

# UD 6

# CSS Responsive

DISEÑO DE INTERFACES WEB

**Técnico de Grado Superior Desarrollo de Aplicaciones Web**

**2024-25**

# Resultados de aprendizaje

## 2. Crea interfaces Web homogéneos definiendo y aplicando estilos.

- a) Se han reconocido las posibilidades de modificar las etiquetas HTML.
- b) Se han definido estilos de forma directa.
- c) Se han definido y asociado estilos globales en hojas externas.
- d) Se han definido hojas de estilos alternativas.
- e) Se han redefinido estilos.
- f) Se han identificado las distintas propiedades de cada elemento.
- g) Se han creado clases de estilos.
- h) Se han utilizado herramientas de validación de hojas de estilos.
- i) Se ha utilizado y actualizado la guía de estilo.

# Contenidos

- Introducción a layouts y CSS Responsive
- ¿Por qué Flexbox?
- El modelo flexible en profundidad
- Propiedades para el contenedor flexbox
- Propiedades para los elementos hijos en Flexbox
- CSS Grid: Introducción y fundamentos

# Introducción a layouts y CSS *responsive*

El diseño responsive *asegura* que las páginas web se adapten y funcionen adecuadamente en dispositivos de diferentes tamaños de pantalla (ordenadores, tablets, móviles, o el cacharro que sea)

¿Cómo hemos enfocado esto hasta ahora?

- **Usando unidades relativas**

Permiten ajustar el contenido proporcionalmente al *viewport* o al contenedor.

Ejemplo: %, vw, vh, em, rem

- **Usando media queries**

Reglas que aplican estilos específicos según características del dispositivo.

# Introducción a layouts y CSS *responsive*

Unidad	Nombre	Equivale a
cm	Centímetros	1cm = 96px/2,54
mm	Milímetros	1mm = 1/10 de 1cm
Q	Cuartos de milímetros	1Q = 1/40 de 1cm
in	Pulgadas	1in = 2,54cm = 96px
pc	Picas	1pc = 1/6 de 1in
pt	Puntos	1pt = 1/72 de 1in
px	Píxeles	1px = 1/96 de 1in

## Unidades de longitud absoluta

Salvo pixels, el resto de valores son útiles en salidas para formatos impresos más que en pantalla

Unidad	Relativa a
em	Tamaño de letra del elemento padre, en el caso de propiedades tipográficas como <code>font-size</code> , y tamaño de la fuente del propio elemento en el caso de otras propiedades, como <code>width</code> .
ex	Altura x de la fuente del elemento.
ch	La medida de avance (ancho) del glifo "0" de la letra del elemento.
rem	Tamaño de la letra del elemento raíz.
lh	Altura de la línea del elemento.
vw	1% del ancho de la ventana gráfica.
vh	1% de la altura de la ventana gráfica.
vmin	1% de la dimensión más pequeña de la ventana gráfica.
vmax	1% de la dimensión más grande de la ventana gráfica.

## Unidades de longitud relativa

# Introducción a layouts y CSS *responsive*

Estas técnicas están limitadas.

¿Qué ocurre con la disposición de los elementos en la pantalla?

Se han ido usando técnicas para definir el *layout* como:

- **Floats y blocks**  
Diseños basados en flotación de elementos (`float: left/right`)  
Causan problemas con la alineación vertical y el flujo de los elementos.
- **inline-blocks**  
Usados para alinear elementos en fila.  
Requieren trucos como eliminar espacios en blanco entre elementos.
- **frameworks como Bootstrap:**  
Aunque útiles, requieren clases adicionales y pueden limitar personalización.

```

<style>
#div1 {
  width: 50%;
  height: 100px;
  float: left;
  background-color: lightblue;
}

#div2 {
  width: 50%;
  height: 100px;
  float: right;
  background-color: lightgreen;
}
</style>
</head>

<body>

<div id="div1">Columna 1</div>
<div id="div2">Columna 2</div>

</body>

</html>

```

Columna 1

Columna 2

Los elementos *block* se apilan uno debajo de otro de forma predeterminada.

Los elementos ***float*** permiten colocar cosas como imágenes al lado del texto.

## PROBLEMAS...

- Control limitado del espacio entre elementos
- Difícil alinear elementos verticalmente
- Propenso a errores de control de flujo entre contenedores

¿Recordáis el clearfix?

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...



```

<style>
#div1 {
  display: inline-block;
  width: 30%;
  background-color: lightcoral;
}

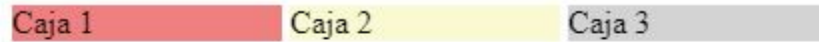
#div2 {
  display: inline-block;
  width: 30%;
  background-color: lightgoldenrodyellow;
}

#div3 {
  display: inline-block;
  width: 30%;
  background-color: lightgray;
}
</style>
</head>

<body>

  <div id="div1">Caja 1</div>
  <div id="div2">Caja 2</div>
  <div id="div3">Caja 3</div>

```



Se introdujo **inline-block** para permitir colocar elementos uno al lado del otro sin usar float.

