Introducción A

C S S

El lenguaje de diseño

De la web.

Creado por Juan Rubén Chávez Ultima edición del documento: 16 de junio 2023

índice

[Introducción. 2](#_Toc139373143)

[1. CSS. Inicio 3](#_Toc139373144)

[ Origen de CSS. 3](#_Toc139373145)

[ Estado actual de CSS. 3](#_Toc139373146)

[ Como trabajar con CSS. 3](#_Toc139373147)

[2. CSS. De 0 a intermedio 4](#_Toc139373148)

[ Unión de HTML Y CSS 4](#_Toc139373149)

[ Sintaxis CSS 5](#_Toc139373150)

[ Tipos de Selectores 6](#_Toc139373151)

[ Selectores simples elementales 6](#_Toc139373152)

[ Selectores compuestos 7](#_Toc139373153)

[ Selectores combinados 8](#_Toc139373154)

[ Selector Hermanos e hijos 9](#_Toc139373155)

[ Selector hijos directos 9](#_Toc139373156)

[ Especificidad 10](#_Toc139373157)

[3. Flexbox: . 10](#_Toc139373158)

[ Flexbox: Fundamentos. 11](#_Toc139373159)

[ Flex-direction 12](#_Toc139373160)

[ Flex-wrap 15](#_Toc139373161)

[ Alineamiento 16](#_Toc139373162)

[ Main Axis. 16](#_Toc139373163)

[ Cross Axis. 18](#_Toc139373164)

[ Aplicar propiedades flex a los hijos. 20](#_Toc139373165)

[Fuentes 20](#_Toc139373166)

***CSS:***

***El lenguaje de diseño de la web.***

# Introducción.

Cascade Style sheet (Hojas de estilo en cascada) nombrada de forma abreviada CSS, es el lenguaje que se utiliza en la web para darle diseño y estilo a las páginas de internet.

Esta necesita de HTML, ya que, su función es dotar de diseño y estética a dicho lenguaje y junto a javascript darle dinamismo a cualquier página.

Este lenguaje no es de programación, pero comparte muchas similitudes básicas de lo que un lenguaje de programación es, como lo son las variables, funciones, cálculos entre otras cosas.

Este tiene mucho potencial ya que es lo que dota de imagen a la web y crea toda una experiencia el poder visualizar cada página en internet.

Si se busca emprender en el desarrollo web, se debe conocer de forma fundamental HTML, CSS y Javascript.

# CSS. Inicio

### Origen de CSS.

Es el único lenguaje de diseño que existe y es utilizado para darle color y estética a las páginas web.

Esta nació en 1996, donde se decidió dividir la estructura, diseño y funcionalidad de una página web en tres lenguajes principalmente.

Aquí CSS sería el encargado del diseño, el cual se apoya sobre el lenguaje de estructura ya mencionado HTML.

Su nombre nace de una de sus principales características que es la cascada.

### Estado actual de CSS.

Actualmente, CSS funciona de forma escalable bajo el agregado de módulos. Esto marco que CSS como lenguaje, en lugar de ir agregando características y especificar su versión (1.0, 1.2,2,…), paso de un diseño monolítico a uno modular desde su versión 2.1. Su última versión fue 2.2 y fue conocida para forma de marketing como CSS 3.

Cuando lo modularizaron, esto permitió que todas las funciones agregadas a CSS sean analizadas y pasar por niveles de prueba, para al final llegar a colocarse como un estándar en el lenguaje; aquello que ya estaba, en dicho lenguaje permanecerá vigente. En todo caso, con el paso de los años puede llegar módulos que dejen obsoletos a otros que ya estaban, pero no dejaran de funcionar los anteriormente establecidos.

### Como trabajar con CSS.

Para poder trabajar con el lenguaje necesitaremos un interpretador de este, que vendría siendo cualquier navegador web, al igual se requiere de un editor de código (en mi caso Visual Studio Code) y practicar mucho :)

extensiones recomendadas para la eficiencia al trabajar en web hay muchas, yo recomendaría una extensión que recargue la página automáticamente para visualizar el progreso en tiempo real, al igual que un buen formato de escritura del código. En visual studio Code existe una extensión llamada live server, cuyo propósito es ver la página en tiempo real en el navegador.

# CSS. De 0 a intermedio

### Unión de HTML Y CSS

Como mencione al principio, CSS funciona utilizando de antemano la estructura de una página web, dicha estructura no es más que el código del lenguaje HTML.

