educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 2 - Java Web Developer

Módulo 14

Ancho y alto

Propiedades de modelo caja

`width`

Es una propiedad que permite hacer referencia **al ancho de un elemento**.

Sus valores pueden ser:

- **`px`** (unidad de medida referente a la resolución de pantalla. Dependiendo del dpi de la pantalla del usuario esta medida se visualiza de distinta manera).
- **`em`** (en base al tamaño de fuente del elemento padre).
- **`rem`** (en base al tamaño de fuente del elemento padre, pero sin heredar los tamaños de los elementos intermedios que también lo aniden).
- **`%`** (en relación al tamaño del elemento contenedor padre más inmediato).

`height`

Es una propiedad que permite hacer referencia **al alto de un elemento**.

Los valores de esta propiedad pueden ser::

- **`px`** (unidad de medida referente a la resolución de pantalla. Dependiendo del dpi de la pantalla del usuario esta medida se visualiza de distinta manera).
- **`em`** (en base al tamaño de fuente del elemento padre).
- **`rem`** (en base al tamaño de fuente del elemento padre, pero sin heredar los tamaños de los elementos intermedios que también lo aniden).
- **`%`** (en relación al tamaño del elemento contenedor padre más inmediato).

Margin

¿Qué es Margin?

Espacios

Esta propiedad fija un espacio entre elementos contiguos, se puede trabajar con **medidas de longitud absolutas**:

- `cm`
- `mm`
- `etc`

También se puede **trabajar con medidas relativas:**

- `px`
- `%`
- `em`
- `ex`

¿Cómo escribirlo?

Reglas de estilo

El **margin** se puede trabajar de diversas maneras. **La propiedad margin me permite indicar el espacio específico de cada lado:**

```
div { margin-left: 1px;
      margin-right: 10px;  }
```

De la misma forma, **podemos implementar el espacio común a todos los lados**, solo trabajando con un **valor en la propiedad margin:**

```
div { margin: 20px; }
```

El **margin** también se puede trabajar de la siguiente manera:

```
p { margin: 0;}
```

En el caso anterior, estamos trabajando con la **ausencia de margen.**

Esto es muy útil cuando queremos quitar el margen predeterminado del body, de los enunciados o de los párrafos, como en el ejemplo.

Otra manera de trabajar el **margin** es la que se muestra debajo:

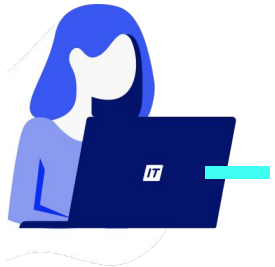
```
p { margin:10px 20px;}
```

Expresamos en la regla de estilo que el margen de arriba y abajo serán iguales en **10px** y, por el otro lado, el margen de izquierda y derecha serán iguales en **20px**.

Otra opción para establecer el **margin** es:

```
p { margin:10px 5px 20px;}
```

El resultado será similar al anterior, solo que **el primer valor se asignará al margen superior**. El margen izquierda y derecha serán iguales en 5px dejando finalmente el margen de abajo en 20px.



El **margin** también se puede trabajar así:

```
p { margin:10px 5px 20px 2px;}
```

El resultado es fácil de comprender si seguimos a las agujas del reloj como en la imagen debajo.

¿Cómo es?

En el sentido de las agujas del reloj

TOP RIGHT BOTTOM LEFT

¿Sabías que...?

Margin

Cuando queremos alinear en forma horizontal un elemento en nuestro documento, o dentro de cualquier contenedor, debemos poner el valor de **margin-left** y **margin-right** en **auto.**

```
p { margin: auto;}
```

Sin embargo, para que esto funcione, debemos indicar el **width** del elemento y cómo debe ser (un elemento de bloque, un contenedor, un párrafo). Un vínculo o un `em` no podrían ser afectados ya que son elementos de línea.

```
p { margin: auto; width: 200px;}
```

educación IT
CODE YOUR FUTURE

Padding

¿Qué es Padding?

