El id de un elemento tiene que ser único, no se debería dar un mismo id a más de un elemento como se hace con las clases. En css no se nota porque no hay problema, pero en JS va a causar problemas. Además google lo posiciona peor.

Selectores:

H2 p { } solo selecciona a los p que estén dentro del h2. Se selecciona de cajas contenedoras a hijas.

Especificidad:

¡important

Estilos en línea en html se asigna: style=”color:red” no se hace anda en css

Identificadores: ID

Clases: class=”” con el . en css

Pseudoclases: con hover x ej

Atributos : [atributo=”valor] { en css

Elementos: body header, h2

Pseudoelementos: a b

Colores según nivel de jerarquia

Cascada: Si se dan distintos valores a selectores iguales cual prevalece? El primero o el ultimo?

P {color: red}

P {color: blue}

O

{atributo=”valordelH1”} { color red}

.claseDelH1 {color:blue} estamos seleccionando al mismo elemento de distinta forma. El atributo tiene = jerarquía que la clase.

EL ÚLTIMO. Va a ser azul en ambos casos.

Metodologia BEM:

Para html se ponen los nombres así:

“nombreContenedor\_\_cajaHija--nombreUnicoDeLaCajaHija” (esto ultimo es x si las cajas hijas se llaman = o podemos poner a la caja “nieta”).

Unidades de medida:

Fijas y relativas.

Relativas: REM y EM

1 em son 16px por defecto que da el navegador. Aunque si en la caja madre se puede definir para que la hereden las hijas. Si en la caja madre especificamos x ej el tamaño de la letra (tiene que ser la misma propiedad para que se herede) en 20 px y en la hija le damos un tamaño de letra en em, cada em va a valer 20px.

Viewport: hidth y height. 100vp hacen al ancho y largo total. Vw y vh se usan.

Width: 50vw es la mitad del ancho de la pantalla

Heingth: 50vh es la mitad del largo de la pantalla.

Los % tmb son relativos, pero toman como referencia la caja contenedora y no el viewport.

Box-sizing: border-box. es para que si agregamos padding, borde,etc la caja no cambie de tamaño. Cuando se le da un paddin a una caja esta se agranda, salvo que le demos box-sizing.

Esta propiedad se suele dar con el selector \* para que se aplique a todo.

T de cajas:

Elementos en bloque: el tamaño es del ancho de su contenedor de la página si no tiene contenedor. Entonces los elementos que sigan van a ir abajo. Como se le puede dat width y heigt puede pasar que le asignemos un tamaño menor al de su contenido y éste se sobresalga de la caja. Para evitar esto podemos usar overflow.

Elementos en línea: la caja va a ser grande como el contenido. Esto permite poner elementos al lado de otros.

A los elementos en línea no se les puede dar height ni width.

Elementos inline-block: la diferencia con los en linea es que aca le podemos dar un ancho y un alto. Se va a comportar como bloque, pero el tamaño va a ser acorde al contenido como los inline. Esto permite que se puedan poner unos al lado de otros.

Border-radius: medida; es para redondear bordes.

Box-shadow: xPX xPX xPX xPX #color; las medidas es para direccionar la sombra.

Text-shadow: xPX xPX xPX #color; solo se le pueden dar 3 medidas. Si copiamos y pegamos los valores unos al lado de otros se intensifica el efecto.

Transform: rotate(40deg); rota la caja

Los bloques se posicionan con margin (1 de las formas). Con margin auto se posicionan en el medio.

POSITION:

Afecta el flujo de los elementos html. Es decir que los puede desordenar. Puede hacer que un h2 este arriba de un h1.

Caundo popsicionamos un elemento hacemos que adquieta las 4 propeidades (TLBR).

Relative: conserva el espacio de la caja por mas que se mueva. Si por ej le das un margin top de 10px y la caja se mueve para abajo, esos 10px que se mueven quedan en blanco reservados para esa caja.

Con absolut no se reserva el espacio.

Static es por defecto.

NO ES LO MISMO PONER “MARGIN –TOP/L/B/R: “, QUE PONER SOLAMENTE “TOP: “.

EL MARGIN ES PARA SEPARAR ENTRE CAJAS, Y AFECTA A LAS DEMAS CAJAS. EN CAMBIO LAS MEDIADS TOP/L/R/B SE USAN PARA POSICIONAR CAJAS SIN AFECTAR A LAS DEMAS.

Al posicionar una caja con las medidas de arriba o con “position” adquieren z-index.

La única forma de poner a uan caja hijo por encima del padre es darle zindex -1 al hijo y no darle zindex al padre.

Position absolute: es como relativ pero cambian 3 cosas.

1 No se le reserva el espacio, por lo que lo que tenga abajo va a subir.

2 Además, las cajas hijas van a tomar como referencia no a la caja madre sino al viewport cuando les demos top, button, left o rigth.

3 Ademas, aunque sea un elemento bloque, con absolut, el tamaño de la caja se va a ajustar al contenido de la caja. Con relative, no pasa.

