Fase 3-Diseño y construcción II

Jesús Ferney Reyes Pinzón

1052407044

Grupo: 301122_50

Diseño de sitios web

Tutor:

Mario Luis Ávila Pérez

Universidad Nacional Abierta y a Distancia (UNAD)

Escuela de Ciencias Básicas Tecnología e Ingeniería

Ingeniería de Sistemas

Duitama-2021

Qué es FlexBox

Flexbox es uno de los **nuevos valores HTML5 para propiedad CSS display**, que nos permite maquetar nuestras páginas web de una manera mucho más fácil de lo que se hacía con la forma tradicional, en la que utilizábamos propiedades como float o position, entre otras.

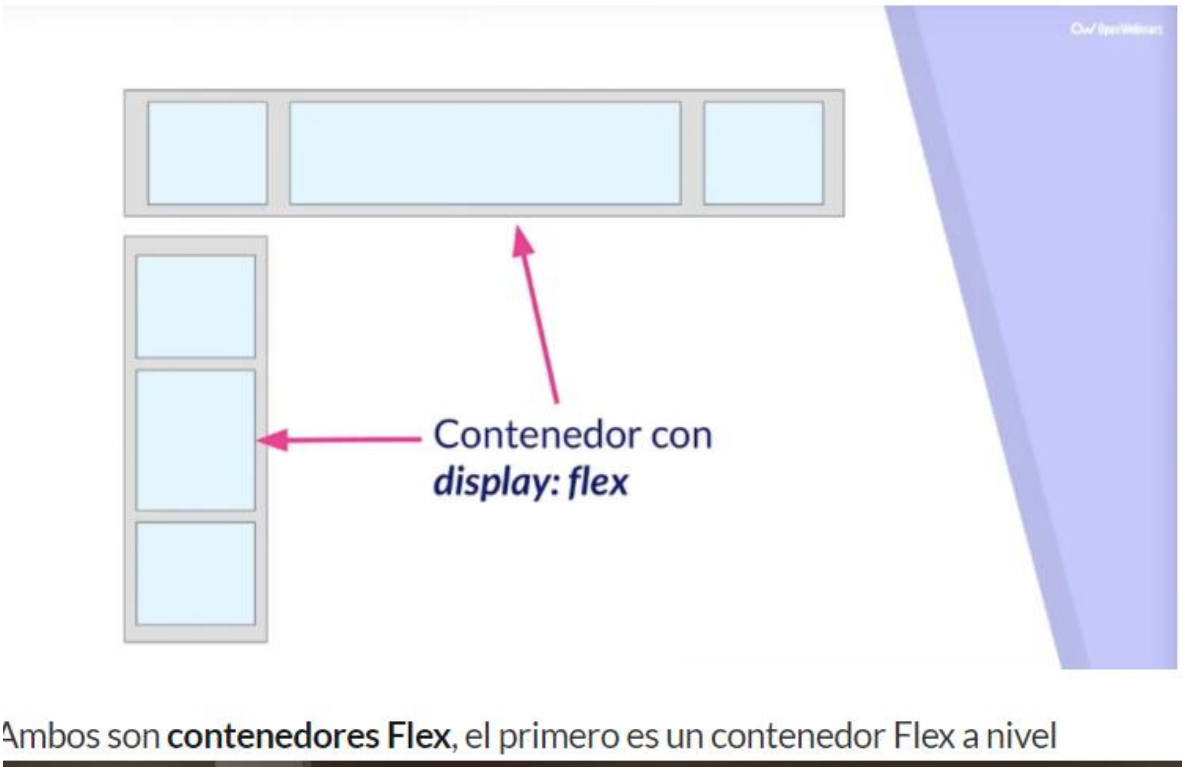
Por qué Flex

Se llama Flex porque tenemos un contenedor, llamado **contenedor Flex**, que es el elemento que contiene la propiedad display: flex.

Desde ese contenedor vamos a poder especificar la alineación de los elementos que hay dentro, el tamaño de los elementos que contienen y distribuir el espacio restante que hay entre los elementos del contenedor Flex, y todo esto en una sola dirección, ya sea una horizontal o vertical. Es decir, podemos distribuir los elementos que contiene la etiqueta sin poner nada dentro de esos elementos.

Flex a nivel visual

Vamos a ver dos ejemplos para ver cómo funciona de verdad:



Ambos son **contenedores Flex**, el primero es un contenedor Flex a nivel

Ambos son **contenedores Flex**, el primero es un contenedor Flex a nivel horizontal que contiene tres elementos flexibles, el segundo es un contenedor Flex a nivel vertical que contiene tres elementos flexibles.

Únicamente dando propiedades al contenedor Flex podemos alinear todos los elementos que están dentro del mismo.

Aspectos a tener en cuenta

Hay ciertas cosas con las que tener cuidado, como por ejemplo que a día de hoy no todos los navegadores soportan contenedores Flex. Aunque es cierto que la gran mayoría de los navegadores que usamos diariamente ya lo soportan, no obstante, hay que tener cuidado porque no todos los navegadores de los móviles lo soportan.

Incluso Twitter y su librería Bootstrap ya empieza a utilizar Flex, lo cual nos indica que ya es algo totalmente aceptado como estándar y que podemos utilizar está técnica con total tranquilidad para llegar a más del 90% de todos los dispositivos.

El Módulo de Caja Flexible, comúnmente llamado flexbox, fue diseñado como un modelo unidimensional de layout, y como un método que pueda ayudar a distribuir el espacio entre los ítems de una interfaz y mejorar las capacidades de alineación. Este artículo hace un repaso de las principales características de flexbox, las que exploraremos con mayor detalle en el resto de estas guías.

