**Modulo 2**

**Desarrollo Front End Estático**

**Bienvenida al Módulo II**

Te damos la bienvenida al módulo II de #Yo Programo, en él comenzaremos a trabajar con lenguajes etiquetado y de programación que les dan forma a todas las páginas web, incorporaras los conocimientos elementales para poder diseñar tu propio porfolio web full stack que te servirá para volcar los conocimientos adquiridos y luego publicarlo así las empresas pueden conocerte.

Te invitamos en este módulo a que comiences a diseñar la maqueta web de lo que será tu proyecto integrador.

Proyecto Integrador

Tu trabajo final o proyecto integrador será la construcción de tu propio perfil, En esta ocasión te vamos a dejar un diseño base como idea inicial para tu portfolio, pero recuerda que puedes crear el diseño que desees aplicando lo aprendido, puedes descargarlo haciendo clic aquí.

**¿Qué es HTML?**

HTML significa Hyper Text Markup Language, es el lenguaje más utilizado en la Web para desarrollar páginas web. HTML fue creado por Berners-Lee a finales de 1991, pero "HTML 2.0" fue la primera especificación HTML estándar que se publicó en 1995.

HTML 4.01 era una versión principal de HTML y se publicó a finales de 1999. Aunque la versión HTML 4.01 se usa ampliamente actualmente tenemos la versión HTML-5, que es una extensión de HTML 4.01 y esta versión se publicó en 2012.

**¿Por qué aprender HTML?**

Originalmente, HTML se desarrolló con la intención de definir la estructura de documentos como encabezados, párrafos, listas, etc., para facilitar el intercambio de información científica entre investigadores. Ahora, HTML se está utilizando ampliamente para formatear páginas web con la ayuda de diferentes etiquetas disponibles en lenguaje HTML.

HTML es una OBLIGACIÓN para que los estudiantes y los profesionales se conviertan en grandes ingenieros de software, especialmente cuando están trabajando en el dominio de desarrollo web.

Enumeraremos algunas de las ventajas claves de aprender HTML:

Crear un sitio web: podemos crear un sitio web o personalizar una plantilla web existente si conocemos bien HTML.

Convertirnos en diseñadores web: si deseamos comenzar una carrera como diseñador web profesional, el diseño de HTML y CSS es una habilidad imprescindible.

Comprender la Web: si deseamos optimizar nuestro sitio web para aumentar su velocidad y rendimiento, es bueno conocer HTML para obtener los mejores resultados.

Aprender otros lenguajes: una vez que comprendamos los conceptos básicos de HTML, otras tecnologías relacionadas como javascript, php o angular se volverán más fáciles de entender.

**Aplicaciones de HTML**

Como se mencionó anteriormente, HTML es uno de los lenguajes más utilizados en la web. Vamos a enumerar algunos ejemplos y aplicaciones:

Desarrollo de páginas web: HTML se utiliza para crear páginas que se representan en la web. Casi todas las páginas de la web tienen etiquetas html para mostrar sus detalles en el navegador.

Navegación por Internet: HTML proporciona etiquetas que se utilizan para navegar de una página a otra y se utiliza mucho en la navegación por Internet.

Interfaz de usuario responsiva: las páginas HTML hoy en día funcionan bien en todas las plataformas, dispositivos móviles, pestañas, computadoras de escritorio o portátiles debido a la estrategia de diseño responsivo.

Las páginas HTML de soporte sin conexión, una vez cargadas, pueden estar disponibles sin conexión en la máquina sin necesidad de Internet.

Desarrollo de videojuegos: HTML5 tiene soporte nativo para una experiencia rica y ahora también es útil en el campo del desarrollo de videojuegos.

**Introducción a HTML**

**¿Qué es HTML?**

Por sus siglas en inglés HTML (Hypertext Markup Language) se lo define como un lenguaje de marcado estándar que nos permite crear y presentar páginas web de una manera versátil y eficiente.

El HTML describe la ESTRUCTURA y el CONTENIDO de una página web y es un lenguaje que consiste en etiquetas agrupadas o estructuradas de una manera lógica en función de lo que necesitamos como vista. Estas etiquetas le dicen al “navegador web” cómo debe mostrar el contenido.

HTML es un lenguaje y como tal tiene su propio vocabulario (palabras) y su propia gramática (reglas).

En la actualidad y desde hace varios años, el HTML está definido por el W3C. “El Consorcio World Wide Web (W3C) es una comunidad internacional que desarrolla estándares abiertos para asegurar el crecimiento a largo plazo de la Web”.

**¿Qué NO es HTML?**

HTML se diferencia de un lenguaje de programación, porque no define el COMPORTAMIENTO (lógica) de las páginas web.

HTML utiliza "marcas" para etiquetar texto, imágenes y otro contenido para mostrarlo en un navegador Web. Las marcas HTML incluyen "elementos" especiales como <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, <div>, <span>, < img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video>, <ul>, <ol> , <li> y muchos otros.

Es decir, con el lenguaje HTML en sí mismo, sólo podremos presentar texto e imágenes básicas en el navegador web. Para la lógica y la presentación estética de una página web, utilizaremos otros lenguajes complementarios al HTML, tales como Javascript y CSS.

Un elemento HTML se distingue de otro texto en un documento mediante "etiquetas", que consiste en el nombre del elemento rodeado por "<" y ">". El nombre de un elemento dentro de una etiqueta no distingue entre mayúsculas y minúsculas. Es decir, se puede escribir en mayúsculas, minúsculas o una mezcla. Por ejemplo, la etiqueta <title> se puede escribir como <Title>, <TITLE> o de cualquier otra forma.

Las etiquetas vienen de a pares, la primera etiqueta del par se llama etiqueta o tag de apertura y se ve como esta: <p>. El tag o etiqueta de cierre se escribe igual que el de apertura, pero con la barra de división y se ve como esto: </p>.

Si quieres especificar que se trata de un párrafo, podrías encerrar el texto con la etiqueta de párrafo (<p>)

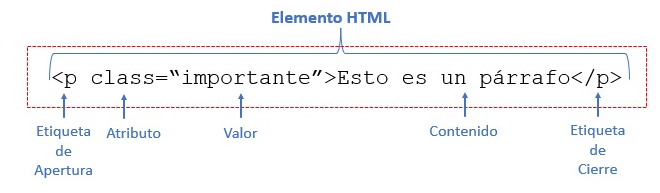
Veamos un ejemplo y su anatomía en la siguiente imagen



Las partes principales del elemento son:

1. **La etiqueta de apertura** : consiste en el nombre del elemento (en este caso p), encerrado por paréntesis angulares (<>) de apertura y cierre. Establece dónde comienza o comienza dónde a tener efecto el elemento —en este caso, dónde es el comienzo del párrafo—.
2. **La etiqueta de cierre** : es igual que la etiqueta de apertura, excepto que incluye una barra de cierre (/) antes del nombre de la etiqueta. Establece dónde termina el elemento —en este caso dónde termina el párrafo—.
3. **El contenido** : este es el contenido del elemento, que en este caso es sólo texto.
4. **El elemento** : la etiqueta de apertura, más la etiqueta de cierre, más el contenido equivalente al elemento.

Los elementos y las etiquetas no son las mismas cosas. Las etiquetas comienzan o terminan un elemento en el código fuente, mientras que los elementos son parte del DOM (Document Object Model) tema que abordaremos más adelante en detalle.



Las **etiquetas de apertura** contienen información adicional acerca del elemento, el cual no desea que aparezca en el contenido real del elemento. Aquí **clase** es el nombre del atributo e  **"importante"** es el valor del atributo. En este caso, el atributo class permite darle al elemento un nombre identificativo, que se puede utilizar luego para encontrarlo dentro de toda la página y poder aplicarle estilos entre otras cosas.

Un atributo debe tener siempre:

1. Un espacio entre este y el nombre del elemento (o del atributo previo, si el elemento ya posee uno o más atributos).
2. El nombre del atributo, seguido por un signo de igual (=).
3. Comillas de apertura y de cierre, encerrando el valor del atributo.

Los atributos siempre se incluyen en la etiqueta de apertura de un elemento, nunca en la de cierre.

**Elementos anidar**

Puedes también colocar elementos dentro de otros elementos —esto se llama anidamiento—. Si, por ejemplo, quieres resaltar una palabra del texto (en el ejemplo la palabra «muy»), podemos encerrarla en un elemento <strong>, que significa que dicha palabra se debe enfatizar:



Los elementos deben abrirse y cerrarse ordenadamente, de forma tal que se encuentren claramente.

Te mostramos el mismo ejemplo pero con los elementos mal anidados, en el ejemplo de abajo, La etiqueta de apertura del elemento <p> primero, luego la del elemento <strong>, por lo tanto, debes cerrar esta etiqueta primero, y luego la de <p>. Esto es incorrecto:

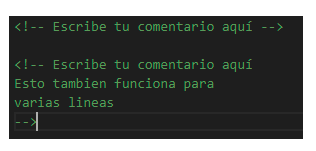


Si esto te llega a pasar, te vas a dar cuenta porque en el navegador hace cosas raras y no lo que vos esperabas ver. También el editor de código HTML que utilizas te avisará que hay algo incorrecto.

**Comentarios HTML:**

Las etiquetas de comentario son usadas para insertar comentarios en el código HTML.

Ejemplo:



Observemos que el signo de exclamación de la etiqueta de apertura no está presente en el de cierre. Los comentarios no son mostrados en el navegador web, sirven de ayuda para documentar el código fuente dentro del mismo documento HTML.

**Atributos**

Hemos visto algunas etiquetas HTML y su uso, como por ejemplo las etiquetas de encabezado  **<h1>, <h2>,**  etiqueta de párrafo  **<p>**  y otras etiquetas. Las etiquetas HTML también pueden tener atributos, que son bits adicionales de información.

Un atributo se utiliza para definir las características de un elemento HTML y se coloca dentro de la etiqueta de apertura del elemento. Todos los atributos se componen de dos partes: un **nombre** y un **valor**

* El  **nombre**  es la propiedad que desea establecer. Por ejemplo, el elemento de párrafo  **<p>**  en el ejemplo lleva un atributo cuyo nombre es **align**, que puede usar para indicar la alineación del párrafo en la página.
* El  **valor**  es lo que desea que se establezca como valor de la propiedad y siempre entre comillas. El siguiente ejemplo muestra tres valores posibles del atributo de alineación:  **left, center**  y  **right**.

Los nombres y valores de los atributos no se distinguen entre mayúsculas y minúsculas. Sin embargo, el Consorcio World Wide Web (W3C) recomienda atributos / valores de atributo en minúsculas en su recomendación HTML 4.



¿Dónde escribo el HTML?

Antes de continuar con el aprendizaje de HTML, vamos a hablar un poco de **los editores de código** .

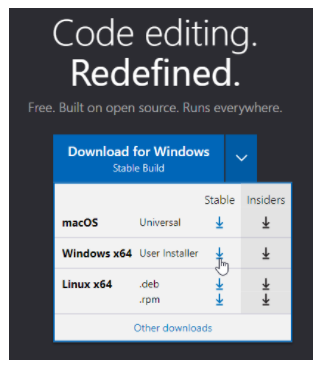
Los editores de código son programas muy sencillos que requieren poco espacio de almacenamiento y no mucho rendimiento del sistema. El dispositivo y el sistema operativo donde se utiliza básicamente depende de los gustos y necesidades del usuario. Al igual que otros tipos de software, estos programas se clasifican en software propietario, de código abierto (open source) y freeware (libre de pago), esto podría condicionar nuestra elección por uno u otro editor de código.

Existen muchos editores de código, pero aquí sólo nombraremos algunos, como por ejemplo, Visual Studio Code, Sublime Text, Notepad + + y Atom, entre otros.

Además del editor que puedes descargar y utilizar desde tu computadora, también existen editores en línea. Te vamos a dejar un editor HTML online para que puedas probarlo sin salir del curso y hagas tus experimentos, pero siempre te vamos a recomendar que lo hagas también en tu computadora con el editor que uses. Te dejamos una imágenes con algunos tips para que lo conozcas, puedes obtener más información en  [este link](https://stackblitz.com/) . 

Nosotros nos centraremos y utilizaremos para los ejemplos del editor de código a Visual Studio Code. Este editor de código es gratuito, estable, open source, robusto y posee una buena velocidad de trabajo. Estas son algunas de las principales características de Visual Studio Code que es desarrollado y mantenido por Microsoft. A continuación veremos brevemente cómo instalarlo:

1.- Desde la [página oficial de Visual Studio Code](https://code.visualstudio.com/) , descargaremos el archivo para la instalación según el sistema operativo que utilicemos. En este caso mostraremos cómo hacerlo en Windows.

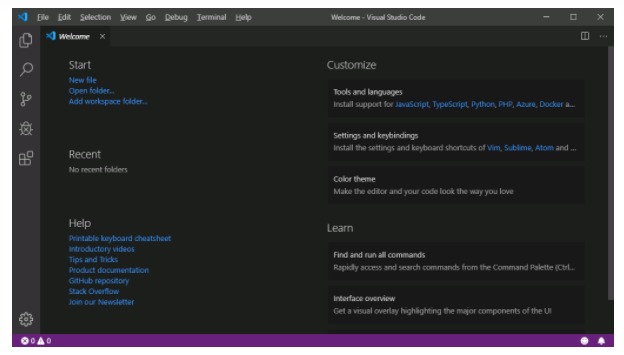


2.- Una vez descargado el archivo, hacemos clic en la barra inferior a la izquierda, donde se descargó el archivo .exe y ejecutamos el mismo.

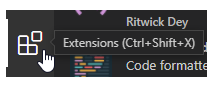


3.- Se abrirá un cuadro de diálogo y procederemos a la instalación hasta llegar a la finalización de la misma.

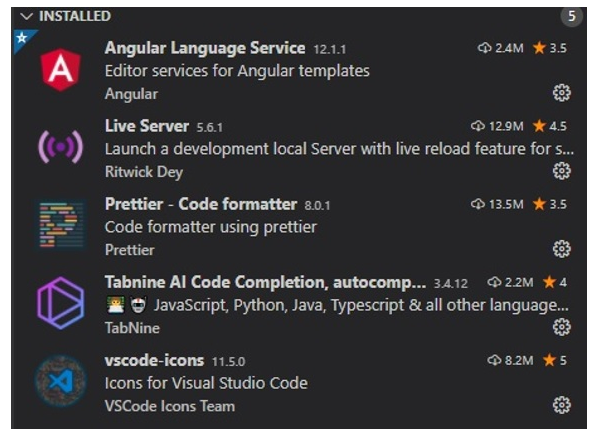
4.- Luego, una vez instalado procederemos a abrir la aplicación:



Podemos hacer clic en extensiones del menú lateral izquierdo, para agregar las extensiones que sean de nuestro agrado y nos ayuden como complemento a la hora de escribir o editar código.



Por ejemplo, podremos instalar las siguientes extensiones:



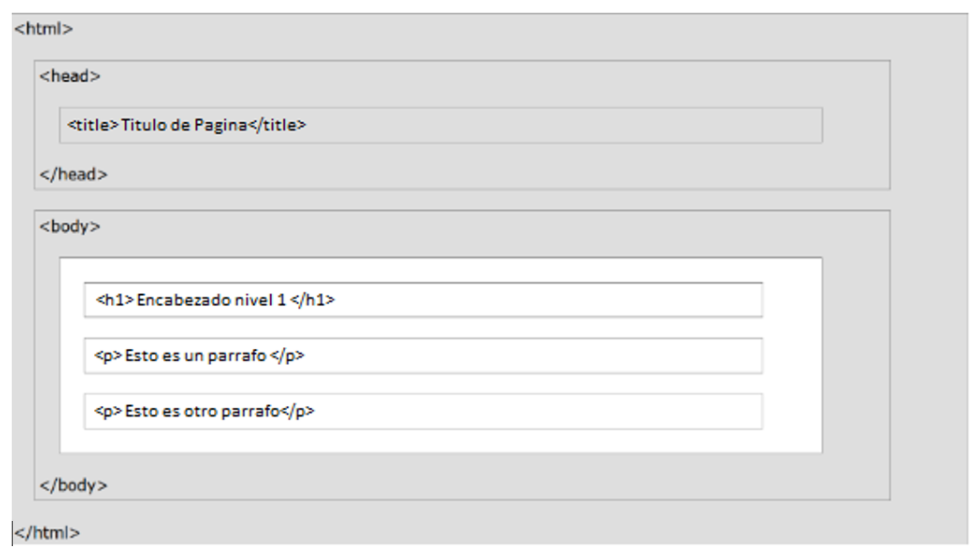
Haciendo [clic aquí](https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/editor-de-codigo/) podrás obtener más información de éste y otros editores de código.