Position fixed: para que el bloque no se mueva al scrolear. Se usa en barras de nav y en publicidades.

Position: stiky mezcal position relative (porque conserva el espacio en el dom) y cuando llegamos al punto donde le indicamos se mueve como fixex.

Display.

Modifica el comportamiento entre las cajas

Ya vimos block, in line e inline-block.

Grid y flex se comportan como bloque, pero los ítems que están adentro NO.

Con la **combinación** de grid y flex se puede hacer de todo.

Con control f y después control h podemos reemplazar muchas veces la misma palabra.

Overflow: scroll; se usa para elementos en bloque, para que su contenido se meta adentro de la caja y no sobresalga. Pone una barra de desplazamiento en el eje x e Y. Sirve para texto. Las imágenes se autoajustan, no se salen del contenido bloque.

Podemos crear un lindo efecto dándole position relative a la img, y desplazarla un poquito hacia abajo y a la derecha para que se sobresalga apenas de la caja pero que se le reserve el lugar. Para que se note el efecto la contenedora debe tener un background y un borde.

Con overflow: hidden; si la imagen esta corrida de la caja madre, la img se va a ver recortada

Y con un img:hover; y volviendo la imagen a la posición de la caja (top y left 0) se crea un efecto como que la imagen vuelve a la caja. (usar transition).

Pseudoelementos:

Se usan asi:

Elemento::pseudoelemento {

Placeholder: primero tiene que estar en html. Es el texto que va dentro de los imputs. Se le puede cambiar el color, tamaño, etc.

Selection: cambia el color de caundo seleccionar un texto.

Pseudoclases:

Son para eventos.

Se usan asi:

Elemento:pseudoclase {

Hover: caundo pasas el mouse x arriba en modo escritorio. en celu el hover funciona caundo haces clik.

Active: caundo apretas clik pasa algo.

Focus: cuando apretas en un **imput** pasa algo.

La porpiedad cursor sirve para cambiar el cursor. Hay varios valores. La lista de cursores se puede encontrar en w3school.

Con objet-fit: cover podemos hacer que una imagen se adapte a la caja contenedora, entinces si la achicamos la resol, la imagen se veria bien.

En @media, caundo ponemso max-width lo que decimos es: caundo la pantalla alcance resoluciones menores a 800px…

Para hacer que agun objeto que este al costado pase para debajo de otro objeto y se vea bien para mobile, se le debe dar display block y un width de 100%.

**NO VOLVERME LOCO CON HACER MENU RESPONSIVE SIN JS PORQUE SE HACE CON JS Y NO CON CSS.**

**Flexbox:**

**Si damos display flex a la caja contenedora, ésta se va a comportar como bloque, lo que cambia son los elementos o cajas internas.**

**Dentro de la caja tendremos los ejes cross axis (y) y main axis (x). Tienen direcciones, main-start y end, y cross-start y end.**

**Los ítems hijos se ponen unos al lado del otro. Flexbox ajusta el contenido de las cajas hijas para que todas tengan la misma altura. Para cambiar esto se trabaja con felx-direction, para modificar las direcciones axis. Se aplican al contenedor. Por defecto el contenido del contenedor se comprota como una fila, x eso se ponen unos al lado de otros. Con flex-direction: colum se van a comportar como columna.**

**El principio, si achicamos la resolución se van a encoger los ítems. Si en vez de esto, queremos que se vayan acomodando abajo, se usa flex-wrap: wrap en la contenedora.**

**Si le pones el valor wrap-reverse, pasan para arriba.**

**Si tenemos varias cajas tipo mamushka, la caja a la que le demos display flex va a ser la contenedora.**

Propiedades del contenedor:

Justify-content: propiedad que va en la caja madre y alinea a las cajas hijas en el eje x. Si usamos como valor “center” queda igual que si le pusiéramos margin:auto.

Align-items: alinea en eje Y (debe haber un height considerable). Se usa cuando hay solo una línea de ítems felx item

Align-content: alinea en eje Y. Se usa cuando hay mas de una línea de ítems. Las porpiedades son = que align ítem solo que se usan en distintas ocasiones.

Propiedades de los ítems:

Align-self: alinea el ítem seleccionado en el eje Y.

(el margin dentro de una caja flexible se comporta de una manera distinta. Ej si das un margin top auto se va hasta abajo del otro, si le das margin left auto se va a la derecha, es como que hace lo contrario. Si dan margin top y button auto se centra. Si le das margin auto: se le da las 4 propiedades y se centra en los 2 ejes.)

Flex-grow: reparte el espacio sobrante entre las cajas que hay. Le aumenta el ancho a medida que se aumenta la resolución. Se le puede dar a todos os ítems o solo a 1. Esa caja va a agrandarse o achicarse según el espacio disponible que halla. Esta bueno para hacer un **nav** si se le da a todas las cajas y en cada caja se pone una opción.

