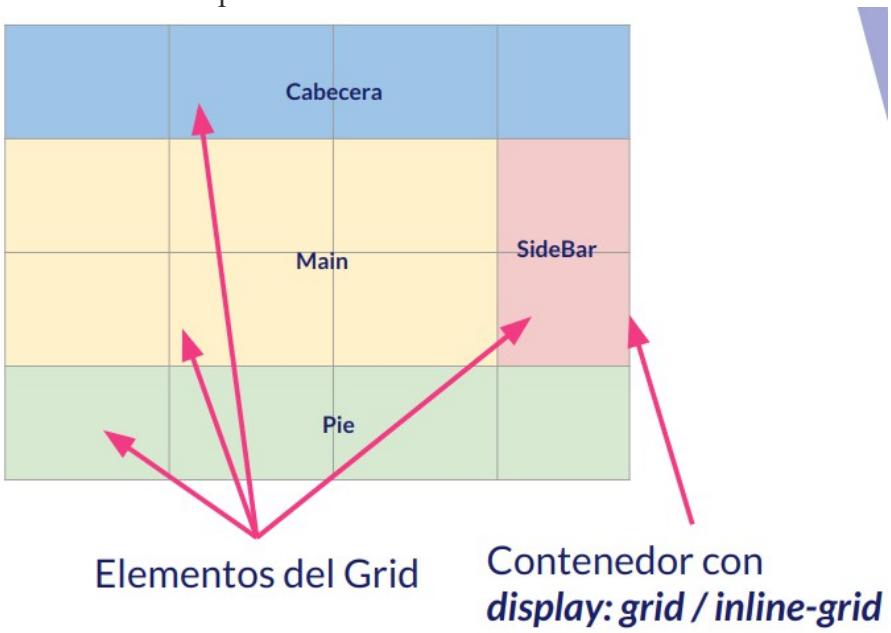
Módulo GRID

Con grid vamos a poder maquetar en dos dimensiones.

Distinguimos dos tipos de elementos:

El **contenedor GRID** que tendrá asignada la propiedad CSS *display:grid* o *display:inline-grid* . Esta última nos permite tener varios elementos grid en línea, lo normal es que tengamos sólo un elemento grid (la primera opción).
Los **elementos GRID** que son los elementos que están dentro del contenedor, elementos que distribuiremos y cuyas propiedades modificaremos.
De manera visual podemos ver una distribución de los elementos en la siguiente imagen:



Podremos modificar desde el contenedor:

- La estructura en filas y columnas y la separación entre ellas.
- Definir áreas del GRID con nombre.
- La alineación horizontal y vertical de los elementos del GRID y del propio GRID dentro del elemento que lo contiene.

Es decir, vamos a poder controlar propiedades que usamos para maquetar y además, vamos a poder maquetar de manera mucho más ágil que con otras técnicas tradicionales de maquetado.

Para empezar a maquetar usando GRID lo primero que tenemos que hacer es definir cuál de nuestras etiquetas HTML se va a convertir en el **contenedor GRID**. Una vez lo hemos decidido le daremos una de estas propiedades:

- **display:grid** si queremos que nuestra rejilla (nuestro grid) sea un elemento de bloque.
- **display:inline-grid** si queremos que nuestro grid sea un elemento en línea.

Una vez hemos asignado esta propiedad al contenedor, todos los elementos que contiene pasan a convertirse de manera automática en elementos del GRID cuya colocación y propiedades podremos empezar a modificar desde el contenedor.

Definición de la estructura del GRID

Normalmente el primer paso que daremos para maquetar con GRID es definir la estructura que va a tener nuestra rejilla. Es un paso importante y para evitar problemas después es conveniente planificarlo bien.

Una vez hemos decidido que estructura queremos usaremos una serie de propiedades para definirla. Las más importantes para empezar son las siguientes:

- **grid-template-columns:** Para definir el número y tamaño de las diferentes columnas de mi estructura. Debo de poner tantos valores de anchura como columnas quiero que tenga el GRID.
- **grid-template-rows:** Para definir el número y tamaño de las diferentes filas de mi estructura. Debo de poner tanto valores de altura como filas quiero que tenga el GRID.
- **grid-row-gap:** Para establecer la separación entre las diferentes columnas.
- **grid-column-gap:** Para establecer la separación entre las diferentes filas.

