

Curso Profesional de CSS Grid Layout

⌚ Created	@April 12, 2022 5:51 PM
📎 Materials	
☰ Curso	
⌚ Status	Done
🔗 https://platzi.com/cursos/locucion/	https://platzi.com/cursos/css-grid-layout/

En el siguiente link, está el repositorio con las diapositivas y material complementario del curso:

<https://github.com/platzi/CSS2020#2-c%C3%B3mo-fue-pensado-css-cuando-se-cre%C3%A9>

<https://github.com/platzi/CSS2020#2-c%C3%B3mo-fue-pensado-css-cuando-se-cre%C3%A9>

Contexto histórico

Hace algún tiempo, era difícil tener control de la **altura** de elementos. También con relación a las medidas relativas o absolutas y la necesidad de adaptarse a diferentes **dispositivos**.

Luego se quiso manejar el contenido **flotante** alrededor del cual se organizan otras cosas, pero q al cambiar su tamaño se desorganizaba la composición.

También se han desarrollado arquitecturas, metodologías (BEM, Atomic Design, etc) y frameworks. Con pre y post procesadores podemos **complementar | cualificar** el uso de CSS.

En ese contexto nace **CSS Grid** para resolver necesidades identificadas en el desarrollo web. Su robustez le ha permitido consolidarse como un sistema de diseño y no simplemente como un truco más.

Grid para CSS es algo muy poderoso y ha tenido una gran aceptación por la comunidad de desarrolladores. Permite abreviar | agilizar | simplificar técnicas y trucos q se usaban en el pasado.

Control de alineamiento (antes de CSS Grid)

Antes de CSS Grid se usaban trucos como “margin”, “line-height”, “table-cell” y “positions” para alinear contenido. Aún hoy muchas personas lo usan y por tanto huyen de CSS Grid. Cada una tiene sus ventajas y desventajas.

Aquí hay un recurso para profundizar en este aspecto:

<https://www.wextensible.com/temas/css3-alinear/block.html>

<https://www.wextensible.com/temas/css3-alinear/block.html>

De esas, la profe aún hoy en día, usa “positions”. En ella se requieren propiedades como: position absolute, position relative, top, right, bottom, left, transform: translate().

	static	relative	absolute	fixed
Posicionado de acuerdo al flujo normal	✓	✓	✗	✗
Su posición final la determinan top, right, bottom, y left	✗	✓	✓	✓
Crea un nuevo contexto de apilamiento	✗	✓ (z-index != auto)	✓ (z-index != auto)	✓

La mayor limitante de estas técnicas es q se usan propiedades físicas, pero hoy debemos pensar en **propiedades lógicas**, las cuales tienen q ver con la manera en q se escribe (**modos de escritura**).

En latín escribimos de izq a der y de arriba hacia abajo, pero en otros lenguajes esto es diferente. El top de una página en Israel no será el mismo q uno nuestro.

Modos de escritura y ejes de alineamiento

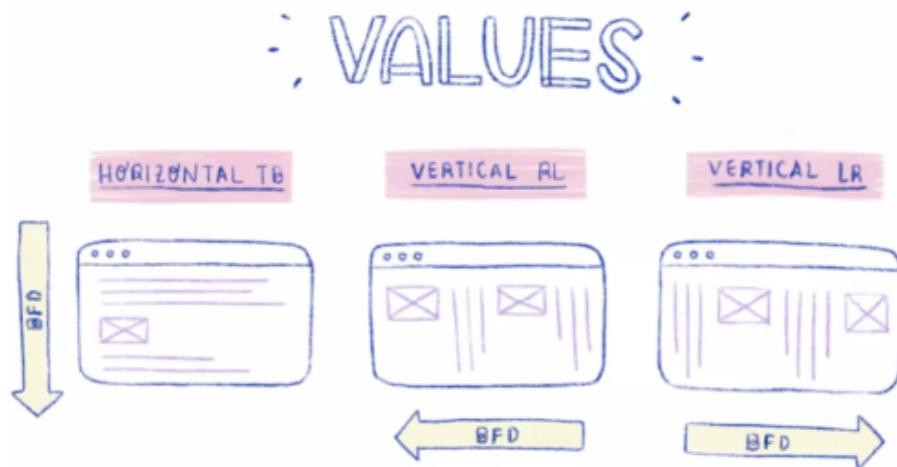
De esto se derivan las propiedades para **flexbox** y **cssgrid**.

