  - Ver <http://javascript-minifier.com/>

- **Variables:** sus nombres pueden ser cualquier identificador. Es decir, una palabra formada por letras, números, \$ ó \_ y que no comienza con un número.

- Para declarar una variable en **JavaScript** escribimos la palabra reservada **let** seguido del nombre de la variable y opcionalmente podemos inicializarla asignándole un valor.

- Declaración de una variable en **JavaScript**:

```
let miVariable; //Declaración.
```

```
let miVariable = 42; //Declaración y asignación.
```

- Todas las variables declaradas fuera de una función tienen alcance global.

# Variables (2) – Formas de definir variables

- Sin palabra reservada: si no anteponemos al nombre de una variable alguna de las palabras reservadas **var**, **let**, ó **const** se creará una variable global (no importa donde se la defina).
  - **Evitar su uso! Es la forma más fácil de crear un bug!!!.** 🤦
- **var**: Antes de ES6, la palabra clave **var** era la única forma de declarar una variable en JavaScript.
  - **var** crea una variable que tiene un ámbito de función o un ámbito global.
  - Las variables definidas con **var** se elevan a la parte superior de su bloque.
  - En general, **evitar su uso**, excepto por la compatibilidad con versiones anteriores de navegadores más antiguos. Algunos desarrolladores todavía usan **var** para una variable de alcance global. 🤦
- **let**: Crea una variable de ámbito de bloque que se puede reasignar a un valor diferente.  
**Usar cuando se necesite reasignar el valor de una variable.** 👍
- **const**: Crea una variable de ámbito de bloque cuyo valor no se puede reasignar.
  - Esto no crea una variable inmutable (como la palabra clave final en Java). Si se asigna a un objeto o array, puede cambiar los valores de las propiedades o los valores de los elementos.
  - **Usar cuando no necesite reasignar el valor de una variable y desee que el navegador detecte y evite las reasignaciones.** 👍

# Tipos de datos

- JavaScript tiene dos categorías de tipos de datos:
  - Tipos Referencia: generalmente denominados objetos
  - Tipos Primitivos: tipos atómicos o simples que no son objetos.
- Lo que hace que las cosas sean un poco confusas es que el lenguaje permite usar tipos primitivos como si fueran objetos.
  - Esto tiene origen en que los objetos en JavaScript son fundamentales.
  - Casi todo dentro del lenguaje es un objeto, por lo que el lenguaje proporciona formas fáciles de usar primitivas como si fueran objetos.
- Los **tipos primitivos** representan **formas simples de datos**.
- ES2015 define seis tipos primitivos: Boolean, Number, String, Null, Undefined y Symbol.

# Tipos de datos simples

- Los tipos de datos atómicos con los que cuenta el lenguaje son:
- Número (number):
  - Permite almacenar valores enteros y reales. Ejemplo: **571**, **2.998e8** y **66.4**.
  - Existen valores especiales **Infinity**, **-Infinity** y **NaN**.
  - Soporta las operaciones aritméticas: negación, adición, sustracción, multiplicación, división, módulo. Para alterar el orden de evaluación de las operaciones podemos utilizar **( )**. Ejemplos: **-6**, **4 + 3**, **23 - 14**, **6\* 21**, **57 % 10**.
  - Si el número entero es demasiado grande usar **BigInt** (disponible a partir de JS 2020).
- Cadenas de caracteres (string):
  - Se delimitan con **' '** (comillas simples), **""** (comillas dobles) o **``** (doble acento grave).
  - Los strings no pueden ser divididos, multiplicados o sustraídos pero el operador **+** permite concatenarlos.
  - Los strings delimitados **``** (doble acento grave) se conocen con el nombre de **template literals** y permiten strings multilínea y embeber otros valores utilizando el símbolo  **\${ }** . Ejemplo: **`la mitad de 100 is \${100 / 2}`** → la mitad de 100 es 50.

# Tipos de datos Simples (2)

- Booleanos (boolean):
  - Tiene dos posibles valores: **true** y **false**.
  - Los operadores lógicos son: ! (negación), && (and / y / conjunción), || (or, ó, disyunción).
- Valores vacíos:
  - Los valores **null** y **undefined** son utilizados para denotar la ausencia de un valor representativo. Si bien son valores, no brindan ninguna información.
  - Muchas operaciones del lenguaje no tienen como resultado ningún valor representativo y se limitan simplemente a devolver **null** o **undefined** para devolver un valor.
  - La diferencia entre **null** y **undefined** es un accidente en el diseño del lenguaje y la mayoría de las veces no hay que preocuparse por cuál de ellos se trata. Deberíamos considerarlos como intercambiables.
- **Symbol:** Permite obtener valores que no pueden volver a ser creados, es decir, son identificadores únicos e inmutables.

# Coerción de tipos

- Es la conversión de tipos que realiza automáticamente el lenguaje cuando aplicamos un operador a variables o valores de tipos incorrectos.
- Todos los valores tienen asociado una noción de verdad y falsedad. Es decir, pueden ser tratados como **true** o **false** sin realmente ser booleanos.
- **0** (cero), **null**, **undefined** y el string vacío son "**falsos**" y todo los demás "**verdaderos**".
  - **console.log(8 \* null) → 0**
  - **console.log("5" - 1) → 4**
  - **console.log("5" + 1) → 51**
  - **console.log("five" \* 2) → NaN**
  - **console.log(false == 0) → true**
- Expresiones como **0 == false && "" == false** resultan en **true** por la conversión automática de tipos. Por tanto en muchas operaciones de comparación en vez de usar **==** ó **!=** tenemos que usar **====** y **!==**.
- Estos operadores no realizan conversión de tipos para verificar igualdad. Por tanto **"" == false** resulta en **false**.

## Valores, tipos de datos y operadores (4)

- Como **no se trata** de un lenguaje fuertemente tipado podemos asignar a una variable valores de diferentes tipos de datos durante su ciclo de vida.
- El tipo de datos de una variable se determina utilizando el operador **typeof**.

```
typeof miNumero; //Resultado: "number"  
typeof miString; //Resultado: "string"  
typeof miBoolean; //Resultado: "boolean"  
typeof miArray; //Resultado: "object"
```

- Los arrays **no son tipos de datos primitivos** por ello **typeof** nos devuelve como resultado **object**. La forma de determinar si una variable es un **Array** es mediante **instanceof**.

```
miArray instanceof Array; //Resultado: true
```

- Si aplicamos **typeof** a una variable a la que aún no se le ha asignado un valor obtenemos como resultado: **undefined**.

# JavaScript en el navegador - Salidas

- Una de las primeras operaciones que aprendemos cuando estudiamos un nuevo lenguaje mostrar información.
- Si **JavaScript** se ejecuta en el navegador hay varias opciones:
  - **Alertas**: le indica al navegador que muestre una ventana emergente o cuadro de diálogo modal mostrando el string pasado a la función.
    - El usuario no puede hacer ninguna acción hasta que se descarte la ventana emergente.
    - Las alertas generalmente no se utilizan en producción, pero proporcionan una forma rápida de mostrar o recopilar información simple temporalmente.
    - Usar ventanas emergentes puede volverse tedioso rápidamente. El usuario tiene que hacer clic en **Aceptar** para descartar la ventana emergente y, si la usa en un bucle, puede pasar más tiempo haciendo click en **Aceptar** que haciendo una depuración significativa.
    - Para mostrar información podemos usar la función **alert()** y para solicitar el ingreso de datos **prompt()** y **confirm()**.

# JavaScript en el navegador - Salidas

- **Consola:** La mayoría de los sistemas **JavaScript** (browsers y Node.js) incluyen la función **console.log** que escribe los argumentos en algún dispositivo de salida.
  - En los navegadores la salida aparece en la consola de **JavaScript**.
  - La abrimos apretando **F12** o accediendo al menú **Developer Tools** o similar.
  - La expresión **console.log** hace referencia a la función **log** que es mantenida por la variable a la que hace referencia **console**.
  - Algunas alternativas a **console.log** son: **console.info()**, **console.warn()**, **console.error()**, **console.table()**, **console.group()**, **console.groupEnd()**.
- **document.write():** es una forma útil de generar contenido en formato **HTML** desde JavaScript. Este método se usa a menudo para generar elementos **HTML** o para combinar elementos **HTML** con variables **JavaScript**.

# Estructuras de Control

- En JavaScript tenemos las siguientes estructuras de control:
  - Repetitivas:
    - `while`
    - `for / for-in / for-of / forEach()`
    - `break` y `continue`
  - Condicionales:
    - `if / if-else / else`
    - `switch / case`
    - Operador ternario: `( ) ? :`
    - Operador Coalescencia nula (Nullish Coalescing): `??` (JS 2020)
  - Manejo de Excepciones:
    - `try / catch / finally`
    - `throw`

# Funciones

- Las funciones en **JavaScript** son muy flexibles, nos permiten realizar muchas operaciones. Sin embargo, esta flexibilidad suele generar confusiones en lo que respecta a la interacción entre funciones, variables y objetos.
- Declaración:

```
function nombreFuncion(arg1, arg2, argN) {  
    //Sentencias  
}
```

- Los intérpretes de **JavaScript**, antes que cualquier otra cosa, buscan declaraciones de funciones. Por lo que se las puede invocar antes de declararlas.
- Funciones anónimas:
  - No es necesario especificar un nombre para una función.
  - Las funciones que no tienen nombre se denominan expresiones función o **funciones anónimas**.
  - Se puede almacenar una expresión función en una variable o establecerla como el resultado de otra función.

## Funciones (2)

- Funciones anónimas (cont.):

```
(function() { // Sentencias });

(function() { //Se ejecuta de manera inmediata })();
// Esta función devuelve una función anónima

function funcionExterior() {
    return function() { console.log("Función interior"); };
}
```

- Funciones expresión:

```
const nombreFuncion = function(arg1, arg2, argN) { // Sentencias };

nombreFuncion(); //Invocación

nombreFuncion.call(); //Invocación
```

## Funciones (3) – Funciones Flecha o Arrow Functions

- En vez de usar la palabra reservada **function** usamos una flecha (`=>`) se compone por los caracteres `=` (igual) y `>` (signo mayor) no confundir con `>=` (mayor o igual)

```
const power = (base, exponent) => {  
    let result = 1;  
    for (let count = 0; count < exponent; count++){  
        result *= base;  
    }  
    return result;  
}
```

- Después de la lista de parámetros viene la flecha.
- Cuando tenemos un único parámetro podemos omitir los paréntesis.
- Si el cuerpo de la función es una única sentencia podemos omitir las llaves.

```
const square1 = (x) => { return x * x };  
const square2 = x => return x * x;
```

## Funciones (3) - Argumentos

- Los argumentos de las funciones en **JavaScript** son muy flexibles.
- Los parámetros (o argumentos nominados) que aparecen en la **signatura** de la función son más que nada una convención.
- No es necesario proveer un valor para cada argumento.

```
function miFunc(arg0, arg1, arg2) {  
    console.log(arg0);  
    console.log(arg1);  
    console.log(arg2);  
}  
  
miFunc("a", "b");  
"a"  
"b"  
undefined
```

- También se puede proveer más argumentos de los que están definidos en la **signatura** de la función. Los argumentos son almacenados en una variable local llamada **arguments** para su fácil acceso.

## Funciones (4) – Argumentos (cont.)

```
function miFunc(arg0) {  
    console.log(arg0);  
    console.log(arguments[1]);  
    console.log(arguments[2]);  
}
```

```
miFunc("a", "b", "c");  
"a"  
"b"  
"c"
```

- La variables **arguments** es un objeto tipo array. Sin embargo, no se trata realmente de un array.
- Los objetos tipo array **JavaScript** son comúnmente llamados **colecciones**.
- Se accede a los elementos de las **colecciones** a través de índices numéricos (**arguments[0]**) y tienen la propiedad **length** pero no tienen ninguna de las propiedades restantes con las que cuentan los arrays.

# Strings

- **Caracteres de Escape:** Secuencias como: `\'`, `\"`, `\n`, `\t` nos permiten escapar caracteres especiales.
- **Template Strings:** `let nombre = "Nacho"; console.log(`Hola soy ${nombre}`);`
- **Operadores:**
  - Concatenación: `+` y `+=`
  - Comparación: `>`, `<`, `>=`, `<=`
- **Propiedades:** `length` es la única propiedad de los strings.
- **Métodos comunes:**
  - Cambio de mayúsculas: `toUpperCase()`, `toLowerCase()`,
  - Extracción de secciones: `charAt()`, `slice()`, `substring()`,
  - Conversión a array: `split()`.
  - Búsqueda y reemplazo: `index0f()`, `lastIndex()`, `search()`, `match()`, `replace()`.
  - Conversión: `parseInt()`, `parseFloat()`.

# Arrays

- Los arrays **JavaScript** pueden almacenar elementos de cualquier tipo de datos, en cualquier orden.
- **Creación:**

```
let miArray = [1, 2, 3, 4];  
  
let miArrayLlenado = new Array(4); //Nuevo array de cuatro posiciones con valor undefined  
  
console.log(miArrayLlenado); -> [ <4 empty items> ]  
  
console.log(miArrayLlenado[0]); → undefined
```

- **Propiedades:** **length** es la única propiedad de los arrays en **JavaScript**.
- **Recorridos:** podemos utilizar un bucles **for** o el método **forEach()**.
- **Métodos:**

- Adición: **concat()**, **push()**, **unshift()**.
- Remoción: **pop()**, **shift()**.
- Extracción: **slice()**, **splice()**.
- Ordenación: **reverse()**, **sort()**.
- Conversión a String: **toString()**.

