Capitulo 1 curso css
Formato:
selector {
propiedad:valor;
}
Selector: Acá indicamos que elemento o conjunto de estos queremos cambiar.
Propiedad: En la propiedad indicamos que es lo que vamos a cambiar, color, fondo,etc.
Valor: indicamos el valor de la propiedad. El color por el que lo vamos a cambiar, el tipo de fuente, etc.

SELECTORES:
Universal: Es el *(asterisco), selecciona todos los elementos de la estructura html.
De tipo: Estos son los que seleccionamos por elemento o etiqueta, directamente por sus nombres. Button, h2,etc.
Clases: En el elemento html colocamos una clase mediante el atributo class, y luego lo llamamos en el archivo css con el nombre que le colocamos y un punto por delante. (class:"hola") (.hola). Esto cambiara todos los elementos con esa clase.
Id: Para seleccionar un elemento por id, se usa el numeral (#), y luego se le coloca el nombre del id que tiene el elemento. El id tiene que ser siempre único.
Por atributo: Para seleccionar por atributo, colocamos entre [ ] (corchetes) en nombre del atributo del elemento junto con su valor, que vallamos a modificar.
Descendiente: Primero seleccionamos al elemento contenedor, y luego al elemento hijo. (h2 p {). También se puede hacer esto con las clases, primero colocamos la clase y luego el elemento. (.hola p{) Acá seleccionamos un (p) que esta dentro de un elemento con clase (hola).
Pseudo clase: Son eventos donde podemos dar propiedades distintas a las normales y se seleccionan de la siguiente manera. Coloco el elemento seguido de dos puntos y la seudo clase que elija. (P:hover { color:red;}) de esta manera selecciono el elemento p con el evento hover, que cuanto pase el mouse por arriba del p, se ponga rojo.

Especificidad: La especificidad es la manera mediante la cual los navegadores deciden qué valores de una propiedad CSS son más relevantes para un elemento y, por lo tanto, serán aplicados. La especificidad está basada en las reglas de coincidencia que están compuestas por diferentes tipos de selectores CSS.

El css da prioridad al cambio en forma de cascada si dos selectores son iguales, siempre va a dar prioridad al de mas abajo. Pero mientras no sea así, css le va a dar prioridad según la especificidad o nivel de jerarquía que tenga el selector, como se muestra en la imagen de mas arriba.
Y por sobre toda la jerarquía esta el !important, que se ubica mas arriba de los estilos en linea. Con este le podemos dar prioridad al selector o propiedad que queramos, y css le va a dar prioridad sin importar su jerarquía. EJ: h1 { color:red !important}, en este caso a pesar de ser el selector mas bajo en la jerarquía, con el important se le da la mayor importancia dentro de css (no es muy recomendable de usar).

Metodologia BEM:

BEM CSS es una metodología de nomenclatura para definir las clases en los nodos HTML del documento.
Es decir, es **una manera de nombrar las clases de los nodos de tu HTML para posteriormente atacarlos con CSS de una manera fácil, sencilla y clara**.
El objetivo de BEM es dar mucha más **transparencia y claridad en tu estructura HTML y CSS**.
BEM te dice cómo se relacionan las clases entre sí, lo que es particularmente útil en secciones complejas del documento.
Todas las clases del proyecto pueden encajar con esta filosofía.

```
<!-- EJEMPLO 1 -->
<div class="block">
    <div class="block__element">Elem 1</div>
    <div class="block__element">Elem 2</div>
    <div class="block__element block__element--modifier">Elem 3</div>
</div>

<!-- EJEMPLO 2 -->
<div class="item item--modifier">
    <div class="item__element">Elem 1</div>
    <div class="item__element">
        <div class="item__another-element">Elem 2</div>
        <div class="item__another-element">Elem 3</div>
    </div>
    <div class="item__element item__element--modifier">Elem 4</div>
</div>
// EJEMPLO 1
.block{ color: inherit; }
.block__element{ color: inherit; }
.block__element--modifier{ color: inherit; }

// EJEMPLO 2
.item{ color: inherit; }
.item--modifier{ color: inherit; }
.item__element{ color: inherit; }
.item__element--modifier{ color: inherit; }
.item__another-element{ color: inherit; }
```

Se trabaja por secciones o contenedores. Las clases de los nodos hijos va cambiando pero al principio en la clase siempre lleva la misma del nodo padre. Los guiones bajo se agregan para ir separando según se va avanzando en nodo.

Unidades: Son las unidades de medida que se usan dentro de css, están las RELATIVAS y las FIJAS.
Medidas relativas: Son las medidas que dependen de otro factor, por ejemplo. Al irse achicando la

pantalla, estas se van ajustando a la misma. Osea dependen del tamaño de la pantalla.
em: 1em por defecto según el navegador son 16px. Las medidas son hereditarias, por lo tanto en este caso 1em va a ser igual a la medida que pongamos en la caja contenedora, y no 16px. Sin importar la propiedad.
v**w:** Esta medida es porcentual a la pantalla, si ponemos 50vw, nos referimos al 50% de la pantalla. Y esta medida generalmente se usa con el whidth(ancho). Y para manejar el height(alto) se usa el **VH(**viuportHeight). Estas son mas optimas de usar que el %(porcentaje).

Propiedades de texto:
font-size: Es el tamaño de la letra.
Font-family: Es la tipografía.
Line-height: Es el tamaño que OCUPA la letra, donde el valor de 1 se divide desde el medio de la letra, 0,5 del medio para arriba y 0,5 del medio para abajo.
Font-weight: Con esto se modifica el grosor de la letra, según la tipografía. Con el valor sin ningún símbolo. (100)
Desde google fonts se puede importar tipografía, copiando las etiquetas html y luego cambiando la misma con font-family desde el archivo css.