### PROBLEMAS...

- Los espacios en blanco entre elementos afectan el diseño.
- Difícil de manejar alineaciones y distribuciones.

```

</body>

```



```

<style>
  #div1 {
    position: relative;
    width: 100px;
    height: 100px;
    background-color: lightblue;
  }

  #div2 {
    position: absolute;
    top: 10px;
    left: 10px;
    width: 50px;
    height: 50px;
    background-color: lightcoral;
  }
</style>
</head>

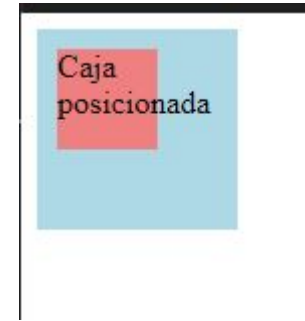
<body>

  <div id="div1">
    <div id="div2">
      Caja posicionada
    </div>
  </div>

</body>

</html>

```



También se utiliza **position** con valores absolute o relative.

### PROBLEMAS...

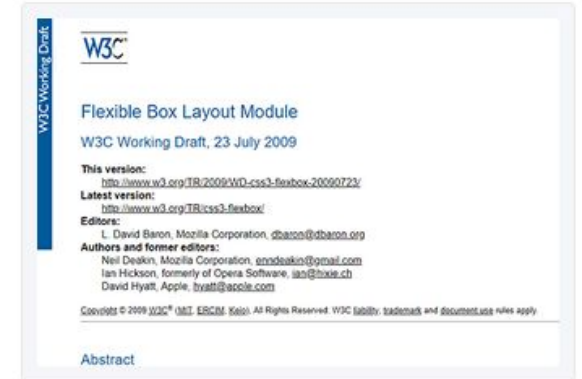
- Es engorroso
- Es difícil de gestionar con muchos elementos

2009

July 23<sup>rd</sup>

## CSS Flexible Box Layout

W3C issued the first proposal of the CSS Flexible box layout (Flexbox) specification. Flexbox introduces a new way of creating web layout, an easier alignment of elements and a better distribution of space with respect to the device's display resolution. Flexbox features are currently supported in most major browsers.



2010

May 25<sup>th</sup>

## Responsive Web Design

Web designer **Ethan Marcotte** published an article entitled **"Responsive Web Design"** in the online magazine A List Apart. The author describes a new way of styling HTML documents which allows for an optimization of website content display with regard to resolution or display size. Basic responsive web design techniques include fluid grid, flexible images, and CSS3 module media queries.



# ¿Por qué Flexbox?

...es una abreviatura de "Flexible Box Layout"

Fue diseñado para:

- Facilitar la alineación y distribución de espacio entre elementos
- Ajustarse automáticamente a distintos tamaños de pantalla
- Resolver problemas que las técnicas tradicionales no pueden
  - Reorganización sencilla de elementos sin modificar el HTML.
  - Distribuir el espacio sobrante
  - ...

Propiedades clave del modelo:

- Contenedor (**padre**) define el comportamiento general
- Elementos (**hijos**) se distribuyen y alinean según las reglas del contenedor

# El modelo flexible

- Paso 1:  
Activar Flexbox con `display: flex` en el contenedor

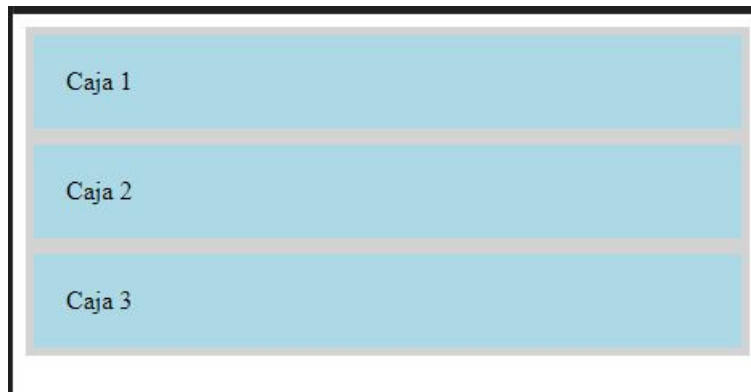


Los elementos hijos ahora están dispuestos en una fila automáticamente.

```
<style>
  .flex-container {
    display: flex;
    background-color: lightgray;
  }
  .flex-item {
    background-color: lightblue;
    padding: 20px;
    margin: 5px;
  }
</style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item">Caja 1</div>
    <div class="flex-item">Caja 2</div>
    <div class="flex-item">Caja 3</div>
  </div>
</body>
```

# El modelo flexible

- Paso 2:  
Dirección del eje flex-direction
  - row (predeterminado)
  - row-reverse: De derecha a izquierda
  - column: De arriba hacia abajo
  - column-reverse: De abajo hacia arriba



```
<style>
    .flex-container {
        display: flex;
        flex-direction: column;
        background-color: lightgray;
    }
    .flex-item {
        background-color: lightblue;
        padding: 20px;
        margin: 5px;
    }
</style>
</head>
<body>
    <div class="flex-container">
        <div class="flex-item">Caja 1</div>
        <div class="flex-item">Caja 2</div>
        <div class="flex-item">Caja 3</div>
    </div>
</body>
```