# Desestructuración de arrays

- Una característica del lenguaje agregada en **ES6** es la posibilidad de **desestructurar arrays**.
- Con una sintaxis simple permite extraer múltiples escalares desde un array.
- Si tenemos el array: `const league = ["Liverpool", "Man City", "Arsenal", "Chelsea"];`
- Y queremos tener cada uno de esos elementos en una variable. Con lo que sabemos hasta el momento haríamos:
  - `let first = league[0];`
  - `let second = league[1];`
  - `let third = league[2];`
- Lo mismo con desestructuración de arrays lo podemos hacer en una sola línea:
  - `let [first, second, third] = league;`
- Podemos omitir elementos también haciendo lo siguiente:
  - `let [first,,,fourth] = league;`
- Podemos utilizar la sintaxis para llenar un array con el contenido de otro:
  - `let [first, ...everyoneElse] = league;`
- Que es lo mismo que hacer:
  - `let first = league[0];`
  - `const everyoneElse = [league[1], league[2], league[3]];`
- La desestructuración de arrays nos permite intercambiar los valores de dos variables en una única línea:  
`[second, first] = [first, second];`



# Objetos

- En JavaScript un **objeto** es una colección de valores llamados **propiedades**.
- Podemos agregar a un **objeto** la cantidad de **propiedades** que queramos. Pueden tratarse de valores, objetos y otras funciones.
- En JavaScript hay varias formas de definir/crear objetos: objetos literales, usando el constructor **Object** ó funciones constructoras.
- Objetos literales:

```
const persona = {  
    nombre: "Lionel Andrés",  
    apellido: "Messi",  
    goles: 700,  
    altura: 1.70,  
    pieHabilDerecho: false,  
    agregarGol: function() { this.goles++; } };  
    
```

- Cuando la **propiedad** es una **función** nos referimos a ella con el nombre de **método**.

## Objetos (2) – Acceso / Modificación / Eliminación de propiedades

- Usualmente se accede a una propiedad a través del operador . (punto) seguido del nombre de la propiedad. Ejemplo: `persona.apellido`
- También podemos acceder las propiedades utilizando [ ]. Ejemplo: `persona["apellido"]`
- Utilizamos = para asignar un nuevo valor a una propiedad de un objeto. Ejemplo: `persona.edad = 36;`
  - Si la propiedad no existe entonces se creará y se asignará el valor.
- Para quitar o eliminar una propiedad utilizamos el operador `delete`. Ejemplo: `delete persona.pieHabilDerecho;`
- Utilizando la versión `for-in` ó `for-of` del bucle `for` podemos iterar a través de las propiedades de un objeto.
- Podemos saber si un objeto tiene o no una propiedad utilizando el método `hasOwnProperty()`.
- Existen métodos comunes a todos los objetos como por ejemplo `toString()`.

# JSON

- Una variante de la notación literal de objetos llamada es **JSON** o **JavaScript Object Notation** que se utiliza como un formato de intercambio de datos independiente del lenguaje similar al uso de **XML**.
- La principal diferencia entre **JSON** y la notación literal de objetos es que los nombres de las propiedades están entre comillas, como se muestra en el siguiente ejemplo.

```
// Una cadena que luce como un objeto literal  
  
const text = '{ "name1" : "value1",  
    "name2" : "value2",  
    "name3" : "value3"  
}';
```

- Debemos tener en cuenta que esta variable se asigna como un string que contiene una definición de objeto en formato **JSON** (pero sigue siendo solo una cadena). Para convertir esta cadena en un objeto **JavaScript** real, es necesario usar el objeto **JSON** integrado.

```
const anObj = JSON.parse(text); // Esto convierte el JSON en un objeto.  
  
console.log(anObj.name1); // Muestra "value1"  
  
const strAgain = JSON.stringify(anObj); // Esto convierte el objeto a un string que  
// contiene un JSON.
```

# Bibliografía

- **Libro:** Randy Connolly, Ricardo Hoar. **“Fundamentals of Web Development, Global Edition”**. 3era Edition. Ed. Pearson. 2022.
- **Libro:** Marijn Haverbeke. **“Eloquent JavaScript”**. 3Era Edición. No Starch Press. 2018
- **Libro:** Lenny Burdette. **“The JavaScript Pocket Guide”**. 1st Ed. Peachpit Press. 2010.
- **Libro:** John Pollock. **“JavaScript: A Beginner’s Guide”**. Ed. McGrall Hill. 2013.
- **Web:** MDN Web Docs. **“Symbol”**. URL:  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Symbol](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol)
- **Web:** Medium. Latte and Code. **“JavaScript. ¿Conoces el tipo Symbol?”** URL:  
<https://latteandcode.medium.com/javascript-conoces-el-tipo-symbol-197c8338924a>



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

## Unidad 6 – Formularios

Composición y elementos de entrada

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

# Objetivos de la clase

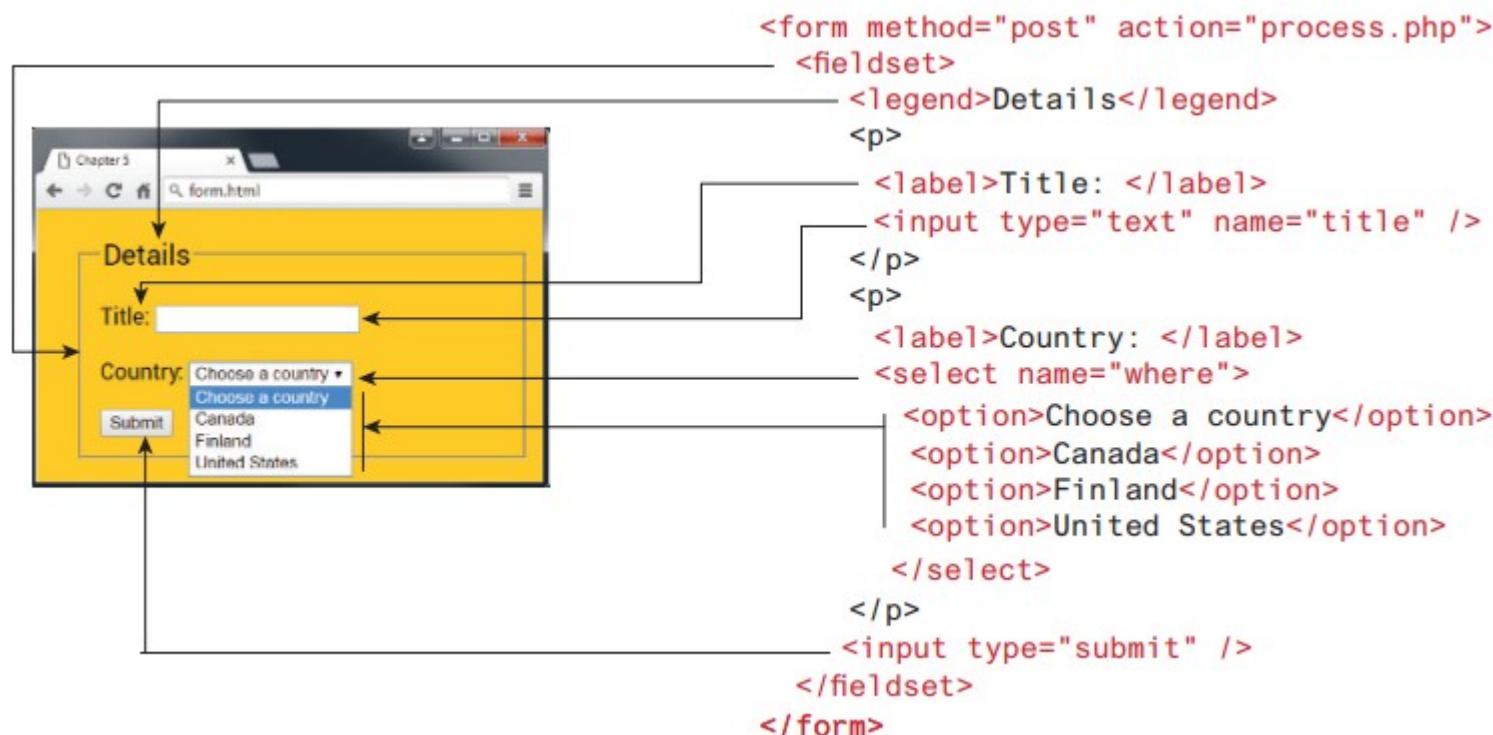
- **Objetivos**
  - Construir formularios.
  - Conocer los formatos de intercambio de información en Web.
  - Conectarse con un servidor web y transferir información entre este y el navegador.
- **Temas a desarrollar:**
  - Formularios. Composición.
  - Elementos de Entrada: cajas de texto, botones, botones radio, áreas de texto.
  - Listas desplegables. Botones de Control.
  - Comunicación asíncrona e intercambio de datos en formato JSON.

# Introducción

- Todos los **elementos HTML** que hemos utilizado tienen por finalidad mostrar contenido al usuario, siendo los enlaces el único mecanismo del que hasta ahora disponemos para establecer una comunicación con el servidor.
  - Los **formularios** constituyen una alternativa para que el usuario interactúe con el servidor web.
- *Un formulario es un conjunto de campos de entrada agrupados todos dentro de un único elemento **form**, cuyo propósito es recopilar la información ingresada por el usuario.*
- A través de un formulario el usuario puede ingresar texto, seleccionar ítems de una lista y apretar botones.
- Los programas corriendo en el servidor tomarán la información desde los formularios HTML y llevarán a cabo alguna acción con ella, como guardarla en una base de datos, interactuar con un web service externo o tomar esas entradas para personalizar código HTML que será mostrado posteriormente.

# Estructura de un formulario

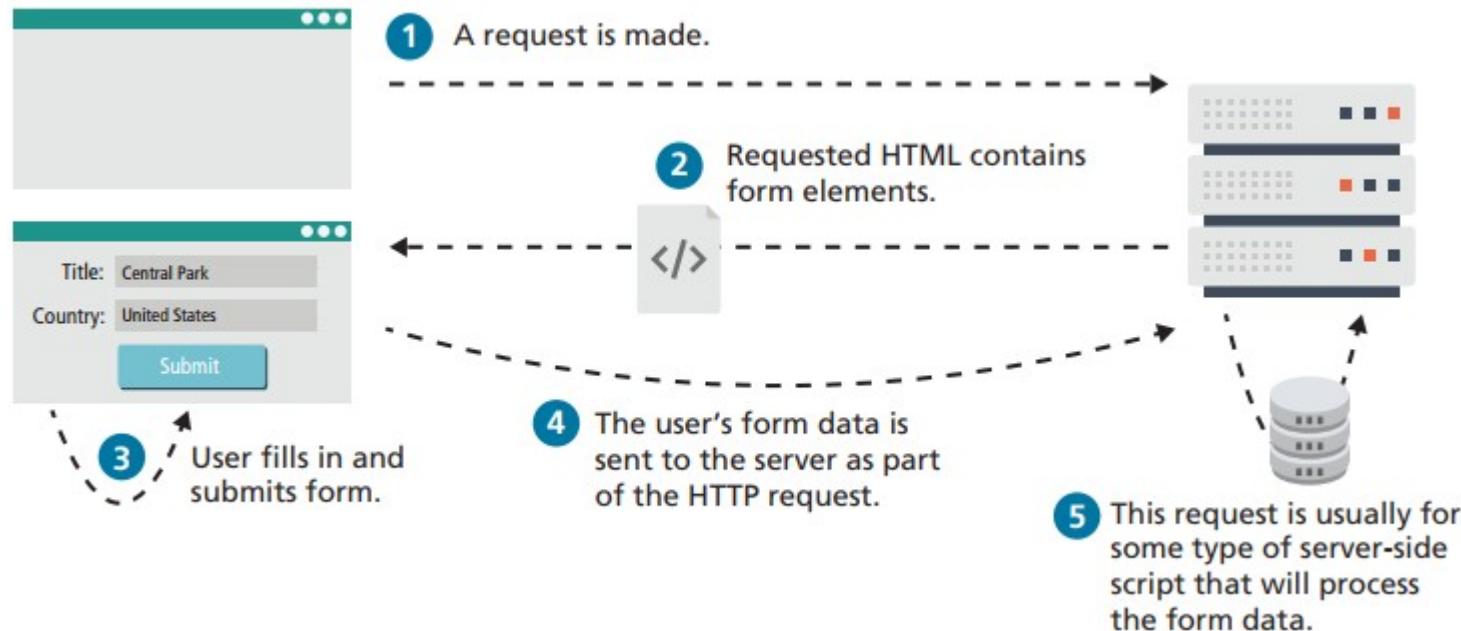
- En **HTML** construimos un formulario combinando **elementos HTML** especiales.
- El elemento **form** será el contenedor de otros elementos que representan diversos controles de entrada, de texto y otros **elementos HTML** (excepto otro elemento **form**).



# ¿Cómo funcionan los formularios?

- Tradicionalmente, si bien los formularios se construyen con **elementos HTML**, un formulario también requiere algún **recurso del lado del servidor** que procese los datos enviados desde el formulario.
- Los pasos que se siguen para este proceso son:
  - Primero, el usuario solicita visualizar una **página HTML** que contiene algún tipo de formulario.
  - Una vez que el usuario completa el formulario, debe existir un mecanismo para enviar los datos del formulario al servidor.
    - Esto se logra a través de un botón **submit** o usando **JavaScript**.
  - Debido a que la interacción entre el navegador y el servidor web se rige por el protocolo HTTP, los datos del formulario deben enviarse al servidor a través de una solicitud HTTP estándar.
  - Esta solicitud suele estar dirigida a algún tipo de programa en el servidor que procesará los datos del formulario de alguna manera; esto podría incluir verificar su validez, almacenarlos en una base de datos o enviarlo por correo electrónico.