Ahora, una vez instalado el editor de código, continuamos con el desarrollo del aprendizaje de HTML.

**Estructura documento HTML**

**Estructura básica de una página HTML**

La siguiente figura es una visualización abstracta de la estructura de una página HTML. En ella podemos observar lo que representa cada etiqueta y el contenido de texto que mostraría cada elemento. También podemos visualizar mejor la analogía con la estructura de cajas de cartón. Podríamos decir con exactitud que la caja <html> contiene en su totalidad al resto de las cajas.



Te dejamos una  [Referencia de Elementos HTML - HTML: Lenguaje de etiquetas de hipertexto | MDN (mozilla.org)](https://developer.mozilla.org/es/docs/Web/HTML/Element)  para que puedas explorar.

Ahora veamos esa estructura escrita en el código:



Observamos:

* **<! DOCTYPE html>**  - el tipo de documento. Es un preámbulo requerido. Anteriormente, cuando HTML era joven (cerca de 1991/2), los tipos de documentos actuaban como vínculos a un conjunto de reglas que el código HTML de la página debía seguir para ser considerado bueno, lo que podía significar la verificación automática de errores y algunas otras cosas de utilidad. Sin embargo, hoy día es simplemente un artefacto antiguo que a nadie le importa, pero que debe ser incluido para que todo funcione correctamente.
* **<html> </ html** > - el elemento <html>. Este elemento encierra todo el contenido de la página entera y, a veces, se conoce como el elemento raíz.
* **<head> </head>**  - el elemento <head>. Este elemento actúa como un contenedor de todo aquello que desea incluir en la página HTML que no es contenido visible por los visitantes de la página. Incluye cosas como palabras clave (keywords), una descripción de la página que quieres que aparezca en resultados de búsquedas, código CSS para dar estilo al contenido, declaraciones del juego de caracteres, etc.
* **<meta charset = "utf-8">**  - <meta>. Este elemento establece el juego de caracteres que tu documento usará en utf-8, que incluye casi todos los caracteres de todos los idiomas humanos. Básicamente, puede manejar cualquier contenido de texto que pueda incluir. No hay razón para no establecerlo, y puede evitar problemas en el futuro.
* **<title> </ title >** - el elemento <title> establece el título de tu página, que es el título que aparece en la pestaña o en la barra de título del navegador cuando la página es cargada, y se usa para describir la página cuando es añadida a los marcadores o como favorita.
* **<body> </body>**  - el elemento <body>. Encierra todo el contenido que deseas mostrar a los usuarios web que visitan tu página, ya sea texto, imágenes, videos, juegos, pistas de audio reproducibles, y demás.
* **<h1>,…, <h6>**  Los elementos de encabezado implementan seis niveles de encabezado del documento, <h1> es el más importante, y <h6>, el menos importante. Un elemento de encabezado describe brevemente el tema de la sección que presenta. La información de encabezado puede ser usada por los agentes usuarios, por ejemplo, para construir una tabla de contenidos para un documento automáticamente. Sus etiquetas son <h1>,…, <h6> y </h1>,…, </h6>.
* **<p>**  El elemento <p> (párrafo) es apropiado para distribuir el texto en párrafos. Sus etiquetas son <p> y </p>.
* **<b>** El elemento HTML <b> indica que el texto debe ser representado con una variable bold, o negrita, de la tipografía que se esté usando. Sin darle al texto importancia adicional. Sus etiquetas son <b> y </b>.
* **<strong>**   El elemento <strong> destaca el texto. Sus etiquetas son <strong> y </strong>. El elemento <strong> le da al texto más énfasis que el elemento <b>, con una importancia más alta semánticamente.
* **<i>**  El elemento HTML <i> muestra el texto marcado con un estilo en cursiva o itálica. Sus etiquetas son <i> e </i>.
* **<em>**   El elemento HTML <em> es apropiado para marcar con énfasis en el texto. El elemento <em> puede ser anidado, con cada nivel de anidamiento indicando un mayor grado de énfasis. Sus etiquetas son <em> y </em>.
* **<br>**  El elemento HTML <br> produce un salto de línea en el texto (retorno de carro). Es útil para escribir un poema o una dirección, donde la división de las líneas es significante.  No lo utilices para incrementar el espacio entre líneas de texto; para ello utiliza la propiedad margin de CSS o el elemento <p>.
* **<li>**  El elemento HTML <li> o elemento de lista declara cada uno de los elementos de una lista. Sus etiquetas son <li> e </li>.
* **<ol>**  El elemento <ol> permite definir listas o viñetas ordenadas con numeración o alfabéticamente. Sus etiquetas son <ol> y </ol>.
* **<ul>**  El elemento HTML <ul> de "lista desordenada" - lista no ordenada. crea una lista no ordenada. Sus etiquetas son <ul> y </ul>.
* **<div>**  El elemento HTML <div> es exclusivamente usado como contenedor para otros elementos HTML. En conjunto con CSS, el elemento <div> puede ser usado para agregar formato a un bloque de contenido. Sus etiquetas son <div> y </div>.
* **<img>**  El elemento HTML <img> posee los atributos src y alt pero no tiene etiqueta de cierre. Se puede representar así <img src = "imagen.png" alt = "Mi descripción de imagen"> Un elemento <img> no encierra contenido. También a este tipo de elemento se lo conoce como elemento vacío. El propósito del elemento <img> es desplegar una imagen en la página web, en el lugar que corresponde según la estructura del documento.
* El nombre de archivo de la imagen de origen está especificado por el atributo src. Los navegadores web no siempre muestran la imagen a la que el elemento hace referencia. Es el caso de los navegadores no gráficos (incluidos aquellos usados ​​por personas con problemas de visión), si el usuario elige no mostrar la imagen, o si el navegador es incapaz de mostrarla porque no es válida o soportada. En ese caso, el navegador la reemplazará con el texto definido en el atributo alt
* **<a>** El Elemento HTML Anchor <a> crea un enlace a otras páginas de Internet, archivos o ubicaciones dentro de la misma página, direcciones de correo, o cualquier otra URL que especifiquemos en sus atributos. Se puede representar así <a href="https://developer.mozilla.org/es/docs/Web/HTML/Element/a"> </a> donde la dirección del enlace está especificada por el atributo href.
* Dentro del atributo href la URL puede escribirse de forma absoluta (incluyendo el dominio) o relativa (sin incluir el dominio) solo para enlaces dentro del mismo dominio. Tanto de una forma u otra, la ruta de carpetas debe especificarse.

Siguiendo con la descripción del atributo href del elemento <a>, podemos dividir los enlaces o links en [3 tipos](https://3con14.biz/html/enlaces/13-tipos-de-enlaces.html):

1. Enlaces internos: son los que se dan entre páginas web del mismo dominio.
2. Enlaces externos: son los que se dan entre páginas web de distinto dominio.
3. Enlaces de posición (o marcadores):

A. De un lugar a otro dentro de la misma página.

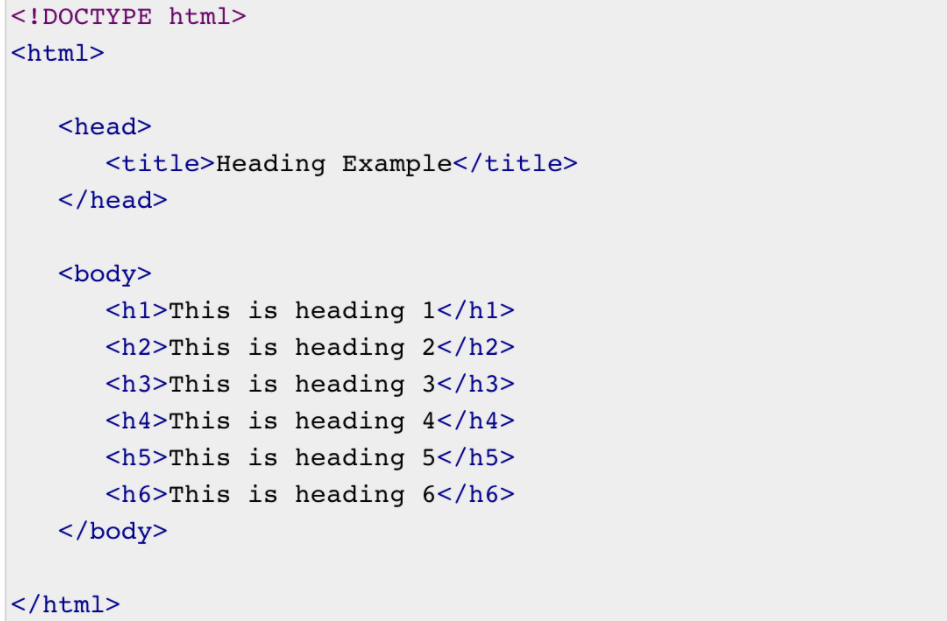
B. De un lugar a otro lugar concreto de otra página del mismo dominio.

C. De un lugar a otro lugar concreto de una página de otro dominio.

**Encabezados**

**Etiquetas de encabezado**

Cualquier documento comienza con un encabezado. Puede utilizar diferentes tamaños para sus títulos. HTML también tiene seis niveles de encabezados, que utilizan los elementos <h1>, <h2>, <h3>, <h4>, <h5> y <h6>. Mientras muestra cualquier encabezado, el navegador agrega una línea antes y una línea después de ese encabezado.



**Hagamos algunas prácticas**

Te dejamos un [IDE](https://www.redhat.com/es/topics/middleware/what-is-ide#:~:text=Un%20entorno%20de%20desarrollo%20integrado,gr%C3%A1fica%20de%20usuario%20(GUI).) con la estructura básica de una pagina HTML para que puedas escribir el código siguiendo las instrucciones de abajo:

1. Escribir las 6 líneas de código que se muestra en la imagen de arriba con las distintas etiquetas de encabezado para experimentar cómo se ve.
2. Te dejamos un texto sin formato y deberás darle formato agregando las etiquetas de encabezado.
   1. Descarga [este código HTML](https://argentinaprograma.inti.gob.ar/pluginfile.php/49694/mod_page/content/15/Etiquetas_Encabezado-CodigoPorCompletar.txt?time=1630728872464) , este texto deberás pegarlo dentro de la etiqueta <body> (cuerpo) del archivo  **index.html**  para darle formato al texto, agregando las etiquetas de encabezado, deberás dejarlo igual a  [este documento pdf](https://argentinaprograma.inti.gob.ar/pluginfile.php/49694/mod_page/content/15/Etiqueta%20encabezado%20-%20resultado%20esperado.pdf) .

Si no recuerdas algo búscalo en la web, repasa los temas anteriores, escribí una pregunta en el foro del módulo.

**Formato**

Si usas un procesador de texto, debes estar familiarizado con la capacidad de hacer que el texto esté en **negrita**, *cursiva* o subrayado; estas son solo tres de las diez opciones disponibles para indicar cómo puede aparecer el texto en HTML y XHTML:

**Texto en negrita**

Todo lo que aparece dentro del elemento  **<b> ... </b>**  se muestra en negrita

*Texto en cursiva*

Todo lo que aparece dentro del elemento  **<i> ... </i>**  se muestra en cursiva

**Texto subrayado**

Todo lo que aparece dentro del elemento  **<u> ... </u>**  se muestra subrayado

Todo lo que aparece dentro del elemento  **<strike> ... </strike>**  se muestra tachado

Fuente monoespaciada

El contenido de un elemento  **<tt> ... </tt>**  está escrito en fuente monoespaciada. La mayoría de las fuentes se conocen como fuentes de ancho variable porque diferentes letras tienen diferentes anchos (por ejemplo, la letra 'm' es más ancha que la letra 'i'). Sin embargo, en una fuente monoespaciada, cada letra tiene el mismo ancho.

Existen muchos otros tipos de fuentes, te invitamos a profundizar más sobre ello en [este link](https://www.w3schools.com/html/html_formatting.asp).

**Meta Tags**

HTML te permite especificar metadatos: información adicional importante sobre un documento de diversas formas. Los elementos META se pueden utilizar para incluir pares de nombre - valor que describen las propiedades del documento HTML, como el autor, la fecha de caducidad, una lista de palabras clave, el autor del documento, etc.

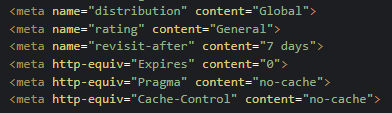
La etiqueta  **<meta>**  se utiliza para proporcionar dicha información adicional. Esta etiqueta es un elemento vacío y, por lo tanto, no tiene una etiqueta de cierre, pero lleva información dentro de sus atributos.

Puede incluir una o más meta etiquetas en su documento en función de la información que desee mantener, pero en general, las meta etiquetas no indican la apariencia física del documento, por lo que desde el punto de vista de la apariencia, no importa si se incluye o no.

Agregar meta etiquetas a sus documentos

Puedes agregar metadatos a sus páginas web colocando etiquetas <meta> dentro del encabezado del documento que está representado por las etiquetas  **<head>**  y  **</head>**. Una meta etiqueta puede tener los siguientes atributos además de los atributos principales:





**listas, imágenes y enlaces HTML**

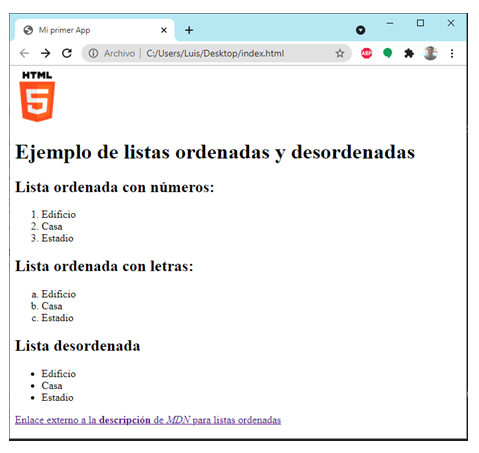
A continuación desarrollamos un ejemplo en donde utilizamos los elementos <ol>, <ul>, <img> y <a>.

En el caso del elemento <ol> si le asignamos el atributo type = “a” le indicamos al explorador web que vamos a realizar una lista ordenada alfabéticamente en minúscula. Vale aclarar que lo que se muestra ordenado es el prefijo que se asigna al ítem y no los propios nombres de cada ítem. Si no indicamos el atributo type, por defecto, la lista sale ordenada numéricamente.

En el caso del elemento <img>, se utilizó como referencia, una imagen del sitio <https://3con14.biz/html/> y se le aplicaron atributos de ancho y alto para reducir su tamaño relativo a la pantalla (los explicaremos con más profundidad en el módulo de CSS).

Con el elemento <a> insertamos un enlace a una página externa, en este caso, de [Mozilla Developer Network](https://developer.mozilla.org/es/docs/Web/HTML/Element/ol)





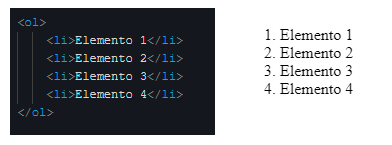
Las listas en HTML son muy importantes, no solo se utilizan para listar, sino también se utilizan para el menú, cuando ingresas en una web y tiene un menú desplegable, seguramente usaron listas que tienen un estilo y comportamiento específico, pero en el fondo son listas.

En HTML tenemos 3 tipos de listas a utilizar para diferentes usos:

1. Listas Ordenadas.
2. Listas Desordenadas.
3. Lista de Definición.

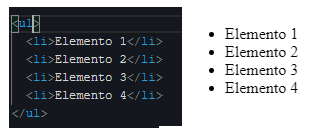
**Listas Ordenadas**

Las listas ordenadas son listas en las que el orden de los elementos **SI importa** y se indican con la etiqueta <ol> </ol>, dentro se le anida otra etiqueta <li> </li> quedando de la siguiente manera:



**Listas Desordenadas**

Las listas desordenadas son listas en las que el orden de los elementos **NO importa** y se indican con la etiqueta <ul> </ul>, dentro, se le anida otra etiqueta <li> </li> quedando de la siguiente manera:



¿Recordarás que las etiquetas pueden tener Atributos?