Cuando describimos a flexbox como unidimensional destacamos el hecho que flexbox maneja el layout en una sola dimensión a la vez — ya sea como fila o como columna. Esto contrasta con el modelo bidimensional del Grid Layout de CSS, el cual controla columnas y filas a la vez.

Los dos ejes de flexbox

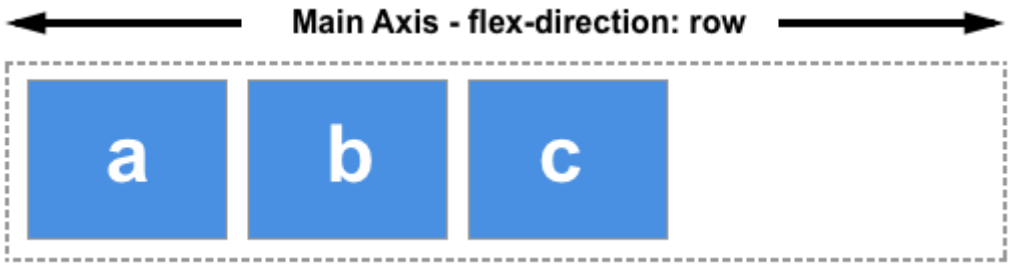
Cuando trabajamos con flexbox necesitamos pensar en términos de dos ejes — el eje principal y el eje cruzado. El eje principal está definido por la propiedad flex-direction, y el eje cruzado es perpendicular a este. Todo lo que hacemos con flexbox está referido a estos dos ejes, por lo que vale la pena entender cómo trabajan desde el principio.

El eje principal

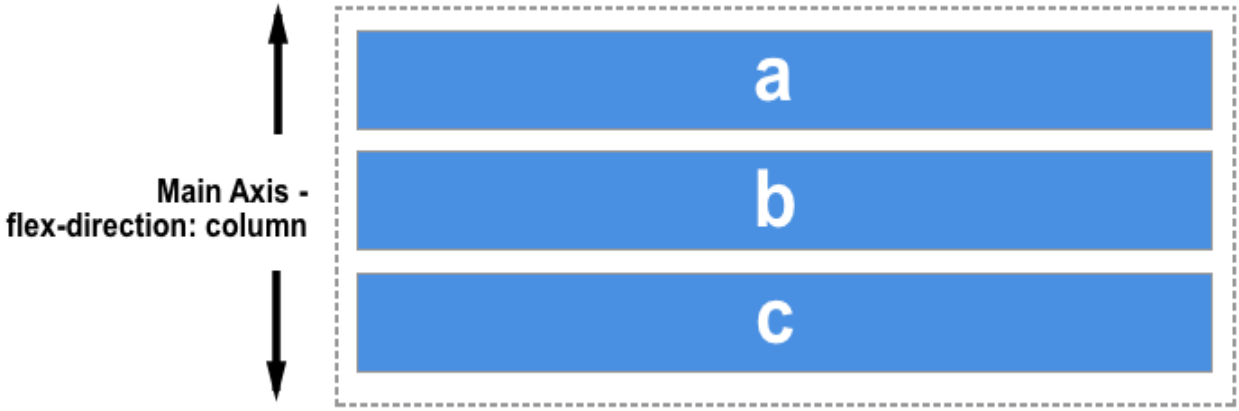
El eje principal está definido por flex-direction, que posee cuatro posibles valores:

- row
- row-reverse
- column
- column-reverse

Si elegimos row o row-reverse, el eje principal correrá a lo largo de la fila según la **dirección de la línea** .



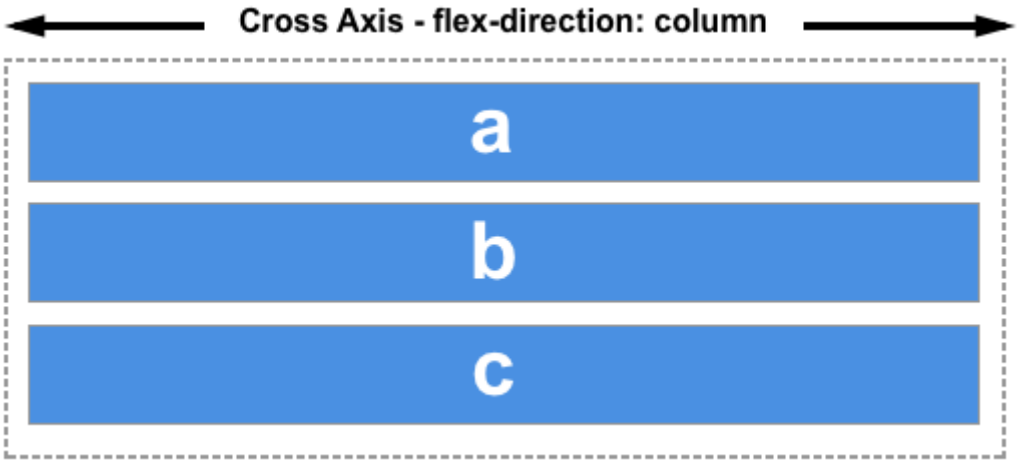
Al elegir column o column-reverse el eje principal correrá desde el borde superior de la página hasta el final — según la **dirección del bloque**.



El eje cruzado

El eje cruzado va perpendicular al eje principal, y por lo tanto si flex-direction (del eje principal) es row o row-reverse el eje cruzado irá por las columnas.

Si el eje principal es `column` o `column-reverse` entonces el eje cruzado corre a lo largo de las filas.



Entender cuál eje es cuál es importante cuando empezamos a mirar la alineación y justificación flexible de los ítems; flexbox posee propiedades que permiten alinear y justificar el contenido sobre un eje o el otro.