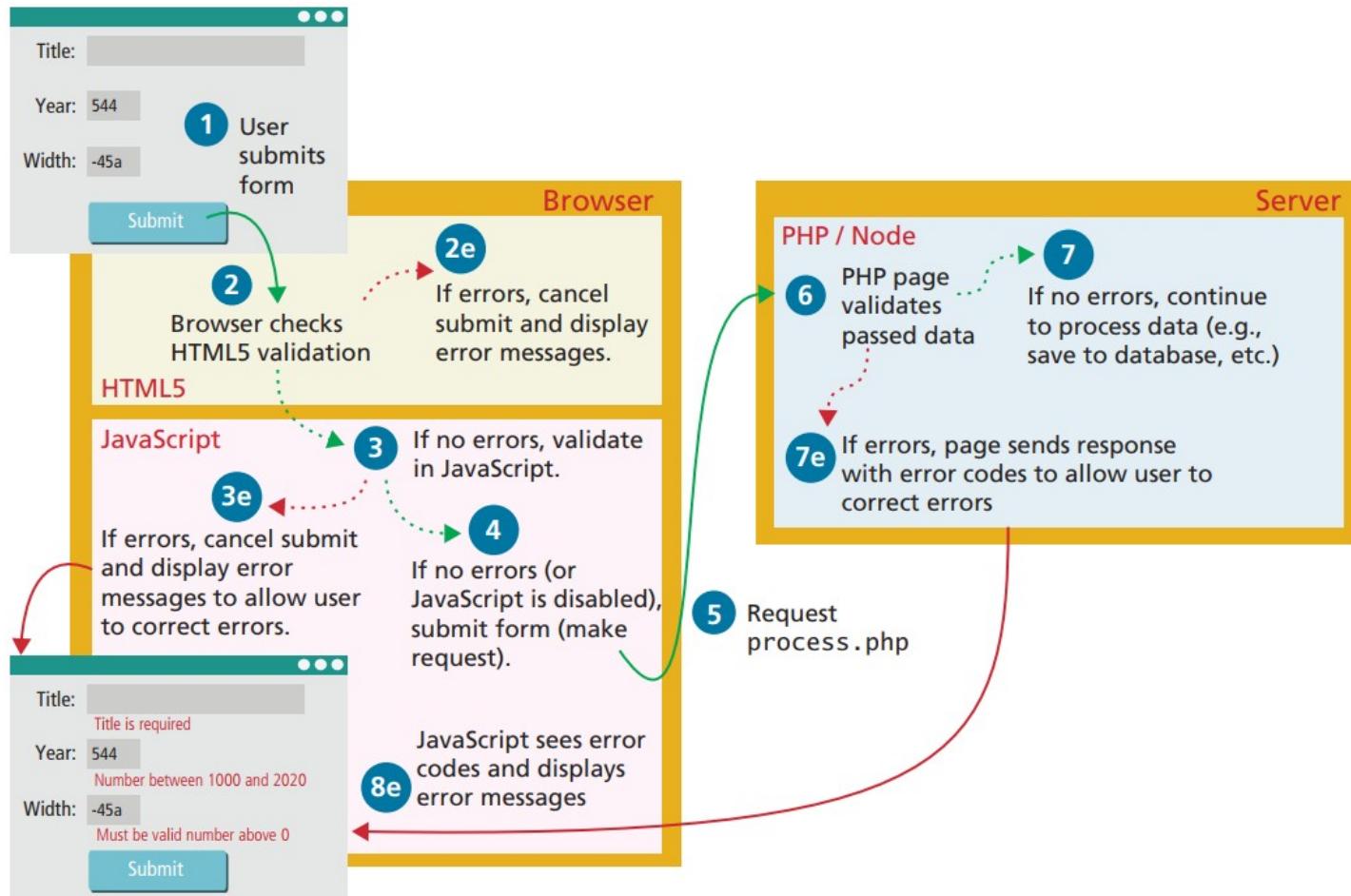
# ¿Cómo funcionan los formularios? (2)



# Elementos de Formularios

- Existe un número relativamente pequeño de elementos de formulario. A continuación se detallan categorizados:
  - Controles de Entrada de Texto:
    - **email, password, search, tel, text, textarea, url.**
  - Controles de elección:
    - **select, radio buttons, checkboxes**
  - Botones:
    - **input: submit, reset, image.**
    - **button:** se utiliza también como botón, con mayor flexibilidad.
  - Fecha y hora:
    - **date, time, datetime, datetime-local, month, week.**
  - Otros controles:
    - **file, number, range**

# Validación de datos de entrada



# Bibliografía

- **Libro:** Randy Connolly, Ricardo Hoar. **“Fundamentals of Web Development, Global Edition”**. 3era Edición. Ed. Pearson. 2022.
- **Libro:** Marijn Haverbeke. **“Eloquent JavaScript”**. 3era Edición. No Starch Press. 2018
- **Libro:** Lenny Burdette. **“The JavaScript Pocket Guide”**. 1ra Edición. Peachpit Press. 2010.
- **Libro:** John Pollock. **“JavaScript: A Beginner’s Guide”**. Ed. McGraw-Hill. 2013.
- **Web:** MDN Web Docs. **“Symbol”**. URL:  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Symbol](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol)
- **Web:** Medium. Latte and Code. **“JavaScript. ¿Conoces el tipo Symbol?”** URL:  
<https://latteandcode.medium.com/javascript-conoces-el-tipo-symbol-197c8338924a>



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

**Unidad 6 – Formularios**  
Programación Asíncrona

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

# Objetivos de la clase

- **Objetivos**

- Construir formularios.
- Conocer los formatos de intercambio de información en web.
- Conectarse con un servidor web y transferir información entre este y la web.

- **Temas a desarrollar:**

- Formularios. Composición.
- Elementos de Entrada: cajas de texto, botones, botones radio, áreas de texto.
- Listas desplegables. Botones de Control.
- Comunicación asíncrona e intercambio de datos en formato JSON.

# AJAX - Fundamentos

- La Web fue diseñada para operar con el **protocolo HTTP**: un protocolo simple de petición/respuesta.
- Los navegadores web solicitan un recurso y la respuesta es recibida y renderizada.
- Esto funciona bien para sitios web **centrados en documentos** ya que la idea es recuperar un documento o imagen y mostrarlos directamente en el browser.
- Sin embargo, este modelo no siempre es el más adecuado cuando programamos **sitios web dinámicos** donde un requerimiento común es solicitar información adicional luego que la página fue cargada.
- Pensemos en una **aplicación web** de chat:
  - Deberíamos controlar continuamente si hay mensajes disponibles e inmediatamente mostrarlos usando manipulación **DOM** o similar. Caso contrario, el usuario debería hacer un refresh de la página web para controlar si hay nuevos mensajes.
  - Si bien es sencillo refrescar una página web en determinados intervalos de tiempo (usando **Javascript**) el proceso de recarga es notorio e interrumpiría lo que el usuario está escribiendo y/o haciendo.

- **AJAX** es el acrónimo de **Asynchronous Javascript And XML**.
- Si analizamos la sigla:
  - **Asynchronous (asíncrono)**: significa que la **solicitud HTTP** no bloquea el hilo o pestaña del navegador. Esto muy importante, ya que significa que el browser podrá realizar otras tareas mientras la **solicitud HTTP** viaja a través de la red y es procesada.
    - Si la tecnología solo fuese sincrónica, el usuario de la aplicación imaginaria de chat no podría escribir mientras la aplicación busca mensajes del otro usuario.
  - **Javascript**: se basa en el objeto Javascript **XMLHttpRequest**.
  - **XML**: fue originalmente diseñado para intercambiar datos en formato **XML**.
- Con **AJAX** contamos con la habilidad de enviar y recibir datos desde el servidor sin tener que recargar la página completa.

# AJAX - XMLHttpRequest

- Para que en la web se pueden hacer aplicaciones como la de chat, en 1999 Microsoft introdujo una característica a JavaScript en Internet Explorer llamada objeto XMLHttpRequest.
- Trabajar con XMLHttpRequest permite que solicitudes HTTP puedan ser llevadas a cabo utilizando JavaScript una vez que la página ha sido cargada y la respuesta haya sido procesada e incorporada directamente en el árbol DOM.
- XMLHttpRequest fue incorporado eventualmente a los demás navegadores y este enfoque se llamó AJAX.
- Si bien AJAX es anterior a HTML5, la tecnología subyacente fue estandarizada con HTML5.
- Comúnmente nos referimos a recursos web que retornan datos en vez de HTML, CSS y JavaScript como Web APIs.
- La interacción con las Web APIs es naturalmente **asíncrona** ya que sucede sobre el protocolo HTTP.

# Programación Asíncrona

- El término **asíncrono** se refiere al concepto que más de una cosa ocurre al mismo tiempo, o múltiples cosas relacionadas ocurren **sin esperar** a que la previa se haya completado.
- Cuando se ejecuta dentro del navegador **JavaScript** es principalmente **single-threaded**. ¿Cómo se logra entonces la asincronía?
  - **Single-threaded** significa que JavaScript en sí corre en un único hilo, pero puede delegar tareas al entorno del navegador (event loop y Web APIs).
- El navegador administra múltiples hilos:
  - Uno de estos es para la ejecución del **JavaScript** de la página.
  - Otros hilos son el para el temporizador (timer), trabajar con archivos, acceder a una cámara web u obtener datos de la red.
- El código **JavaScript** interactúa con estos hilos a través de **funciones callback**.
- Por lo tanto, ya hemos programado código asíncrono:
  - Al establecer los **manejadores de eventos** programamos una **función callback** que se invocaría cuando se disparen.

# Funcionamiento de Web APIs

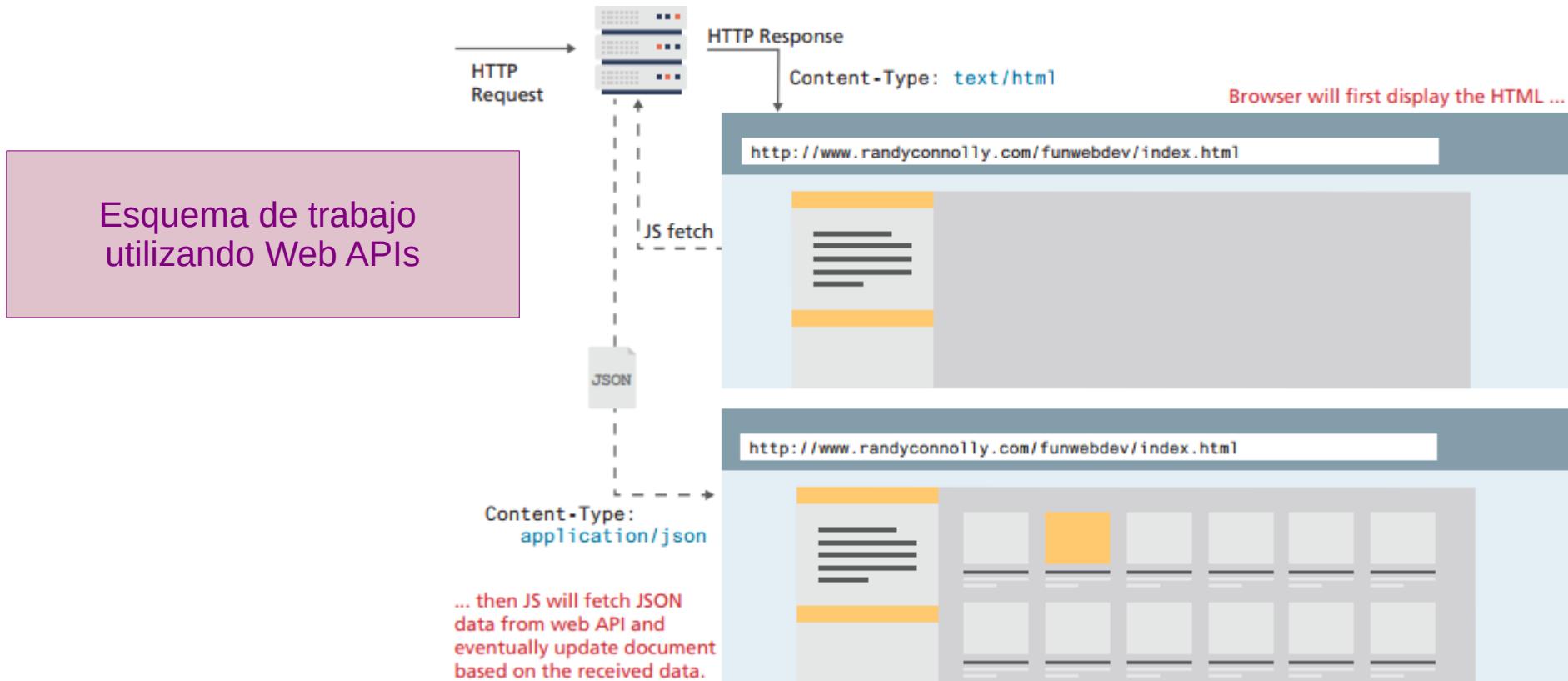
Cómo veníamos trabajando.  
Formato: text/html



Solicitud de Datos.  
Formato: application/json



# Funcionamiento de Web APIs (2)



# Fetch API

- Originalmente las solicitudes asíncronas en **JavaScript** requerían programas complejos que utilicen el objeto **XMLHttpRequest**.
- La librería **jQuery** creció en popularidad porque permitía, independientemente del navegador, hacer solicitudes a **Web APIs** ocultando detalles de implementación a los programadores.
- Los navegadores actuales tienen la **función fetch**. Lo que significa que **no se necesitan más librerías externas para hacer solicitudes asíncronas**.
- Puede ser que el recurso solicitado no esté disponible, por tanto, la **función fetch** en vez de devolvernos datos, nos devuelve un **Promise**.
- Una **Promise** es un objeto especial **JavaScript** que analizaremos en profundidad más adelante pero podemos pensarla como el lugar donde eventualmente arribarán los datos.
- Las **Promises** se trabajan invocando dos métodos:
  - **then()**: cuando la solicitud finaliza correctamente y se nos suministran los datos.
  - **catch()**: cuando no se pudo completar de manera exitosa la solicitud.
- Cada uno de estos métodos necesita una **función callback** como parámetro que es invocada cuando uno de estos eventos ocurre.

## Fetch API (2)

```
const prom = fetch('/api/cities.php?country=italy');
prom.then( response => {
    if (!response.ok) throw Error('Error consultando los datos')
    return response.json() //response.json() devuelve otra promise
})
.then( data => { console.log(data) })
.catch( error => { console.error(error) } );
})
.catch( error => { console.error(error) } );
```

# Fetch API (3)

- **Response.ok:** Puede pasar que una llamada **fetch** no sea exitosa. El método **catch** capturará los errores de red o **CORS**. Sin embargo, si el servidor nos retorna un error 404, no se disparará **catch**. Porque no hubo un error de red.
  - Esta situación nos obliga a siempre controlar si la propiedad **response.ok** es **true**.
- **Cross-Origin Resource Sharing (CORS):** es un error común con el que nos solemos encontrar al querer consumir datos de una **Web API** que no tiene el mismo dominio que la aplicación que estamos construyendo.
  - Es un mecanismo de seguridad implementado por los navegadores para prevenir cross-site scripting y así no permitir que código malicioso obtenga acceso al contenido de otra web.
- **Indicadores:** Los browsers no informan al usuario que se está ejecutando una solicitud **fetch**. Por ello debemos utilizar algún indicador para que el usuario sepa que algo está sucediendo. Ejemplo: <https://loading.io/>
- **Botones atrás y marcadores:** Las solicitudes **fetch** no permiten cambiar la URL en la barra de direcciones del browser. Por tanto: El usuario no puede agregar a marcadores o compartir la URL que corresponde al estado actual de la página y los botones adelante y atrás del browser ya no son útiles.
- **Accesibilidad y motores de búsqueda:** Los motores de búsqueda y los usuarios sin **JavaScript** no podrán ver los comportamientos “**AJAXtizados**”. Siempre es una buena idea estructurar el contenido de nuestros documentos de manera que las páginas puedan ser visualizadas aún si los scripts no funcionan.

