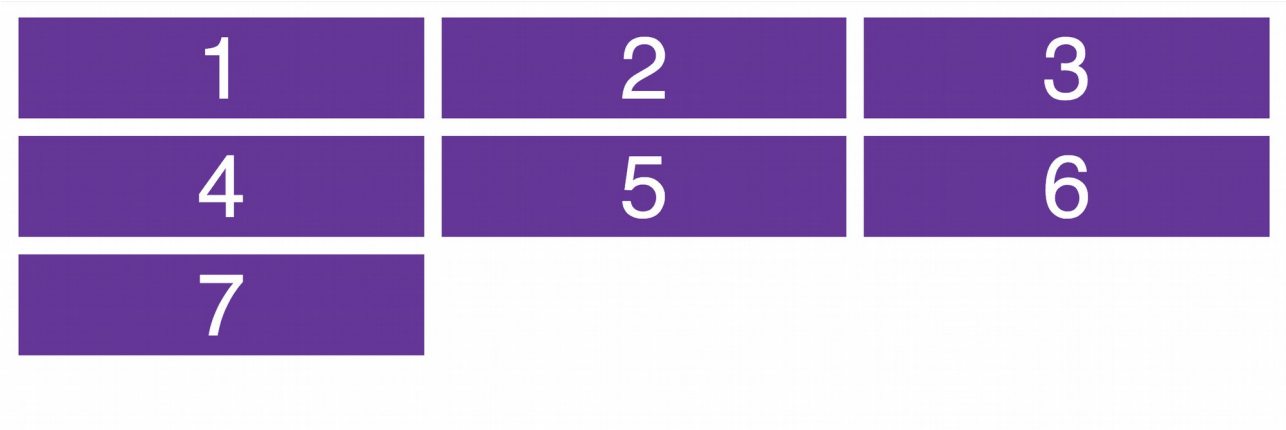
CSS Grid

CSS grid es un sistema de maquetación basado en grillas y se caracteriza por ser **bidimensional**, **independiente del orden** del markup y **flexible**. *Extremadamente* flexible.

Veamos ejemplos. ¿Cómo haríamos el siguiente layout?



Pues con CSS Grid sería tan fácil como a partir del siguiente HTML:

```html
<main>
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
</main>
```

Haríamos lo siguiente:

```css
main {
  display: grid;
  grid-gap: 1rem;
  grid-template-columns: 200px 200px 200px;
  grid-template-rows: 150px 150px;
}
```

Como vemos en el CSS de la clase **`container`** decimos a este elemento que su modo `display` es **`grid`**.

