

## Unidad 1

# Introducción a las aplicaciones Web y HTML5



Universidad Tecnológica Nacional  
Facultad Regional Córdoba  
Cátedra: Programación de Aplicaciones Visuales 2

# Arquitectura de n-capas de una aplicación web



La Arquitectura de N-capas es probablemente uno de los modelos más utilizados en programación. Se utiliza tan a menudo porque es escalable, extensible, segura, fácil de mantener en el tiempo, reutilizable y se puede trabajar por diferentes programadores según su especialidad sin interferirse el trabajo.

Para el desarrollo de una arquitectura en capas robusta y perdurable en el tiempo, se deben tener en cuenta los siguientes aspectos:

- Las capas de una aplicación deben ser independientes entre sí, como las fichas de un lego. De esta manera podemos reemplazar cualquiera de las capas por una más actualizada en cualquier momento.
- Se debe mantener el principio de responsabilidad modular, esto quiere decir que cada capa es responsable de un tema o característica.
- Principio de caja negra. Cada componente es independiente de otros y solo interactúa en sus entradas y salidas.
- Don't Repeat Yourself (DRY). Principio de No repitas. Cada funcionalidad debe estar una sola vez en el sistema (Sin duplicados).
- Establecer normas en la codificación del desarrollo. Esto asegura la consistencia de código y el mantenimiento del mismo.

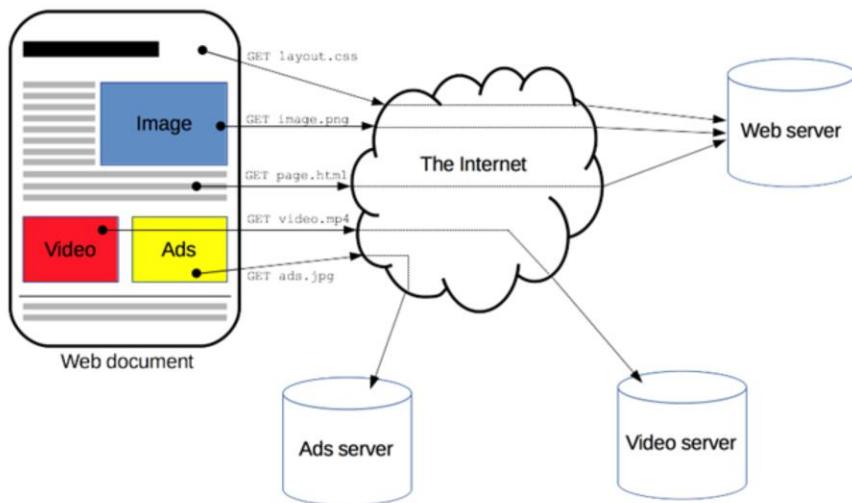
## Ventajas

Siendo la programación en capas una arquitectura que tiene como objetivo principal separar la capa de diseño de la de lógica, existen otros valores adicionales que

aportan a proyectos grandes entre otros como:

- Facilita la creación de código estructurado
- Facilita el trabajo en equipo, permitiendo que varios programadores trabajen paralelamente en una misma funcionalidad sin interferirse mutuamente.
- Permite separar algunas de las capas de código en diferentes servidores para balanceo de cargas, buses de datos empresariales, etc.

# ¿Qué es una aplicación web SPA?



## SPA, un paradigma de Arquitectura de Aplicaciones Web en auge

Dentro del desarrollo de aplicaciones web hay una tendencia importante a las denominadas SPA, Single Page Apps. Uno de los principales objetivos es conseguir una mejora importante en la experiencia de usuario: La mejora de los tiempos de espera o latencia entre vistas (similar a las aplicaciones nativas).

En esencia los motivos de las ventajas de las SPAs son sencillos, aunque creo que conviene ponerlo un poco en contexto.

Un SPA se carga completo durante el cargado de la página inicial y luego las regiones se reemplazan o se actualizan con los fragmentos de las nuevas páginas según petición del servidor. Para evitar descargas excesivas de características inutilizadas, un SPA descarga progresivamente las características cuando se necesiten, pueden ser fragmentos de las páginas o módulos completos de la pantalla.

De esta forma existe una analogía entre los "estados" de un SPA y las "páginas" de un sitio web tradicional. Como la navegación de estados en la misma página es análogo a la navegación de las páginas, en teoría, cualquier página de sitio web podría convertirse a un sitio de página única reemplazando las páginas solamente en las partes donde se generen un cambio. El enfoque de SPA en la página web es similar a Single Document Interface (SDI) que es una técnica para las aplicaciones de escritorio.

## **¿Qué diferencia hay respecto a aplicaciones web clásicas?**

Habitualmente la lógica de negocio (el código ejecutable) de aplicaciones web se realiza íntegramente en el lado del servidor, y se confía la propia naturaleza del sistema de URLs el mostrar una “vista de aplicación” u otra. Por ej. en un e-commerce el carrito estará en una URL concreta, mientras que la pantalla de login estará bajo otra URL totalmente diferente. Para el navegador, cada URL diferente es completamente independiente del resto: Aunque tenga los mismos estilos y/o plantillas, estos tienen que volver a ser procesados desde cero. Esto, para la gran mayoría de páginas web dinámicas, implica que al cambiar entre vistas se sufrirá el problema de la latencia en la web (artículo de Ilya Grigorik, developer advocate de Google).

Otra característica negativa de esta arquitectura es que el estado de la aplicación del cliente es difícil de mantener, teniendo que hacer auténticos malabarismos para poder gestionar una simple transferencia de información de una vista a otra: Bien lo sabe aquel programador que haya tenido que implementar el típico asistente que se compone de diversas vistas, por ejemplo.

¿Podríamos encontrar más pegas? Puede, pero no es el objetivo. Es más, el sistema REST y URLs tiene mucho sentido y aporta muchas otras ventajas desde la misma arquitectura de servicios web hasta el facilitar el lado humano de compartir un recurso vía un enlace, aunque en ciertos aspectos juegue en nuestra contra.

## **Arquitectura básica de una SPA**

En esencia una SPA es la interfaz de la aplicación web implementada casi íntegramente en el navegador (en lenguaje Javascript actualmente), aunque como toda página web tenga una base importante de HTML y CSS.

Todas las vistas de la interfaz de la aplicación están contenidas en la SPA, realizando una única carga inicial y potencialmente solo posponiendo recursos pesados: Grandes cantidades de datos, Imágenes, videos, ...

Por tanto con las SPAs eliminamos por completo uno de los principales cuellos de botella: El problema de la latencia (ver artículo de Grigorik). Así podemos conseguir que la aplicación web alcance la velocidad de cualquier aplicación nativa en lo que se refiere al tiempo de espera al cambiar de vistas. Esto se aprecia notablemente sea cual sea el dispositivo, aunque especialmente cuando se utilizan dispositivos móviles.

A su vez, podemos ahorrar mucho ancho de banda y tiempo de proceso de cálculo (en servidor y cliente) si gestionamos el estado de la aplicación desde el cliente con una SPA: Por ejemplo, la SPA del típico e-commerce al cambiar de vista podría mantener en memoria el estado del carrito de compras y no tener que transferir constantemente esa información en el servidor (por supuesto esa información conviene que esté también en el servidor, pero una cosa no está reñida con la otra).

## **HTML5 y SPAs**

Sin lugar a dudas técnicamente se pueden crear SPAs bajo los estándares de HTML4, por tanto dando cobertura a la totalidad de navegadores utilizados hoy en día que tengan soporte de HTML y Javascript.

Sin embargo, algunas APIs incorporadas en el marco de HTML5 aportan soluciones a ciertos retos bastante específicos de las SPAs:

**HTML5 History API:** Las URLs clásicas eran un problema para las SPAs hasta cierto punto, aunque eso ya no ocurre gracias a esta API. Esto a su vez simplificaría en buena medida el problema de “Search Engine Optimisation” SEO (a pesar de otros intentos de Google como el AJAX Crawling) ya que con PhantomJS en tu servidor puedes renderizar una URL concreta de tu SPA.

**HTML5 Application Cache:** Si consigues sacarle partido a esta API puedes conseguir que tu aplicación funcione incluso cuando el dispositivo cliente está sin conexión a Internet. Toda una killer feature de las SPA como bien apunta Jakub Mrozek en su charla para la WebExpo 2013.

## Herramientas

Hay muchas herramientas para desarrollar SPA. Por mencionar alguna, AngularJS tiene entre otros muchos objetivos crear aplicaciones SPA y a su vez mantener las ventajas del sistema de URLs.

Como este tipo de aplicaciones son bastante complejas (comparado con el uso previo de Javascript), es mejor realizarlas de forma estructurada a partir de un framework como Angular, siguiendo paradigmas de diseño de software similares a MVC.

# HTML

## Introducción

### Un poco de contexto

Tim Berners-Lee creó el HTML original en 1989 para solucionar las deficiencias de los métodos existentes para acceder a información en Internet. Desde que se concibió, encontrar su camino en Internet era una tarea difícil. El contenido en Internet era tratado como documentos individuales, sin que hubiesen métodos sencillos para navegarlos. En esencia, usted tenía que conocer la dirección del documento que estaba buscando e ingresarla manualmente. Para solucionar este problema, Berners-Lee creó dos tecnologías: Hypertext Transfer Protocol (HTTP) y HTML.

HTTP es un protocolo de servicio utilizado para entregar contenido. El comienzo de un URL en su navegador Web (suponiendo que el navegador muestre el URL completo) muy probablemente comenzará con `http://`. Esta parte del URL le dice al navegador qué tipo de protocolo usar cuando esté haciendo la solicitud al servidor Web. Cuando el servidor recibe una solicitud de documento, es probable que ese documento esté escrito o sea convertido a HTML. El documento HTML es lo que se envía de regreso al navegador que hace la solicitud.

HTML es un lenguaje de scripting que le dice al navegador Web cómo presentar el contenido. En este contenido puede haber enlaces a otros documentos, proporcionando un método fácil de usar para navegar entre documentos en Internet.

La combinación de HTTP y HTML ofrece una navegación rápida y fácil por el contenido en Internet, al permitirle simplemente hacer clic en los enlaces de texto para navegar entre documentos. Después de crear estas dos tecnologías, Berners-Lee continuó y fundó el World Wide Web Consortium (W3C). El W3C fue la fuerza guía detrás de las cuatro primeras versiones de HTML.

La intención original de Internet era servir documentos de texto simples. Los primeros navegadores todos estaban basados en texto (sin ventanas lujosas — sólo texto en una pantalla). Incluso la adición de imágenes era un gran problema cuando se introdujo al principio. Ahora, las personas hacen de todo, desde enviar mensajes de e-mail hasta ver televisión en Internet. Internet se ha convertido en mucho más que un mecanismo para transportar documentos de texto simples. Con los recursos y usos llegaron nuevos retos y problemas que el lenguaje HTML nunca fue diseñado para manejar.

El W3C intentó resolver los problemas del Internet de hoy con el estándar Extensible Hypertext Markup Language (XHTML) 2.0. Sin embargo, este estándar no fue bien recibido y fue abandonado en gran medida. En el 2004, mientras el W3C se estaba enfocando en el estándar XHTML 2.0, un grupo llamado el Web Hypertext Application Technology Working Group (WHATWG) comenzó a trabajar en el estándar HTML5, que tuvo una acogida más cálida que el estándar XHTML 2.0. El W3C abandonó el estándar XHTML 2.0 y está trabajando ahora con WHATWG en el desarrollo del HTML5.1.

# ¿Qué es HTML ?

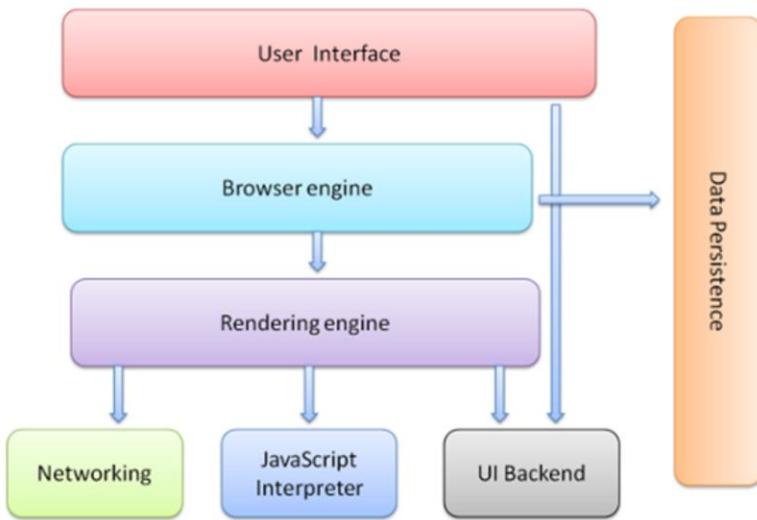
Es el bloque de construcción más básico de la web

- Describe y define el contenido de una página web
- Para describir la apariencia de una página se utilizan hojas de estilo CSS
- Para agregarle funcionalidad se utiliza el lenguaje JavaScript
- HyperText se refiere a vínculos o links que conectan una página a otra
- Utiliza un lenguaje de marcas o “markup” para especificar el texto a mostrar, imágenes, videos y cualquier otro contenido para mostrar en el navegador web.

## ¿Qué es un servidor web?

- Es un programa que se ejecuta en un equipo servidor que procesa peticiones HTTP generalmente por el puerto 80 y devuelve documentos HTML.
- Posee una estructura de directorios y archivos que constituyen un sitio web
- Los servidores web más utilizados son:
  - Internet Information Services (IIS10),
  - Apache, NGINX, GWS
- Lenguajes más utilizados
  - ASP.NET Web/API, MVC, WebForms
  - PHP
  - Java
  - JavaScript/TypeScript
  - Python

# Navegadores web - componentes



Interesante presentación del funcionamiento del navegador

[https://www.youtube.com/watch?v=A4oNo\\_nhYA](https://www.youtube.com/watch?v=A4oNo_nhYA)

La función principal de un navegador es solicitar al servidor los recursos web que elija el usuario y mostrarlos en una ventana. El recurso suele ser un documento HTML, pero también puede ser un archivo PDF, una imagen o un objeto de otro tipo. El usuario especifica la ubicación del recurso mediante el uso de una URI (siglas de Uniform Resource Identifier, identificador uniforme de recurso).

La forma en la que el navegador interpreta y muestra los archivos HTML se determina en las especificaciones de CSS y HTML. Estas especificaciones las establece el consorcio W3C (World Wide Web Consortium), que es la organización de estándares de Internet.

Durante años, los navegadores cumplían solo una parte de las especificaciones y desarrollaban sus propias extensiones. Esto provocó graves problemas de compatibilidad para los creadores de contenido web. En la actualidad, la mayoría de los navegadores cumplen las especificaciones en mayor o menor grado.

Fuente: <https://www.html5rocks.com/es/tutorials/internals/howbrowserswork/>

**Los componentes principales de un navegador son:**

1. **Interfaz de usuario:** incluye la barra de direcciones, el botón de

avance/retroceso, el menú de marcadores, etc. (en general, todas las partes visibles del navegador, excepto la ventana principal donde se muestra la página solicitada).

2. **Motor de búsqueda:** coordina las acciones entre la interfaz y el motor de renderización.
3. **Motor de renderización:** es responsable de mostrar el contenido solicitado. Por ejemplo, si el contenido solicitado es HTML, será el responsable de analizar el código HTML y CSS y de mostrar el contenido analizado en la pantalla.
4. **Red:** es responsable de las llamadas de red, como las solicitudes HTTP. Tiene una interfaz independiente de la plataforma y realiza implementaciones en segundo plano para cada plataforma.
5. **Servidor de la interfaz:** permite presentar widgets básicos, como ventanas y cuadros combinados. Muestra una interfaz genérica que no es específica de ninguna plataforma. Utiliza métodos de la interfaz de usuario del sistema operativo en segundo plano.
6. **Intérprete de JavaScript:** permite analizar y ejecutar el código JavaScript.
7. **Almacenamiento de datos:** es una capa de persistencia. El navegador necesita guardar todo tipo de datos en el disco duro (por ejemplo, las cookies). La nueva especificación de HTML (HTML5) define el concepto de "base de datos web", que consiste en una completa (aunque ligera) base de datos del navegador.

## El motor de renderización

La responsabilidad del motor de renderización es "renderizar", es decir, mostrar el contenido solicitado en la pantalla del navegador.

De forma predeterminada, el motor de renderización puede mostrar imágenes y documentos HTML y XML. Puede mostrar otros tipos mediante el uso de complementos (o extensiones); por ejemplo, puede mostrar documentos PDF mediante un complemento capaz de leer archivos PDF.

## Motores de renderización

Nuestros navegadores de referencia (Firefox, Chrome y Safari) están basados en dos motores de renderización. Firefox utiliza Gecko, un motor de renderización propio de Mozilla. Tanto Safari como Chrome utilizan WebKit.

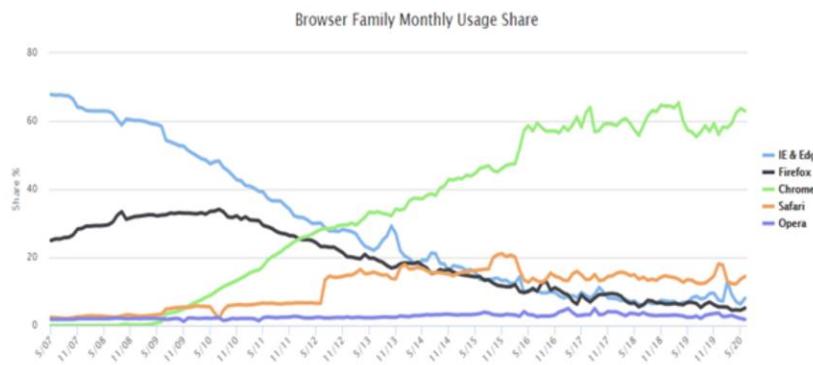
**Trident** es el nombre del motor de renderizado propietario usado por Microsoft

Internet Explorer 11 para Windows.

**EdgeHTML** es un motor de renderizado desarrollado por Microsoft para el navegador Microsoft Edge.

**WebKit** es un motor de renderización de código abierto que empezó siendo un motor de la plataforma Linux y que fue modificado posteriormente por Apple para hacerlo compatible con Mac y Windows.

# Navegadores de internet y resolución de pantalla



Valores a junio 2020. Fuente w3counter.com



Top 10 Screen Resolutions

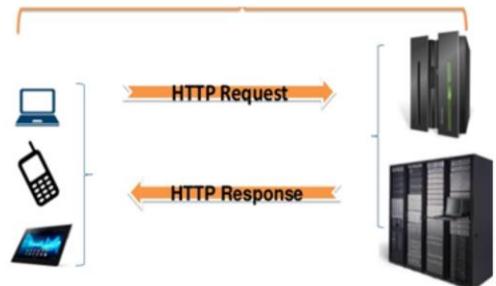
1	1366x768	11.98%
2	640x360	11.79%
3	1920x1080	6.98%
4	667x375	4.65%
5	1024x768	4.25%
6	760x360	3.33%
7	896x414	3.27%
8	812x375	3.23%
9	780x360	3.11%
10	1536x864	3.03%

## Demo: Cuál es el uso de navegadores en Argentina

- Ingrese a <http://gs.statcounter.com> para ver el mercado de navegadores en argentina y la resolución de pantalla utilizadas

# HTTP - Protocolo

- Especifica las reglas de comunicación entre un cliente web y un servidor web
- Protocolo orientado a transacciones
- Sigue el esquema request-response
- El cliente web se denomina user agent
- La petición (request) se realiza enviando un mensaje con un determinado formato. Dicho formato especifica la acción
- La respuesta (response) contiene el documento, archivo, imagen solicitado



# Protocolo HTTP (Hypertext Transfer **protocol**)

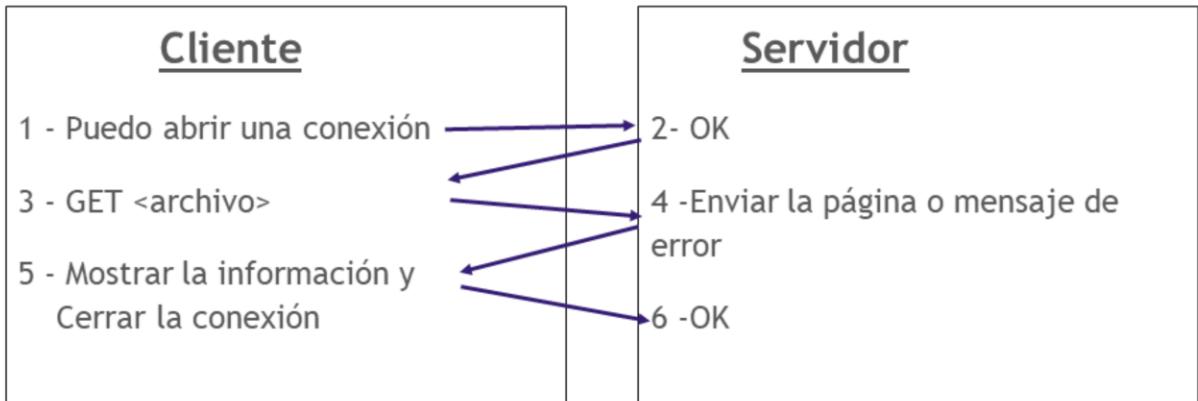
**Stateless:** cada transacción entre el cliente y el servidor es independiente, es decir, no guarda ninguna información sobre conexiones anteriores.

Para mantener el estado de la aplicaciones web se usan las cookies, que permiten instituir la noción de sesión, y también permite rastrear usuarios

HTTP ha sido diseñado como un **protocolo sin estado** (stateless protocol) lo que significa que cada solicitud (request) o respuesta (Response) es una transacción independiente

Las peticiones al servidor se denominan **request** y las respuestas del servidor **responses**

## Una conversación HTTP

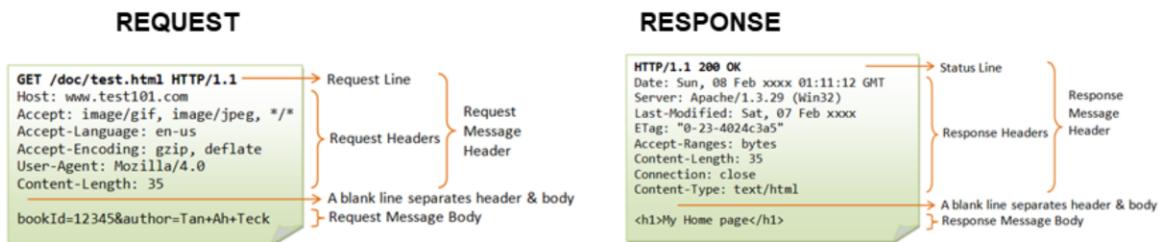


**HTTP** es el conjunto de reglas que gobiernan el formato y el contenido de una conversación entre un cliente web y un servidor.

# Mensaje HTTP

Un mensaje del tipo **request** y **response** consta de la siguiente información:

1. Una línea de inicio
2. Opcionalmente una o más cabeceras del mensaje
3. Opcionalmente el cuerpo del mensaje



## Métodos o verbos HTTP

- **GET:** obtiene el recurso especificado por la URL.
- **POST:** envía o somete datos para que sean procesados por el recurso identificado en la URL.
- **PUT:** sube o actualiza un recurso especificado.
- **DELETE:** Borra el recurso especificado.
- **PATCH:** solicita la actualización parcial de un recurso
- **HEAD:** Obtiene sólo las cabeceras de una petición. Es un verbo similar a GET, sólo que GET además de las cabeceras también obtiene el cuerpo de la respuesta.
- **TRACE:** este método se emplea con fines de diagnóstico ya que solicita al servidor que envíe todos los datos de la solicitud enviada.
- **OPTIONS:** sirve para obtener una lista de los métodos HTTP que soporta el servidor
- **CONNECT:** se utiliza para saber si tenemos acceso a un determinado host.

# Códigos de estado HTTP

Algunos códigos de estado son:

- 2xx - Successful response
  - 200 OK
  - 201 Created
- 3xx - Redirect response
  - 301 Moved permanently
- 4xx - Client error Response
  - 401 Unauthorized
  - 403 Forbidden
  - 404 File not found
- 5xx - Service error response
  - 500 Server Error
  - 503 Service unavailable

## Significados de código en función del número de partida

El primer dígito de cada código de tres dígitos comienza con uno de los números del 1 al 5. Existen los siguientes códigos de estado de 100 a 500:

Los códigos en los 100s son informativos e indican que la solicitud fue recibida y el proceso continúa. Los códigos en los 200s indican la recepción de la solicitud y su procesamiento exitoso. 300s indican la recepción de una solicitud, pero la necesidad de llevar a cabo un paso más para que la solicitud se ha completado. Los 400s simbolizan error del cliente en que la solicitud fue hecha pero la página ha perdido su validez. El 500 indican el error del servidor - hubo una solicitud válida desde el cliente, pero servidor no pudo completar la misma.

## Códigos 200 y 301

Código 200 simboliza la solicitud correcta y es la correcta para la mayoría de los escenarios. El código 301 indica que "se trasladó de forma permanente". El recurso se ha dado una nueva dirección URL que es permanente. referencias futuras deben utilizar una de las URL que fueron devueltos. La redirección 301 debe ser utilizado en cualquier momento una URL debe ser redirigido a una URL diferente.

## Código 302

El código 302 indica que el servidor está respondiendo a la solicitud con una página ubicación diferente.

Sin embargo, el solicitante mantiene mediante la dirección original para futuras peticiones. 302 no es favorable, ya que los rastreadores de motores de búsqueda tendrá en cuenta la redirección como un arreglo temporal y no proporcionar a la página con el poder de los enlaces de 301 redirecciones.

## Códigos 404 y 200

El código 404 indica archivo no encontrado. El servidor no pudo igualar la URL de la solicitud. No hay ninguna indicación del estado de la enfermedad, ya sea permanente o temporal. Esto ocurre cuando el servidor no puede encontrar una solicitud de página que se pongan en venta. Webmasters suelen mostrar texto de error 404 cuando el código de respuesta es de 200. Esto le da a los rastreadores de motores de búsqueda de la indicación de que la página se procesa de forma correcta. A veces, la página web está indexado erróneamente.

## Códigos 410 y 404

El código 410 indica que la página no existe y no se dispone de más en el servidor y también que no hay una dirección de reenvío conocido. Esto es más o menos una situación permanente. Los clientes que son capaces de modificar el vínculo necesario suprimir toda referencia a la URL de solicitud tras la aprobación del usuario. Si el servidor no puede determinar el recurso, se debe utilizar el código de estado 404.

## código 503

El código de estado 503 indica la incapacidad del servidor de la concesión de la dirección URL solicitada por el mantenimiento o la sobrecarga temporal del servidor. 503 debe ser utilizado en el caso de una interrupción temporal. Se da la indicación de los motores de búsqueda que el sitio es sólo temporal hacia abajo.

## Consejos para recordar

Básicamente aquí hay algunos consejos que es absolutamente necesario recordar - 301 redirecciones deben utilizarse en lugar de 302 redirecciones al redirigir las direcciones URL. páginas web que regresan 404 durante largos períodos de tiempo y que tienen vínculos valiosos deben ser redirigidos a otras páginas con el código 301. Cuando los visitantes solicitan páginas que regresan un código de respuesta 404 debe haber personalizado 404 páginas con opciones de navegación.

[Para ver más información](#)

# HTML - Versiones

Versión	Año
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2014

## Diferentes versiones de HTML y CSS

Con el tiempo, HTML y CSS han evolucionado. La primera versión de HTML (HTML 1.0) ni siquiera ofrecía la posibilidad de mostrar las imágenes.

Una muy breve historia de estos lenguajes para conocimiento general:

HTML 1: fue la primera versión, creada por Tim Berners-Lee en 1991.

HTML 2: En 1995 se publica el estándar HTML 2.0. A pesar de su nombre, HTML 2.0 es el primer estándar oficial de HTML, es decir, el HTML 1.0 no existió como estándar. HTML 2.0 no soportaba tablas. Se simplificaba al máximo la estructura del documento para agilizar su edición, donde la declaración explícita de los elementos body, html y head es opcional.

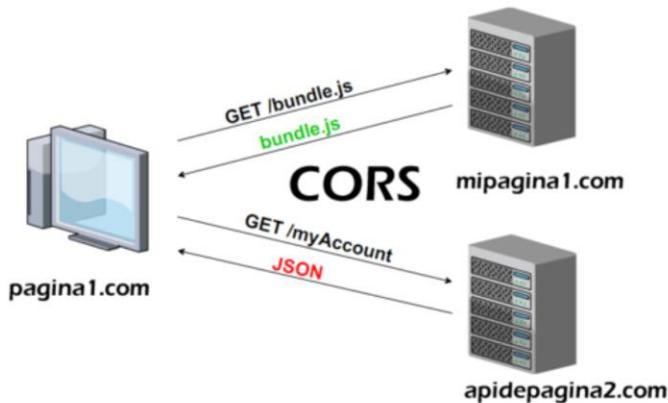
HTML 3.0, esta es la versión que en realidad plantea las bases de las siguientes versiones de HTML. Las reglas y el funcionamiento de esta versión están dadas por el W3C. HTML 3: apareció en 1996, esta nueva versión de HTML, añade muchas posibilidades al lenguaje como tablas, applets, scripts, posicionamiento de texto alrededor de imágenes, etc.

HTML 4: Esta es la versión más común de HTML (en concreto, es HTML 4.01). Apareció por primera vez en 1998 y propone el uso de marcos (que dividen una

página web en varias partes), tablas más complejas, mejoras en las formas, etc. Más importante aún, esta versión permite por primera vez utilizar hojas de estilo del famoso CSS. La última especificación oficial de HTML se publicó en diciembre de 1999 y se denomina HTML 4.01. Desde la publicación de HTML 4.01, el W3C se centró en el desarrollo del estándar XHTML. Por este motivo, en el año 2004, las empresas Apple, Mozilla y Opera mostraron su preocupación por la falta de interés del W3C en HTML y decidieron organizarse en una nueva asociación llamada WHATWG (Web Hypertext Application Technology Working Group) que comenzó el desarrollo del HTML 5, cuyo primer borrador oficial se publicó en enero de 2008. Debido a la fuerza de las empresas que forman el grupo WHATWG y a la publicación de los borradores de HTML 5.0, en marzo de 2007 el W3C decidió retomar la actividad estandarizadora de HTML, dentro del cual decidió integrar el XHTML.

HTML 5: El consorcio internacional W3C, después de una evolución de varios años, liberó el HTML5 como estándar oficial a finales de octubre de 2014. HTML5 incorpora nuevos elementos no contemplados en HTML 4.01. Hay diversos cambios respecto a HTML 4.01. Hay nuevas etiquetas, se introduce la posibilidad de introducir audio y video de forma directa en la web sin necesidad de plugins o complementos en los navegadores, entre otras novedades

## CORS: intercambio de recursos de origen cruzado



## CORS: qué es y cómo funciona el cross-origin resource sharing

Cuando se abre una página web, cargar datos de servidores ajenos está, en teoría, estrictamente prohibido. Sin embargo, puede haber excepciones: si los administradores de ambas webs han acordado trabajar juntos, no hay por qué evitar el intercambio. En estos casos, el llamado cross-origin resource sharing (CORS) regula la colaboración.

### ¿Cómo funciona el CORS?

La **same-origin policy** (SOP o política de seguridad del mismo origen) prohíbe que se carguen datos de servidores ajenos al acceder a una página web. **Todos los datos deben provenir de la misma fuente, es decir, corresponder al mismo servidor.** Se trata de una medida de seguridad, ya que JavaScript y CSS podrían cargar, sin que el usuario lo supiese, contenido de otros servidores (y, con este, también contenido malicioso). Tales intentos son denominados “cross-origin requests”. Si, por el contrario, ambos administradores web saben del intercambio de contenido y lo aprueban, no tiene sentido impedir este proceso. El servidor solicitado (es decir, aquel del que se quiere

cargar contenido) puede permitir entonces el acceso mediante cross-origin resource sharing, en castellano, **intercambio de recursos de origen cruzado**.

Este permiso se da, no obstante, únicamente a clientes concretos, es decir, el CORS no es un comodín para realizar cualquier cross-origin request. En lugar de eso, el segundo servidor permite al primero un acceso exclusivo mediante una cabecera HTTP. En dicha **cabecera de la respuesta HTTP** está indicado específicamente qué servidores pueden cargar datos y ponerlos a disposición del usuario. El acceso generalizado a todos los clientes se permite únicamente mediante una “wildcard” o certificado comodín. Esta solución, sin embargo, solo es conveniente para servidores cuyo contenido debe estar a disposición del público general, como es el caso, por ejemplo, de las tipografías web.

Si todo sale bien, el usuario no se percatará en absoluto del intercambio entre ambos servidores. Todos los **navegadores actuales** soportan el CORS, y el envío de solicitudes y respuestas sucede rápidamente al solicitar una página web sin que el usuario lo note.

## Estructura de la CORS header

De acuerdo con la política de seguridad del mismo origen, en una conexión entre servidores, los datos referentes al origen se componen de tres elementos: **host, puerto y protocolo**. De este modo, y tomando el ejemplo de la imagen, la directriz prohíbe que '<https://example.com>' acceda a '<http://example.com>' o a '<https://example.org>'. En el primer caso, **el protocolo no es el mismo** y, en el segundo, los **datos de host no coinciden**.

Una petición de origen cruzado es, en teoría, una **petición HTTP**. Los métodos específicos no suelen dar problemas. GET y HEAD no pueden alterar datos y, por lo tanto, no suelen considerarse como un riesgo para la seguridad. No se puede decir lo mismo de PATCH, PUT o DELETE: con ellos sí se puede llevar a cabo acciones maliciosas, por lo que en estos casos también hay que activar el cross-origin resource sharing, ya que CORS no solo puede contener información sobre el origen permitido, sino también acerca de qué peticiones HTTP están permitidas por la fuente.

Si se trata de métodos HTTP de seguridad, el cliente envía en primer lugar una **solicitud preflight**(preflight request) en la que solo se indica qué método HTTP se piensa transmitir al servidor a continuación y se pregunta si la solicitud es considerada segura. Para ello, se usa la cabecera OPTIONS (OPTIONS header). Una vez se haya recibido una respuesta positiva, ya se puede realizar la solicitud propiamente dicha.

Existen diferentes cabeceras o CORS headers y cada una aborda un aspecto distinto. Ya hemos mencionado dos cabeceras importantes para identificar orígenes seguros y métodos permitidos, pero hay más:

- **Access-Control-Allow-Origin**: ¿qué origen está permitido?
- **Access-Control-Allow-Credentials**: ¿también se aceptan solicitudes cuando el modo de credenciales es *incluir (include)*?
- **Access-Control-Allow-Headers**: ¿qué cabeceras pueden utilizarse?
- **Access-Control-Allow-Methods**: ¿qué métodos de petición HTTP están permitidos?
- **Access-Control-Expose-Headers**: ¿qué cabeceras pueden mostrarse?
- **Access-Control-Max-Age**: ¿cuándo pierde su validez la solicitud preflight?
- **Access-Control-Request-Headers**: ¿qué header HTTP se indica en la solicitud preflight?
- **Access-Control-Request-Method**: ¿qué método de petición HTTP se indica en la solicitud preflight?
- **Origin**: ¿de qué origen proviene la solicitud?

El primer header es especialmente importante, ya que especifica desde qué otro host se puede acceder al servidor solicitado. Además de una dirección concreta, en dicha cabecera también se puede incluir una **wildcard** en forma de asterisco. De esta manera, el servidor permitirá cross-origin requests de cualquier origen.

## Ejemplo de cross-origin resource sharing

En el siguiente ejemplo, suponemos que el host A (example.com) quiere enviar una petición DELETE (DELETE request) al host B (example.org). Para ello, el servidor A envía, en primer lugar, una preflightrequest:

```
/OPTIONS  
Origin: http://example.com  
Access-Control-Request-Method: DELETE
```

Si el host B no se opone a esta cross-origin request, responderá con los CORS headers correspondientes:

```
Access-Control-Allow-Origin: http://example.com  
Access-Control-Allow-Methods: PUT, POST, DELETE
```

Si las cabeceras de la respuesta no correspondiera con las especificaciones de la

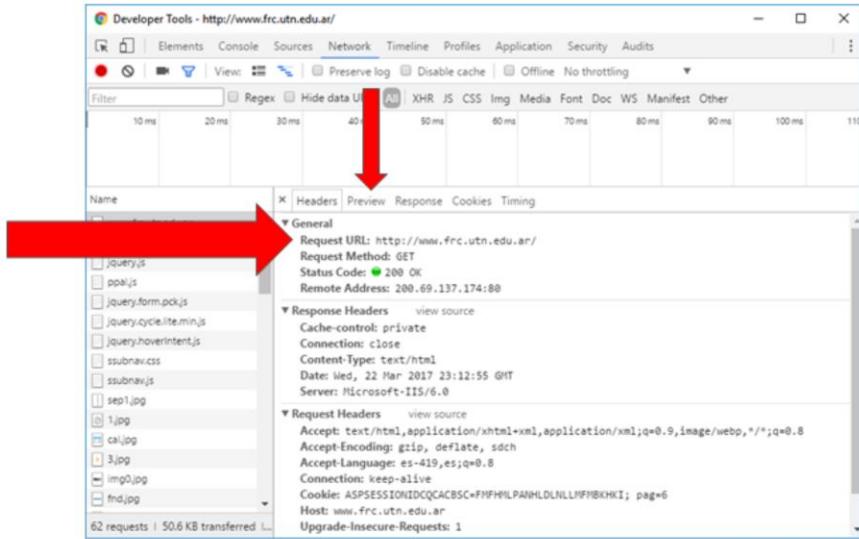
solicitud, o si el servidor solicitado no respondiese, la cross-origin request no se podría realizar.

## Ventajas e inconvenientes del CORS

El propósito del CORS es eludir la medida de seguridad establecida como configuración predeterminada (la política de seguridad del mismo origen). Dicha política es, de hecho, un medio muy eficaz para **bloquear conexiones potencialmente peligrosas**. Internet, sin embargo, se basa a menudo precisamente en este tipo de cross-origin requests, ya que muchas de las conexiones entre hosts sí son deseadas.

Por eso, el CORS ofrece una solución intermedia, permitiendo hacer excepciones a la prohibición en aquellas situaciones en que las solicitudes de origen cruzado son expresamente requeridas. No obstante, se corre el riesgo de que los administradores web se aprovechen de las wildcards por comodidad, haciendo que la protección de la SOP sea en vano. Por eso, es importante utilizar el CORS **solo en casos especiales** y configurarlo de la manera más restrictiva posible.

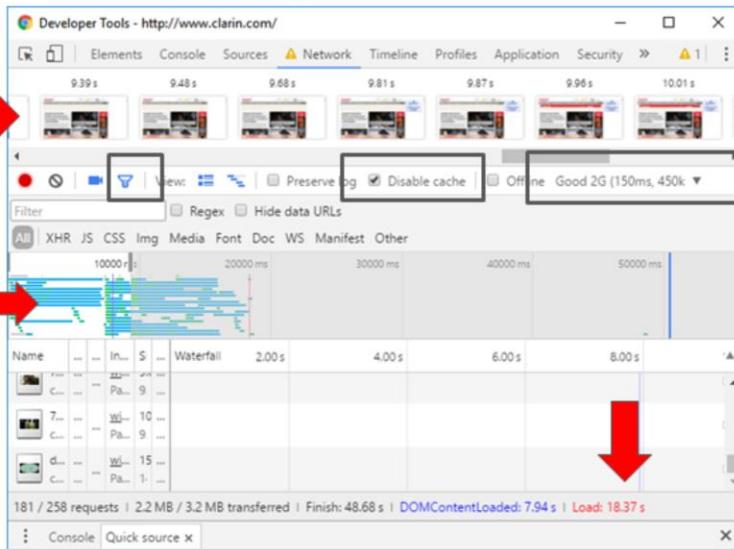
## Demo: Chrome DevTools, encabezados, archivos...



Para ver las características de chrome para desarrollo ver el siguiente link:

<https://developers.google.com/web/tools/chrome-devtools/>

## Demo: Emular experiencia desde un smartphone



Para mas información ir a <https://developers.google.com/web/tools/chrome-devtools/network-performance/>

# Editores de texto para HTML

## Editores livianos

1. Visual Studio Code Editor
2. Notepad ++
3. Sublime Text
4. Atom

## Entornos de desarrollo y programación

1. Visual Studio 2019, 2017 / 2015 / 2013 update 4
2. Eclipse
3. Netbeans

# Nuevas características de HTML 5

**Semántica:** Permite describir con mayor precisión cuál es su contenido y definir elementos propios de html como así también atributos

**Conectividad:** Permite comunicarse con el servidor de formas innovadoras.

**Sin conexión y almacenamiento:** Permite a las páginas web almacenar datos localmente en el lado del cliente y operar sin conexión de manera más eficiente.

**Multimedia:** soporte contenido multimedia de audio y video nativamente.

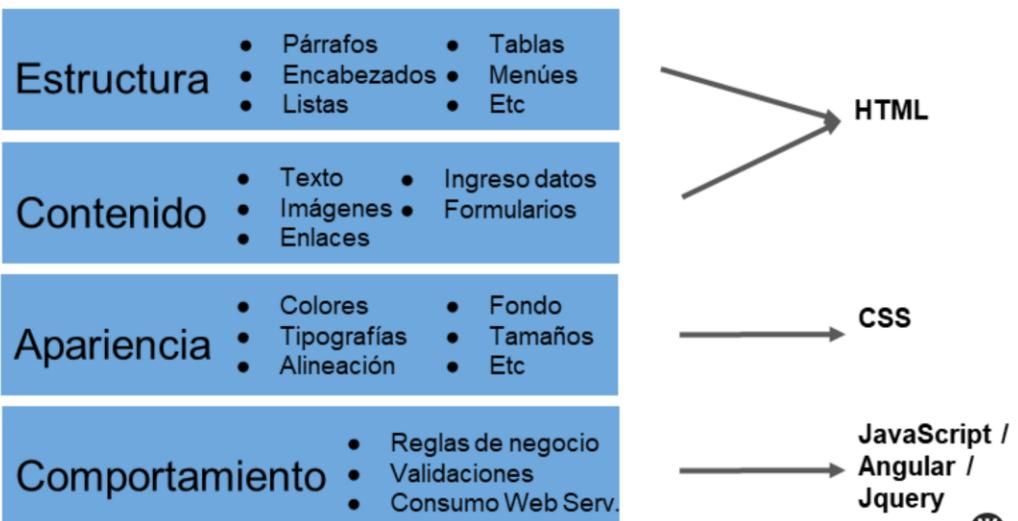
**Gráficos y efectos 2D/3D:** Incorpora nuevas características que se ocupan de los gráficos en la web como lo son canvas 2D, WebGL, SVG, etc.

**Rendimiento e Integración:** Proporciona una mayor optimización de la velocidad y un mejor uso del hardware.

**Acceso al dispositivo:** Proporciona APIs para el uso de varios componentes internos de entrada y salida de nuestro dispositivo, tales como geolocalización

**CSS3:** Gran variedad de opciones para hacer diseños más sofisticados.

# Elementos de un documento HTML



# Estructura de un documento HTML5

- La declaración <!DOCTYPE html> define el tipo de documento.
- El texto entre <html> y </html> describe el documento HTML.
- El texto entre <head></head> proporciona información sobre el documento.
- El texto entre <body> y </body> describe el contenido de la página visible en el navegador

```
<!DOCTYPE html>
<html>
  <head>
    <title>Aprendiendo HTML</title>
  </head>
  <body>
    <h1>Bienvenido a PAVII</h1>
    <p>Este es un documento html 5</p>
  </body>
</html>
```

# Para qué sirve el tag <head> y <body>

- Head:
  - Es parte fundamental de la estructura de un documento html
  - Se usa siempre
  - No se puede poner dentro de la etiqueta <body>
  - Se incluye información acerca del documento
  - Sección meramente técnica e informativa
  - Dentro de esta sección se pueden establecer
    - Título de la página
    - Configuraciones de Meta Tags
    - Estilos CSS
    - Scripts (javascript y/o enlaces a archivos)
- Body
  - Representa el contenido de un documento HTML
  - Sólo puede haber uno por documento

HEAD traducido del inglés al español significa "cabeza", por eso a esta parte del documento la llamamos cabecera. En el documento HTML comienza con la etiqueta <head> y se indica su final con la etiqueta </head>, se escribe así:

```
<html>
  <head>
    Todo lo que esté aquí pertenece al HEAD
  </head>
  <body>
    </body>
</html>
```

El HEAD es la parte donde se incluye la información acerca del documento, podríamos atrevernos a decir que el HEAD es una sección de un documento HTML meramente "técnica e informativa", pues la mayoría de esta información no la muestra el navegador al usuario e inclusive pudiéramos dejarla vacía y esto no afectaría al funcionamiento o la forma en que se visualiza la página, y si bien el HEAD de un documento HTML pudiera ir vacío siempre es mejor darles la suficiente importancia a las etiquetas que el HEAD contiene, mucho más aún si nuestro objetivo es publicar nuestro trabajo en la web, pues muchas de las etiquetas del HEAD son importantes para los buscadores y para un buen posicionamiento en los resultados de búsqueda.