En los dos últimos simplemente estamos expresando distancias pero los dos primeros tienen muchas posibilidades así que vamos a mostrar varios ejemplos:

Ejemplos con columnas:

Tres columnas que se reparten el 100% del contenedor
```
grid-template-columns: 20% 50% 30%;
```

Cuatro columnas. Tres de tamaño fijo 100px y la otra ocupa el espacio libre restante
```
grid-template-columns: 100px auto 100px 100px;
```

Cuatro columnas. Todas con un tamaño igual
```
grid-template-columns: auto auto auto auto
```

Ejemplos con filas:

Tres filas que se reparten toda la altura que tiene el contenedor
```
grid-template-rows: 20% 50% 30%;
```

Cuatro filas. Tres de altura fija 100px y la otra ocupará el resto del espacio libre hasta llenar todo el contenedor en altura.
```
grid-template-rows: 100px auto 100px 100px;
```

Cuatro filas que se reparten de manera equitativa el alto del contenedor
```
grid-template-rows auto auto auto auto;
```

En estas dos propiedades también puedo repetir valores y usar la unidad fr que me sirve para establecer ratios para que los elementos se repartan el espacio restante. Podemos verlo mejor con un par de ejemplos.

Cuatro columnas. Tres de 20%. Y la último que ocupará el resto del espacio libre
```
grid-template-columns: repeat(3, 20%) auto;
```

Cuatro columnas. Una de tamaño fijo y las demás se reparten el espacio libre en 5 partes de la siguiente manera (2+1+2)

```
grid-template-columns: 2fr 100px 1fr 2fr;
```
fr es una fracción del espacio disponible.

Aunque puede parecer complejo es sencillo y con muy pocas líneas podemos conseguir estructuras complejas. Para verlo vamos a poner un ejemplo:
Suponed que tengo el siguiente HTML:

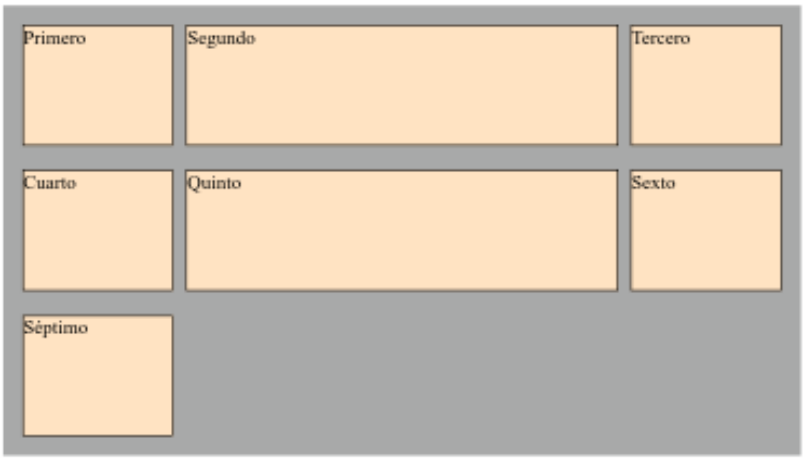
```
<div class="container">
  <div>Primero</div>
  <div>Segundo</div>
  <div>Tercero</div>
  <div>Cuarto</div>
  <div>Quinto</div>
  <div>Sexto</div>
  <div>Séptimo</div>
</div>
```

Usando solo el siguiente CSS:

```
.container {
  background-color: #aaa;
  display: grid;
  grid-column-gap: 10px;
  grid-row-gap: 20px;
  grid-template-columns: 20% auto 20%;
  grid-template-rows: repeat(3, 100px);
  margin: 20px auto;
  padding: 1em;
  width: 80%;
}

.container > div {
  background-color: bisque;
  border: 1px solid black;
}
```