Líneas de inicio y de fin

Otra área vital de entendimiento es cómo flexbox no hace suposiciones sobre la manera de escribir del documento. En el pasado, CSS estaba muy inclinado hacia el modo de escritura horizontal y de izquierda a derecha. Los métodos modernos de layout acogen la totalidad de modos de escritura así que no es necesario asumir que una línea de texto empezará arriba del documento y correrá de izquierda a derecha, con nuevas líneas dispuestas una abajo de la otra.

Puede leer más acerca de la relación que hay entre flexbox y la especificación de los Modos de Escritura en un artículo posterior, sin embargo la siguiente descripción debería ayudar para explicar porqué no se habla de izquierda y derecha ni de arriba o abajo a la hora de describir la dirección en la que fluyen los ítems flex.

Si `flex-direction` es `row` y estoy trabajando en español, entonces el margen inicial del eje principal quedará a la izquierda, y el margen final a la derecha.

Si fuera a trabajar en árabe, entonces el margen inicial de mi eje principal quedaría a la derecha y el margen final a la izquierda.

En ambos casos el margen inicial del eje cruzado estará en el extremo superior del contenedor flex y el margen final en el extremo inferior, ya que ambos idiomas tiene un modo de escritura horizontal.

Después de un tiempo, pensar en inicial y final en vez de izquierda y derecha se hará natural, y será útil cuando interactúe con otros métodos de layout tales como el CSS Grid Layout que sigue los mismos patrones.

CSS GRID

Al crear un sitio web, gran parte del trabajo consiste en disponer correctamente los distintos elementos que la componen. El diseño debe ser atractivo, pero claro al mismo tiempo, y se debe comprender de forma intuitiva. Las hojas de estilo en cascada (CSS) constituyen una herramienta para diseñar sitios web con esas características. Mientras que HTML muestra el contenido de forma rudimentaria, CSS es apto para diseños

complejos. La tecnología de la web está en constante desarrollo y ahora, con CSS3, han aparecido técnicas nuevas que permiten aplicar el lenguaje en el Internet móvil y utilizar el diseño responsivo.

¿Por qué se usa CSS grid layout?

El internet móvil presenta un gran número de desafíos para los diseñadores de páginas web: debido a la enorme variedad de diseños de los dispositivos móviles, es imposible saber **qué formato tiene la pantalla** en la que se visualizará el contenido web. Por esto, es esencial que los elementos individuales (cajas de texto, gráficos, elementos interactivos) se distribuyan de forma independiente y al mismo tiempo de forma clara, teniendo en mente las respectivas condiciones de espacio dadas por cada pantalla.

Hace un tiempo se trabajaba con los llamados *floats*, pero es una técnica compleja que daba lugar a muchos errores. Hoy los diseñadores cuentan con dos métodos para implementar un **diseño dinámico:** además de CSS grid, también se puede usar Flexbox. Sin embargo, las dos técnicas difieren en algunos aspectos.

Flexbox es unidimensional. Esto quiere decir que los elementos solo se pueden mover a lo largo de un eje. En cambio, un diseño CSS grid ofrece al diseñador web dos dimensiones para la colocación de los objetos porque, en lugar de solo un eje, permite crear una **rejilla con filas y columnas**.

CSS grid emplea dos unidades diferentes: **contenedores y elementos**. El contenedor es el nivel superior y en él se definen las propiedades que luego tendrán todos los elementos. Desde un punto de vista jerárquico, un elemento está dentro de un contenedor. Además de eso, se sigue necesitando HTML para el diseño de la rejilla o *grid*. En la parte HTML del código fuente, se crean los elementos individuales (texto, gráficos, etc.), que luego se recogen dentro de CSS grid y se disponen en la posición correcta.

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
}


</style>
</head>
<body>
```

```
<div class="grid-container">
  <div class="grid-item1">1</div>
  <div class="grid-item2">2</div>
  <div class="grid-item3">3</div>
  <div class="grid-item4">4</div>
  <div class="grid-item5">5</div>
  <div class="grid-item6">6</div>
</div>

</body>
</html>
```

De este modo, hemos creado seis elementos, hemos definido cada uno de ellos como *grid-item* y los hemos empaquetado en el *grid-container*. Para activar CSS grid, tienes que iniciar la función en el contenedor con *display: grid*. El código generará únicamente 6 números, que aparecerán uno debajo del otro. Después de crearlos, pueden colocarse con relativa libertad en la pantalla.

Rejilla, columnas y filas

Con CSS grid layout, se trabaja con **filas y columnas para crear una cuadrícula**, en la cual se colocan y distribuyen los distintos elementos. El usuario es quien decide el tamaño de las filas y las columnas, añadiendo las preferencias al contenedor.

```
.grid-container {
  display: grid;
  grid-template-rows: 100px; 100px
  grid-template-columns: 100px; 100px; 100px;
}
```

Con estos dos comandos hemos abierto una cuadrícula de 2 por 3. Como puedes ver, **el tamaño de cada fila y columna se puede definir por separado**. Los datos se indican de forma sucesiva (separados por un punto y coma y un espacio). Además de la información exacta de los píxeles, también puedes usar porcentajes u otras unidades que son comunes en CSS. Las especificaciones *max-content* y *min-content* permiten crear una cuadrícula según el contenido.

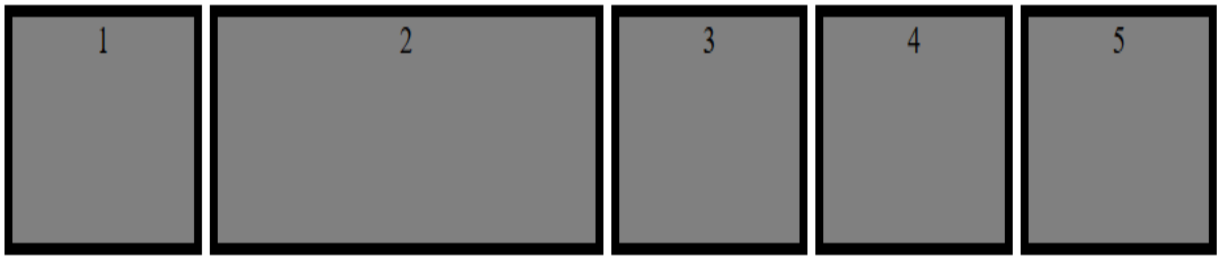
La opción *grid-gap* crea un **espacio vacío**.