Bueno vamos al grano y conozcamos las etiquetas del HEAD.

**TITLE:**

```
<head>
```

```
<title>Título de la página</title>
</head>
```

Un elemento del HEAD visible desde el navegador, muy importante para los buscadores pues es el texto que se visualiza en los resultados de búsqueda. Me parece importante señalar que este elemento nada tiene que ver con el nombre del archivo pues son dos cosas totalmente distintas e independientes una de la otra, por ejemplo podríamos tener un archivo llamado index.html y en el código un <title>Página principal - Bienvenido</title> y esto estaría bien.

#### **META:**

```
<head>
  <meta name="description" content="Artículos sobre HTML" />
  <meta name="keywords" content="HTML,manual de HTML" />
  <meta name="author" content="Israel Romero" />
  <meta http-equiv="refresh" content="30" />
</head>
```

Una etiqueta con atributos. La función de esta etiqueta va a depender totalmente de los atributos y valores que contenga, en nuestro ejemplo las primeras 3 tienen los atributos "name" y "content" pero tienen valores diferentes, la primera (con valor "description") indica la descripción de la página, la segunda indica las palabras clave con la que los buscadores deberían de relacionarla, en el ejemplo se pretende que los buscadores muestren la página en sus resultados cuando alguien busque "HTML" o "manual de HTML", la tercera incluye el nombre del autor de la página. Ahora la cuarta etiqueta tiene atributos diferentes, en este caso esta etiqueta le dice al navegador que la página se debe actualizar cada 30 segundos. Hay algunos otros argumentos posibles para la etiqueta <meta>, pero el tema principal aquí es el HEAD por lo que tocará hablar del META mas a fondo en otro post.

#### **STYLE:**

```
<head>
  <style type="text/css">
    p {color:blue;}
  </style>
</head>
```

Con esta etiqueta se puede incluir código CSS dentro del documento HTML, en este ejemplo se le da un color azul a los párrafos, es decir a los fragmentos de texto contenidos por las etiquetas <p> </p>.

#### **SCRIPT:**

```
<head>
  <script type="text/javascript">
    document.write("Hola")
  </script>
</head>
```

Otra mas para agregar código de lenguajes ajenos a HTML, en este ejemplo agregamos código de javascript que lo único que hace es mostrarnos un texto que dice "Hola".

## Ejemplo tag <head> y <body>

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>PAVII - UTN FRC</title>
    <link href="css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <h1>Ejemplo de tag HEAD para utilizar Bootstrap como hoja de estilo</h1>
    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <!-- Include all compiled plugins (below), or include individual files as needed -->
    <script src="js/bootstrap.min.js"></script>
</body>
</html>
```

**<meta charset="utf-8">** : Especifica el juego de caracteres (codificación) de la página html

Valores comunes:

UTF-8 - Codificación de caracteres para Unicode

ISO-8859-1 - Codificación de caracteres para el alfabeto latino

En teoría, cualquier codificación de caracteres se puede utilizar, pero no todos los navegadores entienden todas las codificaciones.

para ver mas información ir aqui <http://www.iana.org/assignments/character-sets/character-sets.xhtml>

**<meta http-equiv="X-UA-Compatible" content="IE=edge">**

El atributo content especifica el modo de la página; por ejemplo, para imitar el comportamiento de Windows Internet Explorer 7, especifique IE=EmulateIE7. Del mismo modo, especifique IE=5, IE=7 o IE=8 para seleccionar uno de estos modos de compatibilidad. También puede especificar IE=edge para indicar a Windows Internet Explorer 8 o 10 que use el máximo modo disponible.

El encabezado X-UA-compatible no distingue entre mayúsculas y minúsculas; no obstante, debe aparecer en el encabezado de la página web (la sección HEAD) antes que todos los demás elementos, excepto el elemento TITLE y otros elementos META.

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

La etiqueta meta para el viewport fue introducida por Apple en Safari para móviles, originalmente se utilizó para ayudar a los desarrolladores a mejorar la presentación de sus aplicaciones web en un iPhone, iPod Touch o iPad.

La etiqueta viewport nos permite a los que construimos sitios web o web apps, definir el ancho, alto y escala del área usada por el navegador para mostrar contenido.

Al fijar el ancho o alto del viewport, los desarrolladores podemos usar un número fijo de pixeles (ej: 320, 480, etc) o usar dos constantes, device-width y device height respectivamente. Se considera una buena práctica configurar el viewport con algunas de estas dos constantes, en vez de un ancho o alto fijo.

```
<link href="css/bootstrap.min.css" rel="stylesheet">
```

Esta etiqueta permite enlazar un archivo JavaScript u Hoja de estilo (css) externos con el documento actual. El atributo rel="stylesheet" especifica que el archivo **css/bootstrap.min.css** es de hoja de estilo, y está ubicado en el directorio css del árbol de directorios del sitio web

```
<script
```

```
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
```

Script javascript que está en otro sitio web. En html5 ya no se usa el atributo type="text/javascript". En este caso se posee una referencia a la librería de JQuery que necesita bootstrap.

Observar que el tag **<script>** puede estar en el **<head>** o a finalizar el **<body>** o en ambos lugares.

La colocación de secuencias de comandos en la parte inferior del elemento **<body>** mejora la velocidad de visualización, porque la compilación de secuencias de comandos disminuye la visualización.

## Ejemplo

Utilizar el bloc de notas y desarrollar un documento HTML “Hola Mundo!”

# Content delivery network (CDN)



Servidores con copias de datos en varios puntos de la red muy utilizado para librerías javascript

- Reduce la carga de los servidores.
- Red de tráfico distribuida.
- Reduce la latencia.
- Incrementa el ancho de banda.
- Aumenta el web caching.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
```

**fuente:** wikipedia

## Introducción

Las CDN emergen como la solución al actual problema que presenta una web centralizada: lograr bajo tiempo de respuesta y mínima pérdida de información moviendo el contenido de la información más cerca de los usuarios. El objetivo es lograr un equilibrio entre los costos en que incurren los proveedores de contenido web y la calidad de servicio para los usuarios finales.<sup>2</sup>

## Las ventajas de la implementación de este modelo son las siguientes:

Reduce la carga de los servidores.  
Red de tráfico distribuida.  
Reduce la latencia.  
Incrementa el ancho de banda.  
Aumenta el web caching.

## Arquitectura de una CDN

Componente de entrega de contenidos

Se cuenta con un servidor de origen y un conjunto de servidores sustitutos para replicar el contenido.

Componente de enrutamiento de solicitudes

Usuarios solicitan directamente a los servidores sustitutos.

Interactúa con el componente de distribución para mantener y actualizar el contenido

#### Componente de distribución de contenido

Mueve el contenido desde el origen a los servidores sustitutos y asegura consistencia

#### Componente de contabilidad

Mantiene registros de los accesos de los clientes y los registros de uso de los servidores.

Ayuda a la presentación de informes de tráfico y facturación basada en el uso

### **Servicios y contenidos compatibles con CDN**

Contenido estático: Páginas estáticas HTML, imágenes, documentos, parches de software.

Distribución de audio y video por internet: Audio y video en tiempo real. Videos generados por el usuario

Servicio de contenido: Directorio, e – commerce, servicio de transferencia de archivos.

Fuentes de contenido: Grandes empresas, proveedores de servicios Web, compañías de medios de comunicación, y emisoras de noticias

Clientes: Medios de comunicación y empresas de publicidad por internet, centros de datos, proveedores de Internet, minoristas de música en línea, operadores móviles y fabricantes de electrónica de consumo.

Interacción del usuario: Celular, smartphone/PDA, notebooks y computadores de escritorio.

### **Objetivos de negocio**

#### Escalabilidad

Habilidad para expandirse con el objetivo de manejar nuevos y grandes cantidades de datos. Usuarios y transacciones.

Requiere capacidad para la entrega de contenido dinámico de aprovisionamiento y de alta calidad, con bajo costo operacional

Tendencia futura: los proveedores de contenidos, así como usuarios finales pagarán para obtener contenido de alta calidad

#### Seguridad

Protección del contenido contra modificaciones y accesos no autorizados.

Requiere red física, software, datos y procedimientos de seguridad.

Tendencia futura: reducir la interrupción del negocio mediante la lucha contra los ataques de negación de servicio y otras actividades maliciosas.

#### Fiabilidad, Capacidad de respuesta y rendimiento

Disponibilidad de servicios, manejo de posibles interrupciones y experiencia del usuario final.

Requiere una red tolerante a fallas con balanceo de carga adecuada.  
Tendencia futura: ubicación del contenido distribuido, la coherencia de caché y los mecanismos de enrutamiento

### **Beneficios del CDN**

Debido a la arquitectura del CDN, se pueden detallar los siguientes beneficios:

Mayor capacidad de conexión.

Disminución del tiempo de respuesta de entrega de información al usuario.

Disminución de los costos asociados a la entrega de contenidos.

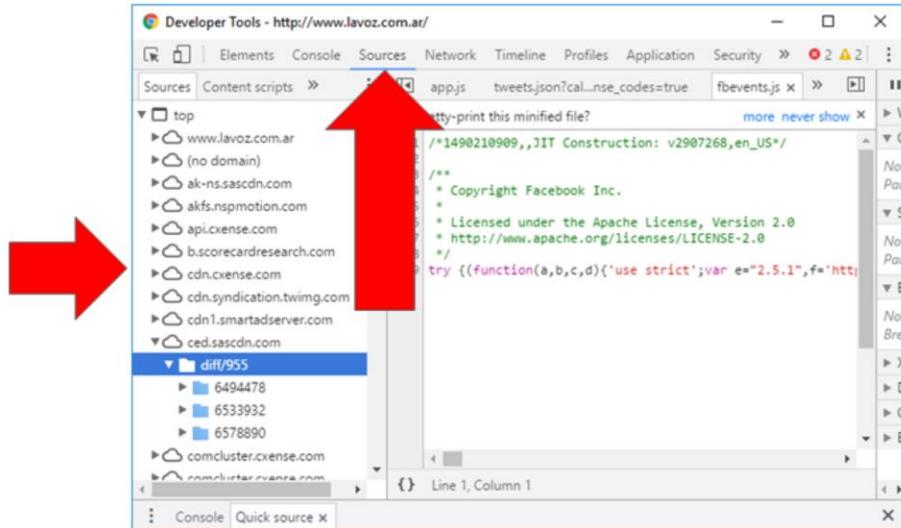
Reducción de la pérdida y demora de paquetes ya que trabajan con nodos cercanos al usuario.

Disminución de carga de la red.

Se tiene 100% de disponibilidad de información, incluso ante la caída de uno de los servidores.

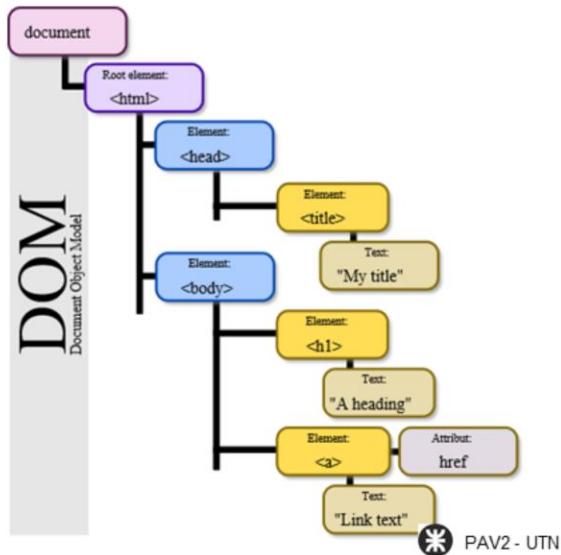
Permite obtener estadísticas de comportamiento de usuarios basado en el registro de páginas visitadas, ubicación geográfica, entre otras.

## Demo: mostrar CDN del sitio www.lavoz.com.ar



## Qué es el DOM de un documento HTML

- Cuando se carga una página web, el navegador crea un DOM (Document Object Model) de la página.
- El HTML DOM está construido como un árbol de objetos.
- El DOM permite ser manipulado desde JavaScript.



PAV2 - UTN FRC

## ¿Qué es el DOM?

El DOM es una W3C (World Wide Web Consortium) estándar.

El DOM define un estándar para acceso a los mismos:

*"El modelo de objetos de documento del W3C (DOM) es una interfaz de plataforma y lenguaje neutro que permite a los programas y scripts acceder y actualizar el contenido, la estructura y el estilo de un documento de forma dinámica."*

El estándar W3C DOM se separa en 3 partes diferentes:

- Núcleo del DOM - modelo estándar para todos los tipos de documentos
- DOM XML - modelo estándar para documentos XML
- HTML DOM - modelo estándar para documentos HTML

## ¿Qué es el DOM de HTML?

El DOM HTML es un estándar **de objetos** de modelo y de **programación de interfaces** para HTML. Se define:

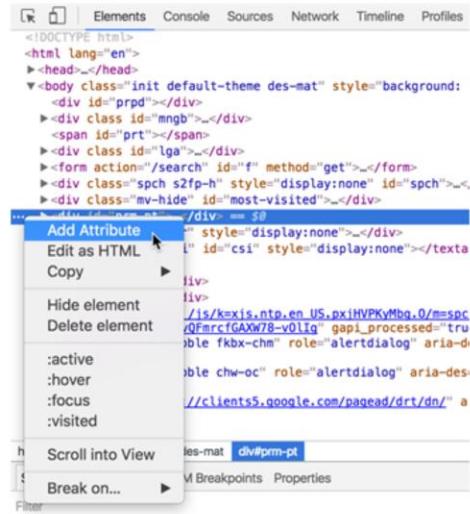
- Los elementos HTML como **objetos**
- Las **propiedades** de todos los elementos HTML
- Los **métodos** para acceder a todos los elementos HTML
- Los **eventos** de todos los elementos HTML

En otras palabras: **El DOM HTML es un estándar de cómo obtener, cambiar, añadir o eliminar elementos HTML.**

Con el modelo de objetos, JavaScript posee toda la funcionalidad necesaria para crear HTML dinámico:

- Puede alterar todos los elementos HTML en la página
- Puede modificar todos los atributos de HTML en la página
- Permite cambiar todos los estilos CSS en la página
- Es posible eliminar elementos y atributos HTML existentes
- Contiene la funcionalidad para agregar nuevos elementos y atributos HTML
- Permite implementar los eventos de HTML existentes en la página
- Se puede agregar nuevos manejadores de eventos en la página HTML

## Demo: Editar DOM con Chrome DevTools



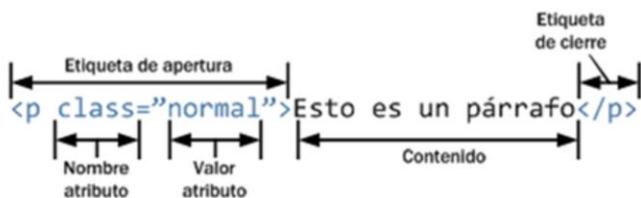
Abrir la página web de la utn y presionar F12 para ver las DevTools de Chrome.  
Mostrar el DOM como HTML

## Ejercitación

Resolver el ejercicio:

**Ejercicio HTML1 - análisis de un documento.docx**

# Definición de elemento de un documento HTML



- Los elementos son los componentes fundamentales del HTML
- Cuentan con 2 propiedades básicas:
  - Atributos
  - Contenido
- En general se conforman con una Etiqueta de Apertura y otra Cierre.
- Los atributos se colocan dentro la etiqueta de apertura, y el contenido se coloca entre la etiqueta de apertura y la de cierre.

Todos los elementos del estándar HTML5 están listados aquí, descritos por su etiqueta de apertura y agrupados por su función.

Esta lista solamente los elementos válidos de HTML5. Solamente aquellos elementos listados aquí son los que deberían ser usados en nuevos sitios Web.

El simbolo This element was added as part of HTML5 indica que el elemento fue añadido en HTML5. Nótese que otros elementos listados aquí pueden haber sido modificados o extendido en su significado por la especificación HTML5.

## Elemento raíz

Elemento	Descripción
<!doctype html>	Define que el documento está bajo el estándar de HTML 5
<html>	Representa la raíz de un documento HTML o XHTML. Todos los demás elementos deben ser descendientes de este elemento.

## Metadatos del documento

Elemento	Descripción
<head>	Representa una colección de metadatos acerca del documento, incluyendo enlaces a, o definiciones de, scripts y hojas de estilo.
<title>	Define el título del documento, el cual se muestra en la barra de título del navegador o en las pestañas de página. Solamente puede contener texto y cualquier otra etiqueta contenida no será interpretada.

<base>	Define la URL base para las URLs relativas en la página.
<link>	Usada para enlazar JavaScript y CSS externos con el documento HTML actual.
<meta>	Define los metadatos que no pueden ser definidos usando otro elemento HTML.
<style>	Etiqueta de estilo usada para escribir CSS en línea.

## Scripting

Elemento	Descripción
<script>	Define ya sea un script interno o un enlace hacia un script externo. El lenguaje de programación es JavaScript
<noscript>	Define una contenido alternativo a mostrar cuando el navegador no soporta scripting.

## Secciones

Elemento	Descripción
<body>	Representa el contenido principal de un documento HTML. Solo hay un elemento <body> en un documento.
<section>	(HTML5)Define una sección en un documento.
<nav>	(HTML5)Define una sección que solamente contiene enlaces de navegación
<article>	(HTML5)Define contenido autónomo que podría existir independientemente del resto del contenido.
<aside>	(HTML5)Define algunos contenidos vagamente relacionados con el resto del contenido de la página. Si es removido, el contenido restante seguirá teniendo sentido
<h1>,<h2>, <h3>,<h4> , <h5>,<h6>	Los elemento de cabecera implementan seis niveles de cabeceras de documentos; <h1> es la de mayor y <h6> es la de menor importancia.

Un elemento de cabecera describe brevemente el tema de la sección que introduce.

<header> (HTML5)Define la cabecera de una página o sección. Usualmente contiene un logotipo, el título del sitio Web y una tabla de navegación de contenidos.

<footer> (HTML5)Define el pie de una página o sección. Usualmente contiene un mensaje de derechos de autoría, algunos enlaces a información legal o direcciones para dar información de retroalimentación.

<address> Define una sección que contiene información de contacto.

<main> (HTML5)Define el contenido principal o importante en el documento. Solamente existe un elemento <main> en el documento.

## Agrupación de Contenido

Elemento	Descripción
<p>	Define una parte que debe mostrarse como un párrafo.

<hr>	Representa un quiebre temático entre párrafos de una sección o artículo o cualquier contenido.
<pre>	Indica que su contenido está preformatado y que este formato debe ser preservado.
<blockquote>	Representa un contenido citado desde otra fuente.
<ol>	Define una lista ordenada de artículos.
<ul>	Define una lista de artículos sin orden.
<li>	Define un artículo de una lista enumerada.
<dl>	Define una lista de definiciones, es decir, una lista de términos y sus definiciones asociadas.
<dt>	Representa un término definido por el siguiente <dd>
.	
<dd>	Representa la definición de los términos listados antes que él.
<figure>	(HTML5) Representa una figura ilustrada como parte del documento.
<figcaption>	(HTML5) Representa la leyenda de una figura.
<div>	Representa un contenedor genérico sin ningún significado especial.

### Semántica a nivel de Texto

Elemento	Descripción
<a>	Representa un hiperenlace , enlazando a otro recurso.
<em>	Representa un texto enfatizado , como un acento de intensidad.
<strong>	Representa un texto especialmente importante .
<small>	Representa un comentario aparte , es decir, textos como un descargo de responsabilidad o una nota de derechos de autoría, que no son esenciales para la comprensión del documento.
<s>	Representa contenido que ya no es exacto o relevante .
<cite>	Representa el título de una obra .
<q>	Representa una cita textual inline.
<dfn>	Representa un término cuya definición está contenida en su contenido ancestro más próximo.
<abbr>	Representa una abreviación o un acrónimo ; la expansión de la abreviatura puede ser representada por el atributo title.
<data>	(HTML5) Asocia un equivalente legible por máquina a sus contenidos. (Este elemento está sólamente en la versión de la WHATWG del estandar HTML, y no en la versión de la W3C de HTML5).
<time>	(HTML5) Representa un valor de fecha y hora; el equivalente legible por máquina puede ser representado en el atributo datetime.
<code>	Representa un código de ordenador .
<var>	Representa a una variable, es decir, una expresión matemática o contexto de programación, un identificador que represente a una constante, un símbolo que identifica una cantidad física, un parámetro de una función

o un marcador de posición en prosa.

`<samp>` Representa la salida de un programa o un ordenador.

`<kbd>` Representa la entrada de usuario, por lo general desde un teclado, pero no necesariamente, este puede representar otras formas de entrada de usuario, como comandos de voz transcritos.

`<sub>`,`<sup>` Representan un subíndice y un superíndice, respectivamente.

`<i>` Representa un texto en una voz o estado de ánimo alterno, o por lo menos de diferente calidad, como una designación taxonómica, un término técnico, una frase idiomática, un pensamiento o el nombre de un barco.

`<b>` Representa un texto hacia el cual se llama la atención para propósitos utilitarios. No confiere ninguna importancia adicional y no implica una voz alterna.

`<u>` Representa una anotación no textual sin-articular, como etiquetar un texto como mal escrito o etiquetar un nombre propio en texto en Chino.

`<mark>` (HTML5)Representa texto resaltado con propósitos de referencia, es decir por su relevancia en otro contexto.

`<ruby>` (HTML5)Representa contenidos a ser marcados con anotaciones ruby, recorridos cortos de texto presentados junto al texto. Estos son utilizados con regularidad en conjunto a lenguajes de Asia del Este, donde las anotaciones actúan como una guía para la pronunciación, como el furigana Japonés.

`<rt>` (HTML5)Representa el texto de una anotación ruby .

`<rp>` (HTML5)Representa los paréntesis alrededor de una anotación ruby, usada para mostrar la anotación de manera alterna por los navegadores que no soporten despliegue estandar para las anotaciones.

`<bdi>` (HTML5)Representa un texto que debe ser aislado de sus alrededores para el formateado bidireccional del texto. Permite incrustar un fragmento de texto con una direccionalidad diferente o desconocida.

`<bdo>` Representa la direccionalidad de sus descendientes con el fin de anular de forma explícita al algoritmo bidireccional Unicode.

`<span>` Representa texto sin un significado específico. Este debe ser usado cuando ningún otro elemento semántico le confiere un significado adecuado, en cuyo caso, provendrá de atributos globales como class, lang, o dir.

`<br>` Representa un salto de línea.

`<wbr>` (HTML5)Representa una oportunidad de salto de línea, es decir, un punto sugerido de envoltura donde el texto de múltiples líneas puede ser dividido para mejorar su legibilidad.

## Ediciones

Elemento Descripción

`<ins>` Define una adición en el documento.

`<del>` Define una remoción del documento.

## Contenido incrustado

Elemento Descripción

`<img>` Representa una imagen.

<iframe> Representa un contexto anidado de navegación, es decir, un documento HTML embebido.

<embed> (HTML5)Representa un punto de integración para una aplicación o contenido interactivo externo que por lo general no es HTML.

<object> Representa un recurso externo, que será tratado como una imagen, un sub-documento HTML o un recurso externo a ser procesado por un plugin.

<param> Define parámetros para el uso por los plugins invocados por los elementos <object> .

<video> (HTML5)Representa un video , y sus archivos de audio y capciones asociadas, con la interfaz necesaria para reproducirlos.

<audio> (HTML5)Representa un sonido o stream de audio.

<source> (HTML5)Permite a autores especificar recursos multimedia alternativos para los elementos multimedia como <video> o <audio> .

<track> (HTML5)Permite a autores especificar una pista de texto temporizado para elementos multimedia como <video> o <audio> .

<canvas> (HTML5)Representa un área de mapa de bits en el que se pueden utilizar scripts para renderizar gráficos como gráficas, gráficas de juegos o cualquier imagen visual al vuelo.

<map> En conjunto con <area> , define un mapa de imagen.

<area> En conjunto con <map> , define un mapa de imagen.

<svg> (HTML5)Define una imagen vectorial embebida.

<math> (HTML5)Define una fórmula matemática.

## Datos tabulares

Elemento	Descripción
<table>	Representa datos con más de una dimensión.
<caption>	Representa el título de una tabla.
<colgroup>	Representa un conjunto de una o más columnas de una tabla.
<col>	Representa una columna de una tabla.
<tbody>	Representa el bloque de filas que describen los datos contretos de una tabla.
<thead>	Representa el bloque de filas que describen las etiquetas de columna de una tabla.
<tfoot>	Representa los bloques de filas que describen los resúmenes de columna de una tabla.
<tr>	Representa una fila de celdas en una tabla.
<td>	Representa una celda de datos en una tabla.
<th>	Representa una celda encabezado en una tabla.

## Formularios

Elemento	Descripción
<form>	Representa un formulario, consistendo de controles que puede ser enviado a un servidor para procesamiento.

<fieldset>	Representa un conjunto de controles.
<legend>	Representa el título de un <fieldset> .
<label>	Representa el título de un control de formulario.
<input>	Representa un campo de datos escrito que permite al usuario editar los datos.
<button>	Representa un botón .
<select>	Representa un control que permite la selección entre un conjunto de opciones.
<datalist>	(HTML5)Representa un conjunto de opciones predefinidas para otros controles.
<optgroup>	Representa un conjunto de opciones, agrupadas lógicamente.
<option>	Representa una opción en un elemento <select> , o una sugerencia de un elemento <datalist> .
<textarea>	Representa un control de edición de texto multi-línea.
<keygen>	(HTML5)Representa un control de par generador de llaves.
<output>	(HTML5)Representa el resultado de un cálculo.
<progress>	(HTML5)Representa el progreso de finalización de una tarea.
<meter>	(HTML5)Representa la medida escalar (o el valor fraccionario) dentro de un rango conocido.

### **Elementos interactivos**

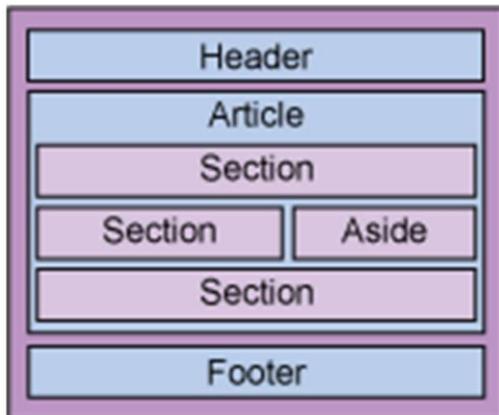
Elemento	Descripción
<details>	(HTML5)Representa un widget desde el que un usuario puede obtener información o controles adicionales.
<summary>	(HTML5)Representa un resumen, título o leyenda para un elemento <details> dado.
<command>	(HTML5)Representa un comando que un usuario puede invocar.
<menu>	(HTML5)Representa una lista de comandos .

## Recomendaciones para escritura de html

- Uso minúsculas en nombres de elementos
- HTML5 permite mezclar letras mayúsculas y minúsculas en nombres de elementos. Se recomienda utilizar nombres de elementos minúsculas porque: mezclar mayúsculas y minúsculas los nombres es una mala práctica en HTML
- El texto en minúsculas es más limpio y fácil de leer
- Encerrar los valores de los atributos siempre con comilla doble
- Evitar espacios entre el signo igual (=) en la especificación del atributo="valor" en los elementos de HTML.

# Elementos estructurales de HTML

- header
- section
- article
- aside
- footer
- nav



<https://www.ibm.com/developerworks/ssa/web/library/wa-html5structuraltags/>

## Nuevos elementos estructurales

La razón para crear nuevas etiquetas estructurales es dividir las páginas Web en partes lógicas que describan el tipo de contenido que incluyen. Conceptualmente, piense en la página Web como un documento. Los documentos tienen encabezados, pies de página, capítulos y otras convenciones diferentes que dividen el documento en partes lógicas.

Esta sección revisa los métodos actuales de dividir un documento HTML usando código genérico de muestra. Durante el resto de este artículo, usted revisará el código original usando las nuevas etiquetas estructurales HTML5 para ver paso a paso cómo el documento es transformado en secciones lógicas.

## Enfoque HTML 4

Si usted ha creado incluso los documentos HTML más simples, entonces estará familiarizado (a) con la etiqueta div . La etiqueta div es el principal mecanismo de la era pre-HTML5 para crear bloques de contenido en un documento HTML. Por ejemplo, el [Listado 2](#) muestra cómo usted puede usar etiquetas div para crear una página simple con un encabezado, un área de contenido y un pie de página.

## Listado 2. Página HTML simple usando etiquetas div

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html>  
<head>  
<title>  
Una página HTML simple usando Divs  
</title>  
</head>  
<body>  
<div id='header'>Header</div>  
<div id='content'>Content</div>  
<div id='footer'>Footer</div>  
</body>  
</html>
```

Esto funciona bien; la etiqueta div es una buena etiqueta de propósito general. Sin embargo, además de mirar el atributo id de cada etiqueta div, es difícil decir qué sección del documento representa cada etiqueta div . Aunque usted puede argumentar que la id es un indicador suficiente si se nombra adecuadamente, los atributos id son arbitrarios. Hay muchas variaciones que se pueden considerar ids igualmente válidas. La etiqueta en sí no ofrece ninguna indicación sobre el tipo de contenido que se pretende que represente.

## **Enfoque HTML5**

HTML5 responde a este problema proporcionando un conjunto de etiquetas que definen con mayor claridad los bloques principales de contenido que componen un documento HTML. Sin importar el contenido final mostrado por la página Web, la mayoría de páginas Web consisten en combinaciones de variantes de secciones y elementos de página comunes.

El código siguiente crea una página simple con un encabezado, un área de contenido y un pie de página. Estos y otras secciones y elementos de página son bastante comunes, de manera que HTML5 incluye etiquetas que dividen los documentos en las secciones comunes y que indican el contenido de cada una. Las nuevas etiquetas son:

[header](#)  
[section](#)  
[article](#)

[aside](#)

[footer](#)

[nav](#)

### **El área header**

Como su nombre lo sugiere, la etiqueta header tiene por objeto marcar una sección de la página HTML como el encabezado. Listado 3 muestra el ejemplo de código del [Listado 2](#) modificado para que use un header .

#### **Listado 3. Añadiendo una etiqueta header .**

```
<!DOCTYPE html>
<html>
<head>
<title>
Una página HTML simple
</title>
</head>
<body>
<header>Header</header>
<section>
<p>
Esta es una sección importante de la página.
</p>
</section>
<div id='footer'>Footer</div>
</body>
</html>
```

El doctype en el Listado 3 también se cambió para indicar que el navegador debería utilizar HTML5 para presentar la página. Desde este punto en adelante, todos los ejemplos suponen que usted está utilizando el doctype correcto.

### **El área section**

La etiqueta section tiene por objeto identificar porciones significativas del contenido de la página. Esta etiqueta es de alguna forma análoga a dividir un libro en capítulos. Añadiendo una etiqueta section al código de ejemplo da como resultado el código en el Listado 4.

#### **Listado 4. Añadiendo una etiqueta section .**

```
<!DOCTYPE html>
```

```
<html>
<head>
<title>
Una página HTML simple
</title>
</head>
<body>
<header>Header</header>
<section>
<article>
<p>
    Esta es una sección importante del contenido de la
    página.
    Tal vez una publicación en blog.
</p>
</article>
<article>
<p>
    Esta es una sección importante del contenido de la
    página.
    Tal vez una publicación en blog.
</p>
</article>
</section>
<div id='footer'>Footer</div>
</body>
</html>
```

## El área article

La etiqueta article identifica las secciones principales del contenido dentro de la página Web. Piense en un blog, donde cada publicación de cada individuo constituye una porción significativa de contenido. Añadiendo etiquetas article al código de ejemplo da como resultado el código en el Listado 5.

### Listado 5. Añadiendo etiquetas article

```
<!DOCTYPE html>
<html>
<head>
<title>
Una página HTML simple
```

```

</title>
</head>
<body>
<header>Header</header>
<section>
    <article>
        <p>
            Esta es una sección importante del contenido de la
            página.
            Tal vez una publicación en blog.
        </p>
    </article>
    <article>
        <p>
            Esta es una sección importante del contenido de la
            página.
            Tal vez una publicación en blog.
        </p>
    </article>
</section>
<div id='footer'>Footer</div>
</body>
</html>

```

## **La etiqueta aside**

La etiqueta aside indica que el contenido dentro de ella está relacionado el contenido principal de la página pero que no es parte de ella. En cierta forma es análogo a usar paréntesis para hacer un comentario en un cuerpo de texto (como este). El contenido entre paréntesis proporciona información adicional sobre el elemento que lo contiene. Añadiendo una etiqueta aside al código de ejemplo da como resultado el código en el Listado 6.

### Listado 6. Añadiendo una etiqueta aside .

```

<!DOCTYPE html>
<html>
<head>
<title>
    Una página HTML simple
</title>
</head>

```

```
<body>
<header>Header</header>
<section>
<article>
<p>
Esta es una sección importante del contenido de la página.
Tal vez una publicación en blog.
</p>
</aside>
</p>
Este es un aparte de la primera publicación en blog.
</p>
</aside>
</article>
<article>
<p>
Esta es una sección importante del contenido de la página.
Tal vez una publicación en blog.
</p>
</article>
</section>
<div id='footer'>Footer</div>
</body>
</html>
```

## La etiqueta footer

La etiqueta footer marca el contenido dentro del elemento que es el pie de página del documento. Añadiendo una etiqueta footer al código de ejemplo da como resultado el código en el Listado 7.

### Listado 7. Añadiendo una etiqueta footer .

```
<!DOCTYPE html>
<html>
<head>
<title>
Una página HTML simple
</title>
</head>
<body>
```

```
<header>Header</header>
<section>
<article>
<p>
Esta es una sección importante del contenido de la página.
Tal vez una publicación en blog.
</p>
</aside>
</p>
Este es un aparte de la primera publicación en blog.
</p>
</aside>
</article>
<article>
<p>
Esta es una sección importante del contenido de la página.
Tal vez una publicación en blog.
</p>
</article>
</section>
<footer>Footer</footer>
</body>
</html>
```

en este punto, todas las etiquetas div originales han sido reemplazadas con etiquetas HTML5 estructurales.

### **La etiqueta nav .**

El contenido dentro de la etiqueta nav tiene por objeto propósitos de navegación. Añadiendo una etiqueta nav al código de ejemplo da como resultado el código en el Listado 8.

### **Listado 8. Añadiendo una etiqueta nav .**

```
<!DOCTYPE html>
<html>
<head>
<title>
Una página HTML simple
</title>
</head>
```

```
<body>
<header>Header
<nav>
  <a href="#">Algún enlace de navegación</a>
  <a href="#">Algún enlace de navegación adicional</a>
  <a href="#">Un tercer enlace de navegación</a>
</nav>
</header>
<section>
<article>
<p>
  Esta es una sección importante del contenido de la página.
  Tal vez una publicación en blog.
</p>
</aside>
</p>
  Este es un aparte de la primera publicación en blog.
</p>
</aside>
</article>
<article>
<p>
  Esta es una sección importante del contenido de la página.
  Tal vez una publicación en blog.
</p>
</article>
</section>
<footer>Footer</footer>
</body>
</html>
```

## Conclusión

Las nuevas etiquetas HTML5 describen los tipos de contenido que contienen, y ayudan a dividir el documento en secciones lógicas. Todavía depende de usted decidir cuándo y dónde utilizar las nuevas etiquetas dentro de un documento, de forma similar a como un autor escribe un libro. Mientras dos autores escribiendo el mismo libro pueden optar por diferentes formas de dividir el libro en capítulos, la acción de usar capítulos ofrece un método consistente de dividir el libro en secciones. De manera similar, aunque los dos autores de una página Web dada

pueden optar por estructuras diferentes, las nuevas etiquetas estructurales HTML5 proporcionan nuevas convenciones que los desarrolladores de páginas Web pueden usar y que las viejas etiquetas div no ofrecían.

# Elementos HTML más utilizados

Elemento	Descripción
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>	Los elementos de cabecera implementan seis niveles de cabeceras de documentos; <h1> es la de mayor y <h6> es la de menor importancia. Un elemento de cabecera describe brevemente el tema de la sección que introduce.
<p>	Define una parte que debe mostrarse como un párrafo.
<strong>	Negrita
 	Salto de línea
<label>	Representa el título de un control de formulario.
<div>	contenedor

# Caracteres especiales de texto en HTML

En HTML los caracteres propios de los idiomas diferentes al inglés pueden ser problemáticos.

La correcta visualización depende de la codificación del documento UTF-8, del editor de HTML, y del servidor.

Para evitar problemas de visualización con los caracteres reservados de HTML se debe utilizar el carácter de **entidad** HTML.

```
<p>Esto es una frase: "Que los a&ntilde;os no te hagan  
m&aacute;s viejo, sino m&aacute;s sabio"</p>
```

Entidad HTML	Caráter
&ampnbsp	espacio
&ntilde;	ñ
&Ntilde;	Ñ
&aacute;	á
&eacute;	é
&iacute;	í
&oacute;	ó
&uacute;	ú
&quot;	"
&lt;	<
&gt;	>



Las entidades son unas estructuras que, mediante el uso de una codificación, nos permiten representar un símbolo.

La estructura de la entidad HTML es un ampersand(&) seguido del código o nombre de la entidad y terminado en un punto y coma.

# HTML Listas

Permiten estructurar mejor el texto y para ordenar la información. Hay dos tipos de lista

- Listas no ordenadas o listas de viñetas

- C#
- Entity Framework
- ASP.NET WebAPI
- AngularJS

```
<h1>Contenido sin ordenar PAVII</h1>
<ul>
<li>C#</li>
<li>Entity Framework</li>
<li>ASP.NET WebAPI</li>
<li>AngularJS</li>
</ul>
```

- Listas ordenadas o numeradas

1. C#
2. Entity Framework
3. ASP.NET WebAPI
4. AngularJS

```
<h1>Contenido ordenado PAVII</h1>
<ol>
<li>C#</li>
<li>Entity Framework</li>
<li>ASP.NET WebAPI</li>
<li>AngularJS</li>
</ol>
```

# HTML - Imágenes

Se especifican los siguientes atributos:

- El archivo a visualizar (src),
- texto alternativo, o para personas con discapacidad visual (alt),
- width, y height

```

```



 Universidad Tecnológica Nacional  
FACULTAD REGIONAL COORDOBA

# HTML - Enlaces

Define un hiper vínculo a una ubicación en la misma página o cualquier otra página en la Web.