# El modelo flexible

- Paso 3:  
Alineación con justify-content
  - flex-start (predeterminado)
  - flex-end
  - center
  - space-between  
Espacio igual entre elementos
  - space-around  
Espacio igual alrededor de los elementos



```
<style>
    .flex-container {
        display: flex;
        justify-content: space-around;
        background-color: lightgray;
    }
    .flex-item {
        background-color: lightblue;
        padding: 20px;
        margin: 5px;
    }
</style>
</head>
<body>
    <div class="flex-container">
        <div class="flex-item">Caja 1</div>
        <div class="flex-item">Caja 2</div>
        <div class="flex-item">Caja 3</div>
    </div>
</body>
```

# El modelo flexible

- Paso 4:  
Alineación con align-items
  - stretch (predeterminado)  
Los elementos llenan el espacio.
  - flex-start
  - flex-end
  - center



```
<style>
    .flex-container {
        display: flex;
        align-items: center;
        height: 200px;
        background-color: lightgray;
    }
    .flex-item {
        background-color: lightblue;
        padding: 20px;
        margin: 5px;
    }
</style>
</head>
<body>
    <div class="flex-container">
        <div class="flex-item">Caja 1</div>
        <div class="flex-item">Caja 2</div>
        <div class="flex-item">Caja 3</div>
    </div>
</body>
```

# El modelo flexible

## Reto 1:

Crear una fila de tres elementos que estén centrados tanto **horizontal** como **verticalmente** en un contenedor.

## Reto 2:

Crear un diseño con elementos **distribuidos uniformemente** a lo largo de una columna.

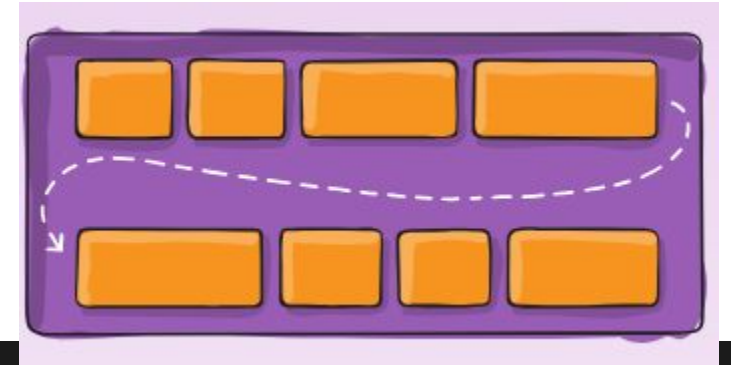
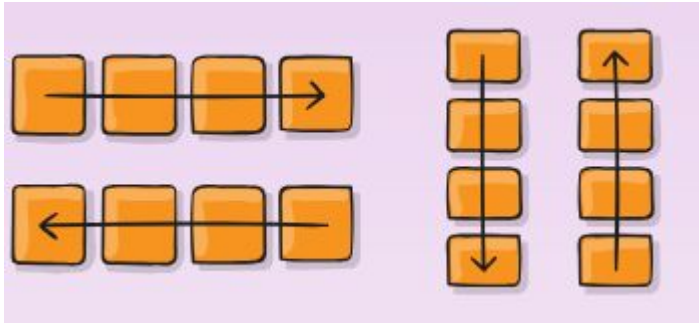
## Reto 3:

Experimentar con **flex-wrap** para permitir que los elementos salten a la siguiente fila cuando no hay suficiente espacio.



# Propiedades para el contenedor flexbox

```
.container {  
  display: flex; /* or inline-flex */  
}
```



```
.container {  
  flex-direction: row | row-reverse | column | column-reverse;  
}
```

```
.container {  
  flex-wrap: nowrap | wrap | wrap-reverse;  
}
```

```
.container {  
  flex-flow: column wrap;  
}
```

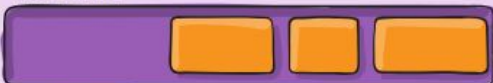
# Propiedades para el contenedor flexbox

```
.container {  
  justify-content: flex-start | flex-end | center | space-between  
}
```

flex-start



flex-end



center



space-between



space-around



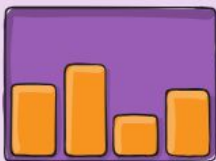
space-evenly



flex-start



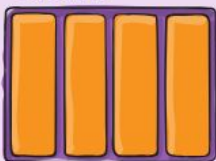
flex-end



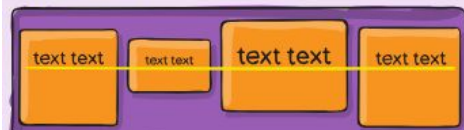
center



stretch

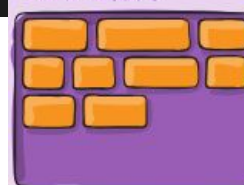


baseline



```
.container {  
  align-content: flex-start | flex-end | center | space-between  
}
```