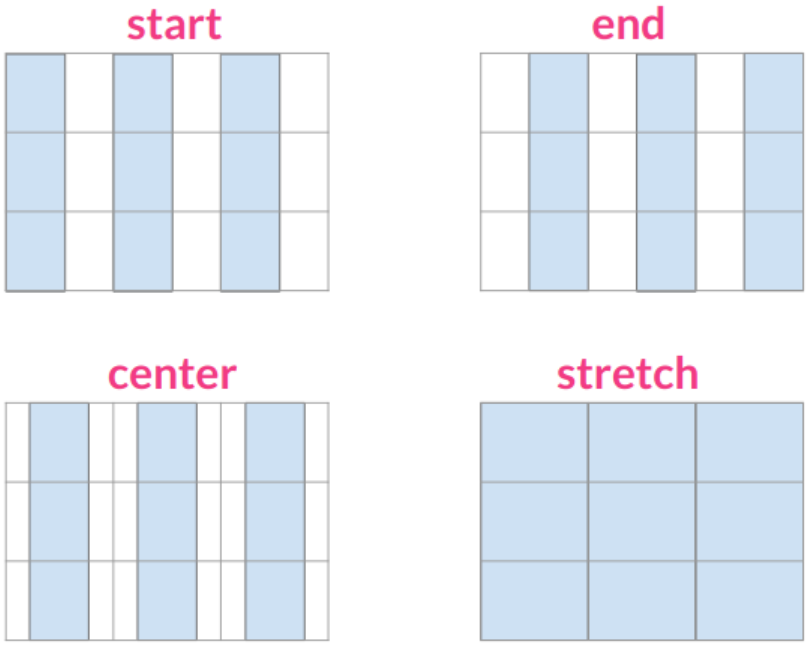
Obtengo la siguiente estructura:



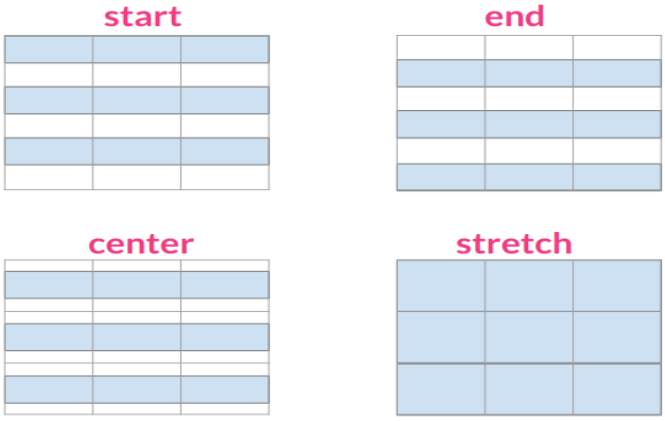
Vemos cómo se han distribuido los 7 elementos en una cuadrícula, en un grid de 3x3 siguiendo la estructura que le hemos dicho.

Alineación Horizontal
Por defecto los elementos del GRID ocupan todo el ancho de la celda que le corresponde pero podemos optar por otro tipo de alineaciones horizontales dando valores a la propiedad **justify-items**.
Los diferentes valores que puede tomar son los siguientes: **start, end, center y stretch** que es la opción por defecto.

Se entenderá mejor que hace cada uno mediante una explicación visual:

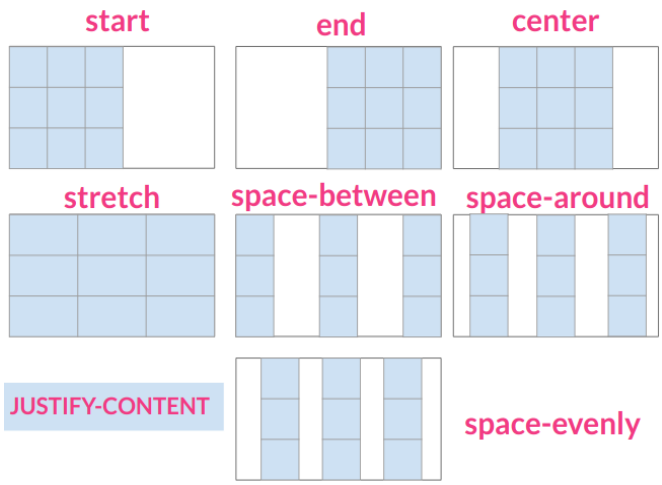


Alineación Vertical
Muy similar a lo anterior. Por defecto los elementos del GRID ocupan todo el alto de la celda que le corresponde pero podemos optar por otro tipo de alineaciones verticales dando valores a la propiedad **align-items**.
Los diferentes valores que puede tomar son los siguientes: **start, end, center y stretch** que es la opción por defecto.
Se entenderá mejor que hace cada uno mediante una explicación visual:



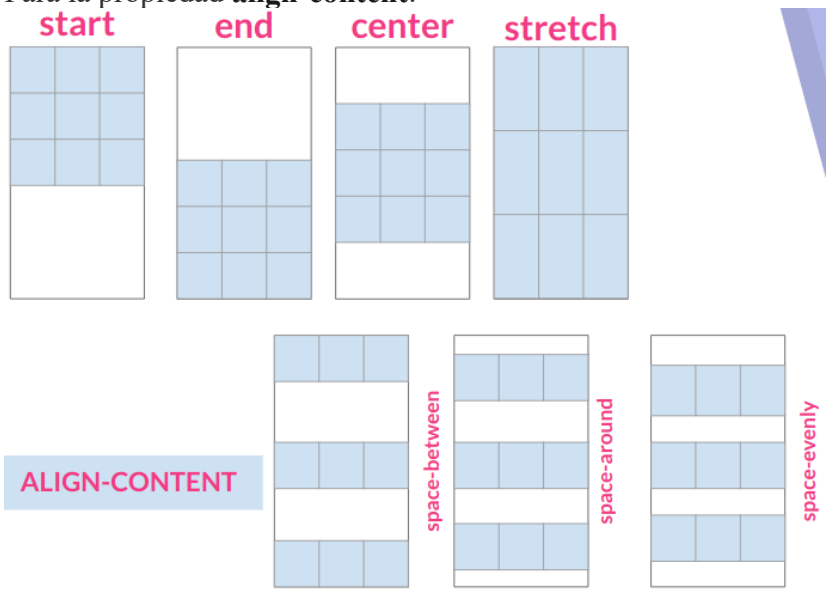
Si quiero juntar estas dos últimas alineaciones usaré la propiedad **place-items** indicando primero el valor para **align-items** y después el valor para **justify-items**.

Distribución dentro del contenedor
En determinados casos puede suceder que los elementos del GRID no ocupen todo el ancho o todo el alto del contenedor GRID. En estas ocasiones puedo distribuir las columnas y las filas usando las propiedades **justify-content** (horizontal) y **align-content** (vertical). Ambas puede tomar los mismos valores y para entender mejor esos valores y cómo funcionan vamos a presentar dos imágenes.
Para la propiedad **justify-content**:


start
end
center
stretch
space-between
space-around
JUSTIFY-CONTENT
space-evenly

.

Para la propiedad **align-content**:



La zona azul representa las columnas que están dentro de un rectángulo que es el contenedor GRID.
Si quiero juntar estas dos últimas alineaciones usaré la propiedad **place-content** indicando primero el valor para **align-content** y después el valor para **justify-content**.

PROPIEDADES EN LOS ELEMENTOS GRID

Los **Elementos GRID** son los hijos directos, dentro del árbol DOM de nuestra página, del elemento con la propiedad CSS ***display:grid***.

De manera individual podemos modificar las propiedades de estos elementos para conseguir lo siguiente:

- Definir el área o zona del GRID (rejilla) que va a ocupar.
- Especificar la alineación horizontal del elemento.
- Especificar la alineación vertical del elemento.