Espacio interno

El `padding` es el espacio interior. Es decir aquel que **está entre los bordes de un contenedor y su contenido.**

Es importante entender que si un elemento, por ejemplo, tiene 400px de ancho, y colocamos 10px de **`padding`** a ambos lados -derecho e izquierdo- este elemento pasará a tener un total de 420px.

Por esa razón, muchas veces se usa la propiedad **`box-sizing`** que permite elegir si ese **`padding`** sumará o se tomará del ancho total.

```
* { box-sizing: border-box;}
```

¿Cómo se escribe Padding?

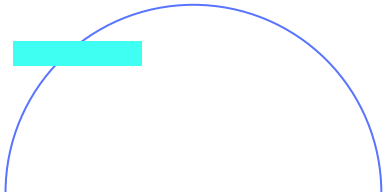
Reglas de estilo

El **padding** tiene una sintaxis igual al **margin**, es decir podemos especificar su lado:

```
div { padding-top: 1px;
      padding-left: 11px;}
```

O podemos trabajar especificando un valor igual para todos los lados:

```
div { padding: 11px;  }
```



El **padding**, al igual que el `margin`, **permite especificar en una sola línea múltiples valores,** por ejemplo, en el sentido de las agujas del reloj:

```
p { padding: 10px 5px 20px 2px;}
```

```
p { padding:10px 20px;}
```

¿Cómo es?

En el sentido de las agujas del reloj

TOP RIGHT BOTTOM LEFT

Box-sizing

Valores

La propiedad **`box-sizing`**, como mencionamos, tiene dos valores.

El valor **`border-box`** permite que el padding se tome del ancho, es decir, que si tenemos un elemento con **400px de `width`**, y colocamos **10px de `padding` a ambos lados, este elemento seguirá teniendo 400px, porque el `padding` se tomará del ancho inicial**.

En cambio el valor **`content-box`** (valor predeterminado) se encarga de sumarlo como mencionamos anteriormente.

El **selector `*` es el selector universal** y se encarga de afectar a todos los elementos por igual. Cuando **se escribe `*`, quiere decir *todos***.

Propiedad Display

Propiedad Display

`display:inline` y `display:block`

Permite indicar cómo se verán los elementos en nuestro flow o estructura.

Existen diversos **valores posibles, algunos ya predeterminados**.

Por ejemplo, si alguna vez te preguntaste por qué los párrafos o los enunciados se ubicaban debajo y no al lado - **como los `strong` o a** - es porque estos son elementos **de bloque, a diferencia de los llamados elementos de línea**.

Otra diferencia es que los **elementos de línea no aceptan propiedades como `width` o `height` y los elementos de bloque sí**.

Sin embargo, **es posible modificar** tal característica.

```
p { display: inline;}
a { display: block;}
```

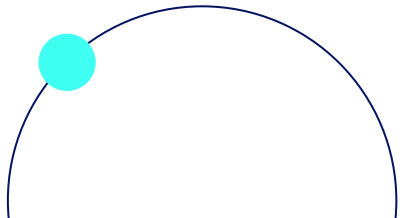
`display: block`, usos

Existen varios valores posibles, por ejemplo **`inline-block`** son aquellos elementos que, como las imágenes, pueden tener o se les puede aplicar **`width`** *y* **`height`** pero, sin embargo, **se ubican uno al lado del otro**.

```
img { width: 50px; height: 50px;}
```

Para poder lograr que **se ubiquen una debajo** de la otra, se debe transformar el **valor de `display`**:

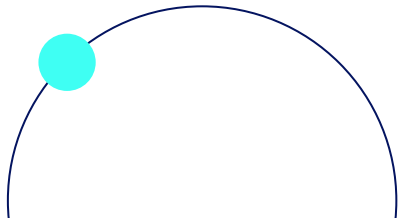
```
img  {  display: block; margin: auto;}
```



Cuándo queremos un elemento con una alineación distinta a la distribución general con respecto al **eje Y, trabajamos con una propiedad especial llamada `align-self`:**

```
#contenedor { display: flex; align-items: flex-end;}
img { align-self: flex-start;}
```

Esta propiedad tiene **como valores posibles**, los mismos valores que tiene la propiedad **`align-items`**.



display: none

El valor **none** de **display** permite ocultar un elemento.

```
p  {  display: none; }
```

De esta manera, los elementos no se visualizan pero tampoco ocupan espacio a diferencia del valor de otra propiedad similar.

```
p  {  visibility: hidden; }
```

flex

El valor **flex** de display genera un nuevo universo de trabajo. En principio, permite encolumnar elementos en nuestra **interfaz**.

```
main { display: flex;}
```

Propiedades de caja

`display: flex`

Una opción muy interesante al momento de **encolumnar elementos de nuestra estructura** es hacerlo a través de **`flexbox`**.