Como escribimos de izq-der y arriba-abajo, el top-left nuestro está arriba a la izquierda.

Propiedad “writing mode”:

Tiene valores como:

- Horizontal TB: escribir de manera horizontal y al terminar línea, ir del top al bottom (arriba a abajo)
- Vertical LR: escribir de manera vertical y al terminar columna, ir de izq a derec.
- Vertical RL: de manera vertical y al terminar columna ir de derecha a izq.
- Entre otras...



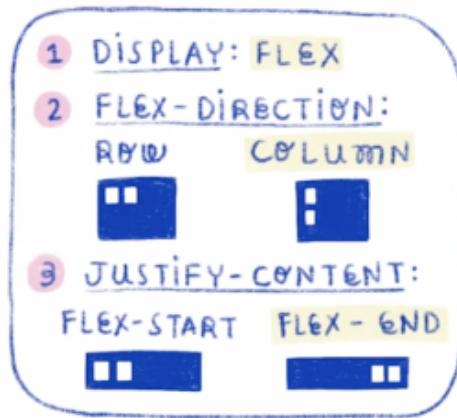
- Latin: left to right
- Arabic: right to left
- Latin + arabic: bi-directional
- Asian: vertical

Mi sitio abierto desde otro lugar del mundo, el navegador lo renderiza diferente

direction:

"display: block"= Block flow direction va de arriba a abajo. Una etiqueta "div" en HTML viene por defecto como bloque... estirándose en todo el ancho cada etiqueta y siguiendo con el resto.

"display: inline"= Inline base direction va de izquierda a derecha. Etiquetas como "input" o "span" son inline por defecto, funcionando como palabras una al lado de la otra.



ATENCIÓN!!! No todas las propiedades son compatibles en todos los navegadores, por lo cual se debe chequear.

direction - CSS | MDN

La propiedad direction se utiliza para indicar en qué dirección fluye el texto: rtl para hebreo o árabe o ltr para otros tipos de escritura. Esto tendría que especificarse como parte del documento (por ejemplo, usando el código dir

 <https://developer.mozilla.org/es/docs/Web/CSS/direction>

 mdn web docs

Link con pequeño ejercicio para entender cómo funciona esto:

writing-modes-challenge

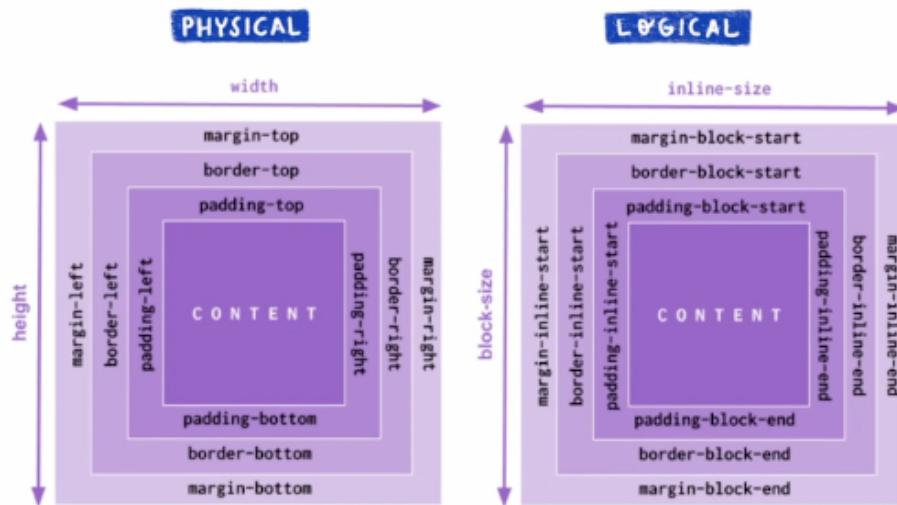
Add External Stylesheets/Pens Any URL's added here will be added as s in order, and before the CSS in the editor. You can use the CSS from another Pen by using it's URL and the proper URL extention. JavaScript