Para unir a estos dos podemos optar por diversos métodos. Normalmente podemos hacerlo ya sea directamente en el HTML o en un documento de extensión CSS aparte.

Para hacerlo directamente en el HTML existe una etiqueta llamada “style”, la cual dentro de esta estará todos los estilos de CSS.

Texto

Descripción generada automáticamenteLa otra forma de trabajar seria con una referencia de nuestro archivo CSS dentro del encabezado del HTML. Para este necesitamos la etiqueta “link” donde el URL será la dirección actual del archivo CSS y indicarle a su atributo “rel” que es una hoja de estilo. Ejemplo en la imagen posterior.

Con ello, ya le indicamos a el navegador de antemano que renderice el CSS que este alojado en el enlace. [Enlace del ejemplo](https://codepen.io/juan-ruben/pen/NWXKVyj)

### Sintaxis CSS

La sintaxis es la forma en la que se escribe un lenguaje. En este caso la forma de CSS es muy simple. Esta se compone de la siguiente estructura:

**Una captura de pantalla de un celular

Descripción generada automáticamente con confianza mediaSelector**. Se refiere a el elemento en HTML a la que hacemos referencia.

**{}**: Dentro de este está todos los atributos que podemos aplicarle a el elemento.

**Propiedad**. Son aquellas características que podemos modificar de la entidad.

**Valor.** Es el valor que tomara la propiedad.

### Tipos de Selectores

Existen diversos tipos de selectores en CSS, estos tienen varias utilidades y funciones que podemos utilizar a nuestro favor.

De forma conceptual existen dos categorías:

Selectores simples:

* Elementales
* Selector Universal 🡪 \*
* Selector de tipo etiqueta 🡪 Nombre de la etiqueta.
* De atributo.
* ID 🡪 id del elemento
* Clase 🡪 clase del elemento
* Otros Atributos:
* [atributo del elemento]
* [atributo del elemento= valor]
* [atributo del elemento^= valor]
* [atributo del elemento\*= valor]
* [atributo del elemento$= valor]
* [atributo del elemento|= valor]

Selectores compuestos

* Selectores agrupados.
* Selectores combinadores
* Seudo clases- seudo elementos

### Selectores simples elementales

**Universal [ \* ]:** es como su nombre lo indica, un selector que directamente selecciona todos los elementos de nuestro HTML.

**Selector de etiqueta [<html>]**: es el más clásico; es escribir directamente el nombre del elemento de HTML para modificar sus propiedades de diseño.

**El selector ID [#ID]:** solo es recomendable si queremos modificar el diseño de un elemento único. Una de las ideas de CSS es poder reutilizar propiedades, pero con ID no podemos hacer esto, ya que solo es un elemento único.

**El selector de clase [.class]:** es el más habitual de utilizar, ya que podemos hacer alusión a un elemento o múltiples elementos con una sola clase.

**Otros atributos [atributo]:** actúa igual que los otros selectores, pero colocamos el atributo del elemento dentro de los corchetes y con ello especificamos que todo elemento con dicho atributo será seleccionado.

Podemos condicionar que está definiendo, que dicho atributo, debe tener un valor en específico *[atributo=valor].*

Otra modificación seria especificarle que siempre y cuando el valor empiece con el que le hallamos escrito *[atributo^=valor]* lo seleccione*.*

Cambiando el **^** con un **\*** le indicamos que siempre y cuando tenga ese valor en la oración lo seleccionara.

He de especificar que si es con **$** solo lo selecciona si termina con dicho valor.

La última modificación seria en un caso muy específico, en donde el valor del atributo tiene un guion ( - ) y hay texto después de este y queremos seleccionarlo, podemos utilizar el palito ( | ).

De forma mas explicada, el selector de atributo es más personalizada ya que podemos seleccionar diversos elementos de un HTML que compartan un atributo en particular y queramos que su apariencia cambie, o igual si tenemos una clase que se llama igual, pero en diferentes elementos tiene un nombre adicional con guion, podemos especificar que la apariencia cambie en unos elementos y en otros no. Un pequeño ejemplo:

Texto

Descripción generada automáticamente

[Clic aquí para el ejemplo](https://codepen.io/juan-ruben/pen/WNdeBqd?editors=1100)

### Selectores compuestos

Este tipo de selectores son los que requieren más de una palabra, una palabra y un símbolo u otras cosas a modo de ejemplo.