Para comprender esto, es mejor separar en dos partes las propiedades que veremos en la imagen de la derecha.

```
<style>
#contenedor {
    display: flex;
    width: 900px;
    padding: 2%;
    margin: auto;
    background-color: grey;
}

.items {
    background-color:pink;
    width: 50%;
    height: 100px;
    margin: 2%;
    padding: 1%;
    box-sizing: border-box;

}
</style>
```

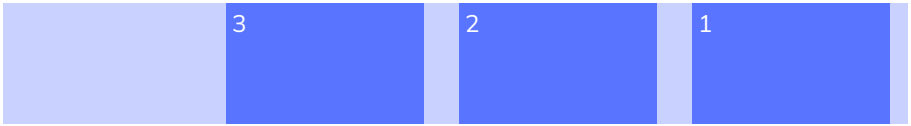
Algo importante para trabajar es implementar flex en el contenedor de los elementos **que deseamos tener encolumnados**.

A su vez, el **contenedor puede tener una dirección. Esta tiene los siguientes valores posibles y se llama:**

- **ROW**, valor predeterminado, los ítems van de izquierda a derecha.
- **ROW-REVERSE**, de derecha a izquierda.
- **COLUMN**, desde arriba hacia abajo.
- **COLUMN-REVERSE**, de abajo hacia arriba.

```
<style>
#contenedor {
    display: flex;
    width: 900px;
    padding: 2%;
    margin: auto;
    background-color: grey;
   flex-direction: row-reverse
}

.items {
    background-color:pink;
    width: 50%;
    height: 100px;
    margin: 2%;
    padding: 1%;
    box-sizing: border-box;

}
</style>
```

El ejemplo anterior, donde utilizamos la dirección de derecha a izquierda se verá de la siguiente manera:



Si hubiésemos utilizado column, se vería:

Flex: entender su funcionamiento

Los elementos contenidos en el contenedor con la **propiedad `flex`** implementada, **se van a adaptar al contenedor**.

Por más que la medida del **contenedor** hiciera que los elementos tuvieran necesariamente que desbordarse, estos se adaptan acomodando su altura **automáticamente**.

```
header { width: 100%; height: 500px; background-image: url(../imagenes/banner.jpg);
background-repeat: no-repeat; background-size: cover;
 display: flex; align-items: center;
  justify-content: center;
  text-align:center; }
```

Flex: justify-content

Es otra propiedad complementaria que determina cómo estarán los elementos dentro de nuestro contenedor con los siguientes valores con respecto a su alineación horizontal o X:

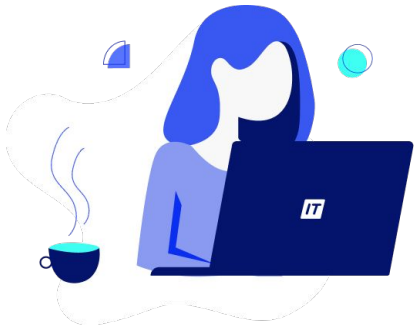
- **FLEX-START:** Valor predeterminado, están al principio del contenedor.
- **FLEX-END:** Al contrario del valor anterior, están al final del contenedor.
- **CENTER:** Los elementos están centrados en el contenedor.
- **SPACE-BETWEEN:** Genera espacios sólo entre los elementos encolumnados.
- **SPACE-EVENLY:** Genera espacios iguales antes y después de los elementos.
- **SPACE-AROUND:** Genera espacio antes y después de los elementos, pero no equivalentes como evenly sino que suma los espacios contiguos de dos elementos adyacentes.
- **STRETCH:** Estira los elementos hasta cubrir todo el espacio del contenedor.