Normalize:
El navegador agrega estilos a la pagina por defecto. Pero a veces el navegador le agrega cosas que nos generan problemas (ej: padding,margin,etc). Se busca "normalize.css" en internet y se descarga el archivo, luego lo enlazamos a nuestra pagina con una etiqueta link, o copiamos el link de la pagina y también lo pegamos en el href de una etiqueta link.

Cajas: Hay dos tipos de cajas, las que son en linea y las que son en bloque.
En bloque: Siempre el ancho de las caja en bloque es el total de su contenedor.
En linea: El elemento siempre va a tener el ancho del contenido.

Display: Es una propiedad que te permite cambiar el comportamiento de las cajas.
Display: inline/block. De esta manera cambiamos el formato de caja de los elementos, si por defecto son (block), los cambiamos con display:inline(en linea) y si son inline lo cambiamos con Display:block
A los elementos en linea no se les puede dar un height ni un widght, a los elementos en bloque si se puede.

Propiedades de caja: Son las propiedades que modifican o afectan a las cajas, su contorno o el entorno de estas.
Background-color: Esta propiedad cambia el color de fondo del elemento.
Padding: Es la distancia que hay entre el texto y los bordes de la caja. Se pueden dividir en 4.
padding-left: Que es el espacio entre el texto y los bordes izquierdos de la caja.
padding-top: Que es el espacio entre el texto y los borde superior de la caja.
padding-right: Que es el espacio entre el texto y los bordes derechos de la caja.
padding-bottom: Que es el espacio entre el texto y los bordes inferiores de la caja.
En vez de poner uno por uno, se puede resumir solo con la propiedad padding.
padding: 20px; Esto le da un padding de 20 px a todos los bordes por igual.
padding: 20px 30px; en el primer termino(20px) se coloca los valores del eje Y(vertical), y luego en el segundo termino se coloca los valores del eje X (horizontal).
padding: 20px 30px 10px 30px; En el primer termino le damos valor a la distancia con el borde superior, en el segundo termino le damos valor a la distancia con el borde derecho, el tercer termino la distancia con el borde inferior, y el 4 termino la distancia con le borde inferior. Va en el sentido de las agujas de reloj comenzando desde arriba.
Margin: Es la distancia que hay entre dos cajas. Funciona exactamente igual que padding a la hora

de colocar medidas.
Height: Con este modificamos el alto.
Widht: Con este modificamos el ancho.
Border-radius: Es cuanto vamos a redondear el borde.
Border: Es una propiedad acortada también. Primero colocamos de cuanto queremos que sea nuestro borde (grosor), segundo va el estilo, “doble”,”solid”,etc. Y por ultimo del color que queremos que sea.
Box-shadow: Esta propiedad nos permite darle sombra a la caja. Primero va el valor del eje X, segundo del eje Y, después ponemos el tamaño del desenfoque o tamaño de la sombra, después cuanto borde va a tener (generalmente se le da 0 cero). Y por ultimo el color de la sombra.
Text-shadow: Es lo mismo pero con el texto.

Outline: Es una propiedad acortada que lo que hace es generar un borde que no ocupa espacio, se usa para enmarcar elementos.

Position: Esta propiedad lo que hace es posicionar los elementos. Osea que adquiere nuevas propiedades. Las cuales son 4, que son “top”, “left”, “right”, “bottom”. Las “top” y “left” son las mas importantes para el navegador.
Position: static= Este es el valor por defecto, osea que no esta posicionado.
Position: relative= Cuando usamos esta propiedad hacemos que el espacio de la caja se siga conservando, por mas que la caja con esta propiedad se mueva, ninguna otra va a poder ocupar su lugar. Y ademas adquieren la propiedad z-index, que lo que hace es según la jerarquía de las cajas, las superpone una a la otra.
Z-index: La propiedad CSS z-index indica el orden de un elemento posicionado **y sus descendientes. Cuando varios elementos se superponen, los elementos con mayor valor z-index cubren aquellos con menor valor.**
Z-index tiene un conflicto de padres e hijos. La unica manera de superponer un elemento padre arriba de un elemento hijo, es ponerle al elemento hijo un (z-index: -1) y al elemento padre, no colocarle un valor (z index), que no este definido. De esta manera el lemento padre se superpone al hijo.
Opacity: Es la opacidad o la transparencia que le damos a un elemento. El valor 0(cero) es invisible, y el valor 1 es totalmente visible, se puede poner valores en el medio, como 0.5 o 0.8 y así.
Position: absolute= A diferencia de relative, el espacio del elemento con position:absolute, su espacio ya no esta mas reservado. Osea que la caja o el elemento que le sigue, se superpone con este. Pero NO desaparece, sigue estando ahí. Si yo a un elemento absolute no le doy un ancho o alto, este se ajusta al contenido del mismo. Va a tomar de punto de referencia el viewport o el contenedor en caso de que este esté posicionado.
Position: fixed = Es igual que absolute pero queda fijado en un lugar. Generalmente se usa en menú, propaganda,etc. Para que el menú no nos tape lo de abajo, al body le metemos padding-top: 100px y al menu le ponemos margin: -100px, entonces el menú quedaría sin sobreponerse al resto de los elementos, y cuando bajamos el scroll este también bajaría.
Position: sticky= Es una mezcla entre relative y fixed. Porque mantiene el espacio del elemento al igual que relative. Básicamente hace que el elemento se comporte como en relaitve hasta que, debido al scroll de la pagina, el elemento es alcanzado por la parte superior del viewport.

Display: Esta propiedad modifica el comportamiento de las cajas. Y tiene varios valores.
Block: Cambia el comportamiento de un elemento en linea a bloque. Los en bloque son los elementos que ocupan todo el ancho del viewport.
Inline: Cambia el comportamiento de un elemento en bloque a en linea. Los en linea son los que ocupan el ancho según su contenido, se ajustan a este.
Inline-block: Le podemos dar un ancho y un alto. Podemos modificar las dimensiones de las cajas.