**Agrupados[ , ]:** Si queremos proporcionarle el mismo estilo a diferentes elementos, pero no queremos que tengan exactamente la misma propiedad o simplemente son elementos que comparten cosas en común, podemos agruparlos y definir sus propiedades en común, como lo muestra la sig imagen:

**Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente**

De esta forma podemos no solo es más fácil leer el código, si no también para el navegador, por que en lugar de tener que estar leyendo cada uno de los párrafos y aplicar el color, el ahora sabe que ese color se aplicara a todos; esto a gran escala da una gran eficiencia.

### Selectores combinados

**Descendiente:** Esto es cuando queremos seleccionar a un elemento que este dentro de otro en específico. Esto mismo también puede llamarse el seleccionar hijos, puesto que se tiene como concepto que un contenedor es padre, si dentro de este tiene otros contenedores/elementos.

Digamos que queremos cambiar el color del título que esté dentro de un div, podemos usar una clase y especificarlo únicamente a ese título, pero si queremos que cada div que tenga un título de nuestro html también cambie, en lugar de colocarle la clase, podemos indicarle en el css que todo div que tenga un elemento con la clase título cambie de color.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteComo vemos en la imagen, al seleccionar en CSS primero el div y después el h3, le estamos diciendo que queremos hacer referencia a todo h3 que este dentro de un div se le aplicara las sig propiedades.

[Enlace del ejemplo.](https://codepen.io/juan-ruben/pen/NWYEzea?editors=1100)

No hay un limite de descendencia, podemos especificar cuanto mas queramos, pero las buenas practicas indican que es mejor solo descender un solo nivel por cuestiones de especificidad (cuestión que veremos más adelante).

Como ultimo a recalcar, es imposible subir de nivel, si queremos seleccionar no el h3 del ejemplo, si no al div esto es imposible.

### Selector Hermanos e hijos

Repasando el concepto de acuñar un elemento padre o hijo y en este caso hermanos, debemos saber cómo es dicho orden. Si un elemento tiene otro elemento dentro, este será llamado padre, mientras el elemento dentro de este será su hijo. SI dicho padre tiene otro elemento junto el anterior, también será llamado hijo, y dicho hijo tendrá un hermano.

Texto

Descripción generada automáticamente

Sabemos que el h3 y los elementos p son hermanos, porque estan al mismo nivel, así como el mismo h2 y el div son hermanos.

**Hermano directo [ + ]:** Si queremos seleccionar el hermano de un elemento, siempre y cuando este este por debajo de el podemos seleccionarlo en CSS con la ayuda del selector +.

**Todos los hermanos[ ~ ]:** Si lo que queremos seleccionar es a todos los hermanos, solo debemos usar la virgulilla (esta se coloca con el alt-gr más la tecla de +~\* o con el num 4).

[Clic aquí para el ejemplo](https://codepen.io/juan-ruben/pen/gOeQjYm?editors=1100)

### Selector hijos directos

Este en particular nos es útil en caso de querer seleccionar todos los hijos de un elemento sin que este afecte a sus nietos. Es decir, si le damos un fondo rojo a todos los elementos p de un div, este afectara a todos los p, pero si solo queremos que afecto a los p que sean hijos del div podemos usar el signo [ > ].

[Clic aquí para el ejemplo](https://codepen.io/juan-ruben/pen/oNqJzqq?editors=1100)

### Especificidad

La especificidad es una de las tres propiedades básicas que tiene CSS. Su función es seleccionar cual selector es mas especifico en nuestro CSS y aplicarlo en el HTML.

Existen múltiples selectores que ya hemos visto, así como otras que veremos mas adelante, por lo que debemos saber que selector es más específico.