```
.grid-container {
  display: grid;
  grid-template-rows: 100px; 100px;
  grid-template-columns: 100px; 100px; 100px;
  grid-gap: 5px;
}
```

Con este código, las celdas quedarán separadas con espacios uniformes. Si no quieres un espaciado uniforme, puedes usar *grid-column-gap* y *grid-row-gap* para personalizar el tamaño de los espacios.

Las **fracciones** son una función especial. Esta unidad de medida permite dividir la pantalla en secciones diferentes. Por ejemplo, imaginemos que queremos dividirla en 7 unidades horizontales, teniendo una de las columnas el doble del tamaño de las otras. Para ello, puedes usar este código:

```
.grid-container {
  display: grid;
  grid-template-rows: 100px 100px;
  grid-template-columns: 1fr 2fr 1fr 1fr 1fr;
  grid-gap: 5px;
}
```



El CSS grid layout permite crear columnas y filas de distintos tamaños.

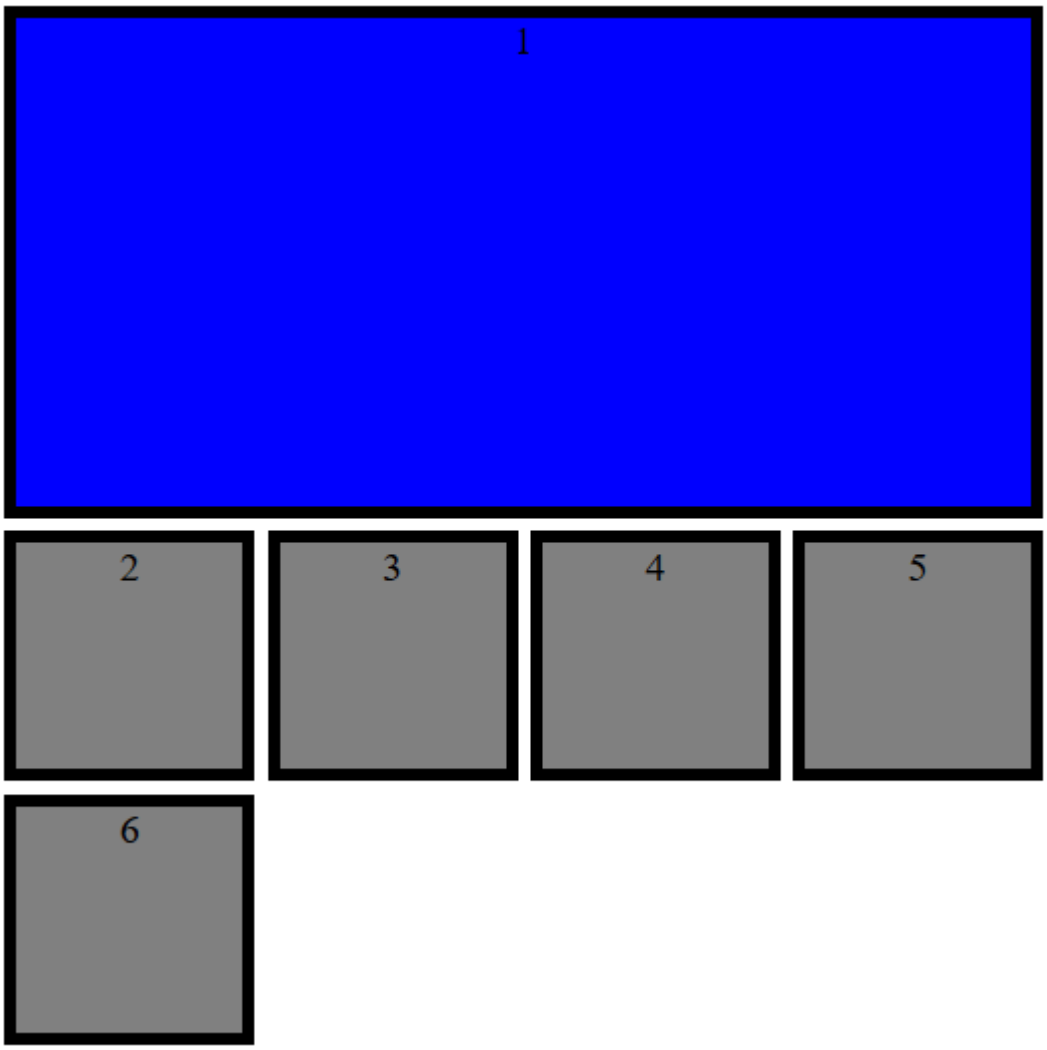
La ventaja de trabajar con tamaños relativos en lugar de especificaciones estáticas es que la cuadrícula puede adaptarse **automáticamente al tamaño de la pantalla**. Aunque las columnas tengan que modificar su tamaño, la segunda columna (en nuestro ejemplo) siempre será dos veces más grande que las otras. Si quieres fijar una fila, es decir, no adaptarla al tamaño de la pantalla, debes asignarle un valor estático.

Disposición de los elementos

Después de definir la rejilla, se colocan en ella los elementos. Para esto, primero tienes que **crear los elementos** y también especificar los valores de inicio y fin. No obstante, no todos los elementos tienen que ocupar necesariamente una sola celda dentro de la cuadrícula.

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
```

```
  display: grid;
  grid-template-rows: 100px 100px 100px 100px;
  grid-template-columns: 100px 100px 100px 100px;
  grid-gap: 5px;
}

.grid-item1 {
  background: blue;
  text-align: center;
  border: black 5px solid;
  grid-column-start: 1;
  grid-column-end: 5;
  grid-row-start: 1;
  grid-row-end: 3;
}

.grid-item2 {
  background: grey;
  text-align: center;
  border: black 5px solid;
}

</style>
</head>
<body>

<div class="grid-container">
  <div class="grid-item1">1</div>
  <div class="grid-item2">2</div>
  <div class="grid-item2">3</div>
  <div class="grid-item2">4</div>
  <div class="grid-item2">5</div>
  <div class="grid-item2">6</div>
</div>

</body>
</html>
```