En las listas ordenadas **existen varios Atributos**(por ejemplo: start, type, reverse, etc) que se aplican al elemento <ol> </ol>. Investiga qué se puede hacer con esos Atributos, algunas aplican solo para las listas ordenadas y otras para listas desordenadas. Usa tus habilidades de búsqueda ya que en los ejercicios siguientes lo vas a necesitar.

**Listas de Definiciones**

Estas listas de definiciones no son tan comunes, por lo que te dejaremos este [documento PDF](https://argentinaprograma.inti.gob.ar/pluginfile.php/49697/mod_page/content/12/Practica_ListasDefiniciones_ResultadoEsperado.pdf) para que observes cómo se ven estos tipos de lista y te invitamos a que busques en internet cuáles son las etiquetas HTML que se utilizan para crear esta lista.

**Práctica** :

En el IDE de abajo, te dejamos un código HTML para que puedas completar, tienes que agregarle las etiquetas de la **lista que corresponda** para que quede igualito a [este documento PDF](https://argentinaprograma.inti.gob.ar/pluginfile.php/49697/mod_page/content/12/Practica_ListasOrdenadas_ResultadoEsperado.pdf?time=1635946973745) , para eso debes hacer lo siguiente:

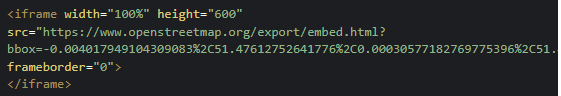
1. Descarga el documento PDF.
2. Usando el IDE de tu computadora o el que te dejamos debajo, tienes que agregar  las etiquetas y atributos necesarios en el código HTML para que se pueda visualizar igual al documento PDF.

Te dejamos el IDE, y ya sabes que si necesitas ayuda nos vemos en el foro.

**Iframe**

Puedes definir un marco en línea con la etiqueta HTML  **<iframe>**. La etiqueta <iframe> no está relacionada de alguna manera con la etiqueta <frameset>, sino que puede aparecer en cualquier parte de su documento. La etiqueta <iframe> define una región rectangular dentro del documento en la que el navegador puede mostrar un documento separado, incluidas las barras de desplazamiento y los bordes. Un marco en línea se utiliza para incrustar otro documento dentro del documento HTML actual.

El atributo  **src**  se utiliza para especificar la URL del documento que ocupa el marco en línea:



**Fuentes**

Las fuentes juegan un papel muy importante para hacer que un sitio web sea más fácil de usar y aumentar la legibilidad del contenido. La cara y el color de la fuente depende completamente de la computadora y el navegador que estás utilizando para ver tu página, pero puedes usar la etiqueta HTML  **<font>**  para agregar estilo, tamaño y color al texto de su sitio web. Puedes usar una etiqueta  **<basefont>**  para configurar todo tu texto en el mismo tamaño, cara y color.

La etiqueta de **font** tiene tres atributos llamados **size, color** y **face**para personalizar tus fuentes. Para cambiar cualquiera de los atributos de fuente en cualquier momento dentro de tu página web, simplemente usa la etiqueta <font>. El texto que sigue permanecerá cambiado hasta que cierres con la etiqueta </font>. Puedes cambiar uno o todos los atributos de fuente dentro de una etiqueta <font>.

**Nota:** La  *fuente* y *BASEFONT* son etiquetas que están en desuso y se supone que será eliminado en una futura versión de HTML. Por lo tanto, no deben usarse, en su lugar se sugiere usar estilos CSS para manipular tus fuentes. Pero aún con el propósito de aprender, este capítulo explicará en detalle las etiquetas de fuente y fuente base.

Establecer tamaño de fuente

Puedes establecer el tamaño de la fuente del contenido mediante el atributo de **size**. El rango de valores aceptados es de 1 (menor) a 7 (mayor). El tamaño predeterminado de una fuente es 3:



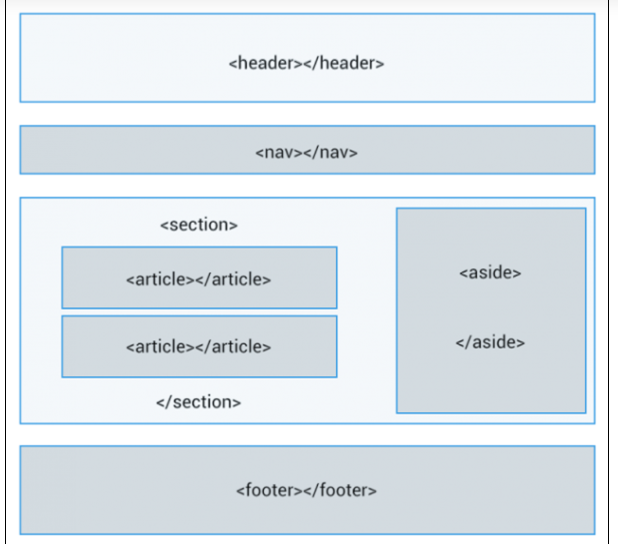
**HTML5 lo nuevo**

Hasta ahora hemos leído y hemos visto ejemplos sobre HTML básico, pero, en la actualidad existe en plena vigencia la versión HTML 5.

Es decir, HTML5 es la versión más reciente de HTML homologada por la W3C.

Una de las principales ventajas de HTML5 es la inclusión de [elementos semánticos](https://www.eniun.com/html5-estructura-basica-elementos-semanticos/), o marcadores semánticos, que nos ayudan a definir las distintas divisiones de una página web. Es decir, los elementos semánticos de HTML5, nos ayudan a identificar cada sección del documento y organizar el cuerpo de una página web de una manera eficiente y estandarizada.

En la figura siguiente, observamos el esquema básico de los elementos semánticos introducidos por la última y por la versión actual de HTML5:



**Según el W3C una Web Semántica:**

"*Permite que los datos sean compartidos y reutilizados en todas las aplicaciones, las empresas y las comunidades*."

**¿Por qué utilizar elementos semánticos?**

En HTML4 los desarrolladores usaban sus propios nombres como identificación de los elementos, tales como: cabecera, superior, inferior, pie de página, menú, navegación, principal, contenedor, contenido, artículo, barra lateral, etc. , se hacía imposible que los motores de búsqueda identificaran el contenido correcto de la página web.

**Elementos semánticos = elementos con significado.**

Un [elemento semántico](https://www.w3schools.com/html/html5_semantic_elements.asp) describe claramente su significado tanto para el navegador web como para el desarrollador.

* Ejemplos de elementos no semánticos: <div> y <span> - No dice nada sobre su contenido.
* Ejemplos de elementos semánticos: <form>, <table> y <article> - Definen claramente su contenido.

**Descripción de elementos semánticos HTML5**

**Elemento HTML <section>**

El  [elemento](https://www.w3schools.com/html/html5_semantic_elements.asp) <section> define una sección en un documento. Según la documentación HTML del W3C: " *Una sección es una agrupación temática de contenido, normalmente con un encabezado* ".

Una página web normalmente se puede dividir en secciones para la introducción, el contenido y la información de contacto.

**Elemento HTML <article>**

El elemento <article> especifica contenido autónomo e independiente. Un artículo debe tener sentido por sí solo y debe ser posible distribuirlo independientemente del resto del sitio web.

Ejemplos de dónde se puede utilizar un elemento <article>:

* Publicación en el foro
* Entrada en el blog
* Artículo de periódico

**Elemento HTML <header>**

El elemento <header> sirve para especificar contenido de tipo introductorio o un conjunto de enlaces de navegación.

Un elemento <header> normalmente contiene:

1. Uno o más elementos de encabezado (<h1> - <h6>).
2. Logo o icono.
3. Información de autoría.

**Nota:** puede tener varios elementos <header> en un documento HTML. Sin embargo, el elemento <header> no puede ser colocado dentro de elementos <footer>, <address> o dentro de otro elemento <header>.

**Elemento HTML <footer>**

El elemento <footer> define un pie de página para un documento o sección.

Un elemento <footer> normalmente contiene:

1. Información de autoría.
2. Información registrada.
3. Información de contacto.
4. Mapa del sitio.
5. Volver a los enlaces superiores.
6. Documentos relacionados.

Puede tener varios elementos <footer> en un documento.

**Elemento HTML <nav>**

El elemento HTML [<nav>](https://developer.mozilla.org/es/docs/Web/HTML/Element/nav) representa una sección de una página cuyo propósito es proporcionar enlaces de navegación, ya sea dentro del documento actual o en otros documentos. Ejemplos comunes de secciones de navegación son: menús, tablas de contenido e índices.

Debemos tener en cuenta que NO todos los enlaces de un documento deben estar dentro de un elemento <nav>. El elemento <nav> está destinado sólo para el bloque principal de enlaces de navegación.

Los navegadores, como los lectores de pantalla para usuarios con capacidades diferentes, pueden utilizar este elemento para determinar si se debe omitir la representación inicial de este contenido.

**Elemento HTML <aside>**

El elemento HTML [<aside>](https://developer.mozilla.org/es/docs/Web/HTML/Element/aside) representa una sección de una página que consiste en contenido que está indirectamente relacionado con el contenido principal del documento. Estas secciones son a menudo representadas como barras laterales o como inserciones y contienen una explicación al margen como una definición de glosario, elementos relacionados indirectamente, como publicidad, la biografía del autor, o en aplicaciones web, la información de perfil o enlaces a blogs relacionados .

**Colores**

Los colores son muy importantes para darle una buena apariencia a tu sitio web. Puedes especificar colores a nivel de página usando la etiqueta <body> o puedes establecer colores para etiquetas individuales usando el atributo **bgcolor**.

La etiqueta <body> tiene los siguientes atributos que se pueden usar para establecer diferentes colores:

* **bgcolor**: establece un color para el fondo de la página.
* **text**: establece un color para el cuerpo del texto.
* **alink**: establece un color para los enlaces activos o los enlaces seleccionados.
* **link**: establece un color para el texto vinculado.
* **vlink**: establece un color para los enlaces visitados, es decir, para el texto vinculado en el que ya ha hecho clic.

Métodos de codificación de colores HTML

Existen tres métodos diferentes para configurar los colores en tu página web:

* **Nombres de colores**: puede especificar nombres de colores directamente como **green**, **blue** o**red.**
* **Códigos hexadecimales**: un código de seis dígitos que representa la cantidad de rojo , verde y azul que compone el color.
* **Valores decimales o porcentuales de color**: este valor se especifica mediante la propiedad rgb ().

Ahora veremos estos esquemas de coloración uno por uno.

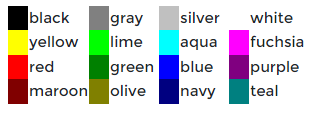
Colores HTML - Nombres de colores

Puedes especificar un nombre de color directo para establecer el texto o el color de fondo. W3C ha enumerado 16 nombres de colores básicos que se validarán con un validador HTML, pero hay más de 200 nombres de colores diferentes compatibles con los principales navegadores.

**Nota**: te dejamos para que consultes una lista completa de nombres de [colores HTML.](https://www.tutorialspoint.com/html/html_color_names.htm)

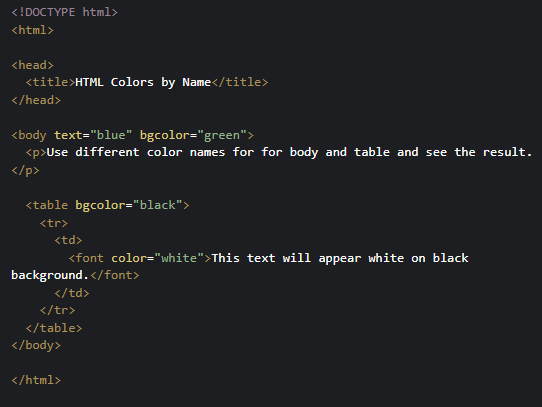
Estándar W3C 16 colores

Aquí está la lista de nombres de 16 colores estándar del W3C y se recomienda usarlos.



Ejemplo

Estos son los ejemplos para establecer el fondo de una etiqueta HTML por nombre de color:



**Introducción CSS**

El gran impulso de los lenguajes de hojas de estilos se produjo con el boom de Internet y el crecimiento exponencial del lenguaje HTML. Entre la fuerte competencia de navegadores web y la falta de un estándar para la definición de los estilos, se dificultaba la creación de documentos con la misma apariencia en diferentes navegadores.

A mediados de la década de 1990 el W3C, encargado de crear todos los estándares relacionados con la web, propuso la creación de un lenguaje de hojas de estilos específicos para el lenguaje HTML y se presentaron propuestas. Las dos propuestas que se tuvieron en cuenta fueron la CHSS (Cascading HTML Style Sheets) y la SSP (Stream-based Style Sheet Proposal). La propuesta CHSS fue realizada por [Håkon Wium Lie](https://en.wikipedia.org/wiki/H%C3%A5kon_Wium_Lie" \t "_blank) y SSP fue propuesto por [Bert Bos](https://es.wikipedia.org/wiki/Bert_Bos) . Entre finales de 1994 y 1995 Lie y Bos se unieron para definir un nuevo lenguaje que tomaba lo mejor de cada propuesta y lo llamaron [CSS](https://developer.mozilla.org/es/docs/Web/CSS) , Cascading Style Sheets.

En 1995, el W3C decidió gestionar el desarrollo y estandarización de CSS y agregó el proyecto a su grupo de trabajo de HTML. A finales de 1996, el W3C publicó la primera recomendación oficial, conocida como " [CSS nivel 1](https://www.ecured.cu/CSS) ".

**¿Qué es CSS?.**  
Css es un leguaje que trabaja junto con HTML para proporcionar estilos visuales a los elementos de un documento web.

**Características:**

* .- Ahorra trabajo. Se puede controlar el diseño de varias páginas HTML a la vez.
* .- Se pueden almacenar en archivos \* .css

CSS 3 es la última versión estándar.

¿Para qué utilizar CSS?

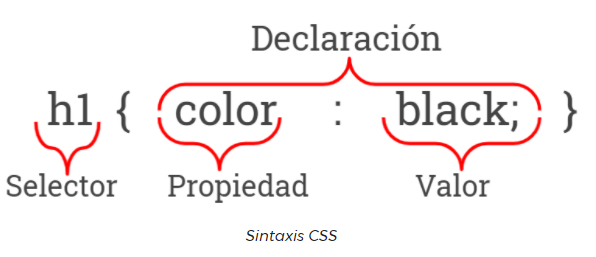
Para definir estilos en los documentos web, incluyendo el diseño, la disposición de los elementos y para responder a las variaciones en la pantalla en cuanto a diferentes dispositivos y tamaños de pantalla.

Ventajas

1. Control centralizado de la presentación de un sitio web completo con lo que se agiliza considerablemente la actualización y mantenimiento.
2. Los Navegadores permiten a los usuarios especificar su propia hoja de estilo local que será aplicada a un sitio web, con lo que aumenta considerablemente la accesibilidad. Por ejemplo, las personas con deficiencias visuales pueden configurar su propia hoja de estilo para aumentar el tamaño del texto o remarcar más los enlaces.
3. Una página puede disponer de diferentes hojas de estilo según el dispositivo que la muestra o incluso una elección del usuario. Por ejemplo, para ser impresa, mostrada en un dispositivo móvil, o ser "leída" por un sintetizador de voz.
4. El documento HTML en sí mismo es más claro de entender y se consigue reducir considerablemente su tamaño (siempre y cuando no se utiliza estilo en línea).

**Reglas CSS**

CSS define un conjunto de reglas que permiten describir cada una de las partes que componen los estilos CSS.



**Selector:** indica el elemento o elementos HTML a los que se aplica la regla CSS.

**Declaración:** especifica los estilos que se aplican a los elementos.

**Propiedad:** permite modificar el aspecto de una característica del elemento.

**Valor:** indica el nuevo valor de la característica modificada en el elemento.

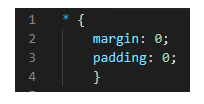
Selectores

Nos indican qué elemento HTML hay que aplicar el estilo. Una misma regla puede aplicar a varios selectores y, a un mismo selector se le pueden aplicar varias reglas.

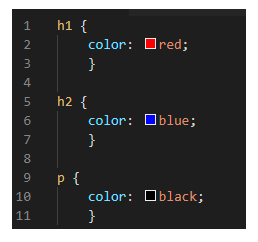
Para mayor información referirse a [este enlace](https://www.w3.org/wiki/CSS_/_Selectores_CSS#Lista_de_selectores_css) .