# Promises (Promesas)

- Uno de los problemas clave con las invocaciones de **funciones callback** en **JavaScript** es cuando tenemos que anidar llamadas callback.
- Este esquema rápidamente pueden volverse bastante complicado de entender y depurar en especial con la programación asincrónica, en la que parte del código no se puede ejecutar hasta que primero se produzca otra devolución de llamada.
- Las **promesas** proporcionan un mecanismo del lenguaje para hacer la programación de callbacks menos complicada. Como sugiere el nombre, una **Promise** es un lugar reservado para recibir **un valor que ahora no tenemos** pero que **llegará más tarde** (es decir, está pendiente).
- Eventualmente, esa **promesa** se completará (es decir, se resuelve o se cumple), y recibiremos los datos, o no, y en su lugar obtendremos un error (es decir, se rechaza).
- Ya vimos que el método **fetch()** para realizar solicitudes asincrónicas de datos utiliza objetos **Promise**.
  - **fetch()** devuelve un objeto **Promise** que tiene los métodos **then()** y **catch()** a los que se les pasa una **función callback** que se invoca cuando se resuelve la promesa o se rechaza.
  - Nunca lo usamos pero también hay un método **finally()** que se puede llamar después de todos los métodos **then()** y **catch()** hayan sido invocados.

# Promesas - Creación

- Las promesas pueden ser utilizadas fuera de `fetch()`. De hecho podemos crear nuestras propias promesas para evitar anidar **callbacks**.
- La complejidad de crear una promesa reside en comprender que el manejador pasado por parámetro a su constructor debe tener dos funciones: `resolve()` y `reject()`.

```
const promiseObj = new Promise( (resolve, reject) => {
    if (someCondition)
        resolve(someValue);
    else
        reject(someMessage);
});

promiseObj
    .then( someValue => { /* Éxito, se ejecutó la promesa */ })
    .catch( someMessage => { /* Fallo. La promesa no fue satisfecha. */});
```

- Las funciones `resolve()` y `reject()` no deben invocarse simultáneamente ni múltiples veces.

# Async y Await

- ES7 introdujo las palabras reservadas **async** y **await** ambas pueden simplificar e incluso eliminar los típicos anidamientos de la programación asíncrona.
- En algunas situaciones necesitamos que nuestro código espere a que los datos retornen desde el servidor y estén disponibles para continuar con una tarea, disminuyendo también la cantidad de callbacks.
- La palabra reservada **await** (en inglés esperar) nos permite exactamente eso, esperar. **Es decir, trata una llamada asíncrona que retorna un objeto Promise como si fuera sincrónica.**
- Una limitación importante de **await** es que debe usarse dentro de una función que tenga como prefijo la palabra clave **async**. Por ejemplo:

```
async function getData(url) {  
  let response = await fetch(url);  
  let data = await response.json();  
  return data;  
}
```

- La razón por la que necesitamos **async** como prefijo de una función es para indicar que la función sigue siendo asíncrona y la espera solo tendrá lugar dentro de una función **async**.
- Las palabras **async** y **await** pueden utilizarse en cualquier función que retorne una **Promise**.

# Bibliografía

- **Libro:** Randy Connolly, Ricardo Hoar. ***“Fundamentals of Web Development, Global Edition”***. 3era Edition. Ed. Pearson. 2022.
- **Libro:** Lenny Burdette. ***“The JavaScript Pocket Guide”***. 1st Ed. Peachpit Press. 2010.
- **Libro:** John Pollock. ***“JavaScript: A Beginner’s Guide”***. Ed. McGraw Hill. 2013.



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

**Unidad 7 – Introducción a React JS**

Frameworks web. React. JSX y Componentes

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

# Objetivos de la clase

## • Objetivos

- Comprender la utilización del virtual DOM mediante los desarrollos de React JS.
- Conocer los hooks más importantes y estructurar una aplicación ReactJS.

## • Temas a desarrollar:

- ReactJS. Elementos de página. ReactDOM. Componentes React.
- ReactJS con JSX. Creación de una App React.
- Manejo de estados en React. Hooks useState, useEffect y useContext.
- Manejo de rutas con React Router.

# Frameworks JavaScript

- Un **framework** es una biblioteca de código que se utiliza para simplificar, mejorar y facilitar el proceso de desarrollo de una aplicación.
- La capacidad de usar un **framework front-end** se ha convertido en una **habilidad esencial** esperada **para la mayoría de los desarrolladores web**.
- Idealmente, los **frameworks**:
  - Mejoran la **productividad** del desarrollador realizando tareas comunes o simplificando tareas complejas.
  - **Reducen los errores** ya que están bien probados y son confiables.
  - Aumentan la **capacidad de mantenimiento** (al imponer estándares de diseño y patrones de mejores prácticas).
- Sin embargo, el uso de un **framework** generalmente implica una curva de aprendizaje adicional para el programador.
  - En el peor de los casos, un **framework** ofuscará el código simple con muchas abstracciones innecesarias y vinculará el éxito de un proyecto a una externalidad.

# ¿Por qué necesitamos un framework?

- ***iNo necesitamos un framework!*** Se pueden crear interfaces de usuario utilizando vainilla **JavaScript**.
- No obstante, los **frameworks** brindan algunos beneficios adicionales:
  - La creación de experiencias de usuario enriquecidas en **JavaScript** puede ser difícil. Esto significa tiempos de desarrollo más lentos y más errores. **Un framework simplifica el proceso de creación de estas interfaces de usuario.**
  - Los **frameworks** **pueden mejorar la velocidad de ejecución de la manipulación y el recorrido del DOM.**
    - Cada vez que modifica el **DOM** de alguna manera, por ejemplo, cambiando **innerHTML** o llamando a **appendChild()**, el navegador tiene que reconstruir el árbol de representación utilizado para el diseño y luego volver a pintar todos los nodos en la página. A veces el navegador **parpadea** cuando ejecutamos código **JavaScript** que hace manipulaciones **DOM** dentro de un bucle. Esto se debe a una limitación de rendimiento de **JavaScript** que un **framework** de front-end puede abordar.
    - Por ejemplo, **React** tiene un **DOM virtual** que nuestro código manipulará. Detrás de escena, **React** esperará el momento apropiado y luego realizará múltiples cambios en el **DOM real** en un solo lote eficiente, eliminando así la amenaza de **parpadeo**.
  - Los **frameworks** permiten al desarrollador construir la **interfaces de usuario como una serie de componentes anidados**. Por lo tanto, un componente es un bloque autónomo (encapsulado) de presentación, datos y comportamiento.
  -

# ¿Por qué necesitamos un framework? (2) - React Virtual DOM

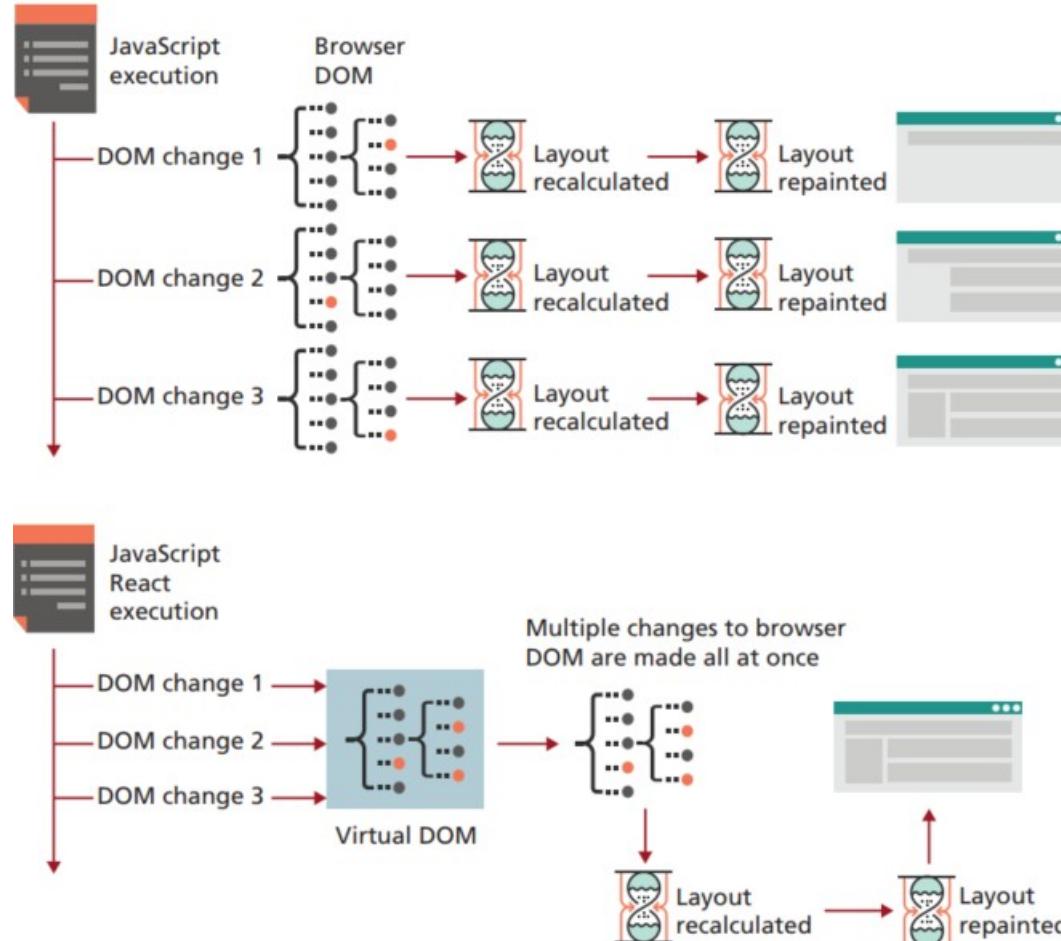


FIGURE 11.1 Virtual versus real DOM manipulations

# React, Angular y Vue

- Casi todo el tiempo están surgiendo nuevos frameworks JavaScript, sin embargo, los tres más grandes frameworks front-end son: Angular, React y Vue.
  - **Angular:** es un framework que nos obliga a adoptar una variante del patrón MVC para estructurar e implementar una aplicación web.
    - Desarrollar con Angular implica escribir **modelos** para representar sus datos, **plantillas** para manejar la presentación, **data bindings** para conectar la vista y el modelo, y **enrutamiento** para describir cómo los usuarios interactúan con la aplicación.
    - Fue creado y es mantenido por **Google**. La versión original de Angular usaba **JavaScript**, las versiones actuales usan **TypeScript**, un superconjunto sintáctico de JavaScript.
  - **React:** es el framework de **Facebook** para construir interfaces complejas en **JavaScript**. Hoy día es el framework más popular.
    - A diferencia de Angular, **React se enfoca solo en la vista** (es decir, la interfaz de usuario). Un componente de **React** está escrito en **JavaScript** y **JSX**, una extensión de **JavaScript** que permite que el markup se incruste en **JavaScript**. **React** tiene una curva de aprendizaje relativamente benigna, al menos en comparación con Angular.
  - **Vue.js:** es similar a **React** en que se enfoca en la vista. **Vue.js** usa **plantillas HTML** con **datos y comportamiento "inyectados"** a través de **directivas y atributos personalizados**.
    - A diferencia de **React** o **Angular**, **Vue.js** es completamente de código abierto y no está conectado a ninguna empresa de tecnología específica. Como resultado, **Vue.js** es particularmente popular fuera de Norteamérica y Europa.

# Single Page Application (SPA)

- Los tres **frameworks** son especialmente adecuados para construir una aplicación de una sola página (SPA). Como sugiere el nombre, **un SPA es un sitio Web que se construye a partir de una sola página.**
- Las **SPA** pueden ser bastante desafiantes de implementar a medida que crece su funcionalidad.
  - En una aplicación web que no es **SPA**, la funcionalidad se distribuye en diferentes páginas, lo que modulariza explícitamente la aplicación.
- En un **SPA** de **JavaScript**, toda la funcionalidad posible de la aplicación debe estar contenida en una sola página.
  - Esto puede resultar en archivos **JavaScript** monolíticamente grandes llenos de una mezcolanza de callbacks confusos y funciones anidadas dentro de funciones anidadas dentro de funciones, etc.
- Un **SPA** construido con un **framework** normalmente contiene **HTML** mínimo y **JavaScript** abundante para llenar el **DOM** con elementos **HTML** usando la lógica contenida dentro del framework y los datos extraídos de alguna API.
- Por lo tanto, los **frameworks** proporcionan un mecanismo para simplificar los datos o los requisitos de estado de una aplicación y una forma de manejar los eventos del usuario de manera independiente del **DOM**.