El primer elemento ocupa cuatro filas y dos columnas.

Hasta ahora hemos hablado de **dos tipos de elementos.** Mientras que los últimos cinco elementos solo ocupan una celda, el primer elemento se extiende sobre cuatro columnas y dos filas (para mayor claridad, hemos variado el diseño de los elementos en nuestro ejemplo, pero en un CSS grid no se define el color, los bordes o la sangría de los textos).

Los valores del principio y el final de los elementos se refieren de forma indirecta a las filas y columnas, porque, de hecho, se refiere a la fila respectiva de la rejilla. En el ejemplo, la cuarta columna termina con la quinta línea. Sin embargo, tienes varias opciones para especificar los rangos.

- **Numeración**: las líneas se cuentan de izquierda a derecha y de arriba a abajo.
- **Nombres**: dentro de *grid-template-rows* y *grid-template-columns* se puede poner nombres a las líneas (entre corchetes) y luego referirse a estos nombres.

- **Alcance**: con *span* se indica cuántas celdas debe comprender el objeto en una determinada dirección.

En lugar de definir los puntos de inicio y final en comandos separados, los diseñadores web pueden combinar ambos bajo uno solo. El siguiente código te dará el resultado del ejemplo anterior:

```
<style>
.grid-container {
  display: grid;
  grid-template-rows: 100px [Line1] 100px [Line2] 100px [Line3] 100px [Line4];
  grid-template-columns: 100px 100px 100px 100px;
  grid-gap: 5px;
}

.grid-item1 {
  background: blue;
  text-align: center;
  border: black 5px solid;
  grid-column: 1 / span 4;
  grid-row: 1 / Line2;
}

.grid-item2 {
  background: grey;
  text-align: center;
  border: black 5px solid;
}

</style>
```

Asignación de áreas

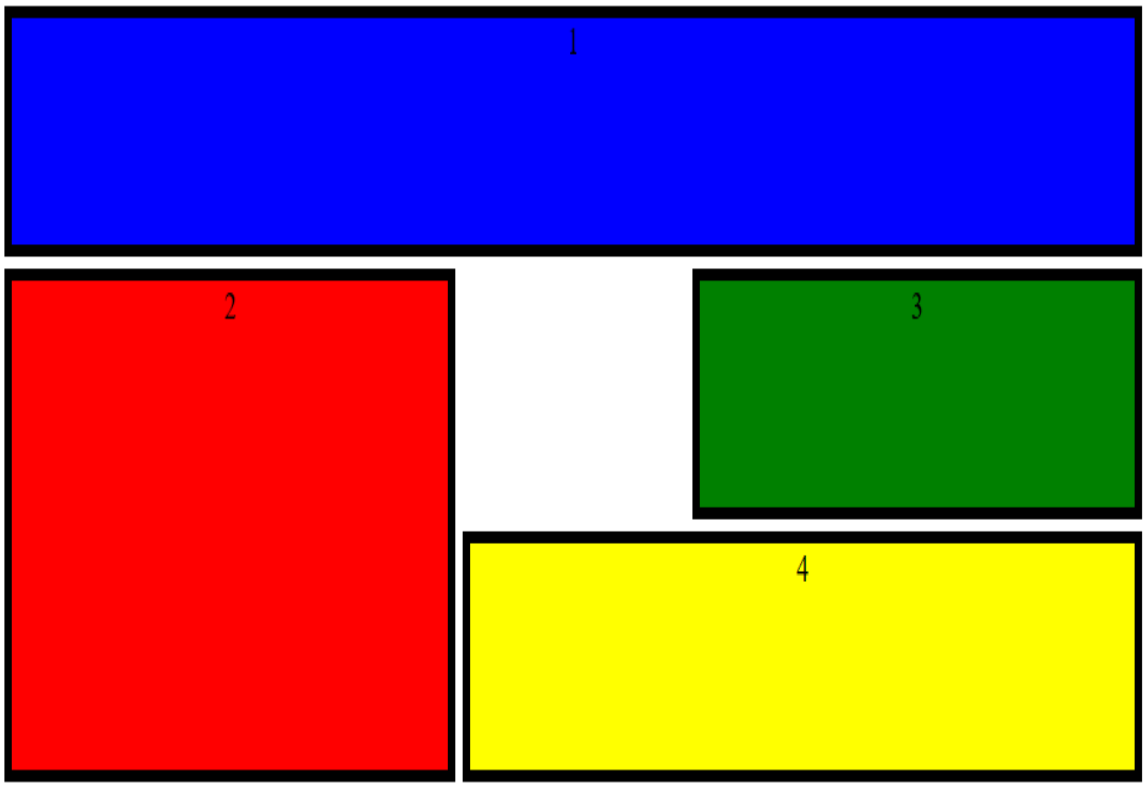
CSS grid layout permite **combinar celdas en áreas** y nombrarlas. Esto facilita la tarea de dividir los elementos en la rejilla. Los ajustes para esto se hacen en el contenedor. Para ello, usa el comando *grid-template-areas* y escribe los nombres de las áreas deseadas en las celdas línea por línea. Si no quieres asignar una celda y dejarla en blanco, puedes insertar un punto en esta ubicación. Cada fila queda entrecomillada.

```
.grid-container {
  display: grid;
  grid-template-rows: 100px 100px 100px;
  grid-template-columns: 1fr 1fr 1fr 1fr 1fr;
  grid-gap: 5px;
  grid-template-areas:
```

```
  "area1 area1 area1 area1 area1"
  "area2 area2 . area3 area3"
  "area2 area2 area4 area4 area4";
}
```