🔗 https://codepen.io/teffcode_/pen/YzGZGNw?editors=1100

Propiedades lógicas

Necesario conocer el “modelo de caja” pero abandonar la caja tradicional (la física q tiene top, right, bottom y left) y pensar en términos de una “caja lógica”.

Allí, lo **vertical** se piensa en clave de “**display: block**” (q se apilan unos sobre otros) y lo **horizontal** a través de “**display: inline**” (q se ubican uno al lado del otro).



PROPERTY	LOGICAL PROPERTY
margin-top	margin-block-start
margin-left	margin-inline-start
margin-right	margin-inline-end
margin-bottom	margin-block-end

Slides de profe

https://www.canva.com/design/DAEPwadrvmg/ldmhPG0L9qzRRhjTaYO9KQ/view?utm_content=DAEPwadrvmg&utm_campaign=designshare&utm_medium=link&utm_source=sharebutton#13

CSS Logical Properties - Ultimate Courses™

In CSS we have a lot of keywords to describe physical position: left and right, top and bottom. Let's take the next image for example: If we want to move an element on the screen we would use a container with a relative

U <https://ultimatecourses.com/blog/css-logical-properties>



Técnicas de alineamiento con Flexbox

Usamos propiedades de: “display: flex”, “justify-content” y “align-items”.

Aplican para elementos “padres”.

Aclaración en el chat: justify-content y align-items no se enfocan en posicionamiento horizontal y vertical respectivamente, ya que esto va a depender de la dirección que le demos. justify-content se enfoca en el **eje principal** y align-items se enfoca en el **eje secundario**.

A Complete Guide to Flexbox | CSS-Tricks

Our comprehensive guide to CSS flexbox layout. This complete guide explains everything about flexbox, focusing on all the different possible properties for the parent element (the flex container) and the child elements

* <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>



Reto: dibujar con CSS

Lo hice!

Link's recomendados por profe:

A Single Div

A CSS drawing experiment to see what's possible with a single div.

🔗 <https://a.singlediv.com/>

🎃 No tengas miedo a dibujar con CSS

La temporada de Halloween está a la vuelta de la esquina es por eso que dibujaremos una decoración escalofriante a tu página web con CSS.

DEV <https://dev.to/raulmar/no-tengas-miedo-a-dibujar-con-css-1ck>

🎃 No tengas miedo a dibujar con CSS

Raúl Martín • Oct 4 '20



dotCSS 2016 - Wenting Zhang - Make CSS your secret super drawing tool

Filmed at <http://2016.dotcss.io> on December 2nd in Paris. More talks on <http://thedotpost.com> As we all know, everything rendered out of CSS is in a box model...

YouTube https://www.youtube.com/watch?v=Y0_FMCji3iE

MAKE CSS YOUR
SECRET SUPER
DRAWING TOOL

Wenting Zhang
dotCSS 2016

Watch this talk on
<http://thedotpost.com>



CSS ICON -- project by Wenting Zhang

C <https://cssicon.space/#/>

sitio de ilustraciones:

Gigantic Flat Design Illustrations Bundle

I am sharing with you everything. So, from now you can use my flat design illustrations as well. I created the bundle with some of my best illustrations, and you can use it for whatever you want: commercial projects, games,

g <https://gigantic.store/gigantic-flat-design-illustration-bundle-2/>



Relaciones padre e hijo con CSS Grid

Grid se estructura a partir de líneas verticales y horizontales (**cuadrículas**).