flex-start



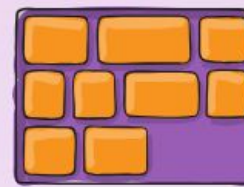
flex-end



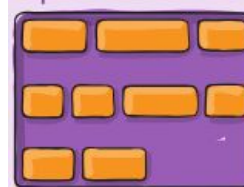
center



stretch



space-between



space-around



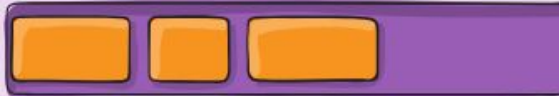
Esta propiedad solo tiene efecto en contenedores flexibles de varias líneas, donde flex-wrap está configurado en wrap o wrap-reverse. Un contenedor flexible de una sola línea (es decir, donde flex-wrap está configurado en su valor predeterminado, no-wrap) no reflejará align-content.

```
.container {  
  align-items: stretch | flex-start | flex-end | center | baseline  
}
```

# Propiedades para el contenedor flexbox

```
.container {  
  display: flex;  
  ...  
  gap: 10px;  
  gap: 10px 20px; /* row-gap column gap */  
  row-gap: 10px;  
  column-gap: 20px;  
}
```

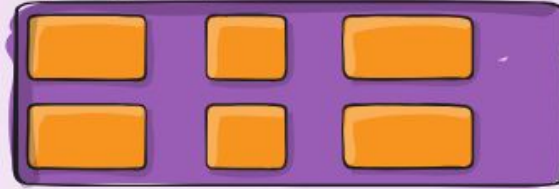
gap: 10px



gap: 30px



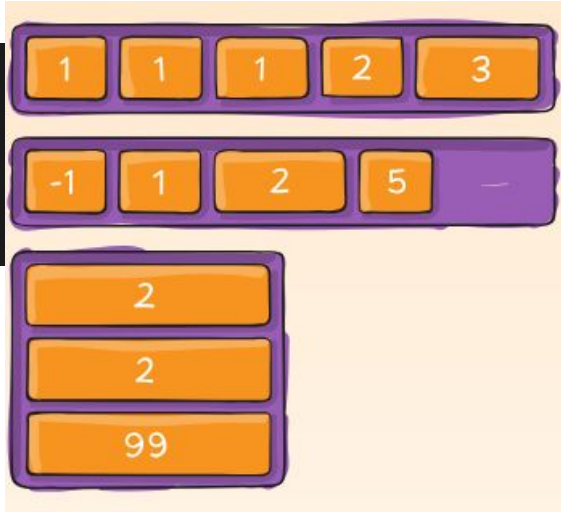
gap: 10px 30px



# Propiedades de los hijos

```
.item {  
  order: 5; /* default is 0 */  
}
```

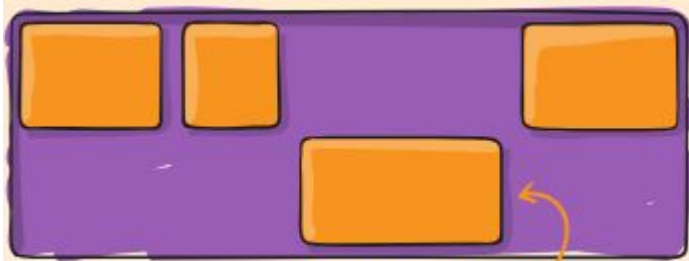
Elementos con menor número aparecen primero.



```
.item {  
  align-self: auto | flex-start | flex-end | center | baseline;  
}
```

float, clear y vertical-align no tienen efecto sobre un elemento flexible.

flex-start



flex-end

```
.item {  
  flex-grow: 4; /* default 0 */  
}
```



```
.item {  
  flex-shrink: 3; /* default 1 */  
}
```

```
.item {  
  flex-basis: | auto; /* default auto */  
}
```

```
.item {  
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]  
}
```



# CSS Flexbox

a guide from  
\* CSS-TRICKS



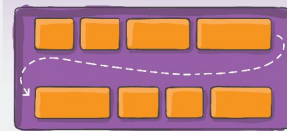
```
.container {
  display: flex; /* or inline-flex */
}
```

## flex-direction



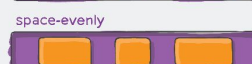
```
.container {
  flex-direction: row | row-reverse |
  column | column-reverse;
}
```