Selectores básicos

**Selector universal** : se utiliza para seleccionar todos los elementos de la página:



**Selector de tipo o etiqueta:** selecciona todos los elementos de la página cuya etiqueta HTML coincide con el valor del selector:

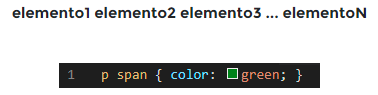


Selector descendente

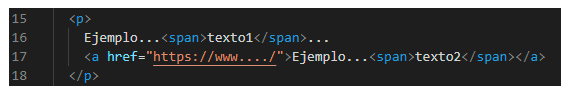
Selecciona los elementos que se encuentran dentro de otros elementos. Un elemento es descendente de otro cuando se encuentra entre las etiquetas de apertura y de cierre de otro elemento.



La sintaxis formal del selector descendente se muestra a continuación:



Si el código HTML de la página es el siguiente:



El selector p span selecciona tanto texto1 como texto2. El motivo es que en el selector descendente, un elemento no tiene que ser descendiente directo del otro. La única condición es que un elemento debe estar dentro de otro elemento, sin importar el nivel de profundidad en el que se encuentre.

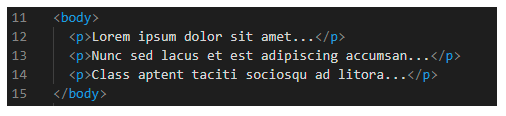
No se les aplica la regla CSS anterior si el resto de los elementos <span> de la página no están dentro de un elemento <p>.

Los selectores descendentes nos permiten aumentar la precisión del selector de tipo o etiqueta. Así, utilizando el selector descendente podemos aplicar diferentes estilos a los elementos del mismo tipo.

Para conocer más te invitamos hacer clic en el este [enlace](https://uniwebsidad.com/libros/css/capitulo-2/selectores-basicos).

Selector de clase o selector Class

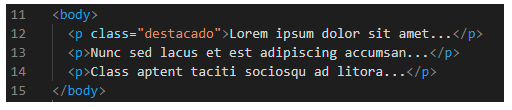
Si consideramos el siguiente código HTML de ejemplo:



**¿Cómo podemos aplicar estilos CSS sólo al primer párrafo?**

El selector universal (\*) no se puede utilizar porque seleccionaría todos los elementos de la página. El selector de tipo o etiqueta (p) tampoco se puede utilizar porque seleccionaría todos los párrafos. Por último, el selector descendente (body p) tampoco se puede utilizar porque todos los párrafos se encuentran en el mismo sitio.

Una de las soluciones más sencillas para aplicar estilos a un solo elemento de la página consiste en utilizar el atributo class de HTML sobre ese elemento para indicar directamente la regla CSS que se le debe aplicar:

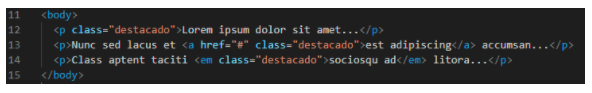


A continuación, creamos en el archivo CSS una nueva regla llamada destacado con todos los estilos que vamos a aplicar al elemento. Para que el navegador no confunda este selector con los otros tipos de selectores, prefijamos el valor del atributo class con un punto (.) Tal como vemos en la siguiente línea:



El selector destacado se interpreta como "cualquier elemento de la página cuyo atributo class sea igual a destacado", por lo que solamente el primer párrafo del anterior ejemplo cumple esa condición.

Este tipo de selectores de clase son los más utilizados junto con los selectores de ID que veremos a continuación. La principal característica de este selector es que en una misma página HTML varios elementos diferentes pueden utilizar el mismo valor en el atributo class:

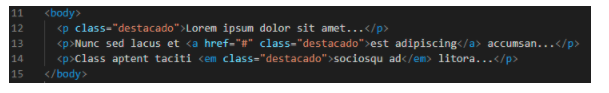


Los selectores de clase resultan imprescindibles para diseñar páginas web complejas, ya que nos permiten disponer de una precisión total al seleccionar los elementos. Además, estos selectores nos permiten reutilizar los mismos estilos para varios elementos diferentes:



El elemento <span> tiene un atributo class = "error", por lo que se le aplica las reglas CSS indicadas por el selector .error. Por su parte, el elemento <div> tiene un atributo class = "aviso", por lo que su estilo es el que define las reglas CSS del selector .aviso.

En ciertos casos, es necesario restringir el alcance del selector de clase. Si utilizamos el ejemplo anterior:



**¿Cómo podemos aplicar estilos solamente al párrafo cuyo atributo class sea igual a destacado?**

Combinando el selector de tipo y el selector de clase, se obtiene un selector mucho más específico:



Interpretamos al selector p.destacado como "aquellos elementos de tipo <p> que dispongan de un atributo class con valor destacado". De la misma forma, el selector a.destacado solamente seleccionaría los enlaces cuyo atributo class sea igual a destacado.

De lo anterior deducimos que el atributo .destacado es equivalente a \* .destacado, por lo que todos los diseñadores obvian el símbolo \* al escribir un selector de clase normal.

No debemos confundir el selector de clase con los selectores anteriores:

/ \* Todos los elementos de tipo "p" con atributo class = "aviso" \* /

p.aviso {...}

/ \* Todos los elementos con atributo class = "aviso" que estén dentro

   de cualquier elemento de tipo "p" \* /

p .aviso {...} / \* notar el espacio entre py la clase \* /

/ \* Todos los elementos "p" de la página y todos los elementos con

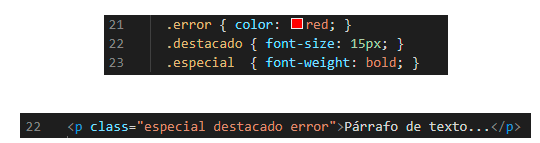
   atributo class = "aviso" de la página \* /

p, .aviso {...} / \* notar la, (coma) entre py la clase \* /

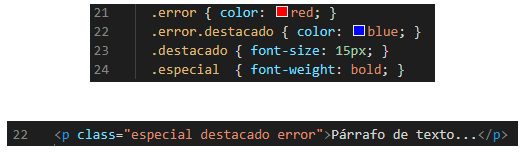
Para completar, es posible aplicar los estilos de varias clases CSS sobre un mismo elemento. La sintaxis es similar, pero los diferentes valores del atributo class se separan con espacios en blanco.



Al párrafo anterior le aplicamos los estilos definidos en las reglas .especial, .destacado y .error, por lo que en el siguiente ejemplo, el texto del párrafo se vería de color rojo, en negrita y con un tamaño de letra de 15 píxel:



Si un elemento dispone de un atributo class con más de un valor, es posible utilizar un selector más avanzado:



En el ejemplo anterior, el color de la letra del texto es azul y no rojo. Esto se debe a que hemos utilizado un selector de clase múltiple .error.destacado, que se interpreta como "aquellos elementos de la página que están disponibles de un atributo class con al menos los valores error y destacado".

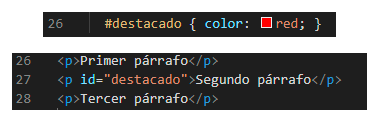
Para saber más podes hacer clic en [este enlace](https://uniwebsidad.com/libros/css/capitulo-2/selectores-basicos).

**Selectores de ID**

En ocasiones, es necesario aplicar estilos CSS a un único elemento de la página. Aunque puede aplicar un selector de clase para un único elemento, existe otro selector más eficiente en este caso.

El selector de ID permite seleccionar un elemento de la página a través del valor de su atributo id. Este tipo de selectores sólo selecciona un elemento de la página porque el valor del atributo id no se puede repetir en dos elementos diferentes de una misma página.

La sintaxis de los selectores de ID es muy parecida a la de los selectores de clase, salvo que se utiliza el símbolo de la almohadilla (#) en vez del punto (.) Como prefijo del nombre de la regla CSS:



En el ejemplo anterior, el selector #destacado solamente selecciona el segundo párrafo (cuyo atributo id es igual a destacado).

La principal diferencia entre este tipo de selector y el selector de clase tiene que ver con HTML y no con CSS. En una misma página, el valor del atributo id debe ser único, de forma que dos elementos diferentes no pueden tener el mismo valor de id. Sin embargo, el atributo class no es obligatorio que sea único, de forma que muchos elementos HTML diferentes pueden compartir el mismo valor para su atributo class.

De esta forma, la recomendación general es la de utilizar el selector de ID cuando se quiere aplicar un estilo a un solo elemento específico de la página y utilizar el selector de clase cuando queremos aplicar un estilo determinado a varios elementos diferentes de la página HTML.

Al igual que los selectores de clase, en este caso también podemos restringir el alcance del selector mediante la combinación con otros selectores. El siguiente ejemplo aplica la regla CSS solamente al elemento de tipo <p> que tenga un atributo id igual al indicado:



A primera vista, restringir el alcance de un selector de ID puede parecer absurdo. En realidad, un selector de tipo p # aviso sólo tiene sentido cuando el archivo CSS se aplica sobre muchas páginas HTML diferentes.

En este caso, algunas páginas pueden disponer de elementos con un atributo id igual a aviso y que no sean párrafos, por lo que la regla anterior no se aplica sobre esos elementos.

No debe confundirse el selector de ID con los selectores anteriores:

/ \* Todos los elementos de tipo "p" con atributo id = "aviso" \* /

p # aviso {...}

/ \* Todos los elementos con atributo id = "aviso" que estén dentro

de cualquier elemento de tipo "p" \* /

p #aviso {...}

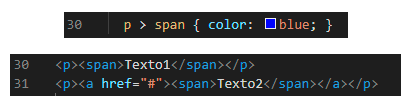
/ \* Todos los elementos "p" de la página y todos los elementos con

atributo id = "aviso" de la página \* /

p, #aviso {...}

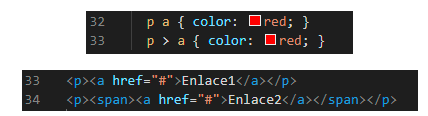
**Selectores avanzados**

Se trata de un selector similar al selector descendente, pero muy diferente en su funcionamiento. Se utiliza para seleccionar un elemento que es hijo directo de otro elemento y se indica mediante el "signo de mayor que" (>):



En el ejemplo anterior, el selector p> span se interpreta como "cualquier elemento <span> que sea hijo directo de un elemento <p>", por lo que el primer elemento <span> cumple la condición del selector. Sin embargo, el segundo elemento <span> no la cumple porque es descendiente pero no es hijo directo de un elemento <p>.

El siguiente ejemplo muestra las diferencias entre el selector descendente y el selector de hijos:

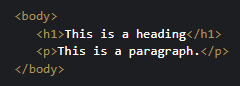


El primer selector es de tipo descendente y por tanto se aplica a todos los elementos <a> que se encuentran dentro de los elementos <p>. En este caso, los estilos de este selector aplican se a los dos enlaces.

Por otra parte, el selector de hijos obliga a que el elemento <a> sea hijo directo de un elemento <p>. Por lo tanto, los estilos del selector p> a no se aplican al segundo enlace del ejemplo anterior.

Para conocer más sobre selectores avanzados podes hacer clic en [este enlace](https://uniwebsidad.com/libros/css/capitulo-2/selectores-avanzados).

**Ejemplo Selectores**

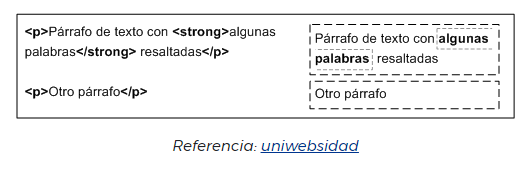


**Modelo de Cajas**

Tal como dijimos en la unidad de HTML, si hacemos una analogía con una estructura de cajas de cartón, podemos decir que hay ciertas cajas que van dentro de otras y ciertas cajas que van una al lado de otra.

El "modelo de caja" es probablemente la característica más importante del lenguaje de hojas de estilos CSS, ya que condiciona el diseño de todas las páginas web. El modelo de cajas es el comportamiento de CSS que hace que todos los elementos de las páginas sean representados mediante cajas rectangulares.

Cada vez que se inserta un elemento HTML, se crea una nueva caja rectangular que encierra los contenidos de ese elemento. La siguiente imagen muestra tres cajas rectangulares que crean los tres elementos HTML de una porción de página de ejemplo.



Los navegadores web crean y colocan las cajas de forma automática, pero CSS permite modificar todas sus características. Cada una de las cajas está formada por seis partes que se describen a continuación:

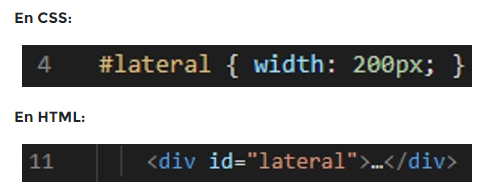
1. **Content** (contenido): se trata del contenido HTML del elemento (las palabras de un párrafo, una imagen, el texto de una lista de elementos, etc.)
2. **Padding** (relleno): espacio libre opcional existente entre el contenido y el borde.
3. **Border** (borde): línea que encierra completamente el contenido y su relleno.
4. **Background-image** (Imagen de fondo): imagen que se muestra por detrás del contenido y el espacio de relleno.
5. **Background-color** (color de fondo): color que se muestra por detrás del contenido y el espacio de relleno.
6. **Margin** (margen): separación opcional existente entre la caja y el resto de cajas adyacentes.

**Dimensiones de las cajas**

Anchura

Para los elementos de bloque y las imágenes, la propiedad width (anchura) permite establecer la anchura directamente mediante una medida.

Anchura = width



Si se utilizan unidades de medida, los valores indicados no pueden ser negativos. Si en vez de una unidad de medida se indica un porcentaje, la referencia de ese valor es la anchura del elemento que lo contiene. El valor heredado indica que la anchura del elemento se hereda de su elemento padre.

Si se establece la anchura de un elemento con la unidad de medida em, el valor indicado toma como referencia el tamaño de letra del propio elemento.

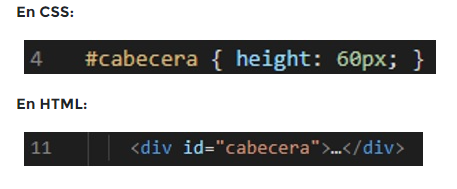
El valor auto es el valor por defecto de la anchura de todos los elementos. En este caso, el navegador determina la anchura de cada elemento teniendo en cuenta, entre otros, el tipo de elemento que se trata (de bloque o en línea), el sitio disponible en la pantalla del navegador y los contenidos de los elementos.

**Altura**

Al igual que sucede con ancho, la propiedad altura no admite valores negativos. Si se indica un porcentaje, se toma como referencia la altura del elemento padre. Si el elemento padre no tiene una altura definida explícitamente, se asigna el valor auto a la altura.

El valor heredado indica que la altura del elemento se hereda de su elemento padre. El valor automático, que es el que se utiliza si no se establece altura de forma explícita un valor a esta propiedad, indica que el navegador debe calcular automáticamente el elemento, teniendo en cuenta sus contenidos y el sitio disponible en la página.

Altura = height

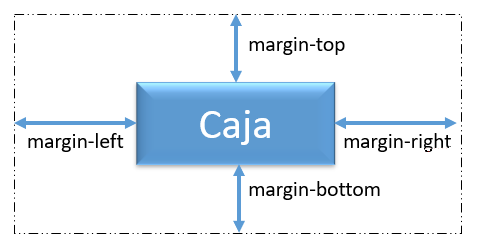


**Margen**

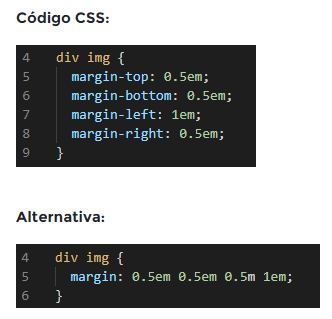
CSS define cuatro propiedades para controlar cada uno de los márgenes horizontales y verticales de un elemento.

1. margin-top => Margen superior
2. margin-right => Margen derecho
3. margin-bottom => Margen inferior
4. margin-left => Margen izquierdo

Cada una de las propiedades establece la separación entre el borde lateral de la caja y el resto de cajas adyacentes:



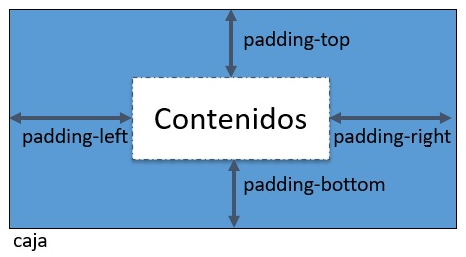
Los márgenes verticales (margin-top y margin-bottom) sólo se pueden aplicar a los elementos de bloque y las imágenes, mientras que los márgenes laterales (margin-left y margin-right) se pueden aplicar a cualquier elemento.