Es necesario pensar | trabajar claramente en función de **contenedores padre** (ej: el borde de una sección) y los **ítems hijos** q estás en su interior. Es fundamental tener una estructura de etiquetas en HTML muy clara.

```
<div class="container">
  <div class="item"></div>
  <div class="item"></div>
  <div class="item"></div>
</div>
```

Un hijo a su vez puede ser padre de otras etiquetas:

```
<div class="container">
  <div class="item">
    <div class="sub-item"></div>
    <div class="sub-item"></div>
  </div>
  <div class="item"></div>
  <div class="item"></div>
</div>
```

Todos los padres deben tener la propiedad “**display: grid**” para q puedan funcionar.

Lines, tracks, cell, area

PADRES

lines

asda

track

varias celdas juntas... no tienen q estar completa la columna o la fila

area

similar track

HIJOS

cell

celdas independientes dentro de la grid.

Gutters, grid axis, grid row, grid column

PADRE

gutters

espacios entre filas y columnas

grid axis

eje horizontal: “row axis” o “inline axis”

eje vertical: “column axis” o “block axis”

HIJOS

grid row

aquí se invierte el criterio de fila horizontal. Con esto se cuentan las líneas en sentido vertical (es decir a través de la columna)

grid column

también se invierte criterio. Se cuentan las líneas en sentido horizontal (a través de la línea)

grid axis

Grid se estructura a p

La grid debe **hacerse completa**, así hayan elementos q se troquen... todas las líneas deben ir de borde a borde... las columnas y filas deben ser completas.

Recomendable **separar por componentes** para usar grids distintas (como el ejercicio q hice de abuelo / padre hijo)

Proyecto creativo

Ideas (Paula Scher):

Diseñando el mundo

Libro de autor. Editado por María José Taboada

⌚ https://issuu.com/dg4editorial/docs/paulascherfinal_majotaboada



Esta señora hace lo q estoy buscando!

Web Design Experiments by Jen Simmons

Demos for 2019 Intro to CSS Grid Study of Flexibility Study of Overlap
Return of the 1990s Multiple Column Layout works has worked in browsers forever - including IE10+. There are some remaining bugs however, and we
<https://labs.jensimmons.com/>



explicitly placed

Posibles pistas para lo q quiero hacer:

Rotated Grid

🔗 <https://codepen.io/cssgrid/pen/OgQWje?editors=1100>



Imágenes gratuitas:

Freepik| Recursos gráficos para todos

Millones de recursos gráficos gratis ✓ Vectores ✓ Fotos de stock ✓ PSD ✓
Iconos ✓ Todo lo que necesitas para tus proyectos creativos

🌐 <https://www.freepik.es/>



<https://pixabay.com/es/>

Idea inicial

Jugar con textos en diferentes orientaciones, con bordes de ciertas partes de la cuadrícula.

Girar composición

Alto contraste de fuentes y fondo en color vivo.

Puede haber una imagen o vector de deportista corriendo, o previo a correr o un escalador.

Textos para ubicar:

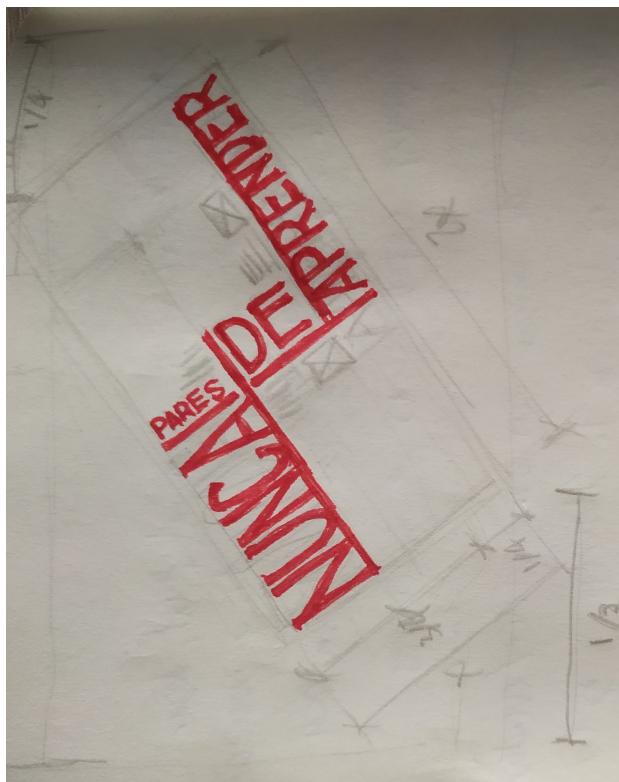
NUNCA PARES DE APRENDER

Reinventarse, explorar nuevas posibilidades, cambiar de carrera, transformación digital, descubrir habilidades, desarrollar nuevas competencias, aprender haciendo, generar nuevas conexiones.

Disciplina, autogestión, constancia, concentración, empoderamiento,

Platzi.

Bocetos:



Resultado final:



Proyecto profe

Eligiendo contenedor

¿"display: grid" o "display: inline-grid"?

Existen dos valores: "**inner**" (cómo se comportan sus hijos directos) o "**outer**" (valor externo: cómo se comporta en relación a otros)

Cuando seleccionamos "display: grid" estoy refiriéndome a ("display:**block** grid"), le estoy pidiendo q se comporte como un bloque por fuera (que ocupe toda la pantalla), pero q internamente sea una grid (un in-line q pueden tener otros elementos siempre q haya espacio).

Al ser **bloque** puede tener atributos como: margin, padding, width y height. Resumen comparativo en chat del grupo:

Elementos in-line	Elementos block
-------------------	-----------------

Se posicionan horizontalmente en línea con otros elementos (uno al lado del otro)	Forman un bloque y se posicionan de vertical con un nuevo salto de linea
Ancho se define con base al contenido del elemento	Ancho es el máximo que puede tomar dentro de su elemento contenedor
Altura está en función del contenido del elemento	Altura cambia con base al contenido que posea
Solo puede contener elementos tipo in-line	Puede contener elementos in-line y block
Alto y ancho fijos (no se puede ajustar en CSS)	A través de CSS se puede modificar altura y ancho
ej de elementos in-line: <a> <iframe> <code> <input> <label> <button>	ej de elementos block: <p> <h1> al <h6> <div> <form> etiquetas como header, footer, section

Creando filas, columnas y espaciado

Para construir columnas usamos “grid-template-columns: ...”. Una retícula con 5 columnas y 5 columnas puede hacerse así:

```
grid-template-columns: 20px, 20px, 20px, 20px, 20px;
grid-template-rows: 20px 20px 20px 20px;

/* o de esta otra manera: */
grid-template-columns: repeat(5, 20px);
grid-template-rows: repeat(5, 20px);
```

Lo anterior puedo sintetizarlo en una sola línea:

```
grid-template: repeat(5, 20px) / repeat(5, 20px)
/* el primer valor es el de las filas
y el segundo el de las columnas */
```

Para generar espacio entre columnas y filas:

```
column-gap: 5px;
row-gap: 5px;

/* se puede sintetizar en una sola: */
gap: 5px 5px;
```

Para crear áreas debo asignar nombre a cada celda. Es un nombre arbitrario según lo q yo quiera:

```
grid-template-areas:  
  "header header header header header"  
  "header header header header header"  
  "main main . . sidebar sidebar"  
  "main main . . sidebar sidebar"  
  "footer footer footer footer footer"
```



Alineamiento de contenedor con sus hijos

Estamos hablando si queremos q estén arriba, abajo, derecha, izquierda o centro.