# Single Page Application (2)

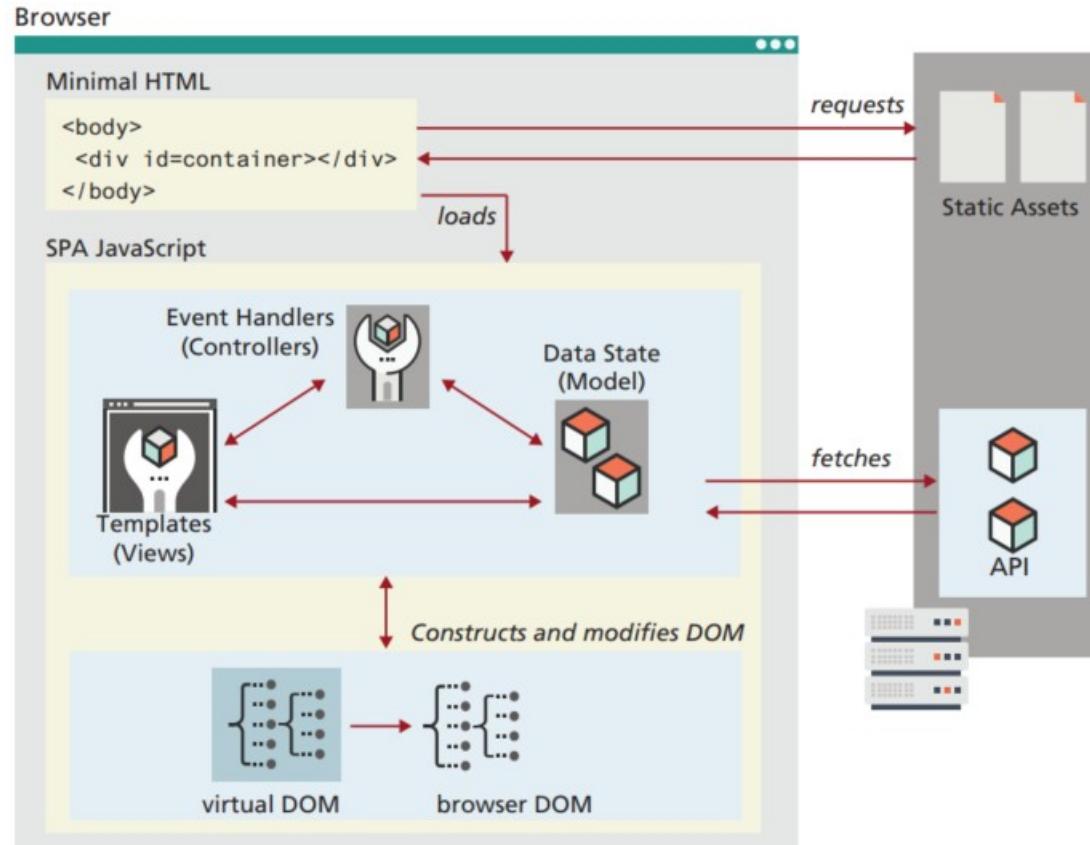


FIGURE 11.2 SPA using a framework

# Comenzando con React

- **ReactJS** es una biblioteca **JavaScript** creada por **Facebook** con el propósito de crear interfaces de usuario basada en componentes.
- Vamos a construir nuestra primer aplicación **React** que consiste simplemente en mostrar un enlace.
- Para mantenerlo simple utilizaremos bibliotecas **JavaScript** entre las que incluimos **Babel** que convertirá el código **JSX** a **JavaScript** en tiempo de ejecución (si bien el ejemplo no utiliza **JSX**).
- La página tiene un único elemento **HTML** (un **<div>**). Se usa **JavaScript** para completar ese elemento (muy similar a la que creábamos elementos utilizando **DOM**).

```
<!DOCTYPE html>
<html lang="es">
<head>

    ...
    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
    <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
    <script type="text/babel">
        const link = React.createElement("a",
            { href: "https://fcad.uner.edu.ar", target: "_blank" }, "Ir a fcad.uner.edu.ar");
        ReactDOM.render(link, document.querySelector("#react-container"));
    </script>
</head>
<body>
    <div id="react-container"></div>
</body>
</html>
```

## Comenzando con React (2)

- El ejemplo anterior no ilustra las ventajas de **React** ya que crear elementos usando **React.createElement** no dista mucho de lo que veníamos viendo en el curso.
- Como alternativa, **React** nos permite utilizar **JSX (JavaScript XML)**, al principio es extraño, pero luego nos damos cuenta que se trata de una forma fácil de crear elementos. Para ponerlo a prueba reemplazamos nuestro link con la siguiente línea:

```
const link = <a href="http://www.reactjs.org">Ir a la Web de React</a>;
```

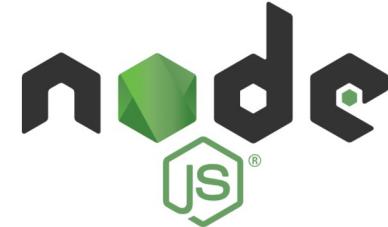
- Este es un ejemplo de **JSX**. Este código se convierte eventualmente en **JavaScript** para que el navegador lo entienda.
- **JSX** puede ser interpretado en tiempo de diseño o de ejecución. Lo ideal es que la interpretación suceda en tiempo de diseño. El código **JSX** se puede expandir varias líneas. Ejemplo:

```
const ejemplo = <div>
    <span>Empezando con React en IDW!</span>
    <a href="http://www.reactjs.org">Ir a la Web de React</a>;
</div>;
```

# Sintaxis JSX

- **JSX** sigue las siguientes reglas sintácticas:
  - Los nombres de los elementos deben estar compuestas por caracteres válidos (sin signos de puntuación o espacios) .
  - Los nombres de los elementos no pueden comenzar con un número.
  - Debe existir un único elemento raíz. Un elemento raíz es el que contiene otros elementos.
  - Todos los elementos deben tener su correspondiente etiqueta de cierre (o ser vacíos).
  - Los elementos deben estar apropiadamente anidados.
  - Los elementos pueden contar con atributos.
  - Los atributos deben siempre consignarse entre comillas.
  - Los elementos y atributos son case sensitive.
- Una de las más importantes diferencias entre **JSX** y **HTML** es que el atributo **class** en **JSX** es **className** (para no confundir con las clases **JavaScript**) .
- Ejemplo: `const heading = <h1 className="text-dark">Aquí va el título</h1>`

- **Node.js** (o simplemente **Node**) es un entorno de ejecución asíncrono y controlado por eventos que utiliza **JavaScript**.
- Fue desarrollado por Ryan Dahl en 2009 como una mejor manera de gestionar los problemas de concurrencia entre clientes y servidores.
- Es equivalente a otras tecnologías del lado del servidor como **PHP**. Una aplicación de **Node** puede generar **HTML** en respuesta a solicitudes **HTTP**, excepto que usa **JavaScript** como lenguaje de programación.
- **Node** es un **entorno de ejecución** extremadamente eficiente y eficaz. **No es un lenguaje de programación**.
  - Un **entorno de ejecución** es todo lo que se ejecuta para que podamos correr nuestros programas.
  - Otros lenguajes de programación tienen su propio entorno de ejecución:
    - Java → Java Runtime Environment (JRE).
    - .NET → Common Language Runtime (CLR).



# Node.js y Vite

- En este curso, no vamos a programar directamente en **Node.js**. Sin embargo, es un requerimiento para instalar herramientas como **Vite**.
- **Vite** es un servidor local de desarrollo programado por Evan You (creador de **Vue.js**) que es usado por defecto como plantilla de proyectos **Vue** y **React**. Tiene soporte para **TypeScript** y **JSX**. Internamente usa **Rollup** y **esbuild** para la construcción interna del proyecto.
  - Se distribuye bajo la licencia MIT.
  - Levanta un servidor local, compila el código, y transforma **JSX** a **JS** entendible por el navegador.
  - Es rápida, moderna, y reemplaza a **Webpack** en muchos proyectos **React**.
- **Vite** monitorea los archivos que son editados y los recarga en el navegador a través de un proceso que se llama **Hot Module Replacement (HMR)** que funciona solamente recargando los archivos específicamente modificados haciendo uso de los módulos de **ES6** en vez de recompilar la aplicación completa. Por defecto, atiende en el puerto **5173**.
- ¿Por qué no usar **create-react-app**? Está quedando obsoleto. **Vite** es más rápido, más simple, y tiene mejor soporte para nuevas herramientas.



```
npm create vite@latest «app»  
cd «app»  
npm install  
npm run dev
```

# React Components

- En **React**, un **componente** es una unidad autónoma de código que encapsula lógica y representación visual relacionada.
- Los **componentes** en **React** se utilizan para construir la interfaz de usuario de una aplicación web de manera modular, dividiéndola en **partes más pequeñas** y **manejables** que pueden ser **reutilizables, combinadas y anidadas** con múltiples componentes.
  - Lo que hace que el modelo de componentes de **React** sea especialmente atractivo es que podemos usar los componentes de forma programática o declarativa. Por ejemplo, el componente **Header** podría agregarse a la página mediante el siguiente JSX:

```
<Header>
  <Logo/>
  <Menu/>
  <Search initial="Find product"/>
  <Profile userId="34"/>
</Header>
```

- Como si se tratase de cualquier elemento **HTML**, los componentes de **React** también pueden incluir atributos.
- Los **componentes React** se definen usando **JavaScript**. Esencialmente, un **componente** es simplemente una función que devuelve un solo elemento (recordar que un elemento de React puede contener otros elementos). Hay dos tipos de componentes: **componentes de función** y **componentes de clase**.

# Componentes de Función

- La forma más fácil de crear un componente es usando funciones. El siguiente código crea un **componente funcional**:

```
const Logo = function(props) {  
  return ;  
};
```

- Debemos prestar atención a que **Logo** es una función que espera un único parámetro que se llama **props**.
  - Este parámetro es una parte importante de cómo funcionan los componentes ya que provee un mecanismo para pasar información al componente.
- Una vez que creamos el **componente funcional** lo podemos referenciar utilizando código markup. Por ejemplo, para usar **ReactDOM.render()** podemos agregarlo a través del código siguiente:

```
ReactDOM.render(<Logo/>, document.querySelector("#react-container"));
```

- Esta habilidad de referenciar los componentes a través de código markup nos permite componer las páginas como una serie de **componentes anidados**. Nótese que si necesitamos devolver un componente que cuenta con varias líneas debemos encerrar el código entre paréntesis.

```
const Title = function(props) { return <h2>Site Title</h2>; };  
  
const Header = function(props) {  
  return ( <header>  
    <Logo/>  
    <Title/>  
  </header>);  
}
```

# Componentes de clase

- Una forma un poco más complicada de crear un componente **React** es usar la palabra clave **JavaScript class**.
- Podríamos reescribir nuestro componente **Header** como el siguiente componente de clase:

```
class Header extends React.Component {  
  render() {  
    return (<header>  
      <Logo />  
      <Title />  
    </header>);  
  }  
}
```

- La función que muestra el componente debe llamarse **render** y es requerida en todos los componentes de clase.

# Props

- Cuando programamos nuestro componente funcional dijimos que las funciones deben contar con un único parámetro que por convención se llama **props**.
- El parámetro **props** es un objeto que contiene cualquier valor pasado a la función a través de atributos de markup. Esto nos brinda una manera fácil de crear componentes **React** de propósito general y adaptable a múltiples usos.
- Los componentes **React** no deben modificar sus parámetros props. Es decir, deben ser funciones puras.
- Utilizamos `{ }` en los componentes **Logo** y **Title** para inyectar **JavaScript** en el markup **JSX**.

```
const Logo = function(props){  
  return ;  
};  
  
const Title = (props) => { return <h2> {props.label} </h2> };  
  
const Header = (props) => { return (<header>  
  <Logo filename="logo.png" alt="Site logo here" />  
  <Title label = "Site Title" />  
</header>);  
};
```

# Props – Pasando objetos complejos

- Los **props** pueden usarse para pasar cualquier tipos de datos. Por ejemplo, imagina que tenemos un array de películas:

```
const movieData = [ { id: 17, title: "American Beauty", year: 1999 }, { ... }, { ... } ];
```

- Podemos pasar un array a un componente utilizando **props**:

```
<MovieList movies={movieData} />
```

- En el componente **MovieList** mostramos los ítems del array así:

```
const MovieList = (props) => {  
  const items = [];  
  for (let m of props.movies) {  
    items.push( <li>{m.title}</li> );  
  }  
  return <ul>{ items } </ul>;  
}
```

- Usando **map** podríamos hacer:

```
const MovieList = (props) => {  
  const items = props.movies.map( m => <li> {m.title} </li> );  
  return <ul> { items } </ul>;  
}
```

# Props – Pasando objetos complejos (2)

- La versión anterior del componente hará que **React** nos arroje una alerta en la consola: **Each child in a list should have a unique key prop.**
- **React** usa claves para identificar únicamente elementos hijos en una colección de elementos. Si bien no siempre necesitamos esta funcionalidad, es relativamente sencillo agregar una clave única a cada ítem proporcionando un índice a través de map:

```
props.movies.map((n, index) => <li key={index}> {m.title} </li>);
```

- Hasta ahora vimos ejemplos donde los atributos de los componentes en formato markup se pasan a props.
- Una forma alternativa de pasar información es anidando los datos como contenido del elemento.

**<Title> Iterating a Props Array <Title>**

- En este caso accedemos a la información usando **props.children**:

```
const Title = (props) => <h2> {props.children} </h2>
```

# Comportamientos

- En unidades previas del curso vimos como responder a eventos JavaScript usando `addEventListener()`.
- Los eventos también son importantes para los componentes React. La forma de gestionar los eventos es similar a la gestión de eventos en línea.
- Por ejemplo, supongamos que queremos agregar un botón a cada una de nuestras películas que nos permita conocer más información esa película en particular.
- Podemos definir un método handler dentro de `MovieListItem` y luego referenciarlo a través del atributo `onClick`:

```
const MovieListItem = (props) => {  
  const handleclick = () => { alert("handling the click id=" + props.movie.id);  
  return (  
    <li> {props.movie.title}  
      <button onClick={handleClick}> View </button>  
    </li>);  
}
```

# Bibliografía

- **Libro:** Randy Connolly, Ricardo Hoar. “***Fundamentals of Web Development, Global Edition***”. 3era Edition. Ed. Pearson. 2022.
- **Libro:** Alex Bank and Eve Porcello. “***Learning React. Modern Patterns for Developing React Apps.***” 2da Edición. O'Reilly Media, Inc. 2020.
- **Web:** Learn React. URL: <https://react.dev/learn>



Facultad de Ciencias  
de la Administración

TECNICATURA  
UNIVERSITARIA EN  
**DESARROLLO  
WEB**



# INTRODUCCIÓN AL DESARROLLO WEB

**Unidad 7 – Introducción a React JS**

Hooks y Router

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración - Universidad Nacional de Entre Ríos

# Objetivos de la clase

## • Objetivos

- Comprender la utilización del virtual DOM mediante los desarrollos con React JS.
- Conocer los hooks más importantes y estructurar una aplicación ReactJS.