En este ejemplo, hemos definido 4 áreas diferentes. Una celda se ha dejado en blanco. Para definir los elementos, ya no será necesario especificar los valores desde-hasta. Basta con referenciar el área correspondiente.

```
.grid-item1 {
  background: blue;
  text-align: center;
  border: black 5px solid;
  grid-area: area1;
}

.grid-item2 {
  background: red;
  text-align: center;
  border: black 5px solid;
  grid-area: area2;
}

.grid-item3 {
  background: green;
  text-align: center;
  border: black 5px solid;
  grid-area: area3;
}
.grid-item4 {
  background: yellow;
  text-align: center;
  border: black 5px solid;
  grid-area: area4;
}
```



Usar áreas para la asignación de objetos te facilitará mucho la tarea.

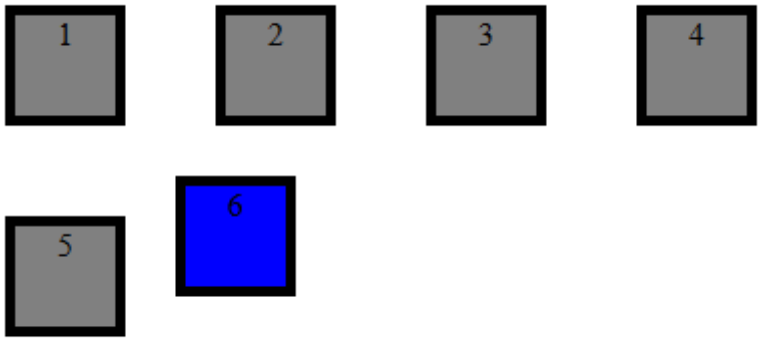
Ajustar la alineación

¿Cómo se alinean los elementos individuales dentro de CSS grid? Por defecto, los elementos se estiran para que encajen de forma precisa en la cuadrícula. Por esto, el **tamaño del elemento no tiene un papel determinante** y hasta ahora no se ha especificado en los ejemplos. En lugar de esto, hemos especificado sobre qué celdas debe estar el objeto. Sin embargo, también se puede asignar un tamaño fijo a un elemento e integrarlo igualmente en la cuadrícula.

Los diseñadores web pueden establecer la alineación de todos los elementos en el **contenedor** o alinear individualmente los elementos seleccionados. Para la primera variante, usa *justify-items* y *align-items*. El primer comando controla la alineación horizontal, el segundo, la vertical. Si solo quieres alinear un elemento, utiliza las opciones *justify-self* y *align-self*. Todos los comandos contienen las mismas opciones.

- ***Stretch***: el tamaño del objeto se ajusta al tamaño de las celdas seleccionadas.
- ***Start***: el objeto se alinea a la izquierda o arriba.
- ***End***: el objeto se alinea a la derecha o abajo.
- ***Center***: el objeto se alinea en el centro.

```
.grid-container {
  display: grid;
  grid-template-rows: 100px 100px 100px 100px;
  grid-template-columns: 100px 100px 100px 100px;
  grid-gap: 5px;
  justify-items: center;
  align-items: center;
}

.grid-item1 {
  background: grey;
  text-align: center;
  border: black 5px solid;
  width: 50px;
  height: 50px;
}

.grid-item2 {
  background: blue;
  text-align: center;
  border: black 5px solid;
  width: 50px;
  height: 50px;
  justify-self: start;
  align-self: start;
}
```



CSS grid permite cambiar la alineación de forma global o individual.