No es posible crear una alineación diferente para cada hijo desde el contenedor.

Hay dos conceptos claves: justify y align.

- “justify”: alude al **inline axis** (row axis)... q sino se cambiara el lenguaje, sería el sentido horizontal.
 - Si asigno el valor “start” sería a la izquierda.
 - Si asigno el valor “end” sería la derecha.
- “align”: alude al **block-axis** (column axis)... q sino cambiara el lenguaje, sería el sentido vertical.
 - Si asigno el valor “start” sería arriba
 - Si asigno el valor “end” sería abajo

Las propiedades q usamos se dividen en dos grupos:

- Valores dentro de la grid:
 - “justify-items”: posibles valores son start, end, center o stretch
 - “align-items”: igual q el anterior solo q va en el otro eje
 - “place-items”: nos permite mezclar align-items y justify-items

```
place-items: <align-items> / <justify-items>;
```

- Alinea la grid en su contenedor:
 - “justify-content”: start, end, center o stretch, space-around, space-between, space-evenly.
 - “align-content”: start, end, center, stretch, space-around, space-between y space-evenly.
 - “place-content”: mezcla ambos.

Generación automática de track's

El track es la unión de 2 o más celdas.

Util para dejar configurado el frontend ante un contenido q varía en el backend (desde una API... una base de datos q usamos para lo q se ve en el front).

Se le llama “grid implícita”... crea filas y columnas si se llegan a necesitar con anchos sin tamaño.

Se usan dos propiedades. Una es “grid-autocolumns”. En el caso del ejemplo, crea columnas de 60px:

```
.container {
  grid-auto-columns: 60px;
}
```

La otra es “grid-auto-flow”... define grid's q se van creando de manera automática, sin tener q saber previamente qué cantidad de items se deben ubicar:

```
.container {  
    grid-auto-flow: row | column |  
        row dense | column dense
```

FUNCIONES: repeat(), minmax() y fit-content()

La de repeat ya la escribí más arriba en este cuaderno.

minmax(): Surge como solución cuando los elementos se des-escalan dependiendo del tamaño de una pantalla. Se fija cuál es el ancho mínimo y máximo q puede tener un elemento, independiente del dispositivo q se use.

```
.container {  
    display: grid;  
    grid-template-columns: repeat(3,  
        minmax(200px, 300px));
```

Con lo anterior le digo q el elemento no sea menor a 200px ni mayor a 300px. En caso de q sea menor o mayor, la grid implícita, bajará o subirá los elementos q sean necesarios para mantener la composición.

auto-fit(): ubica columnas en el tamaño disponible del navegador

fit-content (): organiza un contenido establecido.... después entenderé y exploraré mejor

Alineamiento de hijos

Identifico el elemento en CSS y le digo en qué lugar quiero q se ubique, tanto en columnas como líneas

```
/* Esta es la ubicación de columnas*/  
.item-one {  
    grid-column-start: 1;  
    grid-column-end: 2;  
}  
/* ahí le estoy diciendo q el elemento  
lo ubique entre la línea 1 y 2 de la grid*/
```

```
/* o abreviando: */
.item-one {
  grid-column: 1 / 2;
}

/* Aquí defino alineación en filas*/
.item-one {
  grid-row-start: 1;
  grid-row-end: 3;
}

/* abreviando el anterior: */
.item-one {
  grid-row: 1 / 3;
}
```

Posteriormente lo ubico en fxn del nombre q le haya dado a esa parte de la grid

```
.container {
  display: grid;
  grid-template-columns: repeat(2, 1fr);
  grid-template-areas:
    "card-1 card-2"
    "card-1 card-3";
}
/* en la anterior cree la grid y asigné
nombres a cada área*/

.item-one {
  grid-area: card-1;
}
```