Ademas de pode poner un elemento al lado de otro.
Table: Que se comporte como tabla.
Inline-table: Que se comporte como tabla en linea.

Overflow: Es una propiedad que nos ayuda a acomodar los elementos que sobran de la caja contenedora. Como por ejemplo con un texto que sobresale de la caja.
Overflow:auto= Lo que hace es que detecta que cuando el contenido sobrepase la caja, nos da la opción de que podamos scrollear dentro de la caja.
Overflow:scroll= Pone obligatoriamente la barrita del scroll, por mas que no sea necesario.
Overflow:hidden= Este valor sin duda alguna es el más usado, sirve para ocultar o cortar todo aquel contenido que sobresalga de un contenedor. Por ejemplo, digamos que tenemos un y dentro de ese elemento hay una imagen <img/>, pero esta imagen es más grande que el elemento que la contiene, si le aplicamos overflow: hidden a el todo el contenido de la imagen que sobresalga será ocultado o recortada**.**

Float: La propiedad CSS float ubica un elemento al lado izquierdo o derecho de su contenedor, permitiendo a los elementos de texto y en línea aparecer a su costado. El elemento es removido del normal flujo de la página, aunque aún sigue siendo parte del flujo.

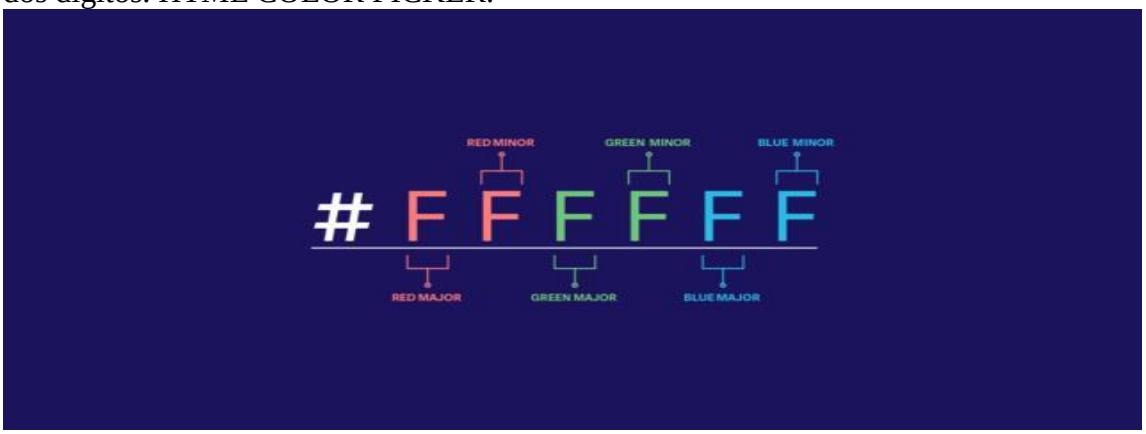
PseudoElementos: No forman parte del DOM. Podemos ver un cambio visual, pero no afecta al DOM.
::**First-line:** Solo funciona en elementos de bloque, o inline-block. Se aplica directamente sobre el selector. Este los que hace es permitirnos hacer cambios visuales específicamente en la primer linea de un texto. Ademas responde al responsive.
::**First-letter:** Funciona exactamente igual pero solo para la primer letra del texto.
::**placeholder:** Nos permite modificar visualmente el placeholder de los input text.
::**selection:** Nos permite modificar visualmente o los estilos del efecto de seleccionar texto con el mouse.
::**Before: En CSS, ::before crea un pseudoelemento que es el** primer hijo del elemento seleccionado**. Es usado normalmente para añadir contenido estético a un elemento, usando la propiedad content. Este elemento se muestra en línea con el texto de forma predeterminada.**
::after: En CSS, ::after crea un pseudo-elemento que es el último hijo del elemento seleccionado**. Es comúnmente usado para añadir contenido cosmético a un elemento con la propiedad content . Es en linea (inline) de forma predeterminada.**

PseudoClases: Son escuchas de eventos.
:hover= Escucha el evento cuando el mouse esta encima.
:link= Se usa enlaces. Si no esta visitado el elemento, podemos hacer que cambie, tenga otro color, etc.
:visited= Con este, todos los enlaces que ya visite, le vamos a poder dar un estilo para diferenciarlos.
:active= Con esta, vamos a poder cambiar los estilos de la caja o elemento en cuanto mantengamos apretado el botón del mouse. En cuanto dejemos de apretar, se vuelve al anterior estilo.
:focus= Funciona igual que el anterior, pero esta vez cuando hacemos focus. O cuando el elemento esta seleccionado.
:lang= Es una función que sirve para cambiar el estilo según el lenguaje de la pagina. En el selector luego de escribir “lang” abrimos paréntesis y le pasamos como parámetro la abreviación del idioma o lenguaje que queramos cambiar. Quedaría así [“lang(es) {“]. Ademas para que funcione, al elemento que queramos modificar, en el html le tenemos que poner la propiedad “lang:es” para seleccionar cuales están en español y así, para que css pueda hacer los cambios.
Object-fit: Es una propiedad que se usa en imágenes generalmente. Se aplica mucho para poder cambiar las resoluciones de las imágenes.

Contain: Es el valor que viene por defecto. Hace que las resoluciones de la imagen, se ajusten al contenido o a sus resoluciones originales.
Cover: La imagen se ajusta al contenedor, recortando algunos bordes. Agranda las resoluciones para ajustarse al contenido.
None: Actuá como si fuese la imagen original. Usa las propiedades por defecto.
Scale-down: Se queda con la mejor propiedad o el valor mas bajo de la resolución, entre none y contains. Ve cual de esas propiedades tira la resolución mas baja, y se queda con esa.
Object-position: Funciona para hacer que dentro de la caja la imagen nos muestre mas de un lado que de otro. Funciona con left, right, bottom y top.