## flex-wrap



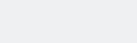
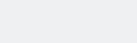
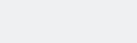
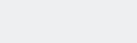
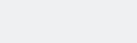
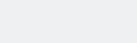
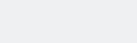
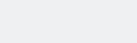
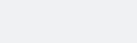
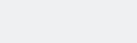
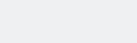
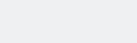
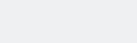
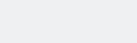
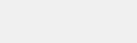
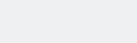
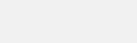
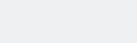
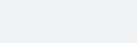
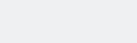
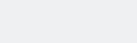
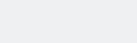
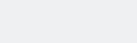
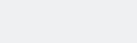
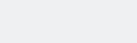
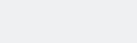
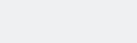
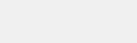
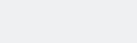
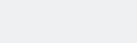
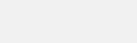
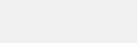
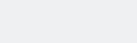
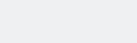
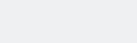
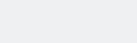
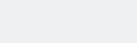
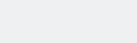
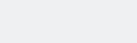
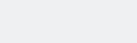
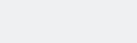
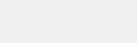
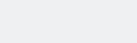
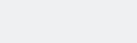
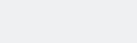
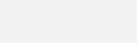
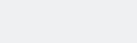
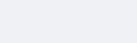
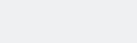
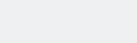
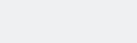
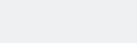
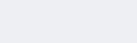
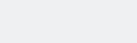
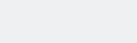
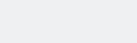
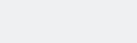
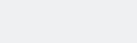
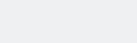
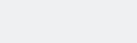
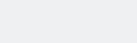
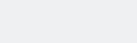
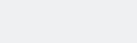
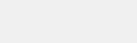
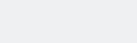
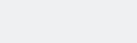
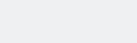
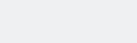
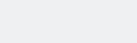
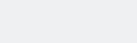
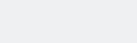
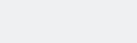
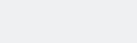
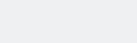
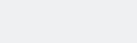
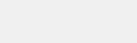
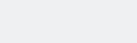
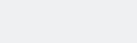
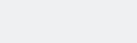
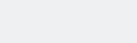
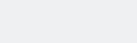
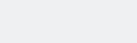
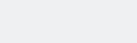
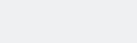
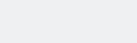
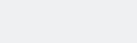
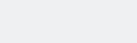
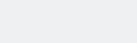
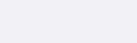
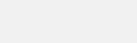
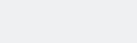
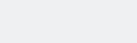
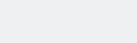
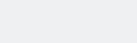
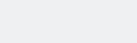
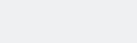
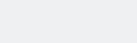
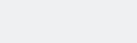
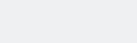
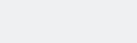
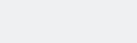
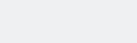
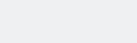
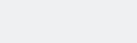
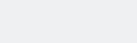
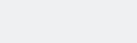
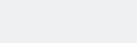
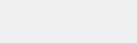
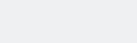
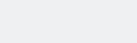
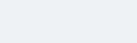
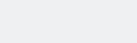
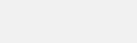
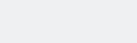
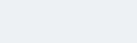
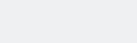
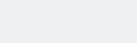
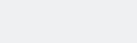
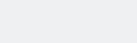
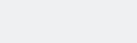
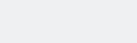
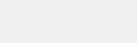
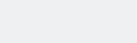
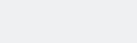
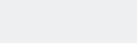
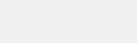
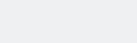
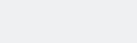
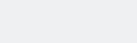
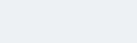
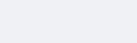
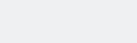
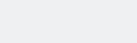
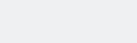
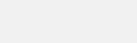
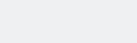
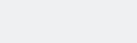
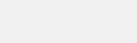
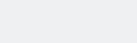
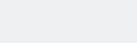
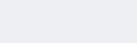
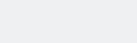
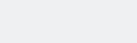
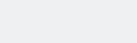
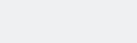
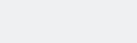
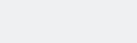
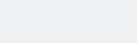
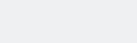
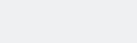
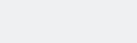
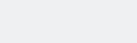
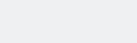
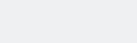
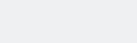
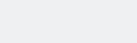
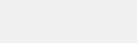
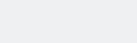
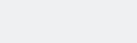
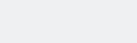
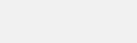
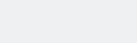
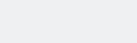
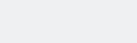
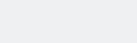
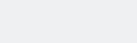
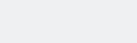
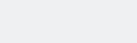
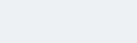
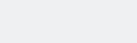
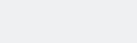
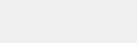
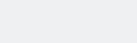
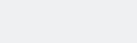
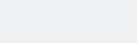
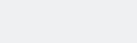
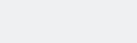
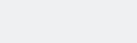
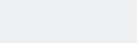
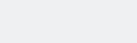
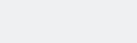
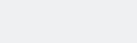
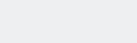
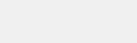
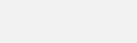
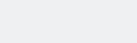
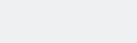
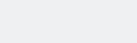
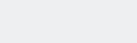
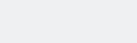
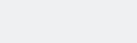
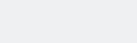
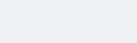
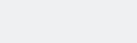
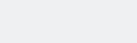
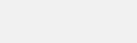
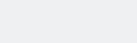
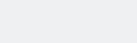
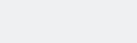
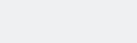
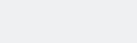
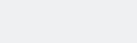
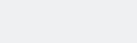
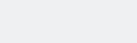
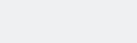
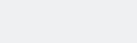
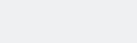
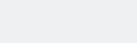
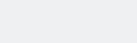
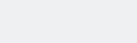
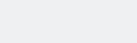
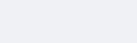
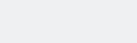
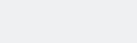
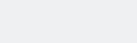
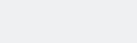
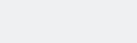
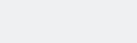
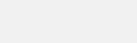
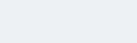
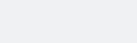
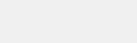
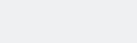
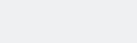
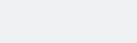
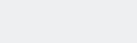
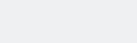
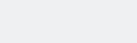
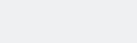
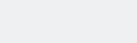
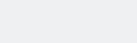
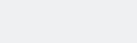
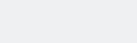
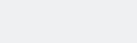
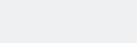
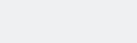
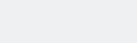
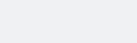
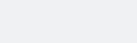
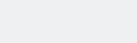
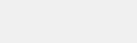
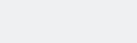
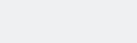
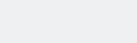
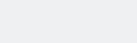
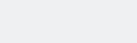
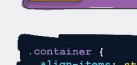
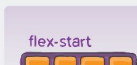
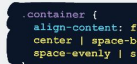
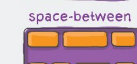
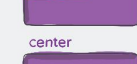
```
.container {
  flex-wrap: nowrap | wrap | wrap-reverse;
}
```