```
<h1>Enlace con texto</h1>
<a href="http://www.frc.utn.edu.ar">Ir al sitio de la UTN - FRC</a>

<h1>Enlace con imagen</h1>
<a href="http://www.institucional.frc.utn.edu.ar/sistemas/">
    
</a>

<h1>Enlace a un teléfono, muy &acute;til en smartphones</h1>
<a href="tel:+54351598-6000">Para llamar a la facultad 598-6000</a>
```

## Enlace con texto

[Ir al sitio de la UTN - FRC](http://www.frc.utn.edu.ar)

## Enlace con imagen

 Universidad Tecnológica Nacional  
FACULTAD REGIONAL CÓRDOBA

## Enlace a un teléfono, muy

[Para llamar a la facultad 598-6000](tel:+54351598-6000)

# Ejercitación

Resolver el ejercicio

**Ejercicio HTML2 - formateo básico.docx**

# HTML -Tablas

```
<table>
  <tr>
    <th>Documento</th>
    <th>Apellido y Nombre</th>
  </tr>
  <tr>
    <td>34.434.532</td>
    <td>Perez Juan</td>
  </tr>
  <tr>
    <td>32.524.922</td>
    <td>Lopez Mar&iacute;a</td>
  </tr>
  <tr>
    <td colspan="2">Total de
    clientes: 2</td>
  </tr>
</table>
```



Documento Apellido y Nombre	
34.434.532	Perez Juan
32.524.922	Lopez María
Total de clientes: 2	

- <table> indica que se debe renderizar una tabla.
- Cada fila es un tag <tr> (**table row**)
- La cabecera se especifica como <th>
- Cada celda es un tag <td> (**table data**)
- El atributo **colspan** indica que una celda va a ocupar “n” columnas. Por defecto colspan=”1”

# Ejercitación

Resolver el ejercicio:

**Ejercicio HTML3 - tablas.docx**

# HTML - Elementos de línea y de bloque

- Elementos **inline**: Se posiciona horizontalmente en línea con los otros elementos.
  - La altura y el ancho se define en base al contenido que posea.
  - Solo puede contener elementos de tipo inline.
  - No se puede aplicar una anchura y un altura fija por medio de CSS.
  - Solo ocupan el espacio necesario para mostrar sus contenidos
  - Ej.: `<a>, <br>, <span>, <img>, <i>, <b>, <big>, <small>, <u>, <s>, <em>, <strong>, <input>, <select>, <textarea>, <label>, <button>`
- Elementos **block**: Forma un bloque y se posiciona de forma vertical con un nuevo salto de línea.
  - Las anchura es la máxima que puede tomar dentro de su elemento contenedor (padre)
  - La altura cambia en base al contenido que posea.
  - Puede contener otros elementos de tipo inline y block
  - Por medio de CSS se le puede aplicar una anchura y un altura fija.
  - Ej: `<p>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <div> <ul>, <ol>, <li>, <table>, <form>`

`<big>` No es soportado en HTML5

Se puede consultar los InLine en [https://www.w3schools.com/html/html\\_blocks.asp](https://www.w3schools.com/html/html_blocks.asp)

Se puede consultar los tags en <https://www.w3schools.com/tags/>

## HTML - Etiqueta <div>

- Define un bloque de contenido o sección de la página.
- Puede contener otros elementos tipo block o inline de html
- Muy utilizado para el maquetado de la página y para aplicar hojas de estilo
- El atributo `style="color:red"` especifica el color

```
<div style="color:red">
    <h3>Esto es una cabecera</h3>
    <p>El párrafo está en rojo.</p>
</div>
<div style="color:blue">
    <h3>Esto es una cabecera</h3>
    <p>El párrafo está en azul.</p>
</div>
```



Esto es una cabecera  
El párrafo está en rojo.  
Esto es una cabecera  
El párrafo está en azul.

## Demo: Análisis de una página html

- Para analizar que un documento conforme el estándar HTML ir a  
<https://validator.w3.org/>
- Para recomendaciones para mejorar la velocidad de carga de la página ir a  
<https://developers.google.com/speed/pagespeed/>

## Minificar recursos (HTML, CSS y JavaScript)

- Técnica que permite la eliminación de bytes innecesarios, como los espacios adicionales, saltos de línea y sangrías en HTML, CSS y JavaScript
- Permite acelerar la descarga, el análisis y el tiempo de ejecución
- En CSS y JavaScript, es posible reducir aún más el tamaño del archivo al cambiar el nombre de las variables.
- En JavaScript y CSS los archivos minificados aparecen con la extensión .min.js o .min.css
- Existen varios minificadores online y analizadores como:
  - PageSpeed Insights de google
  - cssmin.js
  - www.minifier.org
  - Closure Compiler

## Minificar recursos (HTML, CSS y Javascript)

### Información general

La minificación de recursos se refiere a la eliminación de bytes innecesarios, como los espacios adicionales, saltos de línea y sangrías. Al minimizar los códigos HTML, CSS y JavaScript, es posible acelerar la descarga, el análisis y el tiempo de ejecución. Además, en CSS y en JavaScript, es posible reducir aun más el tamaño del archivo al cambiar el nombre de las variables, siempre y cuando el código HTML esté actualizado correctamente para garantizar que los selectores sigan funcionando.

### Recomendaciones

Debes minificar el código HTML, CSS y JavaScript.

Para minificar HTML, puedes utilizar la [extensión de PageSpeed Insights para Chrome](#) con la que generar una versión optimizada del código HTML. Analiza tu página HTML y busca la regla "Minificar HTML". Haz clic en "Ver el contenido optimizado" para obtener el código HTML optimizado.

Para minificar CSS, usa [YUI Compressor](#) y [cssmin.js](#).

Para minificar JavaScript, prueba [Closure Compiler](#), [JSMin](#) o [YUI Compressor](#). Se puede crear un proceso de construcción que utilice estas herramientas para minificar y cambiar el nombre de los archivos de desarrollo, y guardarlos en un directorio de producción.

# HTML - Formularios

Es un elemento de html definido por la etiqueta <form> que permite agrupar múltiples **elementos de ingreso de datos y botones**

Atributos más utilizados

- **method**: indica cómo se envían los valores del formulario. POST y GET
- **action**: indica la ubicación hacia donde será enviada la información

Sólo se enviará la información de los elementos de ingreso de datos

```
<form method="post" action="producto.html">
    <!--aquí van los elementos para ingreso de datos que se envían a la página
    especificada en el atributo action--&gt;
&lt;/form&gt;</pre>
```

Los formularios están delimitados con la etiqueta <form>, que permite reunir varios elementos de un formulario, como botones y casillas de texto.

La etiqueta form tiene dos atributos muy importantes:

## **method**

Indica la manera en que serán enviados los valores del formulario. Los métodos permitidos son POST y GET.

POST Envía los datos de manera oculta. GET envía los datos agregándolos a la dirección URL y los separa de la URL base con un signo de interrogación

## **action**

Indica la ubicación a la cual será enviada la información. Puede ser un correo electrónico un script, inclusive, otra página HTML

```
<form method="post" action="file2.html"></form>
```

Dentro de un formulario se pueden insertar cualquier elemento HTML en como texto, botones, tablas y enlaces, pero los elementos interactivos son los más interesantes

# HTML - Elementos de ingreso de datos

## FULL COMPATIBLE

- input (previo a html 5)
  - text
  - password
  - button
  - submit
  - reset
  - image
  - hidden
  - radio
  - checkbox
- textarea
- button
- select

## PARCIALMENTE COMPATIBLE



- input
  - color
  - date, datetime, datetime-local, month
  - email
  - number
  - range
  - search
  - tel
  - time
  - url
  - week

<http://caniuse.com/#search=input>

Atención: no todos los input de html5 son compatibles con los últimos navegadores.

En este link <http://caniuse.com/#search=input> se puede verificar la compatibilidad con los diferentes navegadores y versiones de los elementos de html.

# HTML - atributos nuevos para <input>

## HTML 4

value  
readonly  
disabled  
size  
maxlength

## HTML 5

autocomplete  
autofocus  
form  
formaction  
formenctype



## HTML 5

formmethod  
formnovalidate  
formtarget  
height and width  
list  
min and max  
multiple  
pattern (regexp)  
placeholder  
required  
step



PAV2 - UTN FRC

En la siguiente pagina se puede ver información de los atributos:

[https://www.w3schools.com/html/html\\_form\\_attributes.asp](https://www.w3schools.com/html/html_form_attributes.asp)

# HTML - Elemento <input> parte 1

Permite ingreso de datos en un cuadro de texto

```
<input type="text" name="Nombre" required maxlength="45" />
```

Para el ingreso de contraseña:

```
<input type="password" name="Clave" placeholder="Clave" value="" />
```

Para establecer valores no visibles al usuario pero que se envían al servidor

```
<input type="hidden" value="valor oculto" name="id" />
```

Permite definir un botón que es una imagen

```
<input type="image" src="img_submit.gif" alt="Submit">
```

## Atributos:

### **type:**

- text: texto claro
- password: El ingreso de los caracteres aparece en la interfaz del navegador de manera oculta, pero a través del DOM se puede acceder al valor que contiene

**name:** identificador para trabajar con el DOM

**required:** requerido

**maxlength:** cantidad máxima de caracteres

**placeholder:** El atributo de placeholder especifica una sugerencia que describe el valor esperado de un campo de entrada (un valor de muestra o una breve descripción del formato). La indicación se muestra en el campo de entrada antes de que el usuario introduzca un valor. Funciona con los siguientes tipos de entrada: texto, búsqueda, url, tel, correo electrónico y contraseña.

**value:** valor inicial que contiene el control

**readonly:** solo lectura, no permite edición

**disabled:** deshabilitado, no se permite edición pero el valor se envía con el formulario

**size:** El atributo size especifica el tamaño (en caracteres) del campo de entrada.

## Atributos nuevos en HTML 5

**pattern:** El atributo pattern especifica una expresión regular que comprueba el valor

del elemento <input>.

El atributo patrón funciona con los siguientes tipos de entrada: texto, búsqueda, url, tel, correo electrónico y contraseña.

**autocomplete**

**autofocus**

**form**

**formaction**

**formenctype**

**formmethod**

**formnovalidate**

**formtarget**

**height and width**

**list**

**min and max**

**multiple**

**pattern (regexp)**

**placeholder**

**required**

**step**

## HTML - Elemento <input> parte 2

Botón para envío de datos comprendidos entre <form> y</form>

```
<input type="submit" name="Enviar" value="Enviar datos" />
```

Botón para ejecución de acciones en javascript

```
<input type="button" name="alerta" value="Mostrar alerta"  
onclick="alert('Alerta!!!!')"/>
```

Otro tipo de botón

```
<button type="button">Click aquí!!!</button>
```

Dentro de un elemento <button> puedes poner contenido, como texto o imágenes. Esta es la diferencia entre este elemento y los botones creados con el elemento <input>. el type puede ser button,reset,submit

## HTML - <input> botones de selección y checkbox

- **Radio:** establecer el mismo identificador para el atributo **name** para funcionamiento en conjunto mutuamente excluyente.

```
<input type="radio" name="nacionalidad" value="argentina" checked> Argentina<br><input type="radio" name="nacionalidad" value="uruguay"> Uruguay<br><input type="radio" name="nacionalidad" value="chile"> Chile<br><input type="radio" name="nacionalidad" value="otro"> Otro<br>
```

<input checked="" type="radio"/> Argentina
<input type="radio"/> Uruguay
<input type="radio"/> Chile
<input type="radio"/> Otro

- **Checkbox:** permite seleccionar algún elemento o ninguno

```
<input type="checkbox" name="idioma1" value="ingles" checked> Ingles<br><input type="checkbox" name="idioma2" value="portugues" checked> Portugues<br>
```

Ingles  
 Portugues

atributo checked: establece de manera predeterminada la opción seleccionada

## HTML - <textarea>

```
<textarea name="observaciones" cols="30" rows="4" placeholder="Ingrese su  
observacion"></textarea>
```

El atributo rows especifica el número visible de líneas en un área de texto.

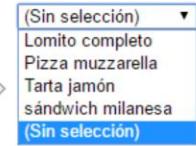
El atributo cols especifica el ancho visible de un área de texto.



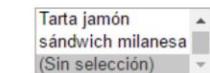
# HTML - Elemento <select>

Permite crear un cuadro de lista desplegable o cuadro de lista

```
<select name="menu">
    <option value="lcom">Lomito completo</option>
    <option value="pmuz">Pizza muzzarella</option>
    <option value="tjam">Tarta jam&oacute;n</option>
    <option value="smil">S&aacute;ndwich milanesa</option>
    <option value="" selected>(Sin selecci&oacute;n)</option>
</select>
```



```
<select name="menu" size="3">
    <option value="lcom">Lomito completo</option>
    <option value="pmuz">Pizza muzzarella</option>
    <option value="tjam">Tarta jam&oacute;n</option>
    <option value="smil">S&oacute;ndwich milanesa</option>
    <option value="" selected>(Sin selecci&oacute;n)</option>
</select>
```



El atributo **selected** permite establecer cuál es la opción seleccionada de manera predeterminada.

El atributo **value** es el que valor que va a tomar el elemento <select> cuando se seleccione la opción del cuadro de lista

Otros atributos:

Attribute	Value	Description
autofocus ( <a href="#">html5</a> )	autofocus	Specifies that the drop-down list should automatically get focus when the page loads
disabled	disabled	Specifies that a drop-down list should be disabled
disabled		
form ( <a href="#">html5</a> )	form_id	Defines one or more forms the select field belongs to
multiple	multiple	Specifies that multiple options can be selected at once
selected at once		
name		Defines a name for the drop-down list
required ( <a href="#">html5</a> )	required	Specifies that the user is required to select a value before submitting the form
size		
visible options in a drop-down list	number	Defines the number of

## Bibliografía

- <https://www.w3schools.com/html/default.asp>
- <https://developers.google.com/web/>
- <https://developer.mozilla.org/es/docs/HTML/HTML5>
- <https://developer.mozilla.org/es/docs/Learn/HTML/Forms>
- <https://www.w3.org/TR/html5/Overview.html>
- <https://validator.w3.org/>

## Unidad 2

# Diseño del frontend con CSS3 y Bootstrap



Universidad Tecnológica Nacional  
Facultad Regional Córdoba  
Cátedra: Programación de Aplicaciones Visuales 2

# CSS3

Hojas de estilo en cascada



# ¿Qué es CSS?

- CSS quiere decir Hojas de Estilo en Cascada (**Cascading Style Sheets**)
- Un Archivo **CSS** es típicamente un archivo de texto con extensión .css que contiene una serie de comandos y reglas
- Estas reglas le dicen a HTML como se debe mostrar

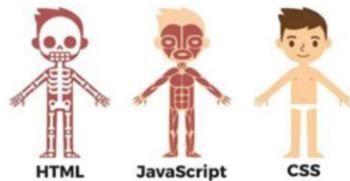
```
body {  
font-family:Arial;  
background: #000;  
}  
  
#contenido {  
text-align:left;  
width:1020px;  
}  
  
#encabezado{  
height:232px;  
}  
  
#pie{  
width: 100%;  
padding: 0 10px;  
margin-bottom: 10px;
```

# HTML y CSS

HTML y CSS trabajan en conjunción para ofrecer características **estéticas y funcionales** a los sitios Web

**HTML** = estructura

**CSS** = estilo



# Adjuntar una hoja de estilos

Hay tres formas de agregar CSS a un documento HTML

Inline

```
<p style="color: red">Algún texto</p>
```

Hoja de estilos interna

```
<style type="text/css">
h1 {color: red}
</style>
```

Hoja de estilos externa

```
<link rel="stylesheet" type="text/css" media="all" href="css/styles.css" />
```

# Estructura de una regla CSS

Una regla CSS consiste en un selector, seguido por una o más declaraciones.

Una declaración es un par propiedad valor.



## Selectores: selector Tag

- Corresponde al tag HTML del elemento
- Se aplica en todo el documento

```
body { property: value; }
h1 { property: value; }
em { property: value; }
p { property: value; }
```

## Selectores: selector ID

- El selector id se usa para especificar el estilo para un único elemento.
- El selector id usa el atributo id del elemento HTML y se define con un "#"

```
/* Esta regla se aplica al elemento con id = "para1" */  
#para1 {  
    text-align:center;  
    color:red;  
}
```

## Selectores: selector Class

- El selector class se usa para especificar un estilo para un grupo de elementos. A diferencia del selector id, el selector class se utiliza para aplicar estilo a múltiples elementos.
- Esto permite tener un conjunto de estilos para muchos elementos de la misma clase.
- El selector class utiliza el atributo HTML class y se define con un ":".

```
/* Este estilo se aplica a todos los elementos con class="center"
*/
.center {text-align:center;}
```

## Demo 4.1.

Demostración de asociar hojas de estilo y aplicar estilos

Paso 1: Abrir el archivo HolaMundo.html (o cualquier documento html simple)

Paso 2: Agregar un estilo inline y refrescar el documento en el browser

Paso 3: Agregar un estilo a nivel de página y visualizar el resultado

Paso 4: Crear el archivo de hoja de estilos: estilos.css y adjuntarla al documento

Paso 5: Crear un estilo inline para mostrar que redefine el estilo definido en el archivo css

## Otros selectores: pseudo clases

:first-child

:last-child

:nth-child

:first-of-type

:last-of-type

:nth-of-type

Hermana adyacente +

Todas las hermanas ~

:not

:active

:checked

:disabled

## Otros selectores: atributo

Se puede usar como selector la presencia o el valor de un atributo

```
/* todos los elementos <a> que tengan el atributo target */
a[target] {
    background-color: yellow;
}

/* todos los elementos cuyo atributo target contenga la palabra 'logo' */
[title*="logo"] {
    background: white;
}
```

# Agrupar selectores

Los estilos

```
h1 {  
color:green;  
}  
h2 {  
color:green;  
}  
p {  
color:green;  
}
```

Se pueden agrupar como

```
h1,h2,p {  
color:green;  
}
```

## Anidar selectores

Se puede aplicar un estilo para un selector dentro de otro selector

```
p  
{  
color:blue;  
text-align:center;  
}  
.marked  
{  
background-color:red;  
}  
.marked p  
{  
color:white;  
}
```

## Orden de la cascada

¿Qué estilo se aplica cuando hay varios estilos definidos para un mismo elemento?

1. Browser default
2. Style sheet externa
3. Style sheet interna
4. Style sheet inline

## Utilizar Chrome DevTools

- Ctrl+Shift+C o F12
- Inspeccionar el DOM y CSS
- Modificar estilos en tiempo real

## Demo 4.2.

Demostración de Chrome DevTools

Paso 1: Abrir el documento HolaMundo.html y abrir DevTools

Paso 2: Seleccionar el tab **Elements**

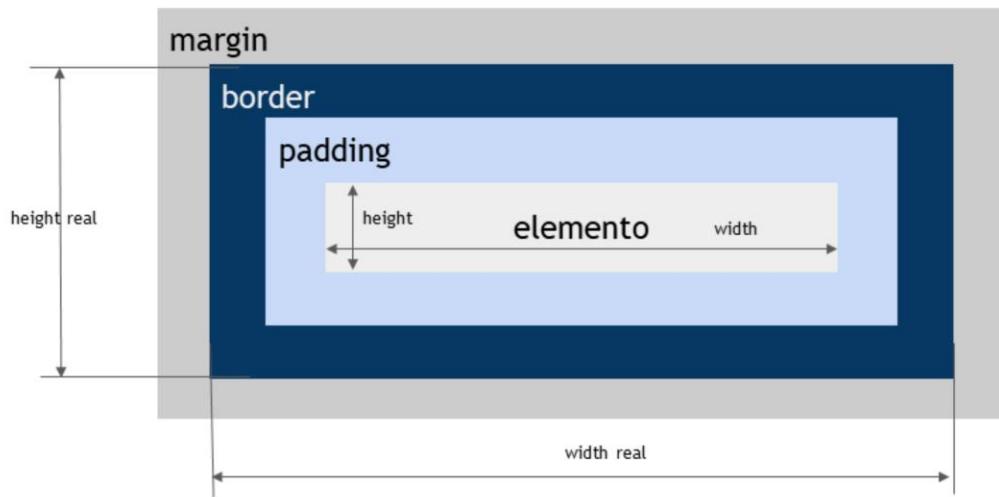
Paso 3: Seleccionar la herramienta puntero y mostrar las propiedades de los distintos elementos del documento

Paso 4: Seleccionar un elemento y visualizar las ventanas **Styles** y **Computed**

Paso 5: Editar un estilo en tiempo real y verificar los cambios en el documento

Paso 6: Para el elemento que tiene dos estilos aplicados (uno inline y otro en el archivo css) mostrar como se puede seleccionar y deseleccionar los estilos y verificar de donde provienen

# CSS Box Model



## height y width

- se utilizan para especificar el alto y el ancho de un elemento
- pueden tomar el valor auto o un valor específico en % o px

```
div {  
    height: 200px;  
    width: 50%;  
}  
  
div {  
    width: auto;  
}
```

## border

- Tiene propiedades de estilo, ancho y color (border-style, border-width, border-color)
- se pueden especificar de forma abreviada o individualmente

```
p{  
    border-style: solid;  
    border-width: 2px;  
    border-color: black;  
}  
  
div{  
    border: 5px solid red;  
}  
  
div{  
    border-width: 5px 3px 2px 7px;  
}
```

# padding

- se usa para generar un espacio alrededor de un elemento

```
p {  
    padding-top: 50px;  
    padding-right: 30px;  
    padding-bottom: 50px;  
    padding-left: 80px;  
}  
  
div{  
    padding: 5px;  
}  
  
div{  
    padding: 5px 5px 7px 7px;  
}
```

# margin

- se usa para generar espacio entre los elementos

```
p {  
    margin-top: 12px;  
    margin-right: 12px;  
    margin-bottom: 16px;  
    margin-left: 14px;  
}  
  
div{  
    margin: 10px;  
}  
  
div{  
    margin: 15px 15px 17px 17px;  
}
```

## Demo 4.3.

Demostración del box model

# Colores

Los colores en CSS se pueden definir de tres formas distintas

- Un valor hexadecimal como "#ff0000"
- Un valor RGB como "rgb(255,0,0)"
- Un nombre de color como "red"

# Background

- background-color
- background-image
- background-repeat
- background-attachment
- background-position

```
background-color: green;  
background-image: url("logo.gif");  
background-repeat: repeat-x;  
background-position: right top;  
background-attachment: fixed;
```

# Texto

- color
- text-align
- text-decoration
- text-transform
- text-indent

```
color: blue;  
text-align: left; /* center | right | justify */  
text-decoration: overline; /* line-through | underline */  
text-transform: uppercase; /* lowercase | capitalize */  
text-indent: 50px;
```

# Fuente

- font-family
- font-style
- font-size

```
font-family: "Times New Roman", Times, serif;  
font-style: normal; /* italic | oblique */  
font-size: 40px;  
font-size: 1.2em;  
font-weight: bold; /* normal */
```

## Ejercicio 4.1.

Modificar la apariencia de un documento html con css

# Bootstrap 4.4

Framework



# Necesidad de Bootstrap

- No en todos los proyectos web tenemos un diseñador web
  - Y a pesar de las buenas intenciones no siempre sale todo bien:  
<http://www.lingscars.com/>
- Podemos aprender de los cientos de sitios web bien diseñados
  - Hay ciertos elementos que se repiten en la mayor parte de ellos
    - Navegación
    - Contenido
    - Pie de página
    - etc

Ver templates en <https://expo.getbootstrap.com/>

## ¿Qué es bootstrap?

Bootstrap es un framework open source para construir sitios web

- CSS
- JavaScript
- Modular
- Mobile First
- Open Source
- Cubre la mayoría de los elementos constitutivos de un website

## Diseño web responsive y mobile-first

- **Responsive web design:** diseño web capaz de adaptarse a un amplio rango de dispositivos:
  - Computadora
  - Tablet
  - Smartphone
  - etc
- Más información en: <http://www.lukew.com/ff/entry.asp?1514>
- **Mobile-first:** Bootstrap 4.x adopta un abordaje que permite que los diseños se visualicen correctamente en dispositivos móviles desde su concepción

# Empezar con Bootstrap 1/3...

## 1. Bajar bootstrap

Compilado y minimizado desde: <https://getbootstrap.com/docs/4.4/getting-started/download/>

Alternativamente se pueden referenciar la CDN (Content Delivery Network).  
<https://getbootstrap.com/docs/4.4/getting-started/download/#bootstrapcdn>

También necesitamos hacer referencia a dos dependencias:

- JQuery
- Popper.js

# Empezar con Bootstrap 2/3...

## 2. Incluir las referencias en nuestra página web

```
<!-- Bootstrap CSS -->
<link href="css/bootstrap.min.css" rel="stylesheet" />
```

```
<!-- jQuery (necesario para los plugins) -->
<script src="js/jquery-3.3.1.js"></script>

<!-- Incluye todos los plugins, también se pueden incluir plugins individuales -->
<script src="js/bootstrap.bundle.min.js"></script>
```

# Empezar con Bootstrap 3/3...

## 3. Utilizar la plantilla de inicio: starter template

<https://getbootstrap.com/docs/4.3/getting-started/introduction/#starter-template>

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8xKXpsPk8YIwqHZLSpOmqO0YHnSUZfHDc4tFQ8aMnKTbGrKUhtkuZWpJj" crossorigin="anonymous">

    <title>Hello, world!</title>
  </head>
  <body>
    <h1>Hello, world!</h1>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa484bIE2+poT4khyKhv5vZFS5rPo0IEjw@VKU7imGFAV@wwj1yYfoRSjoZ+n" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-QGE9RhbIzZFJef(+2nJbHaEWid1vI91OYy5n3zV9zzTmI3UksdQRVvoxFroko" crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wf5DF2E558Y1Wf7J8qBZ3D7C4D0IaMwqOj9vLk+X3TqPfPZ7zXn+jwD0D8GCCx130g81#W6" crossorigin="anonymous"></script>
  </body>
</html>
```

Si vamos a empezar un documento desde cero podemos partir del starter template que incluye los elementos necesarios para tener un documento con Bootstrap 4.3

## Demo 4.4.

Demostración como incluir bootstrap en un sitio web

**Objetivo: comprender cómo incluir bootstrap en un sitio web**

Paso 1: Bajar Bootstrap compilado y minimizado desde:

<http://getbootstrap.com/getting-started/>

Paso 2: Crear un proyecto web

Paso 3: Crear un archivo html

Paso 4: Copiar y pegar la plantilla básica (basic template)

Paso 5: <http://getbootstrap.com/getting-started/#template>

Paso 6: Verificar que los enlaces estén correctos

Paso 7: Visualizar la página en el browser

## Demo 4.5.

Obtener un panorama general de las posibilidades y características de Bootstrap

**Objetivo: adquirir un panorama de las posibilidades del framework, su sintaxis y sus características generales**

Paso 1: Inspeccionar los distintos ejemplos en:

<https://getbootstrap.com/docs/4.4/examples/>

Paso 2: Entrar a algunos de ellos y visualizar el código fuente

Paso 3: Eventualmente copiar y pegar el código fuente en el proyecto del ejercicio anterior y realizar cambios para experimentar con los distintos elementos

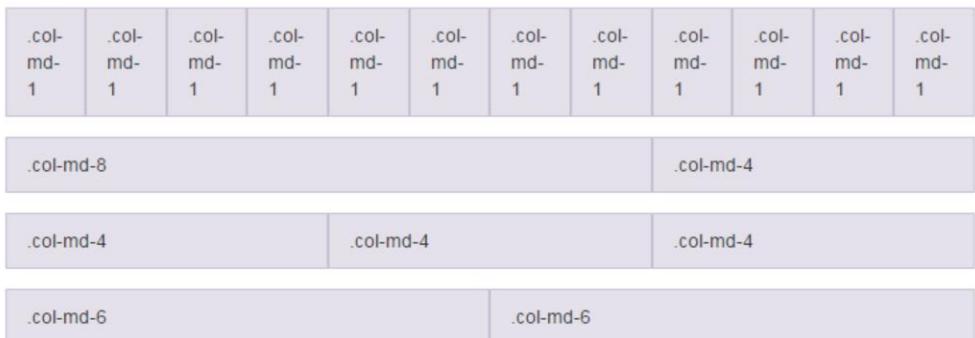
## Doctype de html 5

Bootstrap utiliza ciertos elementos de HTML y propiedades de CSS que necesitan usar el doctype de HTML5. Debemos colocarlo al inicio de nuestros proyectos.

```
<!DOCTYPE html>
<html lang="en">
...
</html>
```

## Sistema de grilla

Bootstrap utiliza un sistema de grilla fluida que escala hasta 12 columnas en la medida en que el tamaño del dispositivo o viewport aumenta.



<https://getbootstrap.com/docs/4.4/examples/grid/#containers>

<https://getbootstrap.com/docs/4.4/examples/grid/#containers>

# Contenedores

Los contenedores son el elemento básico de la distribución del espacio en Bootstrap y son necesarios cuando trabajamos con el sistema de grillas.

Tenemos dos tipos de contenedores disponibles:

- De ancho fijo

```
<div class="container">  
  <!-- contenido -->  
</div>
```

- Ancho total

```
<div class="container-fluid">  
  <!-- contenido -->  
</div>
```

<http://getbootstrap.com/examples/grid/>

## Sistema de grilla: viewports

El sistema de grilla de bootstrap tiene cinco clases

<b>xs</b>	Teléfonos móviles modo vertical	< 576 px
<b>sm</b>	Teléfonos móviles modo landscape	$\geq 576$ px
<b>md</b>	Dispositivos medianos (tablets)	$\geq 768$ px
<b>lg</b>	Desktops	$\geq 992$ px
<b>xl</b>	Desktops grandes	$\geq 1200$ px

## Sistema de grilla: Opciones de configuración

	Extra pequeño <576px	Pequeño ≥576px	Mediano ≥768px	Grande ≥992px	Extra grande ≥1200px
Ancho máximo del contenedor	Ninguno (auto)	540px	720px	960px	1140px
Prefijo de clase	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
# de columnas	12				
Ancho de canal	30px (15px a cada lado de una columna)				
Anidables	si				
Ordenamiento de columnas	si				

Ver ejemplos en <https://getbootstrap.com/docs/4.4/examples/grid/#containers>

## Sistema de grilla

- Bootstrap trabaja con un sistema de 12 columnas
- Las filas se deben colocar dentro de un `.container` (ancho fijo) o `.container-fluid` (ancho completo).
- Las filas `.row` se utilizan para crear grupos horizontales de columnas `.col`
- Se debe colocar el contenido entre columnas y solamente estas pueden ser los hijos inmediatos de las filas.

## Celdas de igual ancho

```
<div class="row">
    <div class="col">celda 1 de 2</div>
    <div class="col">celda 2 de 2</div>
</div>
<br />
<div class="row">
    <div class="col">celda 1 de 3</div>
    <div class="col">celda 2 de 3</div>
    <div class="col">celda 3 de 3</div>
</div>
<br />
<div class="row">
    <div class="col">celda 1 de 4</div>
    <div class="col">celda 2 de 4</div>
    <div class="col">celda 3 de 4</div>
    <div class="col">celda 4 de 4</div>
</div>
```

celda 1 de 2	celda 2 de 2		
celda 1 de 3	celda 2 de 3	celda 3 de 3	
celda 1 de 4	celda 2 de 4	celda 3 de 4	celda 4 de 4

## Celdas de ancho determinado

celda de 4 unidades	celda de 8 unidades	
celda de 3 unidades	celda de 3 unidades	celda de 6 unidades

```
<div class="row">
    <div class="col-4">celda de 4 unidades</div>
    <div class="col-8">celda de 8 unidades</div>
</div>
<br />
<div class="row">
    <div class="col-3">celda de 3 unidades</div>
    <div class="col-3">celda de 3 unidades</div>
    <div class="col-6">celda de 6 unidades</div>
</div>
```

## Sistema de grilla

`.container o .container-fluid`

.row

# Sistema de grilla y los viewports

## .col-sm-8

<b>xs</b>	Teléfonos móviles modo vertical	< 576 px
<b>sm</b>	Teléfonos móviles modo landscape	$\geq 576$ px
<b>md</b>	Dispositivos medianos (tablets)	$\geq 768$ px
<b>lg</b>	Desktops	$\geq 992$ px
<b>xl</b>	Desktops grandes	$\geq 1200$ px

# Sistema de grilla y los viewports

## .col-xs-4

<b>xs</b>	Teléfonos móviles modo vertical	< 576 px
<b>sm</b>	Teléfonos móviles modo landscape	$\geq 576$ px
<b>md</b>	Dispositivos medianos (tablets)	$\geq 768$ px
<b>lg</b>	Desktops	$\geq 992$ px
<b>xl</b>	Desktops grandes	$\geq 1200$ px

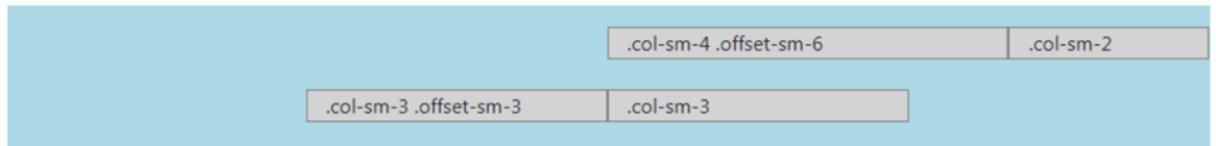
# Sistema de grilla y los viewports

.col-sm-4 .col-lg-2

<b>xs</b>	Teléfonos móviles modo vertical	< 576 px
<b>sm</b>	Teléfonos móviles modo landscape	$\geq 576$ px
<b>md</b>	Dispositivos medianos (tablets)	$\geq 768$ px
<b>lg</b>	Desktops	$\geq 992$ px
<b>xl</b>	Desktops grandes	$\geq 1200$ px

# Desplazamiento de celdas

.offset-sm-4



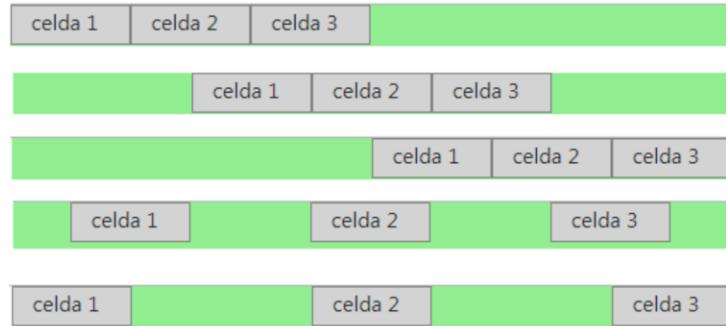
# Alineación vertical

```
<div class="row align-items-start">  
<div class="row align-items-center">  
<div class="row align-items-end">
```

celda 1	celda 2	celda 3
celda 1	celda 2	celda 3

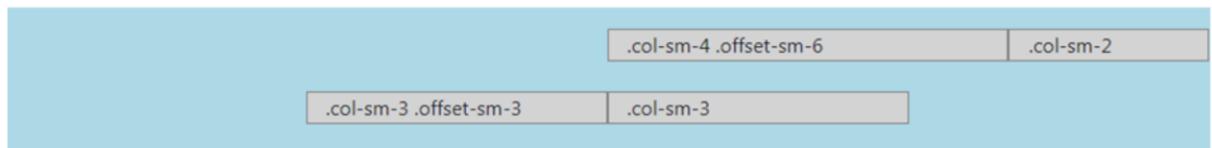
# Alineación horizontal

```
<div class="row justify-content-start">  
<div class="row justify-content-center">  
<div class="row justify-content-end">  
<div class="row justify-content-around">  
<div class="row justify-content-between">
```



# Desplazamiento de celdas

.offset-sm-4



## Ocultar y visibilizar celdas

```
.d-none          // oculta la celda  
  
.d-lg-none      // oculta la celda en  
lg  
  
.d-xl-block    // muestra la celda en xl como  
block  
  
.d-md-inline   // muestra la celda en md como
```

 PAZ-UN FRC

Alinear contenido en una celda

**.float-left**

**.float-right**

**.float-none**

**.mx-auto**

también se pueden combinar con el tamaño de viewport

**.float-md-left**

## Ejercicio 4.2.

Objetivo: experimentar con los elementos fundamentales del sistema de grillas de bootstrap

1. Crear una grilla con la distribución que se muestra a continuación
2. Modificar el tamaño del browser y verificar cómo la grilla cambia en función de este tamaño

Celda 1 - ancho 4	Celda 2 - ancho 4	Celda 3 - ancho 4
Celda 4 - ancho 2	Celda 5 - ancho 10	
	Celda 6 - ancho 4	Celda 7 - ancho 4

# Tipografía

Encabezados **<h1>** - **<h6>**

Clases **.h1** - **.h6**

Encabezados **.display-1** **.display-4**

Texto destacado **.lead**

**<small>** crea un elemento secundario en un encabezado

**<mark>** resalta un texto

**<del>** texto eliminado

**<blockquote>** define una cita

**<code>** permite especificar código fuente

**<kbd>** permite definir elementos del teclado

# Tipografía

.text-muted .text-primary .text-success .text-info .text-warning .text-danger

Este texto está silenciado.

Este texto es importante.

Este texto indica éxito.

Este texto representa información.

Este texto representa una advertencia.

Este texto representa peligro.

.text-center .text-right

.bg-primary .bg-success .bg-info .bg-warning .bg-danger

Este texto es importante.

Este texto indica éxito.

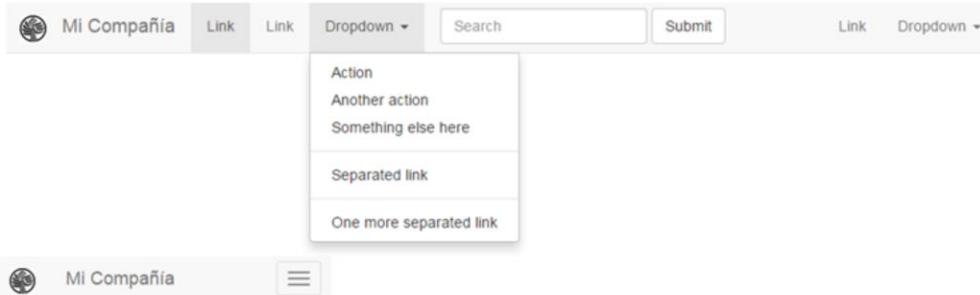
Este texto representa información.

Este texto representa una advertencia.

Este texto representa peligro.

# Navegación

.navbar  
.navbar-default  
.navbar-toggler  
.navbar-brand



Ejemplo:  
<http://jsfiddle.net/KyleMit/k98Bn/>

## Imágenes

```
.img-fluid  
.img-thumbnail
```

Imagen responsive

```
class="img-fluid" style="width: 100%;height: auto;"
```

# Botones

Las clases de botón se pueden aplicar a los siguientes elementos: `<a>`, `<button>`, `<input>`

```
.btn-default  
.btn-primary  
.btn-success  
.btn-info  
.btn-warning  
.btn-danger  
.btn-link
```



# Botones

Tamaño

.btn-lg  
.btn-md  
.btn-sm  
.btn-xs

Large      Medium      Small      XSmall

Activo / Deshabilitado

.active  
.disabled

Activo      Deshabilitado

Ejercicio: buscar como agregar un icono a los Botones...

## Tablas

```
.table  
.table-dark  
.thead-dark .thead-light  
.table-striped /* para hacer que las filas alternen color */  
.table-bordered .table-borderless  
.table-hover /* fondo de color gris sobre fila posicionada */  
.table-condensed /*elimina espacios entre filas */
```

# Formularios

```
.form-group .form-inline  
.form-control placeholder=""
```

## Formularion Básico

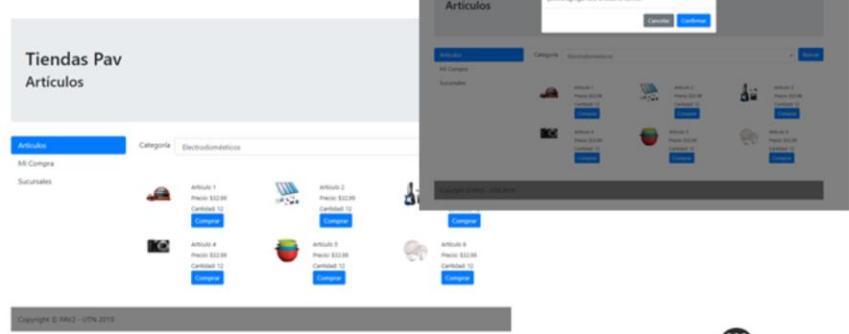
Email:

Password:

Recordarme

## Ejercicio 4.3.

1. A partir del documento tiendas.html modifique su apariencia y distribución usando el sistema de grilla de bootstrap y elementos adicionales
2. Verifique que la distribución adoptada sea compatible con todos los tamaños de dispositivos



# Documentación

En español

- [http://librosweb.es/libro/bootstrap\\_3/](http://librosweb.es/libro/bootstrap_3/)

En inglés

- <https://www.w3schools.com/bootstrap4>

Documentación (en inglés)

- <https://getbootstrap.com/docs/4.4/getting-started/introduction/>

Temas gratuitos (en inglés)

- <https://bootswatch.com/>

# Recursos

Diretorios de recursos

- <http://bootsnipp.com/resources>
- <http://startbootstrap.com/bootstrap-resources/>

Templates (plantillas)

- <http://startbootstrap.com/>

Code snippets

- <http://bootsnipp.com/>

## Unidad 3

# Javascript y jQuery



Universidad Tecnológica Nacional  
Facultad Regional Córdoba  
Cátedra: Programación de Aplicaciones Visuales 2

# Javascript

El lenguaje



# Javascript y los browsers

Lo más difícil de usar JavaScript

- El problema puede ser la implementación de DOM del browser
- Los motores de JavaScript son diferentes

Estrategias

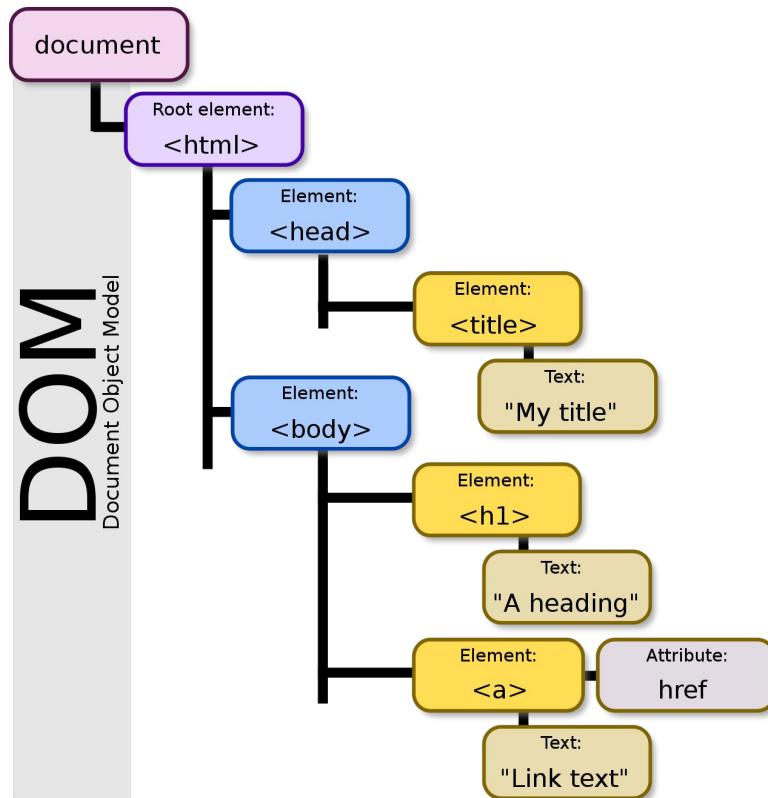
- Escribir código compatible con los estándares
- Escribir código que tenga en cuenta el entorno (complicado → mucho código)

JavaScript puede no estar disponible

- Deshabilitado o no disponible en el browser



# Javascript y DOM



# Herramientas

- Google Chrome DevTools / Firebug
- JSLint (<http://www.jslint.com/>)
- JSHint (<http://jshint.com/>)
- Mozilla Developer Network:  
<https://developer.mozilla.org/es/docs/Web/JavaScript>



# El elemento script en HTML

- Es clave a la hora de usar JavaScript en la página
- Dos formas de usarlo:
  - Inline

```
<script>
    console.log('Hola pagina');
</script>
```

- Archivo JavaScript externo

```
<script src="jquery-1.7.1.js" />
```

- De forma opcional se puede especificar el contenido
  - text/javascript
- Se pueden incluir tantos scripts como sea necesario



# ¡Hola mundo!

## `write()`

```
<script language=javascript>
    document.write("¡Hola Mundo!");
</script>
```

## `alert()`

```
<script language=javascript>
    alert("¡Hola Mundo!");
</script>
```



# Demo 5.1.

Hola Mundo utilizando el elemento script



# Tipos de dato

JavaScript no es un lenguaje fuertemente tipado (strongly typed)

Javascript soporta los siguientes tipos de datos core:

- string
- number (integer y float)
- boolean
- null
- undefined
- object



# Tipos de dato

Hay otros tipos de datos, pero no pertenecen al core sino que son especializaciones del tipo object, se los denomina built-in objects:

- String
- Boolean
- Number
- Date
- Array
- Object
- Function
- RegEx
- Error



# Variables

## Crear una variable

```
var a = "cadena";
var b = 15;
var c = 12.3, d = "otra cadena"
```

## Asignar una variable

```
a = "nueva cadena";
b = 2555;
```

# Variables

¿Y si no pongo var?

```
nueva = 25;
```

En el ámbito global no hay diferencias

En el ámbito local de una función si ponemos var creará una variable local, de lo contrario buscará en la cadena de ámbitos y cuando la encuentre asignará esa variable. Si no la encuentra crea la variable global.



# Variables: tipos

typeof es un operador que devuelve el tipo básico de una variable

```
var a = 25;  
typeof 25; /* devuelve: "number" */
```

La propiedad constructor devuelve el constructor de la variable y puede servir para tener información más precisa sobre el tipo de variable de que se trata.

```
var b = new Date(2017,01,01);  
typeof b; /* devuelve: "object" */  
b.constructor; /* devuelve function Date() { [native code] } */
```



# Variables: conversión de tipos

En JavaScript la conversión de tipos se puede dar por:

- **Explicitamente:** Usando una función de conversión
- **Implicitamente:** por JavaScript automáticamente



# Variables: conversión a string

Se pueden convertir datos a string usando:

- la función global: `String()`
- el método `.toString()`

```
String(a);
String(123);
x.toString();
(123).toString();

String(false);
String(true);
false.toString();
true.toString();

String(Date());
Date().toString();
```



# Variables: conversión a number

Se pueden convertir datos a number usando:

- la función global: `Number()`
- el método `.parseInt()`
- el método `.parseFloat()`
- el operador unario `+`

```
Number("210"); /* devuelve 210 */  
Number("21ax"); /* devuelve NaN */  
  
parseInt("21ax") /* devuelve 21 */  
parseInt("aa") /* devuelve NaN */  
  
parseFloat("12.21")  
parseFloat("15.aa") /* devuelve 15 */
```



# Variables: conversión a number

```
+ "32";  
+ "21a" /* devuelve NaN */  
  
Number(false)      /* devuelve 0 */  
Number(true)       /* devuelve 1 */  
  
d = new Date();  
Number(d)          /* devuelve 1404568027739 */
```



# Operadores aritméticos

+	suma
-	resta
*	producto
%	módulo
++	incremento
--	decremento



# Operadores lógicos

&&	and
	or
!	not
?:	operador ternario



# Operadores de comparación

<code>==</code>	igualdad
<code>!=</code>	desigualdad
<code>&gt;</code>	mayor
<code>&lt;</code>	menor
<code>&gt;=</code>	mayor o igual
<code>&lt;=</code>	menor o igual
<code>====</code>	estrictamente iguales
<code>!==</code>	no estrictamente iguales



# Sentencias condicionales: if / if..else

```
if (condicion) {  
    código que se ejecuta si la condición es verdadera  
}
```

```
if (condicion) {  
    código que se ejecuta si la condición es verdadera  
} else {  
    código que se ejecuta si la condición es falsa  
}
```



# Sentencias condicionales: else if

```
if (condicion1) {  
    código que se ejecuta si la condición1 es verdadera  
} else if (condicion2) {  
    código que se ejecuta si la condición1 es falsa y la  
    condición2 es verdadera  
} else {  
    código que se ejecuta si la condición1 es falsa y la  
    condición2 es falsa  
}
```



# Sentencias condicionales: switch

```
switch(expresión) {  
    case n:  
        código  
        break;  
    case m:  
        código  
        break;  
    default:  
        código por defecto  
}
```



# Sentencias iterativas: for

```
for (sentencia1; sentencia2; sentencia3) {  
    código  
}
```

```
for (var i = 0; i < Cosas.length; i++) {  
    Cosas[i]  
}
```



# Sentencias iterativas: for in

```
/* Recorre las propiedades de un objeto */

for (var propiedad in Cosa) {
    Cosa[propiedad]
}
```



# Sentencias iterativas: while

```
while (condición) {  
    código  
}
```

```
do {  
    código  
} while (condición);
```



# Sentencias iterativas: break y continue

**break:** sale de la iteración y continúa ejecutando la sentencia siguiente, si la hay:

```
break;
```

**continue:** omite la iteración corriente y continúa la iteración siguiente en el caso de que la condición del bucle se satisfaga.

```
continue;
```



# Arrays

Sintaxis:

```
var nombre-array = [item1, item2, ...];
```

Ejemplo:

```
var colores = ["Verde", "Rojo", "Azul"];
```

Acceder a los elementos de un array:

```
var color1 = colores[0];
colores[2] = "Amarillo";
```



# Arrays

Los arrays son una clase especial de objeto cuyas propiedades son valores numéricos.