Los selectores que existen en CSS son:

**Etiquetas y pseudo-elementos:** 0,0,1

**Clases, atributos, pseudo-clases:** 0,1,0

**IDs:** 1,0,0

**Estilos en línea:** 1,0,0,0 (Aquellos estilos que colocamos directamente en el HTML)

**¡Important:** Le gana a todo. No se debe de utilizar ya que ignora toda especificidad en CSS.

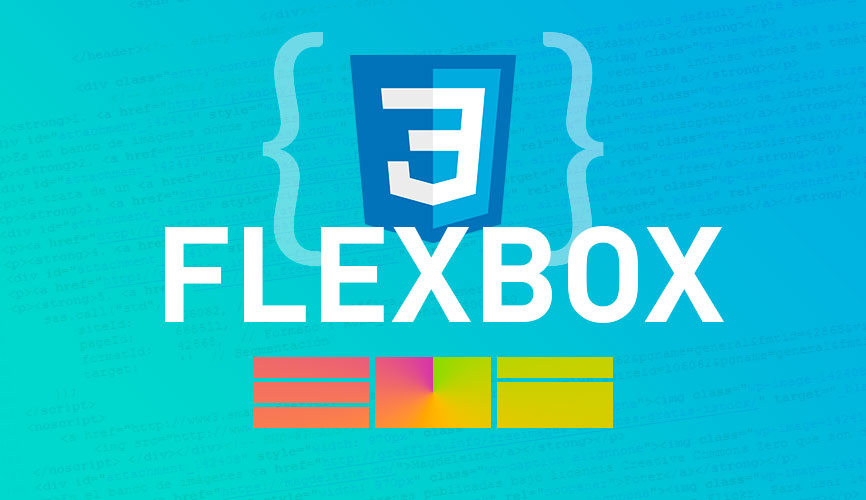
A modo de ejemplo, si tenemos un h1 y le aplicamos un color diferente en CSS escribiendo su etiqueta [h1] esta cambiara de color. Si posterior a ello colocamos en lugar de su etiqueta una clase, y esta con un color distinto, CSS le colocara al html el color de dicha clase, porque dicha clase tiene mas valor en la especificidad.

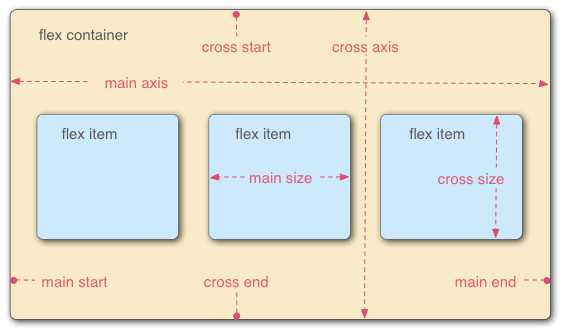
Como consejo se debe utilizar solo clases y nunca utilizar ¡important.

# Flexbox: .

Repasando un poco la historia, podemos decir que habido diversos cambios en cómo se trabaja el diseño web con HTML y CSS. En un principio se “maquetaba” con tablas y columnas en html, y con css se dotaba de un diseño para que fuera más agradable la página para el usuario. Posterior a ello, se intentó desarrollar métodos o una nueva forma de estructurar el diseño en nuestras paginas webs; mano de esto, se creó:

### Flexbox: Fundamentos.



* Flexbox es un modelo de layout que forma parte del módulo de la propiedad Display, por ende, para hacer uso de esta debemos aplicarle dicha propiedad a un contenedor.
* Este permite que las cajas sean flexibles, de allí su nombre.
* Existe display: flex y ***display:inline-flex*** para el contenedor, sea bloque o línea respectivamente.
* Se necesita un contenedor (Flex-container) y por lo menos un elemento hijo (flex-item).
* Las cajas flexibles son los hijos directos del contenedor padre, no el padre. Ello no evita que un elemento hijo pueda igualmente ser un flex-container y tener ítems dentro de este.
* Existen dos ejes para colocar y alinear los elementos (main axis y cross axis).
* Por defecto el eje principal es horizontal que va de izquierda a derecha, y el eje secundario es vertical de arriba abajo.
* Se puede intercambiar los ejes y la dirección de ambos si así lo queremos.

*Ilustración de ejemplo de la distribución de Flexbox*

Una vez que ya sabemos el orden de cómo se organiza Flexbox pasaremos a sus funciones y lo que podemos lograr con este.

### Flex-direction

Con flex-direction podemos como la propiedad dice, modificar la dirección de cómo se organizan los flex-items. Para aplicar esta debemos aplicar con anterioridad claramente la propiedad display:flex, de lo contrario cualquier modificación de flex no funcionara. Dicho esto, procederé a mostrar los diferentes escenarios de la propiedad Flex-direction.

* Forma

  Descripción generada automáticamenteA modo de ejemplo tenemos esta imagen de la derecha que está compuesta por un simple **div** con cinco elementos **li** que estan enumerados para saber el orden de cada **item** y con un poco de estilo.