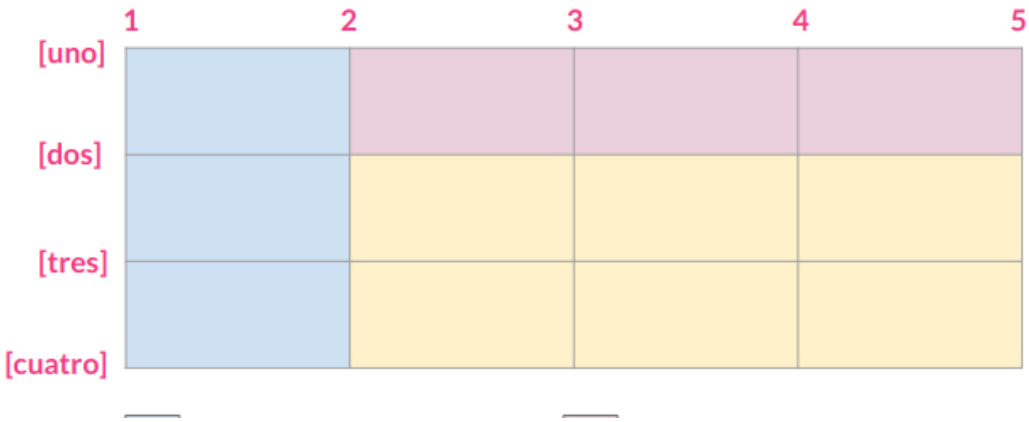
Área del elemento GRID

Para especificar el área que va a ocupar el elemento GRID lo haremos con las siguientes propiedades:

- **grid-column-start**
- **grid-column-end**
- **grid-row-start**
- **grid-row-end**

El significado de cada uno de ellos es prácticamente una traducción directa, pero lo vamos a entender mejor con un ejemplo:

Si tenemos el siguiente GRID:



Para definir el área de los elementos del GRID, que son tres, usaremos el siguiente CSS:

```
#azul {
  background-color: blue;
  grid-column-start: 1;
  grid-column-end: 2;
  grid-row-start: uno;
  grid-row-end: cuatro;
}

#rojo {
  background-color: red;
  grid-column-start: 2;
  grid-column-end: 5;
  grid-row-start: uno;
  grid-row-end: dos;
}

#amarillo {
  background-color: yellow;
  grid-column-start: 2;
  grid-column-end: 5;
  grid-row-start: dos;
  grid-row-end: span 2;
}
```

Vemos que se puede especificar el área que ocupa de tres maneras principalmente:

- Indicando el número de línea donde empieza y donde acaba.
- Indicando con nombres de líneas, que habremos definido al definir el contenedor, donde empieza y donde acaba.
- Indicando cuánto ocupa en la dirección en cuestión (fila o columna) y usando **span** y el valor de la extensión.

Podemos juntar estas propiedades:

Para juntar las dos propiedades referentes a columnas
grid-column: start / end;

Para juntar las dos propiedades referentes a filas
grid-row: start / end;

Para juntar las cuatro propiedades que nos permiten definir el área
grid: row-start column-start row-end column-end;

Alineación horizontal
La alineación horizontal de un elemento de manera individual se consigue usando la propiedad **justify-self** que puede tomar los mismos valores y funciona igual que la propiedad **justify-items** que dábamos al contenedor GRID. Estos valores son: start, end, center y stretch (por defecto).

Alineación vertical
La alineación vertical de un elemento de manera individual se consigue usando la propiedad **align-self** que puede tomar los mismos valores y funciona igual que la propiedad **align-items** que dábamos al contenedor GRID. Estos valores son: start, end, center y stretch (por defecto)

Propiedades de los elementos HIJOS (Grid Items)

Propiedad / valores	Descripción

`grid-column-start`
`grid-column-end`
número | nombre |
span número | span nombre | auto
	Determina la ubicación de un elemento de cuadrícula dentro de la cuadrícula haciendo referencia a líneas verticales de la cuadrícula.
	
	`span numero` indica el número de columnas que queremos que ocupe el elemento

`grid-row-start`
`grid-row-end`
número | nombre |
span número | span nombre | auto
	Determina la ubicación de un elemento de cuadrícula dentro de la cuadrícula haciendo referencia a líneas horizontales de la cuadrícula.