Como reconocer un array:

```
Array.isArray(colores); // Solo en ECMAScript 5 o sup
```

```
colores.constructor.toString().indexOf("Array") > -1;
```

```
colores instanceof Array;
```



# Arrays: propiedades y métodos

`length`: devuelve el tamaño del array

`push()`: agrega un elemento al final

`pop()`: remueve el último elemento

`unshift()`: agrega un elemento al principio del array

`shift()`: remueve el primer elemento y mueve el resto de los elementos

`delete`: elimina un elemento (dejando un elemento `undefined`)

```
delete colores[1];
```

`splice(posicion, elementos_a_remover, nuevo1, nuevo2, ...)`: agrega elementos a un array removiendo aquellos especificados

`slice(posicionInicio, posicionFin)` Obtiene n elementos de un array

`sort()`: ordena el array

`reverse()`: invierte el orden de los elementos del array



# Demo 5.2.

Usar arrays en javascript



# Funciones

```
function nombre(par1, par2, par3) {  
    /* código */  
}
```

```
function sumar(a,b){  
    return a+b;  
}
```



# Funciones

Para invocar una función se utiliza el operador ()

```
var a = sumar(2,3);
```

Se puede asignar una función como cualquier objeto

```
var c = sumar;  
d = c(5,6); /* d contendrá 11 */
```



# Objetos

Se puede definir un objeto asignando sus propiedades y métodos

```
var persona = new Object();
persona.nombre = "Juan";
persona.apellido = "Perez";
persona.nombreCompleto = function(){
    return this.nombre + " " + this.apellido;
};
```

También se puede declarar y definir mediante literales

```
var auto = {marca:"Ford", modelo:"Focus", año:2016};
```



# Objetos

Se puede acceder a las propiedades mediante la notación punto

```
var m = auto.marca;  
auto.marca = "Fiat";
```

O usando corchetes [ ]

```
var m = auto["marca"];  
auto["marca"] = "Fiat";
```

# JSON

JSON (se pronuncia Yeison) viene de JavaScript Object Notation

- Es un formato de intercambio de datos independiente del lenguaje
- Es auto-descripto
- Es fácil de entender

```
{  
  "clientes": [  
    {"nombre": "Juan", "apellido": "Perez"},  
    {"nombre": "Pedro", "apellido": "Garcia"},  
    {"nombre": "Maria", "apellido": "Sanchez"}  
  ]  
}
```



# JSON

Sintaxis:

- Los datos están como pares nombre/valor
- Los datos se separan por coma
- Las llaves contienen objetos
- Los corchetes contienen arrays

Convertir texto JSON a un objeto JavaScript y viceversa

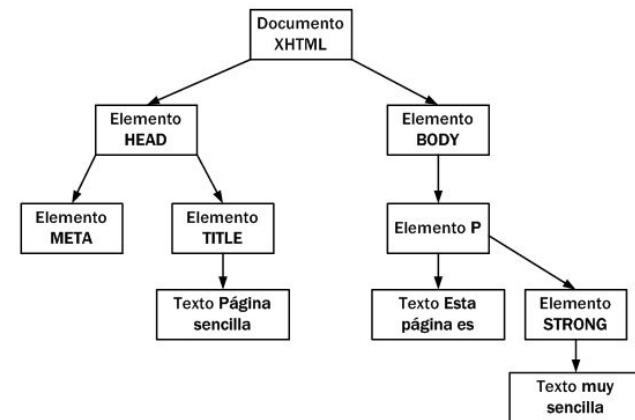
```
var objeto = JSON.parse(textoJson);
var otroTextoJson = JSON.stringify(objeto);
```



# DOM

## DOM (Document Object Model) Modelo de Objetos del Documento

- Es una API para documentos validos HTML y bien construidos XML
- Permite el acceso a la estructura de una página HTML mediante el mapeo de los elementos de esta página en un árbol de nodos.
- define:
  - Los elementos HTML como objetos
  - Las propiedades de todos los elementos HTML
  - Los métodos para acceder a todos los elementos HTML
  - Los eventos para todos los elementos HTML



# DOM: Tipos de nodo

La especificación DOM define 12 tipos de nodo, pero los de uso más habitual son:

- **Document:** nodo raíz del que derivan todos los demás nodos del árbol
- **Element:** representa cada una de las etiquetas HTML
- **Attr:** representa cada uno de los atributos de los elementos HTML
- **Text:** contiene el texto encerrado por una etiqueta HTML.
- **Comment:** representa cada comentario en el documento.



# DOM: Encontrar nodos

`getElementById()`

```
var cabecera = document.getElementById("cabecera");
```

`getElementsByTagName()`

```
var parrafos = document.getElementsByTagName("p");
var primerParrafo = parrafos[0];
```

`getElementsByClassName()`

```
var parrafos = document.getElementsByClassName("resaltado");
```



# DOM: Cambiar nodos

## innerHTML

```
document.getElementById("cabecera").innerHTML = "Cabecera";
```

## Atributos

```
document.getElementById("foto").src = "caripela.jpg";
document.getElementById("btnOk").value = "Aceptar";
```

## Estilos CSS

```
document.getElementById("p2").style.color = "blue";
document.getElementById("demo").style.textAlign = "center";
```



# DOM: Eventos

```
document.getElementById("btnFecha").onclick = mostrarFecha;  
  
function mostrarFecha() {  
    document.getElementById("txtFecha").innerHTML = Date();  
}
```

```
<div onmouseover="mOver(this)" onmouseout="mOut(this)"  
style="background-color:red">  
Pasar el mouse</div>  
  
<script>  
function mOver(obj) {obj.innerHTML = "Excelente!"}  
function mOut(obj) {obj.innerHTML = "Pasar el mouse"}  
</script>
```



# Ejercicio 5.1.

Cargar un listado de artículos contenidos en un JSON al presionar un botón en el documento HTML



# jQuery

## Framework



PAV2 - UTN FRC

# ¿Qué es jQuery?

- Una librería JavaScript liviana
- Su propósito es permitir usar JavaScript fácilmente
- Soluciona la mayoría de los problemas derivados de usar distintos browsers (cross-browser issues)
- Es fácil de aprender
- Es extensible



# Esencialmente

Nos permite:

- Obtener ciertos elementos html
- Hacer algo con ellos
  - agregar o remover contenido o datos
  - modificar el estilo
  - agregar eventos
  - agregar efectos
  - analizar el contenido



# Usar jQuery - Paso 1

## Agregar la librería a la página:

Referencia a la librería descargada

```
<script src="/jquery/1.10.2/jquery.min.js"></script>
```

Referencia a CDN (Content Delivery Network)

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
```



# Usar jQuery - Paso 2

Agregar código inicial

Esto espera que la página (en realidad la jerarquía DOM) cargue (no queremos código que refiera a elementos que todavía no existen)

```
<script>
    $(document).ready(function() {
        // Acá va el código
    });
</script>
```



# Demo 5.3.

Demostración de `$(document).ready()`

# Demo 5.4.

La documentación de JQuery: [api.jquery.com](http://api.jquery.com)

A screenshot of the jQuery API documentation website. The header features the jQuery logo and navigation links for Download, API Documentation, Blog, Plugins, and Browser Support. A search bar is also present. The main content area is titled "jQuery API" and contains an introduction to the library, a note about upgrading from version 1.9, and a note about other API documentation locations. On the right, there's a "SUPPORT THE PROJECT" button and a message about donations funding the project's development and growth. A sidebar on the left lists categories like Ajax, Attributes, Callbacks Object, Core, and CSS, each with sub-links.

# Sintaxis

```
$("#div_mensaje").show();
```

selector

acción



# Seleccionar

\$()

<http://api.jquery.com/category/selectors/>

# Selectores

- Selector de elemento

```
$("p")
```

- Selector #id

```
$("#seccion_principal")
```

- Selector .class

```
$(".importante")
```



# Selectores: seleccionar múltiples elementos

- Seleccionar varios elementos html

```
$("p,a,div")
```

```
$(".articulos,.detalles")
```

- Seleccionar descendientes

```
$("table tr")
```

- Combinar

```
$("li.itemMenu")
```



# Selectores: seleccionar por atributo

- Se usan los corchetes

```
$("input[value]")
```

```
//todos los elementos input que tienen el atributo value
```

- Seleccionar por el valor de un atributo

```
$("input[type='text'])")
```

```
//todos los textboxes
```

```
$("input[type='text']").attr('placeholder','ingrese texto aqui');
```



# Selectores

<code>\$("")</code>	Selecciona todos los elementos
<code>\$(this)</code>	Selecciona el elemento HTML corriente
<code>\$("p.encabezado")</code>	Selecciona todos los elementos <p> con clase encabezado
<code>\$("p:first")</code>	Selecciona el primer elemento <p>
<code>\$("ul li:first")</code>	Selecciona el primer elemento <li> del primer <ul>
<code>\$("ul li:first-child")</code>	Selecciona el primer <li> de cada <ul>
<code>\$("[href]")</code>	Selecciona todos los elementos que tienen el atributo href
<code>\$("a[target='_blank']")</code>	Selecciona todos los elementos <a> con el valor del atributo target igual a _blank
<code>\$("a[target!='_blank']")</code>	Selecciona todos los elementos <a> con el valor del atributo target distinto de _blank
<code>\$(":button")</code>	Selecciona todos los elementos <button> y los elementos <input type="button">
<code>\$("tr:even")</code>	Selecciona todos los elementos <td> pares
<code>\$("tr:odd")</code>	Selecciona todos los elementos <td> impares
<code>\$("[id\$='txtApellido']")</code>	Selecciona todos los elementos cuyo id termina con txtApellido



# Selectores: Ejemplo

```
$(document).ready(function(){
    $("button").click(function(){
        $("#mensaje").hide();
    });
});
```

# Iterar en los elementos seleccionados

Se usa `each` para iterar los elementos seleccionados

```
.each(function(index,element))
```

# Iterar en los elementos seleccionados

```
$(document).ready(  
    function() {  
        $('div').each(  
            function() {  
                alert($(this).html());  
            } );  
    } );
```



# Demo 5.4.

En cada div del documento agregar el número de div entre paréntesis

# Manipular el DOM

**.html()** //obtiene o asigna el html de un elemento dado

**.attr()** //permite recuperar o asignar valor a un atributo, se pueden modificar varios atributos a la vez pasando un JSON como parámetro

**.css()** //permite recuperar o asignar estilos css se pueden modificar varios a la vez pasando un JSON como parámetro

**.append() .appendTo() .prepend() .prependTo() .remove()** //agregan y remueven nodos antes o después del elementos seleccionado

**.addClass() .hasClass() .removeClass() .toggleClass()** //permiten manipular las clases css de el o los nodos seleccionados



# Eventos

```
$("p").click(function(){  
    // acá va la acción  
});
```

## Eventos DOM más comunes

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload



# Eventos: Ejemplos

```
$("input").blur(function(){
  $(this).css("background-color","#ffffff");
});
```



# Demo 5.5.

Esconder y mostrar alternativamente todos los divs cada vez que se presiona el botón "Mostrar/Ocultar"



# Demo 5.6.

Agregar la funcionalidad de que al presionar un botón, se agrege un texto al formulario de 3 formas distintas: Con html, con Jquery y con DOM



# Observaciones

Ambito de window -> vida de una pagina

## Unidad 3

# TypeScript



Universidad Tecnológica Nacional  
Facultad Regional Córdoba  
Cátedra: Programación de Aplicaciones Visuales 2

# ¿JavaScript o TypeScript?

¿Que lenguaje elijo para la programación del  
front-end?

<https://desarrolloweb.com/articulos/introduccion-a-typescript.html>

<https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>



# Características principales de TypeScript

- Superconjunto “Superset” de Javascript desarrollado por Microsoft
- Los navegadores NO PUEDEN ejecutar código TypeScript
- Se programa en TypeScript, y el código se transpila generando código 100% compatible JavaScript para ejecutarlo en cualquier entorno que soporta JavaScript
- Orientado a objetos.
- Muy utilizado para desarrollo de grandes proyectos.
- Permite detectar errores de programación durante la escritura del código, antes de la ejecución del script.

<https://desarrolloweb.com/articulos/introduccion-a-typescript.html>



PAV2 - UTN FRC

# TypeScript - Resumen

TypeScript agrega a la  
funcionalidad existente de  
JavaScript.....

Tipos de datos

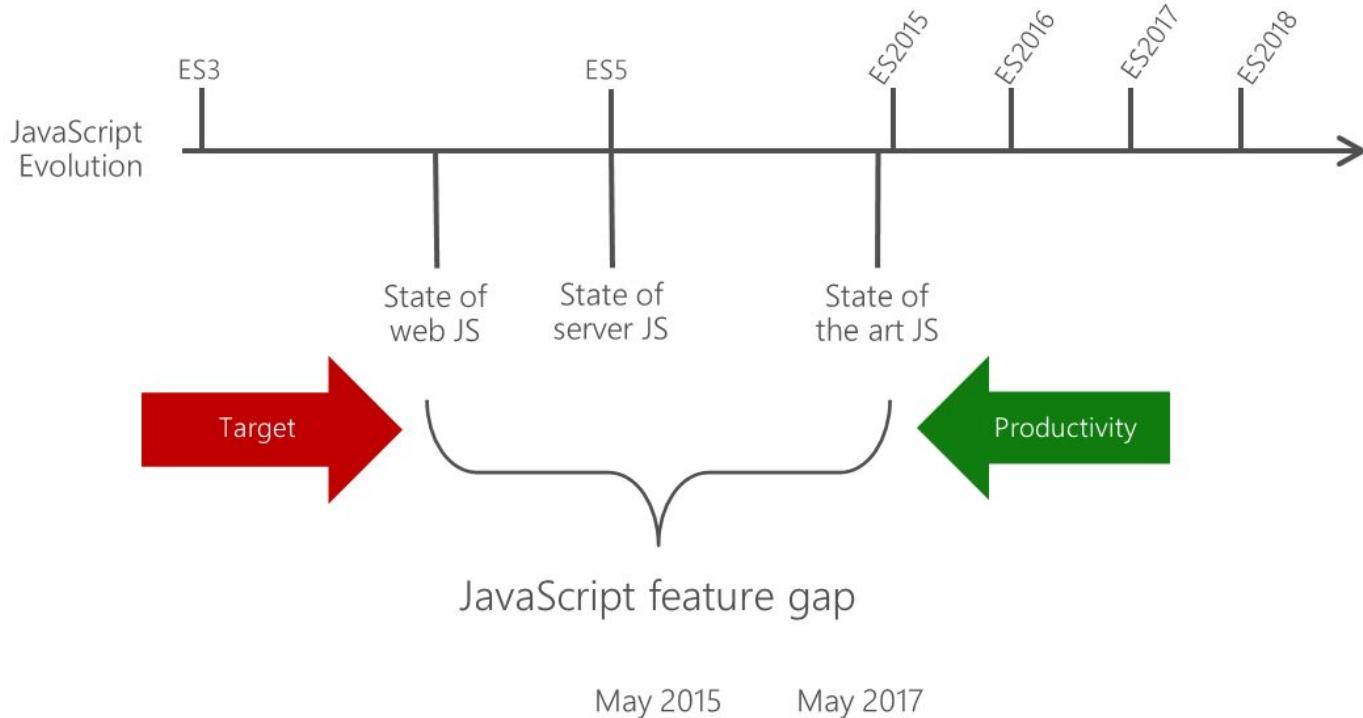
Características de las  
próximas generaciones de  
JavaScript

Interfaces y Generics

Decoradores



# TypeScript - Evolución



# Requerimientos para usar TypeScript

- Conocimientos básicos de JavaScript (Condicionales, funciones, operadores condicionales, bucles, for....)
- Conocimientos básicos de Node.js
  - [Ver tutorial instalación de Node](#)
- Instalar un editor de código
  - [Ver tutorial instalación Visual Studio Code](#)
- Manejo de la consola (npm - Node package manager)

<https://nodejs.org/es/download/>

<https://code.visualstudio.com/>

<https://code.visualstudio.com/docs/typescript/typescript-tutorial>



PAV2 - UTN FRC

# Instalación de TypeScript

- Visual Studio incluye el lenguaje TypeScript pero no el compilador **tsc**
- **tsc** es el transpilador que genera código javascript desde código typescript
- La manera más sencilla de instalar tsc es a través de npm

```
C:\>npm install -g typescript
```

- Verificar la instalación con

```
C:\>tsc --version
```

Version 3.8.3



<https://code.visualstudio.com/docs/typescript/typescript-tutorial>



PAV2 - UTN FRC

Ver demostración paso a paso  
holamundo en TypeScript

[Ver procedimiento aquí](#)



# Hola Mundo desde Visual Studio Code

1. Desde Visual Studio Code crear el archivo holamundo.ts
2. Escribir el siguiente código

```
let message: string = 'Hello World';
console.log(message);
```

3. En la ventana de Terminal: **tsc holamundo.ts**  
Esta instrucción compila el archivo .ts, y genera el archivo .js (JavaScript)
4. En la ventana de Terminal: **node holamundo.js**



# Depuración con Visual Studio Code

## 1. Realizar el tutorial

- Depuración de TypeScript con Visual Studio Code 1 de 2.pdf
- Depuración de TypeScript con Visual Studio Code 2 de 2.pdf



# Archivo de configuración en un proyecto TypeScript

- **tsconfig.json** en directorio raíz
- Contiene configuraciones para el transpilador de typescript.
- target: indicamos que queremos que compile a código Javascript escrito con el estándar es5 (ECMAScript5).
- module: define el sistema de resolución del módulo de salida. “commonjs” Es un estándar para estructurar y organizar código JavaScript
- outDir: directorio de salida de los archivos transpilados. Archivos .js
- sourceMap:archivo que enlaza información con los archivos de código original, permitiendo reconstruir los mismos aún en producción

```
{  
  "compilerOptions": {  
    "target": "es5",  
    "module": "commonjs",  
    "outDir": "out",  
    "sourceMap": true  
  }  
}
```



# Tipos de datos en TypeScript

- boolean
- number
- string
- array
- null
- undefined
- object
- enum
- any
- void
- tupla



# Declaración de variables

**let** declara variables limitando su alcance (scope) al bloque, declaración, o expresión donde se está usando

```
let variable:tipo = valor;
```

```
let materia: string = "PAVII";
```

```
let materia: string;
```

```
materia = "PAVII";
```

```
let materia = "PAVII"; //Definición por inferencia de tipos
```

**var** igual que la declaración en JavaScript. Define una variable global o local en una función sin importar el ámbito del bloque. (opción no recomendada)

```
var materia = "PAVII";
```

**const** el valor no puede ser cambiado una vez establecido;

```
const pi = 3.1416;
```

```
const boton: HTMLButtonElement = document.getElementById("btnGrabar")! as HTMLButtonElement;
```



# Tipo de dato **number**

Se puede asignar cualquier valor numérico incluyendo decimales, hexadecimales, binarios y octales:

```
let integerValue: number = 10;  
let decimalValue: number = 123.87;  
let hexaDecimalValue: number = 0xf10b;
```



# Tipo de dato **string**

Se puede usar doble comilla (") o simple para definir una cadena de tipo string.

```
let nombre: string = "Juan"; // usando comilla doble  
let apellido: string = 'Pérez'; // usando comilla simple
```

Si se quiere expandir a múltiples líneas, o incluir expresiones \${ expresion } se puede usar (`).

```
let nombre: string = "Juan"; // usando comilla doble  
let apellido: string = 'Pérez'; // usando comilla simple  
let saludo: string = `Hola, mi nombre es: ${ nombre } ${ apellido }`;  
let saludoMultilinea : string = `Hola,  
Mi nombre es: ${ nombre } ${ apellido }`;
```



# Tipo de dato **boolean**

```
let esAdministrador: boolean = true;  
let tieneSaldoNegativo: boolean = false;
```

# Tipo de dato enum

Los tipos de datos enum (enumeraciones) son un conjunto de valores numéricos con nombres más amigables.

```
enum TipoDeTarjeta { Debito , Credito , Virtual }
let tarjeta: TipoDeTarjeta = TipoDeTarjeta.Debito ;
```

De manera predeterminada, los valores de enumeración comienzan desde 0, pero también se puede establecer manualmente el valor de sus miembros.

```
enum TipoDeTarjeta { Debito = 1, Credito , Virtual }
```

```
enum TipoDeTarjeta { Debito = 1, Credito = 3, Virtual = 5}
```



# Tipo de dato **null**

Cualquier tipo de dato de typescript puede tener el valor de nulo.

```
let valorNumerico: number = null;  
if (valorNumerico == null)  
    console.log("Contiene un valor nulo");
```

# Tipo de dato **void**

Se utiliza generalmente en funciones que no devuelven ningún valor.

```
function mostrarEnConsolaLog(mensaje: string): void  
{  
    console.log(mensaje);  
}
```



# Tipo de dato **undefined**

Como **undefined** es un subtipo de todos los demás tipos, al igual **null** que puede asignarlo a **number** o a un valor **boolean** por ejemplo. Es cuando una variable ha sido declarada pero no se le ha asignado ningún valor.

```
let valorNumerico: number = undefined;  
let valorNumerico2: number;  
if (valorNumerico === undefined) console.log("No está definido el valor");  
if (valorNumerico2 === undefined) console.log("No está definido el valor");
```

Nota: **undefined** y **null** son dos tipos distintos: **undefined** es un tipo en sí mismo (indefinido) mientras que **null** es un objeto

```
null === undefined // false  
null == undefined // true  
null === null // true
```



# Tipo de dato **array**

Se pueden definir los arrays de las siguientes maneras:

```
let marcasObtenidas: number[] = [80, 85, 75];
```

```
let marcasObtenidas: Array<number> = [80, 85, 75];
```

```
class ArticuloFamilia {
  IdArticuloFamilia: number;
  Nombre: string;
}

const ArticulosFamilias: ArticuloFamilia[] = [
  { IdArticuloFamilia: 1, Nombre: "Accesorios" },
  { IdArticuloFamilia: 2, Nombre: "Audio" },
  { IdArticuloFamilia: 10, Nombre: "Led - Lcd" }
];
```



# Tipo de dato **any**

Cuando no está seguro del tipo de datos de un valor, debido a su contenido dinámico, puede usar la palabra clave **any** para declarar dicha variable.

```
let valorDinamico: any = "Perez, Juan";
valorDinamico = 100;
valorDinamico = true;

let listaDinamica: any[] = ["Perez, Juan", "Usuario administrador", 21, true];

let otroValorDinamico;
otroValorDinamico = 10;
otroValorDinamico = "Juan"
```



# Aserciones de tipo

Sirve para indicar al compilador de TypeScript que el tipo de datos contenido dentro de una variable es del tipo indicado dentro de <>

```
let algunValor: any = "esto es una cadena";  
  
let longitud: number = (<string>algunValor).length;
```

También se puede expresar de la siguiente manera:

```
let algunValor: any = "esto es una cadena";  
  
let longitud: number = (algunValor as string).length;
```



# Operadores - Encadenamiento opcional (?)

Permite detener inmediatamente la ejecución de código si el valor de una expresión es **null** o **undefined**

```
let materia: string;  
materia = "PAVI";  
let x = materia?.replace("PAVI", "PAVII");
```

De esta manera cuando **materia** sea **null** o **undefined**, directamente retorne **undefined**, sin invocar al método **replace**. Por lo que es lo mismo que:

```
let x = (materia === null || materia === undefined) ? undefined : materia.replace("PAVI", "PAVII");
```

# Operadores - de fusión nulo (??)

El operador analiza la expresión, y si la misma es nula o undefined devuelve el valor indicado después de ??

```
let materia: string;
materia = "PAVII"
let x = materia ?? "Otra Materia"; //PAVII
materia = null;
x = materia ?? "Otra Materia"; //Otra Materia
```

# Definición de clases

El modificador de acceso por defecto para miembros de la clase (propiedades, métodos es **public**)

```
class Alumno {  
    nombre: string; // es lo mismo que definirlo de esta forma => public nombre: string;  
    legajo: number;  
}  
  
let alumno: Alumno = new Alumno();  
alumno.nombre = 'Agustín';  
alumno.legajo = 74552;  
  
let alumnoPAVII: Alumno = { nombre: "Juan", legajo: 45842 };  
  
let alumnos: Alumno[] = [  
    { nombre: "Juan", legajo: 45842 },  
    { nombre: "Maria", legajo: 65833 },  
    { nombre: "Pedro", legajo: 48320 },  
];
```

Los modificadores de acceso pueden ser **public** (default), **private** o **readonly**



PAV2 - UTN FRC

# Definición de clases - Constructor

```
class Alumno{  
    nombre: string; // es lo mismo que definirlo de esta forma => public nombre: string;  
    legajo: number;  
    constructor(nombre: string, legajo: number) {  
        this.nombre = nombre; // 'this' se usa para acceder a los atributos de la clase  
        this.legajo = legajo;  
    }  
    imprimir(){ // es lo mismo que definir como => public imprimir()  
        console.log(`El nombres es ${this.nombre} y el legajo ${this.legajo}`);  
    }  
}  
let alumno: Alumno = new Alumno('Juan Perez',45000);  
alumno.imprimir();
```



# Clases- Método abreviado de definición propiedades

Se debe especificar el modificador de acceso en los parámetros del constructor

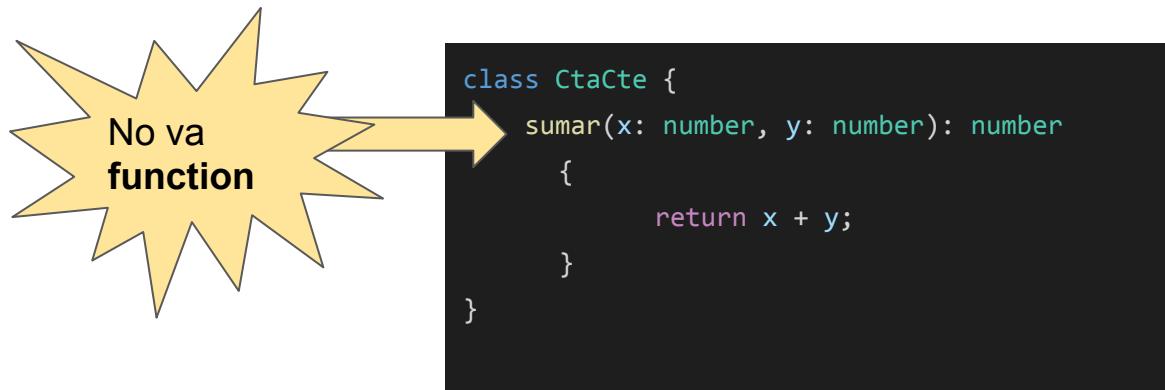
```
class Alumno{  
    constructor(public nombre: string, private legajo: number) { }  
    //las propiedades se pueden definir con modificador de acceso public o private  
    imprimir(){  
        console.log(`El nombres es ${this.nombre} y el legajo ${this.legajo}`);  
    }  
}  
  
let alumno: Alumno = new Alumno('Juan Perez',45000);  
alumno.imprimir();
```

El constructor tiene un bloque de llaves vacías {} ya que no tenemos que implementar ningún código en su interior, pero al anteceder el modificador de acceso en la zona de parámetros los mismos pasan a ser propiedades de la clase y no parámetros.



# Clases - definición de métodos

Cuando se definen funciones en las clases, una de sus peculiaridades es que **NO** aparece **function**, sino que se escriben directamente:



# Métodos y propiedades estáticos

Se usa la palabra reservada **static**, para indicar los métodos y propiedades que pertenecen a la clase y no a una instancia de la misma.

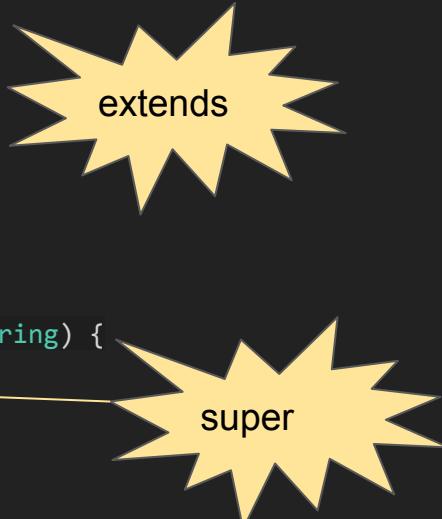
```
class Matematica {  
    static mayor(v1: number, v2: number): number {  
        if (v1 > v2) return v1;  
        else return v2;  
    }  
    static menor(v1: number, v2: number): number {  
        if (v1 < v2) return v1;  
        else return v2;  
    }  
    static aleatorio(inicio: number, fin: number): number {  
        return Math.floor((Math.random() * (fin + 1 - inicio)) + inicio);  
    }  
}  
console.log('El mayor es ' + Matematica.mayor(10, 9));
```



# Clases - Herencia

- TypeScript soporta herencia tanto de clases como de interfaces.

```
class Persona {  
    nombre: string;  
    constructor(nombre: string) {  
        this.nombre = nombre;    }  
    }  
  
class Empleado extends Persona {  
    legajo: number;  
    constructor(legajo: number, nombre: string) {  
        super(nombre);  
        this.legajo = legajo;  
    }  
    mostrarInfo(): void {  
        console.log("Nombre = " + this.nombre + ", Legajo = " + this.legajo);  
    }  
}
```



```
let emp = new Empleado(100, "Juan");  
emp.mostrarInfo(); // Nombre = Juan,  
Legajo = 100
```



# Interfaces

- Es una estructura que define el contrato en una aplicación.
- Las clases que la implementan deben seguir el contrato especificado en la interfaz.
- Una interfaz puede contener definiciones de propiedades y métodos

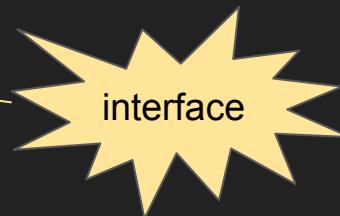
```
interface IEmppleado {  
    legajo: number;  
    nombre: string;  
    getSalarioMes(mes: number, anio: number): string;  
}
```



# Clases - Implementación de interfaces

- Se puede implementar en una clase más de una interface

```
interface IPersona {  
    nombre: string;  
    mostrarInfo(): void;  
}  
  
interface IEmpleado {  
    legajo: number;  
}  
  
class Empleado implements IPersona, IEmpleado {  
    legajo: number;  
    nombre: string;  
    mostrarInfo(): void {  
        console.log("Nombre = " + this.nombre + ", Legajo = " + this.legajo);  
    }  
}
```



```
let empleado: Empleado = new  
Empleado();  
empleado.nombre = "juan";  
empleado.legajo = 100;  
empleado.mostrarInfo();  
  
let persona: IPersona = empleado;  
console.log(persona.nombre);
```



# Módulos

- Permite encapsular código y determinar que se expone públicamente y que se mantiene resguardado
- En TypeScript cada archivo **.ts** constituye un módulo.
- Con la palabra reservada **export** se determina lo que es accesible fuera del módulo. Ej `export class ArticuloFamilia {...}` caso contrario solo puede ser accedido dentro del módulo.
- Para usar otros módulos se debe indicar con la sentencia **import**.  
Ej: `import { ArticulosFamilias, ArticuloFamilia } from "./articulo-familia";`
- Generalmente se exportan funciones, clases o interfaces



# Módulos - ejemplos (1 / 2)

articulo-familia.ts

```
export class ArticuloFamilia {  
    IdArticuloFamilia: number;  
    Nombre: string;  
}
```

array-articulos-familias.ts

```
import { ArticuloFamilia } from "./articulo-familia";  
export const ArticulosFamilias: ArticuloFamilia[] = [  
    { IdArticuloFamilia: 1, Nombre: "Accesorios" },  
    { IdArticuloFamilia: 2, Nombre: "Audio" },  
    { IdArticuloFamilia: 3, Nombre: "Celulares" },  
    { IdArticuloFamilia: 4, Nombre: "Cuidado Personal" },  
    { IdArticuloFamilia: 5, Nombre: "Dvd" },  
];
```



# Módulos - ejemplos (2 / 2)

articulos-familias.ts

```
export class ArticuloFamilia {  
    IdArticuloFamilia: number;  
    Nombre: string;  
}  
  
export const ArticulosFamilias: ArticuloFamilia[] = [  
    { IdArticuloFamilia: 1, Nombre: "Accesorios" },  
    { IdArticuloFamilia: 2, Nombre: "Audio" },  
    { IdArticuloFamilia: 3, Nombre: "Celulares" },  
    { IdArticuloFamilia: 4, Nombre: "Cuidado Personal" },  
];
```

negocio-articulos.ts

```
import { ArticulosFamilias, ArticuloFamilia } from  
"./articulo-familia";  
  
for (var i = 0; i < ArticulosFamilias.length; i++)  
    console.log(ArticulosFamilias[i].Nombre);  
  
let art: ArticuloFamilia = new ArticuloFamilia();  
art.Nombre = "Computadora";  
art.IdArticuloFamilia = 100;
```



# Funciones con nombre

Se utilizan en los módulos con la palabra reservada **function**, se puede indicar en cada parámetro el tipo de dato que puede recibir y el que retorna la función

```
function foo(msg: string, obj: any): void
{
    console.log(msg + obj);
}
```

O también:

```
function foo(msg: string, obj)
    //obj es de tipo any
    //la función al no tener declarado tipo de retorno, de manera predeterminada es void
{
    console.log(msg + obj);
}
```



# Funciones - Parámetros opcionales y por defecto

En Typescript cada parámetro que se declare se asume que es requerido por la función. En javascript si el parámetro no se provee, se asigna undefined

```
function mensajePersonal(nombre: string, apellido?: string, pais = "argentina") {  
    if (apellido) return nombre + " " + apellido + " de " + pais;  
    else return nombre + " de " + pais;  
}  
  
let result1 = mensajePersonal("Juan");  
let result2 = mensajePersonal("Juan", "Perez", "España", "Sr."); // error,  
let result3 = mensajePersonal("Juan", "Perez", "España"); // correcto!
```



# Funciones anónimas

Una función anónima no especifica un nombre. Son semejantes a JavaScript con la salvedad de la definición de tipos para los parámetros:

```
let comparaValores = function (x: number, y: number): string {
  if (x == y) return `${x} es igual que ${y}`;
  else if (x >= y) return `${x} mayor o igual que ${y}`;
  else return `${y} mayor que ${x}`;
};
console.log(comparaValores(50, 40));
```



# Funciones Flecha (Arrow)

Son funciones normalmente de una instrucción que se almacenan en una variable o se ejecutan directamente.

```
let suma = (x: number, y: number): number => {
    return x + y;
}
suma(10, 20); //devuelve 30
```

Si es una única instrucción, no hace falta las llaves ni la palabra reservada **return**;

```
let foo = (x: number, y: number) => x+y;
let res = foo(10,20); //devuelve 30
```



# Funciones Flecha (Arrow) en clases

```
class EmpleadoEmp {
    legajo: number;
    nombre: string;
    constructor(legajo: number, nombre: string) {
        this.nombre = nombre;
        this.legajo = legajo;
    }
    mostrar = () => console.log(this.legajo + ' ' + this.nombre)
}
let empl = new EmpleadoEmp(1, 'Maria');
empl.mostrar();
```



# Realizar la siguiente ejercitación de TypeScript

[https://docs.google.com/document/d/1Fj3knBLt6DQKUymbc-POYr\\_b81gJQKRHuBwmjkuBE5o/edit?usp=sharing](https://docs.google.com/document/d/1Fj3knBLt6DQKUymbc-POYr_b81gJQKRHuBwmjkuBE5o/edit?usp=sharing)

Para descargar la resolución del mismo [ingresar aquí](#)

# Ejemplo de Typescript: una lista de tareas

<https://uv.frc.utn.edu.ar/mod/forum/discuss.php?d=27846>

# Decoradores

- Son una característica muy potente de TypeScript
- Son una serie de metadatos adicionales que se pueden añadir a clases, métodos, propiedades para modificar su comportamiento
- En Angular lo podemos ver la siguiente forma:

```
@Component({
  selector: "app-articulos",
  templateUrl: "./articulos.component.html",
  styleUrls: ["./articulos.component.css"]
})
export class ArticulosComponent
{
  ...
}
```



# Decoradores - implementaciones en métodos

Los decoradores de métodos reciben 3 parámetros llamados:

1. **target**: la clase a la que pertenece el método decorado.
2. **key**: una cadena con el nombre del método decorado.
3. **descriptor**: las propiedades descriptoras del método decorado.

```
function logear(target: Object, key: string, descriptor: any) {  
    console.log('Clase: ', target.constructor.toString());  
    console.log('Método: ', key);  
    console.log('Características del método: ',  
descriptor);  
}
```

```
class Ejercicio {  
    respuesta!: string;  
    @logear  
    responder(respuesta:string) {  
        this.respuesta = respuesta;  
    }  
}  
let ejercicio = new Ejercicio();  
ejercicio.responder('jump!');
```



# Decoradores - Implementaciones en Clases

Funcionan de manera parecida a los métodos, pero reciben sólo un parámetro, que es la clase sobre la que actúa llamado **target**.