Cursor: Es una propiedad que nos permite cambiar el estilo del cursor.

Colorizacion: Hay varias maneras de definir colores, por nombre, formato rgb, formato hexadecimal.
Formato RGB: Las medidas usadas en rgb van de 0 (nada) a 255 (el color entero). Y son tres datos, rojo, verde y azul, se representa de esta manera (rgb 0,0,0). según el color que queramos vamos añadiendo dígitos a cualquiera de los tres valores.
RGBA: Es igual que el rgb pero tiene un cuarto parámetro, la opacidad (con valores entre 0 y 1). La cual se la añadimos a lo ultimo de los datos. Rgba(0,0,0,1), simplemente cambia eso.
Hexadecimal: Es un sistema de 16 unidades. Con números y letras. 0-1-2-3-4-5-6-7-8-9-10-A-B-C-D-E-F. Se usa un # para comenzar a trabajar con hexadecimal, donde F es el valor mas alto.
Se pueden poner 3 o 6 unidades. También se puede añadir opacidad en este formato a lo ultimo, con otros dos dígitos. HTML COLOR PICKER.



Responsive Design y mobile first: Es un concepto que lo que hace es trabajar con las distintas resoluciones. En el que lo que hacemos es adaptar un diseño a varias resoluciones.

Etiqueta metaviewport: Básicamente, indica cómo debería de comportarse el navegador móvil a la hora de mostrar en pantalla la página web.

Conceptos del flexbox: Antes se maquetaban las paginas web con tablas, ahora se trabaja con flexbox o cajas flexibles. Flexbox requiere de dos cosas, un flex container y un flex item. El flex container va a ser el que se va a encargar de contener esos flex item. Si le damos display flex a cualquier elemento, este se va a comportar como un bloque. En donde vamos a notar los cambios es adentro del contenedor, en los items que estén adentro de este.
Flexbox tiene dos ejes, el eje vertical conocido como el cross axis y el eje horizontal conocido como el main axis.
El main axis se compone por el main-start (izquierda) y el main-end (derecha), va de izquierda a derecha. Lo mismo con el cross-axis, tenemos el cross-start (superior) y el cross-end (inferior), va de arriba a abajo.
Con flexbox lo que hacemos es invertir o cambiar las direcciones. Lo que sucede en darle display flex al contenedor, es que a los elementos de su interior se van a adaptar al contenedor y en base a

su contenido se posicionan una al lado de la otra. Pero la altura seria siempre la misma. Y si un elemento es mas grande que el otro, no es que se cambien las alturas, sucede que se ajusta el contenido para que la altura se mantenga, esto es algo que sucede por defecto con flexbox.

Flex-grow La propiedad flex-grow de CSS especifica el factor de crecimiento de un elemento flexible (que tiene asignado display:flex), en su dirección principal. El factor de crecimiento especifica qué cantidad del espacio restante dentro del contenedor flexible, debería ocupar el item en cuestión.
Flex-direction: Nos permite cambiar la propiedad o la dirección del main-axis. Es una propiedad de container o que se aplica al contenedor, pero que afectan a los items.
Flex-direction: row= Este es el valor por defecto, los items van a comportarse como una fila horizontal.
Flex-direction: column= Hace que los items en vez de comportarse como una fila, lo hagan como una columna. Cambia la dirección del main axis de horizontal a vertical.
Flex-direction: row-reverse= El main axis va a cambiar su dirección, en vez de ir de izquierda a derecha va a ir al revés, de derecha a izquierda o efecto espejo.
Flex-direction: column-reverse= Actuá igual que el anterior, lo hace columna y ademas cambia, en vez de ir de arriba a abajo de abajo a arriba, se cambia el orden.
Flex-wrap: wrap= Lo que hace es respetar el ancho de las cajas sin modificar su contenido interior, y cuando ya no entran, empieza a mandarlas para abajo.
Flex-wrap: wrap-reverse=Esto lo que hace es que en vez de irse para abajo los elementos, se van para arriba.
Flex-wrap: nowrap=Es el valor que viene por defecto.
Flex-flow: column wrap= Esta propiedad se usa para resumir el flex direction y el flex wrap.

Justify-Content: Se usa para mover el contenido.
Justify-Content: center= Esto te centra al contenido.
Justify-Content: space-around= Los elementos aparecen distribuidos uniformemente, y con un espacio igual entre ellos.
Justify-Content: space-between= Los elementos aparecen distribuidos uniformemente: al principio, en el centro y al final del contenedor flex.
Justify-Content: space-evenly= Le da es mismo margen a todo.

Align-items: Esta sirve para cuando hay una linea de items que son flex items.
Align-items: stretch= Es la propiedad por defecto.
Align-items: start= Nos coloca al inicio los items. Ademas evita que el contenido se estire a lo largo del axis.
Align-items: center= Lo que hace es que centra verticalmente los items.
Align-items: flex-end= Nos pone al final a los items.
Align-items: baseline=

Align-content: Esta se usa cuando hay mas de una linea de items. Pero las propiedades con la anterior (align-items) son las mismas, solo que se usan en distintos casos.