`grid-column`
`grid-row`
start-line / end-line
start-line / span valor
	Propiedad abreviada (Shorthand) de `grid-column-start` / `grid-row-end` y de `grid-row-start` / `grid-row-end`
	
	`start-line` es el número de fila donde comienza el elemento.
	`end-line` es el número de fila donde termina el elemento.
	
	Podemos mezclar valores numéricos de línea, con nombres de línea o con `span valor`

`grid-area` nombre | row-start/column-start/ row-end/column-end	Da un nombre a un elemento para que pueda ser referenciado por una plantilla creada con la propiedad `grid-template-areas`.
`justify-self` start | end | center | stretch	Alinea el contenido dentro de un elemento de cuadrícula a lo largo del eje de la fila (en oposición a `alinear-self` que se alinea a lo largo del eje de la columna). Este valor se aplica al contenido dentro de un único elemento hijo (grid item).
`align-self` start | end | center | stretch	Alinea el contenido dentro de un elemento de cuadrícula a lo largo del eje de la columna (en contraposición a `justificar-self` que se alinea a lo largo del eje de la fila). Este valor se aplica al contenido dentro de un único elemento hijo (grid item).

Propiedades del contenedor grid padre

Propiedad / valores	Descripción

`display`
grid | inline-grid | subgrid
 	
`.container { display: grid; }`
	Define un elemento como contenedor Grid y establece el contexto de rejilla necesario para los elementos hijos que estén en su interior.

`grid-template-columns`
track-size ... | [line-name] track-size ...
 	
`.container { grid-template-columns: 1fr 1fr; }`
	Define las columnas de la cuadrícula con una lista de valores separados por espacios. Los valores representan el tamaño de la columna y el espacio entre ellos representa una línea de cuadrícula. El ejemplo crea 3 columnas.
	
	`track-size` en % o en px o en fr (fracción de espacio). Si queremos indicar un número de columnas iguales podemos utilizar el valor `repeat(numero, medida)`

`grid-template-rows`
track-size ... | [line-name] track-size ...
 	
`.container{ grid-template-columns: 40px auto 50%; grid-template-rows: 25% 100px auto; }`
	Define las filas de la cuadrícula con una lista de valores separados por espacios. Los valores representan el tamaño de la fila y el espacio entre ellos representa una línea de cuadrícula.
	
	El ejemplo crea una cuadrícula 3 columnas y 3 filas, con las dimensiones indicadas.

grid-template-areas "grid-area-name | . | none ..." "..." `.item-a { grid-area: cabecera; } .item-b { grid-area: cuerpo; } .item-c { grid-area: pie; } .container { grid-template-columns: 1fr 1fr 1fr; grid-template-rows: auto; grid-template-areas: "cabecera cabecera cabecera" "cuerpo cuerpo cuerpo" ". pie pie"; }`	Define un área de cuadrícula haciendo referencia a los nombres de las áreas de cuadrícula que se especifican con la propiedad `grid-area`. Hay que poner las comillas indicadas. El punto deja un lugar en blanco. El ejemplo crea una rejilla de 3 columnas y 3 filas. Toda la fila superior estará formada por la cabecera. Toda la fila del medio estará formada por el cuerpo. La tercera fila estará formada por una celda vacía (el punto) y el pie ocupará las dos últimas fracciones de espacio. Cada fila en su declaración debe tener el mismo número de celdas. Se puede usar cualquier número de puntos adyacentes para declarar una sola celda vacía. Mientras los puntos no tengan espacios entre ellos, representan una sola celda.

`grid-template`
none | grid-template-rows / grid-template-columns
 	
`.container { grid-template: 100px 1fr 50px / 1fr 1fr; }`
	Propiedad abreviada (shorthand) de `grid-template-rows`, `grid-template-columns`, y `grid-template-areas` en una sola declaration.
	
	El valor `none` establece las tres propiedades a sus valores iniciales.
	
	El ejemplo crea una rejilla de 3 filas de alto distinto y dos columnas iguales.

`grid-column-gap` ó `grid-row-gap`
numero_con_unidades
 	
`.container { grid-template-columns: 200px 50px 100px; grid-template-rows: 180px auto 180px; grid-column-gap: 20px; grid-row-gap: 15px; }`
	Especifica la separación entre las columnas y/o las filas. La especificación W3C indica que es el ancho de las Grid-Lines.
	