## • Temas a desarrollar:

- JavaScript avanzado: Desestructuración de Objetos y Arrays. Operador Spread. Async y Await.
- Promises. Módulos y clases en ES6. Conceptos de programación funcional.
- ReactJS. Elementos de página. ReactDOM. Componentes React.
- ReactJS con JSX. Creación de una App React.
- Manejo de estados en React. Hooks useState, useEffect y useContext.
- Manejo de rutas con React Router.

# React Hooks

- Los **Hooks** son un mecanismo que nos permite usar diferentes funciones de **React** desde los componentes que construimos.
- Permiten encapsular lógica reutilizable que puede mantenerse separada del árbol de componentes. Nos permiten **conectar** o **asignar funcionalidades** a nuestros componentes.
- Desde su versión 16.8, **React** cuenta con hooks integrados que podemos usar de forma inmediata.
- Algunos **hooks built-in** (integrados) de **React** son:
  - Estado: **useState**, **useReducer**.
  - Efecto: **useEffect**.
  - Contexto: **useContext**.
  - Referencia: **useRef**.
  - Performance: **useMemo**.

# El Hook useState

- **useState** nos permite agregar una variable de estado a nuestros componentes.
- Este hook está directamente disponible a partir del paquete **react**. Por tanto sólo necesitamos importarlo. A continuación un ejemplo de uso:

```
import { useState } from "react";

function MyComponent() {

  const [age, setAge] = useState(28);
  const [name, setName] = useState("Taylor");
  const [todos, setTodos] = useState(() => createTodos());
```

## • Parámetros

- Por convención se nombra a las variables de estado de esta forma **[persona, setPersona]** usando desestructuración de arrays.
- El parámetro de **useState** es el valor inicial con el que queremos inicializar el estado. Este valor solo se usa en el primer renderizado; en renderizados posteriores, **React** conservará el estado actualizado.
  - Puede ser un valor de cualquier tipo pero si es una función debe para las mismas entradas responder con iguales resultados (se conoce como **función pura**).

# El hook useState (2)

- **Valores de retorno**
  - Los valores devueltos por el hook pueden ser desestructurados en dos partes:
    - 1) El estado actual que durante el primer renderizado coincidirá con el estado inicial.
    - 2) La función que nos permite actualizar el estado con un valor diferente y que desencadenará un nuevo renderizado.
- **Actualizar objetos y arrays en estado**
  - Podemos usar objetos y arrays como variables de estado pero en **React** el estado es considerado una variable de solo lectura. Por tanto es necesario **reemplazar** en vez de **mutar** los objetos existentes. Por ejemplo, si tenemos un objeto persona:

```
form.firstName = "Taylor"; // ✗ incorrecto.  
  
setForm({  
  ...form,  
  firstName: "Taylor"  
}); // ✓ Correcto.
```

# El hook useEffect

- **useEffect** es un Hook que nos permite sincronizar un componente con un **sistema externo**.
- Usamos **useEffect** cuando un render necesita causar **efectos secundarios**. Un **efecto secundario** es algo que hace una función que no es parte del retorno.
- La función representa la interfaz de usuario pero es posible que queramos que el componente haga más que eso. Aquellas tareas que queremos que haga el componente además de devolver la interfaz de usuario se denominan efectos.

```
function Checkbox {  
  const [checked, setChecked] = useState(false);  
  
  useEffect(() => { alert(`checked: ${checked.toString()}`) }, [checked]);  
  
  return (  
    <>  
      <input type="checkbox"  
            checked={checked}  
            onChange={() => setChecked(checked => !checked)} />  
      {checked ? "checked" : "not checked"}  
    </>  
  );  
};
```

# El hook useEffect – Array de dependencias

- **useEffect** está diseñado para trabajar en conjunto con otros hooks como **useState** y/o **useReducer**.
- **React** siempre vuelve a renderizar un componente cuando su estado cambia.
- En caso que usemos **useState** y **useEffect** es posible que invoquemos más veces de las necesarias a **useEffect** causando problemas de performance.
- En estas situaciones es necesario especificar en un **array** los cambios de qué variables de estado van a exigir que se invoque nuevamente a **useEffect**. Este array se conoce como **array de dependencias**.
- Especificar como parámetro `[]` (un array vacío) equivale a decir que **useEffect** debe ejecutarse sólo una vez.

# Routing

- **React Router** es un paquete que nos permite configurar **rutas**. Pero... ¿qué es una ruta?
- En **HTML** normal, utilizamos **hipervínculos** (`<a>`) para “saltar” de una página a otra. Pero en una aplicación de una sola página.... solo hay una página.
- Por esto, debemos usar **React Router**, para convertir enlaces a diferentes páginas en enlaces para mostrar diferentes **componentes React**.
  - Debido a que es posible que los usuarios deseen utilizar marcadores, **React Router** gestiona la URL del navegador utilizando la API de historial para permitir navegación sin recargar la página.
- **React Router**, como cualquier biblioteca de componentes para **React**, se puede agregar a nuestro proyecto instalándolo usando **npm**.
- Los componentes más destacados de este paquete son `<BrowserRouter>`, `<Route>` y `<Link>`.

# React Context

- Con lo visto hasta el momento, si queremos que todos los componentes de un árbol de componentes tengan acceso a información de estado es necesario **pasar** ese estado **hacia abajo y arriba** utilizando **propiedades**.
- Esto está bien cuando el árbol de componentes es simple pero no podemos aplicar el mismo enfoque a una aplicación real. En este caso, el enfoque se vuelve tedioso y muy propenso a errores.
- Haciendo una analogía con el mundo real es como hacer un viaje de larga distancia en colectivo parando en todas las terminales que están de paso a nuestro destino.
  - Lo que necesitamos es un vuelo directo.
- Los **contextos** de **React** son la forma de “viajar en avión”. Podemos colocar información en un contexto de **React** utilizando un **Context Provider**.
- Un **Context Provider** es un componente que envuelve todos los componentes del árbol que queremos que tengan acceso al estado.
  - Podemos pensar en el **Context Provider** como el aeropuerto de salida donde la información hace el embarco y el destino de cada vuelo es un **Context Consumer**.
- Un **Context Consumer** es un componente que recupera los datos desde el contexto utilizando `useContext()`.



# Context API – Crear contexto

```
import { createContext, useState } from "react";
const AuthContext = createContext();
export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  return (
    <AuthContext.Provider value={{ user, setUser }}>
      {children}
    </AuthContext.Provider>
  );
};
```

# Context API – Disponibilidad

```
import { AuthProvider } from "./AuthContext";  
  
const App = () => (  
  <AuthProvider>  
    <MyComponent/>  
  </AuthProvider>  
);
```

# Context API – Consumir el contexto

```
import { useContext } from "react";
import { AuthContext } from "./AuthContext";
const MyComponent = () => {
  const { user, setUser } = useContext(AuthContext);
  return (
    <div>
      {user ? <p>Bienvenido, {user.username}</p> : <p>Por favor inicie sesión.</p>}
    </div>
  );
};
```

# Bibliografía

- **Libro:** Randy Connolly, Ricardo Hoar. “***Fundamentals of Web Development, Global Edition***”. 3era Edition. Ed. Pearson. 2022.
- **Libro:** Alex Bank and Eve Porcello. “***Learning React. Modern Patterns for Developing React Apps.***” 2da Edición. O'Reilly Media, Inc. 2020.
- **Web:** **useState**. URL: <https://react.dev/reference/react/useState>
- **Web:** **useEffect**. URL: <https://react.dev/reference/react/useEffect>
- **Web:** **useContext**. URL: <https://react.dev/reference/react/useContext>



A thin horizontal bar consisting of two segments: a teal segment on the left and an orange segment on the right.

# GIT

# Introducción al desarrollo web

Lic. Faure Analia

Sistema de control de versiones

Sistema distribuido

Velocidad de actualización, es casi instantánea

Diseño sencillo

Gran soporte para desarrollo



Name

- v1.0
- v2.0
- v2.1
- v2.2



Name

- project
- project-revised
- project-final
- project-final-for-real



Name

- project
- projecttt
- projectttttttt
- .....



- 
- 1. Instalación
  - 2. Configuración
  - 3. Primera parte
  - 4. Segunda parte
  - 5. Tercera parte

## Instalación

<https://git-scm.com/>

Descargar la versión del sistema operativo que tengamos.

Verificar el número de versión de Git que hemos instalado: **git --version**

```
PS C:\Users\admin> git --version
git version 2.43.0.windows.1
PS C:\Users\admin>
```

---

## Configuración de datos iniciales de Git

Establecemos el nombre de usuario y dirección de correo electrónico que luego cada vez que confirmamos los cambios en el repositorio de Git se usarán esta información que se introduce.

Debemos ejecutar los siguientes comandos:

**git config --global user.name [ nombre ]**

**git config --global user.email [ correo electronico ]**

Ver datos de configuración:

**git config --list**

---

## Primera parte.

`git init`

`git clone`

`git pull`

`git checkout -b rama-nueva`

`git remote add origin https://gitlab.com/usuario/nombre-del-repositorio.git`

---

## Segunda parte.

**git status (ver como esta mi ámbito de trabajo)**

**git pull origin main**

**git add ./\* (el punto agrega todos los archivos, se puede especificar qué archivo subir)**

**git commit -m “comentario”**

**git push origin rama-nueva**



## Tercera parte.

Fusionar las ramas:

Dentro de git crear un pull request de la rama-nueva contra la rama main

---

— — —

## ¿ Preguntas ?

**Gracias por su atención!**

# EJERCICIO DE CLASE N.º 1: JavaScript

INTRODUCCIÓN AL DESARROLLO WEB - 2025 1er cuatrimestre  
TECNICATURA UNIVERSITARIA EN DESARROLLO WEB

1. Programe la función `min` que reciba dos valores por argumento y determine cuál de ellos es el menor.
2. Escriba un programa que use `console.log` para imprimir todos los números de `1` a `100` con dos excepciones. Para números divisibles por `3`, imprimir “**Fizz**” en vez de el número, y para números divisibles por `5` y no por `3`, imprimir “**Buzz**”.
3. Modifique el ejercicio anterior para que imprima “**FizzBuzz**” para números que son divisibles por `3` y por `5` (y continúe mostrando “**Fizz**” o “**Buzz**” para números que son divisibles para `3` o `5` individualmente).
4. En Python podemos hacer `sum(range(1, 10))`. Escriba la función `range` de modo que tome dos argumentos: `inicio` y `fin` y retorne un array de todos los números enteros desde `inicio` hasta `fin` (inclusive).
  - a) Programe la función `sum` que tome todos los números de un array y retorne la sumatoria de esos números. Por ejemplo: `sum(range(1, 10)) = 55`.
  - b) Modifique la función `range` para tomar un tercer argumento opcional de nombre `paso` que indique el salto que debe hacerse para construir el array. Si no se suministra un valor para este argumento el valor debe ser `1`.  
Por ejemplo: `range(1, 10, 2)` debe devolver `[1, 3, 5, 7, 9]`.
  - c) Asegúrese que funciona con valores de `paso` negativo.  
Por ejemplo: `range(5, 2, -1)` produce `[5, 4, 3, 2]`.

Comenzado el miércoles, 30 de abril de 2025, 21:36

Estado Finalizado

Finalizado en miércoles, 30 de abril de 2025, 21:43

Tiempo 6 minutos 56 segundos  
empleado

Calificación 10,00 de 10,00 (100%)

### Pregunta 1

Correcta

Se puntúa 1,00 sobre 1,00

Bootstrap 5 requiere Popper.js para ciertos componentes interactivos.

- Verdadero ✓  
 Falso

La respuesta correcta es 'Verdadero'

### Pregunta 2

Correcta

Se puntúa 1,00 sobre 1,00

¿Cuántas columnas tiene el sistema de rejillas de Bootstrap?

- a. 10  
 b. 12 ✓  
 c. 4  
 d. 6

La respuesta correcta es: 12

### Pregunta 3

Correcta

Se puntúa 1,00 sobre 1,00

Sin establecer un viewport correcto, el navegador móvil escalará la versión de escritorio.

- Verdadero ✓  
 Falso

La respuesta correcta es 'Verdadero'

**Pregunta 4**

Correcta

Se puntuá 1,00 sobre 1,00

Bootstrap es un framework que incluye herramientas tanto para front-end como para back-end.

 Verdadero Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 5**

Correcta

Se puntuá 1,00 sobre 1,00

El elemento HTML `<picture>` permite servir diferentes versiones de una imagen según el tamaño de pantalla.

 Verdadero ✓ Falso

La respuesta correcta es 'Verdadero'

**Pregunta 6**

Correcta

Se puntuá 1,00 sobre 1,00

El diseño adaptativo busca crear múltiples sitios web específicos para cada tipo de dispositivo.

 Verdadero Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 7**

Correcta

Se puntuá 1,00 sobre 1,00

¿Cuál es un ejemplo correcto de media query para pantallas medianas?

- a. @media width >
- b. 768px
- c. @media screen-width: 768px
- d. @media (min-width: 768px) ✓
- e. @media screen(768px)

Las respuestas correctas son: 768px, @media (min-width: 768px)

**Pregunta 8**

Correcta

Se puntuá 1,00 sobre 1,00

¿Qué propiedad CSS se usa para hacer que las imágenes no excedan el tamaño del contenedor?

- a. max-width: 100% ✓
- b. min-width: 100%
- c. height: 100%
- d. width: auto

La respuesta correcta es: max-width: 100%

**Pregunta 9**

Correcta

Se puntuá 1,00 sobre 1,00

El enfoque mobile-first propone diseñar primero para dispositivos de escritorio y luego adaptar para móviles.

- Verdadero
- Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 10**

Correcta

Se puntuá 1,00 sobre 1,00

¿Qué técnica permite separar reglas CSS en distintos archivos según el tamaño de pantalla?

