**M1S3: Flexbox y CSS Grids**

**Hola, te doy la bienvenida a la tercera semana de tu Bootcamp, hasta ahora has aprendido sobre HTML y CSS, en esta oportunidad daremos inicio a Flexbox que se creó para mejorar la forma en la que se hace diseño responsive.**

**Continúa con la lectura del contenido...**

**ÍNDICE**

* [Flexbox](https://github.com/U-Camp/BOOT-M1-SEM3/blob/main/README.md" \l "flexbox-cajas-flexibles)
  + [Flex Container](https://github.com/U-Camp/BOOT-M1-SEM3/blob/main/README.md#flexbox-cajas-flexibles)
  + [Display](https://github.com/U-Camp/BOOT-M1-SEM3/blob/main/README.md#display)
  + [Flex direction](https://github.com/U-Camp/BOOT-M1-SEM3/blob/main/README.md#flex-direction)
  + [Flex wrap](https://github.com/U-Camp/BOOT-M1-SEM3/blob/main/README.md#flex-wrap)
  + [Flex flow](https://github.com/U-Camp/BOOT-M1-SEM3/blob/main/README.md#flex-flow)
  + [Justify content](https://github.com/U-Camp/BOOT-M1-SEM3/blob/main/README.md#justify-content)
  + [Align items](https://github.com/U-Camp/BOOT-M1-SEM3/blob/main/README.md#align-items)
  + [Align content](https://github.com/U-Camp/BOOT-M1-SEM3#align-content)
  + [Order](https://github.com/U-Camp/BOOT-M1-SEM3#align-order)
* [CSS Grids](https://github.com/U-Camp/BOOT-M1-SEM3#css-grids)
  + [grid-template-columns](https://github.com/U-Camp/BOOT-M1-SEM3#grid-template-columns)
  + [grid-template-rows](https://github.com/U-Camp/BOOT-M1-SEM3#grid-template-rows)
  + [grid-template-areas](https://github.com/U-Camp/BOOT-M1-SEM3#grid-template-areas)
  + [grid-column-gap](https://github.com/U-Camp/BOOT-M1-SEM3#grid-column-gap)
  + [grid-row-gap](https://github.com/U-Camp/BOOT-M1-SEM3#grid-row-gap)
  + [grid-gap](https://github.com/U-Camp/BOOT-M1-SEM3#grid-grap)

**Flexbox: Cajas Flexibles**

Flexbox nos permite generar de una manera más eficiente alineación y distribución de espacios entre elementos en un contenedor, considerando incluso cuando desconocemos el tamaño o qué tanta flexibilidad se necesitará.

Trabaja en una **sola dirección**, es decir, se mueven en filas o columnas.

Con esto, vamos a poder alterar su ancho, altura y orden para llenar los espacios disponibles.

Un contenedor flex permitirá expandir sus elementos para adaptarse al espacio disponible o reducirse para evitar salidas del contenedor.

**Display**

Esta propiedad define un contenedor flex. Los valores son: inline o block

Ejemplo:

.container {

display: flex;

}

**Flex direction**

Se establece el eje en el cual los elementos se ordenarán.

Los valores de esta propiedad son:

* Izquierda a derecha: row
* Derecha a izquierda: row-reverse
* Arriba a abajo: column
* Abajo a arriba: column-reverse

Ejemplo:

.container{

display: flex;

flex-direction: column;

}

**Flex wrap**

Los elementos dentro de su contenedor flex tratarán de integrarse en una sola línea. En caso de que el espacio no sea suficiente, abarcarán otra fila debajo de la actual para poder situarse.

Las propiedades son:

* nowrap
* wrap
* wrap-reverse

.container {

display: flex;

flex-wrap: nowrap | wrap | wrap-reverse

}

**Flex flow**

Atajo para utilizar flex-direction u flex-wrap al simultáneo.

La sintaxis para ejecutar esta propiedad sería:

flex-flow: <flex-direction> <flex-wrap>

.container{

display: flex;

flex-flow: column

}

**Justify content**

Esta propiedad define la alineación del eje x. Distribuye en un espacio específico los diferentes elementos, hasta que llega a un punto máximo o límite.