Flex-basis: es lo mismo que widht pero tiene prioridad basis.

Flex-shrink: hace que una caja ceda mas espacio que las otras cuando se achica la resolución. Por defecto todas las caja tienen valor 1, hay que poiner valor 2 o mas para que una caja empice a ceder espacio al achicar la resolución. Si la resolución alcanza no se va a achicar, va a tener la medida dada específicamente. Si se le da valor menos de 1, en vez de ceder espacio va a ocupar mas caundo no alcance la resolución para que entren todas las cajas. Con el valor 0 no va a ceder espacio.

Flex: shorthand que agrupa grow, basis y shrink.

Para cajas de texto: rezise:none hace que no se pueda agrandar.

Efecto de colores: background: linear-gradient(to bottom, transparent, color); en vez de transparent puede ser otro color tmb. Se le puede dar un % a la transparencia.

También existe radial-gradient

Min y max width sirven para responsive d, ya que al darle un tamaño máximo o minimo podemos establecer a partir de qué medida pasa el ítem para abajo al achicar la resolución (con min widht) , y qué medida va a tener cuando esté abajo (mas widht para que no baje y se agrande mucho).

GRID:

En principio va a haber tantas celdas como elementos html hayamos creado, luego hay que acomodarlos en columnas y filas.

Es aconsejable al principio darle a rodos los grid-items un border de 1px por lo menos para visualizar la grilla.

Grid cap sirve para separar las seldas y áreas. Puede ser usado para separar el articulo del aside x ej si son 2 celdas dintintas.

Puede usarse colum gap o row gap o simplemente grid-gap como shorthand.

Para crear áreas:

Grid-row y grid-colum. Son propiedades que se dan al grid ítem.

Ej con grid-row: 1/3 estamos diciendo que el área empiece en la línea 1 y termine en la 3.

Si sobra una celda en las fials y columnas (que se pone abajo), va a formar parte de lgrid implícito, que tiene otras propiedades. Sucede cuando hay mas elementos que celdas. Por defecto se crea una fila nueva abajo, aunque lo la hayamos declarado.

Si ponemos grid-auto-floy esa fila creada por defecto va a pasar a ser una columna.

Grid-auto-rows y grid-auto-column es como grid-template.row y colum, solo que para el grid implícito.

GRID dinamico:

Se debe trabajar con estructuras dinámicas y cantidades dinámicas.

Se usa minmax, mincontent, maxcontent auto-fill y auto-fit

Los primero 3 son valores que se ponen dentro de la función repeat()

Ej:

.grid\_container {

Grid-template-columns: repeat(3 minmax(min-content,200px))

o

Grid-template-columns: repeat(3 minmax(200px, 1 fr))

Alineación en grid:

Propiedades de caja madre:

Justify-item: alinea en eje X los grid ítems. Tiene los sgts valores: start, center y end, y strech x defecto

Align-item: alinea en eje Y los grid ítems. Tiene los sgts valores: start, center y end, y strech x defecto

Estas 2 siguientes sirven cuando el contenedor es mas grande que todos los ítems. Sino no hay espacio para mover los items

Align-content: maneja las filas (valores: start, center, end, space-betwin, evenly, y arround)

Justify content: maneja las columnas (valores: start, center, end space-betwin, evenly, y arround)

Propiedades de ítems hijos:

Align-self: eje Y valores: start, end, center

Justify-self: eje x valores: start, end, center

Place-self: es shorthand de lso 2 anteriores, primero se pone el valor de align y después el de justify.

Para crear áreas, en el contenedor con display grid hay que darle lo siguiente:

Grid-template-areas:

“header header header” (estas son los nombres de las clases)

“aside main main”

“aside main main”

“footer footer footer”

Y a las cajas hijas:

Grid-area: header/main/aside/footer (así se posicionarán como le marcamos en el grid-template-areas).

Luego se le debe dar medidas en el container, si tenemos 4 filas ponemos grid-templare-rows: valores en fr (ancho). Al header y al footer se le pueden dar medidas fijas. Podriamos poner “grid-template-rows: repeat(autofill, 1fr) e ir agregando mas filas en el grid-template-areas y se van a autoajustar solas.

El height puede darse con VH.

Luego de hacer toda esta estructura, adentro del div header podemos hacer una lista (ul li li li li) y a estos li darles inline-block

Responsive:

@media acreen and (min-width: 400px) Si se cumple que el width minimo es de 400px pasa x cosa y se cumple lo que esté entre {}

Si fuese max-width: cuando supere el máximo deja de cumplir la ocndición y pasa lo que esté afuera de las {}

Se pueden poner varian condiciones con and.

Transform: translatey o translatex o translate, sirve para mover elementos en ejes x e yaún afuera de su caja.

El valor scalex o y o solo scale agranda o achica el elemento

Skew(xdeg) deforma el elemento

El transform:translate es la mejor forma de ahcer que se muevan animaciones, consume menos recursos que lo visto en “animaciones”