- a. Cambio de tema manual
- b. Uso de JavaScript dinámico
- c. Multiples archivos HTML
- d. Uso de media queries en los links de CSS ✓

La respuesta correcta es: Uso de media queries en los links de CSS

[◀ Libro: Fundamentals of Web Development 3rd Edition](#)

Ir a...

[Libro: Learning Bootstrap ►](#)

Comenzado el domingo, 23 de marzo de 2025, 13:10

Estado Finalizado

Finalizado en domingo, 23 de marzo de 2025, 13:20

Tiempo 10 minutos 1 segundos  
empleado

Puntos 10,00/12,00

Calificación 8,33 de 10,00 (83,33%)

#### Pregunta 1

Correcta

Se puntúa 1,00 sobre 1,00

¿Qué elementos pueden tener las páginas Web además de texto?

- a. Enlaces a otras páginas ✓
- b. Imágenes ✓
- c. Videos ✓
- d. Fibra Óptica
- e. Tablas ✓

Respuesta correcta

Las respuestas correctas son:

Imágenes,

Videos,

Enlaces a otras páginas,

Tablas

#### Pregunta 2

Correcta

Se puntúa 1,00 sobre 1,00

Seleccionar cuáles de las siguientes capas componen el protocolo TCP/IP.

- a. Capa de Aplicación ✓
- b. Capa de Internet ✓
- c. Capa de Enlace ✓
- d. Capa Física
- e. Capa de Red

Respuesta correcta

Las respuestas correctas son:

Capa de Aplicación,

Capa de Internet,

Capa de Enlace

**Pregunta 3**

Correcta

Se puntuó 1,00 sobre 1,00

Indicar las características correctas de la llamada Web 1.0

- a. Los usuarios podían realizar modificaciones sobre los sitios web.
- b. Muy comunes en la década del 2000.
- c. Sitios web estáticos. ✓
- d. El webmaster se encargaba de las modificaciones sobre los sitios web. ✓

Respuesta correcta

Las respuestas correctas son:

El webmaster se encargaba de las modificaciones sobre los sitios web.,

Sitios web estáticos.

**Pregunta 4**

Incorrecta

Se puntuó 0,00 sobre 1,00

Una página Web se compone de uno o más sitios Web.

Seleccione una:

- Verdadero ✕
- Falso

Correcto!

La respuesta correcta es 'Falso'

**Pregunta 5**

Correcta

Se puntuó 1,00 sobre 1,00

Un sitio Web no puede tener contenido dinámico.

Seleccione una:

- Verdadero
- Falso ✓

Correcto!

La respuesta correcta es 'Falso'

**Pregunta 6**

Incorrecta

Se puntuá 0,00 sobre 1,00

Siempre es necesario contar con un servidor DNS para poder acceder un sitio web desde un navegador. Incluso si el sitio se accede en una red local de computadoras.

 Verdadero ✗ Falso

La respuesta correcta es Falso. Para poder acceder a un sitio web contar con un servidor DNS como intermediario. Si conocemos la dirección IP del servidor podremos acceder a los recursos que exponga.

La respuesta correcta es 'Falso'

**Pregunta 7**

Correcta

Se puntuá 1,00 sobre 1,00

La capa Internet resuelve problemas como la creación, transmisión y recepción de paquetes.

Seleccione una:

 Verdadero Falso ✓

Incorrecto.

La respuesta correcta es 'Falso'

**Pregunta 8**

Correcta

Se puntuá 1,00 sobre 1,00

La siguiente dirección IP 8.8.8.8, ¿es IPv4?

Seleccione una:

 Verdadero ✓ Falso

Correcto!

La respuesta correcta es 'Verdadero'

**Pregunta 9**

Correcta

Se puntúa 1,00 sobre 1,00

La siguiente dirección IP: **192.168.1.270** es una dirección IP válida?

Seleccione una:

 Verdadero Falso ✓

Correcto!

La respuesta correcta es 'Falso'

**Pregunta 10**

Correcta

Se puntúa 1,00 sobre 1,00

La capa de Aplicación posee menor nivel abstracción que el resto de las capas.

Seleccione una:

 Verdadero Falso ✓

Correcto!

La respuesta correcta es 'Falso'

**Pregunta 11**

Correcta

Se puntúa 1,00 sobre 1,00

Los programadores o desarrolladores suelen trabajar en la Capa de Aplicación.

Seleccione una:

 Verdadero ✓ Falso

Correcto!

La respuesta correcta es 'Verdadero'

**Pregunta 12**

Correcta

Se puntúa 1,00 sobre 1,00

En la capa de Transporte en caso que se pierda un paquete. El emisor lo retransmite hacia el receptor.

Seleccione una:

Verdadero ✓

Falso

Correcto!

La respuesta correcta es 'Verdadero'

◀ Lectura sugerida: Fundamentals of Web Development (Contraseña IDW-FCAD-UNER)

Ir a...

U1 (Práctica) - Introducción a GIT (act. 28/03/2025) ►



## Cuestionario Semana N° 2

Comenzado el	miércoles, 9 de abril de 2025, 18:22
Estado	Finalizado
Finalizado en	miércoles, 9 de abril de 2025, 18:29
Tiempo empleado	6 minutos 47 segundos
Calificación	10,00 de 10,00 (100%)

## Pregunta 1

Correcta

Se puntúa 1,00 sobre 1,00

Marcar pregunta

¿Cuál de las siguientes opciones describe correctamente el propósito de un método POST en HTTP?

- a. Obtener un recurso sin cuerpo de respuesta.
- b. Borrar un recurso del servidor.
- c. Enviar datos al servidor para ser procesados. ✓
- d. Verificar si se tiene acceso a un host.

La respuesta correcta es: Enviar datos al servidor para ser procesados.

## Pregunta 2

Correcta

Se puntúa 1,00 sobre 1,00

Marcar pregunta

¿Cuál de los siguientes puertos corresponde típicamente a un servidor web?

- a. 21
- b. 80 ✓
- c. 443 ✗
- d. 3306

La respuesta correcta es: 80

## Pregunta 3

Correcta

Se puntúa 1,00 sobre 1,00

Marcar pregunta

Un servidor puede atender a varios clientes simultáneamente.

- Verdadero ✓
- Falso

La respuesta correcta es 'Verdadero'

## Pregunta 4

Correcta

HTTP fue desarrollado por:

## Navegación por el cuestionario

1	2	3	4	5	6	7
8	9	10				
✓	✓	✓				

[Mostrar una página cada vez](#)[Finalizar revisión](#)

Comenzado el martes, 8 de abril de 2025, 17:39

Estado Finalizado

Finalizado en martes, 8 de abril de 2025, 17:47

Tiempo 7 minutos 19 segundos  
empleado

Calificación 10,00 de 10,00 (100%)

### Pregunta 1

Correcta

Se puntúa 1,00 sobre 1,00

Para mostrar texto en negrita se puede utilizar el elemento **texto**

- Verdadero ✓  
 Falso

La respuesta correcta es 'Verdadero'

### Pregunta 2

Correcta

Se puntúa 1,00 sobre 1,00

Es recomendable utilizar **texto** e *texto* para delimitar texto en negrita y cursiva

- Verdadero  
 Falso ✓

La respuesta correcta es 'Falso'

### Pregunta 3

Correcta

Se puntúa 1,00 sobre 1,00

Las etiquetas **,** , *,*  son recomendadas en HTML5.

- Verdadero  
 Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 4**

Correcta

Se puntuá 1,00 sobre 1,00

En un página web desarrollada con HTML la única forma de implementar un enlace a otro recurso es a través del elemento `<a>`

 Verdadero Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 5**

Correcta

Se puntuá 1,00 sobre 1,00

¿Cuál de las siguientes opciones representa un elemento vacío en HTML?

- a. `<p>`
- b. `<strong>`
- c. `<em>`
- d. `<img>` ✓

La respuesta correcta es: `<img>`

**Pregunta 6**

Correcta

Se puntuá 1,00 sobre 1,00

Los encabezados HTML son h1, h2, h3, h4, h5 y h6 y su tamaño va en orden ascendente siendo el menos importante h1 y el más importante h6

 Verdadero Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 7**

Correcta

Se puntuá 1,00 sobre 1,00

Para desplegar una lista no ordenada se utiliza `<ul></ul>`

 Verdadero ✓ Falso

La respuesta correcta es 'Verdadero'

**Pregunta 8**

Correcta

Se puntuá 1,00 sobre 1,00

Las listas de tipo `<ul>` son listas ordenadas.

 Verdadero Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 9**

Correcta

Se puntuá 1,00 sobre 1,00

¿Qué atributo del tag `<a>` permite especificar la URL de destino?

- a. href ✓
- b. target
- c. name
- d. title

La respuesta correcta es: href

**Pregunta 10**

Correcta

Se puntuó 1,00 sobre 1,00

¿Cuál de las siguientes afirmaciones sobre enlaces es falsa?

- a. Si antepongo # a un identificador coincidente con el atributo `id` de un elemento de la página web logro que el navegador haga scroll hasta ese elemento.
- b. Los elementos `<a>` son contenedores de nivel de bloque. ✓
- c. Si el valor del atributo `href` comienza con `mailto:` se puede abrir el cliente de correo por defecto del sistema
- d. A través del atributo `target` se puede configurar la forma en la que se mostrará la página destino.

Respuesta correcta

La respuesta correcta es:

Los elementos `<a>` son contenedores de nivel de bloque.

[◀ Lectura Sugerida: Fundamentals of Web Development](#)

Ir a...

[Guía para principiantes de HTML ►](#)

Comenzado el lunes, 14 de abril de 2025, 08:46

Estado Finalizado

Finalizado en lunes, 14 de abril de 2025, 09:00

Tiempo 14 minutos  
empleado

Calificación 10,00 de 10,00 (100%)

### Pregunta 1

Correcta

Se puntúa 1,00 sobre 1,00

La pseudoclase `:hover` se aplica cuando el usuario hace clic en un elemento.

- Verdadero
- Falso ✓

La respuesta correcta es 'Falso'

### Pregunta 2

Correcta

Se puntúa 1,00 sobre 1,00

¿Cuál de los siguientes no es un beneficio de usar CSS?

- a. Mayor velocidad de carga
- b. Mejora del rendimiento del backend ✓
- c. Flexibilidad para múltiples dispositivos
- d. Simplificación del mantenimiento

La respuesta correcta es: Mejora del rendimiento del backend

### Pregunta 3

Correcta

Se puntúa 1,00 sobre 1,00

¿Qué propiedades CSS están relacionadas con la fuente del texto?

- a. text-indent
- b. font-size
- c. font-family ✓
- d. font-weight

Las respuestas correctas son: font-size, font-family, font-weight

**Pregunta 4**

Correcta

Se puntúa 1,00 sobre 1,00

Para reutilizar estilos es conveniente hacer uso de...

- a. CSS importado ✓
- b. CSS incorporado
- c. CSS en línea

Respuesta correcta

La respuesta correcta es:

CSS importado

**Pregunta 5**

Correcta

Se puntúa 1,00 sobre 1,00

El Box Model de CSS incluye las siguientes partes:

- a. Block, Inline, Grid, Flex
- b. Content, Padding, Border, Margin ✓
- c. Height, Width, Radius, Margin
- d. Padding, Border, Container, Font

La respuesta correcta es: Content, Padding, Border, Margin

**Pregunta 6**

Correcta

Se puntúa 1,00 sobre 1,00

El selector ` .menu` aplica estilos a todos los elementos con ID igual a "menu".

- Verdadero
- Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 7**

Correcta

Se puntuá 1,00 sobre 1,00

La unidad `rem` se refiere a:

- a. Al tamaño relativo al ancho del dispositivo
- b. Una medida absoluta
- c. A la fuente base del documento, sin depender del elemento padre ✓
- d. A la altura de la letra x

La respuesta correcta es: A la fuente base del documento, sin depender del elemento padre

**Pregunta 8**

Correcta

Se puntuá 1,00 sobre 1,00

La mejor forma de separar verticalmente dos elementos es utilizando el elemento `<br/>`

- Verdadero
- Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 9**

Correcta

Se puntuá 1,00 sobre 1,00

Módulos CSS 3 se divide en módulos que pueden estar en distintos estados de desarrollo.

- Verdadero ✓
- Falso

La respuesta correcta es 'Verdadero'

**Pregunta 10**

Correcta

Se puntuó 1,00 sobre 1,00

¿Cuál de estos selectores CSS es inválido?

- a. `html`
- b. `head`
- c. `body`
- d. `p.nombreClase`
- e. `<code>#123parrafo</code>` ✓

Respuesta correcta

La respuesta correcta es:

`<code>#123parrafo</code>`

[◀ Proyecto de Clase Práctica \(10/04/2025\)](#)

Ir a...

[CSS Dinner ▶](#)

Comenzado el miércoles, 23 de abril de 2025, 08:02

Estado Finalizado

Finalizado en miércoles, 23 de abril de 2025, 08:08

Tiempo 6 minutos 9 segundos  
empleado

Calificación 9,00 de 10,00 (90%)

### Pregunta 1

Correcta

Se puntúa 1,00 sobre 1,00

Son elementos semánticos

- a. <div>
- b. <span>
- c. <head>
- d. <body>
- e. Todos los anteriores
- f. Ninguno de los anteriores ✓

Respuesta correcta

La respuesta correcta es:

Ninguno de los anteriores

### Pregunta 2

Correcta

Se puntúa 1,00 sobre 1,00

¿Qué propiedades se usan para establecer filas y columnas en Grid?

- a. grid-template-columns y grid-template-rows ✓
- b. row-template y col-template
- c. template-columns y template-rows
- d. grid-positioning

La respuesta correcta es: grid-template-columns y grid-template-rows

**Pregunta 3**

Correcta

Se puntúa 1,00 sobre 1,00

Cuál de las siguientes afirmaciones de Flexbox es incorrecta

- a. Flexbox es especialmente útil en para la creación de sitios web adaptativos o responsive
- b. Antes de Flexbox se debía utilizar combinaciones de float y clear para desplegar elementos alineados horizontalmente
- c. Con Flexbox se pueden crear contenedores que dispongan los elementos vertical u horizontalmente
- d. Flexbox es la única forma de desplegar elementos de bloque en una sola fila sin tener que agregar estilos adicionales a los ítems contenidos.