Flecha

Descripción generada automáticamente con confianza baja

* En este segundo ejemplo vemos lo que sucede si a el elemento padre (div) le aplicamos la propiedad Display: flex.

Como vemos esto hace que automáticamente cada hijo del elemento padre se ordene en dirección del main axis y de una forma equilibrada. Como detalle de esto, si dotamos de mayor tamaño nuestro flex-items de tal forma que no quepan de manera horizontal uno alado del otro, el flex evitara que se desborden los ítems; esto lo consigue limitando el tamaño de estos.

A continuación, un ejemplo de que pasa si los elementos son más grandes que el ancho del navegador.

Tomando como referencia que el ancho de la pantalla en el sig ejemplo mide 1000px, al colocarle que cada item mida 400px, esto automáticamente hará que los ítems limiten su ancho para que estos puedan estar en una sola línea.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* Si aplicamos la propiedad display:inline-flex nuestros flex-container se comportara igual que la propiedad inline-block, solo que con las variantes de las otras características de Flexbox.

(el texto ejemplo 3:inline-flex esta en un span que recordemos que es un elemento inline, de allí que que dicho texto y el flex-container se colocan en una sola línea.)

Imagen que contiene Texto

Descripción generada automáticamente

* Al aplicar la propiedad flex-direction: row no sucederá nada, puesto que este es el valor por defecto que Flexbox tiene. Su función es la de indicar que la dirección de los flex-item debe ser acorde a el main axis, es decir, empezando los ítems de izquierda a derecha en posición horizontal.

Imagen que contiene Forma

Descripción generada automáticamente

* flex-direction: row-reverse dará los mismos resultados que el anterior mencionado, pero con la diferencia que la dirección de los ítems empezara desde la derecha.

Imagen que contiene Forma

Descripción generada automáticamente

* flex-direction: column indicara que ahora el main axis estará en la posición del cross axis, lo que colocara los ítems de forma vertical y de arriba abajo.

Imagen que contiene Forma

Descripción generada automáticamente

* flex-direction: column-reverse podemos intuir que esto será igual, solo cambiara el orden de los ítems. Estos empezaran de abajo arriba.

Imagen que contiene Forma

Descripción generada automáticamente

[Enlace de ejemplo](https://codepen.io/juan-ruben/pen/wvmOoyv)

Básicamente nosotros con esta propiedad (flex-direction) podemos modificar la dirección del main axis, en cuanto el cross axis no podemos alterarlo. Si Utilizamos el flex-direction: column, el main tomara el lugar del cross y el cross el del main.

### Flex-wrap

Con la propiedad flex-wrap podemos indicarle a el elemento Flexbox que queremos modificar que los elementos salten de línea, es decir, que si no caben unos al lados de otros, que se desplacen hacia abajo, arriba o no lo haga.

* El valor por defecto que usa Flexbox es flex-wrap:no wrap; de allí que no se saltan de línea los elementos.
* Si queremos que esto si suceda usaremos flex-wrap:wrap, con ello los ítems se desplazan hacia abajo.
* Para que se placen hacia arriba solo usamos flex-wrap:wrap-reverse;

Ejemplo a continuación:

Interfaz de usuario gráfica

Descripción generada automáticamente

Como dato, se pueden combinar las propiedades flex-direction y flex-wrap en la abreviatura [flex-flow](https://developer.mozilla.org/es/docs/Web/CSS/flex-flow).

Imagen que contiene Tabla

Descripción generada automáticamente

[Enlace del ejemplo](https://codepen.io/juan-ruben/pen/LYdaxBP)

### Alineamiento

Otros valores que podemos utilizar para el acomodamiento de los ítems son justify-content, align-items y align-content.

Con ayuda de justify-content podemos modificar el posicionamiento de nuestro main axis para que los ítems se acomoden de una forma u otra, y con el align-items y align-content es lo mismo solo que en lugar del main, es el cross axis. A continuación, mencionare los valores que podemos colocarle a la propiedad Justyfy-content:

### Main Axis.

* space-evenly: Acomoda los ítems del contenedor de tal manera que estos están separados unos de otros de igual medida.