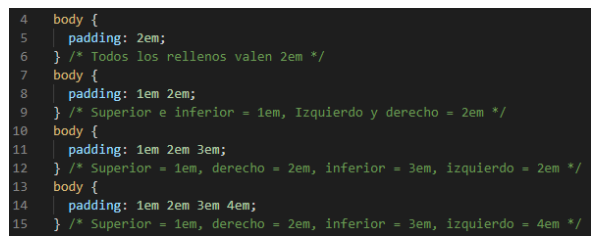
**Relleno o acolchado**

Relleno o acolchado. CSS define cuatro propiedades para controlar cada uno de los espacios de relleno horizontales y verticales de un elemento:

1. padding-top => Relleno superior
2. padding-right => Relleno derecho
3. padding-bottom => Relleno inferior
4. padding-left => Relleno izquierdo



**Algunos ejemplos de acolchado:**

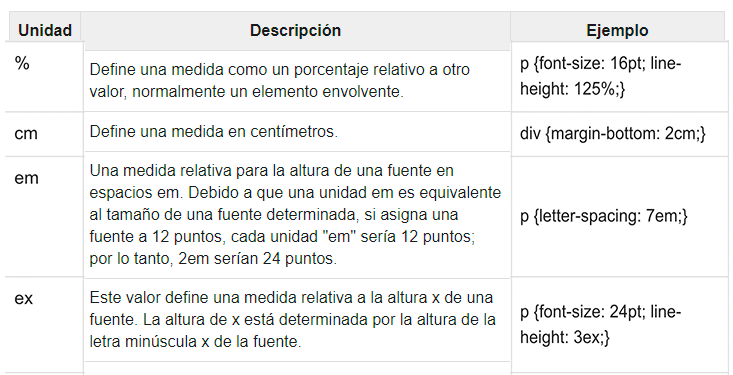


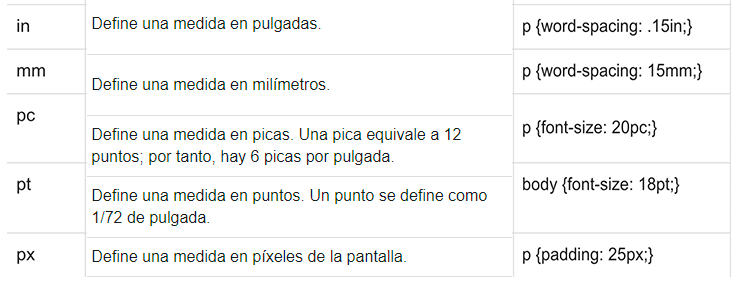
**Unidades de medida**

Antes de comenzar el ejercicio real, nos gustaría dar una breve idea sobre las unidades de medida CSS.

CSS admite una serie de medidas que incluyen unidades absolutas como pulgadas, centímetros, puntos, etc., así como medidas relativas como porcentajes y unidades em. Necesita estos valores al especificar varias medidas en sus reglas de estilo, por ejemplo,  **border = "1px solid red"**  .

Hemos enumerado todas las unidades de medida CSS junto con los ejemplos adecuados:

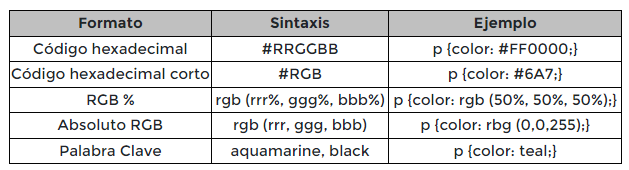




**Colores**

CSS usa valores de color para especificar un color. Por lo general, se utilizan para establecer un color para el primer plano de un elemento (es decir, su texto) o para el fondo del elemento. También se puede usar para afectar el color de los bordes y otros efectos decorativos.

Puede especificar sus valores de color en varios formatos. La siguiente tabla enumera todos los formatos posibles:



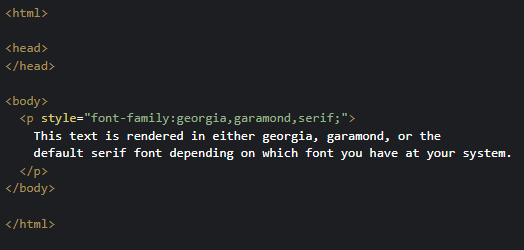
**Fuentes**

Veamos cómo configurar las fuentes de un contenido, disponible en un elemento HTML. Puedes establecer las siguientes propiedades de fuente de un elemento:

* La propiedad **font-family** se utiliza para cambiar la cara de una fuente.
* La propiedad de **font-style** se usa para hacer una fuente en cursiva u oblicua.
* La propiedad **font-variant** se utiliza para crear un efecto de versalitas.
* La propiedad de **font-weight** se utiliza para aumentar o disminuir la negrita o la luz de una fuente.
* La propiedad de **font-size** se utiliza para aumentar o disminuir el tamaño de una fuente.
* La propiedad de **font** se utiliza como forma abreviada para especificar otras propiedades de fuente.

Establecer la familia de fuentes

A continuación se muestra el ejemplo, que demuestra cómo configurar la familia de fuentes de un elemento. El valor posible podría ser cualquier nombre de familia de fuentes:



**Introducción**

El siguiente curso que estás por ver, trata sobre un framework de interfaz de usuario o UI según sus siglas en inglés. Si ya has programado anteriormente sitios web seguramente lo conoces. Bootstrap se ha convertido en unos de los frameworks que utilizan HTML, CSS y JS más popular y a su vez más utilizado por los desarrolladores. Su potente sistema de grilla, su extensa y clara documentación, su gran capacidad de adaptación a los distintos tamaños de dispositivos, su variada colección de componentes y la extensa cantidad de templates desarrollados con bootstrap son algunas de las razones, por lo cual hoy, es tan elegido.

**Objetivos**

El objetivo de este tutorial es brindarles una guía de inicio al framework que les facilite su pronta incorporación a sus proyectos. En la misma trataremos los temas más importantes del mismo tales como instalación, reglas para su uso, características principales entre otros. Al finalizar la lectura de este tutorial ya debemos estar en condiciones de hacer uso de bootstrap en nuestras aplicaciones y tener la capacidad para comenzar a profundizar sobre otros aspectos que se pueden encontrar en su documentación oficial.

**Historia**

*Bootstrap, originalmente llamado Blueprint de Twitter, fue desarrollado por Mark Otto y Jacob Thornton de Twitter, como un marco de trabajo (framework) para fomentar la consistencia entre las herramientas internas.*

*Antes de Bootstrap, se usaron varias bibliotecas para el desarrollo de interfaces de usuario, lo que generó inconsistencias y una gran carga de trabajo en su mantenimiento.*

*El primer desarrollo en condiciones reales ocurrió durante la primera "Semana de Hackeo" (Hackweek) de Twitter. Mark Otto mostró a algunos colegas cómo acelerar el desarrollo de sus proyectos con la ayuda de la herramienta de trabajo. Como resultado, decenas de temas se han introducido en el marco de trabajo.*

*En agosto del 2011, Twitter liberó Bootstrap como código abierto. En febrero del 2012, se convirtió en el proyecto de desarrollo más popular de GitHub.****[[1]](https://argentinaprograma.inti.gob.ar/mod/page/view.php?id=686" \l "_ftn1" \o ")***

**Versiones**

Bootstrap utiliza un versionado semántico incremental “Principal.Menor.Parche”. La versión “Principal” se utiliza para un cambio importante e incompatible con la versión anterior, actualmente se encuentra en la versión v5.0.1. La versión “Menor” agrega funcionalidad de una manera compatible con la misma versión principal. Por ejemplo la v3.4.2 significa que es compatible con la versión 3. Finalmente el cambio “Parche” se utiliza cuando realiza correcciones de errores dentro de la misma versión menor.

[[1]](https://argentinaprograma.inti.gob.ar/mod/page/view.php?id=686" \l "_ftnref1" \o ") <https://es.wikipedia.org/wiki/Bootstrap_(framework)>

**Instalación**

En el siguiente apartado mostraremos cómo incorporar fácil y rápidamente el framework a un proyecto, describiendo tres de las principales formas de instalación. El funcionamiento de Bootstrap requiere tanto de su propia librería de javascript como la de Popper (<https://popper.js.org/>), esta característica hace que se pueda instalar tanto atado (bundle en inglés) o por separado.

Es importante recordar que la referencia a los archivos CSS se debe incluir dentro de la etiqueta <head>...</head> mediante la etiqueta <link>, un ejemplo sería  <head>... <link href="dist/bootstrap/css/bootstrap.min.css" rel="stylesheet"></head>. Para el caso de  los archivos JS (javascript) preferentemente se deben incluir al final de la etiqueta <body> mediante la etiqueta <script>, un ejemplo de que incluye la librería de bootstrap y popper sería <body> … <script src="dist/bootstrap/js/bootstrap.bundle.min.js"></script> </body>.

1. **CSS y Javascript compilados y/o minimizados**

Esta forma de instalación requiere que los archivos de Bootstrap sean descargados previamente de forma local. Por ejemplo, la última versión disponible al momento de la edición de este documento, se encuentra en el siguiente enlace <https://github.com/twbs/bootstrap/releases/download/v5.0.1/bootstrap-5.0.1-dist.zip>.

Luego se incorporan a la estructura del proyecto, para finalmente poder referenciar los archivos del framework . Cumpliendo algunos estándares del desarrollo web, para el ejemplo de la  se copian los archivos dentro de la carpeta “dist” (distribución), pero esta puede tener cualquier otro nombre.

**Ejemplo 1: Instalación atada, compilada y minimizada**

<!doctype html>

<html lang="en">

  <head>

    <!-- Required meta tags -->

    <meta charset="utf-8">

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

    <!-- Bootstrap CSS -->

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

    <title>Instalación atada, compilada y minimizada</title>

  </head>

  <body>

    <h1>Atado, compilado y minimizado</h1>

    <!-- Bootstrap JS -->

    <script src="dist/bootstrap/js/bootstrap.bundle.min.js"></script>

  </body>

</html>

**Ejemplo 2: Instalación atada y compilada**

<!doctype html>

<html lang="en">

  <head>

    <!-- Required meta tags -->

    <meta charset="utf-8">

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

    <!-- Bootstrap CSS -->

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

    <title>Instalación atada y compilada</title>

  </head>

  <body>

    <h1>Atado y compilado</h1>

    <!-- Bootstrap JS -->

    <script src="dist/bootstrap/js/bootstrap.bundle.js"></script>

  </body>

</html>

**Ejemplo 3: Instalación separada, compilada y minimizada**

<!doctype html>

<html lang="en">

  <head>

    <!-- Required meta tags -->

    <meta charset="utf-8">

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

    <!-- Bootstrap CSS -->

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

    <title>Instalación separada, compilada y minimizada</title>

  </head>

  <body>

    <h1>Separado, compilado y minimizado</h1>

    <!-- Bootstrap JS -->

    <script src="dist/bootstrap/js/bootstrap.min.js"></script>

    <script src="dist/bootstrap/js/bootstrap.esm.min.js"></script>

  </body>

</html>

**Figura Ejemplo 4: Instalación separada, compilada y minimizada**

<!doctype html>

<html lang="en">

  <head>

    <!-- Required meta tags -->

    <meta charset="utf-8">

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

    <!-- Bootstrap CSS -->

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

    <title>Instalación separada y compilada</title>

  </head>

  <body>

    <h1>Separado y compilado</h1>

    <!-- Bootstrap JS -->

    <script src="dist/bootstrap/js/bootstrap.js"></script>

    <script src="dist/bootstrap/js/bootstrap.esm.js"></script>

  </body>

</html>

**CDN a través de jsDelivr**

La instalación a través de una Red de Distribución de Contenido (*content delivery network o CDN según siglas en inglés*) reemplaza la necesidad de descargar el contenido del framework y de referenciar sus archivos en forma local. La Red de Distribución de Contenido propuesta por Bootstrap de manera oficial para su versión actual (5.0.1) es [jsDelivr](https://www.jsdelivr.com/package/npm/bootstrap?path=dist" \t "_blank).

<!-- Bootstrap CSS -->

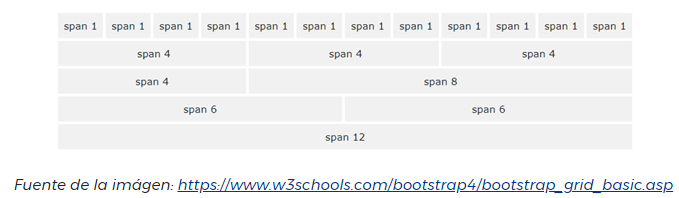
<head>...<link href="<https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css>" rel="stylesheet" integrity="sha384-+0n0xVW2eSR5OomGNYDnhzAbDsOXxcvSN1TPprVMTNDbiYZCxYbOOl7+AMvyTG2x" crossorigin="anonymous"></head>

<!-- Bootstrap JS -->

<body>...<script src="<https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/js/bootstrap.bundle.min.js>" integrity="sha384-gtEjrD/SeCtmISkJkNUaaKMoLD0//ElJ19smozuHV6z3Iehds+3Ulb9Bn9Plx0x4" crossorigin= "anonymous"></script></body>

**Sistema de grillas**

El sistema Grid permite dividir la página web en 12 columnas permitiendo adaptar la disposición de los elementos como se muestra abajo fácilmente.



Antes de continuar, es importante mencionar algunas reglas a la hora de trabajar con el sistema de grillas de Bootstrap:

●       Las filas deben estar dentro de un contenedor ( .container o .container-fluid).

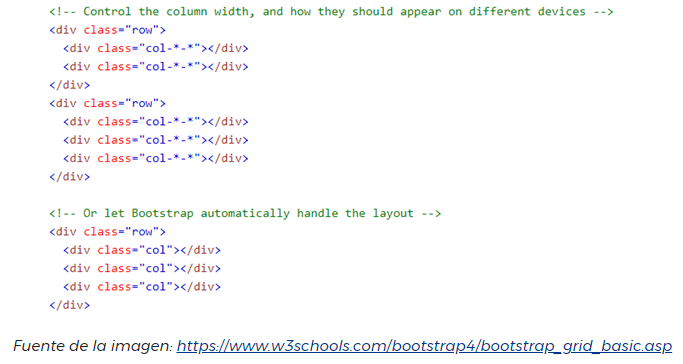
●       Usar filas para crear grupos de columnas. No a la inversa. Esto es posible agregando la clase .row al contenedor horizontal y la clase col al contenedor columna. Ejemplo <div class=”row”><div class =”col”>Contenido</div></div>.

●       El contenido debe estar dentro de las columnas.

●       Los únicos elementos anidados dentro de una fila (row) deben ser columnas (col).

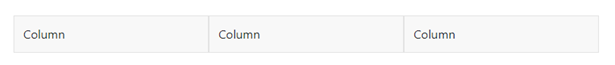
●       Las columnas se deben crear especificando el número de columnas disponibles (12).

A continuación se muestra la estructura básica del sistema de grillas que propone Bootstrap:



**Columnas de Ancho Automático**

Para definir columnas de ancho automático simplemente debemos utilizar la clase “col” como se muestra en el siguiente ejemplo:



<div class="container">

    <div class="row">

      <div class="col"> Column </div>

      <div class="col"> Column </div>

      <div class="col"> Column </div>

    </div>

  </div>

**Columnas de Ancho Fijo**

Las columnas de ancho automático distribuyen el ancho de la pantalla equitativamente para cada columna pero ¿y si deseamos establecer un ancho fijo?. En este caso, simplemente debemos especificar el número de columnas como sigue:



<div class="container">

    <div class="row">

      <div class="col-4"> Column </div>

      <div class="col-8"> Column </div>

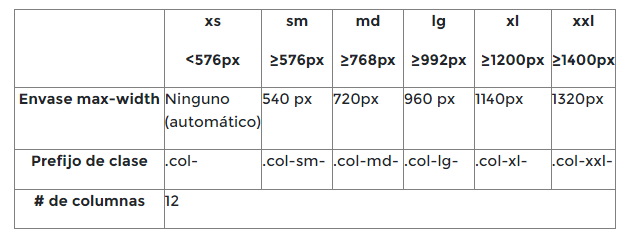
    </div>

  </div>

Las columnas se deben crear especificando de manera explícita teniendo en cuenta el número de columnas disponibles, cuya cantidad total es 12.