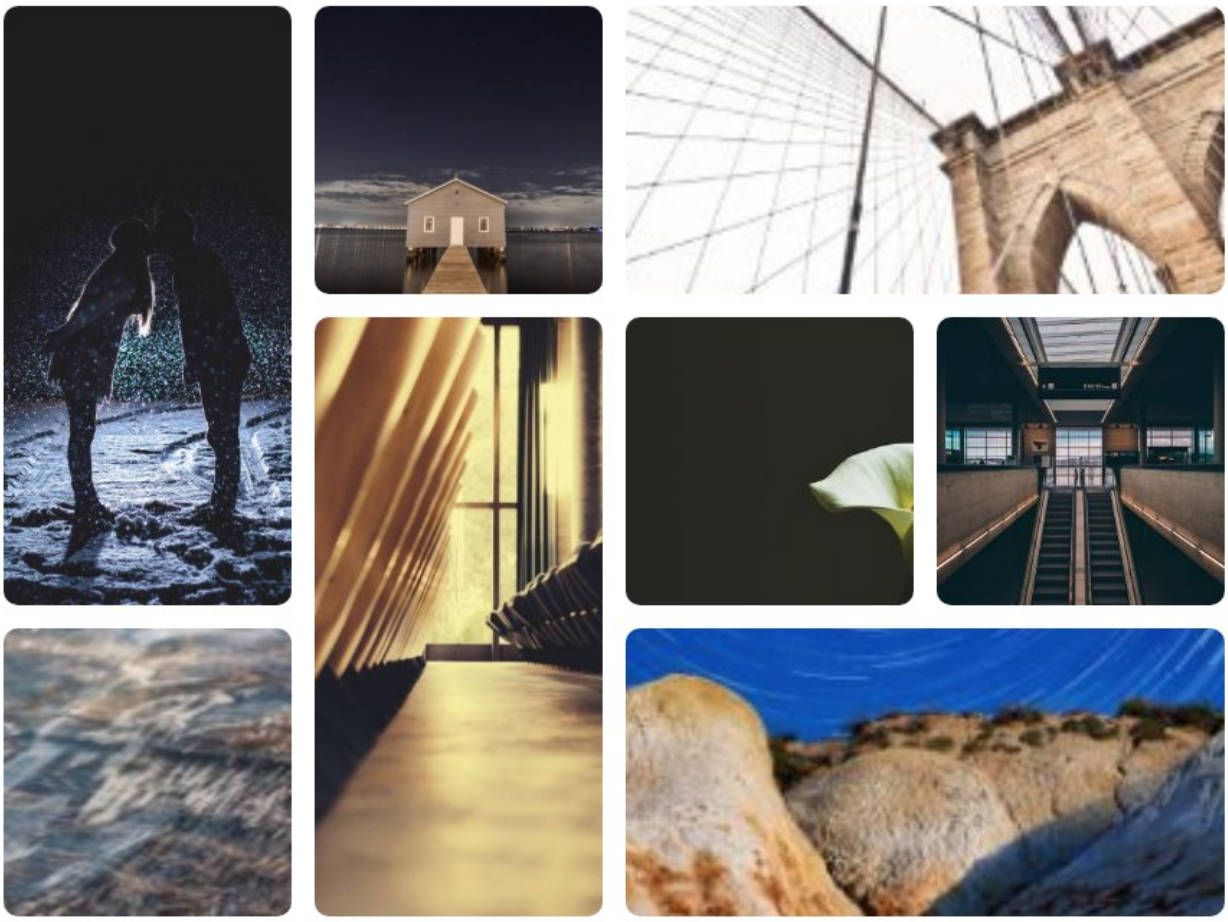
El siguiente punto importante es **`grid-gap: 1rem`**. Esta línea lo que hace es incluir un "gutter" o canal de rejilla de **`1rem`** para separar los elementos de la grilla.

Con `grid-template-columns` lo que hacemos es definir el número de columnas y el ancho de las mismas. En este caso se han definido 3 columnas de 200px cada una.

Con `grid-template-rows` más de lo mismo, solo que esta vez estamos definiendo el número de filas.

Colocar elementos en la grilla

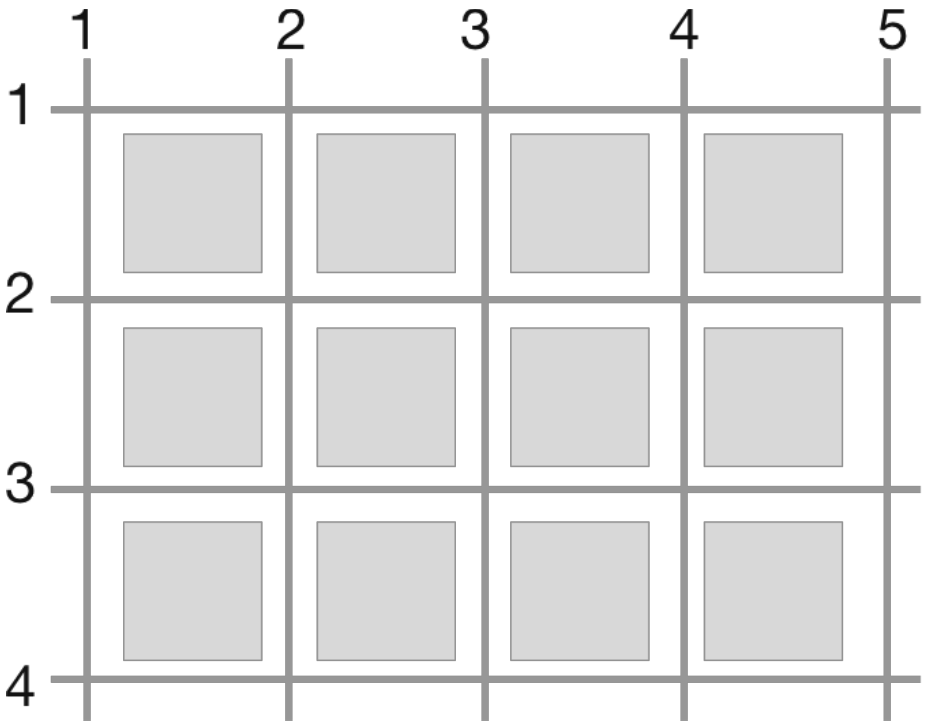
Hemos visto que los elementos se colocan solos. Pero, ¿qué pasa si queremos colocar los elementos nosotros? Un buen caso de uso sería para el siguiente layout:



Por esta razón podemos hacer uso de **`grid-column-start`** y **`grid-column-end`** para colocar elementos en un línea de columna o **`grid-row-start`** y **`grid-row-end`** para las filas.