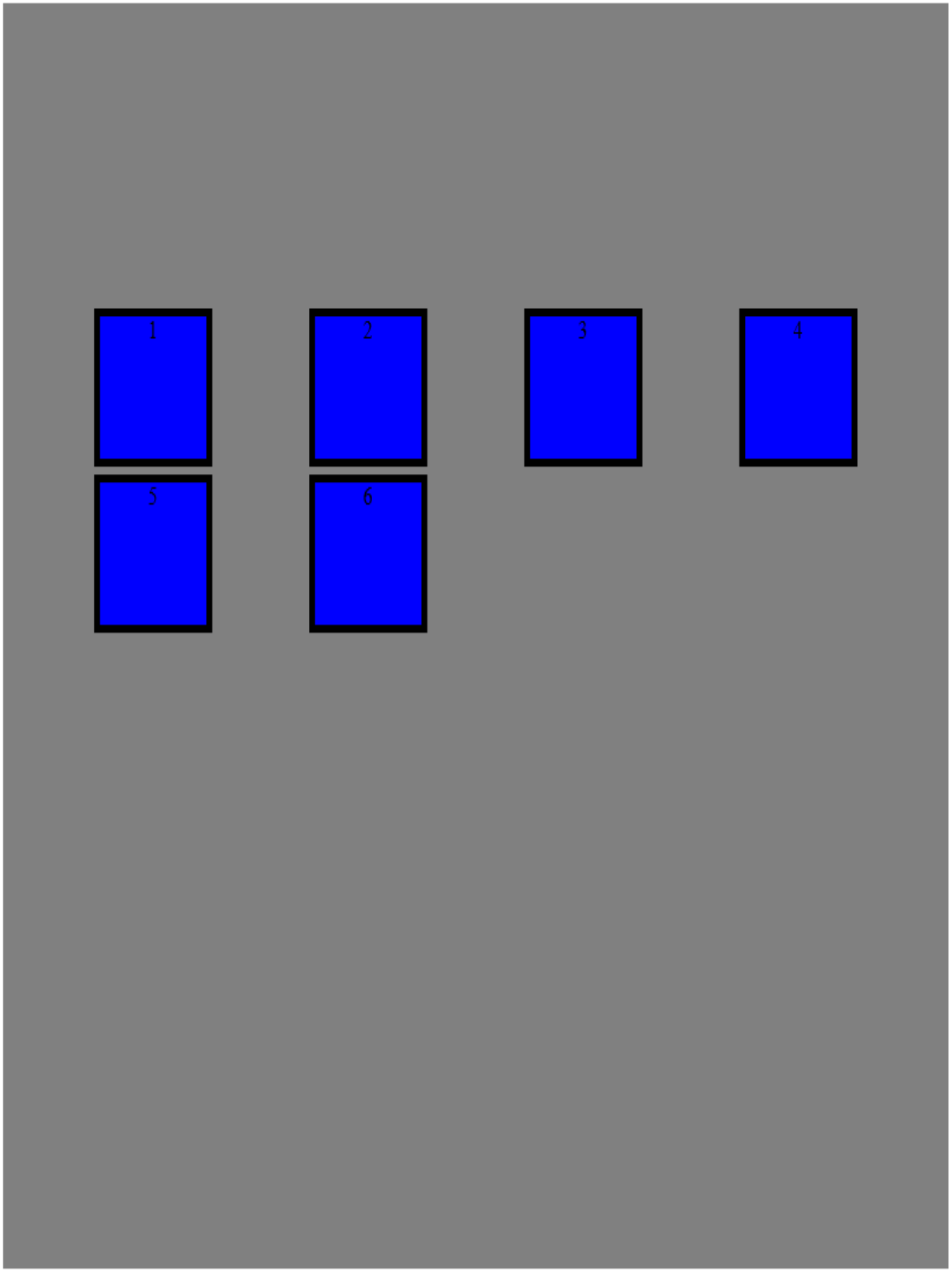
También puedes **abreviar las instrucciones** usando *place-items* o *place-self*. Ambas informaciones, vertical y horizontal, se pueden indicar en una sola línea: *place-items*: *<align-items> / justify-items>*.

```
place-items: center / end;
```

Es posible entonces organizar los elementos en la rejilla. Sin embargo, también puedes mover **toda la rejilla dentro del contenedor.** Si trabajas con una plantilla de CSS grid con especificaciones de tamaño estáticas, la cuadrícula puede no tener el mismo tamaño que todo el contenedor. En este caso puedes usar *justify-content* y *align-content* para alinear la cuadrícula dentro del contenedor, y así también dentro de la pantalla. Para esto hay también varias posibilidades de alineación:

- ***Start***: alineado a la izquierda o alineado hacia arriba
- ***End***: alineado a la derecha o alineado hacia abajo
- ***Center***: alineado al centro
- ***Stretch***: estiramiento por cuadrícula
- ***Space-around***: distribución uniforme del espacio alrededor de las celdas
- ***Space-between***: distribución uniforme del espacio entre las celdas
- ***Space-evenly***: distribución uniforme del espacio alrededor de las celdas, incluyendo el borde

```
.grid-container {
  display: grid;
  width: 800px;
  height: 800px;
  background: yellow;
  grid-template-rows: 100px 100px 100px 100px;
  grid-template-columns: 100px 100px 100px 100px;
  grid-gap: 5px;
  justify-content: space-evenly;
  align-content: center;
}
```



Puedes alinear no solo elementos individuales, sino incluso la cuadrícula completa dentro

del contenedor.

Para esto también hay una abreviatura: *place-content: <align-content> / <justify-content>*.

```
place-content: center / space-evenly;
```

Ajustes automáticos para usar diseño responsivo

Una de las principales ventajas de CSS grid es la flexibilidad de la rejilla. CSS grid se puede configurar para que se ajuste automáticamente. Esto no solo facilita la visualización en los diferentes dispositivos: **la automatización** también te facilitará el trabajo con CSS.

Una función útil es *repeat()*. En vez de definir cada fila o columna individualmente, se puede también especificar un tamaño y **la frecuencia con la que se debe repetir este esquema**. Esto es aún más fácil en combinación con las opciones *auto-fill* y *auto-fit*. De esta forma, se delega a CSS grid la creación de filas y columnas. Con la primera opción, inserta tantas celdas como le sea posible sin exceder los límites del contenedor. Sin embargo, esto puede hacer que quede espacio sin usar. La opción *auto-fit*, por el contrario, ajusta el tamaño de las celdas para que usen todo el espacio disponible.

```
.grid-container {
  display: grid;
  width: 800px;
  height: 800px;
  grid-template-rows: repeat(auto-fit, 100px);
  grid-template-columns: repeat(auto-fit, 100px);
  grid-gap: 5px;
}
```

Las dos funciones *grid-auto-columns* y *grid-auto-rows* también son de gran utilidad. Estas dos instrucciones te dan **mayor libertad a la hora de crear elementos**. Si, por ejemplo, tienes una cuadrícula con las dimensiones de 4x4 celdas y ahora se creara un elemento que debe comenzar a partir de la quinta columna, sería un problema. Con la función de creación automática de filas y columnas, se evita este percance.

```
.grid-container {
  display: grid;
  grid-auto-rows: 100px;
  grid-auto-columns: 100px;
  grid-gap: 5px;
}
```