	Unidades: cualquier valor de unidad válido en css.

`grid-gap`
numero_separacion_filas numero_separacion_columnas
 	
`.container{ grid-template-columns: 100px 50px 100px; grid-template-rows: 50px auto 100px; grid-gap: 10px 15px; }`
	Propiedad abreviada (shorthand) de grid-row-gap y grid-column-gap. Observa que primero son las filas y segundo las columnas.

`grid-auto-columns` ó `grid-auto-rows` track-size ...	Especifica el tamaño de las celdas de cuadrícula generadas automáticamente (implícitamente).
`grid-auto-flow` row | column | row dense | column dense|	Si hay elementos de cuadrícula que no quedan colocados explícitamente en la cuadrícula, el algoritmo de *auto-placement* actúa para colocar automáticamente los elementos. Por defecto es `row`. `dense` : indica al algoritmo de colocación automática que intenta llenar los hecos anteriores en la cuadrícula si aparecen elementos más pequeños despues.

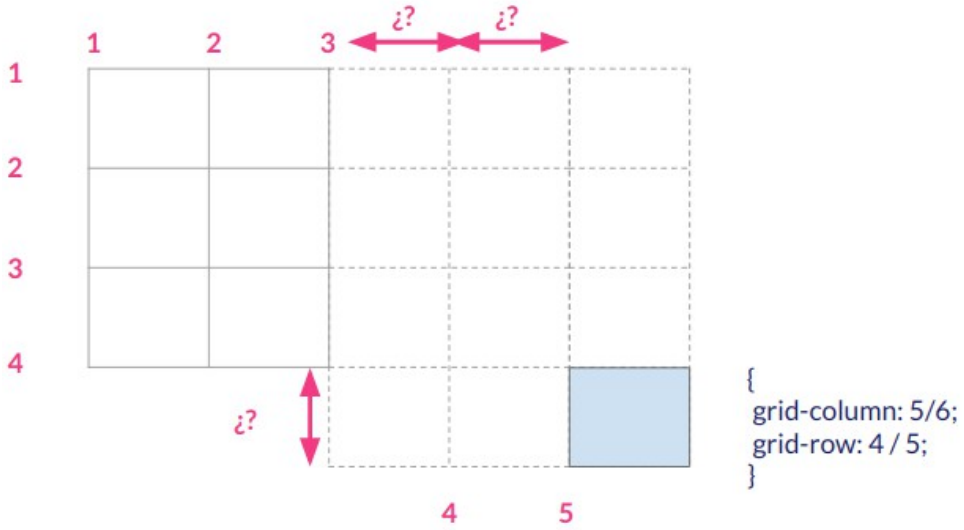
`justify-content`
`start | end | center | stretch | space-around | space-between | space-evenly`
 	
`align-content`
`start | end | center | stretch | space-around | space-between | space-evenly`
	A veces, el tamaño total de la cuadrícula puede ser menor que el tamaño de su contenedor Grid. Esto puede ocurrir si todos los elementos de la cuadrícula están dimensionados con unidades absolutas como px. En este caso, podemos configurar la alineación de la cuadrícula dentro del contenedor de cuadrícula.
	
	Esta propiedad alinea la cuadrícula respecto al eje row (opuesto a lo que hace la propiedad `align-content` que lo hace respecto al eje column).

`justify-items`
`start | end | center | stretch`
 	
`align-items`
`start | end | center | stretch`
	Alinea el contenido dentro de un elemento de cuadrícula respecto del eje row (opuesto a lo que hace `align-items` que lo hace respecto al eje column).
	
	Este valor se aplica a todos los elementos de la cuadrícula que hay dentro del contenedor padre. Esto se puede hacer de forma individual con la propiedad `align-self` de los grid-itmes.