O también podemos hacer uso del método acortado `grid-column: X / Y;` y **`grid-row: X / Y;`** siendo X e Y el número de columna o fila inicial y el final.

Cuando definimos una grilla de 4 columnas por debajo se generan 5 líneas como se puede observar en el siguiente gráfico. Y a su vez como hemos definido 3 filas por tanto se generan 4 líneas.



Ejemplo del código:

```
<main>
  <div class="item-a">
    <img src="https://picsum.photos/300/600/?random">
  </div>
  <div>
    <img src="https://picsum.photos/600/300/?random">
  </div>
  <div class="item-b">
    <img src="https://picsum.photos/200/200/?random">
  </div>
  <div>
    <img src="https://picsum.photos/200/500/?random">
  </div>
  <div class="item-c">
    <img src="https://picsum.photos/200/400/?random">
  </div>
  <div>
    <img src="https://picsum.photos/200/250/?random">
  </div>
  <div>
    <img src="https://picsum.photos/200/100/?random">
  </div>
  <div class="item-d">
    <img src="https://picsum.photos/200/600/?random">
  </div>
</main>
```

```css
main {
  max-width: 800px;
  padding: 1rem;
  margin: 0 auto;
  display: grid;
  grid-template-columns: 200px 200px 200px 200px;
  grid-template-rows: 200px 200px 200px;
  grid-gap: 1rem;
}

img {
  object-fit: cover;
  width: 100%;
  height: 100%;
  border-radius: 10px;
}

.item-a {
  grid-row: 1 / 3;
}

.item-b {
  grid-column: 3 / 5;
}

.item-c {
  grid-row: 2 / 4;
}

.item-d {
  grid-column: 3 / 5;
}
```

Layouts con grid-template-areas

CSS Grid no solamente permite hacer de nuestra maquetación más flexible, sino que además es un motor muy potente para diseñar layouts en **dos dimensiones**. Por ejemplo veamos uno de los layouts más usados a lo largo de la web.

En este ejemplo haremos un diseño *mobile-first*, es decir comenzaremos maquetando para móvil e iremos avanzando por los puntos de ruptura pasando por tablet y posteriormente a desktop.

Primero definimos el HTML:

Shell

```html
<main>
  <header>Header</header>
  <nav>Nav</nav>
  <section>Section</section>
  <aside>Aside</aside>
  <footer>Footer</footer>
</main>
```