Incluso si se debe adaptar el tamaño de la cuadrícula y los elementos a la pantalla, a los diseñadores web les gusta introducir **valores mínimos y máximos.** La declaración *minmax()* sirve para asegurarse de que ningún elemento se vuelva demasiado

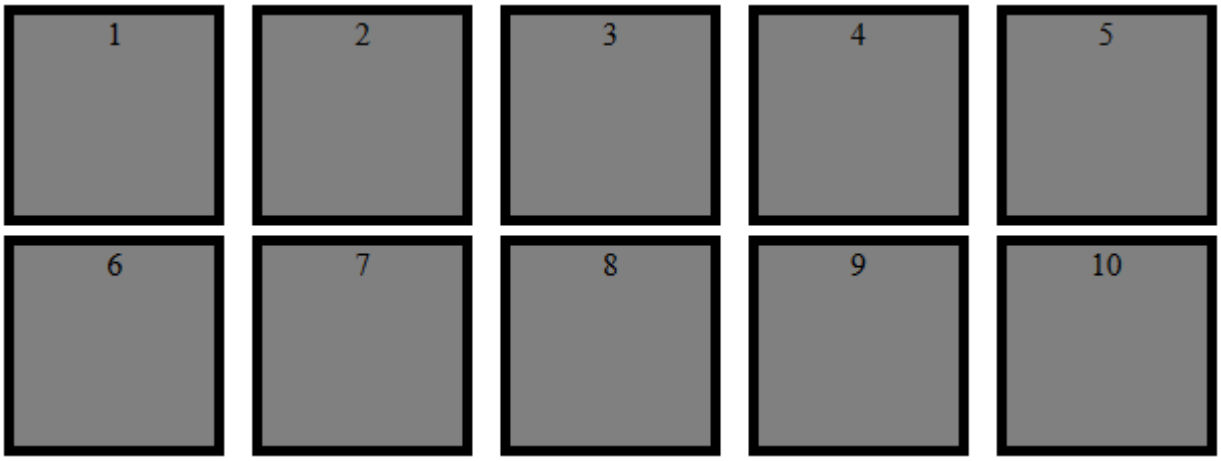
pequeño o demasiado grande. Indica primero el valor más pequeño entre paréntesis, después el más grande.

```
.grid-container {
  display: grid;
  grid-template-rows: 100px 100px 100px;
  grid-auto-columns: minmax(50px, 150px);
  grid-gap: 5px;
}
```

Combinando varias de las funciones presentadas, puedes **crear fácilmente undiseño responsivo**.

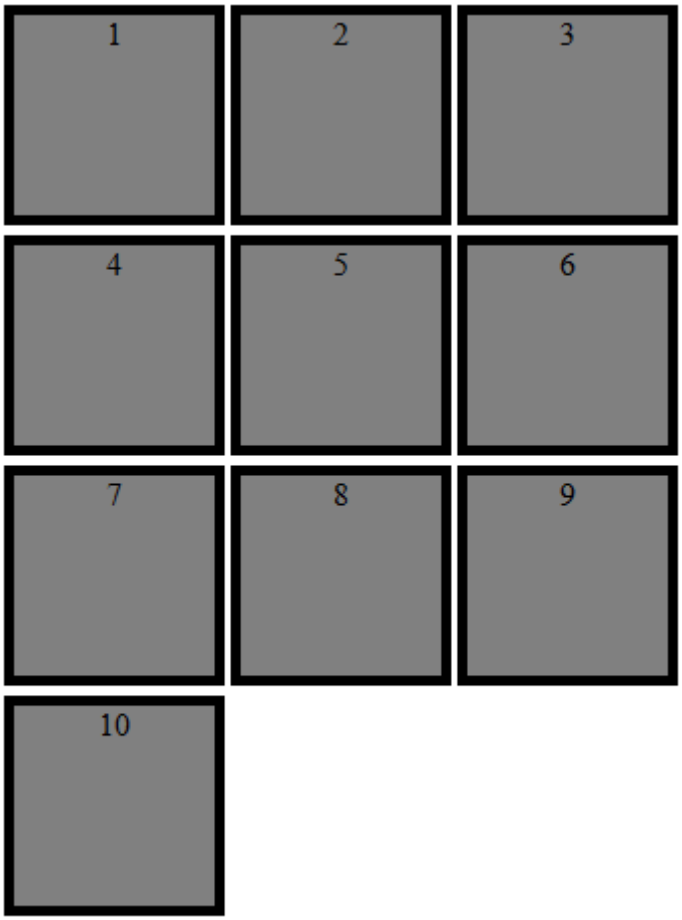
```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
  grid-gap: 5px;
  justify-items: center;
  align-items: center;
}

.grid-item1 {
  background: grey;
  text-align: center;
  border: black 5px solid;
  width: 100px;
  height: 100px;
}


</style>
</head>
<body>

<div class="grid-container">
  <div class="grid-item1">1</div>
  <div class="grid-item1">2</div>
  <div class="grid-item1">3</div>
  <div class="grid-item1">4</div>
  <div class="grid-item1">5</div>
  <div class="grid-item1">6</div>
```

```
  <div class="grid-item1">7</div>
  <div class="grid-item1">8</div>
  <div class="grid-item1">9</div>
  <div class="grid-item1">10</div>

</div>

</body>
</html>
```



Si configuras CSS grid correctamente, puedes crear un diseño responsivo sin muchas herramientas.

Si se aumenta el área de visualización, caben más elementos en una fila.



CSS grid layout permite visualizar el contenido correctamente incluso en las pantallas más pequeñas.