Colocación Implícita. La **colocación implícita** de los **elementos GRID** es lo que sucede cuando colocamos esos elementos **fuera** de la estructura del contenedor o cuando no les damos posición. Cuando colocamos un elemento fuera de la estructura definida para su contenedor GRID podemos mantener cierto control añadiendo al contenedor (que no al elemento) las siguiente propiedades:

- **grid-auto-columns** Que dará tamaño a las columnas de separación entre los límites del contenedor y la posición donde hemos dejado nuestro elemento.
- **grid-auto-rows** Que dará tamaño a las filas de separación entre los límites del contenedor y la posición donde hemos dejado nuestro elemento.

Lo vamos a ver mejor en un ejemplo:




```
.container {
  grid-auto-columns: 50px;
  grid-auto-rows: 50px;
}
```

(por defecto 0)

Al dar al contenedor las siguiente propiedades:

```
.container {
  ... grid-auto-columns: 50px;
  grid-auto-rows: 50px;
  ...;
}
```

Las filas y las columnas con los interrogantes, que están fuera del grid que es 2x3, tendrán unas dimensiones de 50x50. Por defecto estos valores son 0.
En el caso de que no hayamos especificado el área que le corresponde a un elemento de GRID podemos establecer ciertas reglas del comportamiento mediante la propiedad **grid-auto-flow** que añadiremos al contenedor. Esta propiedad puede tomar varios valores:

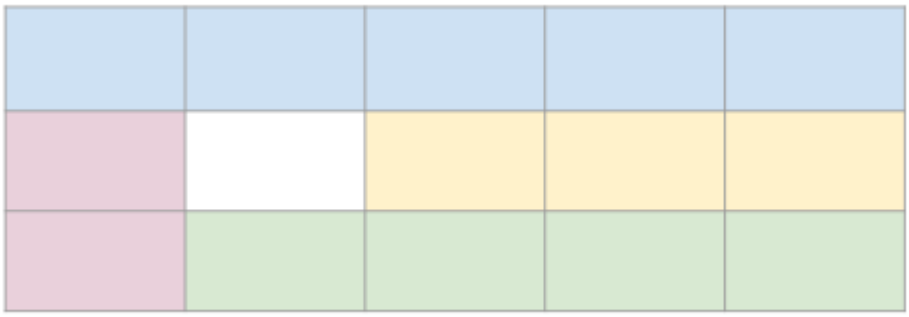
- **row:** Rellena primero las filas. Es la opción por defecto.
- **Column:** Rellena primero las columnas. Es la opción por defecto.

Maquetar únicamente nombrando áreas

Para ello usaremos las siguiente propiedades:

- **grid-area** en los elementos GRID
- **gird-template-area** en el contenedor GRID.

Vamos a verlo mejor con un ejemplo. Si tenemos el siguiente GRID:



Se ha establecido la estructura del contenedor GRID de la siguiente manera:

```
.container {
  grid-template-columns: repeat(5, 20%);
  grid-template-rows: repeat(3, 100px);
}
```

Daremos nombre al área que va a ocupar cada elemento del grid mediante la propiedad **grid-area**:

```
#cab {
  background-color: blue;
  grid-area: cab;
}

#pie {
  background-color: green;
  grid-area: pie;
}
```

```
#menu {
  background-color: red;
  grid-area: menu;
}

#principal {
  background-color: yellow;
  grid-area: main;
}
```

Con esos nombre ya podemos añadir al contenedor la propiedad **grid-template-area** que define la estructura sin dar ninguna propiedad adicional a los elementos del Grid.

```
.container {
  display: grid;
  grid-template-columns: repeat(5, 20%);
  grid-template-rows: repeat(3, 100px);
  grid-template-areas:
    "cab cab cab cab cab"
    "menu  .  main main main"
    "menu  pie pie pie pie";
}
```

Cada nombre representa el elemento que va a ocupar una celda. De esta manera, el contenido de cada fila va entre comillas, en este caso:

- La **primera fila** será ocupada totalmente por el elemento con *grid-area:cab*;
- En la **segunda fila**, la primera celda para el elemento con *grid-area:menu*, luego un hueco que se indica con . y posteriormente tres celdas para el elemento con *grid-area:main*.
- En la **tercera fila** la primera celda es para el elemento con *grid-area:menu* y el resto para el que tenga *grid-area:pie*