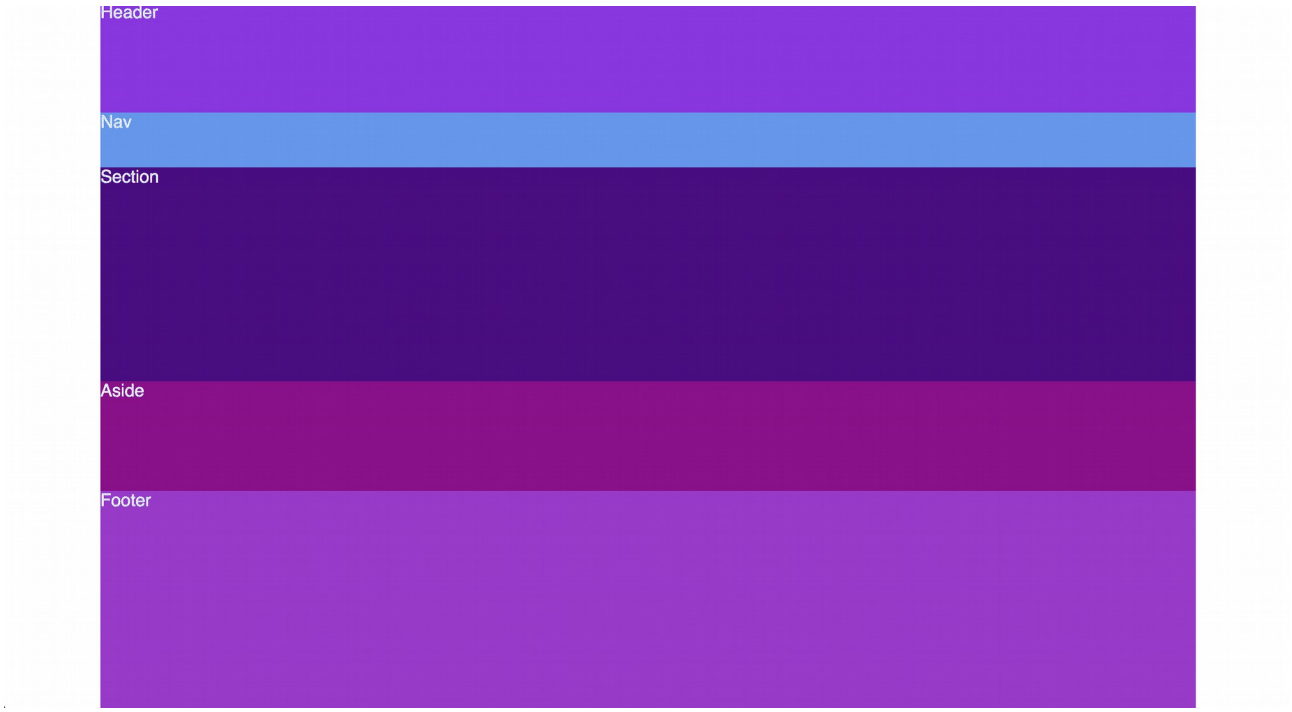
ahora vamos con el css, formateamos el main, y también añadiremos algunos colores, para que se vea todo mejor. Con lo que nuestro CSS sería el siguiente:

Shell

```css
main {
  color: gainsboro;
  margin: 0 auto;
  max-width: 1000px;
  min-height: 100vh;
  display: grid;
  grid-template-columns: 100%;
  grid-template-rows: 100px
    50px
    1fr
    100px
    200px;
}

header {
  background-color: blueviolet;
}

nav {
  background-color: cornflowerblue;
}

aside {
  background-color: darkmagenta;
}

section {
  background-color: indigo;
}

footer {
  background-color: darkorchid;
}
```

NOTA: Y si nos fijamos atentamente veremos un nueva unidad: `fr`. Esta unidad de medida es propia de CSS Grid e indica que un elemento debería ocupar una fracción determinada del espacio restante.

El resultado tendría que ser el siguiente:



La pregunta ahora es la siguiente: ¿Cómo reordenamos los elementos? Es aquí donde entran en juego **`grid-template-areas`** y **`grid-area`**. Es mucho mejor visto en un ejemplo:

A cada *sección* de nuestra web le asignamos un **`grid-area`**. El nombre del **`grid-area`** puede ser el que queramos:

Shell

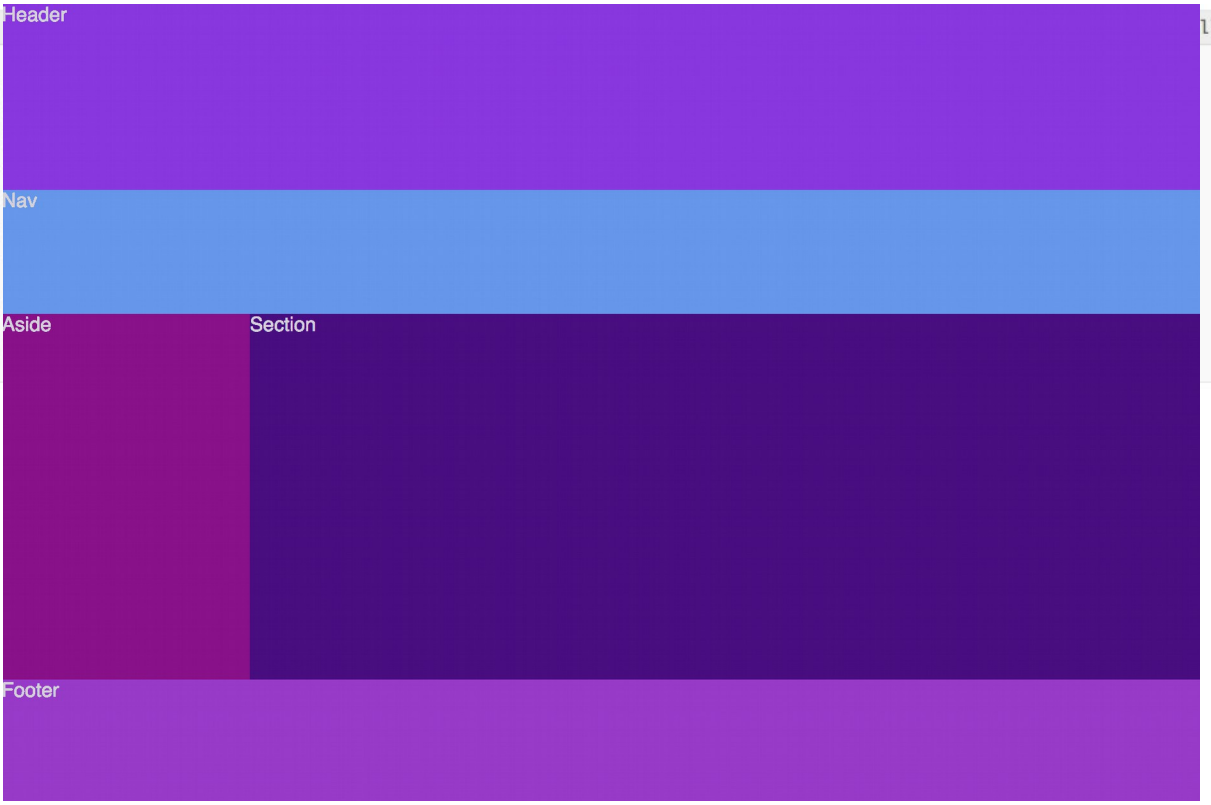
```
header {
  grid-area: my-header;
  background-color: blueviolet;
}

nav {
  grid-area: my-nav;
  background-color: cornflowerblue;
}

aside {
  grid-area: my-aside;
  background-color: darkmagenta;
}

section {
  grid-area: my-section;
  background-color: indigo;
}

footer {
  grid-area: my-footer;
  background-color: darkorchid;
}
```

Y la magia la veremos en el contenedor (main) con **`grid-template-areas.`**

```
grid-template-areas:
  "my-header my-header my-header"
  "my-nav my-nav my-nav"
  "my-section my-section my-section"
  "my-aside my-aside my-aside"
  "my-footer my-footer my-footer"
```

De forma que **en el código veremos cómo quedaría nuestro layout gráficamente**. Y contamos con la ventaja de que el orden de los elementos en HTML es completamente independiente. ¿Os imagináis cómo haremos para reordenar los elementos en tableta? Pues introduciremos una media query y simplemente añadiremos el siguiente CSS:

Visualmente tendríamos lo siguiente:



Y por último nos queda el media query para desktop:

```
@media (min-width: 1001px) {
  main {
    grid-template-columns: 200px 1fr 200px;
    grid-template-rows: 200px
      1fr
      100px;
    grid-template-areas:
      "my-header my-header my-header"
      "my-nav my-section my-aside"
      "my-footer my-footer my-footer"
  }
}
```

Repeat

Entonces, con grid-template-columns si quiero crear una grilla de 12 columnas tendría que hacerlo tal que así:

```
grid-template-columns: 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr;
```

Pues no, dado que tenemos la función **`repeat()`**, cuyo primer argumento es el número de repeticiones y el segundo argumento es la unidad a repetirse:

```
grid-template-columns: repeat(12, 1fr);
```

Responsividad avanzada

Imaginemos ahora que queremos que las celdas ocupen 200px y que se coloquen en una fila siempre y cuando haya espacio. Esto lo podremos hacer con **`auto-fit`**.

```
grid-template-columns: repeat(auto-fit, 200px);
```

De esta forma el número de elementos será dinámico, contando que haya 200px libres. Pero el problema ahora es que van a quedar espacios en blanco en ciertos puntos:

Por esta razón tenemos **`minmax();`** una función que especifica un mínimo y un máximo. Nuestro mínimo en este caso serían 200px y el máximo sería 1fr, de esta forma si hay menos de 200px libres los elementos abarcarían todo el espacio disponible y en el momento en el que hay 200px libres se añade un elemento:

```
grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
```