## justify-content



```
.container {
  justify-content: flex-start | flex-end |
  center | space-between | space-around |
  space-evenly;
}
```

## align-content



# Flexbox

## **Reto 1:**

Crea un contenedor con 8 elementos.

En pantallas grandes, deben mostrarse en una fila.

En pantallas medianas, en dos filas.

En pantallas pequeñas, en tres filas.

## **Reto 2:**

Diseña una galería de 6 tarjetas con Flexbox:

En pantallas grandes, deben mostrarse en 2 filas.

En pantallas pequeñas, deben apilarse.

Usa flex-grow y flex-shrink para ajustar tamaños automáticamente.

# Flexbox

## Reto 3:

Crear una página *responsive* que use las propiedades principales de Flexbox para organizar elementos.

Será la página publicitaria de un espacio de eventos.

- Utiliza un contenedor principal con `display: flex` (puede ser `body`)
- Alinea elementos vertical y horizontalmente.
- Divide la tarjeta en una sección superior (imagen) y una inferior (texto y secciones).

Usa propiedades como `flex-direction`, `justify-content`, `align-items`, `flex-grow`, y `gap`.

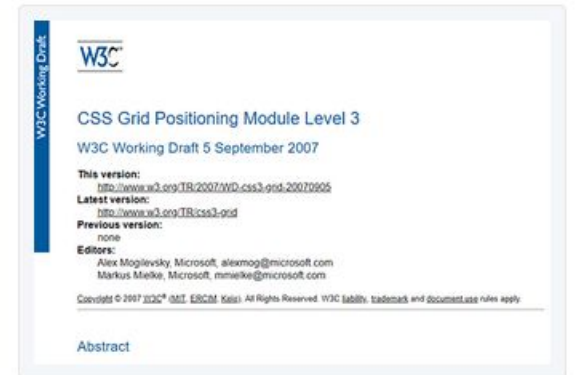
# Grid layout: Introducción

2007

September 5<sup>th</sup>

## CSS Grid

W3C released the first proposal of the **CSS Grid specification**. This CSS module defines a set of properties for creating a layout fitted into a regular grid that consists of rows and columns. The CSS Grid makes it easy to create complex and full-page layouts without the need of using cascading style layout methods involving float and positioning. CSS Grid features are currently supported by most major browsers.