Propiedades de los items: Vamos a poder modificar cada item en particular. La manera de trabajar con los flex items es dándole la propiedad al item en vez de el contenedor.
Align-self: Sirve para mover el item por el axis, usando los mismos valores que e align-items.
Flex-grow: Agarra el espacio sobrante y lo reparte entre las cajas. Lo que los hace muy flexibles, ya que cuando se achique la pantalla es espacio siempre se va a repartir parejo. Pero también se

puede dar a un elemento en particular, dándole a este el sobrante total sin repartirlo. Se le ponen valores numericos (1,2,3,4) para darle mas tamaño a unos que otros.
Flex-basis: Es como el widht, pero es especial del flex. Y ademas entre el widht y este, va a darle mas importancia al flex-basis. Usa los mis valores de px, em, %, etc.
Flex-shrink: Lo que hace es decir que espacio va a tener cada una.La propiedad CSS flex-shrink especifica el factor de contracción de un flex item. Los flex items se encogerán para llenar el contenedor de acuerdo a su número flex-shrink , cuando el tamaño por defecto de los flex items sea mayor al de su contenedor flex container. Decidimos que caja va a ceder tamaño, para que las otras alcanzen el tamaño que queramos.
Flex: Es una propiedad que agrupa las tres anteriores (grow, basis, shrink) en una sola. Necesita al menos un parámetro.
Order: Es como un z-index, pero en el eje en el que apunta en el main axis. El que tenga el valor mas grande se va a posicionar en el final del lado que apunte el main axis.

Grid: Es un estilo o forma de trabajar con el layout y es un valor de la propiedad display. Y lo que se hace es trabajar el layout con grillas.
Grid container: Es el contenedor de la grilla, y es parecido al flex container. Cuando a un elemento le damos el display:grid, modificamos su interior o contenido.
Grid item: Cada elemento que forme parte del conteiner es un item o los que son hijos directos del container..
Grid cell:
Grid truck: Contiene los column y los row. Los row son filas y los column columnas. Y los dos son truck de distintos tipos.
Grid area: Son áreas que nosotros seleccionamos y a su vez son consecutivas, no pueden ser en diagonal. Las áreas las definimos nosotros.
Grid line: Esta compuesta por dos, las column line que son las lineas que son lados para definir las columnas y se comparten con la de los costados. Y las row line que son las que son superior o inferior para definir lineas.

Grid-template-rows: Se aplica a los gird container y permite crear lineas, se aplican junto con el gird container. Para decirle cuantas lineas o columnas crear, se pone las medidas de las que queramos crear. (Gird-template-rows: 100px 100px 100px) de esta manera creo tres lineas de 100 px de tamaño.
Grid-template-columns: Se aplica a los gird container y permite crear columnas. Funciona igual que el anterior.

Unidades “auto” y “fr”: Lo que hacen es darle movilidad a la hora de ajustarse a la pantalla a la grilla. Se colocan en los gird-template a la hora de crear las lineas o columnas.

Grid-row-gap: Tambien se aplican en el gird container. Es como un margin pero entre celdas. La distancia que va a haber entre celdas, respectivamente en todos los bordes.
Grid-column-gap: Tambien se aplican en el gird container. Es como un margin pero entre celdas. La distancia que va a haber entre celdas, respectivamente en todos los bordes.
Grid-gap: Resume las dos anteriores.

Grid-row-start,Grid-row-end: Estas las usamos para indicar desde que linea a que linea queremos que se extienda el elemento. Lo hacemos indicando el numero de la linea deseada.
Grid-row: Esta propiedad resume las dos anteriores, ponemos el numero de la linea que inicia seguida de una barra y luego el numero de donde termina. Ej: grid-row: 1 / 3.
Grid-column-start,Grid-column-end: Funciona exactamente igual que las anteriores.

Grid-column: Resume también las anteriores y se expresa de la misma manera. Se le puede agregar el SPAN antes del ultimo valor para indicar cuantas columnas o filas queremos que ocupe.
Repeat: Se usa a la hora de crear columnas o filas para repetir los valores en vez de colocar uno por uno. Repeat(4,1fr), de esta manera creamos 4 filas o columnas pero sin poner una por una.

Grid implicito y Grid explicito:
Grid-auto-row: Se usa para programar el grid implicito. Este es el vslor por defecto, los elementos sobrantes se desplazan hasta formar otra fila.
Grid-auto-column: Se usa para programar el grid implicito. Los elementos sobrantes se desplazan formando otra columna.
`grid-auto-flow` acepta una palabra reservada para describir el planteamiento de "cargado". Por defecto éste valor es `sparse`, pero podemos alterar ésto a `dense` que intenta cubrir todas los huecos disponibles.

Grid dinamico: Es un grid que tiene estructuras autoajustables o que cambien conjuntamente.
Por defecto grid si ve un espacio, va a tirar el resto de palabras abajo y no va a ser mas chico de lo que la palabra mas larga aguante.
Min-content: hace que el elemento valga el mínimo de lo que se puede achicar su contenido.
Max-content: Hace que el elemento valga el máximo que de el contenido.
Minmax: Nos permite decir cual es el mínimo que va a medir algo y cual es lo máximo que va a medir.
TODAS ESTAS PROPIEDADES DE GRID DINÁMICO DENTRO DE UN **REPEAT.**
Auto-fill: Va a generar la cantidad de columnas que pueda, que cumpla con las cantidades que nosotros queramos.
Auto-fit: Hace que cuando llega a un punto en que cuando no tiene mas elementos los escala.

Alineación y control de flujos:

align-content | justify-content

Define la distribución de las filas / columnas que componen el Grid Container, permitiendo alinear estos elementos de manera simétrica en su espacio vertical / horizontal.
Para definir el tipo de alineación de nuestros Grid Items podemos utilizar cualquiera de los siguientes valores:

- flex-start - ubica la cuadricula al inicio de su espacio vertical (arriba) si la propiedad es align-content o al inicio de su espacio horizontal (izquierda) si la propiedad es justify-content.
- flex-end - ubica la cuadricula al final de su espacio vertical (abajo) si la propiedad es align-content o al final de su espacio horizontal (derecha) si la propiedad es justify-content.
- center - ubica la cuadricula centrada en su espacio vertical si la propiedad es align-content o en su espacio horizontal si la propiedad es justify-content.
- space-around - ubica los Grid Items a lo largo de todo el espacio mediante un espaciado que se suma entre cada fila si la propiedad es align-content o entre cada columna si la propiedad es justify-content.
- space-evenly - ubica los Grid Items a lo largo de todo el espacio mediante un espaciado del mismo ancho entre cada fila si la propiedad es align-content o entre cada columna si la propiedad es justify-content.
- space-between - ubica los Grid Items a lo largo de todo el espacio mediante un espaciado del mismo ancho entre cada fila o columna pero solamente interno, es decir, sin añadirlo a los extremos de las filas si la propiedad es align-content o de las columnas si la propiedad es justify-content.