Gráfico

Descripción generada automáticamente

* flex-start (default): este valor es lo que css toma por predeterminado. Como recordaremos de la imagen de ejemplo de cómo está distribuido Flexbox, este valor hace que los elementos se desplacen hacia la esquina superior izquierda

Gráfico

Descripción generada automáticamente

* flex-end. De forma casi inversa a el valor anterior, los ítems se desplazarán a la esquina superior derecha.

Gráfico

Descripción generada automáticamente con confianza media

* Center: Los item se desplazarán a el centro del eje horizontal.

Icono

Descripción generada automáticamente

* space-around: Consigue separación al principio y ultimo ítem que se ve a reducido a la mitad, debido que la separación entre los ítems es el doble de esta.

Gráfico

Descripción generada automáticamente

* space-between: Todos los ítems se desplazarán separándose unos de otros de tal forma que no dejen espacio entre ellos y el contenedor padre de cada una de sus laterales.

Gráfico

Descripción generada automáticamente

[Enlace de ejemplo](https://codepen.io/juan-ruben/pen/OJvegze)

Ahora, el tema con align-items y align-content es que son lo mismo, solo que se aplican para cuando uno queremos modificar una línea de elementos (align-items) o cuando queremos modificar varias líneas de elementos (align-content).

Con align-items podemos aplicar esto:

### Cross Axis.

* Stretch(Default): Este es el valor por defecto, y tiene como peculiaridad que si los flex-items no tienen un alto definido, estos buscaran expandir su alto todo lo que le sea posible.

Icono

Descripción generada automáticamente

*Ejemplo* ***sin*** *alto definido para los ítems*

* flex-start: los ítems se colocarán arriba del todo, este no es el de por defecto, puede parecer que si pero no lo es. Este, al igual que los demás valores de align-items solo se expanden a lo alto, tanto como su contenido lo permita, como normalmente es si no se define un alto.

Gráfico

Descripción generada automáticamente

*Ejemplo* ***con*** *alto definido para los ítems*

* flex-end: los ítem se colocan al final.

Imagen que contiene Gráfico

Descripción generada automáticamente

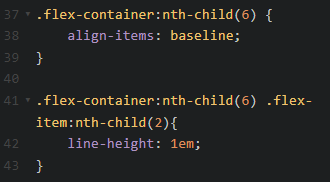
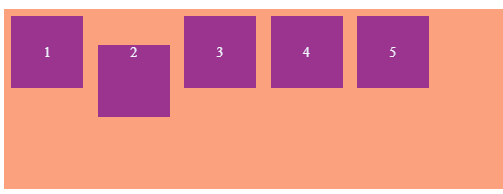
* center: El contenido se centrará en el medio del cross axis, es decir, en el eje vertical.

Imagen que contiene Gráfico

Descripción generada automáticamente

* baseline: Esta fuerza a que el contenido de los flex ítems se ponga a la par, en una misma línea imaginaria si lo queremos ver asi.

En el ejemplo fuerzo a que uno de los ítems tenga un alto mucho mayor, lo que haría en circunstancias normales desplazar dicho item hacia abajo, pero este se queda hasta la altura del contenido de los demás ítems gracias a el valor baseline. Esta propiedad solo podemos darla a destacar si tenemos diferentes ítems uno a lados de otros con altos diferentes.



[Enlace de ejemplo](https://codepen.io/juan-ruben/pen/abYgLbo)

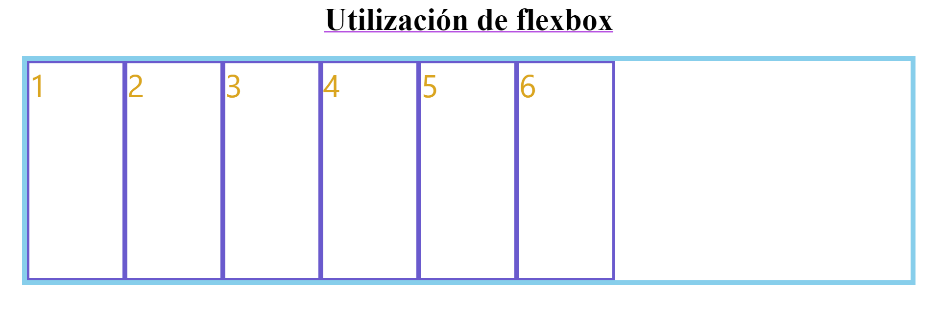
### Aplicar propiedades flex a los hijos.