```
function decoradorClase(param1: string, param2: string) {
    let clase = function (target: any) {
    }
    // hacemos algo...
    return clase;
}

@decoradorClase('foo', 'bar')
class miClase {

}
```



# Referencias

Documentación oficial de typescript

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

Editor en línea

<https://www.typescriptlang.org/play/index.html>

Otra fuentes de información

<https://devdocs.io/typescript/>

<https://desarrolloweb.com/manuales/manual-typescript.html>



## Unidad 4

# Angular



Universidad Tecnológica Nacional  
Facultad Regional Córdoba  
Cátedra: Programación de Aplicaciones Visuales 2



# ¿Qué es Angular?

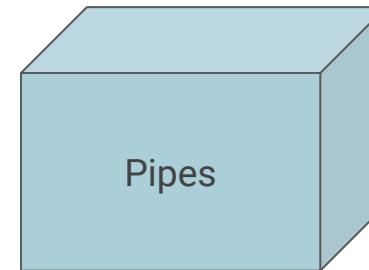
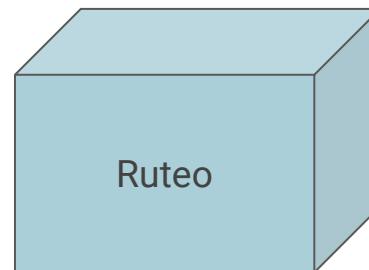
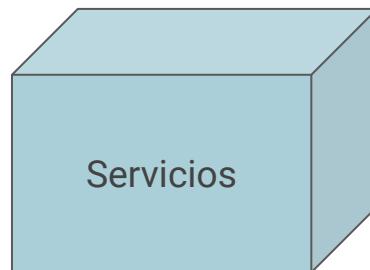
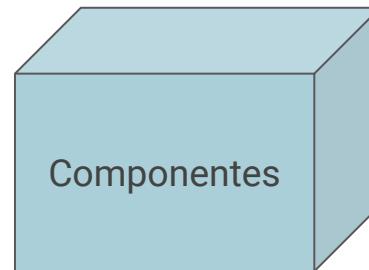
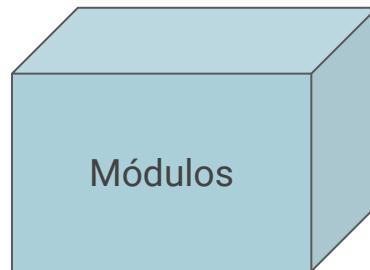


- Un framework JavaScript
- Para construir aplicaciones front end
- Usando HTML, CSS y JavaScript



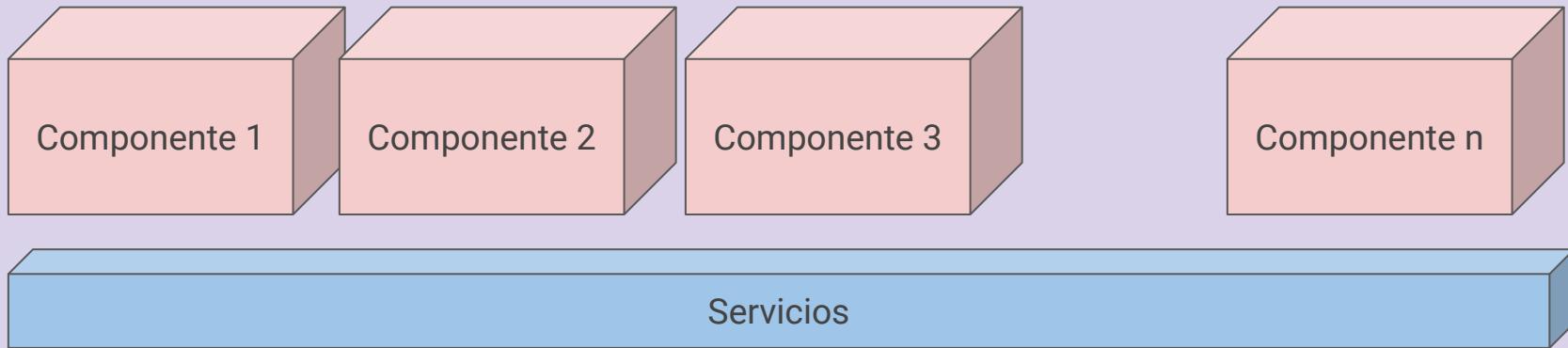
# Arquitectura de una aplicación

# Elementos de una aplicación Angular



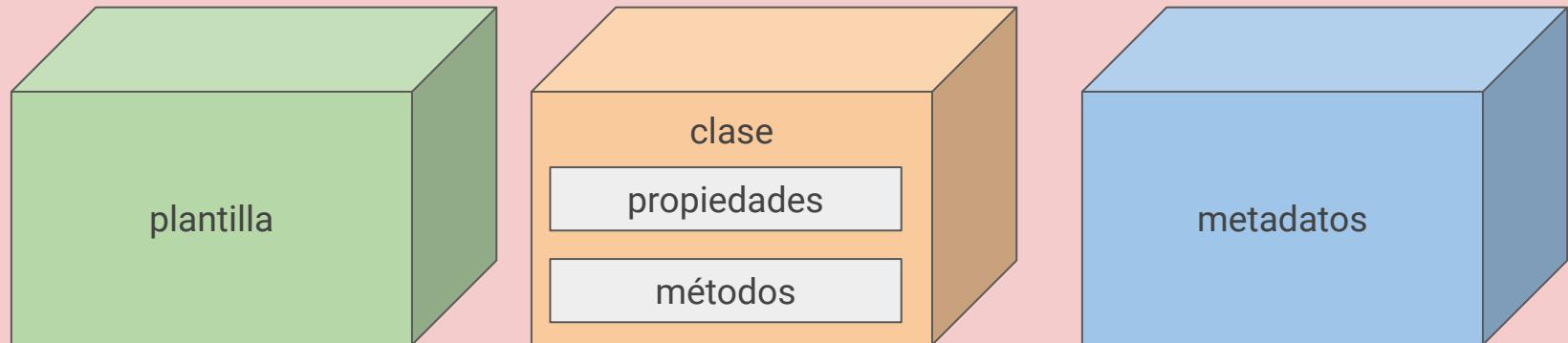
# Anatomía de una aplicación Angular

Aplicación Angular



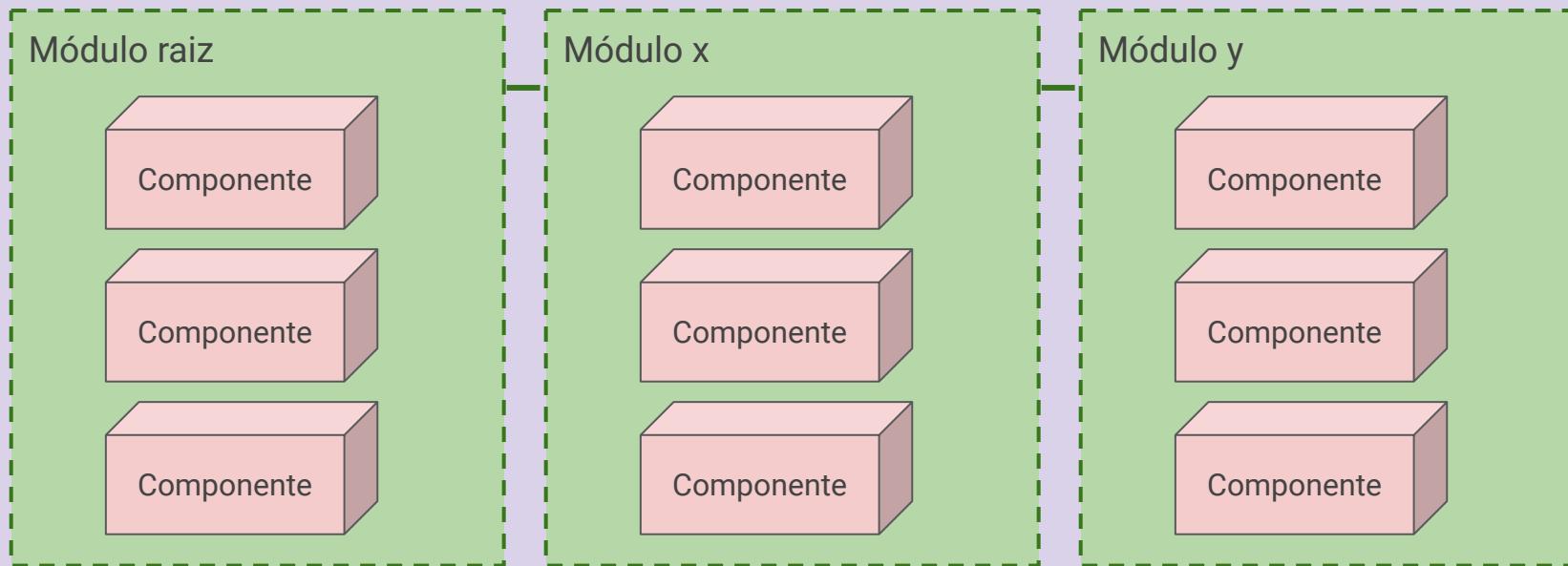
# Anatomía de un componente

Componente



# Módulos

## Aplicación Angular



# Entorno de desarrollo

# Preparación del entorno de desarrollo

## Paso 1

Instalar npm (node package manager): <https://nodejs.org/es/download/>

The screenshot shows the Node.js download page. At the top, there's a navigation bar with links: INICIO, ACERCA, DESCARGAS, DOCUMENTACIÓN, PARTICIPE, SEGURIDAD, NOTICIAS, and CERTIFICATION. Below the navigation bar, the word "node" is displayed with its logo. A dropdown menu is open under the "DESCARGAS" link, showing two tabs: "LTS" (Recommended for the majority) and "Actual" (Latest features). Under "LTS", there are links for "Instalador Windows (.msi)", "Binario Windows (.zip)", "Instalador macOS (.pkg)", "Binario macOS (.tar.gz)", "Binario Linux (x64)", "Binario Linux (ARM)", and "Código Fuente". Under "Actual", there are links for "Instalador Windows (.msi)" (32-bit and 64-bit), "Instalador macOS (.pkg)" (64-bit), "Binario macOS (.tar.gz)" (64-bit), "Binario Linux (x64)" (64-bit), "Binario Linux (ARM)" (ARMv7 and ARMv8), and "Código Fuente" (node-v14.15.4.tar.gz).

	32-bit	64-bit
Instalador Windows (.msi)		
Binario Windows (.zip)		
Instalador macOS (.pkg)	64-bit	
Binario macOS (.tar.gz)	64-bit	
Binario Linux (x64)	64-bit	
Binario Linux (ARM)	ARMv7	ARMv8
Código Fuente	node-v14.15.4.tar.gz	



# Preparación del entorno de desarrollo

## Paso 2

Instalar Angular CLI: <https://github.com/angular/angular-cli>

```
npm install -g @angular/cli
```



# Crear y correr una aplicación angular

```
ng new pymes
```

```
cd pymes  
ng serve --open
```

# Práctica

Crear una aplicación Angular con angular-cli  
Iniciar el servidor y verificar que funciona



# Práctica

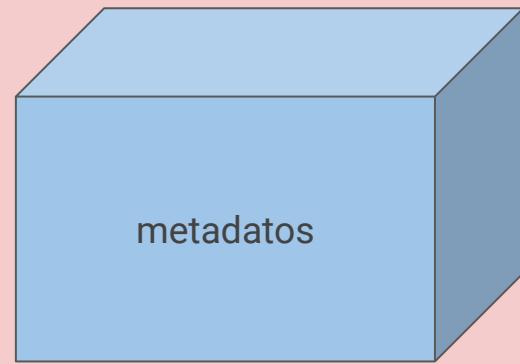
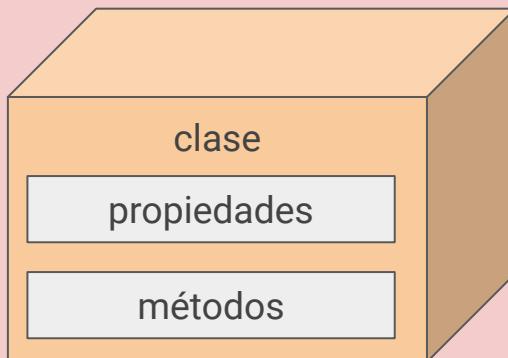
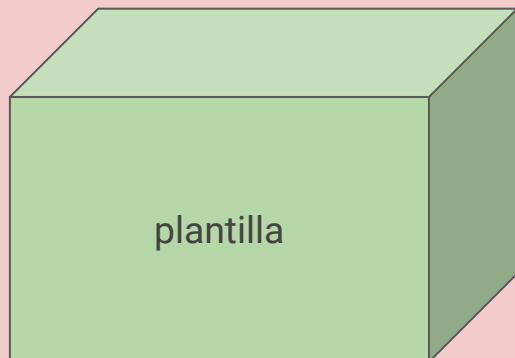
Crear una aplicación Angular con StackBlitz  
<https://stackblitz.com/edit/angular-dkprek>



# Componentes

# Anatomía de un componente

## Componente



- Apariencia visual
- Expresada en HTML
- Incluye directivas y enlace a datos

- Código asociado a la vista
- Creada con TypeScript
- Tiene propiedades: datos
- Tiene métodos. lógica

- Información adicional para Angular
- Se definen mediante un decorator



# Un componente simple

app.component.ts

```
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-root',  
  template: `<div>  
    <h1>{{titulo}}</h1>  
    <div>Mi componente</div>  
  </div>`  
})  
export class AppComponent {  
  titulo = 'Pymes';  
}
```

}

Import

}

Metadata y Plantilla

}

Clase



# Decorator

app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <div>
      <h1>{{titulo}}</h1>
      <div>Mi componente</div>
    </div>`
})
export class AppComponent {
  titulo = 'Pymes';
}
```

La metadata se define mediante un decorator

Éste precede inmediatamente al elemento al que modifica

El decorator se define mediante el carácter @

En este caso el decorator Component indica que la clase que lo sucede es un component

Le pasamos un objeto como parámetro que, en este caso, tiene dos propiedades.



# Selector

## index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Pymes</title>
  <base href="/">
  <meta name="viewport"
content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon"
href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

## app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template:
    <div>
      <h1>{{titulo}}</h1>
      <div>Mi componente</div>
    </div>
})
export class AppComponent {
  titulo = 'Pymes';
}
```



# Práctica

Eliminar el código generado de app.component.ts y crear un componente más simple como el analizado

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <div>
      <h1>{{titulo}}</h1>
      <div>Mi componente</div>
    </div>`
})
export class AppComponent {
  titulo = 'Pymes';
}
```



# Módulos

# Módulos

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
```

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Imports

Decorator

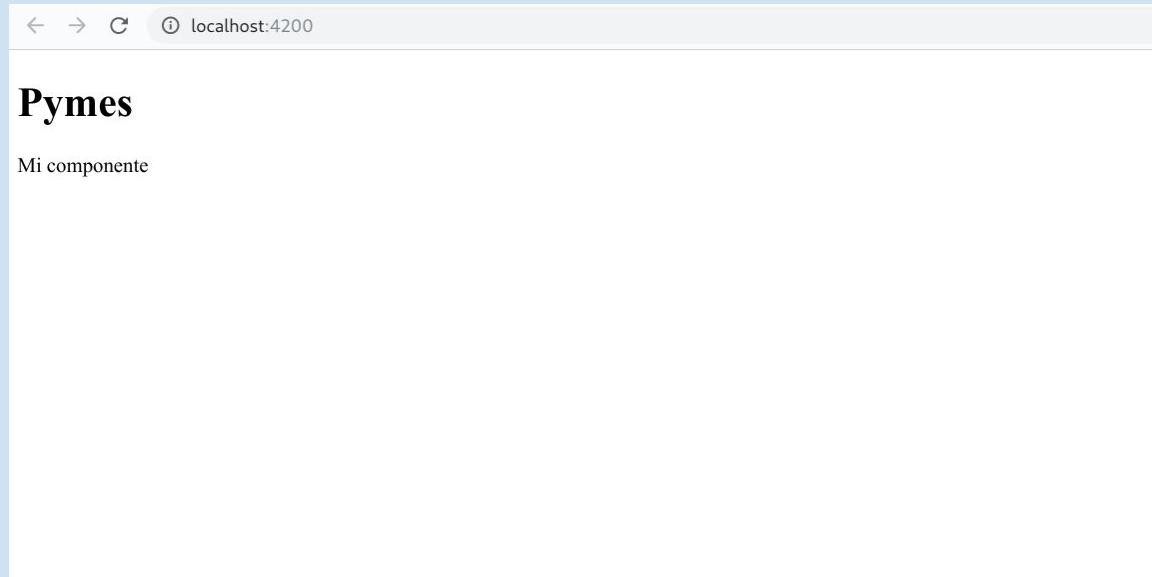
Clase



PAV2 - UTN FRC

# Práctica

Revisar el módulo app.module.ts y la página index.html y a continuación iniciar la aplicación



# Plantillas

# Plantilla inline

app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <div>
      <h1>{{titulo}}</h1>
      <div>Mi componente</div>
    </div>`})
export class AppComponent {
  titulo = 'Pymes';
}
```



# Plantilla enlazada

app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  titulo = 'Pymes';
}
```

app.component.html

```
<div>
  <h1>{{titulo}}</h1>
  <div>Mi componente</div>
</div>
```



# Incluir Bootstrap

# Instalar Bootstrap y Bootstrap Icons o Font Awesome

```
npm install bootstrap@next --save
```

Si vamos a usar Bootstrap Icons:

```
npm install bootstrap-icons --save
```

Alternativamente podemos usar Font Awesome:

```
npm install @fortawesome/fontawesome-free --save
```



# Incluir bootstrap y font awesome

angular.json

```
"styles": [  
    "node_modules/bootstrap/scss/bootstrap.min.css",  
    "node_modules/bootstrap-icons/font/bootstrap-icons.css",  
    "node_modules/@fortawesome/fontawesome-free/css/all.min.css",  
    "src/styles.scss"  
],  
"scripts": [  
    "node_modules/bootstrap/dist/js/bootstrap.min.js",  
    "node_modules/@fortawesome/fontawesome-free/js/all.min.js"  
]
```



# Crear un componente desde cero

# Crear un componente usando angular cli

```
ng generate component lista-articulos
```

```
ng generate component lista-articulos --skipTests
```

```
ng g c lista-articulos --skipTests
```



# Crear un componente manualmente

Crearemos la carpeta:

src/app/articulos/

Dentro de la carpeta los siguientes archivos:

src/app/articulos/lista-articulos.component.ts

src/app/articulos/lista-articulos.component.html



# Modificando el template

lista-articulos.component.html

```
<div>
  <h1>{{titulo}}</h1>
  <div class="table-responsive">
    <table class="table">
      <thead>
        <tr>
          <th>Imagen</th>
          <th>Descripción</th>
          <th>Código</th>
          <th>Cantidad</th>
          <th>Precio</th>
          <th>Puntaje</th>
        </tr>
      </thead>
      <tbody>
        </tbody>
    </table>
  </div>
</div>
```



PAV2 - UTN FRC

# Especificando la clase

lista-articulos.component.ts

```
import { Component } from "@angular/core";

@Component({
  selector: 'lista-articulos',
  templateUrl: 'lista-articulos.component.html'
})
export class ListaArticulosComponent{
  titulo: string = 'Lista de Árticulos';
}
```



# Usar el componente como una directiva

app.component.html

```
<div>
  <h2>{{titulo}}</h2>
  <lista-articulos></lista-articulos>
</div>
```



# Incluir el componente en el módulo

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { ListaArticulosComponent } from './articulos/lista-articulos.component';

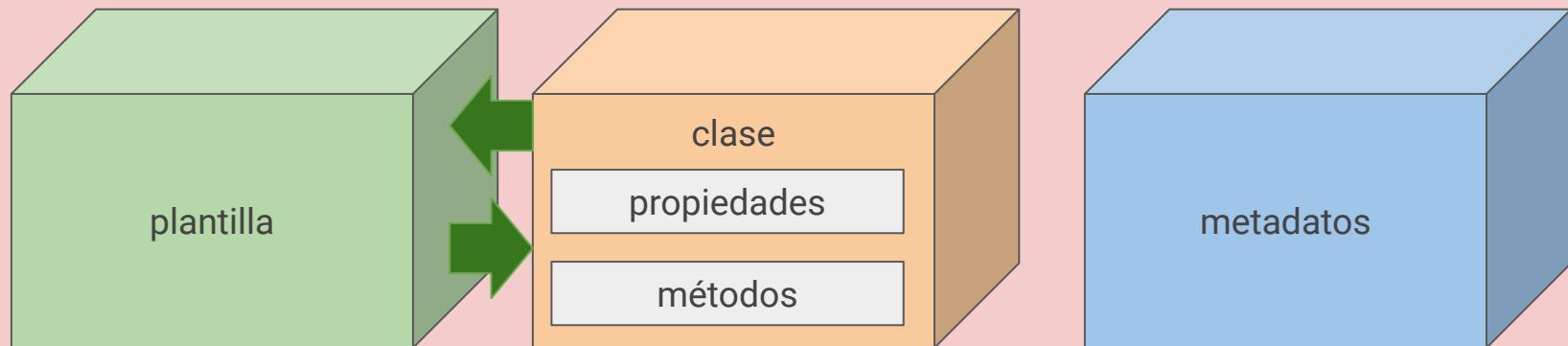
@NgModule({
  declarations: [
    AppComponent,
    ListaArticulosComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



# Enlace a datos (data binding)

# Enlace a datos (data binding)

Componente



# Técnicas de data binding: interpolación

app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  titulo = 'Pymes';
}
```

app.component.html

```
<div>
  <h1>{{titulo}}</h1>
  <div>Mi componente</div>
</div>
```



# Técnicas de data binding: interpolación

```
<h1>{{titulo}}</h1>
<p>{{3*7+12}}</p>
<p>{'Nombre: ' + obtenerNombre()}</p>
```



# Directivas

Las directivas en Angular son elementos que se utilizan para conformar y extender el HTML y pueden ser:

- Directivas del usuario (custom) por ejemplo `<lista-articulos>`
- Directivas provistas por Angular (built in)



# Tipos de Directivas

Desde otra perspectiva podemos caracterizar las directivas en:

- Componentes (también son directivas)
- Directivas estructurales (ej: \*ngIf)
- Directivas de atributo (ej: ngClass)



# Directivas estructurales

Las directivas estructurales modifican la estructura de la vista agregando, quitando o manipulando los elementos que la conforman.

**\*ngIf** implementa una estructura selectiva

**\*ngFor** implementa una estructura iterativa



# \*ngIf

## lista-articulos.component.html

```
<div>
  <h1>{{titulo}}</h1>
  <div class="table-responsive">
    <table class="table" *ngIf="articulos && articulos.length" >
      <thead>
        <tr>
          <th>Imagen</th>
          <th>Descripción</th>
          <th>Código</th>
          <th>Cantidad</th>
          <th>Precio</th>
          <th>Puntaje</th>
        </tr>
      </thead>
      <tbody>
        </tbody>
      </table>
    </div>
  </div>
```



# articulos

## lista-articulos.component.ts

```
...
export class ListaArticulosComponent{
  titulo: string = 'Lista de Árticulos';
  articulos: any[] = [
    {
      id: 2,
      descripcion: 'Articulo X',
      codigo: 'xsd-143',
      cantidad: 139,
      precio: 1221.40,
      puntaje: 4
    },
    {
      id: 5,
      descripcion: 'Articulo Y',
      codigo: 'dlg-912',
      cantidad: 336,
      precio: 400.99,
      puntaje: 3
    }
  ];
}
```



# \*ngFor

## lista-articulos.component.html

```
...
<tbody>
  <tr *ngFor="let articulo of articulos">
    <td></td>
    <td>{{articulo.descripcion}}</td>
    <td>{{articulo.codigo}}</td>
    <td>{{articulo.cantidad}}</td>
    <td>{{articulo.precio}}</td>
    <td>{{articulo.puntaje}}</td>
  </tr>
</tbody>
```



# Práctica

Crear el componente lista-articulos y mostrar los artículos en una tabla

# Enlace de propiedades (property binding)

# Enlazar propiedades

```
<img [src]='articulo.imagen'>
```

propiedad  
[]

expresión  
''



# Comparación con interpolación

```
<img [src]='articulo.imagen'>
```

```
<img src={{articulo.imagen}}>
```



# Práctica

Mostrar la imagen de cada artículo junto al artículo usando enlace a propiedades.

Poner la carpeta de imágenes dentro de la carpeta /src/assets

Agregar la propiedad imagen a los objetos articulos

```
imagen: 'assets/imagenes/articulox.jpg',
```



# Práctica

Seguramente en el paso anterior el tamaño de las imágenes hace que la tabla se vea poco amigable.

Vamos a usar enlace a propiedades para definir un tamaño adecuado para las imágenes, su margen y un título más apropiado

## lista-articulos.component.html

```
<tr *ngFor="let articulo of articulos">
  <td><img [src]='articulo.imagen'
    [title]='articulo.descripcion'
    [style.height.px]='alturaImagen'
    [style.margin.px]='margenImagen'></td>
  <td>{{articulo.descripcion}}</td>
  <td>{{articulo.cantidad}}</td>
  <td>{{articulo.precio}}</td>
  <td>{{articulo.puntaje}}</td>
</tr>
```

## lista-articulos.component.ts

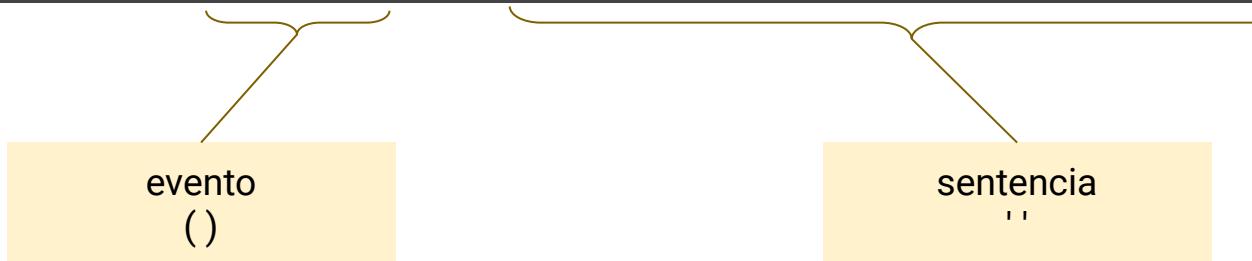
```
...
export class ListaArticulosComponent{
  titulo: string = 'Lista de Artículos';
  alturaImagen: number = 40;
  margenImagen: number = 2;
  ...
}
```



# Enlace de eventos (event binding)

# Enlazar eventos

```
<button (click)='mostrarEsconderImagen()'>
```



# Práctica

Agregar un botón que cuando se presione: muestre o esconda alternativamente las imágenes de los artículos. Para eso haremos lo siguiente:

1. Agregaremos un botón en el encabezado de la columna de las imágenes
2. Creamos una variable mostrarImagen de tipo boolean que va a definir el estado de si las imágenes son visibles o no
3. Creamos el método mostrarEsconderImagen()
4. Enlazamos el método creado con el evento click del botón creado en el paso 1



# Práctica

## lista-articulos.component.ts

```
...
export class ListaArticulosComponent{
    titulo: string = 'Lista de Artículos';
    alturaImagen: number = 40;
    margenImagen: number = 2;
    mostrarImagen: boolean = false;

    ...
    mostrarEsconderImagen(): void{
        this.mostrarImagen = !this.mostrarImagen;
    }
}
```



# Práctica

## lista-articulos.component.html

```
<tr>
  <th>
    <button class="btn btn-primary"
      (click)='mostrarEsconderImagen()'>
      {{mostrarImagen ? 'Esconder' : 'Mostrar'}} Imagen
    </button>
  </th>
  <th>Descripción</th>
  <th>Código</th>
  <th>Cantidad</th>
```

```
<tbody>
  <tr *ngFor="let articulo of articulos">
    <td><img *ngIf='mostrarImagen'
      [src]='articulo.imagen'
      [title]='articulo.descripcion'
      [style.height.px]='alturaImagen'
      [style.margin.px]='margenImagen'
```



# Pipes

# Pipes

Los pipes nos permiten aplicar transformaciones a los datos enlazados antes de ser mostrados

- **incluidos**

- **date**
- **number, decimal, percent, currency**
- **json, slice**

- **custom**

- **implementados por el programador**



# Pipes: ejemplos

```
<h1>{{titulo | uppercase}}</h1>
<h1>{{libro.titulo | titlecase}}</h1>
<td>{{articulo.precio | currency | lowercase}}</td>
<td>{{articulo.precio | currency:'ARS':true:'1.2-2'}}</td>
<td>{{libros | slice:1:6}}</td>
<td>{{libro.avance | percent}}</td>
<td>{{articulo.fechaVencimiento | date:'longDate'}}</td>
```



# Práctica

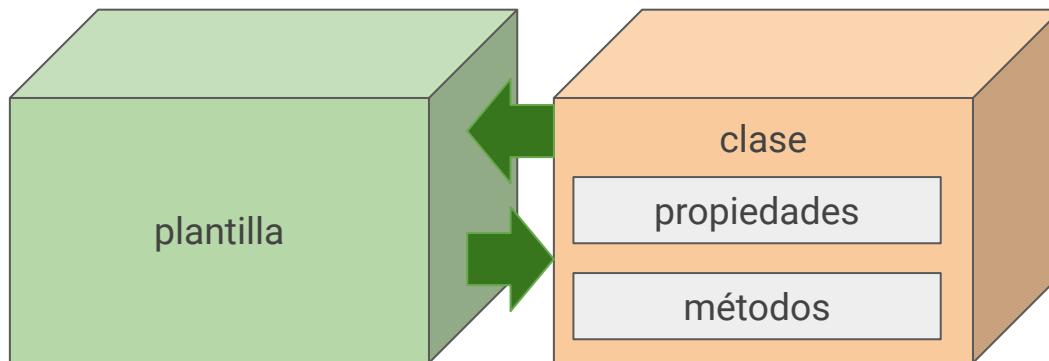
## lista-articulos.component.html

```
<tbody>
    <tr *ngFor="let articulo of articulos">
        <td><img *ngIf='mostrarImagen'
            [src]='articulo.imagen'
            [title]='articulo.descripcion | titlecase'
            [style.height.px]='alturaImagen'
            [style.margin.px]='margenImagen'
        ></td>
        <td>{{articulo.descripcion}}</td>
        <td>{{articulo.codigo}}</td>
        <td>{{articulo.cantidad}}</td>
        <td>{{articulo.precio | currency:'ARS':'symbol-narrow':'1.2-2'}}</td>
        <td>{{articulo.puntaje}}</td>
    </tr>
</tbody>
```



**Enlace a datos en los dos sentidos  
(two-way binding)**

# Enlace a datos en los dos sentidos (two-way binding)

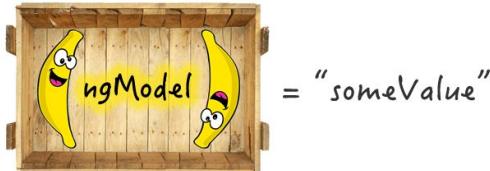


# Two-way binding

```
<input [(ngModel)]='expresionFiltro'>
```

`[( ngModel )] = "someValue"`

bananas en una caja [0]



# ngModel

## app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { ListaArticulosComponent } from './articulos/lista-articulos.component';

@NgModule({
  declarations: [
    AppComponent,
    ListaArticulosComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  ...
})
```



# Práctica

Implementar un filtro para la lista de artículos.

En la primera parte vamos a modificar la interfaz para que el usuario pueda ingresar la expresión del filtro

lista-articulos.component.ts

```
export class ListaArticulosComponent{
    titulo: string = 'Lista de Artículos';
    alturaImagen: number = 40;
    margenImagen: number = 2;
    mostrarImagen: boolean = false;
    expresionFiltro: string = '';

    ...
}
```

# Práctica

## lista-articulos.component.html

```
<div class="card">
  <div class="card-header">
    {{titulo}}
  </div>
  <div class="card-body">
    <div class="row">
      <div class="col-md-2">
        Filtrar por:
      </div>
      <div class="col-md-4">
        <input type='text'
          [(ngModel)] = 'expresionFiltro' />
      </div>
    </div>
    <div class="row">
      <div class="col-md-6"><h4>Filtrado por: {{expresionFiltro}}</h4></div>
    </div>
    <div class="table-responsive">
```



# Custom pipes

# Un custom pipe

convertir-a-espacio.pipe.ts

```
import{Pipe, PipeTransform} from '@angular/core';  
  
@Pipe({  
  name: 'convertirAEspacio'  
})  
  
export class ConvertirAEspacioPipe implements PipeTransform{  
  transform(value: string, character: string): string{  
    return value.replace(character, ' ');  
  }  
}
```

The diagram illustrates the structure of the code. Braces on the right side group the code into three categories:

- Import:** Groups the first line of code: `import{Pipe, PipeTransform} from '@angular/core';`
- Decorator:** Groups the annotation: `@Pipe({ name: 'convertirAEspacio' })`
- Clase:** Groups the class definition: `export class ConvertirAEspacioPipe implements PipeTransform{ ... }`



# Práctica

## lista-articulos.component.html

```
<td>{{articulo.descripcion}}</td>
<td>{{articulo.codigo | uppercase | convertirAEspacio:'-'}}</td>
<td>{{articulo.cantidad}}</td>
<td>{{articulo.precio | currency:'ARS':'symbol-narrow':'1.2-2'}}</td>
<td>{{articulo.puntaje}}</td>
```



# Práctica

## app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { ListaArticulosComponent } from './articulos/lista-articulos.component';
import { ConvertirAEspacioPipe } from './shared/convertir-a-espacio.pipe';

@NgModule({
  declarations: [
    AppComponent,
    ListaArticulosComponent,
    ConvertirAEspacioPipe
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: []
})
```



# Interfaces

# Interface

Es una **especificación** mediante la cual se definen propiedades y métodos

Una clase se compromete a cumplir con esa **especificación** al implementar una interface

La interface es y se usa como un tipo de datos

Solo se utiliza en tiempo de diseño, ya que no existe en ES5, solo en TypeScript



# Interface

En nuestro ejemplo, queremos remplazar el tipo any de la línea que se muestra abajo, por un tipo de datos que nos de más posibilidades, ya que éste no nos permite, entre otras cosas, usar las características de seguridad de tipos que tiene TypeScript

```
lista-articulos.component.ts
```

```
...
articulos: any[] = [
...
]
```



```
lista-articulos.component.ts
```

```
...
articulos: IArticulo[] = [
...
]
```



# Interface

articulo.ts

```
export interface IArticulo{  
    id: number,  
    descripcion: string,  
    codigo: string,  
    imagen: string,  
    cantidad: number,  
    precio: number,  
    puntaje: number  
}
```



PAV2 - UTN FRC

# Componentes anidados

# Componentes anidados

Google Chrome  
Pymes localhost:4200

## Pymes

Lista de Artículos

Filtrar por:

Filtrado por:

Esconder Imagen	Descripción	Código	Cantidad	Precio	Puntaje
	Articulo X	DLG 138	139	\$1,221.40	4.8
	Articulo Y	XWE 388	336	\$400.99	3.5
	Articulo Z	AHR 936	84	\$630.00	5
	Articulo W	LOC 122	256	\$450.00	3.3

Diagrama que ilustra la jerarquía de componentes en un front-end. Una caja grande (que representa el contenedor principal) contiene una caja roja (que representa un componente específico). Dentro de la caja roja, se muestra una lista de datos tabulados. Una flecha apunta de la caja grande hacia la caja roja, indicando la relación de anidamiento entre los componentes.



# el componente puntaje



## puntaje.component.ts

```
import { Component, OnChanges } from '@angular/core';

@Component({
  selector: 'puntaje',
  templateUrl: './puntaje.component.html',
  styleUrls: ['./puntaje.component.css']
})

export class PuntajeComponent implements OnChanges{
  puntaje: number = 4;
  puntajeAncho: number;

  ngOnChanges(): void{
    this.puntajeAncho = this.puntaje * 68 / 5;
  }
}
```



# el template del componente puntaje



## puntaje.component.html

```
<div class="parcial"
[style.width.px]='puntajeAncho'
[title]='puntaje' >
<div style='width:68px'>
    <span class="fas fa-star fa-xs"></span>
    <span class="fas fa-star fa-xs"></span>
    <span class="fas fa-star fa-xs"></span>
    <span class="fas fa-star fa-xs"></span>
    <span class="fas fa-star fa-xs"></span>
</div>
</div>
```

## puntaje.component.css

```
.parcial {
    overflow: hidden;
    color:#FFDD33;
}
```



# Incluir el nuevo componente

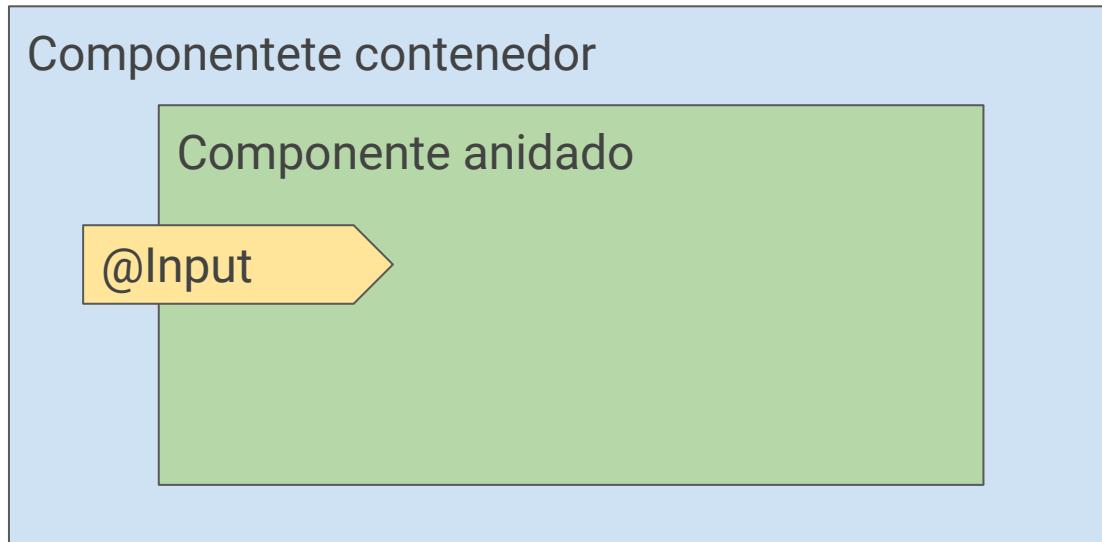
app.module.ts

```
...
import { PuntajeComponent} from './shared/puntaje.component'
...
@NgModule({
  declarations: [
    AppComponent,
    ListaArticulosComponent,
    ConvertirAEspacioPipe,
    PuntajeComponent
  ],
...
}
```



# Pasando data a un componente

# Pasando datos a un componente: @Input



# Pasando datos a un componente: @Input

## puntaje.component.ts

```
import { Component, OnChanges, Input } from '@angular/core';
...
export class PuntajeComponent implements OnChanges{
    @Input() puntaje: number = 4;
    puntajeAncho: number;
...
}
```

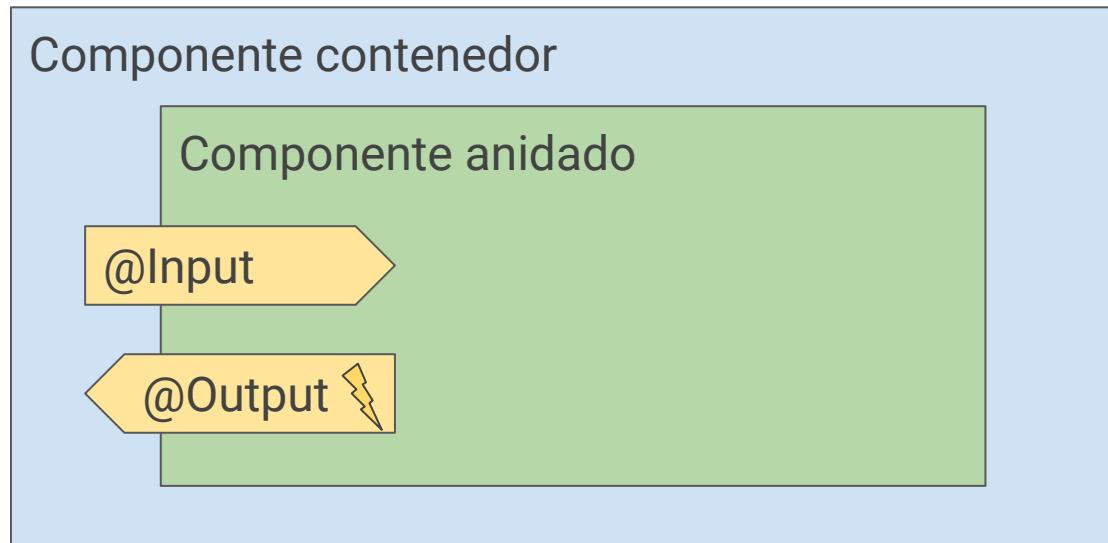
## lista-articulos.component.ts

```
<td><puntaje [puntaje]='articulo.puntaje'></puntaje></td>
```



# Pasando data desde un componente

# Pasando datos desde un componente: @Output



# Pasando datos desde un componente: @Output

En la **primera etapa** vamos a crear un evento en el componente puntaje que se va a desencadenar desde el evento onClick.

Por ahora tendrá una implementación temporal: vamos a hacer que muestre el puntaje en la consola cuando se hace click.

Esto se hace mediante la técnica de enlace de eventos que ya conocemos.



# Pasando datos desde un componente: @Output

puntaje.component.ts

```
...
export class PuntajeComponent implements OnChanges{
    ...
    onClick(): void{
        console.log("El puntaje es:" + this.puntaje );
    }
}
...
```

puntaje.component.html

```
<div class="parcial"
    [style.width.px]='puntajeAncho'
    [title]='puntaje'
    (click)='onClick()'
    >
<div style='width:75px'>
    <span class="fa fa-star"></span>
    <span class="fa fa-star"></span>
```



PAV2 - UTN FRC

# Pasando datos desde un componente: @Output

Ahora, en esta **segunda etapa** vamos a hacer que el componente anidado emita un evento que permita ser capturado en el componente contenedor.

Para eso declararemos el evento con el decoorator `@Output` y remplazaremos la implementación del método `onClick()` desencadenando este evento.

En el componente contenedor, capturaremos el evento mediante enlace a eventos



# Pasando datos desde un componente: @Output

puntaje.component.ts

```
import { Component, OnChanges, Input, Output, EventEmitter } from '@angular/core';
@Component({
  selector: 'puntaje',
  templateUrl: './puntaje.component.html',
  styleUrls: ['./puntaje.component.css']
})
export class PuntajeComponent implements OnChanges{
  @Input() puntaje: number;
  puntajeAncho: number;
  @Output() puntajeClicked: EventEmitter<string> = new EventEmitter<string>();

  ngOnChanges(): void{
    this.puntajeAncho = this.puntaje * 75 / 5;
  }
  onClick(): void{
    this.puntajeClicked.emit('El puntaje es: ' + this.puntaje);
  }
}
```



# Pasando datos desde un componente: @Output

lista-articulos.component.html

```
...
<td><puntaje [puntaje]='articulo.puntaje'
    (puntajeClicked)='onPuntajeClicked($event)'>
</puntaje>
</td>
...
```

lista-articulos.component.ts

```
...
onPuntajeClicked(mensaje:string): void{
    this.titulo = 'Lista de Artículos - ' + mensaje;
}
...
```



# Servicios

# Servicios

Son clases que tienen un propósito específico

Implementan funcionalidad que:

- Es independiente de los componentes (transversal)
- Permite compartir lógica o datos entre componentes
- Encapsula la interacción externa



# Inyección de dependencias (dependency injection)

Cuando en una clase se necesita usar una instancia de otra clase se puede hacer por dos vías alternativas:

1 Servicio

```
export class miServicio{}
```

Componente

```
let svc = new miServicio();
```

svc

2

Servicio

```
export class miServicio{}
```

Injector

Componente

```
constructor(private miServicio){}
```

svc



# Inyección de dependencia (dependency injection)

Es un **patrón de código** en el cual una clase recibe las **instancias** de objetos que necesita (dependencias) de una fuente externa en vez de crearlas por sí misma

En angular esta fuente externa es el **angular injector**



# Crear un servicio

articulos.service.ts

```
import { Injectable } from '@angular/core';
```



Import

```
@Injectable()
```



Decorator

```
export class ArticulosService{
```



Clase

```
}
```



PAV2 - UTN FRC

# Crear un servicio

articulos.service.ts

```
import { Injectable } from '@angular/core';
import { IArticulo } from './articulo';

@Injectable()
export class ArticulosService{
  getArticulos(): IArticulo[]{
    return [
      {
        id: 2,
        descripcion: 'Articulo X',
        ...
        puntaje: 4.8
      },
      ...
    ];
  }
}
```



# Registrar un servicio: en el root injector

articulos.service.ts

```
import { Injectable } from '@angular/core';
import { IArticulo } from './articulo';

@Injectable({
  providedIn: 'root'
})

export class ArticulosService{
  getArticulos(): IArticulo[]{
    return [
      ...
    ];
  }
}
```



# Registrar un servicio: en un componente

lista-articulos.component.ts

```
import { Component } from '@angular/core';
import { IArticulo } from './articulo';
import { ArticulosService } from './articulosService';

@Component({
  selector: 'lista-articulos',
  templateUrl: 'lista-articulos.component.html',
  providers: [ArticulosService]
})
export class ListaArticulosComponent{
  ...
}
```



# Injectar un servicio

lista-articulos.component.ts

```
import { Component, OnInit } from '@angular/core';
import { IArticulo } from './articulo';
import { ArticulosService } from './articulos.service';

@Component({
  selector: 'lista-articulos',
  templateUrl: 'lista-articulos.component.html'
})
export class ListaArticulosComponent implements OnInit{

  constructor(private articulosService: ArticulosService){}
  ...
  articulos: IArticulo[] = [];
  ...
  ngOnInit(): void{
    this.articulos = this.articulosService.getArticulos();
  }
}
```

# Programación reactiva y observables

# Programación Reactiva

## Programación **tradicional**:

Las instrucciones se ejecutan una detrás de otra

## Programación **reactiva**:

La programación reactiva es la programación con **flujos de datos asíncronos**. Casi cualquier cosa puede constituir un flujo de datos (variables, entradas del usuario, propiedades, estructuras de datos). Estos flujos de datos pueden ser "escuchados" y **reaccionar** a estos de forma determinada.



# Observables

El patrón **observer** es un modo de implementación de la programación reactiva

Los elementos principales de este patrón son:

**Observable**: Es aquello que queremos observar, que será implementado mediante una colección de eventos o valores futuros. Un observable puede ser creado a partir de eventos de usuario derivados del uso de un formulario, una llamada HTTP, un almacén de datos, etc.

**Observer**: Es el actor que se dedica a observar. Básicamente se implementa mediante una colección de funciones callback que nos permiten escuchar los eventos o valores emitidos por un observable.

**Subject**: es el emisor de eventos, que es capaz de crear el flujo de eventos cuando el observable sufre cambios. Esos eventos serán los que se consuman en los observers.



# Observables

Los observables nos ayudan a manejar los datos asíncronos

Tratan los eventos como una colección: Un array cuyos ítems llegan asíncronamente mientras transcurre el tiempo

Un método puede suscribirse a un observable para recibir notificaciones cuando los datos van llegando, también es notificado cuando no hay más datos o cuando ocurre un error.