align-items | justify-items

Define la distribución de los elementos contenidos en los Grid Items permitiendo alinear estos elementos de manera simétrica en su espacio vertical / horizontal.
Cualquiera de estas 2 propiedades se definen en el Grid Container y les podemos asignar los siguientes valores:

- flex-start - ubica los elementos al inicio de su espacio vertical (arriba) si la propiedad es align-items o al inicio de su espacio horizontal (izquierda) si la propiedad es justify-items.
- flex-end- ubica los elementos al final de su espacio vertical (abajo) si la propiedad es align-items o al final de su espacio horizontal (derecha) si la propiedad es justify-items.
- center - ubica los elementos centrados en su espacio vertical si la propiedad es align-items o en su espacio horizontal si la propiedad es justify-items.
- stretch - los elementos cubren todo el espacio que tienen disponible verticalmente si la propiedad es align-items o horizontalmente si la propiedad es justify-items.

Align-self: Lo que hace es alinear individualmente cada una de estas verticalmente. Se aplica directamente al item.
Justify-self: Lo que hace es alinear individualmente cada una de estas horizontalmente. Se aplica directamente al item.
Place-self: Resume las dos anteriores propiedades en una. Primero colocamos el valor vertical y luego horizontal.
Order: Funciona igual que en flex. Mayor valor le damos, mayor importancia va a tener.

Grid-areas: Un área es un conjunto consecutivo de celdas. Minimo incluye dos celdas, si trabajamos con areas de 1 celda, es preferible hacerlo por celda.
Grid-template-areas: Se usa para dividir en areas como si fuesen nombres. Con la propiedad CSS grid-template-areas podemos indicar una serie de cadenas de texto delimitadas por “comillas”. Cada cadena de texto representa una fila de la rejilla. Y después, cada palabra de la cadena de texto representa una columna.
Esta vez vamos a partir de un nuevo ejemplo. El código HTML sería el siguiente:

```
<div id="wrapper">
  <header>cabecera</header>
  <main>contenido</main>
  <aside>barra lateral</aside>
  <footer>pie de página</footer>
</div>
```

Y el código CSS va a ser el siguiente:

```
#wrapper {
  display: grid;
  max-width: 1000px;
  margin: 0 auto;
  min-height: 100vh; /* esto es para que como minimo
        ocupe la altura de la ventana del navegador */

  /* indicamos cuantas filas y columnas, y el nombre de cada área */
  grid-template-areas:
    "cabecera cabecera" /* primera fila */
    "contenido barra" /* segunda fila */
    "pie pie"; /* tercera fila */
```

```
  /* indicamos tamaños de las filas y columnas */
  grid-template-rows: auto 1fr auto;
  grid-template-columns: 70% 30%;
}

/* a cada hijo le indicamos su área */
#header { grid-area: cabecera; }
main { grid-area: contenido; }
#footer { grid-area: pie; }
#sidebar { grid-area: barra; }

/* un poquito de aspecto */
#wrapper>* { padding: 10px; }
#header, #footer { background: black; color: white; }
#sidebar { background: gray; color: white; }
```

Nombrar Grid lines: A la hora de crear columnas o filas, antes de colocar las medidas, entre corchetes [ ] ponemos el nombre que le queremos dar a la linea o columna. Y luego mediante el grid column o el grid row, en ves de colocar números para extender las áreas, usamos los nombres que le colocamos a las filas y columnas.

ShortHand: Propiedades que resumen dos o mas propiedades.

Responsive design: Es una forma de trabajo o técnica para adaptar todo el contenido de una pagina web a distintas resoluciones. Y su objetivo es que se vea bien en todas las resoluciones o dispositivos. Se trabaja con estructuras flexibles (contenedores flexibles, imágenes y videos flexibles, etc).
@media: Son condicionales que se usan en consultas de medios para aplicar diferentes estilos, para diferentes tipos / dispositivos de medios.
-Ancho y alto de la ventana grafica.
-Ancho y alto del dispositivo.
-Orientación (la tableta o teléfono esta en modo horizontal o vertical).
-Resolución.
Tipos de @mediaquery:
All: Apto para todos los dispositivos.
Print: Destinado a material impreso y visualización de documentos en una pantalla en el modo de vista previa de impresión.
Screen: Destinado principalmente a las pantallas.
Speech: Destinado a sintetizadores de voz.

Operadores AND / OR :
AND: Si la pantalla tiene 400px **Y**(AND) menos de 500px que haga tal cosa. Acá se tienen que cumplir las dos condiciones si o si.
OR: Vamos a aplicarle tal cosa a una resolución mayor a 1000px **O**(OR) a los que tengan un orientación landscape. Aca se tienen que cumplir alguna de las dos condiciones, no necesariamente ambas.

Orientación:
Landscape: Es cuando es mas ancho que largo. Generalmente es para escritorio.
Portrait: Es cuando es mas largo que ancho. Generalmente es para movile.

Tenemos dos metodologías cuando trabajamos con responsive:
Movil first: Crear primero la web para móviles y luego que se vaya adaptando a resoluciones mayores. (MAS RECOMENDADO).
Destock first: Crear primero la web para escritorio y luego para resoluciones menores.
Content first: Se basa en crear primero el contenido y luego preocuparse por las resoluciones.