# Grid layout: Introducción

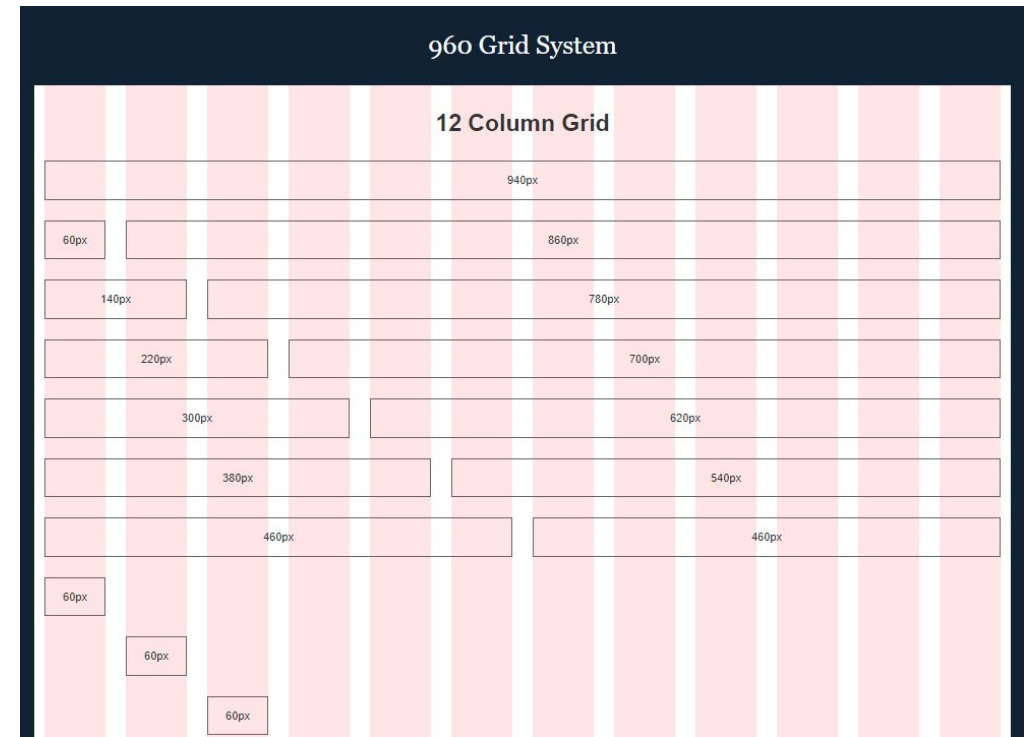
Soluciones que iban surgiendo para maquetar según una cuadrícula...

## Bootstrap Grid System

Bootstrap's grid system allows up to 12 columns across the page.

If you do not want to use all 12 column individually, you can group the columns together to create wider columns:

span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1
span 4				span 4				span 4			
span 4				span 8							
span 6						span 6					
span 12											



# CSS Grid layout: Introducción

Flexbox soluciona muchos problemas al permitir un diseño más flexible **para un solo eje** (horizontal o vertical)

Sin embargo, al diseñar layouts complejos con múltiples ejes (tanto filas como columnas), se requieren combinaciones complicadas de Flexbox y anidaciones.

Grid Layout fue creado para proporcionar un sistema más poderoso y sencillo para crear layouts **bidimensionales**.

Permite controlar filas y columnas simultáneamente.

Esto hace que sea perfecto para diseños complejos y modernos como dashboards, galerías, y layouts responsivos.

# CSS Grid layout: fundamentos

Modelo de diseño bidimensional basado en un sistema de contenedor (grid container) elementos hijos (grid items).

Este sistema utiliza líneas, áreas y celdas para organizar los elementos.

## Conceptos básicos

- **Grid Container:** el contenedor principal que define el grid.
- **Grid Lines:** líneas horizontales y verticales que dividen el grid en celdas.
- **Grid Cells:** las áreas básicas formadas entre dos líneas horizontales y dos líneas verticales.
- **Grid Areas:** áreas personalizadas que abarcan múltiples celdas.

# CSS Grid layout: propiedades container

```
.container {  
  display: grid;  
  grid-template-columns: 100px 100px 100px; /* Tres columnas de 100px */  
  grid-template-rows: auto auto; /* Dos filas con altura automática */  
  gap: 20px; /* Espaciado uniforme */  
  grid-template-areas:  
    "header header"  
    "sidebar content"  
    "footer footer";  
}
```

Se aceptan valores fijos (px, %) o relativos (fr) y combinaciones.

grid-auto-rows y grid-auto-columns para el tamaño de para filas o columnas creadas implícitamente

Además, justify-content y align-content análogo a flexbox

Permite nombrar áreas del grid para organizar visualmente los elementos

# CSS Grid layout: propiedades items

```
.item1 {  
  grid-column: 1 / 3; /* Abarca de la línea 1 a la 3 */  
  grid-row: 1 / 2; /* Abarca solo la fila 1 */  
  grid-area: header;  
  justify-self: center;  
  align-self: end;  
}
```

Alguno de los especificados  
en grid-template-areas

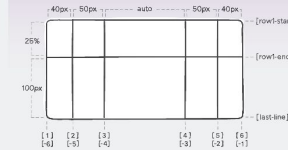
También permite combinar  
grid-row y grid-column en  
una sola propiedad,  
p.ej. 1 / 1 / 3 / 3