El método puede reaccionar cuando los datos llegan u ocurre algún otro evento



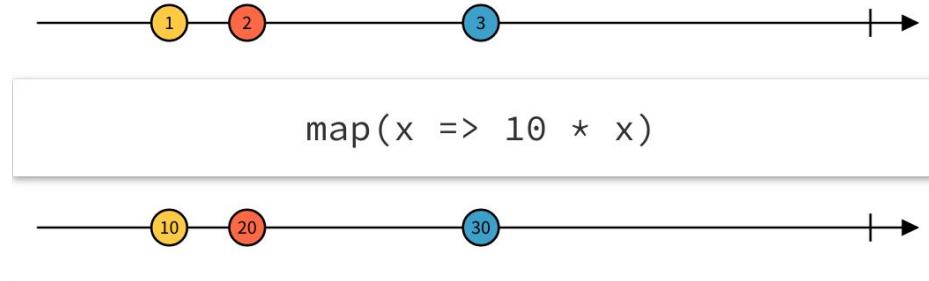
# Operadores observables

Los observables nos permiten manipular conjuntos de eventos con operadores

Los operadores son métodos de los observables que constituyen nuevos observables

Cada operador transforma de alguna manera la fuente observable

Ejemplos: map, filter, merge, take, etc



# Operadores observables: composición

Los observables se pueden componer usando el método pipe

ejemplo.ts

```
import { Observable, range } from "rxjs";
import { map, filters } from 'rxjs/operators';

const source$: Observable<number> = range(0,10);
source$.pipe(
  map(x => x * 3),
  filter( x => x % 2 === 0)
).subscribe(x => console.log(x));
```



# RxJS

Reactive Extensions (Rx) es una librería creada por Microsoft para implementar la programación reactiva

Provee funcionalidad para producir, consumir y manipular streams (flujos de datos)

Es la librería que adopta Angular para implementar la programación reactiva



# Obtener datos usando HTTP

# Enviando una petición HTTP

articulos.service.ts

```
import { Injectable } from '@angular/core';
import { IArticulo } from './articulo';
import { HttpClient } from '@angular/common/http'
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root',
})
export class ArticulosService{
  private articulosUrl = 'http://www.misarticulos.com.ar/api/articulos';
  constructor(private http: HttpClient){}
  getArticulos(): Observable<IArticulo[]>{
    return this.http.get<IArticulo[]>(this.articulosUrl);
  }
}
```



# Suscribiéndose al observable

lista-articulos.component.ts

```
...
export class ListaArticulosComponent implements OnInit{

    ...
    ngOnInit(): void{
        this.articulosService.getArticulos().subscribe({
            next: articulos => this.articulos = articulos,
            error: err => this.mensajeError = err
        });
    }
}
```

# Importar el módulo http

app.module.ts

```
...
import { HttpClientModule } from '@angular/common/http';
...
@NgModule({
  declarations: [
    AppComponent,
    ListaArticulosComponent,
    ConvertirAEspacioPipe,
    PuntajeComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```



# Navegación y routing

# Crear un componente usando angular cli

```
ng generate component articulos/detalle-articulo --skipTests --flat
```

```
ng g c articulos/detalle-articulo --skipTests --flat
```



# Crear otras páginas

```
ng g c components/inicio --skipTests --flat
```

```
ng g c components/pagina-no-encontrada --skipTests --flat
```



# Componente: detalle-articulo

## app.module.ts

```
...
import { DetalleArticuloComponent } from './articulos/detalle-articulo.component';

@NgModule({
  declarations: [
    ...
    DetalleArticuloComponent
  ],
  ...
})
```

## detalle-articulo.component.html

```
<div class="card" *ngIf="articulo">
  <div class="card-header">
    {{titulo + ': ' + articulo.descripcion}}
  </div>
</div>
```



# Componente: detalle-articulo

detalle-articulo.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'detalle-articulo',
  templateUrl: './detalle-articulo.component.html',
  styleUrls: ['./detalle-articulo.component.css']
})
export class DetalleArticuloComponent implements OnInit {

  constructor() { }

  titulo: string = 'Artículo';
  articulo: any;

  ngOnInit() {
  }
}
```



# Funcionamiento del routing

- Configurar una ruta para cada componente
- Definir acciones
- Asociar una ruta a cada acción
- Activar la ruta en función de la acción del usuario
- A partir de la activación mostrar la vista del componente correspondiente



# Configurar las rutas

index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Pymes</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```



# Configurar las rutas

app.module.ts

```
...
import { RouterModule } from '@angular/router';
...
@NgModule({
  ...
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    RouterModule.forRoot([
      {path: 'articulos', component: ListaArticulosComponent },
      {path: 'articulo/:id', component: DetalleArticuloComponent },
      {path: 'inicio', component: InicioComponent },
      {path: '', redirectTo: 'inicio', pathMatch: 'full' },
      {path: '**', component: PaginaNoEncontradaComponent }
    ])
  ],
...
})
```



# Crear acciones

app.component.html

```
<nav class="navbar navbar-expand navbar-light bg-light">
  <a class="navbar-brand">{{titulo}}</a>
  <ul class="nav nav-pills">
    <li><a class="nav-link">Inicio</a></li>
    <li><a class="nav-link">Lista Artículos</a></li>
  </ul>
</nav>
```



PAV2 - UTN FRC

# Asociar acciones a rutas

app.component.html

```
<nav class="navbar navbar-expand navbar-light bg-light">
  <a class="navbar-brand">{{titulo}}</a>
  <ul class="nav nav-pills">
    <li><a [routerLink]=["/inicio"] class="nav-link">Inicio</a></li>
    <li><a [routerLink]=["/articulos"] class="nav-link">Lista Artículos</a></li>
  </ul>
</nav>
```



# Asociar acciones a rutas

## app.component.html

```
<nav class="navbar navbar-expand navbar-light bg-light">
  <a class="navbar-brand">{{titulo}}</a>
  <ul class="nav nav-pills">
    <li><a [routerLink]=["/inicio"] class="nav-link">Inicio</a></li>
    <li><a [routerLink]=["/articulos"] class="nav-link">Lista Artículos</a></li>
  </ul>
</nav>
<div class="container">
  <router-outlet></router-outlet>
</div>
```



# Pasar parámetros a rutas

app.module.ts

```
...
    {path: 'articulo/:id', component: DetalleArticuloComponent },
...

```

lista-articulos.component.html

```
...
<td>
  <a [routerLink]="['/articulo', articulo.id]">
    {{articulo.descripcion}}
  </a>
</td>
...

```



# Recibir parámetros en la ruta

detalle-articulo.component.ts

```
import {ActivatedRoute } from '@angular/router'  
...  
@Component({  
  ...  
})  
export class DetalleArticuloComponent implements OnInit {  
  
  constructor(private route: ActivatedRoute) { }  
  
  titulo: string = 'Artículo';  
  articulo: IArticulo;  
  
  ngOnInit() {  
    let idArticulo = +this.route.snapshot.paramMap.get("id");  
    this.articulo = {  
      id: idArticulo,  
      descripcion: "test",  
      ...  
    };  
  };
```



# Formularios reactivos

# Variantes de formularios en Angular

## Basados en plantillas (template-driven)

La lógica del formulario se implementa principalmente en el template

## Reactivos (reactive o model-driven)

La lógica del formulario se implementa en la clase del componente



# Basados en plantillas vs Reactivos

## Basados en plantillas

Fáciles de usar

Similares a AngularJS

Two way binding + poco código

Seguimiento automático del estado

## Reactivos

Más flexibles (escenarios complejos)

Modelo inmutable

Más sencilla de implementar reacciones a los cambios en los datos

Fácil de agregar elementos dinámicamente

Facilidad para unit testing



# Formularios Validación - estados 1/2

Angular adjunta estas propiedades al formulario representando su estado

- **pristine** Ningún campo hay sido modificado aún.
- **dirty** Uno o más campos han sido modificado
- **invalid** El contenido del formulario es inválido
- **valid** El contenido del formulario es válido
- **submitted** El formulario ha sido enviado

Accesibles como: <FormGroup>.<property>



# Formularios Validación - estados 2/2

Angular adjunta estas propiedades a los **elementos** del formulario representando su estado

- **untouched** El campo no ha sido tocado aún
- **touched** El campo ha sido tocado
- **pristine** El campo no ha sido modificado aún
- **dirty** El campo ha sido modificado
- **invalid** El contenido del campo es inválido
- **valid** El contenido del campo es válido

Accesibles como: <FormGroup>.<FormControlName>.<property>

**TO BE  
CONTINUED...»**

## Unidad 5

# El lenguaje C#



Universidad Tecnológica Nacional  
Facultad Regional Córdoba  
Cátedra: Programación de Aplicaciones Visuales 2

# Resultado del aprendizaje

Conocer las generalidades del lenguaje C# para el desarrollo de aplicaciones .NET. focalizando su uso en servicios web.



# La clase

- Una aplicación C# es una colección de clases, estructuras y tipos.
- Una clase contiene una colección de datos y métodos para manipular los mismos.
- Sintaxis.

```
class nombre  
{  
    ...  
}
```

- Una aplicación C# puede abarcar más de un archivo. Una clase también, para ello se utiliza “partial class”.

# El método Main()

- En C# todas las aplicaciones para consola y winforms deben tener un punto de inicio, que es el método Main().
- Se debe declarar como:

```
class nombre
{
    static void Main()
    {
        ...
    }
}
```

- Puede devolver void o int.

# Namespaces

- Son el conjunto de clases, funciones y tipos de datos que nuestra aplicación puede utilizar.
- .NET Framework ofrece muchas clases útiles provistas a través de la Base Class Library.
- Para indicar que se utilizará un determinado namespace se utiliza la palabra reservada “using”.
- Podemos hacer uso de los namespaces provistos en la plataforma (BCL), creados por nosotros, desarrollados por terceras partes.

# Namespace de la aplicación

- Es una forma lógica de agrupar las clases, funciones y tipos de datos de nuestra aplicación.
- Se declaran con la palabra reservada “namespace”.
- Dentro de un namespace no es posible declarar dos clases con el mismo nombre.
- Puede agrupar uno o archivos de código de nuestra aplicación.
- Se pueden declarar en forma anidada.

# Namespace System y la clase Console

- Los namespace están organizados de forma jerárquica. System es la raíz del namespace más importante provisto por la plataforma.
- Dentro de él podemos encontrar la clase Console.
- Console brinda toda la funcionalidad para crear aplicaciones cuya interfaz es una consola del SO.
- Algunos métodos: WriteLine, ReadLine, Clear, Beep.

# “Hola Mundo” vía consola

```
using System;

namespace HolaMundo
{
    class Holamundo
    {
        static void Main()
        {
            System.Console.WriteLine("¡Hola Mundo!");
        }
    }
}

C:\application folder\csc program.cs
```

# Separación y agrupación de instrucciones. El “;” y las “{}”.

- El punto y coma “;” se utiliza para separar instrucciones.

```
objeto.propiedad = "Valor";  
  
objeto.propiedad =  
"Valor";
```

- Las llaves se utilizan para agrupar instrucciones dentro de un mismo bloque.

```
class nombre{  
    static void Main()  
    {  
        Instrucción1;  
        Instrucción2;  
    }  
}
```

# Mayúsculas y minúsculas (Case Sensitivity)

- Todas las palabras reservadas (using, namespace, public, class, if, for, etc.) se escriben en minúsculas.
- El compilador de C# puede distinguir entre dos variables declaradas con igual nombre pero con al menos una letra que difiera en mayúscula - minúscula.

# Comentarios

- Comentarios en una sola línea.

```
// Obtener el nombre del usuario  
Console.WriteLine("¿Cómo se llama? ");
```

- Comentarios que abarcan mas de una linea.

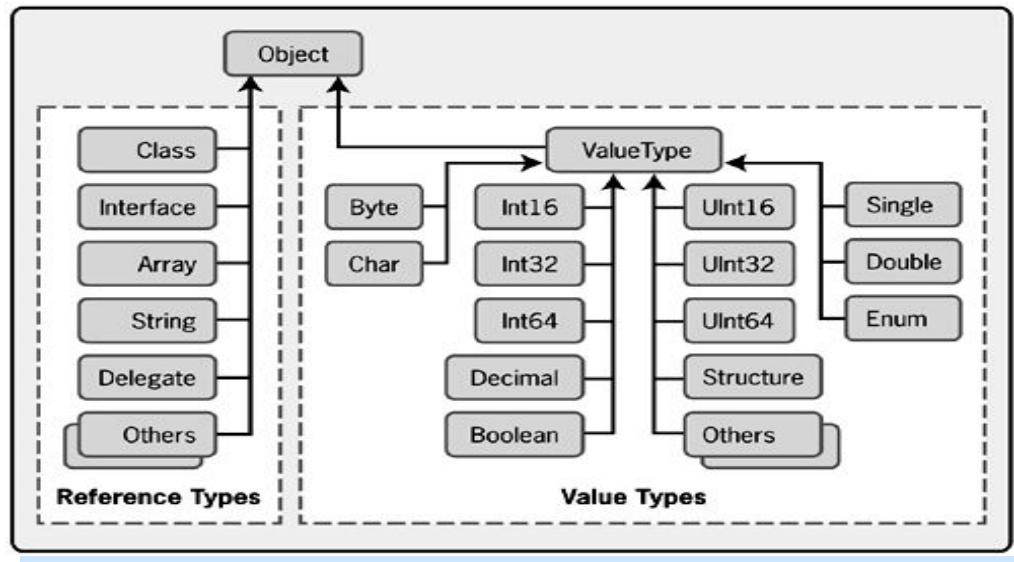
```
/* Encontrar la mayor raíz  
de la ecuación cuadrática */  
x = (...);
```

- Comentarios para generadores de documentación.

```
/// <summary>  
/// Devuelve la suma dos números enteros  
/// </summary>  
/// <param name="a">Número a</param>  
/// <param name="b">Número B</param>  
public int sumar(int a, int b)
```

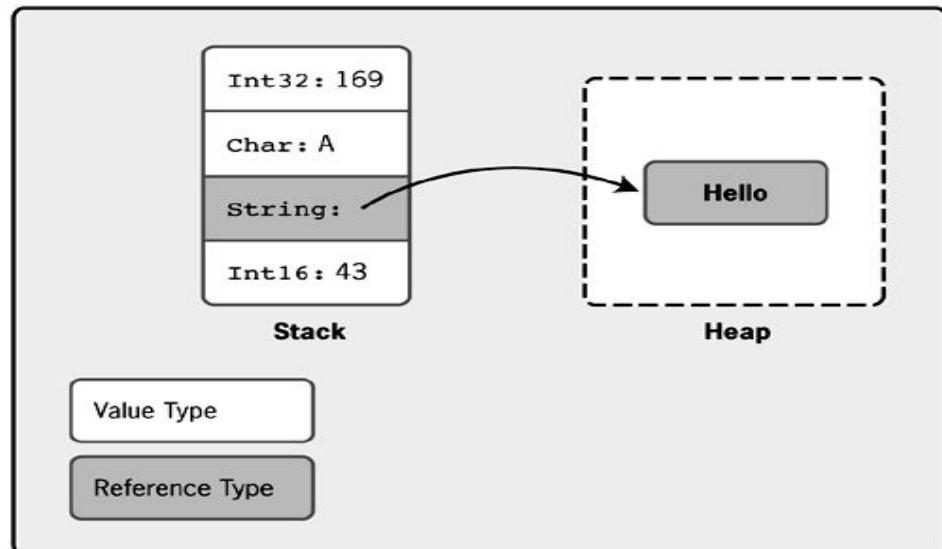
# El sistema de tipos comunes (CTS)

- Define un conjunto común de “tipos” de datos orientados a objetos.
- Todo tipo hereda directa o indirectamente del tipo System.Object.
- Define tipos de VALOR y de REFERENCIA.



# La memoria y los tipos de datos

- El CLR administra dos segmentos de memoria: **Stack (Pila)** y **Heap (Montón)**.
- El **Stack** es liberado automáticamente y el **Heap** es administrado por el **GC (Garbage Collector)**.
- Los tipos **VALOR** se almacenan en el Stack.
- Los tipos **REFERENCIA** se almacenan en el Heap.

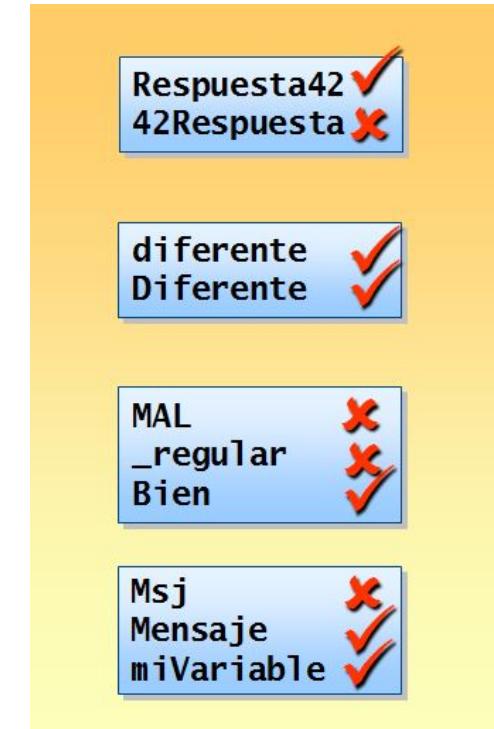


# Tipos de datos

Tipo	Descripción	Bytes	Rango de valores	Alias
SByte	Bytes con signo	1	[-128   127]	sbyte
Byte	Bytes sin signo	1	[0   255]	byte
Int16	Enteros cortos con signo	2	[-32.768   32.767]	short
UInt16	Enteros cortos sin signo	2	[0   65.535]	ushort
Int32	Enteros normales con signo	4	[-2.147.483.648   2.147.483.647]	int
UInt32	Enteros normales sin signo	4	[0   4.294.967.295]	uint
Int64	Enteros largos con signo	8	[-9.223.372.036.854.775.808   9.223.372.036.854.775.807]	long
UInt64	Enteros largos sin signo	8	[0   18.446.744.073.709.551.615]	ulong
Single	Reales con 7 dígitos de precisión	4	[1,5 * 10 -45   3,4 * 10 38]	float
Double	Reales con 15-16 dígitos de precisión	8	[5,0 * 10 -324   1,7 * 10 308]	double
Decimal	Reales con 28-29 dígitos de precisión	16	[1,0 * 10 -28   7,9 * 10 28]	decimal
Boolean	Valores lógicos	1	[true   false]	bool
Char	Valores Unicode	2	[0   65.535]	char
String	Cadenas de caracteres	Variable	Limitado por la memoria	string
Object	Cualquier objeto	Variable	Limitado por la memoria	object

# Reglas y recomendaciones para nombrar variables

- Reglas
  - Use letras, el signo de subrayado y dígitos.
- Recomendaciones
  - Evite poner todas las letras en mayúsculas.
  - Evite empezar con un signo de subrayado.
  - Evite el uso de abreviaturas.
  - Use PascalCasing para nombres con varias palabras.



# Palabras clave de C#

- Las palabras clave son identificadores reservados.

```
abstract, base, bool, default, if, finally
```

- No utilice palabras clave como nombres de variables.
  - Produce errores en tiempo de compilación
- Procure no usar palabras clave cambiando mayúsculas y minúsculas.

```
int INT; // Mal estilo
```

# Declaración de variables locales

- Se declaran indicando el tipo de dato y nombre de variable.

```
int objetoCuenta;
```

- Es posible declarar múltiples variables en una declaración.

```
int objetoCuenta, empleadoNúmero;
```

--0--

```
int objetoCuenta,  
    empleadoNúmero;
```

# Asignación de valores a variables

- Asignar valores a variables ya declaradas.

```
int empleadoNumero;  
empleadoNumero = 23;
```

- Inicializar una variable cuando se declara.

```
int empleadoNumero = 23;
```

- También es posible inicializar valores de caracteres.

```
char inicialNombre = 'J';
```

# Asignación compuesta

- Es muy habitual sumar un valor a una variable.

```
itemCount = itemCount + 40;
```

- Se puede usar una expresión más práctica.

```
itemCount += 40;
```

- Esta abreviatura es válida para todos los operadores aritméticos.

```
itemCount -= 24;
```

# Operadores comunes

## Operadores comunes Ejemplos

Operadores de igualdad	<code>== !=</code>
Operadores relacionales	<code>&lt; &gt; &lt;= &gt;= is</code>
Operadores condicionales	<code>&amp;&amp;    ?:</code>
Operador de incremento	<code>++</code>
Operador de decremento	<code>--</code>
Operadores aritméticos	<code>+ - * / %</code>
Operadores de asignación	<code>= *= /= %= += -= &lt;&lt;= &gt;&gt;= &amp;= ^=  =</code>

[https://msdn.microsoft.com/es-ar/library/s0b0c0z1\(v=vs.90\).aspx](https://msdn.microsoft.com/es-ar/library/s0b0c0z1(v=vs.90).aspx)

[https://msdn.microsoft.com/es-ar/library/6a71f45d\(v=vs.90\).aspx](https://msdn.microsoft.com/es-ar/library/6a71f45d(v=vs.90).aspx)

# Asignación implícita de tipos

A las variables locales se les puede asignar un "tipo" deducido en lugar de un tipo explícito.

La palabra clave **var** indica al compilador que deduzca el tipo de la variable a partir de la expresión que se encuentra en el lado derecho

```
// se compila como int  
var i = 5;
```

```
// se compila como string  
var s = "Hola";
```

```
// se compila como int[]  
var a = new[] { 0, 1, 2 };
```

# Asignación implícita de tipos

En muchos casos, el uso de **var** es opcional y es simplemente una comodidad sintáctica.

Cuando una variable se inicializa con un tipo anónimo, debe declarar la variable como **var** obligatoriamente

```
// tipo anónimo  
var v = new { Nombre = "Juan", Apellido = "Perez", Saldo =  
20.5 };
```

**var** solo se puede utilizar cuando una variable local se declara y se inicializa en la misma instrucción  
la variable no se puede inicializar como null, como un grupo de métodos ni como una función anónima.

# Incremento y decremento

- Es muy habitual cambiar un valor en una unidad.

```
objetoCuenta += 3; //Es igual a objetoCuenta = objetoCuenta + 3;  
objetoCuenta -= 2; //Es igual a objetoCuenta = objetoCuenta - 2;
```

- Se puede usar una expresión más práctica.

```
objetoCuenta++;  
objetoCuenta--;
```

- Existen dos formas de esta abreviatura.

```
++objetoCuenta;  
--objetoCuenta;
```

# Enumeraciones

- Definición de una enumeración.

```
enum Color { Rojo, Verde, Azul }
```

- Uso de una enumeración.

```
Color colorPaleta = Color.Rojo;
```

- Visualización de una variable de enumeración.

```
Console.WriteLine("{0}", colorPaleta); // Muestra Rojo
```

- Se puede obtener la posición del elemento

```
int x = (int)ColorPaleta; // Devuelve el nro 0
```

# Ejercitación

- Ejecutar paso a paso las sentencias del método Demo de la clase “Csharp\_1” del proyecto “1\_Csharp”.
- Ejecutar y analizar los ejercicios varios definidos en la clase “Program” del proyecto “Csharp\_1”.
- Resolver los ejercicios solicitados en el método “Ejercicio” de la clase “CSharp\_1”.

# Conversión implícita de tipos de datos

- Conversión de int a long

```
using System;
class Test
{
    static void Main()
    {
        int intValue = 123;
        long longValue = intValue;
        Console.WriteLine("(long) {0} = {1}", intValue, longValue);
    }
}
```

- Las conversiones implícitas no pueden fallar
  - Se puede perder precisión, pero no magnitud.

# Conversión explícita de tipos de datos

- Para hacer conversiones explícitas se usa una expresión de cast (molde):

```
using System;
class Test
{
    static void Main( )
    {
        long longValor = Int64.MaxValue;
        int intValue = (int) longValor;
        Console.WriteLine("(int) {0} = {1}", longValor, intValue);
    }
}
```

# Conversión explícita de tipos de datos

- Para convertir texto a otro tipo de dato, se utilizan los métodos Parse y Tryparse

```
int IntValor = Int32.Parse("2020");  
  
int IntValor;  
String strValor = "3030";  
bool result = Int32.TryParse(strValor, intValor);
```

- Para convertir cualquier tipo de dato a Texto, se utiliza el método ToString de las distintas clases base.

# Boxing y Unboxing

- Boxing: se utiliza cuando se convierte tipo valor a tipo referencia.

```
int a= 20;  
object b = a;
```

- Unboxing: cuando se convierte tipo referencia a tipo valor

```
int a = 20;  
object b = a;  
int c = (int)b;
```

# Conversión de Datos

- Las clases de datos tienen métodos para convertir objetos a su tipo.
  - Clase.Parse(String) Convierte un string a la clase destino. Si no tiene el formato correcto da error.
  - Clase.TryParse(String, out Objeto Clase): La función convierte el string en un objeto del tipo de la clase y lo devuelve como parametro Out. Si convierte, la función devuelve true y si no lo puede convertir, devuelve false.

# Conversión de Datos

- Las clases de datos tienen métodos para convertir objetos a su tipo.
  - Convert.ToDateTime: Convert tiene un listado de “To” (ToString,ToInt16, ToBoolean, etc) que permite convertir un string u otro tipo de dato a un tipo específico.
  - Objeto = (TipoObjeto) Objeto: Castear un objeto para que sea de otro tipo

# Nullables types (C# 2.0)

- Permite que un tipo de dato valor tenga un valor “null”
- Se define de la siguiente manera:  
`int? x = 125;`  
(notación frecuentemente utilizada) o también como:  
`Nullable<int> i;`
- Para determinar si posee un valor:  
`if (x.HasValue) {...}`  
O  
`if (x != null) {...}`
- Se puede usar el operador ?? para asignar un valor por default que va a ser aplicado cuando el valor es “null”  
`int? x = null; int y = x ?? -1;`

# Funcionalidad de Tipos de Datos

- Caracteres (Strings)
  - ToUpper - Mayúscula
  - ToLower - Minúscula
  - Trim - Remueve espacios en blanco
  - Substring - Cadena parcial de una cadena
  - IndexOf - Busqueda en la cadena
  - Length - Longitud
  - CompareTo - Compara por mayor o menor

# Funcionalidad de Tipos de Datos

- Fechas (Datetime)
  - Datetime.Now: Fecha y Hora del sistema
  - Datetime.Today: Fecha del sistema
  - Datetime.Parse: Convierte un texto a fecha. Si no tiene el formato, da error.
  - Objeto.ToString(Formato): Convierte a texto
  - Objeto.Add(TimeSpan): Suma un TimeSpan a la fecha del objeto.
  - Objeto.AddDays(dias): Existen Add de todas las unidades de tiempo.
  - Objeto.Substract: Resta una Fecha o un TimeSpan a otra fecha.

# Funcionalidad de Tipos de Datos

- Fracción de Tiempo (TimeSpan)
  - Almacena cantidad de años, meses, días, horas, minutos, segundos y fracciones.
  - Se pueden sumar a fechas y es lo que se obtiene al restar 2 fechas.
  - Objeto.Days/TotalDays: Devuelve los días sin y con fracción respectivamente.

# Estructuras de Control

- Bloques de instrucciones

Se usan llaves para delimitar bloques.

```
{  
    // code  
}
```

Un bloque y su bloque padre no pueden tener una variable con el mismo nombre.

```
{  
    int i;  
    ...  
    {  
        int i;  
        ...  
    }  
}
```

Bloques hermanos pueden tener variables con el mismo nombre.

```
{  
    int i;  
    ...  
}  
...  
{  
    int i;  
    ...  
}
```

# Estructuras de Control

- Tipos de instrucciones
  - Instrucciones condicionales: if y switch.
  - Instrucciones de iteración: while, do, for y foreach.
  - Instrucciones de salto: break y continue.

# La instrucción IF

- **Sintaxis**

```
if (expresión-booleana)
    primera-instrucción-incrustada;
else
{
    segunda-instrucción-incrustada;
    tercera-instrucción-incrustada;
}
```

- El bloque `else` es **opcional**.
- Si existen dos o más de una instrucciones en un bloque, estas deben estar agrupadas en {}.
- No hay conversión **implicita** de int a bool

```
int x;
...
if (x) ... // Debe ser if (x != 0) en C#
if (x = 0) ... // Debe ser if (x == 0) en C#
```

# Instrucciones if en cascada

```
enum Palo {Treboles,Corazones, Diamantes,  
Picas}  
  
Palo cartas = Palo.Corazones;  
string color = "";  
if (cartas == Palo.Treboles)  
{   color = "Negro";  
}  
else  
{  
    if (cartas == Palo.Corazones)  
    {   color = "Rojo";  
    }  
    else  
    {  
        if (cartas == Palo.Diamantes)  
        {   color = "Rojo";  
        }  
        else  
        {   color = "Negro";  
        }  
    }  
}
```

```
enum Palo {Treboles,  
          Corazones, Diamantes, Picas}  
Palo cartas = Palo.Corazones;  
string color = "";  
if (cartas == Palo.Treboles)  
{  
    color = "Negro";  
}  
else if (cartas == Palo.Corazones)  
{  
    color = "Rojo";  
}  
else if (cartas == Palo.Diamantes)  
{  
    color = "Rojo";  
}  
else  
{  
    color = "Negro";  
}
```

# La instrucción switch

- Las instrucciones switch se usan en bloques de varios casos.
- Se usan instrucciones break para evitar caídas en cascada (fall through).

```
switch (palo)
{
    case Palo.Treboles :
        color = "Negro";
        break;
    case Palo.Picas :
        color = "Negro";
        break;
    case Palo.Corazones :
        color = "Rojo";
        break;
    case Palo.Diamantes :
        color = "Rojo";
        break;
    default:
        color = "ERROR";
        break;
}
```

```
switch (palo)
{
    case Palo.Treboles :
    case Palo.Picas :
        color = "Negro";
        break;
    case Palo.Corazones :
    case Palo.Diamantes :
        color = "Rojo";
        break;
    default:
        color = "ERROR";
        break;
}
```

# La instrucción while

- Ejecuta instrucciones en función de un valor booleano.
- Evalúa la expresión booleana al principio del bucle.
- Ejecuta las instrucciones mientras el valor booleano sea true.

```
int i = 0;

while (i < 10)
{
    Console.WriteLine(i);
    i++;
}
```

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

# La instrucción do

- Ejecuta instrucciones en función de un valor booleano.
- Evalúa la expresión booleana al final del bucle.
- Ejecuta las instrucciones mientras el valor booleano sea true.

```
int i = 0;

do
{
    Console.WriteLine(i);
    i++;
} while (i < 10);
```

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

# La instrucción for

- La información de actualización está al principio del bucle.

```
for (int i = 0; i < 10; i++)  
{  
    Console.WriteLine(i);  
}
```

- Las variables de un bloque for sólo son válidas en el bloque.

```
for (int i = 0; i < 10; i++)  
    Console.WriteLine(i);  
Console.WriteLine(i); // Error: i está fuera de ámbito
```

- Un bucle for puede iterar varios valores.

```
for (int i = 0, j = 0; ... ; i++, j++)
```

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

# La instrucción foreach

- Elige el tipo y el nombre de la variable de iteración.
- Ejecuta instrucciones incrustadas para cada elemento de la clase collection.

```
ArrayList numeros = new ArrayList();

for (int i = 0; i < 10; i++ )
{
    numeros.Add(i);
}

foreach (int numero in numeros)
{
    Console.WriteLine(numero);
}
```

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

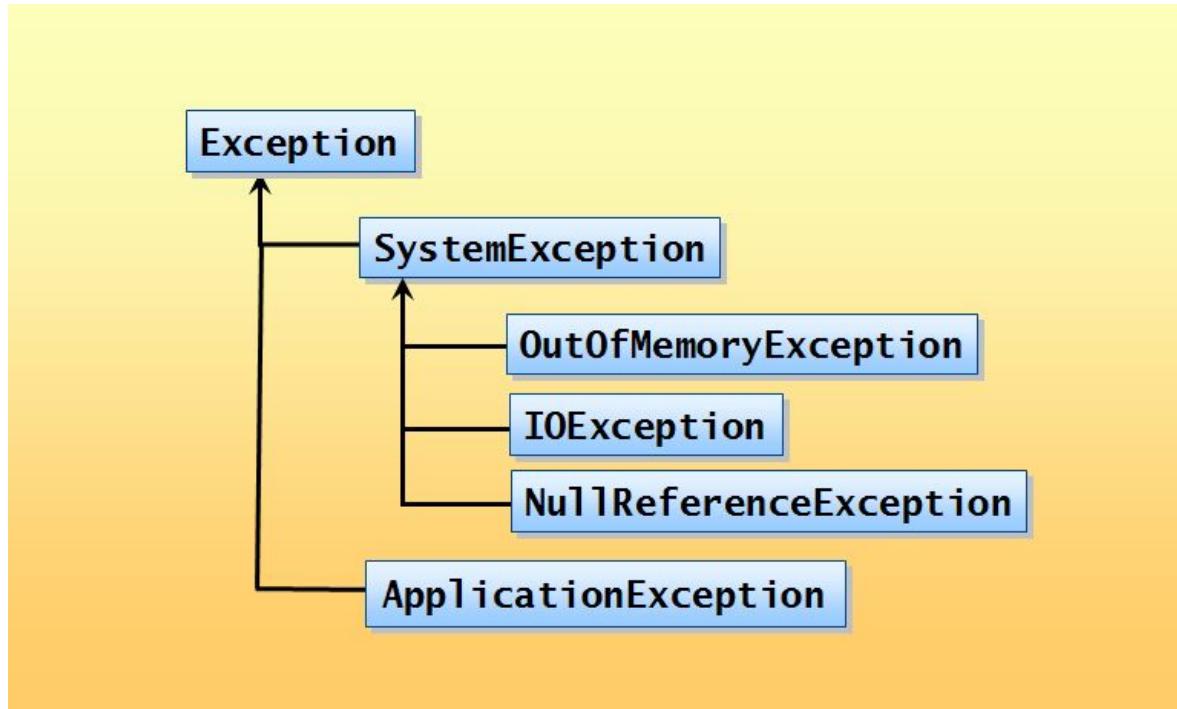
# Instrucciones break and continue

- La instrucción break abandona la instrucción switch, while, do, for o foreach más próxima.
- La instrucción continue salta a la siguiente iteración de una instrucción while, do, for, foreach.

```
int i = 0;
while (true)
{
    Console.WriteLine(i);
    i++;
    if (i < 10)
        continue;
    else
        break;
}
```

# Objetos Excepción

- En .NET Framework se han definido una serie de clases de excepción.



# Uso de bloques try-catch

- Solución orientada a objetos para el tratamiento de errores
  - Poner el código normal en un bloque try.
  - Tratar las excepciones en un bloque catch aparte.
  - Los bloques try-catch se pueden anidar.

```
try
{
    Console.WriteLine("Escriba un número");
    int i = int.Parse(Console.ReadLine());
}
catch (OverflowException capturada)
{
    Console.WriteLine(capturada);
}
```

Lógica del programa

Tratamiento de errores

# Bloques catch múltiples

- Cada bloque catch captura una clase de excepción.
- Un bloque try puede tener un bloque catch general.
- Un bloque try no puede capturar una clase derivada de una clase capturada en un bloque catch anterior.

```
try
{
    Console.WriteLine("Escriba el primer número");
    int i = int.Parse(Console.ReadLine());
    Console.WriteLine("Escriba el segundo número");
    int j = int.Parse(Console.ReadLine());
    int k = i / j;
}
catch (OverflowException capturada) {...}
catch (DivideByZeroException capturada) {...}
```

# La cláusula finally

- Las instrucciones de un bloque finally se ejecutan siempre.
- Normalmente se utilizan para liberar recursos.

```
Monitor.Enter(x);  
try  
{  
    ...  
}  
finally  
{  
    Monitor.Exit(x);  
}
```

Bloques catch opcionales

# La instrucción throw

- Lanza una excepción apropiada.
- Asigna a la excepción un mensaje significativo.

```
throw expression;
```

```
if (minuto < 1 || minuto >= 60)
{
    throw new InvalidTimeException(minuto + " no es un minuto válido");
    // !! Instrucciones no ejecutadas !!
}
```

# Normas para el tratamiento de excepciones

- Lanzamiento
  - Evitar excepciones para casos normales o esperados.
  - Nunca crear ni lanzar objetos de la clase Exception, en el caso más general utilizar SystemException.
  - Incluir una cadena de descripción en un objeto Exception.
  - Lanzar objetos de la clase más específica posible.
- Captura
  - Ordenar los bloques catch de lo específico a lo general.
  - No permitir que se generen excepciones sin tratar en Main.

# Clases

- Definición e instancia de una clase.
- Constructor por defecto.
- Constructores múltiples.
- Modificadores de acceso
- Datos miembros de una clase.
- Datos miembros de tipo y datos miembros de objeto.
- Métodos.
- Propiedades.

# Definición e instancia de una clase

- Las clases se definen con la palabra reservada class.
- Dentro de ellas se definen datos y métodos.
- Las instancias de una clase se obtienen utilizando el operador new.

```
class CuentaBancaria
{
    private decimal saldo;

    public void Depositar(decimal
monto)
    {
        saldo += monto;
    }
}
```

```
class Test
{
    static void Main()
    {
        CuentaBancaria cuenta1;
        cuenta1 = new CuentaBancaria();
        CuentaBancaria cuenta2 = new
            CuentaBancaria();
        cuenta1.Depositar(1000M);
    }
}
```

# Constructor por defecto

- Características de un constructor por defecto
  - Acceso público.
  - Mismo nombre que la clase.
  - No tiene tipo de retorno (ni siquiera void).
  - No recibe ningún argumento.
  - Inicializa todos los campos a cero, false o null.
  - Si no se define el compilador de C# lo hace por nosotros.
- Sintaxis del constructor

```
class CuentaBancaria
{
    public CuentaBancaria()
    {
    }
}
```

# Constructores múltiples

- Es posible definir múltiples constructores.
- La firma de los constructores debe ser distinta.
- Múltiples constructores permiten inicializar objetos de manera diferente.

```
class Alumno
{
}

class Alumno
{
    public Alumno () {.....}
}

class Alumno
{
    public Alumno()
    {.....}
    public Alumno(int edad, string nombre)
    {.....}
}
```

El compilador declara el constructor por defecto.

El programador declara el constructor por defecto.

El programador declara un constructor vacío y otro con parámetros.

# Modificadores de acceso

Modificador	Significado
public	Acceso no restringido.
protected	Acceso limitado a la propia clase y sus derivadas.
internal	Acceso limitado al Assembly donde esta declarada la clase. Este es el modificador por defecto.
protected internal	Acceso limitado a los tipos derivados de este siempre que estén en el mismo Assembly.
private	Acceso restringido a la misma clase.

# Datos miembros de una clase

- Dato común a todos los objetos de una determinada clase.
- Se declaran de la siguiente manera:

```
<tipoCampo> <nombreCampo>;
```

```
class Alumno
{
    private int Edad;
    string Nombre="Ninguno";
    public Alumno(int edad)
    {
        this.Edad = edad;
        Nombre = "Juan Perez";
    }
    public static void Main()
    {
        Alumno A=new Alumno(23);
    }
}
```

Por defecto los campos son privados a la clase.

Se puede inicializar en el momento de declarar.

La palabra reservada **this** permite hacer referencia al propio objeto (instancia).

Se utiliza en aplicaciones Win32, como punto de entrada de ejecución de la aplicación.

# Datos miembros de tipo y datos miembros de objeto

- Si la definición de un miembro va precedida de la palabra static, este va a pertenecer a la clase (Miembros de tipo), de lo contrario pertenecerá al objeto (Miembro de objeto).
- Para acceder a un miembro del objeto se utiliza la notación <identificadorObjeto>.<miembro> sino, para miembros de tipo <clase>.<campo>.

```
class Cuenta
{
    public static decimal interes = 12;
    public decimal saldo;
}
```

```
Class Test
{
    public static void Main()
    {
        Console.WriteLine(Cuenta.interes);
        Cuenta C = new Cuenta();
        C.saldo = 5000;
    }
}
```

# Cláusula Using

- El cláusula **Using** se trata de una herramienta para que el ciclo de vida de los objetos no se extienda más allá de los métodos en que el objeto ha sido construido.

```
using (var con = new  
SqlConnection("connection string"))  
{  
    conn.Open();  
    ...  
}
```

- Esta objeto **SqlConnection** se va a cerrar sin necesidad de llamar explicitamente a la función **.Close()** y esto va a pasar aun si se ejecuta una excepción sin la necesidad de utilizar **try/catch/finally**.

# Propiedades

- Las propiedades permiten controlar la lectura y escritura de los datos miembro.
- Permiten lograr un buen nivel de encapsulación.
- Se pueden definir propiedades de solo lectura o solo escritura definiendo solo el get o el set.
- Value es un parámetro de entrada del mismo tipo que la propiedad que se usa en el bloque set.

```
class Ejemplo
{
    private intX, intY;
    public int X
    {
        set {intX = value;}
        get {return intX;}
    }
    public int Y
    {
        get {return intY;}
    }
}
```

```
Ejemplo ejeA;
ejeA = new Ejemplo();

ejeA.X = 5;
Console.WriteLine("Valor {0}", ejeA.X);

ejeA.Y = 10;
```

Error: la propiedad Y es de solo lectura.

# Propiedades (Parte II)

- C# 2.0 permite definir diferentes modificadores de visibilidad para los bloques **get** y **set**.

```
class A
{
    string miPropiedad;
    public string MiPropiedad
    {
        get { return miPropiedad; }
        protected set { miPropiedad = value; }
    }
}
```

- Se puede configurar la visibilidad del bloque **get** o del bloque **set** de una cierta propiedad, pero no se puede cambiar la de ambos.

# Definición de métodos

- Es un miembro de la clase que lleva a cabo una acción o calcula un valor.
- Tiene un nombre y contiene un bloque de código.
- Todos los métodos pertenecen a una clase.

```
class Alumno
{
    int NotaParcial;
    void EstablecerNota(int notaParcial)
    {
        NotaParcial = notaParcial;
    }
    float Promedio(int n1, int n2)
    {
        return (float) (n1+n2) /2;
    }
}
```

# Definición de métodos - sintaxis

- Todo método debe devolver algún valor, si no devuelve nada se indica void. Si devuelve algo se debe indicar con la instrucción return <objeto> que debe coincidir con <tipoDevuelto>.

```
<tipoDevuelto> <nombreMétodo> (<parámetros>
{
    <instrucciones>
}
```

# Ejercitación

- Ejecutar paso a paso las sentencias del método Demo de la clase “Csharp\_2” del proyecto “1\_Csharp”.
- Ejecutar y analizar los ejercicios resueltos en la clase “Csharp\_2” del proyecto “1\_Csharp”.
- Resolver los ejercicios solicitados en el método “Ejercicios” de la clase “CSharp\_2” del proyecto “1\_Csharp”.

# Llamadas a métodos

- Una vez definido un método, se puede:
  - Llamar a un método desde dentro de la misma clase.
    - Se usa el nombre del método seguido de una lista de parámetros entre paréntesis.
  - Llamar a un método que está en una clase diferente
    - Hay que indicar al compilador cuál es la clase o instancia que contiene el método que se desea llamar.
    - El método llamado debe tener un modificador de acceso que permita la llamada.
  - Usar llamadas anidadas
    - Unos métodos pueden hacer llamadas a otros, que a su vez pueden llamar a otros métodos, y así sucesivamente.

# Llamadas a métodos (Parte II)

- Si es un método de objeto  
`<objeto>.<nombreMetodo>(<valoresParametros>)`
- Si se invoca desde la misma clase a la que pertenece:  
`<nombreMetodo>(<valoresParametros>)`
- Si es un método de tipo (static)  
`<tipo>.<nombreMetodo>(<valoresParametros>)`

# Devolución de valores (return)

- El método se debe declarar con un tipo que no sea void.
- Se añade una instrucción return con una expresión:
  - Fija el valor de retorno.
  - Se devuelve al llamador.
- Los métodos que no son void deben devolver un valor:

```
static int DosMasDos( )
{
    int a,b;
    a = 2;
    b = 2;
    return a + b;
}
```

```
int x;
x = DosMasDos( );
Console.WriteLine(x);
```

# Variables locales

- Variables locales:
  - Se crean cuando comienza el método.
  - Son privadas para el método.
  - Se destruyen a la salida.
- Variables compartidas
  - Para compartir se utilizan variables de clase.
- Conflictos de ámbito
  - El compilador no avisa si hay conflictos entre nombres locales y de clase.

# Métodos estáticos

- Van precedidos de la palabra static
- Métodos que pertenecen a la clase (Tipo), no a la instancia (objeto).
- Las variables que se utiliza dentro del método deben ser privadas del método o estáticas de la clase.

```
int x; //Error de compilacion
static int y;
static void Incrementa()
{
    x++; //Error x es miembro de objeto
    y=9; //Ok
}
```

# Declaración y llamadas a parámetros

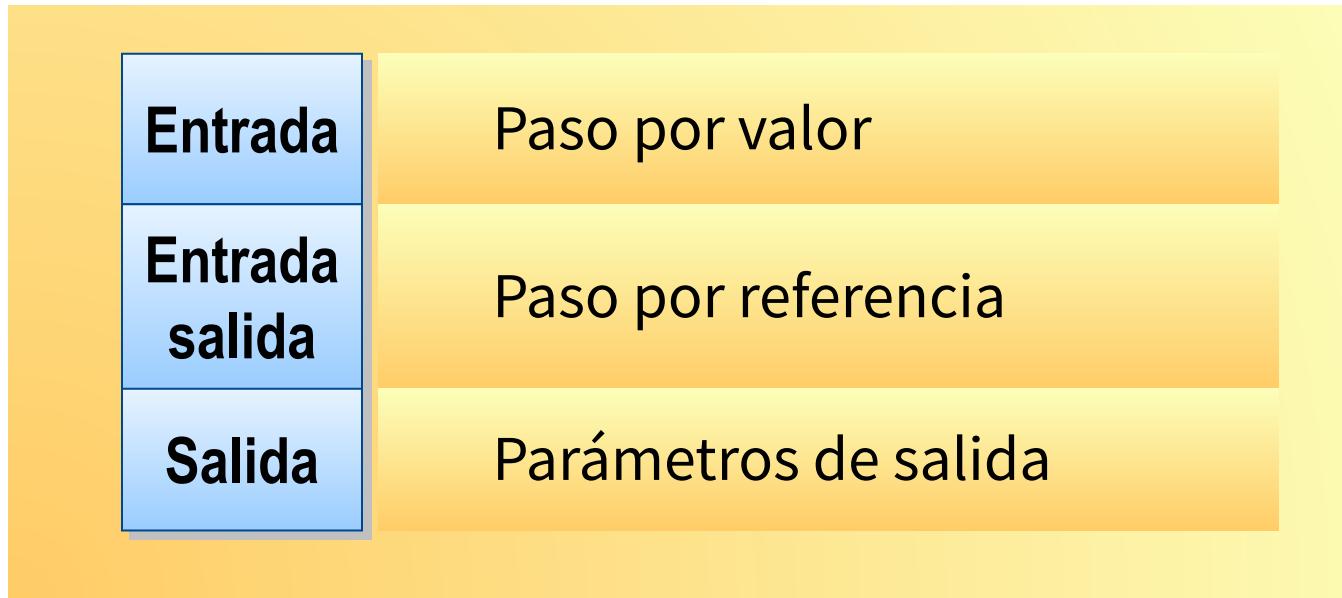
- Declaración de parámetros
  - Se ponen entre paréntesis después del nombre del método.
  - Se definen el tipo y el nombre de cada parámetro.
- Llamadas a métodos con parámetros
  - Un valor para cada parámetro.

```
static void MethodWithParameters(int n, string y)
{ ... }
```

```
MethodWithParameters(2, "Hola, mundo");
```

# Mecanismos de paso de parámetros

- Tres maneras de pasar parámetros.