@media screen and (min-width: 400px) AND (max-width: 600px) { }

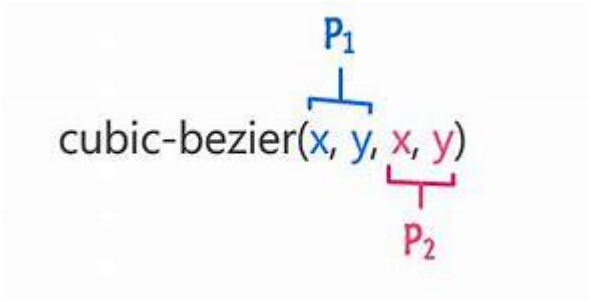
Estructura responsive basica: MOVILE CON FLEX Y ESCRITORIO CON GRID.

Transition: Es una propiedad que lo que nos permite es realizar transiciones dentro de los elementos.
Transition-property: La propiedad CSS transition-property se usa para definir los nombres de las propiedades CSS en las que el efecto de la transición debe aplicarse.
Transition-duration: Esta se usa para dar el tiempo que va a durar la transición. Cuanto tarda en pasar de una propiedad a otra.
Transition-delay: Se usa para que tarde en arrancar el evento de la transición el tiempo que queramos.
Transition-timing-function: Nos dice la curva del tiempo que va a tardar en realizarse la animación. Es la transición en función del tiempo. Como va a ir la velocidad de la aceleración del cambio de la transición en función del tiempo.
Transition: Se pueden abreviar con esta, todas las anteriores.

Animaciones: Las animaciones funcionan igual que las transiciones, solamente que necesitan de una propiedad o regla que se llama @**keyFrames** y es la manera en que se definen las animaciones. Ponemos @**keyFrames** seguido del nombre que le vamos dar a la animación y abrimos llaves. Dentro de estas tenemos dos formas de definir como se va a comportar una animación. Una de estas es **from** y **to.** Dentro de from con llaves, ponemos las propiedades que va a tener la caja cuando comienza la animación. Y dentro de to, ponemos las propiedades que va a tener cuando termine.
Y para que funcione ponemos dentro del elemento o caja en el que va a suceder la animación, la propiedad **animation-name:(nombre que le dimos a la animación).** Y luego también a este elemento, **animation-duration: (tiempo que va a durar la animación)**.
Otra forma de definir una animación es con porcentajes, los porcentajes representan en que momento de la animación van a suceder los cambios. Lo bueno de esta manera es que podemos poner varias etapas de la animación con varios porcentajes.
Animation-timing-function: Funciona igual que **Transition-timing-function.** Con los mismos valores, ease, ease-in, ease-out, ease-in-out.
Animation-iteration-count: Esta sirve para decir cuantas veces queremos que se repita la animacion, se coloca directamente el numero o el valor infinite.
Animation-direction: Sirve para cambiar la dirección de la animación, en vez de que inicie en 0% y termine en 100%, inicia y termina al revés. Tiene los valores, **normal** (no cambia nada es el valor por defecto), **reverse** (se invierte la animación), **Alternate (sirve como normal y reverse juntos, muy útil)**, **alternate-reverse** (funciona igual que alternate, pero de atrás para adelante).
Animation-fill-mode: Define como va a quedarse o cual es el modo final. Definición La propiedad CSS animation-fill-mode especifica el modo en que una animación CSS aplica sus estilos, estableciendo su persistencia y estado final tras su ejecución. Valor **none** significa que la animación se realiza una vez y después vuelve a la normalidad, o a las propiedades que tenia cuando comenzó. **Backwards** que el elemento obtenga los valores de estilo establecidos por el primer fotograma clave dependiendo de la dirección de la animación, y lo conservara durante el periodo de retraso de la animación. **Forwards** se queda como le indicamos en la ultima instancia de

la animación. **Both** va a arrancar con los valores que le indicamos al comienzo de la animación en vez de con los predeterminados antes.

CUBIC BEZIER: Es una propiedad para trabajar con animaciones y transiciones que es un valor de animation-timing-function o el transicion-timing-function. Cambiamos la velocidad de animacion con la que arranca y termina la animacion o transicion con las curvas de bezier. Animation-timing-function: cubic bezier(0, 0, 0, 0).



El primer par de X,Y en azul son las coordenadas del punto de inicio, y las que le siguen en rojo son las coordenadas del segundo punto. En Google esta el generador de curvas de bezier el cual nos da las coordenadas y las podemos copiar. CUBIC BEZIER.COM.

Transform: Es una propiedad que nos permite transformar el objeto de varias formas. Las transformaciones reciben funciones. El primer tipo de transformación es trasladándolo de un lado a otro. (Recibe valores negativos) Esto se hace con la propiedad **transform: traslateX(50px),** de esta manera se traslada 50px en el eje x**. transform: traslateX(50px)** lo mueve 50px en el eje y.
Estas **Propiedades** se pueden abreviar con **transform: traslate(x,y)**, primero colocamos el valor de x y luego de y.
Otra manera de transformarlo es escalandolo, haciéndolo mas grande o mas chico. Con la propiedad **transform: scale(2),** de esta manera hacemos que el elemento escale 2 veces su tamaño, si le ponemos un 3, seria 3 veces su tamaño, y asi. Tambien tiene **scaleX, scaleY.** Se puenden combinar las propiedades, **transform: scaleX(2) traslate(50px),** hacemos que se mueva sobre el eje x y cresca 2 veces su tamaño.
Transform:skew( ): Lo que nos permite es deformar el elemento añadiendo un cambio en grados(deg). Si queremos saber como hacer cambios de formas, una pagina que nos ayuda es CLIP PATH GENERATOR.
Para hacer animaciones es mejor el transform translate porque casi no consume recursos.