La combinación del sistema de grillas con los puntos de interrupción, es la unión perfecta para colocar nuestro contenido en la web según cada criterio de diseño y con la posibilidad de adaptarlo y ordenarlo al tamaño de pantalla donde se esté mostrando.

**Nota:** El sistema de grilla utiliza una cuadrícula flexbox para su diseño.



**Diseño**

El uso correcto del framework hace que los desarrolladores tengan que obedecer cierto estándar en el cumplimiento de algunas reglas, maquetando las páginas web de una forma muy rápida, permitiendo abstraerse de las configuraciones propias de css.

 Todas los elementos deben estar contenidos dentro de un contenedor con la clase  ***.container*** o***.container-fluid***. Ejemplo <div class=”container”>...</>.



**Contenedores**

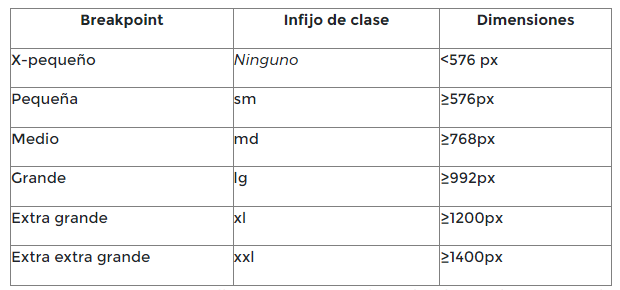
Tal como se indicó al inicio de esta sección, los contenedores son el espacio de construcción fundamental que incluye  Bootstrap, y es donde se ajustan y alinean su contenido dentro de un dispositivo o ventana gráfica determinada.



**Puntos de interrupción**

El framework para lograr un diseño ajustable o responsive incluye seis puntos de interrupción o de ruptura, predeterminados. El uso eficiente de estos puntos, como diseñadores o maquetadores web, nos habilita una herramienta para combinar en conjunto al sistema de grillas que propone bootstrap para generar contenidos que se ajusten correctamente a la pantalla del dispositivo donde se esté mostrando. Por esta razón debemos conocer en detalle cuáles son los infijos de clases y el rango de píxeles de pantallas en los que son identificados.

**Nota**: Es importante recordar que el concepto responsive hace alusión a un diseño que sea accesible y adaptable a todos los dispositivos.



Referencia de Tabla: <https://getbootstrap.com/docs/5.0/layout/breakpoints/>

**Columnas Responsive**

De acuerdo a lo expresado arriba, podemos utilizar los puntos de interrupción para lograr un diseño ajustable como sigue:



<div class="container">

    <div class="row">

      <div class="col-sm-4"> .col-sm-4 </div>

      <div class="col-sm-8"> .col-sm-8 </div>

    </div>

  </div>

**Columnas de Ancho Mixto**

Podemos combinar las columnas de ancho automático y fijo como sigue:



<div class="container">

    <div class="row">

      <div class="col"> .col</div>

      <div class="col-6"> .col-6 </div>

<div class="col"> .col</div>

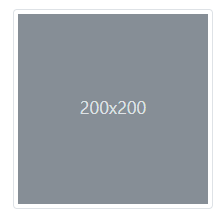
**Imagenes**

El framework, para el caso de las imágenes, nos propone una serie de clases con el objetivo de hacerlas adaptables y/o que nunca sean más grandes que las columnas que la contienen.



 <img src="..." class ="img-fluid" alt="...">

Ejemplo: Imágenes en miniaturas (200 x 200) con un borde de 1px



<img src="..." class= "img-thumbnail" alt="...">

Ejemplo: Alinear imágenes (1 al inicio de la fila y 2 al final de la fila)



 <img src="..." class ="rounded float-start" alt="...">

  <img src="..." class= "rounded float-end" alt="...">

**Tablas**

El uso de tablas en bootstrap es opcional, ya que no es un elemento que se herede del framework. Dado a su uso popular por los programadores, se propone un grupo de clases para trabajar con los estilos de una tabla.

 Ejemplo:



<table class="table">

    <thead>

      <tr>

        <th scope="col">Nro</th>

        <th scope="col">Nombre</th>

        <th scope="col">Apellido/th>

      </tr>

    </thead>

    <tbody>

      <tr>

        <th scope="row">1</th>

        <td>Maximiliano</td>

        <td>Guerra</td>

      </tr>

      <tr>

        <th scope="row">2</th>

        <td>Erik</td>

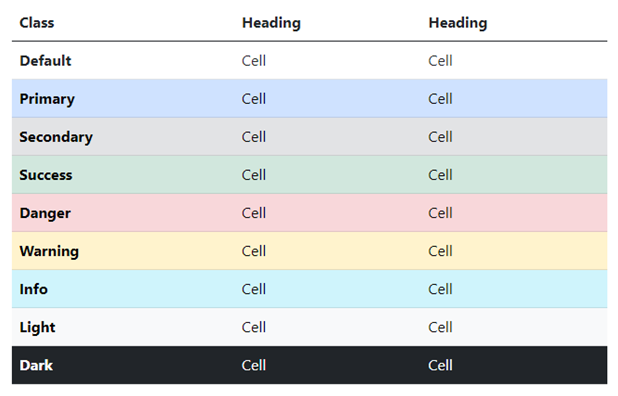
        <td>Thomas</td>

      </tr>

    </tbody>

  </table>

También se puede hacer uso de las clases contextuales para dar color a las tablas, de la misma forma que a cualquier otro elemento:



<!-- On tables -->

<table class="table-primary">...</table>

<table class="table-secondary">...</table>

<table class="table-success">...</table>

<table class="table-danger">...</table>

<table class="table-warning">...</table>

<table class="table-info">...</table>

<table class="table-light">...</table>

<table class="table-dark">...</table>

<!-- On rows -->

<tr class="table-primary">...</tr>

<tr class="table-secondary">...</tr>

<tr class="table-success">...</tr>

<tr class="table-danger">...</tr>

<tr class="table-warning">...</tr>

<tr class="table-info">...</tr>

<tr class="table-light">...</tr>

<tr class="table-dark">...</tr>

<!-- On cells (`td` or `th`) -->

<tr>

  <td class="table-primary">...</td>

  <td class="table-secondary">...</td>

  <td class="table-success">...</td>

  <td class="table-danger">...</td>

  <td class="table-warning">...</td>

  <td class="table-info">...</td>

  <td class="table-light">...</td>

  <td class="table-dark">...</td>

</tr>

**Figuras**

Al usar imágenes con texto relacionado se recomienda el uso de la etiqueta *<figure>*. La etiqueta *<figcaption>* contiene el texto de la imagen, pudiendo este ser alineado de igual manera que los elementos de texto.

Ejemplo:



<figure class="figure">

    <img src="..." class="figure-img img-fluid rounded" alt="...">

    <figcaption class="figure-caption">Figura 1.2</figcaption>

</figure>

**Controles**

Los controles dentro de un formulario pueden ser personalizados e incluso utilizados de forma adaptativa, gracias a las clases provistas por bootstrap.

Dimensionamiento:

●       .form-control-sm (tamaño pequeño)

●       .form-control (tamaño normal)

●       .form-control-lg (tamaño grande)

Habilitado:

●       Atributo  **disabled**(Control deshabilitado)

●       Atributo  **readonly** (Control sólo lectura)

●       Atributos  **disabled readonly** (Control deshabilitado, sólo lectura)

**Layout**

Cada grupo de campos de formulario debe residir en un elemento “*<form>*”. Bootstrap no proporciona un estilo predeterminado para el *<form>*elemento, pero hay algunas funciones de navegador poderosas que se proporcionan de forma predeterminada.

●       *“<button>*s dentro de un *<form>*valor predeterminado de type="submit", así que esfuérzate por ser específico y siempre incluye un archivo type.

●       Puedes deshabilitar todos los elementos del formulario dentro de un formulario con el disabled atributo en *<form>*.

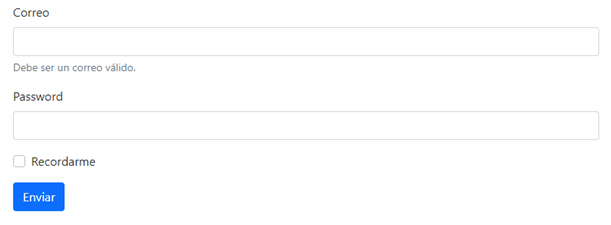
Ya que Bootstrap se emplea display: block y width: 100% para casi todos los controles del formulario, los formularios se mostrarán por defecto de manera vertical. Se pueden usar clases adicionales para variar este diseño en función de cada formulario.”

Puedes hacer [clic aquí](https://getbootstrap.com/docs/5.0/forms/layout/) para leer más sobre el tema.

**Formularios**

Al igual que las imágenes, bootstrap provee un grupo de clases para la presentación de los controles dentro de un formulario.

Ejemplo:



<form>

    <div class="mb-3">

      <label for="idE" class="form-label">Correo</label>

      <input type="email" class="form-control" id="idE" aria-describedby ="idEH">

      <div id="idEH" class="form-text">Debe ser un correo válido.</div>

    </div>

    <div class="mb-3">

      <label for="exampleInputPassword1" class="form-label">Password</label>

      <input type="password" class="form-control" id="exampleInputPassword1">

    </div>

    <div class="mb-3 form-check">

      <input type="checkbox" class="form-check-input" id="exampleCheck1">

      <label class="form-check-label" for="exampleCheck1">Recordarme</label>

    </div>

    <button type="submit" class="btn btn-primary">Enviar</button>

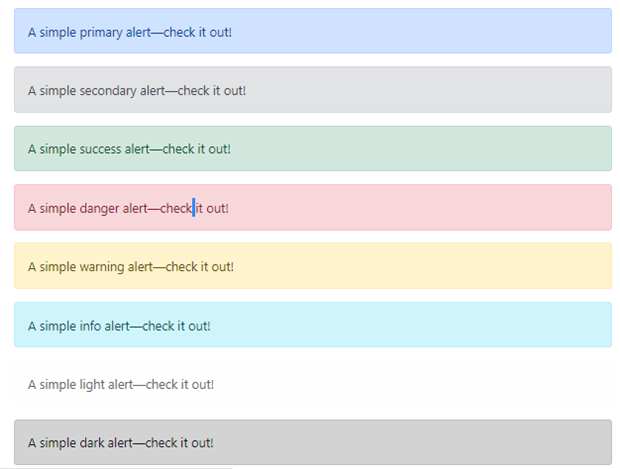
</form>

**Componentes**

**Alertas**

Este componente es muy utilizado a la hora de mostrar mensajes al usuario de forma flexible y de una manera más amigable. Las alertas son consecuentes con los estilos de colores que propone el framework.

Ejemplo:



 <div class="alert alert-primary" role="alert">

      A simple primary alert—check it out!

  </div>

  <div class="alert alert-secondary" role="alert">

    A simple secondary alert—check it out!

  </div>

  <div class="alert alert-success" role="alert">

    A simple success alert—check it out!

  </div>

  <div class="alert alert-danger" role="alert">

    A simple danger alert—check it out!

  </div>

  <div class="alert alert-warning" role="alert">

    A simple warning alert—check it out!

  </div>

  <div class="alert alert-info" role="alert">

    A simple info alert—check it out!

  </div>

  <div class="alert alert-light" role="alert">

    A simple light alert—check it out!

  </div>

  <div class="alert alert-dark" role="alert">

    A simple dark alert—check it out!

  </div>

**Botones**

Al igual que el resto de los controles, los botones incluyen el estilo predefinido.

Ejemplo:

<button type="button" class="btn btn-primary">Primary</button>

<button type="button" class="btn btn-secondary">Secondary</button>

<button type="button" class="btn btn-success">Success</button>

<button type="button" class="btn btn-danger">Danger</button>

<button type="button" class="btn btn-warning">Warning</button>

<button type="button" class="btn btn-info">Info</button>

<button type="button" class="btn btn-light">Light</button>

<button type="button" class="btn btn-dark">Dark</button>

<button type="button" class="btn btn-link">Link</button>



**Barra de navegación**

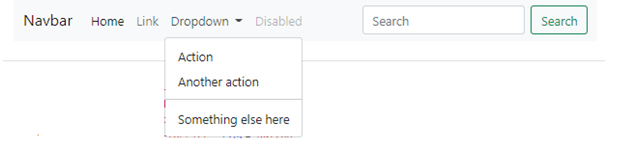
Por último abordaremos el tema de la barra de navegación, elemento muy utilizado en el desarrollo de aplicaciones web. A continuación citamos las principales reglas para el uso de la barra de navegación.

●       Las barras de navegación requieren comenzar con la clase .navbar

●       Las barras de navegación y su contenido son fluidos por defecto.

●       Las barras de navegación responden de forma predeterminada, pero se pueden modificar fácilmente para cambiar su comportamiento.

●       Garantice la accesibilidad mediante el uso de un elemento <nav> o, si usa un elemento más genérico como <div>, agregue a role="navigation" a cada barra de navegación.



**Referencias**

1. [*https://semver.org/*](https://semver.org/)
2. [*https://es.wikipedia.org/wiki/Bootstrap\_(framework)*](https://es.wikipedia.org/wiki/Bootstrap_(framework))
3. [*https://getbootstrap.com/*](https://getbootstrap.com/docs/5.0/getting-started/contents/)

## Introducción a JavaScript

Según [Mozilla (MDN 2020)](https://developer.mozilla.org/es/docs/Web/JavaScript), JavaScript, o simplemente JS, es un lenguaje de [scripting](https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-son-los-lenguajes-de-scripting/) [multiplataforma](https://www.lawebdelprogramador.com/diccionario/Multiplataforma/) y orientado a objetos. Es un lenguaje pequeño y liviano. Dentro de un [ambiente de host](https://es.wikipedia.org/wiki/Host), JavaScript puede conectarse a los objetos de su ambiente y proporcionar control programático sobre ellos. Se trata de un lenguaje de programación tipo script, basado en objetos y guiado por eventos, diseñado específicamente para el desarrollo de aplicaciones cliente-servidor dentro del ámbito de Internet.  
Los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.  
JavaScript es un [lenguaje de programación interpretado](https://lenguajesdeprogramacion.net/diccionario/que-es-un-lenguaje-interpretado/), por lo que no es necesario [compilar](https://developer.mozilla.org/es/docs/Glossary/Compile) los programas para ejecutarlos.  

## Objetivos

Introduciremos el concepto de programación estructurada y el lenguaje de programación interpretado JavaScript. Nos enfocaremos en JavaScript del lado del cliente, lo cual proporciona además una serie de objetos para controlar un navegador y su modelo de objetos (o DOM, por las iniciales en inglés de Document Object Model). Por ejemplo, las extensiones del lado del cliente permiten que una aplicación coloque elementos en un formulario HTML y responda a eventos del usuario, tales como clics, ingreso de datos al formulario, navegación de páginas, etc.

## Un poco de historia

¿Por qué se llama Javascript ?, según [Crockford](https://en.wikipedia.org/wiki/Douglas_Crockford" \t "_blank) (2020), el prefijo de Java sugiere que Javascript está de algún modo relacionado con Java, que es un subconjunto de instrucciones de Java o que es una versión menos potente. (...) “Parece que el nombre fue intencionalmente seleccionado para crear confusión y de esa confusión surge la mala interpretación” (...).

**Javascript es un lenguaje diferente a Java**. Si bien Javascript tiene una sintaxis similar a Java, no significa que tenga algo que ver con Java.  Java tiene una sintaxis similar al lenguaje C y esto no quiere decir que Java sea C. El lenguaje C es otro lenguaje de programación.

Javascript fue creado por [Brendan Eich](https://elpais.com/tecnologia/2020-08-28/el-lenguaje-de-programacion-que-cambio-la-web-y-tu-vida-cumple-25-anos.html) en 1995 para la empresa (y navegador popular en su época) Netscape. "Como Java era un lenguaje nuevo en esa era y con mucha aceptación, se optó por utilizar parte del nombre para ganar adeptos". (Brendan Eich, 2016).

Por último y para cerrar este apartado, vamos a decir que Javascript está estandarizado en [Ecma International](https://www.ecma-international.org/" \t "_blank) (asociación europea para la creación de estándares para la comunicación y la información) con el fin de ofrecer un lenguaje de programación estandarizado e internacional basado en Javascript y llamado [ECMAScript](https://developer.mozilla.org/es/docs/Glossary/ECMAScript) para que todos los navegadores puedan realizar un intérprete que acepte la misma sintaxis de Javascript.