# Paso por valor

- Mecanismo predeterminado para el paso de parámetros:
  - Se copia el valor del parámetro.
  - Se puede cambiar la variable dentro del método.
  - No afecta al valor fuera del método.
  - El parámetro debe ser de un tipo igual o compatible.
  - Si el objeto es de tipo valor se pasa una copia.
  - Si el objeto es de tipo referencia se pasa una copia de la referencia del mismo.

```
static void SumaUno(int x)
    {x++; // Incrementar x
     Return; }

static void Main( )
{   int k = 6;
    SumaUno(k);
    Console.WriteLine(k); // Muestra el valor 6, no 7
}
```

# Paso por referencia

Qué son los parámetros referencia?

- Una referencia a una posición de memoria.
- Uso de parámetros referencia
  - Se usa la palabra clave ref en la declaración y las llamadas al método.
  - Los tipos y valores de variables deben coincidir.
  - Los cambios hechos en el método afectan al llamador.
  - Hay que asignar un valor al parámetro antes de la llamada al método.

# Parámetros de salida

- ¿Qué son los parámetros de salida?
  - Pasan valores hacia fuera, pero no hacia dentro.
- Uso de parámetros de salida
  - Como **ref**, pero no se pasan valores al método.
  - Se usa la palabra clave **out** en la declaración y las llamadas al método.

```
static void OutDemo(out int p)
{
    // ...
}
int n;
OutDemo(out n);
```

# Normas para paso de parámetros

- Mecanismos
  - El paso por valor es el más habitual
  - El valor de retorno del método es útil para un solo valor
  - ref y/o out son útiles para más de un valor de retorno
  - ref sólo se usa si los datos se pasan en ambos sentidos
- Eficiencia
  - El paso por valor suele ser el más eficaz

# Declaración de métodos sobrecargados

- Métodos que comparten un nombre en una clase.
  - Se distinguen examinando la lista de parámetros.

```
class EjemploSobrecarga
{
    static int Suma(int a, int b)
    {
        return a + b;
    }
    static int Suma(int a, int b, int c)
    {
        return a + b + c;
    }
    static void Main( )
    {
        Console.WriteLine(Suma(1,2) + Suma(1,2,3));
    }
}
```

# Firmas de métodos

- Las firmas de métodos deben ser únicas dentro de una clase.
- Definición de firma.

## Forman la definición de la firma

- Nombre del método
- Tipo de parámetro
- Modificador

## No afectan a la firma

- Nombre de parámetro
- Tipo de retorno del método

# Uso de métodos sobrecargados

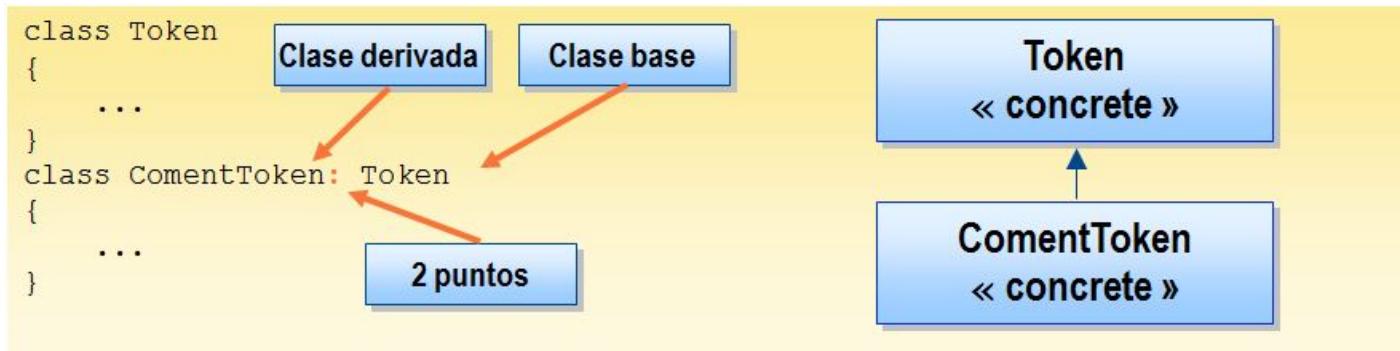
- Conviene usar métodos sobrecargados si:
  - Hay métodos similares que requieren parámetros diferentes.
  - Se quiere añadir funcionalidad al código existente.
- No hay que abusar, ya que:
  - Son difíciles de depurar.
  - Son difíciles de mantener.

# Herencia

- Derivación de clases.
  - Extensión de clases base.
  - Acceso a miembros de la clase base.
  - Llamada a constructores de la clase base.
- Métodos virtuales.
  - Definición.
  - Sustitución (override).
- Clases selladas.
- Clases abstractas.

# Extensión de clases base

- Sintaxis para derivar una clase desde una clase base.



- Una clase derivada hereda la mayor parte de los elementos de su clase base.
- Una clase derivada no puede ser más accesible que su clase base.
- Solo se permite la herencia simple.
- La clase padre se denomina *clase base*, y la hija *clase derivada*.

# Acceso a miembros de la clase base

```
class Token
{
    ...
    protected string name;
}
class ComentToken: Token
{
    ...
    public string Name()
    {
        return name;
    }
}
```

```
class Outside
{
    void Fails(Token t)
    {
        ...
        t.name = "S";      //Error
        ...
    }
}
```

- Los miembros heredados con protección están implícitamente protegidos en la clase derivada.
- Los miembros de una clase derivada sólo pueden acceder a sus miembros heredados con protección.

# Constructores de la clase base

- Las declaraciones de constructores deben usar la palabra base.

```
class Token
{
    protected Token(string name) { ... }
    ...
}
class ComentToken: Token
{
    public ComentToken(string name): base(name) { }
    ...
}
```

- Una clase derivada no puede acceder a un constructor privado de la clase base.
- Se usa la palabra base para habilitar el ámbito del identificador.

# Métodos virtuales

- Son útiles cuando se implementa herencia
- Permite dar una nueva definición al método en las clases hijas
- El método debe ir precedido de la palabra *virtual*
- En la clase hija, si se desea sobreescribir el método se debe preceder al método de la palabra *override*
- Si se precede de *override* un método en una clase hija y el método de la clase padre no va precedido de *virtual*, se produce un error de compilación
- No se puede definir un método como *virtual* y *override* a la vez
- No se pueden declarar como estáticos
- No se pueden declarar como privados

# Definición de métodos virtuales

- **Sintaxis:** Se declara como `virtual`.

```
class Token
{
    ...
    public int LineNumber()
    { ...
    }
    public virtual string Name()
    { ...
    }
}
```

- Los métodos virtuales son polimórficos.

# Sustitución de métodos (override)

- Sólo se sustituyen métodos virtuales heredados idénticos.

```
class Token
{
    ...
    public int LineNumber( ) { ... }
    public virtual string Name( ) { ... }
}
class ComentToken: Token
{
    ...
    public override int LineNumber( ) { ... }      //Error
    public override string Name( ) { ... }
}
```

- Un método override debe coincidir con su método virtual asociado.
- Se puede sustituir un método override ya que es virtual de manera implícita (no se puede declarar explícitamente virtual).
- No se puede declarar explícitamente un override como virtual.
- No se puede declarar un método override como static o private.

# Clases selladas

- Ninguna clase puede derivar de una clase sellada.
- Las clases selladas sirven para optimizar operaciones en tiempo de ejecución.
- Muchas clases de .NET Framework son selladas: String, StringBuilder, etc.
- Sintaxis: Se usa la palabra reservada sealed.

```
namespace System
{
    public sealed class String
    {
        ...
    }
}

namespace Mine
{
    class FancyString: String { ... } /*Error no se puede derivar de una clase sellada*/
}
```

# Clases abstractas

- Es aquella que forzosamente se ha de derivar si se desea que se puedan crear objetos de la misma.
- Se debe anteponer *abstract* a su definición.
- La utilidad de las clases abstractas es que pueden contener métodos para los que no se de directamente una implementación.
- No es necesario que los métodos de una clase abstracta sean abstractos, pero si debe ser abstracta la clase si posee un método abstracto.
- Para los métodos abstractos se debe anteponer el modificador *abstract* y sustituir el código por “;”.
- Todo método abstracto es implícitamente virtual.
- Se puede marcar un método como abstract y override a la vez.

# Declaración de clases abstractas

- Se usa la palabra reservada `abstract`.

```
abstract class Token  
{  
    ...  
}  
class Test  
{  
    static void Main()  
    {  
        new Token();  
    }  
}
```

Token  
{ abstract }

No se pueden crear instancias  
de una clase abstracta

# Implementación de métodos abstractos

- Sintaxis: Se usa la palabra reservada `abstract`.

```
abstract class Token
{
    public virtual string Name( ) { ... }
    public abstract int Longitud( );
}
class ComentToken: Token
{
    public override string Name( ) { ... }
    public override int Longitud( ) { ... }
}
```

- Sólo clases abstractas pueden declarar métodos abstractos.
- Los métodos abstractos no pueden tener cuerpo.

# Uso de métodos abstractos

- Los métodos abstractos son virtuales.
- Los métodos override pueden sustituir a métodos abstractos en otras clases derivadas.
- Los métodos abstractos pueden sustituir a métodos de la clase base declarados como virtuales.
- Los métodos abstractos pueden sustituir a métodos de la clase base declarados como override.

# Generics (C# 2.0)

- Permite que las clases, estructuras, interfaces, delegados y métodos sean parametrizados por el tipo de datos que van a almacenar y manipular
- Es muy útil porque provee chequeo de tipos de datos en tiempo de compilación
- Requiere menos conversiones explícitas entre tipos de datos
- Reduce la necesidad de boxing
- Reduce la necesidad de chequeo en tiempo de ejecución

# ¿Porqué generics?

```
public class Stack
{
    object[] items;
    int count;
    public void Push(object item)
    { ... }
    public object Pop()
    { ... }
}
```

```
Stack stack = new Stack();
stack.Push( new Student() );
Student s = (Student)stack.Pop();
```

```
Stack stack = new Stack();
stack.Push( new Teacher() );
Teacher s = (Teacher)stack.Pop();
```

# Creando y usando Generics

```
public class Stack<T>
{
    T[] items;
    int count;
    public void Push(T item)
{ ... }
    public T Pop()
{ ... }
}
```

```
Stack<int> stack = new Stack<int>();
stack.Push(3);
int x = stack.Pop();
```

```
Stack<Student> stack = new
    Stack<Student>();
stack.Push(new Student());
Student x = stack.Pop();
```

# Listas Genéricas (C# 2.0)

- Es una colección de objetos que posee métodos para agregar, eliminar, buscar, acceder por un índice, etc.
- Es como si fuese un array dinámico.
- Están definidas dentro **System.Collections.Generic**
- Los métodos más comunes son **Add, Insert, Remove, Item, Clear, Count**

# List<T> Ejemplo

```
List<string> dinosaurs = new List<string>();  
Console.WriteLine("\nCapacity: {0}", dinosaurs.Capacity);  
dinosaurs.Add("Tyrannosaurus");  
dinosaurs.Add("Amargasaurus");  
dinosaurs.Add("Mamenchisaurus");  
dinosaurs.Add("Deinonychus");  
dinosaurs.Add("Compsognathus");  
Console.WriteLine();  
foreach(string dinosaur in dinosaurs)  
{  
    Console.WriteLine(dinosaur);  
}  
Console.WriteLine("\nCapacity: {0}", dinosaurs.Capacity);  
Console.WriteLine("Count: {0}", dinosaurs.Count);
```

# Ejercitación

- Ejecutar paso a paso las sentencias del método Demo de la clase “Csharp\_3” del proyecto “1\_Csharp”.
- Ejecutar y analizar los ejercicios varios definidos en la clase Program del proyecto “Csharp\_3”.
- Resolver los ejercicios solicitados en el método “Ejercicios” de la clase “CSharp\_3”.

# Páginas de contenido adicional



# Interfaces

- Es la definición de un conjunto de métodos para los que no se da implementación.
- Se pueden definir clases que deriven de más de una interfaz.
- Toda clase que derive de una interfaz debe implementar todos sus métodos.
- Las interfaces no pueden tener definiciones de campos, operadores, constructores, destructores o miembros estáticos.
- Todos los miembros de la interfaz son implícitamente públicos. No se les puede dar ningún modificador (ni siquiera public).

# Declaración de interfaces

- Sintaxis: Para declarar métodos se usa la palabra reservada interface.

```
interface IToken  
{  
    int LineNumber();  
    string Name();  
}
```

Por convención los nombres de Interfaces empiezan con “I”mayúscula

IToken  
« interface »  
LineNumber( )  
Name( )

Sin modificadores  
de acceso

Métodos sin cuerpo

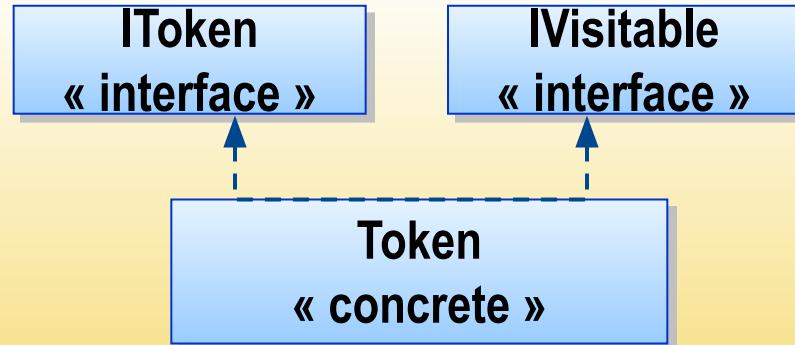
# Implementación de varias interfaces

- Una clase puede implementar cero o más interfaces.

```
interface IToken
{
    string Name();
}

interface IVisitable
{
    void Accept(IVisitable v);
}

class Token: IToken, IVisitable
{ ...
}
```



- Una clase puede ser más accesible que sus interfaces base.
- Una interfaz no puede ser más accesible que su interfaz base.
- Una clase implementa todos los métodos de interfaz heredados.

# Implementación de métodos de interfaz

- El método que implementa debe ser igual que el método de interfaz.
- El método que implementa puede ser virtual o no virtual.

```
class Token: IToken, IVisitable
{
    public virtual string Name( )
    {
        ...
    }
    public void Accept(IVisitable v)
    {
        ...
    }
}
```

**Mismo acceso**  
**Mismo retorno**  
**Mismo nombre**  
**Mismos parámetros**

# Comparación de clases abstractas e interfaces

- Parecidos
  - No se pueden crear instancias de ninguna de ellas.
  - No se puede sellar ninguna de ellas.
- Diferencias
  - Las interfaces no pueden contener implementaciones.
  - Las interfaces no pueden declarar miembros no públicos.
  - Las interfaces no pueden extender nada que no sea una interfaz.

## Unidad 6

# Acceso a datos con Entity Framework y LINQ



Universidad Tecnológica Nacional  
Facultad Regional Córdoba  
Cátedra: Programación de Aplicaciones Visuales 2

# Objetivo

Comprender las características y ventajas que ofrece Entity Framework para agilizar el desarrollo del código de acceso a datos. Desarrollar las habilidades necesarias que permitan utilizar EF y LINQ para acceder y modificar datos almacenados en una base de datos.



# Entity Framework

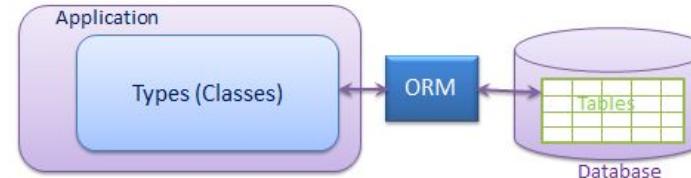


# ORM - Object-Relational mapping

Las bases de datos tradicionales permiten gestionar datos a través de un modelo relacional, en cambio los lenguajes de programación manipulan objetos.

Desarrollar aplicaciones que accedan a datos provenientes de bases de datos relacionales y los manipulen como objetos supone un trabajo de codificación que permite realizar el mapeo entre tablas de una base de datos y objetos entidad de la aplicación.

Se denomina ORM a la técnica para transformar los datos de un modelo relacional (SQL Server, Oracle, MySQL) en datos almacenados en objetos y colecciones de objetos.



# Ventajas de utilizar un ORM

**Abstracción**, un ORM expone una serie de objetos y métodos que permiten al programador enfocarse en la lógica del negocio en vez de la lógica para recuperar y guardar datos en la base de datos.

**Reutilización**, al crear una capa de abstracción que permite manipular los datos de una tabla o conjunto de tablas, la misma puede ser utilizada desde diferentes partes de la aplicación, incluso desde diferentes aplicaciones.

**Estandarización**, utilizar un ORM ayuda a estructurar el código de una manera uniforme a lo largo de toda la aplicación.



# Frameworks ORM

Escribir la capa de abstracción que permite realizar el mapeo entre el modelo relacional y el de objetos implica:

- Escribir cierto código que inicialmente puede parecer extenso y un poco sofisticado.
- Por cada entidad tabla/entidad a mapear se debe repetir algunas partes del código. Esto a su vez requiere conocer cuál es la rutina de pasos a seguir para generar el mapeo, lo cual a su vez suele ser algo laborioso



# Frameworks ORM

Diferentes compañías y comunidades de desarrolladores han creado frameworks que podemos utilizar en nuestras aplicaciones

- Entity Framework (.NET)  
<https://docs.microsoft.com/en-us/ef/ef6/get-started>
- Hibernate (Java) <http://www.hibernate.org/>
- NHibernate (.NET) <http://nhibernate.info/>



# Entity Framework - Definición by MS

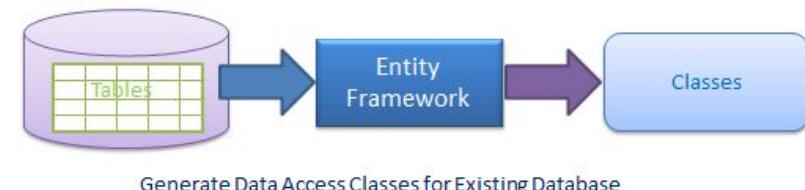
Entity Framework es un conjunto de tecnologías de ADO.NET que permiten el desarrollo de aplicaciones de software orientadas a datos.

Entity Framework permite a los desarrolladores trabajar con datos en forma de objetos y propiedades específicos del dominio, sin tener que preocuparse por las tablas y columnas de la base de datos subyacente donde se almacenan estos datos. Con Entity Framework, los desarrolladores pueden trabajar en un nivel mayor de abstracción cuando tratan con datos, y pueden crear y mantener aplicaciones orientadas a datos con menos código que en las aplicaciones tradicionales.

Desde la versión 5.0, Entity Framework es un framework de código abierto de Microsoft.

# Entity Framework - Escenarios de utilización

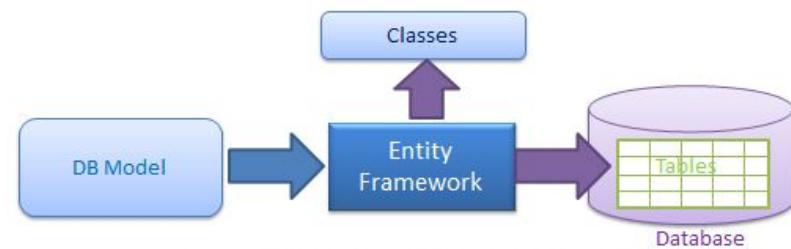
1. Generar las clases de acceso a datos tomando como punto de partida una BD existente, **Database First**.
2. Crear una BD a partir de las clases ya existentes, **Code First**.
3. Crear una BD y el código de acceso a datos mediante el diseño del modelo de datos, **Model First**.



Generate Data Access Classes for Existing Database



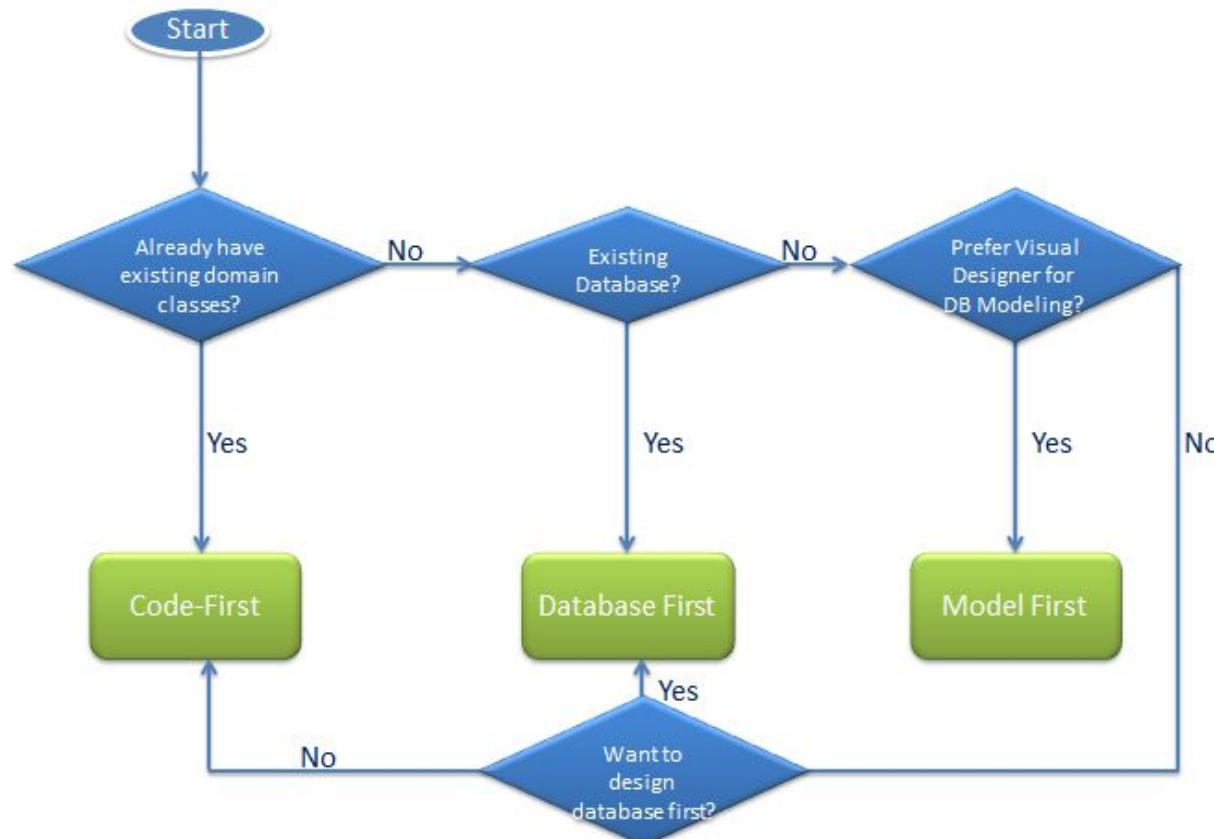
Create Database from the Domain Classes



Create Database and Classes from the DB Model design



# Entity Framework - Elección del escenario



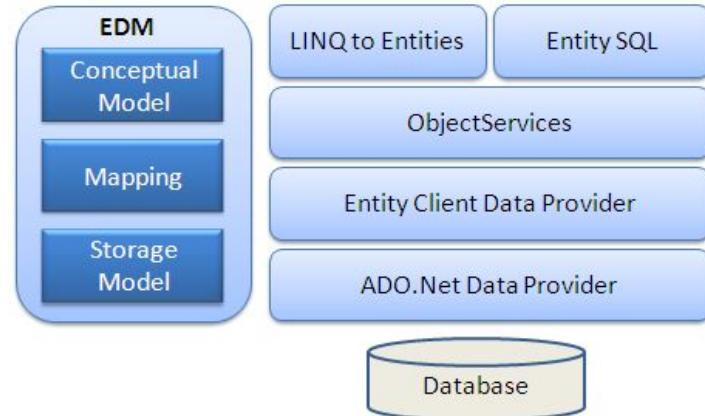
# Entity Framework - Arquitectura

## EDM (Entity Data Model)

**Conceptual Model:** El modelo conceptual contiene el modelo de clases y las relaciones entre ellas. Es independiente del diseño de la base de datos.

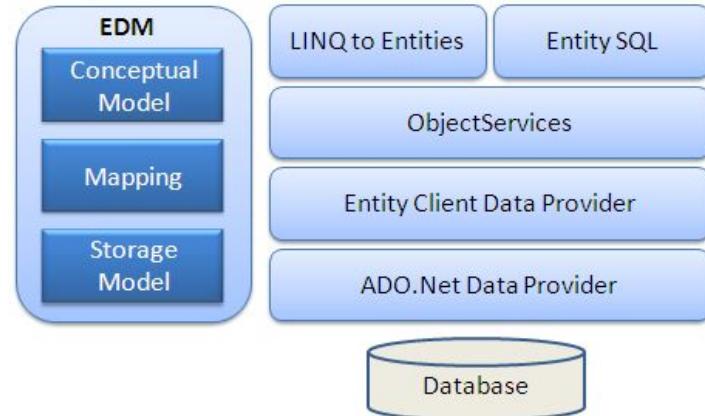
**Storage Model:** Es el modelo de base de datos donde se representan las tablas, vistas, procedimientos almacenados, las relaciones y claves.

**Mapping:** Consiste de toda la información sobre cómo es el mapeo entre el modelo conceptual y el de almacenamiento



# Entity Framework - Arquitectura

**LINQ to Entities:** Lenguaje para escribir consultas a las entidades del modelo de objetos.



**Entity SQL:** Otro lenguaje de consultas aunque no tan simple de utilizar como LINQ to Entities.

**Object Service:** Responsable de convertir los datos entregados por el Entity Client Data Provider en estructuras de objetos.

**Entity Client Data Provider:** Responsable de convertir consultas L2E o Entity SQL en consultas SQL.

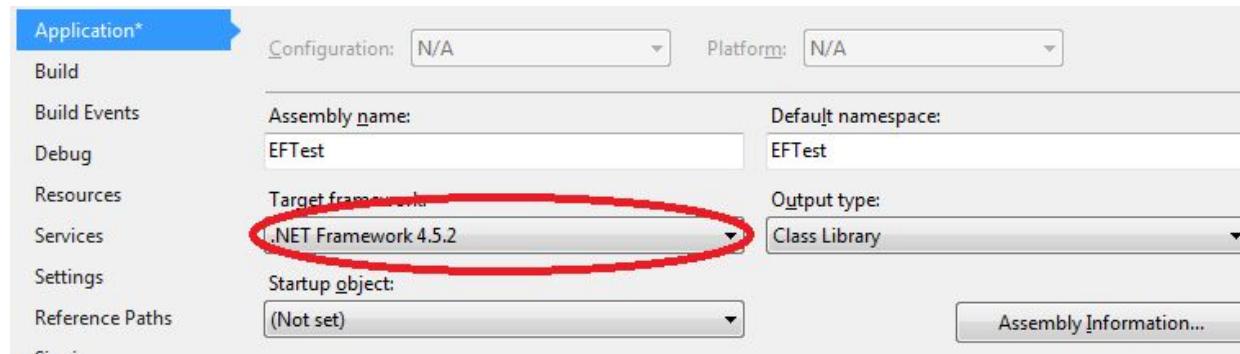
**ADO.Net Data Provider:** Es la capa que se comunica con la base de datos.



# Utilizar el paquete Entity Framework

La última versión disponible de EF es la 6.2, para poder utilizar esta versión es necesario tener el .Net Framework 4.5

Como primera medida crearemos un proyecto del tipo librería de clases y nos aseguraremos de utilizar .NET Framework 4.5, 4.5.1 o 4.5.2



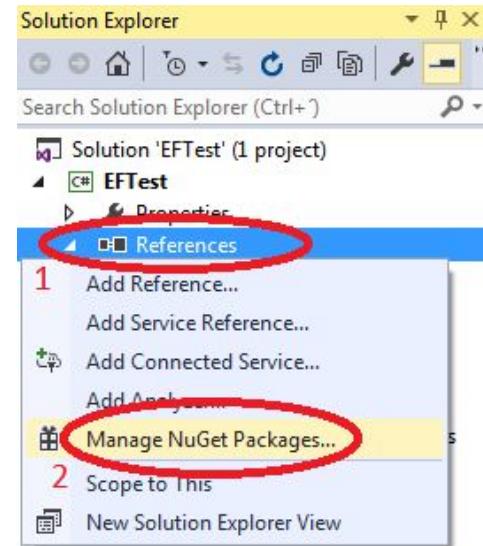
# Utilizar el paquete Entity Framework

La última versión disponible de EF es la 6.2

Para poder utilizar esta versión es necesario tener el .Net Framework 4.5

Para utilizarla en un proyecto se deben seguir los siguientes pasos:

1. Hacer click derecho en referencias del proyecto
2. Seleccionar la opción Manage NuGet Packages



# Instalar el paquete Entity Framework

3. Hacer click en Browse
4. Seleccionar EntityFramework
5. Hacer click en en Install

The screenshot shows the NuGet Package Manager interface. At the top, there are three tabs: 'Browse' (which is highlighted with a red circle), 'Installed', and 'Updates'. Below the tabs is a search bar labeled 'Search (Ctrl+E)' with a red number '3' above it, and a 'Include prerelease' checkbox. To the right, the title 'NuGet Package Manager: EFTest' is displayed, along with a 'Package source: nuget.org' dropdown and a settings gear icon.

The main area displays a list of packages:

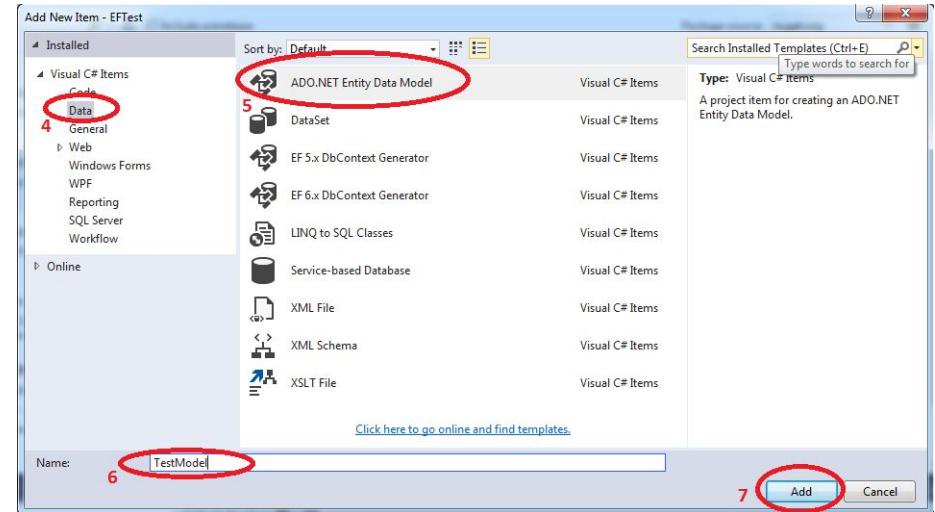
- Newtonsoft.Json** by James Newton-King, 52.9M downloads. Description: Json.NET is a popular high-performance JSON framework for .NET. Version: v10.0.1.
- EntityFramework** by Microsoft, 28.6M downloads. Description: Entity Framework is Microsoft's recommended data access technology for new applications. Version: v6.1.3. This item is circled with a red oval, and the number '4' is placed to its left.
- NUnit** by Charlie Poole, 8,97M downloads. Description: NUnit is a unit-testing framework for all .NET languages with a strong TDD focus. Version: v3.6.1.

On the right side, there is a detailed view for the EntityFramework package. It shows the package name '.NET EntityFramework', the selected version 'Latest stable 6.1.3', and an 'Install' button which is also circled with a red oval and has the number '5' placed to its right. Below the install button, there are 'Options' and 'Description' sections.



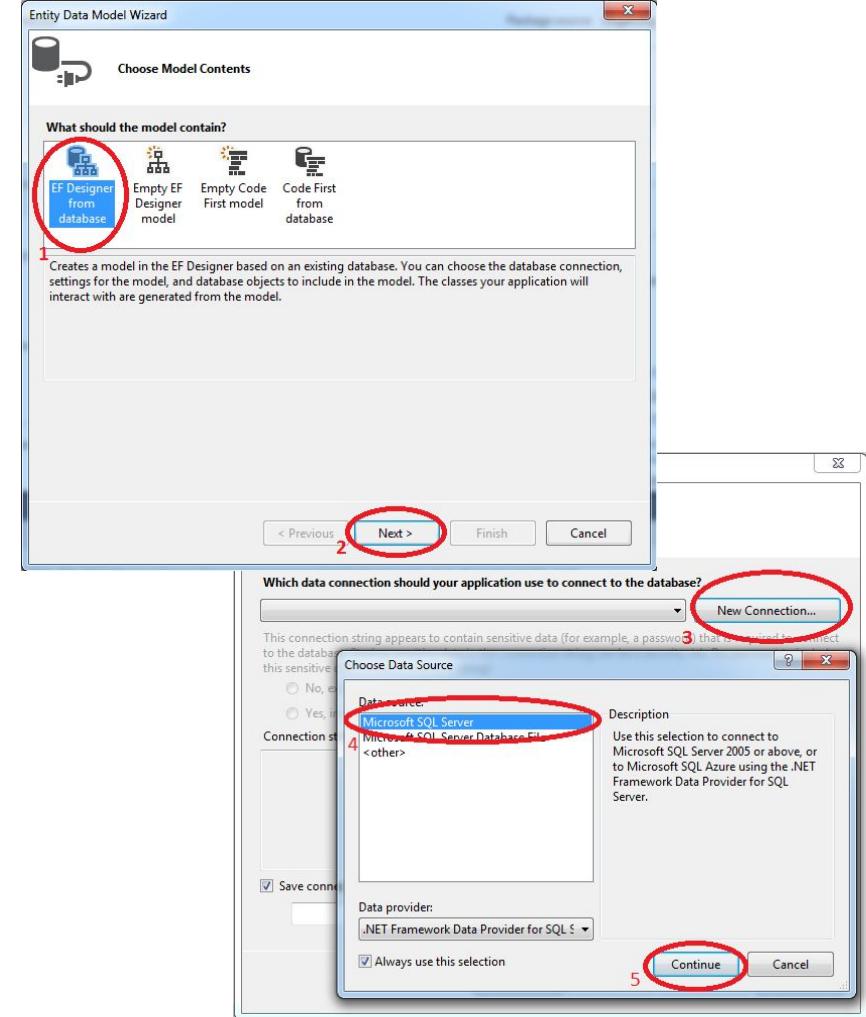
# Agregar un Entity Data Model

1. Hacer click derecho en el proyecto
2. Seleccionar la opción Add
3. Seleccionar la opción New Item...
4. Seleccionar Data
5. Seleccionar ADO.NET Entity Data Model
6. Establecer el nombre del modelo
7. Hacer click en Add



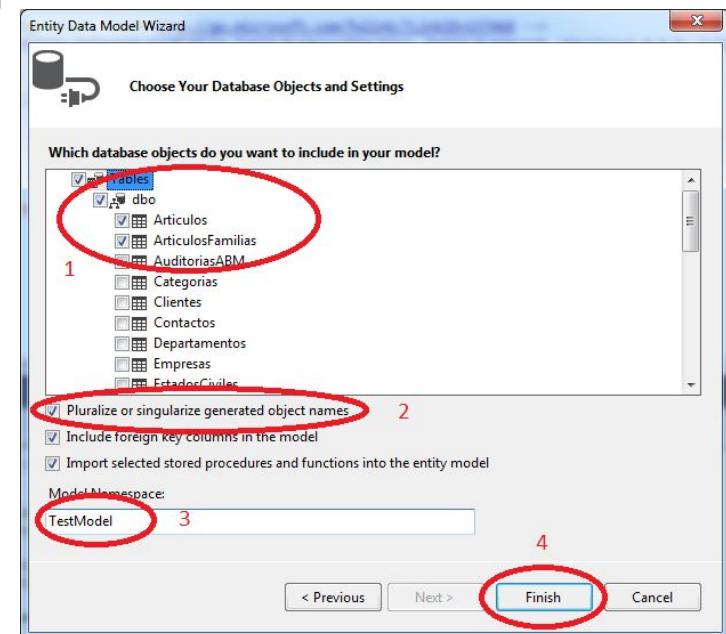
# Configurar el modelo

1. Seleccionar EF Designer from database
2. Seleccionar Add
3. Click en New Connection
4. Click en Microsoft SQL Server
5. Click en Continue
6. Configurar la conexión a la BD
7. Click en Continue



# Configurar el modelo

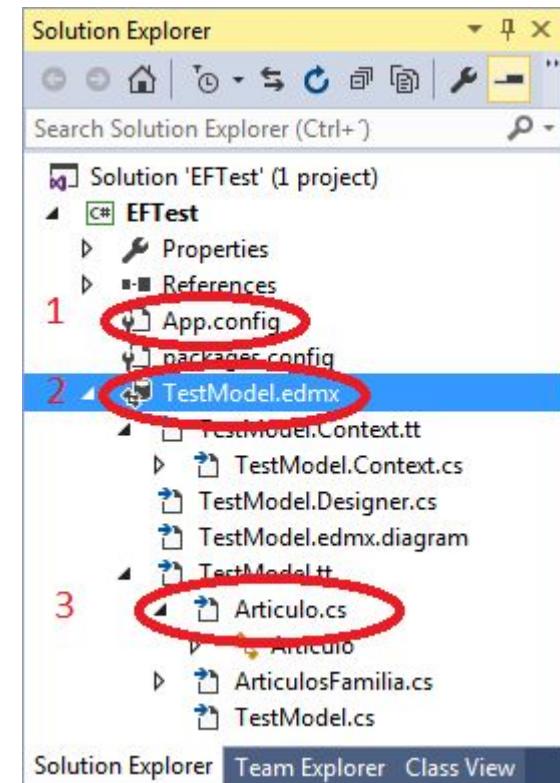
1. En la interfaz para seleccionar los objetos que van a ser parte del modelo seleccionamos las tablas de la BD que serán parte del dominio de la aplicación.
2. Seleccionar el checkbox para que el nombre de las entidades se singularice
3. Indicar el nombre del modelo
4. Click en Finish



# Archivos y Código generado por el modelo

Una vez configurado el Entity Data Model, EF añadirá múltiples archivos al proyecto.

1. En el App.Config se establecen diferentes parámetros generales de EF y la conexión a la BD.
2. El archivo .edmx concentra toda la información relacionada al mapping entre las tablas y las entidades.
3. Clases que representan las entidades.

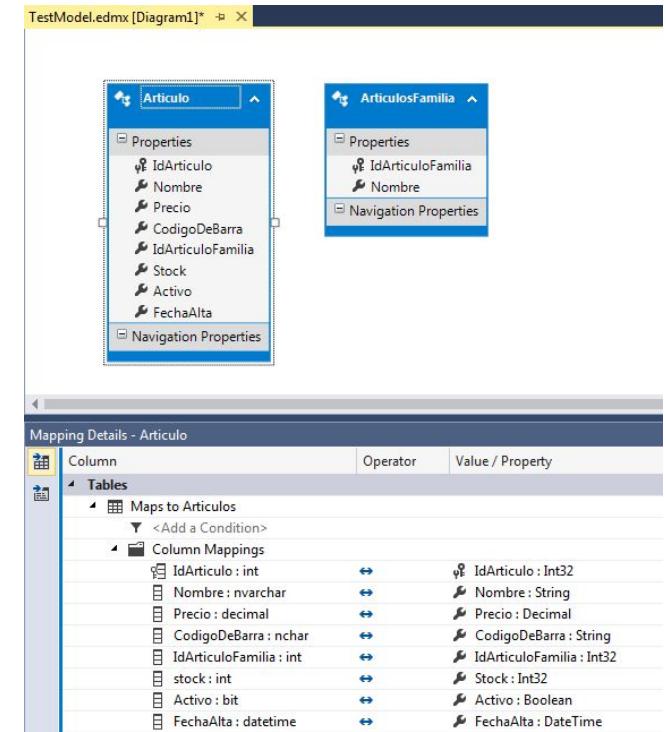


# Revisando el modelo

El archivo .edmx contiene las entidades y define de qué manera se realiza el mapping con las tablas.

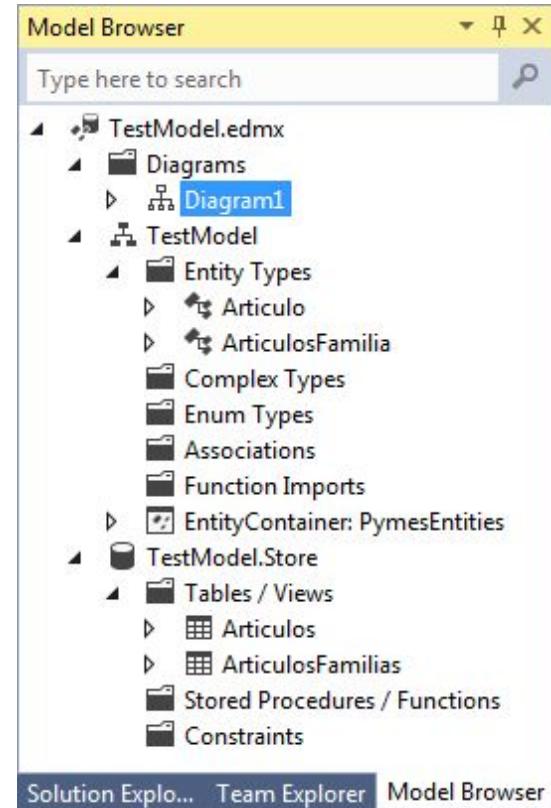
Para realizar cambios en el mapping debemos seleccionar una entidad, hacer click derecho y luego la opción Table Mapping.

Desde esta interfaz podremos modificar nombres de las propiedades, tipos de datos, si los aceptan nulos, etc.



# Model Browser

El diseñador visual no permite ver todo los objetos y el código generado. Haciendo click derecho sobre el diagrama podemos acceder al Model Browser el cual contiene toda la información sobre el modelo conceptual el modelo de datos y el mapeo entre ambos.



Solution Explor... Team Explorer Model Browser



PAV2 - UTN FRC

# DbContext

DbContext es una parte importante de EF, es un nexo entre las clases entidad y la base de datos.



**EntitySet:** DbContext contiene conjunto de entidad (DbSet <TEntity>) para todas las entidades mapeadas contra tablas de la BD.

