CSS SINTAXIS

P {color : red; }

H1, h2 {

Font-family: Georgia, serif;

Color: brown;

}

Selectores:

Universal = \*

Elemento = body, p, h1

De id = solo para el elemento que tiene dicho ID (ejemplo: #primer\_parrafo, se aplica solo al parrafo con id primer\_parrafo) No se pueden repetir los ID

De clases = las clases si pueden repetirse (ejemplo: .destacado aplica a todos los elementos de clase .destacado)

FONT FAMILY

Font-family: primero pongo la que quiero. Luego dos mas genericas por si no esta la primera

Fuentes externas: voy a fonts.google.com y elijo. Me da un link para pegar en link href (en el head) o el @import en el archivo css

Para usar variables dentro de un CSS

:root {

--color-principal: violet

--fuente-principal: arial

}

Esto sirve para no tener que cambiar cada color de cada elemento. En vez del color pongo

H1 {

color: var(--color-principal);

}

Y me usa el color que defini en el :root. En este caso, violeta

FLEX

Atributos que van dentro del contenedor:

Display: flex, inline-flex...

Flex-direction: column or row

Flex-wrap: wrap = va ubicando los elementos en otras lineas para que no se salgan del contenedor. Sirve para que los elementos se compriman y vayan hacia abajo cuando no hay lugar

Flex-wrap: nowrap = si modifico el tamaño del contenedor, los elementos pueden quedar afuera ya que no se comprimen

Flex-wrap: wrap-reverse = cuando no entra el elemento se va hacia arriba en vez de hacia abajo

Flex-flow: colum wrap = abarca los dos atributos anteriores en uno (direction y wrap)

Justify-content: flex-start = los alinea hacia el inicio del eje principal

Justify-content: flex-end = los alinea hacia el final del eje

Justify-content: flex-center = centra todos los items del contenedor

Justify-content: space-between = alinea los elementos para que ocupen todo el contenedor con la misma separacion entre todos

Justify-content: space-around = divide el espacio que no ocupan los elementos, por la cantidad de elementos en el contenedor. Esto dara como resultado que en los bordes quedara menos espacio que en los elementos internos. Ej: si sobran 120 px y son 6 elementos, quedara asi

10px elemento 20px elemento 20px elemento 20px elemento 20px elemento 20px elemento 10px

Justify-content: space-evenly = todos los espacios quedan iguales

Align-items: stretch = los elementos que no tengan definidos un alto, ocuparan todo el alto del contenedor (o el ancho si estuviera flex-direction: column)

Align-items: center = se alinean en el centro del eje secundario

Align-items: flex-end = se alinean al final del eje secundario

Align-items: baseline = crea una linea imaginaria sobre la base del texto del elemento, y alinea los elementos para que los textos queden alineados

Align-content = alinea los elementos

Atributos que van dentro del elemento:

Flex-grow: 1 = permite que el elemento 1 se expanda hasta ocupar todo el espacio libre que queda en el contenedor

Flex-grow: 5 = agarra el espacio disponible, lo divide en 5 y lo asigna entre los elementos a los que le pongo el atributo grow

Flex-shrink: cuanto se va a comprimir un elemento

Flex-basis: 200px = tamaño del eje principal del elemento

Align-self: auto, center, stretch, flex-start, etc = agarro un solo elemento y lo alineo de otra manera que los demas

Order: -1 o 1 u otro numero. Por default todos los items tienen order 0, entonces si a algun item le pongo order 1 lo manda al final, ya que los demas tienen order 0. Si le pongo order -1 lo manda al principio.

GRID

.container {

dislpay: grid; // inline-grid;

}

Filas y columnas. Entre las filas y las columnas hay celdas (cell).

Areas: solo pueden ser cuadradas o rectangulares. Dentro de las celdas o de las areas van los grid items. Podemos ubicarlos donde queramos dentro de la grid.

Grid-template-columns: 200px 25% 1fr // repeat(5, 100px) 🡪 crea 5 columnas de 100 px // minmax 🡪 le pongo el tamaño minimo y maximo que se pueden expandir o achicar los items

Esto creara 3 columnas con dichas medidas cada una respectivamente. Tambien puedo usar auto

Grid-template-rows: igual al anterior pero para las filas. Las filas ya vienen creadas de acuerdo a la cantidad de items que tenga mi grid, pero puedo darles el tamaño que yo quiera. Las columnas no vienen creadas, tengo que crearlas yo la cantidad que quiera.

**Gap**: separacion entre las filas y las columnas

Row-gap: separa las filas

Column-gap: separa las columnas

Gap: separa ambas cosas

Grid-column-start: cambia el lugar desde donde comienza el item al cual le estoy dando este atributo. Es para cambiar de lugar un item, aunque haya sido creado primero que otro.

KEYFRAMES

@keyframes (**nombre de la animacion**) {

From {

Backround-color: red;

}

To {

Background-color: blue;

}

}

Luego en el div donde quiero utilizar dicha animacion uso....

Div {

Animation-name: (**nombre de la animacion**)

Animation-duration: 5s

}

BOOTSTRAP (getbootstrap.com)

Para aplicarlo en mi proyecto descargo los codigos fuentes desde la pagina y los pego en mi carpeta del proyecto. Un bootstrap para CSS y otro para JS

.container

Es una clase (class) que puedo agregar a mis elementos para que el contenido se mueva dentro de un maximo, el cual ira cambiando de tamaño segun en que pantalla lo vea (celular, tablet, etc). En este caso al agrandar la pantalla va dejando un margen y centra el elemento

.container-fluid: no deja el margen sino que ocupa todo el espacio, no importa que dispositivo se use.

Ejemplo

<div class=”container”>

contenido

</div>