Background: Es una propiedad con varios valores distintos.
Background-image: url(url de la imagen). Nos permite añadir una imagen de internet de fondo en la pagina.
Background-size: Le podemos dar a la imagen que añadimos anteriormente otro tamaño para que se adapte correctamente en el elemento que queramos. Un buen valor es **COVER,** es el que mejor ajusta la imagen al elemento.
Background-repeat: Sirve para poner si queremos que la imagen se repita o no dentro del elemento, repeat es el valor por defecto, y si NO queremos que se repita ponemos, no-repeat.
Background-clip: Tiene como valores border-box, padding-box, content-box. Y sirve para decir desde donde queremos que se muestre la imagen, desde el borde, el padding o desde el contenido.
Background-origin: No se va a recortar, se va a originar desde donde queramos nosotros, y usa los mismos valores que el anterior.
Background-position: La propiedad background-position controla la posición en la que se muestra la imagen de fondo de un elemento. Por defecto, las imágenes de fondo que no se repiten se muestran en la esquina superior izquierda del elemento. Si la imagen de fondo se repite

horizontalmente, se muestra en la parte superior del elemento y si se repite verticalmente, se muestra en la parte izquierda del elemento.
La explicación de los valores que se pueden asignar a la propiedad background-position es compleja porque es una propiedad muy flexible. De hecho, se pueden asignar uno o dos valores elegidos entre porcentajes, medidas y las palabras clave top, right, bottom, left y center.
Background-attachment: La propiedad *background-attachment en CSS* **determina si la posición de la imagen (determinada por la propiedad background-image) es fija dentro de la ventana gráfica o si se pasará cuando el elemento (o página) que la contiene esté suspendido.** Sus valores son:
scroll: (valor por defecto) El fondo es estático en relación con el elemento en si y no se mueve con su contenido.
Fixed: La posición del fondo se fija en relación con la ventana gráfica, y si el elemento tiene la capacidad de desplazarse, el fondo no se moverá con el elemento.
Local: La imagen de fondo se fija en relación con el contenido del elemento, por lo que si el elemento tiene la capacidad de desplazarse, el fondo se moverá con el contenido del elemento.

Variables en css: Es un espacio que se almacena en memoria. Tenemos las variables locales y las globales. la variable almacena el valor de una propiedad CSS. Te recuerdo que una propiedad en CSS puede ser el color, el margin, el padding… Realmente no son variables, sino Propiedades Personalizadas, pero variables igualmente.
Variables globales: Se pueden usar desde cualquier elemento.

Declarar variables CSS

Para que tu navegador web identifique la variable hay que indicarlo. Para ello, y siempre **antes de cualquier otra regla de estilo**, utilizaremos la pseudo-clase :root, que seleccionará el <html> del documento; su raíz.

```
:root{

}
```

Dentro de él, añadiremos los nombres y valores de nuestras variables globales.

Por ejemplo, voy a **crear una variable** con un nombre descriptivo de una propiedad, a la que llamaré:

--color-principal.

Así, **solo con leerla sabré qué representa**, y mi yo futuro recordará lo que hizo mi yo pasado

```
:root{
--color-principal: red;
}
```

Un detalle importante es que las variables CSS empiezan con dos guiones: --.

Utilizar la variable CSS

Para llamar, invocar o **utilizar el valor de la variable**, sustituiremos el «valor habitual» de las propiedades, por el **nombre de la variable** a utilizar. O sea, que si normalmente para colorear un párrafo de rojo escribíamos color: red;, ahora escribiremos color: var(--color-principal);

Dando lugar a una regla tal que así:

```
:root{
--color-principal: red;
}

p{
color: var(--color-principal);
}
```

Indicar que, si pensamos usar variables CSS, tendremos que acostumbrarnos a no utilizar nombres de color en las clases, porque «lo que hoy es rojo, mañana puede ser verde». Sobre todo, si eres de esos/as a los que les gusta ir renovando el aspecto de su web de vez en cuando.

Filter: Es una forma de darle filtros. Se usa la propiedad filter, y hay varios valores para trabajar con esta. Al igual que transform, se le pueden dar varios valores distintos dentro.
None: Propiedad por defecto.
Blur(): Se usa para desenfocar una imagen o elemento, y se usan px como medida de preferencia o otras medidas fijas.
Brightness(): Es la escala de brillos, y acepta valores entre 0 y 1, o mas de uno cuando el brillo supera al normal.
Contrast(): Funciona igual que el anterior pero en vez de darle brillo le da contraste. Acepta los mismos valores numéricos.
Drop-shadow(): Es como un box-shadow pero este funciona mucho mejor con imágenes de fondo transparentes ya que da un efecto de sombras de acuerdo a la imagen. Los valores son iguales que el box-shadow.
Grayscale(): Si usamos este valor queda totalmente gris la imagen o elemento. Acepta valores numericos de 0 a 1. O valores por porcentajes.
Hue-rotate(): Lo que hace es rotar la gama de colores de la imagen o elemento. Acepta valores de grados, radianes o gradianes (por grados es mejor).
Invert(): Lo que hace es invertir los colores. Acepta valores porcentuales.
Opacity(): Acepta valores de 0 a 1, y maneja la opacidad de la imagen o elemento.
Saturate(): Hace que cada color se sature en el mas fuerte de cada unos. También acepta valores porcentuales.

Otras propiedades, funciones y selectores curiosos.
Direction: Cambia la dirección del texto. Valor por defecto **ltr(** derecha a izquierda **), rtl (**izquierda a derecha **).**
Letter spacing: Lo que hace es separar las letras el espacio que nosotros digamos. Se pueden usar medidas estaticas como px.
Scroll-behavior: Lo que hace es alivianar o no, el movimiento del scroll. Como valor tiene **smooth,** que por ejemplo a apretar en el botón de un menú, la pagina va a bajar mas lentamente hasta el contenido. **Es bueno para landing page.**
User-selected: Si a esta propiedad le ponemos el valor **none**. Decimos que el usuario no pueda seleccionar ese texto con el mouse para copiar, al que le estemos dando esta propiedad.