Hasta el momento ya sabemos cómo trabajar con Flexbox y apropiarle atributos que modifique el flujo de su contenido. Ahora veremos aquellas propiedades que se utilizan aplicándolas directamente a los hijos de un contenedor Flexbox. Estas propiedades no afectaran a todos los hermanos, si no al hijo que llamamos.

Las propiedes que se aplican a los hijos son:

Flex-basis Flex-grow flex-shrink align-self order flex

A modo de ejemplo eh realizado una platilla, la cual le da estilos a unos cuantos elementos de nuestro HTML y a partir de esta iremos viendo como se comportan las propiedades que utilizaremos.



Ejemplo

Texto

Descripción generada automáticamente

#### Align-self

Esta propiedad es igual que align-items, solo que esta la aplicamos cuando queremos modificar la alineción de un item solamente, sin afectar a los demás.

Las propiedades que se le aplica a align-selft son las mismas align-items. Mostrare un ejemplo de su implementación.

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente Gráfico

Descripción generada automáticamente con confianza baja

Imagen que contiene Diagrama

Descripción generada automáticamenteGráfico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

#### Flex-basis

Esta propiedad modifica el valor del ancho del item. Este sobrescribe el ancho si es que ya se le indico a el item con anterioridad. Su utilidad radica en poder especificar anchos con palabras claves o porcentajes modificamos para diseño responsivo, aparte que dicha propiedad sobre escribe a width Sus valores son los siguientes:

Gráfico

Descripción generada automáticamente con confianza mediaTexto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Ejemplo

El valor content establece que su ancho será el total de su contenido.

Imagen que contiene Gráfico

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Aquí podemos ver un ejemplo de la MDN web docs, en la cual implementa estos diferentes valores.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

#### Flex-grow

Esta propiedad modifica el tamaño o crecimiento de nuestro item. El que tanto va crecer va depender del espacio restante ente el contenedor flexible menos el tamaño de todos los los elementos flexibles juntos. Si todos tienen el mismo valor de crecimiento, todos los elementos reciben la misma cantidad de espacio restante.

A dicha propiedad le podemos incluir valores numéricos como 1,2,3 etc. El numero 0 es el valor por defecto, asi que, al aumentarlo en un item, veremos un crecimiento en el eje principal.

Texto

Descripción generada automáticamente

Imagen que contiene Gráfico

Descripción generada automáticamente

Como vemos, solo afecto a el ancho y no a su alto que es lo que buscábamos.

De hecho, si vemos el inspector este mostraba antes de especificar el flex-grow a 1, este media igual que sus hermanos. Al colocar el valor 1, le estamos diciendo que tome el espacio restante que tiene el contenedor cuando este contiene los ítems. Ejemplo, si el contenedor mine 1500px, al decirle con grow que incrementara su tamaño este toma el valor de 1500 - (200px que mide cada item, lo que en total seria 1000px menos 500px). El resultado serio que el primer item mediría de ancho 500px.

Sabiendo cuanto es lo que nos sobra podemos repartir dicho espacio entre varios ítems, y dependiendo de que tal alto sea su número, le toca más espacio.

Escala de tiempo

Descripción generada automáticamente con confianza mediaTexto

Descripción generada automáticamente

Cuanto mas grande sea el número de flex-grow más será la división del espacio sobrante que tomara dicho item. Por eso mismo vemos que el num 3 es mas grande que num 2 y num 1.

#### Flex-shrink

Existe una propiedad la cual consigue el efecto contrario a flex-grow, la cual es flex-shrink.

Esta actúa cuando no tenemos espacio suficiente. Esta propiedad busca distribuir el espacio que tenemos cuando ya no tenemos. Sabemos que, si colocamos ítems con un ancho mayor a su madre no se desbordara, esto debido a esta propiedad la cual le da una unidad a cada contenedor para que quepan.

Texto

Descripción generada automáticamente con confianza baja

En este caso si nosotros establecemos que los ítems son de 500px esto claramente desborda el contenedor padre.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Al ver nuestros ítems en el editor veremos que estos no respetan el ancho que les dimos, esto a causa que flex por defecto le asigna la propiedad flex-shrink a sus ítems con un valor de 1.

Esto lo que consigue es dividir entre los ítems que existen, y asignarle una unidad de medida de ancho.

Otra forma de verlo es que flex mide cuantos ítems hay, en este caso son 6, entonces divide el espacio total entre 6 y le asigna este valor a cada item (le asigna una unidad de fracción 1 / 6).