**Querying:** DbContext convierte consultas LINQ-to-Entities a consulta SQL.

**Change Tracking:** realiza un seguimiento de los cambios que se produjeron en las entidades.

**Persisting Data:** Realiza las operaciones Insert, Update y Delete.

**Caching:** Almacena las entidades que se han recuperado durante el tiempo de vida de una clase de contexto

**Object Materialization:** DbContext convierte datos de tabla en objetos de entidad.



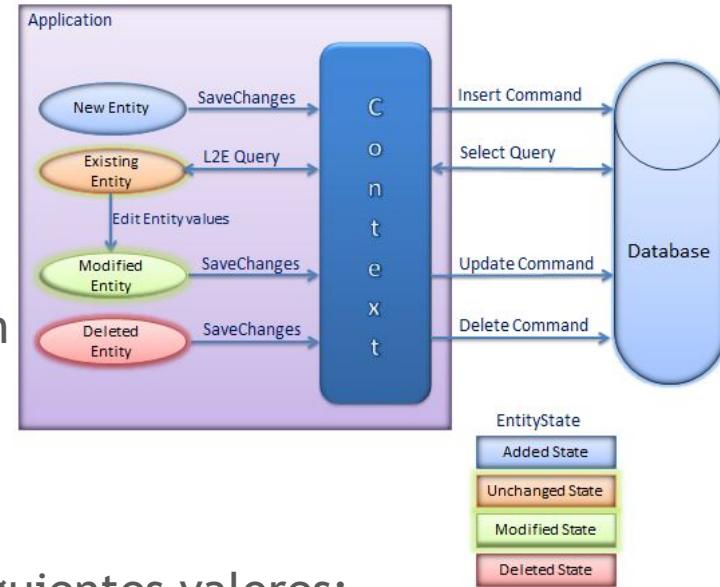
# Estado de las entidades

Durante el tiempo de vida de una entidad, cada entidad tiene un estado de entidad basado en la operación realizada sobre ella a través del contexto (DbContext).

El estado de entidad es un enum de tipo

**System.Data.Entity EntityState** que incluye los siguientes valores:

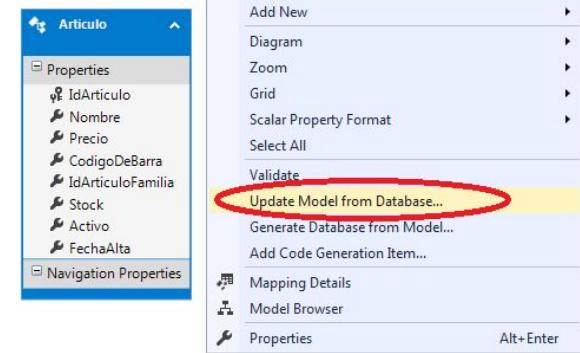
1. **Added**
2. **Deleted**
3. **Modified**
4. **Unchanged**
5. **Detached**



# Modificar el modelo

Es habitual que el modelo de datos sufra modificaciones a lo largo del proceso de desarrollo, por lo cual es necesario “importar” los cambios en las entidades desde la base de datos, para lo cual debemos:

1. Hacer click derecho sobre un diagrama .edmx
2. Seleccionar la opción para actualizar el modelo desde la base de datos.
3. Continuar con los pasos descriptos anteriormente para seleccionar objetos de la base de datos que serán parte del modelo.



# Relaciones entre entidades

EF soporta 3 tipos de relaciones:

1. Uno a Uno
2. Uno a Muchos
3. Muchas a Muchos \*

En la imagen de ejemplo se muestra una relación Uno a Muchos, la cual en código es representada de la siguiente manera:

1 - Entidad Cliente se declara una propiedad de tipo EstadoCivil

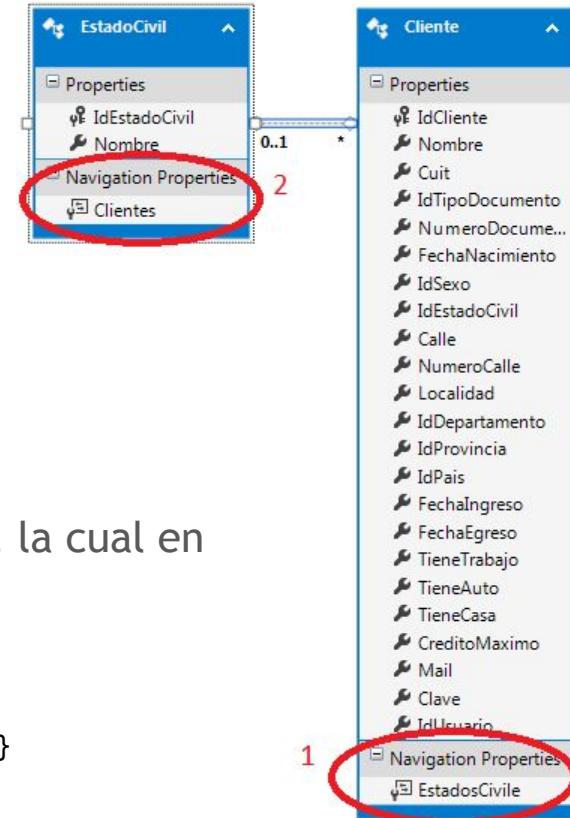
```
public virtual EstadoCivil EstadosCivile { get; set; }
```

2 - Entidad EstadoCivil, se declara una propiedad que devuelve una colección de entidades Cliente

```
public virtual ICollection<Cliente> Clientes { get; set; }
```

Adicionalmente en el constructor se inicializa la colección:

```
this.Clientes = new HashSet<Cliente>();
```



# Ejercitación

- Ejecutar los pasos descritos en las diapositivas 13 a 18.
- Revisar los archivos el código generados por EF desde el Explorador de Soluciones.
- Revisar el mapeo de campos a propiedades de la entidad Articulo y cambiar el nombre de la propiedad stock a Stock.
- Abrir el Explorador de Modelos para revisar los nodos.



# LINQ

# Language Integrated Query



# Que es LINQ

LINQ (Language Integrated Query) es una sintaxis de consulta integrado a C# y VB.NET utilizada para guardar y recuperar datos de diferentes fuentes. Al estar integrado con C#, esto elimina la necesidad de mezclar sintaxis SQL y C#, proporcionando además una única interfaz de consulta para diferentes tipos de fuentes de datos.

LINQ trabaja con objetos para que pueda utilizar los mismos patrones básicos de codificación para consultar y transformar datos en documentos XML, bases de datos SQL, conjuntos de datos ADO.NET, colecciones u otro formato para el que esté disponible un proveedor LINQ.

LINQ está disponible desde el Framework 3.5.

# Que es LINQ

Un desarrollador puede utilizar **LINQ** para trabajar con datos de diferentes orígenes de datos:

- LINQ to SQL
- LINQ to Object
- LINQ to XML
- LINQ to DataSet
- LINQ to Entities
- Otros orígenes de datos que implementen IQueryble



# Un ejemplo sencillo

En el siguiente ejemplo se utiliza LINQ para obtener las palabras que tienen 5 o menos caracteres sin tener que recorrer la lista, seleccionar las palabras que coincidan con el patrón de búsqueda y devuelve una colección con el resultado.

```
using System;
using System.Linq;

class Program
{
    static void Main()
    {
        string[] words = { "hello", "wonderful", "LINQ", "beautiful", "world" };
        //Obtener solo las palabras con 5 o menos caracteres
        var shortWords = from word in words
                        where word.Length <= 5
                        select word;

        //Mostrar por consola
        foreach (var word in shortWords)
        {
            Console.WriteLine(word);
        }
        Console.ReadLine();
    }
}
```

Variable resultado  
Range variable  
IEnumerable o IQueryble Collection  
Operadores de consulta  
select word;  
Operacion condicional



# Un segundo ejemplo

En este caso utilizamos LINQ devolver un objeto o colección de objetos Student(), consultando por los valores de las propiedades del objeto.

```
static void Main(string[] args)
{
    Student[] studentArray = {
        new Student() { StudentID = 1, StudentName = "John", age = 18 } ,
        new Student() { StudentID = 2, StudentName = "Steve", age = 21 } ,
        new Student() { StudentID = 3, StudentName = "Bill", age = 25 } ,
        new Student() { StudentID = 4, StudentName = "Ram" , age = 20 } ,
        new Student() { StudentID = 5, StudentName = "Ron" , age = 31 } ,
        new Student() { StudentID = 6, StudentName = "Chris", age = 17 } ,
        new Student() { StudentID = 7, StudentName = "Rob",age = 19 } ,
    };
    // Use LINQ to find teenager students
    Student[] teenAgerStudents = studentArray.Where(s => s.age > 12 && s.age < 20).ToArray();

    // Use LINQ to find first student whose name is Bill
    Student bill = studentArray.Where(s => s.StudentName == "Bill").FirstOrDefault();

    // Use LINQ to find student whose StudentID is 5
    Student student5 = studentArray.Where(s => s.StudentID == 5).FirstOrDefault();
}
```



# LINQ Query/Method Syntax

Existen dos formas de escribir una consulta LINQ sobre origen de datos basado en una colección `IEnumerable` o `IQueryable`

## 1. Query Syntax o Query Expression Syntax

```
var shortWords = from word in words  
                  where word.Length <= 5  
                  select word;
```

Variable resultado  
Range variable  
IEnumerable o IQueryable Collection  
Operadores de consulta  
Operacion condicional

## 2. Method Syntax o Method Extension Syntax

```
var result = studentArray.Where(s => s.age > 20);
```

Extension method  
Lambda expression



# Operadores de consulta LINQ

Los operadores de consulta estándar en LINQ son en realidad métodos de extensión para los tipos `IEnumerable <T>` e `IQueryable <T>`. Están definidos en las clases `System.Linq.Enumerable` y `System.Linq.Queryable`.

Hay más de 50 operadores de consulta estándar disponibles en LINQ que proporcionan diferentes funcionalidades como filtrado, clasificación, agrupación, agregación, concatenación, etc.



# Operadores de consulta LINQ

## Operadores de consulta en Method Syntax

```
var students = studentList.Where(s => s.age > 20).ToList<Student>();
```

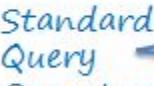
Standard Query Operators  
(Extension methods)



## Operadores de consulta en Query Syntax

```
var students = from s in studentList  
               where s.age > 20  
               select s;
```

Standard Query Operators



# Tipos de operadores de consulta LINQ

- **Filtering:** Where, OfType
- **Sorting:** OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
- **Grouping:** GroupBy, ToLookup
- **Join:** GroupJoin, Join
- **Projection:** Select, SelectMany
- **Aggregation:** Aggregate, Average, Count, LongCount, Max, Min, Sum
- **Quantifiers:** All, Any, Contains
- **Elements:** ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
- **Set:** Distinct, Except, Intersect, Union
- **Partitioning:** Skip, SkipWhile, Take, TakeWhile
- **Concatenation:** Concat
- **Equality:** SequenceEqual
- **Generation:** DefaultEmpty, Empty, Range, Repeat
- **Conversion:** AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList



# Operador Where

Devuelve valores de la colección basados en una función de predicado

## 1. Query Syntax

```
var filteredResult = from s in studentArray  
                      where s.age > 12 && s.age < 20  
                      select s.StudentName;
```

## 2. Method Syntax

```
var filteredResult = studentArray.Where(s => s.age > 12 && s.age < 20);
```



# Operador OfType

El operador OfType filtra la colección basándose en la capacidad de convertir un elemento de una colección a un tipo especificado.

## 1. Query Syntax

```
IList mixedList = new ArrayList();
mixedList.Add(0);
mixedList.Add("One");
mixedList.Add("Two");
mixedList.Add(3);
mixedList.Add(new Student() { StudentID = 1, StudentName = "Bill" });

var stringResult = from s in mixedList.OfType<string>()
                   select s;

var intResult = from s in mixedList.OfType<int>()
                  select s;
```

## 2. Method Syntax

```
var stringResult = mixedList.OfType<string>();
```



# Operadores para particionar resultados

**Skip:** Salta elementos hasta una posición especificada a partir del primer elemento de una secuencia.

```
 IList<string> strList = new List<string>() { "One", "Two", "Three", "Four", "Five" };
var newList = strList.Skip(2);

foreach (var str in newList)
{
    Console.WriteLine(str);
}
```

**SkipWhile :** Salta elementos basados en una condición hasta que un elemento no satisface la condición. Si el primer elemento en sí no satisface la condición, entonces salta 0 elementos y devuelve todos los elementos de la secuencia.

**Take:** Toma elementos hasta una posición especificada partiendo del primer elemento de una secuencia.

**TakeWhile:** Devuelve elementos del primer elemento hasta que un elemento no satisface la condición. Si el primer elemento en sí no satisface la condición, entonces devuelve una colección vacía.



# Operadores de conversión

**AsEnumerable:** Devuelve la secuencia de entrada como `IEnumerable <t>`.

**AsQueryable:** Convierte `IEnumerable` en `IQueryable`, para simular un proveedor de consultas remotas.

**Cast:** Coloca una colección no genérica en una colección genérica (`IEnumerable` a `IEnumerable <T>`).

**ToArray:** Convierte una colección en una matriz.

**ToDictionary:** Coloca elementos en un Diccionario basados en la función selectora de teclas.

**ToList:** convierte colección en lista.

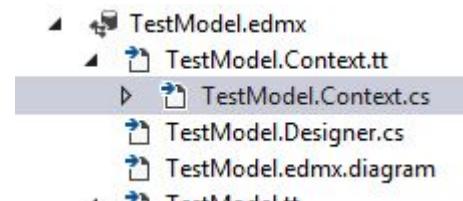


# Utilizar las entidades del modelo



# Punto de acceso a las entidades - DbContext

DbContext es la clase de EF que nos permite interactuar con las entidades del modelo. Al importar las tablas de la BD como las entidades se generará un archivo de nombre [Nombre del modelo].Context.cs



Este archivo contiene una clase que deriva de DbContext la cual contiene una propiedad para cada entidad del modelo. Utilizando esas propiedades podremos obtener colecciones de objetos las entidades.

```
0 references
public virtual DbSet<Articulo> Articulos { get; set; }
0 references
public virtual DbSet<ArticulosFamilia> ArticulosFamilias { get; set; }
0 references
```



# DbSet

La clase DBSet representa un conjunto de entidades que se utiliza para crear, leer, actualizar y eliminar entidades.

Alguno de los métodos más importantes de DbSet son:

- **Add:** Agrega la entidad dada al contexto quedando en el estado Added. Cuando se están guardando los cambios, las entidades en estado Added se insertan en la base de datos. Una vez guardados los cambios, el estado del objeto cambia a Unchanged.
- **Find:** Utiliza el valor de clave principal para intentar encontrar una entidad trackeada por el contexto. Si la entidad no está en el contexto, entonces se ejecutará una consulta y se evaluará con los datos del origen de datos, y se devolverá null si la entidad no se encuentra en el contexto o en el origen de datos.
- **Remove:** Marca la entidad dada como suprimida. Cuando se guardan los cambios, la entidad se elimina de la base de datos.



# Gestores de entidades

EF nos provee de todo el código requerido para realizar las funciones básicas CRUD - Create Read Update Delete, en cambio es nuestra tarea desarrollar gestores de entidades que nos permitan transportar los datos de las entidades a la interfaz de usuario y llamar a los métodos de las entidades que operan contra la base de datos.

En un escenario típico un gestor de entidades va a contener los siguientes métodos:

1. Un método para Obtener un objeto.
2. Un método para Obtener una colección de objetos (utilizando filtros).
3. Un método para Guardar (Insert / Update) los datos un objeto en la base de datos.
4. Un método para Eliminar un objeto.



# Obtener un objeto

Para obtener un objeto de una entidad debemos seguir los siguientes lineamientos:

1. Crear un método estático que retorne un objeto entidad del modelo edmx.
2. El método debe recibir como parámetro con el identificador de la entidad.
3. En la definición del método se debe instanciar a la clase que deriva de DbContext que representa a la base de datos
4. Utilizar el método Find para obtener la entidad devolverla

```
public static Articulo GetByID(int intId)          public static Articulo GetByID2(int intId)
{                                                 {
    using (TestEntities db = new TestEntities())
    {
        return db.Articulos.Find(intId);
    }
}                                                 }

                                                 {
    using (TestEntities db = new TestEntities())
    {
        return db.Articulos.Where(x => x.IdArticulo == intId).FirstOrDefault<Articulo>();
    }
}
```



# Obtener un colección de objetos según un filtro

A diferencia del método para devolver un objeto entidad, en este caso vamos a utilizar LINQ to Entities para aplicar filtros que nos permitan retornar un subconjunto de objetos.

```
public static IEnumerable<Articulo> Buscar(string Nombre, bool? Activo, int numeroPagina, out int RegistrosTotal)
{
    using (TestEntities db = new TestEntities())      //el using asegura el db.dispose() que libera la conexion de la base
    {
        IEnumerable<Articulo> Lista = db.Articulos;

        if (!string.IsNullOrEmpty(Nombre))
            Lista = Lista.Where(x => x.Nombre.Contains(Nombre.ToUpper()));    // equivale al like '%TextoBuscar%'

        if (Activo != null)
            Lista = Lista.Where(x => x.Activo == Activo);

        RegistrosTotal = Lista.Count();

        int RegistroDesde = (numeroPagina - 1) * 10;
        Lista = Lista.OrderBy(x => x.Nombre).Skip(RegistroDesde).Take(10).ToList(); // la instruccion sql recien se ejecuta cuando hacemos ToList()
        return Lista;
    }
}
```



# Grabar los datos de un objeto

En este caso utilizaremos el método SaveChanges para que EF guarde los cambios en la BD.

```
public static void Save(Articulo DTOSel)
{
    // Guardar registro
    using (TestEntities db = new TestEntities())
    {
        try
        {
            if (DTOSel.IdArticulo != 0)
            {
                db.Entry(DTOSel).State = EntityState.Modified;
            }
            else
            {
                db.Articulos.Add(DTOSel);
            }
            db.SaveChanges();
        }
        catch (Exception ex)
        {
            if (ex.ToString().Contains("UK_Articulos_Nombre"))
                throw new ApplicationException("Ya existe otro Artículo con ese Nombre");
            else
                throw;
        }
    }
}
```



# Eliminar un objeto

Para eliminar un objeto debemos recuperarlo a partir de su ID, marcar el estado como Deleted y llamar al método SaveChanges para que EF guarde los cambios en la BD.

```
public static bool Delete(int intId)
{
    using (TestEntities db = new TestEntities())
    {
        Articulo DTOSel = db.Articulos.Where(x => x.IdArticulo == intId).FirstOrDefault<Articulo>();

        if (DTOSel != null)
        {
            db.Entry(DTOSel).State = EntityState.Deleted;

            db.SaveChanges();
            return true;
        }
        else
        {
            return false;
        }
    }
}
```



# Organizar el código en diferentes capas



# Cómo organizar el código de nuestra solución

Un proyecto contiene los archivos que constituyen el sitio Web, archivos de configuración, archivos de código y archivos binarios (código compilado).

Existe un gran número de razones por las cuales los archivos que son parte de una solución deben organizarse en diferentes proyectos. Para nombrar solo algunos podemos mencionar la reutilización, escalabilidad, requisitos de arquitectura, etc.

En el escenario más sencillo es conveniente separar en dos proyectos todo el código asociado con la interfaz de usuario del código que contiene las clases que gestionan las entidades.

En Visual estudio, las definiciones de solución incluyen las relaciones de dependencia entre proyectos.



# Agregar un proyecto a una solución

A los fines de separar la interfaz de usuario del bachead, para este ejemplo vamos a agregar un proyecto de tipo consola a la solución donde ya tenemos el modelo y los gestores de entidades:

1. Click derecho sobre la solución
2. Seleccionar Add > New Project...
3. Elegir el tipo de proyecto (Console Application para el ejemplo)

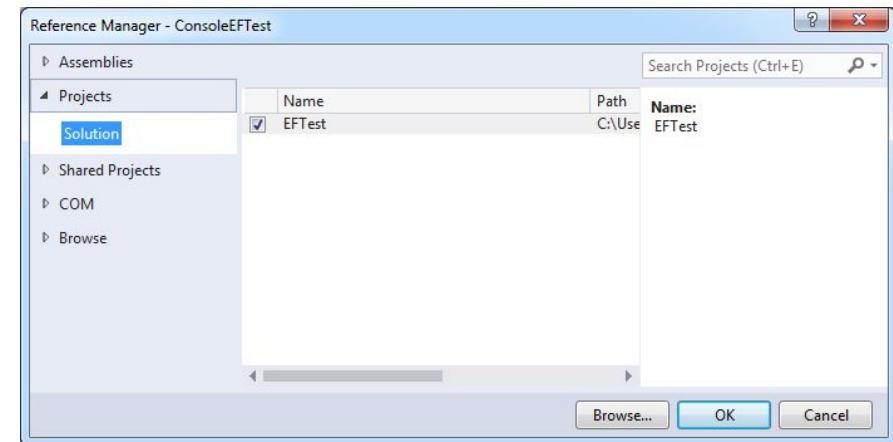


# Agregar una referencia entre dos proyectos

A fin de utilizar objetos declarados en el proyecto “A” desde el proyecto “B”, es necesario agregar una referencia en “B” al proyecto “A”.

Cabe aclarar que desde fuera del proyecto “A” sólo será posible acceder a las clases que sean declaradas como públicas.

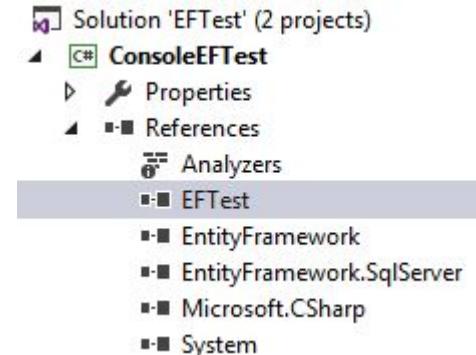
1. Click derecho sobre References
2. Seleccionar Add Reference...



# Referencias adicionales

El proyecto en el cual desarrollaremos la interfaz de usuario también requiere tener una referencia a EF.

Adicionalmente, en las clases desde las cuales se llamen a los gestores de entidades, deberemos agregar la cláusula using a fin de evitar tener que declarar los objetos utilizando el namespace donde está definido el gestor



# Ejercitación

A fin de dotar a la solución de una interfaz de usuario que nos permita utilizar los gestores, realizaremos los siguientes puntos:

1. Agregar a la solución un proyecto de tipo Console Application.
2. Agregar las referencias entre proyectos y a EF.
3. En el método Main, utilizar la instrucción Switch para que a partir de la selección del usuario se ejecuten las diferentes opciones (Agregar un artículo, Eliminar, Listar los artículos)
4. Desarrollar un método por cada opción del menú que llame a los métodos del gestor.



# Observaciones

- **ToList()** o **FirstOrDefault()** ejecuta la consulta LinQ
- Para que funcione en el labsys (no de errores) el asistente de EDMX tener el proyecto sobre el disco d (por tema de permisos).

# Unidad 7

# Implementando el backend con WebApi



Universidad Tecnológica Nacional  
Facultad Regional Córdoba  
Cátedra: Programación de Aplicaciones Visuales 2

# Tecnologías disponibles en ASP.NET

MVC

Web  
Pages

Web  
Forms

Single  
Pages

Web  
API

SignalR

Sites

Services

ASP.NET



# ¿Qué es REST?

Representational State Transfer (transferencia de estado representacional)

**Es una arquitectura para sistemas distribuidos**

Debe cumplir con los siguientes criterios:

- Identificar el recurso
- Interfaces uniformes: Utiliza comandos del protocolo HTTP (GET, PUT, POST, DELETE)
- Mensajes autodescriptivos: Se accede a recursos mediante la URI, y se pueden representar objetos mediante XML o Json
- Navegación a través de enlaces o hipervínculos
- Interacciones sin estado



# ¿Qué es una WebAPI?

- Una API (Application programming interface) es una especificación que se usa como una interfaz para comunicar diferentes componentes de software.
- Web API permite crear APIs del tipo REST
- Permite que sistemas externos usen la lógica de negocios implementadas en la aplicación
- Utiliza URLs en las solicitudes para obtener resultados en formato XML/JSON
- Muy utilizado para el desarrollo de aplicaciones en dispositivos móviles



# ¿Por qué usar Web API?

- Ideal para aplicaciones altamente escalables y para publicarla en la Nube
- Disminuye aproximadamente un 90% la carga del servidor frente a aplicaciones tipo ASP.NET WebForm.
- Independencia de tecnologías/lenguajes.
- Mejora la experiencia del usuario con respecto a la aplicación
- Definido en el protocolo HTTP
- Transferencia de mensajes en formato JSON o XML
- Para aplicaciones RESTful (Representational State Transfer) para arquitectura de sistemas distribuidos como la WWW.
- Permite operaciones CRUD (Create, Read, Update, Delete)



# Sintaxis de la URI

## 1) Protocolo

`http://www.example.com:80/path/`

*Protocol*

## 2) Autoridad

`tp://www.example.com:80/path/to/my`

*Domain Name*

## 3) Puerto

`com:80/path/to/myfile.html?key1=valu`

*Port*

## 4) Ruta

`?n:80/path/to/myfile.html?key1=value1`

*Path to the file*

## 5) Parámetros

`html?key1=value1&key2=value2#Som`

*Parameters*

## 6) Fragmento

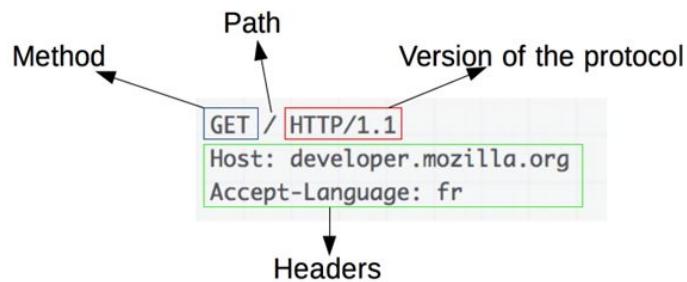
`lue2#SomewhereInTheDocument`

*Anchor*



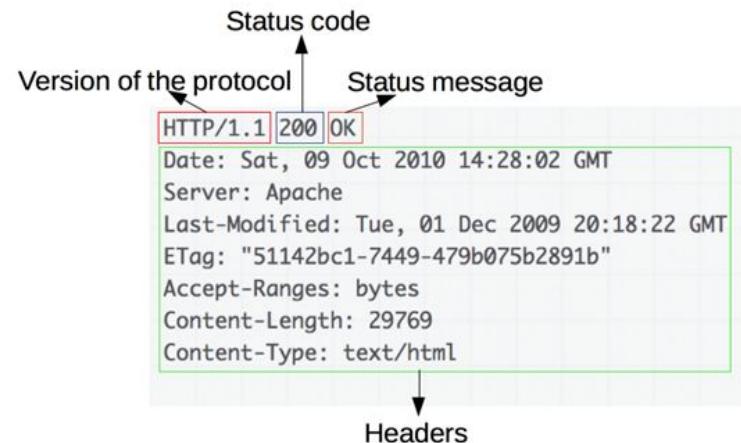
# Análisis de la petición GET

## 1 - CLIENTE - Petición



```
GET / HTTP/1.1
Host: www.w3.org
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0;
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: es-419,es;q=0.8
```

## 2 - SERVIDOR - respuesta



# Práctico

Ejemplo petición GET

- Analizar una petición con las herramientas de depuración de Google Chrome.



# Formato de transferencia de objetos entre capas

## JSON: JavaScript Object Notation

- Estándar abierto basado en texto para intercambio de datos fácilmente entendible por una persona.
- Deriva originalmente de JavaScript para representar estructuras simples de datos o arrays
- Es independiente del lenguaje  
Ejemplo: {"departamento":8,"nombreddepto":"Ventas","director": "juan rodriguez","empleados": [{"nombre": "Pedro", "apellido": "Fernandez"}, {"nombre": "Jacinto", "apellido": "Benavente"} ]}



# ¿Por qué usar JSON en vez de XML?

## JSON

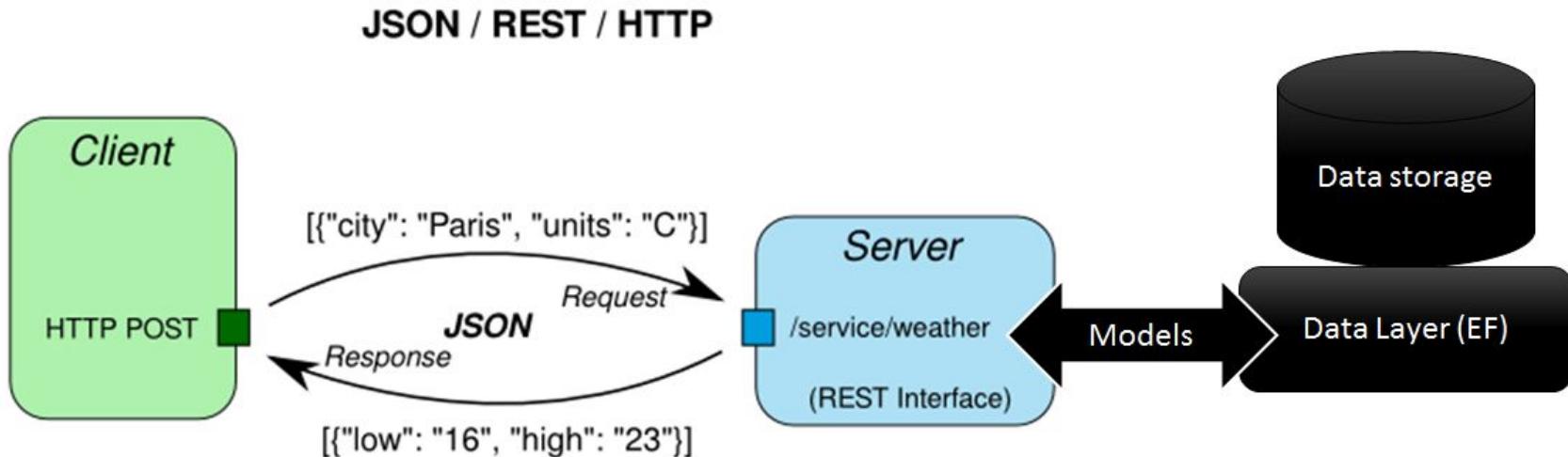
```
{  
  "menu": {  
    "id": "file",  
    "value": "File",  
    "popup": {  
      "menuitem": [  
        {  
          "value": "New", "onclick": "CreateNewDoc()"  
        },  
        {  
          "value": "Open", "onclick": "OpenDoc()"  
        },  
        {  
          "value": "Close", "onclick": "CloseDoc()"  
        }  
      ]  
    }  
  }  
}
```

## XML

```
<menu id="file" value="File">  
  <popup>  
    <menuitem value="New" onclick="CreateNewDoc()" />  
    <menuitem value="Open" onclick="OpenDoc()" />  
    <menuitem value="Close" onclick="CloseDoc()" />  
  </popup>  
</menu>
```



# Cómo funciona una aplicación RESTFull



# Web API Controllers

- Un controller es un objeto que maneja solicitudes HTTP. Todas las API controllers derivan desde ApiController.
- Por defecto ASP.NET Web API va a mapear solicitudes HTTP a métodos específicos llamadas acciones

Acción	Verbo	URI relativa	Método Controlador
Obtener todos los Clientes	GET	/api/clientes	Get()
Obtener un cliente por id	GET	/api/clientes/id	Get(int id)
Crear un nuevo cliente	POST	/api/clientes	Post(PostModel value)
Actualizar un cliente	PUT	/api/clientes/id	Put(int id, PostModel value)
Borrar un cliente	DELETE	/api/clientes/id	Delete(int id)
Obtener un cliente por nombre	GET	/api/clientes?nombre=nombre	Get(string nombre)



# Comportamiento por defecto de WebAPI



`http://localhost:1337/api/posts`



```
public class PostsController : ApiController
{
    public string Get()
    {
        return "Some data";
    }
}
```



# Formato de devolución de datos - MIME Types

- Web API puede devolver datos en formato JSON, XML u otro formato
- Web API usa un objeto para formatear los datos:
  - Formatear o serializar la información que usa o devuelve un Web API REST
  - Controlar el tipo de medio del encabezado HTTP
  - Formatear todo el contenido que el servidor renderiza a los clientes
- El formato de medios está especificado en la clase `MediaTypeFormatter`



# Controlador - Valores de retorno

## Tipo de dato de retorno

**Void**

Devuelve el código 204 (No Content)

**HttpResponseMessage**

Convierte directamente el valor de retorno en un mensaje de respuesta dentro del HTTP

**IHttpActionResult**

Llama a ExecuteAsync para crear un mensaje **HttpResponseMessage**

Otro tipo de retorno

Escribe un valor de retorno serializado dentro del mensaje de respuesta y devuelve el valor 200 (OK)



# Ejemplo valor de retorno void

C#

```
public class ValuesController : ApiController
{
    public void Post()
    {
    }
}
```

HTTP response:

console

```
HTTP/1.1 204 No Content
Server: Microsoft-IIS/8.0
Date: Mon, 27 Jan 2014 02:13:26 GMT
```



# Ejemplo valor retorno HttpResponseMessage

```
public class ValuesController : ApiController
{
    public HttpResponseMessage Get()
    {
        HttpResponseMessage response = Request.CreateResponse(HttpStatusCode.OK, "value");
        response.Content = new StringContent("hello", Encoding.Unicode);
        response.Headers.CacheControl = new CacheControlHeaderValue()
        {
            MaxAge = TimeSpan.FromMinutes(20)
        };
        return response;
    }
}
```

Response:

console	Copy
HTTP/1.1 200 OK Cache-Control: max-age=1200 Content-Length: 10 Content-Type: text/plain; charset=utf-16 Server: Microsoft-IIS/8.0 Date: Mon, 27 Jan 2014 08:53:35 GMT  hello	



# Ejemplo para un valor de retorno IHttpActionResult

```
public IHttpActionResult Get (int id)
{
    Product product = _repository.Get (id);
    if (product == null)
    {
        return NotFound(); // Returns a NotFoundResult
    }
    return Ok(product); // Returns an OkNegotiatedContentResult
}
```



```
public interface IHttpActionResult
{
    Task<HttpResponseMessage> ExecuteAsync(CancellationToken cancellationToken);
}
```



# Ejemplo de devolución de otros tipos de datos

Solicitud

```
GET http://localhost/api/products HTTP/1.1  
User-Agent: Fiddler  
Host: localhost:24127  
Accept: application/json
```

Controlador

```
public class ProductsController : ApiController  
{  
    public IEnumerable<Product> Get()  
    {  
        return GetAllProductsFromDB();  
    }  
}
```

Respuesta

```
HTTP/1.1 200 OK  
Content-Type: application/json; charset=utf-8  
Server: Microsoft-IIS/8.0  
Date: Mon, 27 Jan 2014 08:53:35 GMT  
Content-Length: 56  
  
[{"Id":1,"Name":"Yo-yo","Category":"Toys","Price":6.95}]
```



PAV2 - UTN FRC

# WebAPI - Código de respuesta HTTP de métodos

Algunos códigos de respuesta comúnmente utilizados

- HTTP 200 - todo es OK
- HTTP 201 - algo fue creado
- HTTP 404 - algo no se pudo encontrar
- HTTP 400 - Bad request

```
1 [ResponseType(typeof(Order))]  
2 public IHttpActionResult GetOrder(int or  
3 {  
4     var order = orderFinder.FindOrder(or  
5     if (order == null)  
6         return NotFoundResult(Request);  
7     else  
8         return Ok(order);  
9 }
```



# Ruteo con Web API

- El ruteo es cómo enlaza ASP.NET Web API la URI a un controlador y una acción específica
- Permite utilizar los nombres de los controladores de la API y una convención de nombre para las acciones de ruteo de las solicitudes web
- Web API 2 soporta el ruteo con atributos
- Alternativamente se puede usar los siguientes atributos para controlar el mapeo de las solicitudes (HTTP Verbo + URL) para acciones en el controlador
  - Los atributos `HttpGet`, `HttpPost`, `HttpPut` o `HttpDelete`
  - El atributo `AcceptVerbs`
  - El atributo `ActionName`



# Web API - Utilización de rutas y controladores

El ruteo en ASP.NET involucra lo siguiente:

- ASP.NET agrega una ruta por defecto para:
  - Mapear una URL a un controlador
  - Soportar operaciones del tipo REST WebAPI
- Se puede modificar la ruta predeterminada para incluir múltiples acciones en el mismo método HTTP
- Se puede utilizar la clase WebApiConfig para:
  - Modificar el ruteo
  - Permitir que múltiples versiones de la API coexistan en el mismo proyecto



# Configuración del ruteo

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Attribute routing.
        config.MapHttpAttributeRoutes();

        // Convention-based routing.
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

GET `http://localhost:34701/api/products/1?version=1.5&details=1`

```
public class ProductsController : ApiController
{
    public IEnumerable<Product> GetAll() {}
    public Product GetBy Id(int id, double version = 1.0) {}
    [HttpGet]
    public void FindProductsByName(string name) {}
    public void Post(Product value) {}
    public void Put(int id, Product value) {}
}
```



# Rutas a través del atributo de ruteo

Con un parámetro...

```
http://example.com/customers/1/orders ,
```

```
public class OrdersController : ApiController
{
    [Route("customers/{customerId}/orders")]
    [HttpGet]
    public IEnumerable<Order> FindOrdersByCustomer(int customerId) { ... }
}
```



Con varios parámetros.....

```
[Route("customers/{customerId}/orders/{orderId}")]
public Order GetOrderByCustomer(int customerId, int orderId) { ... }
```



# Utilidad de patrones por atributo - ejemplos

Versionado de APIs

/api/v1/products

/api/v2/products

Sobrecarga de segmentos URI

/orders/1

/orders/pending

Múltiple tipo de parámetros

/orders/1

/orders/2013/06/16



# Métodos HTTP que puede decorar una acción

- **[HttpDelete]** De manera predeterminada Web API comienza la búsqueda con el prefijo del nombre del método ej: GetProducts o PutCustomers para el verbo HTTPGet del controlador Products o el vertbo HTTPPut para el controlador de Customers.
- **[HttpGet]**
- **[HttpHead]**
- **[HttpOptions]**
- **[HttpPatch]** Si el método no posee el prefijo que coincide con el verbo , se puede especificar la asociación con decorartors:
- **[HttpPost]**
- **[HttpPut]**

```
[Route("api/books")]
[HttpPost]
public HttpResponseMessage CreateBook(Book book) { ... }
```



# Restricciones de ruteo para selección del método

Si hay más de un método para el mismo verbo HTTP, se puede restringir cómo se determina el método a ejecutar a través de constraints

```
[Route("users/{id:int}")]  
public User GetUserById(int id) { ... }  
  
[Route("users/{name}")]  
public User GetUserByName(string name) { ... }
```



# Parámetros opcionales y valores por defecto

## Parámetros opcionales

```
/api/books/locale/1033  
/api/books/locale
```

```
public class BooksController : ApiController  
{  
    [Route("api/books/locale/{lcid:int?}")]
    public IEnumerable<Book> GetBooksByLocale(int lcid = 1033) { ... }
}
```

## Valores por defecto

```
public class BooksController : ApiController
{
    [Route("api/books/locale/{lcid:int=1033}")]
    public IEnumerable<Book> GetBooksByLocale(int lcid) { ... }
}
```



# Manejo de excepciones

**HttpError:** Permite devolver información consistente del error en el cuerpo del mensaje

```
public HttpResponseMessage GetProduct(int id)
{
    Product item = repository.Get(id);
    if (item == null)
    {
        var message = string.Format("Product with id = {0} not found", id);
        return Request.CreateErrorResponse(HttpStatusCode.NotFound, message);
    }
    else
    {
        return Request.CreateResponse(HttpStatusCode.OK, item);
    }
}
```



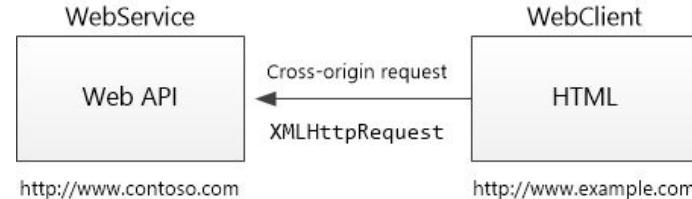
```
HTTP/1.1 404 Not Found
Content-Type: application/json; charset=utf-8
Date: Thu, 09 Aug 2012 23:27:18 GMT
Content-Length: 51

{
    "Message": "Product with id = 12 not found"
}
```



# CORS

Cross Origin Resource Sharing (CORS) es un estándar W3C que permite a un servidor relajar la política del mismo origen.



Para relajar la política del mismo origen:

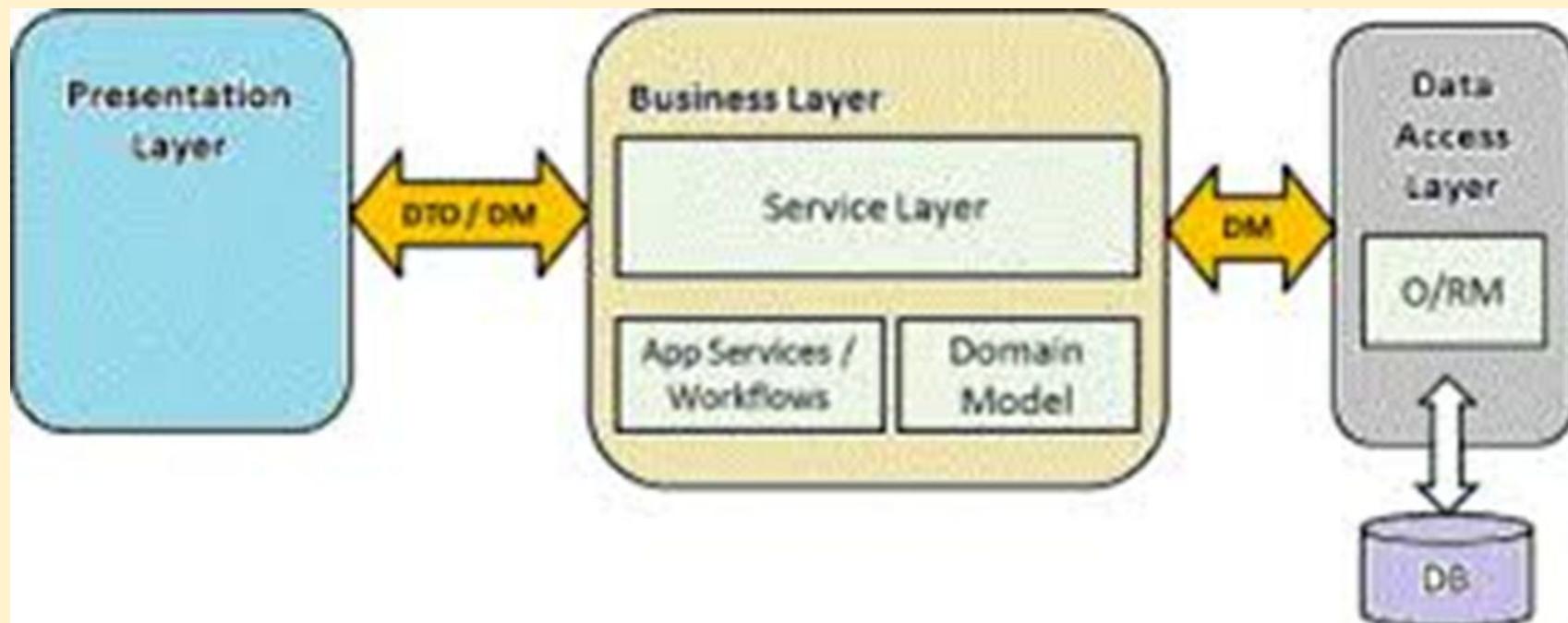
1. Instalar el paquete NuGet **Install-Package Microsoft.AspNet.WebApi.Cors**
2. En el archivo WebApiConfig agregar en el método Register lo siguiente:

```
var cors = new EnableCorsAttribute("*", "*", "*");
config.EnableCors(cors);
```

fuente: <https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api>



# Utilización de DTO (Data Transfer Object) en Web API



# Observaciones:

Global.asax Application\_Start()  
App\_Start/WebApiConfig.cs  
c# parametros opcionales y por defecto