**Sintaxis de JavaScript**

**Sintáxis**

A continuación vamos a ver la sintaxis del lenguaje JS (JavaScript) para familiarizarnos con la forma en que se codifica. Antes de comenzar con apartados más específicos veremos cuestiones generales sobre el lenguaje.

**Javascript es un lenguaje que distingue entre mayúsculas y minúsculas**. Por ejemplo, identificador (o nombre) “OnClickEvent” es distinto a “onclickevent”. Esta característica hace que prestemos mucha atención a la forma correcta de llamar usar y/o llamar a los identificadores. Las instrucciones son llamadas sentencias y son separadas por punto y coma (;) al final de la misma, si bien no es necesario en todos los casos es una buena práctica hacerlo.

Existen dos formas de realizar comentarios: MultiLine o SingleLine. Para el caso de multilínea se utilizan los caracteres “/ \*” y “\* /” por ejemplo:

**/ \* Esto es un comentario**

**multilínea \* /**

En el caso de un comentario de línea solamente es necesario usar los caracteres “//” al inicio de la línea a comentar por ejemplo:

**// Esto es un comentario de una sola línea**

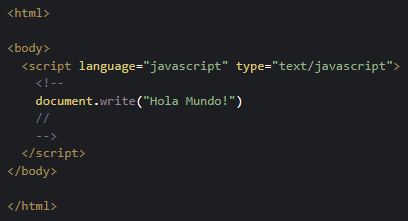
Por último debemos tener en cuenta que como cualquier otro lenguaje de programación, JS también usa palabras reservadas (palabras clave) que no podrán utilizarse por los programadores para usarlas como identificadores. Algunos ejemplo de ellas son: await, break, case, catch, class, const, continue, debugger, default, delete, do, else, enum, export, extensions, false, finally, for, function, if, import, in, instanceof, new, null, return, super, switch, this, throw, true, try, typeof, var, void, while, with, yield, etc.

**Mi primer código en Javascript**

**Hola mundo - JavaScript**

Tomemos un ejemplo de muestra para imprimir "Hola mundo". Agregamos un comentario HTML opcional que rodea nuestro código JavaScript. Esto es para guardar nuestro código de un navegador que no es compatible con JavaScript. El comentario termina con "// ->". Aquí "//" significa un comentario en JavaScript, así que lo agregamos para evitar que un navegador lea el final del comentario HTML como un fragmento de código JavaScript.  A continuación, llamamos a una función  **document.write**  que escribe una cadena en nuestro documento HTML.

Esta función se puede utilizar para escribir texto, HTML o ambos. Echemos un vistazo al siguiente código:



**Tipos de datos en JavaScript**

El último estándar ECMAScript define nueve tipos:

* Seis tipos de datos primitivos, controlados por el operador typeof
  1. **Indefinido**: tipo de instancia === "indefinido"
  2. **Booleano**: tipo de instancia === "booleano"
  3. **Número**: tipo de instancia === "número"
  4. **Cadena**: tipo de instancia === "cadena"
  5. **BigInt**: tipo de instancia === "bigint"
  6. **Símbolo**: tipo de instancia === "símbolo"
* 7. **Nulo:**tipo de instancia === "objeto". Tipo primitivo especial que tiene un uso adicional para su valor: si el objeto no se hereda, se muestra null.
* 8. **Objeto:**tipo de instancia === "objeto". Tipo estructural especial que no es de datos pero para cualquier instancia de objeto construido que también se utiliza como estructuras de datos: new Object, new Array, new Map, new Set, new WeakMap, new WeakSet, new Date y casi todo lo hecho con la palabra clave nuevo
* 9. **Función:** una estructura sin datos, aunque también respondió al operador typeof: typeof instance === "function". Esta simplemente es una forma abreviada para funciones, aunque cada constructor de funciones se deriva del constructor Object.

**Nota:**El único propósito valioso del uso del operador typeof es verificar el tipo de dato. Si deseamos verificar cualquier Tipo Estructural derivado de Object, no tiene sentido usar typeof para eso, ya que siempre recibiremos "object". La forma correcta de comprobar qué tipo de Objeto estamos usando es la palabra clave instanceof.

**Variables y Ámbitos en JavaScript**

El concepto de variable nos va a ser útil durante todo el proceso de programación en los distintos lenguajes. Javascript implementa el mismo concepto: Las variables son un espacio designado en la memoria para almacenar un dato. Se le asocia un nombre que la identifica y un tipo de datos.

Ejemplo de declaración y asignación de valor:

**var mi\_dato\_numerico = 3;**

**var mi\_dato\_texto = "hola";**

**Declaraciones:**

En JS existen tres tipos de declaraciones de variables:

**1. Usando var:**declara una variable, inicializándola opcionalmente a un valor. Si bien esta ha sido exclusivamente la forma de declarar variables durante mucho tiempo, sufre de unos efectos no deseados en el alcance (scope) de la declaración. En otras palabras, se aconseja no continuar utilizándola ya que el nuevo estándar de javascript permite otras formas de declaración que corrigen este problema.

El alcance o bloque de ámbito es el lugar en donde es visible esa variable. Por ejemplo, si se declara una variable dentro de una función, esta variable mantiene su valor y es visible sólo dentro de una función.

**2. Usando let:**declara una variable local en un bloque de ámbito (scope), inicializándola opcionalmente a un valor.

**3. Usando const:**declara una constante de sólo lectura en un bloque de ámbito. Una constante funciona como una variable pero no cambia su valor, es decir, representa un lugar de almacenamiento de tipos de datos en la memoria pero una vez asignado el valor inicial este no puede ser modificado. La sintaxis de la definición del identificador es la misma que la de las variables.

Las variables se usan como nombres simbólicos para valores en tu aplicación. Los nombres de las variables, llamados identificadores, se rigen por ciertas reglas. Un identificador en JavaScript tiene que empezar con una letra, un guion bajo (\_) o un símbolo de dólar ($). Los valores subsiguientes pueden ser números. Debido a que JavaScript diferencia entre mayúsculas y minúsculas, las letras incluyen tanto desde la "A" hasta la "Z" (mayúsculas) como de la "a" hasta la "z". Por ejemplo:

**var \_numero = 100;**

**var numero = 100;**

**let $numero = 100;**

**const numeroPar = 100;**

**const PI = 3.14;**

**Ámbito**

A partir de la versión ES6 / ECMAScript 2015, Javascript habilita tres ámbitos para la declaración de una variable:

**1. Ámbito Global:**Cuando se declara una variable fuera de una función, se le denomina variable global, porque está disponible para cualquier otro código en el documento actual.

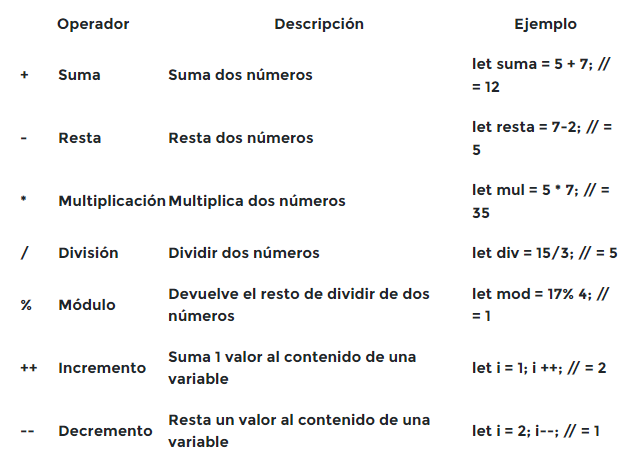
**2. Ámbito Bloque:**Este nuevo ámbito es utilizado para las variables declaradas con la palabra reservada let / const dentro de un bloque de código delimitados por llaves {}, esto implica que no pueden ser accesibles fuera de este bloque.

**3. Ámbito: Función:**Cuando se declara una variable dentro de una función, se le denomina variable local, porque está disponible sólo dentro de esa función donde fue declarada.

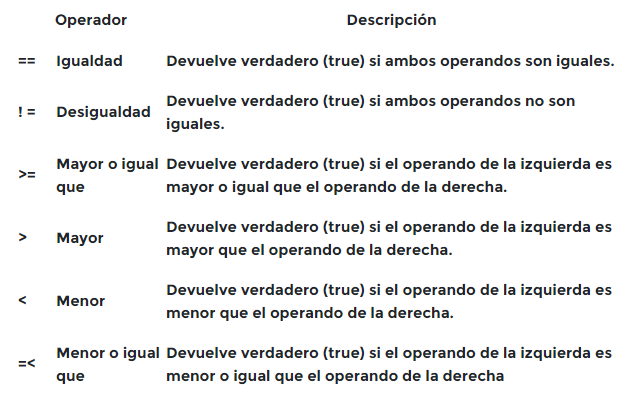
**Operadores en JavaScript**

Los operadores en los lenguajes de programación son símbolos que indican cómo se deben manipular los operandos. Los operadores en conjunto con los operandos forman una expresión, una fórmula que define el cálculo de un valor. Los operandos pueden ser constantes, variables o llamadas a funciones.

**Operadores matemáticos**



**Operadores lógicos**



**Operador condicional (ternario)**

El operador condicional es el único operador de JavaScript que necesita tres operadores. El operador asigna uno de dos valores basado en una condición.

La sintaxis de este operador es: condición? valor1: valor2. Si la condición es verdadera, el operador tomará el valor 1, de lo contrario tomará el valor 2. Se puede usar el operador condicional en cualquier lugar que use un operador estándar. Ejemplo: esta sentencia asigna el valor adulto a la variable estado si la edad es mayor o igual a 18, de lo contrario le asigna el valor menor:

**var estado = (edad>= 18)? "adulto": "menor";**

**Funciones y estructuras en JavaScript**

**SWITCH**

Una sentencia ***switch***permite a un programa evaluar una expresión e intentar igualar el valor de dicha expresión a una etiqueta de caso (case). Si se encuentra una coincidencia, el programa ejecuta la sentencia asociada. Una sentencia ***switch***se describe como se muestra a continuación:

**switch(*expression*) {**

**case *x*:**

***// code block***

**break;**

**case *y*:**

***// code block***

**break;**

**default:**

**// *code block***

**}**

Primero busca una cláusula case con una etiqueta que coincide con el valor de la expresión y entonces, transfiere el control a esa cláusula, ejecutando las sentencias asociadas a ella. Si no se encuentran etiquetas coincidentes, busca la cláusula opcional predeterminada y transfiere el control a esa cláusula ejecutando las sentencias asociadas. Si no se encuentra la cláusula predeterminada, el programa continúa su ejecución por la siguiente sentencia al final del switch. Por convención la cláusula por defecto es la última cláusula aunque no es necesario que sea así.

La sentencia opcional break asociada con cada cláusula case asegura que el programa finaliza la sentencia switch una vez que la sentencia asociada a la etiqueta coincidente es ejecutada y continúa la ejecución por las sentencias siguientes a la sentencia switch. Si se omite la sentencia break, el programa continúa su ejecución por la siguiente sentencia que haya en la sentencia switch.

**Funciones**

Las funciones son uno de los bloques de construcción fundamentales en JavaScript. Una función en JavaScript es similar a un procedimiento - un conjunto de instrucciones que realiza una tarea o calcula un valor, pero para que un procedimiento califique como función, debe tomar alguna entrada y devolver una salida donde hay alguna relación obvia entre la entrada y la salida. Para usar una función, debes definirla en algún lugar del ámbito desde el que deseas llamarla.

La declaración de una función consiste en:

1. Un nombre
2. Una lista de parámetros o argumentos encerrados entre paréntesis.
3. Conjunto de sentencias JavaScript encerradas entre llaves.

**function nombre (parámetro1, parámetro2)**

**{**

**// código que ha de ser ejecutado;**

**}**

En resumen, es un conjunto de instrucciones o sentencias que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente. Realizan varias operaciones invocando un nombre. Esto permite simplificar el código pudiendo crear nuestras propias funciones y usarlas cuando sea necesario.

**Estructuras repetitivas**

Las estructuras repetitivas o cíclicas nos ejecuta varias veces un conjunto de instrucciones. A estas repeticiones se las conoce con el nombre de ciclos o bucles.

Los lenguajes de programación son muy útiles para completar rápidamente tareas repetitivas, desde múltiples cálculos básicos hasta cualquier otra situación en la que tengas un montón de elementos de trabajo similares que completar. Aquí vamos a ver las estructuras de bucles disponibles en JavaScript que pueden manejar tales necesidades:

**Estructuras repetitivas en JavaScript:**

* **FOR**
* **WHILE**
* **DO - WHILE**

**Sentencia for**

Un bucle for se repite hasta que la condición especificada se evalúa como falsa.

**for ([expresion-inicial]; [condicion]; [expresion-final]) {**

**// código ha ser ejecutado;**

**}**

Cuando un bucle for se ejecuta, ocurre lo siguiente: la [expresión-inicial], si existe, se ejecuta. Esta expresión habitualmente inicializa uno o más contadores del bucle, pero la sintaxis permite una expresión con cualquier grado de complejidad. Esta expresión puede también declarar variables. Se evalúa la expresión [condición]. Si el valor de condición es true, se ejecuta la sentencia del bucle. Si el valor de condición es falso, el bucle para, finaliza. Si la expresión condición es omitida, la condición es asumida como verdadera. Se ejecuta la expresión [expresión-final], si hay una, y el control vuelve a evaluar la expresión [condición].

El ejemplo a continuación muestra un ciclo que se ejecuta 10 veces e imprime por consola la expresión “Número: X” donde x va del valor 0 al 9.

**for (var i = 0; i <10; i ++) {**

**console.log ("Número:" + i);**

**}**

**Sentencia while (mientras)**

Una sentencia while ejecuta sus sentencias mientras la condición sea como verdadera. Una sentencia while tiene el siguiente aspecto:

**while ([condicion]) {**

**// código ha ser ejecutado;**

**}**

Si la condición cambia a falsa, la sentencia dentro del bucle deja de ejecutarse y el control pasa a la sentencia inmediatamente después del bucle. La condición se evalúa antes de que la sentencia contenida en el bucle sea ejecutada. Si la condición devuelve verdadero, la sentencia se ejecuta y la condición se comprueba de nuevo. Si la condición es como falso, se detiene la ejecución y el control pasa a la sentencia siguiente al while.

**Sentencia do-while (hacer - mientras)**

Se utiliza para repetir instrucciones un número indefinido de veces, hasta que se cumpla una condición. A diferencia de la estructura mientras (while), la estructura hacer mientras (do while) se ejecutará al menos una vez. Ejemplo:

**let resultado = '';**

**let ​​i = 0;**

**do {**

**i = i + 1;**

**resultado = resultado + i;**

**} while (i <5);**

**console.log (resultado);**

**// resultado esperado: "12345"**

**Sentencia continue (continuar)**

La sentencia continue puede usar para reiniciar una sentencia for, while o do-while, continue termina la iteración en curso del código y continúa la ejecución del bucle con la siguiente iteración. A diferencia de la sentencia break, continue no termina completamente la ejecución del bucle.

**do{**

**i = i + 1;**

**continue;**

**} while(i <5);**

**Sentencia break (romper)**

La sentencia break se utiliza para salir de un bucle, switch. Break finaliza inmediatamente el código encerrado en un for, while, do-while o switch y transfiere el control a la siguiente sentencia.

**do{**

**i = i + 1;**

**if (i == 3) break;**

} while(i <5);

**Búsquedas en JavaScript**

Existen diferentes métodos que se pueden usar en JavaScript para buscar elementos dentro de un arreglo. El método a elegir depende del caso de uso particular, por ejemplo:

1. Obtener todos los elementos del arreglo que cumplen una condición específica. (**Filter**)
2. Obtener al menos uno de los elementos del arreglo que cumple dicha condición. (**Find**)
3. Obtener si un valor específico es parte del arreglo (**Includes**)
4. Obtener el índice de un valor específico (**IndexOf**).