Ejemplo con **space-around**:

`Flex: align-items`

Esta propiedad es similar a **`justify-content`** pero regula la distribución **de los elementos en el eje Y o vertical**.

- **`flex-start:`** Valor predeterminado, están en la parte superior del contenedor.
- **`flex-end:`** Al contrario del valor anterior, están en la parte inferior del contenedor.
- **`center:`** Los elementos están centrados en el contenedor.
- **`stretch:`** Estira los elementos hasta cubrir todo el espacio del contenedor.



Flex: detalles a tener en cuenta

Si los elementos tienen **`flex-direction`** `column` en vez de `row`, se invierten los ejes X e Y, por lo tanto **`justify-content`** regulará la **alineación vertical y `align-items`, la alineación horizontal.**

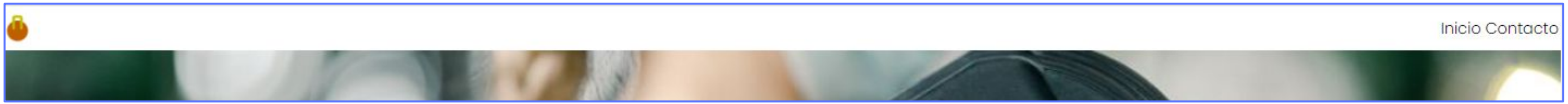
```
#contenedor {
    display: flex;
    width: 900px;
    height: 200px;
    margin: auto;
    background-color: grey;
    align-items:center;

                    }
```

Cuando queremos que un elemento tenga una alineación distinta a la distribución general con respecto al eje X, debemos trabajar con **`margin`**, por ejemplo:

1. Primero, con **`justify-content:flex-end`** ubicamos todos los elementos al final del contenedor.
2. Luego, si al primer elemento le agregamos **`margin-right: auto`**, este se colocará al inicio, es decir, **de esa manera se sobrescribe la ubicación que se le asigna con la propiedad general.**

```
nav img { width: 2%; max-width: 100%; height: auto; margin-right:auto;}
```

De esta forma, generamos que la imagen se oriente hacia la izquierda, cuando el resto de los elementos estaban con un flex **`justify-content: flex-end`**.



Si por el contrario, los elementos tuviesen el valor por defecto **`flex-start` en la propiedad `justify-content`** y al último elemento se le asigna **`margin-left:auto`**, este elemento se irá hacia final de contenedor.

Propiedad `flex-grow`

Para indicar qué **cantidad de espacio dentro del contenedor (`display:flex`)** debería ocupar **un elemento en su dirección principal (`flex-direction`)**, utilizamos la **propiedad `flex-grow` (factor de crecimiento)**.

La dirección principal depende del valor de **`flex-direction`** y puede ser la altura o el ancho del elemento. El espacio restante será la resta de: tamaño del contenedor menos el tamaño de todos los elementos restantes.

Si seteamos 1 como valor de **`flex-grow`**, todos los elementos se distribuirán el espacio del contenedor equitativamente. Si, por el contrario, alguno de los elementos tiene un valor diferente por ejemplo 2, entonces ese elemento ocupará dos veces el espacio de los otros elementos que tienen asignado 1 y así sucesivamente.

Propiedad order

Indica el **orden visual** que se utilizará para ubicar los elementos en el contenedor. En el caso de que dos elementos tengan **el mismo valor de order**, se ubicarán en **el orden en el que aparecen en el código fuente**. El valor por defecto es 1. Si a un elemento le asignamos 2, forzamos para que aparezca en segundo lugar.

Generalmente, esta propiedad se utiliza cuando, por ejemplo, en vista **`mobile`** queremos que las imágenes, en todos los casos,se ubiquen primero y luego el texto, alterando así el orden de **`viewport`** desktop inicial.

Revisión

- Repasar los conceptos aprendidos
- Trabajar con **`margin`** y **`padding`**.
- Repasar los conceptos vistos de **`display:flex`**.
- Para saber más sobre **`flex`**, copiar y pegar en la barra de dirección de tu navegador: https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- Trabajar en el **Proyecto Integrador** que encontrarás al final de este módulo.

¡Sigamos trabajando!