Los valores principales que puedes aplicar a esta propiedad son:

* flex-start - Alinea todos los elementos al inicio del contenedor.
* flex-end - Alinea todos los elementos al final del contenedor.
* center- Centra todos los elementos del contenedor.
* space-between - Se distribuyen todos los elementos en el espacio, comenzando y cerrando en los extremos.
* space-around - Se distribuyen todos los elementos en el espacio, con una separación de los extremos, partiendo del centro.
* space-evenly - Se distribuyen todos los elementos en el espacio, con una separación del centro, partiendo de los extremos.

.container{

display: flex;

justify-content: `center`

}

**Align items**

Esta propiedad define el comportamiento de acuerdo a como cada elemento "flex" se ordena a lo largo del eje x.

Es una versión similar al justify-content para el eje x.

.container{

align-items: stretch

}

**Align content**

Esta propiedad alinea las líneas de un contenedor flex cuando un espacio adicional en el eje x, similar a como justify-content alinea los elementos individuales.

.container{

align-items: flex-start

}

**Align order**

Esta propiedad, asigna un valor de ordenamiento a un elemento específico dentro del contenedor de flexbox.

.item{

order: 1

}

**Como te comente en semanas pasadas, nuestro cuerpo es muy importante y para ello te dejo unos ejercicios que puedes hacer mientras estudias, [presiona aquí](https://www.youtube.com/watch?v=2NSXU19AsCk)**

**¿Cómo te fue con los ejercicios?, ¿te sientes mejor?**

**Continuemos…**

**CSS Grids**

Es un sistema de rejilla bidimensional.

Durante muchos años, CSS utilizó tablas, luego floats, posiciones e inline-block, sin embargo, todo era más un conjunto de técnicas más que un sistema que nos permitiera crear web de manera más organizada.

La propuesta CSS Grids es la propuesta más sólida para resolver el problema de lienzos dentro de los sitios web.

A pesar de tener una perspectiva diferente a Flexbox, ambas pueden complementarse y utilizarse sin complicación.

**display**

Define el elemento como un contenedor de rejilla.

Establece el contexto para poder manipular sus contenidos.

Los valores que se manejan son:

* grid. Genera una rejilla en bloque.
* inline-grid. Genera un rejilla en línea.

.container {

display: grid

}

**grid-template-columns**

Define las columnas y filas de una rejilla, manteniendo espacios separados en su listado de valores.

Los valores representan el tamaño y el espacio entre ellos representa la línea de rejilla

.container {

display: grid;

grid-template-columns: 50px 100px 10px auto

}

**grid-template-rows**

Es una propiedad que especifica el número y la altura de las filas en un lienzo de rejilla.

Los valores son un lista que se separa por espacios, donde cada valor especifica la altura de cada fila.

.grid-container {

display: grid;

grid-template-rows: 100px 300px;

}

**grid-template-areas**

Se especifican espacios dentro del lienzo de la rejilla.

1. Nombras los elementos utilizando grid-area.
2. Se realiza una referencia a través de grid-template-areas.

.element{

grid-area: ejemplo;

}

.grid-container {

display: grid;

grid-template-areas: "ejemplo ejemplo"

}

**grid-column-gap**

Es una propiedad que define el tamaño del espacio entre las columnas, dentro del lienzo de la rejilla.

.grid-container {

grid-column-gap: 50px

}

**grid-row-gap**

Es una propiedad que define el tamaño del espacio entre las filas, dentro del lienzo de la rejilla.

.grid-container {

grid-row-gap: 50px

}

**grid-grap**

Es una propiedad que establece el espacio entre filas y columnas.

.grid-container {

grid-grap: 50px;

}

**Muy bien, llegamos al final de la semana 3, te invito a que vayas repasando todo el contenido visto, ya que, casi llega el momento de la presentación de tu primer proyecto y para este deberás aplicar todo lo aprendido hasta el momento.**

**Si logras obtener 80% o más en el primer proyecto estas más cerca de ganar tu primera insignia.**

**¡Sigue adelante, vas muy bien! 😉**