**Array.filter()**

Podemos usar el método Array.filter() para encontrar los elementos dentro de un arreglo que cumplan con cierta condición. Por ejemplo, si queremos obtener todos los elementos de un arreglo de números que sean mayores a 10, podemos hacer lo siguiente:

let arreglo = [10, 11, 3, 20, 5];

let mayorQueDiez = arreglo.filter(element => element > 10);

console.log(mayorQueDiez) // resultado esperado: [11, 20]

**Array.find()**

Usamos el método Array.find() para encontrar el primer elemento que cumple cierta condición. Tal como el método anterior, toma un [Callback](https://developer.mozilla.org/es/docs/Glossary/Callback_function" \t "_blank) como argumento y devuelve el primer elemento que cumpla la condición establecida. Usemos el método find en el arreglo del ejemplo anterior.

let arreglo = [10, 11, 3, 20, 5];

let existeElementoMayorQueDiez = arreglo.find(element => element > 10);

console.log(existeElementoMayorQueDiez) // resultado esperado: 11

**Array.includes()**

El método includes() determina si un arreglo incluye un valor específico y devuelve verdadero o falso según corresponda. En el ejemplo anterior, si queremos revisar si 20 es uno de los elementos del arreglo, podemos hacer lo siguiente:

let arreglo = [10, 11, 3, 20, 5];

let incluyeVeinte = arreglo.includes(20);

console.log(incluyeVeinte) // resultado esperado: true

**Array.indexOf()**

El método indexOf() devuelve el primer índice encontrado de un elemento específico. Devuelve -1 si el elemento no existe en el arreglo. Volvamos a nuestro ejemplo y encontremos el índice de 3 en el arreglo.

let arreglo = [10, 11, 3, 20, 5];

let indiceDeTres = arreglo.indexOf(3);

console.log(indiceDeTres) // resultado esperado: 2

**Búsqueda de Máximos y Mínimos**

Uno de los problemas académicos más comunes es el de la búsqueda del valor máximo o mínimo dentro de una lista. JavaScript dispone de las funciones Math.max() y Math.min() con las que es posible obtener el máximo y mínimo respectivamente de un conjunto de números, por ejemplo:

Math.max(1, 2, 3, 4, 5); // resultado esperado: 5

Math.min(1, 2, 3, 4, 5); // resultado esperado: 1

El problema de estas funciones es que no permiten entradas de tipo array, solamente de tipo numérico. Normalmente se puede solucionar empleando diferentes aproximaciones como son los métodos reduce() o apply(). La forma más fácil de aplicar una función a un array es utilizando el método apply(). Simplemente se tiene que aplicar apply() a la función pasando como primer parámetro null y como segundo parámetro el array. Así se puede obtener el máximo o mínimo de un array simplemente con el siguiente código.

Math.max.apply(null, values) // resultado esperado: 5

Math.min.apply(null, values) // resultado esperado: 1

**Búsqueda Secuencial**

La búsqueda secuencial se define como la búsqueda en la que se compara elemento por elemento del vector/array con el valor que buscamos. Es decir, un clásico recorrido secuencial (for). De hecho, tenemos varias maneras de implementarlo, pero más allá de eso, el principio es el mismo. Comparar elemento por elemento hasta encontrar el/los que buscamos. Este tipo de búsqueda en el peor de sus casos ejecuta las instrucciones del loop n veces, es decir es la cantidad de elementos del arreglo XNo. En el mejor de los casos, el primer elemento del arreglo es el elemento que estamos buscando, por eso para ese caso ese elemento pasaría a ser X(1).

**Veamos un ejemplo de una función que implementa búsqueda secuencial en un arreglo:**

// Devolverá el índice donde encontró al elemento. Recibe el valor a buscar y el arreglo donde buscará

function sequentialSearch(element, array){

  for (var i in array){

    if (array[i] == element) return i;

  }

  return -1;

**}**

var letters = ["a", "b", "c", "d", "f", "g", "h", "i", "j", "k", "l", "m", "n"];

sequentialSearch("g",letters);

**Ordenamiento**

Javascript provee un método que ordena los elementos de un arreglo localmente y devuelve el arreglo ordenado Array.prototype.sort([compareFunction(a, b)]). El modo de ordenación por defecto responde a la posición del valor del string de acuerdo a su valor en el juego de caracteres Unicode. Cuando se utiliza el método sort(), los elementos se ordenarán en orden ascendente (de la A a la Z) por defecto:

const equipos = ['Real Madrid', 'Manchester Utd', 'Bayern Munich', 'Juventus'];

equipos.sort();

// ['Bayern Munich', 'Juventus', 'Manchester Utd', 'Real Madrid']

Dada esta característica el ordenar números pasa a ser una tarea no tan simple. Si aplicamos el método sort() directamente a un arreglo de números, veremos un resultado inesperado:

const numeros = [3, 23, 12];

numeros.sort(); // --> 12, 23, 3

El método sort() puede ordenar valores negativos, cero y positivos en el orden correcto. Cuando compara dos valores, se puede enviar la función compareFunction(a, b), como función de comparación y luego se ordenan los valores de acuerdo al resultado devuelto:

1. Si el resultado es negativo, a se ordena antes que b.
2. Si el resultado es positivo, b se ordena antes de a.
3. Si el resultado es 0, nada cambia.

Por ende si queremos ordenar los números en orden ascendente, esta vez necesitamos restar el primero parámetro (a) del segundo (b):

var numbers = [4, 2, 5, 1, 3];

numbers.sort(function(a, b) {

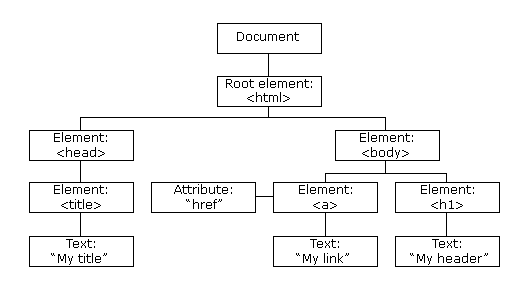
  return a - b;

}); // --> 1, 2, 3, 4, 5

**DOM que es?**

"*El Modelo de Objetos de Documento (*[*DOM*](https://www.w3.org/2005/03/DOM3Core-es/introduccion.html)*) del W3C es una plataforma e interfaz de lenguaje neutro que permite a los programas y scripts acceder y actualizar dinámicamente el contenido, la estructura y el estilo de un documento*"

Con HTML DOM, JavaScript puede acceder y cambiar todos los elementos de un documento HTML.



**Tipos de nodos:**

* **Document,**nodo raíz del que derivan todos los demás nodos del árbol.
* **Element,** representa cada una de las etiquetas HTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
* **Attr,**se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par atributo=valor.
* **Text,**nodo que contiene el texto encerrado por una etiqueta HTML.

**Nota:** Document representa la página web, por ende, para acceder a cualquier elemento de una web, se debe primero acceder a document.

**Métodos**

Los métodos son acciones que se pueden realizar a los elementos HTML mediante DOM.

Los más comunes son los  utilizados para encontrar elementos:

1. Encontrar elementos HTML por ID (**getElementById**)
2. Encontrar elementos HTML por nombre de etiqueta (**getElementsByTagName**)
3. Encontrar elementos HTML por nombre de clase (**getElementsByClassName**)
4. Encontrar elementos HTML mediante selectores CSS (**querySelectorAll**)

Este ejemplo encuentra el elemento con id="intro":

const element = document.getElementById("intro");

Este ejemplo encuentra todos los <p>elementos:

const element = document.getElementsByTagName("p");

Este ejemplo devuelve una lista de todos los elementos con class="intro".

const x = document.getElementsByClassName("intro");

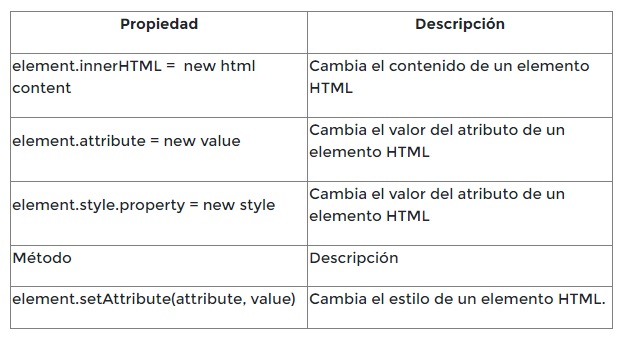
Este ejemplo devuelve una lista de todos los elementos  <p> con class="intro".

const x = document.querySelectorAll("p.intro");

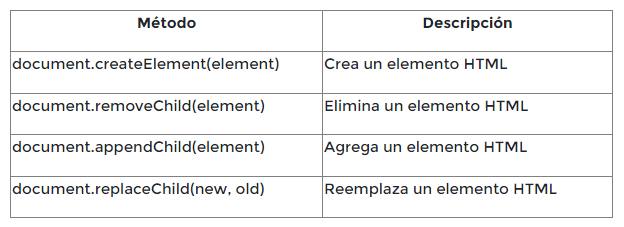
**Elementos**

Los elementos del DOM pueden ser creados, modificados o eliminados. A continuación veremos las principales acciones que se pueden realizar.

**Cambiar elementos HTML:**



**Agregar y eliminar elementos:**



**Ejemplo:**

function addElement () {

  // obtener el elemento div con id = "div\_example"

  const existDiv = document.getElementById("div\_example");

  // crear un nuevo elemento div

  const newDiv = document.createElement("div");

  // agregar el nuevo elemento div existente

  existDiv.appendChild(newDiv);

}

**Archivos JSON**

**JSON (notación de objetos javascript)** es un formato de intercambio de datos. Es muy parecido a un subconjunto de sintaxis JavaScript, aunque no es un subconjunto en sentido estricto. Aunque muchos lenguajes de programación lo soportan, JSON es especialmente útil al escribir cualquier tipo de aplicación basada en JavaScript, incluyendo sitios web y extensiones del navegador. Por ejemplo, es posible almacenar la información del usuario en formato JSON en una cookie o almacenar las preferencias de extensión en JSON en una cadena de valores de preferencias del navegador.  
  
JSON es capaz de representar números, valores lógicos, cadenas, valores nulos, arreglos y matrices (secuencias ordenadas de valores) y objetos (mapas de cadena de valores) compuestos de estos valores (o de otras matrices y objetos). JSON no representa de manera nativa tipos de datos más complejos como funciones, expresiones regulares, fechas, y así sucesivamente (en objetos de fecha serializados por defecto como una cadena que contiene la fecha en formato ISO, al no hacerlo de ida y vuelta, la información no se pierde por completo). Si se necesita que JSON represente tipos de datos adicionales, se puede transformar los valores, ya que son serializados, o antes de su deserialización.

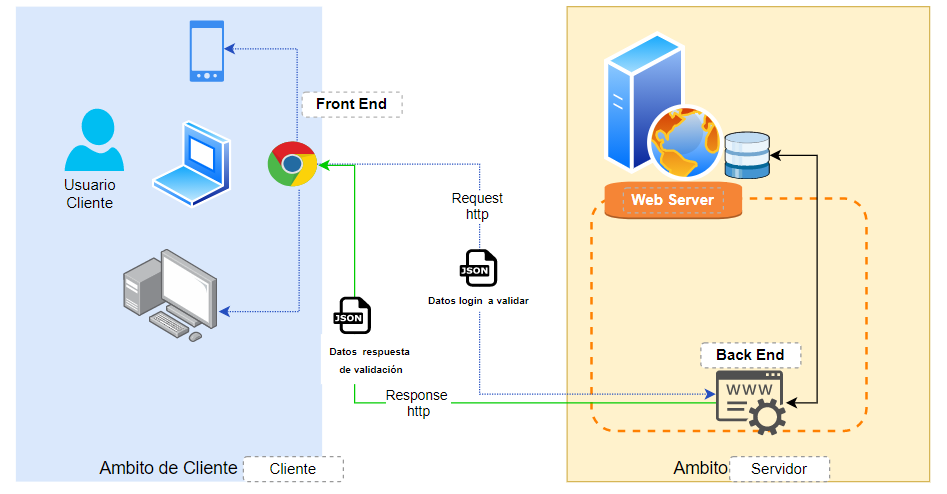
Los programadores conocemos estas convenciones utilizadas por JSON:

* - JSON son las siglas de JavaScript Object Notation.
* - El formato fue especificado por Douglas Crockford.
* - Fue diseñado para el intercambio de datos legibles por humanos.
* - Se ha ampliado desde el lenguaje de secuencias de comandos JavaScript.
* - La extensión del nombre de archivo es .json.
* - El tipo de Internet Media type es application / json.

**¿Para qué se usa?**

* - El formato JSON se utiliza para serializar y transmitir datos estructurados a través de una conexión de red.
* - Se utiliza principalmente para transmitir datos entre un servidor y aplicaciones web.
* - Los servicios web y las API utilizan el formato JSON para proporcionar datos vía internet.
* - Se puede utilizar con los lenguajes de programación y frameworks más actuales como  Typescript, Java, Angular, Spring boot, entre muchos otros.

**Nota:** La mayoría de las bases de datos pueden almacenar los archivos JSON, aunque almacenar un JSON completo rompe la normalización de la base de datos.  
  
En la siguiente imagen veremos un esquema que ilustra cómo viaja el archivo JSON:



Analizando la secuencia del esquema anterior, podemos observar la siguiente:

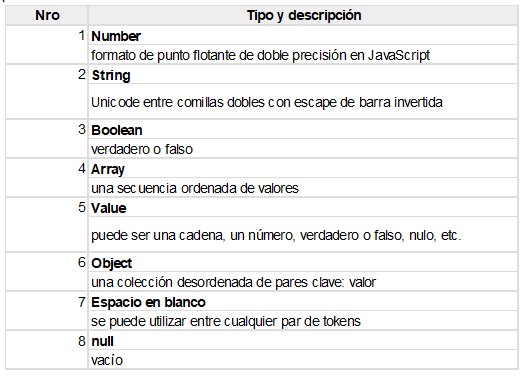
1. Un usuario ingresa sus datos de login en la página (Front End).
2. El código del Fornt End toma los datos y los serializa (convertir los datos/objeto del código a texto) en formato JSON y los envía al backend.
3. El Back End recibe el archivo JSON y lo deserializa (convertir el texto a objeto/código) para ser procesados, consultar la base de datos, etc.
4. Luego del procesamiento el Back End  la respuesta es serializada (convertir los datos/objeto del código a texto) para enviársela  al Front End notificando el resultado.
5. El código de la página Front End recibe el archivo JSON lo deserializa (convertir el texto a objeto/código) y muestra el resultado al usuario en la pantalla del navegador.

En la siguiente imagen te dejamos un ejemplo de cómo se ve un formato JSON en el código o en grilla,



**Tipos de datos**

El formato JSON admite los siguientes tipos de datos:



Reglas de sintaxis JSON:

* - Los datos están en pares de nombre / valor
* - Los datos están separados por comas
* - Las llaves sostienen objetos
* - Los corchetes sostienen matrices

**Los datos en JSON: un nombre y un valor**

Los datos JSON se escriben como pares de nombre / valor, al igual que las propiedades de los objetos de JavaScript. Un par de nombre / valor consta de un nombre de campo (entre comillas dobles), seguido de dos puntos, seguido de un valor:

"firstName":"Juana"

**Los Objetos en JSON:**

Los objetos JSON se escriben entre llaves. Al igual que en JavaScript, los objetos pueden contener varios pares de nombre / valor:

{"firstName":"Juana", "lastName":"Fernandez"}

**Las Matrices en JSON**

Las matrices JSON se escriben entre corchetes. Al igual que en JavaScript, una matriz puede contener objetos, te dejamos un ejemplo de una matriz Empleado con 3 Objetos:

"employees":[  
  {"firstName":"Juana", "lastName":"Fernandez"},  
  {"firstName":"Ana", "lastName":"Rntani"},  
  {"firstName":"Francisco", "lastName":"Petrol"}  
]

**Validar formato JSON:**

Existen muchas páginas para validar formato, te dejamos una para empezar, cada vez que trabajes con un formato JSON siempre debes estar seguro si el formato esta correcto y es válido, para ir a la página has clic [aquí](https://jsongrid.com/json-grid)

**Crear objetos simples de pruebas:**

Los objetos JSON se pueden crear con JavaScript. Veamos las diversas formas de crear objetos JSON usando JavaScript:

* - Creación de un objeto vacío

var JSONObj = {};

* - Creación de un nuevo objeto -

var JSONObj = new Object();

* -  Creación de un objeto con atributo **nombrelibro**con valor en cadena, atributo **precio** con valor numérico. Se accede al atributo usando '.' Operador -

var JSONObj = { "nombrelibro":"El hacedor", "precio":500 };