Entonces al utilizar flex-shrink y aumentamos el valor a 2, lo que ara ya no es repartir el espacio equitativamente, si no quitarle mas espacio a el item con dicha propiedad y repartirla entre los demás ítems.

Ejemplo, si nuestro contenedor padre mide 400px y tenemos 4 items que miden 200px, al ver que mas de dos ítems rebasa el contenedor, flex-shrink repartirá 400/4, dando un ancho de 100px a los ítems.

Al utilizar flex-shrink a un item, y le indicamos que tendrá unidad 2, este se ara mas pequeño ya que en lugar de solo reducir su tamaño a 100px este le dará 40px a sus compañeros para ser mas grandes.

Texto

Descripción generada automáticamente

Imagen que contiene Icono

Descripción generada automáticamente

El tamaño del primer ítem será 40px, mientras que los demás se repetirían el pedazo entre los ítems, los cuales son 3, es decir le toca un tercio de lo que le quitamos a nuestro ítem, es decir,20px, por ende, los demás ítems miden 120px mientras que el primero mide 40, que nos da de total

Los 400px del padre.

De hecho, al utilizar la propiedad basis o width directamente no respeta el acho que le indicamos porque de seguro no caben en el contenedor padre. Si a un item le idicamos que su flex-shrink es de 0. Esto ara que no se encoja e intente acoplase con los demás ítems, si no que los demás ítems tendrán que acomodar su tamaño para caber a la par de dicho item con shrink dentro del contenedor.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza bajaTexto

Descripción generada automáticamente

Al decir que el primer ítem no tendrá que repartir espacio, este no se encoje y por ende seguirá siendo de 200px a diferencia de sus hermanos.

Esto a su vez al alterar el flujo, vemos que no caben todos los ítems en dicho contenedor.

Fácil podríamos ocultar dicho desbordamiento con overflow o utilizar flex-wrap.

#### Flex

Este es un acortador el cual aconseja la documentación utilizarlo el cual podemos especificar los valores de la propiedad flex-grow, flex-shrink y flex-basis. Sus propiedades por defecto serian 1,0, auto. Al aplicar el valor none a flex, le indicara que dichos valores serán 0,0, auto. Dicha propiedad ocupa un valor mínimo para funcionar, en este caso especificar el flex-grow.

#### Order

Con order establecemos que orden va ocupar cada item de nuestro contenedor. Por defecto su valor es 0, es decir, todos los item parten desde 0.

No debemos confundirnos con dicho item, al especificar un numero no estamos indicando en que posición de los ítems se colocara, si no como una prioridad. Esta es parecía a z-index, ya que si colocamos un numero mayor a 0, se podrá al final de la fila porque ‘pesa’ más. Entre mas grande es el valor, con mayor prioridad se colocará al final de la fila.

Entonces con order podemos colocar los ítems como queramos, aunque en el html este ordenado de diferente manera.

Gráfico

Descripción generada automáticamente con confianza mediaTexto

Descripción generada automáticamente con confianza baja

# 

Si colocaríamos como order 1 a el item 2 veremos que este se posicionaría adelante del último número, aunque hayamos utilizado antes order. Esto es efecto de la cascada según yo.

Imagen que contiene Gráfico

Descripción generada automáticamenteTexto

Descripción generada automáticamente

En cualquier caso, es mejor aplicar un aumento en lugar de utilizar un valor, esto puede conllevar a una mala práctica.

Imagen que contiene Gráfico

Descripción generada automáticamenteTexto

Descripción generada automáticamente

# Grid:

### Grid: Fundamentos.

# Fuentes

* by MDN contributors. (2022, 14 agosto). *Conceptos Básicos de flexbox - CSS | MDN*. Developer.Mozilla.Org/. [https://developer.mozilla.org/es/docs/Web/CSS/CSS\_Flexible\_Box\_Layout/Basic\_Concepts\_of\_Flexbox#pr%C3%B3ximos\_pasos](https://developer.mozilla.org/es/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox%23pr%C3%B3ximos_pasos)
* Curso de CSS. (Autor: Adrian Heras. Experto con mas de una década en web,) <https://www.udemy.com/course/draft/2808871/learn/lecture/25720604#overview>
* Curso de CSS.Youtube channel: Dorian Desings Curso ya no disponible.