Respuesta correcta

La respuesta correcta es:

Flexbox es la única forma de desplegar elementos de bloque en una sola fila sin tener que agregar estilos adicionales a los ítems contenidos.

**Pregunta 4**

Incorrecta

Se puntúa 0,00 sobre 1,00

Para alinear elementos a izquierda o derecha puedo utilizar las propiedades

- a. float
- b. flexbox y justify-content
- c. Ninguna de las anteriores
- d. position
- e. Todas las anteriores ✗

Respuesta incorrecta.

Las respuestas correctas son:

float,

flexbox y justify-content

**Pregunta 5**

Correcta

Se puntúa 1,00 sobre 1,00

¿Cuál **NO** es un tipo de posicionamiento CSS válido?

- a. fixed
- b. static
- c. external ✓
- d. relative

La respuesta correcta es: external

**Pregunta 6**

Correcta

Se puntúa 1,00 sobre 1,00

¿Qué característica tiene **position: fixed**?

- a. Se comporta como un elemento flotante.
- b. Permanece en la misma posición aunque se haga scroll. ✓
- c. Solo aplica dentro de Flexbox.
- d. Se mueve con el contenido del documento.

La respuesta correcta es: Permanece en la misma posición aunque se haga scroll.

**Pregunta 7**

Correcta

Se puntúa 1,00 sobre 1,00

¿Cuál de los siguientes elementos es de nivel de bloque por defecto?

- a. span
- b. a
- c. div ✓
- d. img

La respuesta correcta es: div

**Pregunta 8**

Correcta

Se puntúa 1,00 sobre 1,00

¿Cuál de estas afirmaciones es cierta?

- a. Grid es bidimensional y Flexbox es unidimensional. ✓
- b. Flexbox y Grid no son compatibles entre sí.
- c. Flexbox solo funciona con imágenes.
- d. Grid no necesita contenedores.

La respuesta correcta es: Grid es bidimensional y Flexbox es unidimensional.

**Pregunta 9**

Correcta

Se puntuó 1,00 sobre 1,00

Son elementos semánticos

- a. <header>
- b. <main>
- c. <section>
- d. <article>
- e. <aside>
- f. Todos los anteriores ✓
- g. Ninguno de los anteriores

Respuesta correcta

La respuesta correcta es:

Todos los anteriores

**Pregunta 10**

Correcta

Se puntuó 1,00 sobre 1,00

Los elementos semánticos indican qué contienen y no cómo deben formatearse.

- Verdadero ✓
- Falso

La respuesta correcta es 'Verdadero'

[◀ Leer Fundamentals of Web Development](#)

Ir a...

Flexbox Froggy ►

Comenzado el domingo, 18 de mayo de 2025, 14:50

Estado Finalizado

Finalizado en domingo, 18 de mayo de 2025, 14:56

Tiempo 6 minutos 14 segundos  
empleado

Calificación 9,00 de 10,00 (90%)

**Pregunta 1**

Correcta

Se puntúa 1,00 sobre 1,00

¿Cuál es la forma correcta de eliminar una propiedad de un objeto?

- a. objeto.delete("propiedad");
- b. remove objeto.propiedad;
- c. objeto.remove("propiedad");
- d. delete objeto.propiedad; ✓

La respuesta correcta es: delete objeto.propiedad;

**Pregunta 2**

Correcta

Se puntúa 1,00 sobre 1,00

¿Qué valor representa la operación `typeof NaN` en JavaScript?

- a. "NaN"
- b. "undefined"
- c. "null"
- d. "number" ✓

La respuesta correcta es: "number"

**Pregunta 3**

Correcta

Se puntuá 1,00 sobre 1,00

Para definir una variable en JavaScript las alternativas son

- a. const variable = "Hola";
- b. let variable = "Hola";
- c. variable = "Hola";
- d. Todas las anteriores. ✓
- e. Ninguna de las anteriores.

Respuesta correcta

La respuesta correcta es:

Todas las anteriores.

**Pregunta 4**

Correcta

Se puntuá 1,00 sobre 1,00

¿Qué método se usa para encontrar la primera aparición de una subcadena?

- a. find()
- b. searchIndex()
- c. indexOf() ✓
- d. charAt()

La respuesta correcta es: indexOf()

**Pregunta 5**

Correcta

Se puntuá 1,00 sobre 1,00

¿Cuál de estas funciones solicita un valor ingresado por el usuario?

- a. confirm()
- b. prompt() ✓
- c. input()
- d. alert()

La respuesta correcta es: prompt()

**Pregunta 6**

Correcta

Se puntúa 1,00 sobre 1,00

Una arrow function (función flecha) siempre debe

- a. Comenzar y finalizar con llaves {} y {}
- b. Llevar `return` en la última sentencia.
- c. Tener parámetros
- d. Todas las anteriores
- e. Ninguna de las anteriores ✓

Respuesta correcta

La respuesta correcta es:

Ninguna de las anteriores

**Pregunta 7**

Correcta

Se puntúa 1,00 sobre 1,00

Las sentencias en JavaScript siempre deben terminar en ; (punto y coma)

- Verdadero
- Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 8**

Correcta

Se puntúa 1,00 sobre 1,00

Indique el incorrecto. Las funciones de callback

- a. Permiten gestionar eventos.
- b. Pueden ser pasadas por parámetros.
- c. No pueden recibir parámetros. ✓
- d. Todas las anteriores.
- e. Ninguna de las anteriores.

Respuesta correcta

La respuesta correcta es:

No pueden recibir parámetros.

**Pregunta 9**

Incorrecta

Se puntuá 0,00 sobre 1,00

¿Qué resultado produce la siguiente desestructuración: [a,,b] = [1,2,3]; console.log(b);

- a. Error de sintaxis
- b. 3
- c. undefined
- d. 2 ✗

La respuesta correcta es: 3

**Pregunta 10**

Correcta

Se puntuá 1,00 sobre 1,00

¿Qué método de array elimina el primer elemento?

- a. splice()
- b. shift() ✓
- c. unshift()
- d. pop()

La respuesta correcta es: shift()

[◀ Ejercicio de Clase N° 1: JavaScript](#)

Ir a...

Semana 8 - Hoja de Ruta ►

Comenzado el domingo, 18 de mayo de 2025, 14:59

Estado Finalizado

Finalizado en domingo, 18 de mayo de 2025, 15:03

Tiempo 3 minutos 32 segundos  
empleado

Calificación 8,00 de 10,00 (80%)

**Pregunta 1**

Correcta

Se puntúa 1,00 sobre 1,00

Por defecto, todos los eventos en el navegador se manejan en la fase de burbujeo.

- Verdadero ✓  
 Falso

La respuesta correcta es 'Verdadero'

**Pregunta 2**

Correcta

Se puntúa 1,00 sobre 1,00

¿Cuál de estos controles se utiliza para subir archivos desde el cliente?

- a. "text">  
 b. <input type  
 c. <input type  
 d. <textarea>  
 e. <input type  
 f. "submit">  
 g. "file"> ✓

Las respuestas correctas son: "text">, <input type, "file">, "submit">

**Pregunta 3**

Correcta

Se puntuá 1,00 sobre 1,00

¿Cuál de los siguientes elementos se utiliza para ingresar texto largo en un formulario?

- a. "search">
- b. <textarea> ✓
- c. "password">
- d. <input type
- e. "text">
- f. <input type
- g. <input type

Las respuestas correctas son: "text">, "password">, <textarea>, "search">

**Pregunta 4**

Incorrecta

Se puntuá 0,00 sobre 1,00

¿Qué línea de código asigna la clase CSS “rojo” a un elemento HTML en JavaScript?

- a. elemento.className
- b. elemento.addClass("rojo");
- c. elemento.style.color
- d. "rojo";
- e. elemento.setAttribute("class", "rojo"); ✗
- f. "rojo";

Las respuestas correctas son: elemento.className, "rojo";, "rojo";

**Pregunta 5**

Correcta

Se puntuá 1,00 sobre 1,00

El servidor web es el que crea el árbol DOM para que podamos usarlo a través del objeto `document` en JavaScript

- Verdadero
- Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 6**

Correcta

Se puntuá 1,00 sobre 1,00

Una función callback es una función pasada como argumento a otra función.

 Verdadero ✓ Falso

La respuesta correcta es 'Verdadero'

**Pregunta 7**

Correcta

Se puntuá 1,00 sobre 1,00

El método addEventListener está disponible tanto para Node como para NodeList.

 Verdadero Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 8**

Correcta

Se puntuá 1,00 sobre 1,00

Un programa NodeJS siempre construye por nosotros el árbol DOM

 Verdadero Falso ✓

La respuesta correcta es 'Falso'

**Pregunta 9**

Correcta

Se puntuá 1,00 sobre 1,00

El uso de innerHTML puede traer riesgos de seguridad y errores en la estructura HTML.

 Verdadero ✓ Falso

La respuesta correcta es 'Verdadero'

**Pregunta 10**

Incorrecta

Se puntuó 0,00 sobre 1,00

El árbol DOM está formado por todos los componentes de un navegador web

 Verdadero × Falso

La respuesta correcta es 'Falso'

[◀ Leer: Fundamentals of Web Development](#)

[Ir a...](#)

[Event Propagation, Capturing, and Bubbling in Javascript ►](#)

Comenzado el jueves, 29 de mayo de 2025, 20:01

Estado Finalizado

Finalizado en jueves, 29 de mayo de 2025, 20:05

Tiempo 4 minutos 28 segundos  
empleado

Calificación 10,00 de 10,00 (100%)

### Pregunta 1

Correcta

Se puntúa 1,00 sobre 1,00

¿Qué retorna el método `reduce` aplicado a un array?

- a. Un único valor, resultado de aplicar una función acumuladora. ✓
- b. Un nuevo array con los valores transformados.
- c. El primer elemento que cumpla la condición dada.
- d. Un array filtrado con los elementos únicos.

La respuesta correcta es: Un único valor, resultado de aplicar una función acumuladora.

### Pregunta 2

Correcta

Se puntúa 1,00 sobre 1,00

¿Qué es un error CORS en una solicitud HTTP?

- a. Una restricción de seguridad que impide compartir recursos entre orígenes distintos. ✓
- b. Un conflicto de nombres entre formularios HTML.
- c. Un error en la codificación UTF-8 de los datos.
- d. Un error que impide utilizar JSON como formato de respuesta.

La respuesta correcta es: Una restricción de seguridad que impide compartir recursos entre orígenes distintos.

### Pregunta 3

Correcta

Se puntúa 1,00 sobre 1,00

¿Por qué las Promises son preferidas sobre callbacks anidados?

- a. Porque eliminan por completo la asincronía.
- b. Porque permiten uso de eventos DOM directamente.
- c. Porque no necesitan funciones anónimas.
- d. Porque ayudan a evitar el "callback hell" y hacen el código más legible. ✓

La respuesta correcta es: Porque ayudan a evitar el "callback hell" y hacen el código más legible.

**Pregunta 4**

Correcta

Se puntúa 1,00 sobre 1,00

¿Qué condición debe cumplirse para poder usar `await` en una función?

- a. Debe estar dentro de una expresión ternaria.
- b. Debe estar dentro de una función declarada con la palabra clave `async`. ✓
- c. Debe estar dentro de un callback tradicional.
- d. Debe ser parte de una clase con herencia.

La respuesta correcta es: Debe estar dentro de una función declarada con la palabra clave `async` .

**Pregunta 5**

Correcta

Se puntúa 1,00 sobre 1,00

¿Cuál es la forma correcta de capturar errores de red al usar `fetch()`?

- a. Usar el método `.catch()` en la Promise devuelta. ✓
- b. Comprobar que el JSON recibido tenga más de una propiedad.
- c. Encerrar `fetch()` en un bloque `try/catch` sin `await`.
- d. Validar la URL con una expresión regular.

La respuesta correcta es: Usar el método `.catch()` en la Promise devuelta.

**Pregunta 6**

Correcta

Se puntúa 1,00 sobre 1,00

¿Qué devuelve la función `fetch()` en JavaScript moderno?

- a. Un objeto Promise que se resuelve con la respuesta de la solicitud. ✓
- b. Un objeto JSON con los datos obtenidos.
- c. Un objeto XMLHttpRequest ya configurado.
- d. Un string con el HTML completo de la página.

La respuesta correcta es: Un objeto Promise que se resuelve con la respuesta de la solicitud.

**Pregunta 7**

Correcta

Se puntúa 1,00 sobre 1,00

¿Cuál es la principal diferencia entre los métodos `find` y `filter` de los arrays en JavaScript?

- a. `find` devuelve el primer elemento que cumple con la condición, mientras que `filter` devuelve todos los elementos que cumplen.
- b. `find` modifica el array original, `filter` no.
- c. `filter` solo funciona con números, `find` con cualquier tipo.
- d. `find` ordena el array, `filter` lo invierte.



La respuesta correcta es: `find` devuelve el primer elemento que cumple con la condición, mientras que `filter` devuelve todos los elementos que cumplen.

**Pregunta 8**

Correcta

Se puntúa 1,00 sobre 1,00

¿Cuál es el principal propósito de las clases introducidas en ES6 en JavaScript?

- a. Reemplazar funciones tradicionales.
- b. Proveer una sintaxis más amigable para trabajar con prototipos.
- c. Evitar el uso de eventos.
- d. Eliminar completamente el uso de prototipos.

La respuesta correcta es: Proveer una sintaxis más amigable para trabajar con prototipos.

**Pregunta 9**

Correcta

Se puntúa 1,00 sobre 1,00

¿Cuál de los siguientes métodos se utiliza para iterar sobre cada elemento de un array ejecutando una función proporcionada?

- a. reduce
- b. sort
- c. forEach
- d. console.log

La respuesta correcta es: forEach

**Pregunta 10**

Correcta

Se puntuó 1,00 sobre 1,00

¿Qué propósito tiene la palabra clave `super` en el constructor de una clase hija?

- a. Importar funciones externas.
- b. Declarar una nueva propiedad.
- c. Referenciar al objeto global.
- d. Invocar el constructor de la clase padre. ✓

La respuesta correcta es: Invocar el constructor de la clase padre.

[\*\*◀ Ejemplos\*\*](#)[Ir a...](#)[loading.io ►](#)