# CSS Grid

a guide from

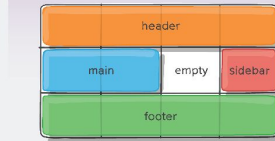
 CSS-TRICKS

## grid-template-columns grid-template-rows



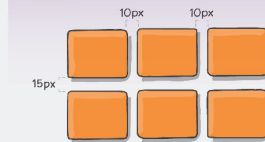
```
.container {
  grid-template-columns: <value> | <name>;
  grid-template-rows: <value> | <name>;
}
```

## grid-template-areas



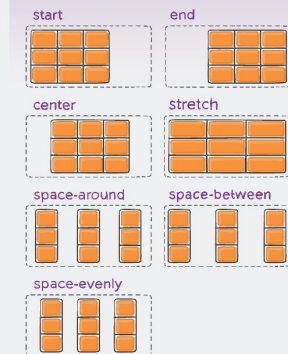
```
.container {
  grid-template-areas: "<name> | . | none";
}
```

## column-gap, row-gap, gap



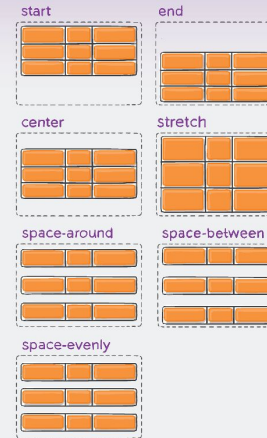
```
.container {
  column-gap: <value>;
  row-gap: <value>;
  gap: <row-gap> <column-gap>;
}
```

## justify-content



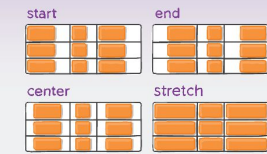
```
.container {
  justify-content: start | end | center |
  stretch | space-around | space-between |
  space-evenly;
}
```

## align-content



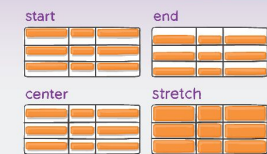
```
.container {
  align-content: start | end | center |
  stretch | space-around | space-between |
  space-evenly;
}
```

## justify-items



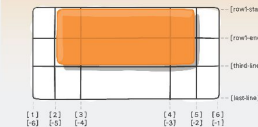
```
.container {
  justify-items: start | end |
  center | stretch;
}
```

## align-items



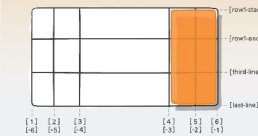
```
.container {
  align-items: start | end |
  center | stretch;
}
```

## grid-column, grid-row



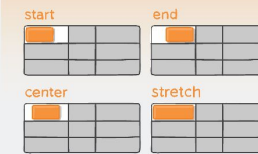
```
.item {
  grid-column: <start-line> / <end-line> |
  <start-line> / span <value>;
  grid-row: <start-line> / <end-line> |
  <start-line> / span <value>;
}
```

## grid-area



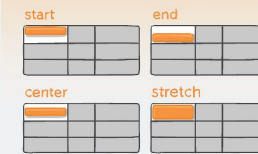
```
.item {
  grid-area: <row-start> / <column-start>
  / <row-end> / <column-end> | <name>;
}
```

## justify-self



```
.item {
  justify-self: start | end | center |
  stretch;
}
```

## align-self



```
.item {
  align-self: start | end | center |
  stretch;
}
```

# CSS Grid. Recomendaciones

- Usa unidades relativas (fr) para layouts flexibles.
- Usa repeat() para evitar repeticiones largas:

```
grid-template-columns: repeat(4, 1fr); /* 4 columnas iguales */
```

- Combina Grid con Flexbox: usa Flexbox dentro de las celdas del Grid para alineación avanzada.
- Usa minmax() para ajustar tamaños entre valores mínimo y máximo:

```
grid-template-columns: repeat(3, minmax(100px, 1fr));
```

# Comparación Flexbox vs Grid

	Flexbox	Grid
Dimensión	Diseñado para layouts en un solo eje.	Diseñado para layouts bidimensionales.
Distribución	Se enfoca en el flujo de elementos.	Controla filas y columnas simultáneamente.
Complejidad	Más sencillo para layouts lineales simples.	Mejor para layouts complejos y estructurados.



### **Reto 1:**

Diseña de un tablero de ajedrez de 8x8 con colores alternos.

### **Reto 2:**

Diseña un layout con header, main, sidebar y footer utilizando grid-template-areas. Añade colores y algo de contenido en cada una para verificar la distribución de las áreas.

### **Reto 3:**

Crea un calendario con 7 columnas para los días de la semana y tantas filas como sean necesarias para los días del mes.

### **Reto 4:**

¿Y la responsividad con Grid? Usa media queries para cambiar un layout de dos columnas a una sola columna en pantallas pequeñas.

Pista: pasar de `grid-template-columns: 1fr 2fr;` a `grid-template-columns: 1fr;`

