Flex y Grid layout

Contenido

FLEX y GRID son dos nuevos valores (HTML 5) que vamos a poder dar a la propiedad CSS ***display*** y que nos van a permitir, junto con otras propiedades, maquetar nuestras páginas web de una manera más fácil a la que se usaba tradicionalmente.

FLEX:

- Elemento contenedor con ***display:flex***
- Podremos:
  - Alinear.
  - Dar tamaño.
  - Distribuir el espacio restante.
- Solo permite aplicar cambios en una sola dimensión en cada momento.

Se aplica a todos los elementos que estén dentro sin saber si ni siquiera el tamaño que tiene.

GRID:

- Elemento contenedor con ***display:grid***
- Es el sistema para maquetación más potente.
- Me permite trabajar con filas y columnas (no en una sola dimensión).

Hay que tener cuidado con estas propiedades, podemos encontrarnos navegadores, sobre todo los de los dispositivos móviles, que no las soportan.

1. Maquetando con contenedores Flex

1.1. Elementos de maquetación con CSS Flex

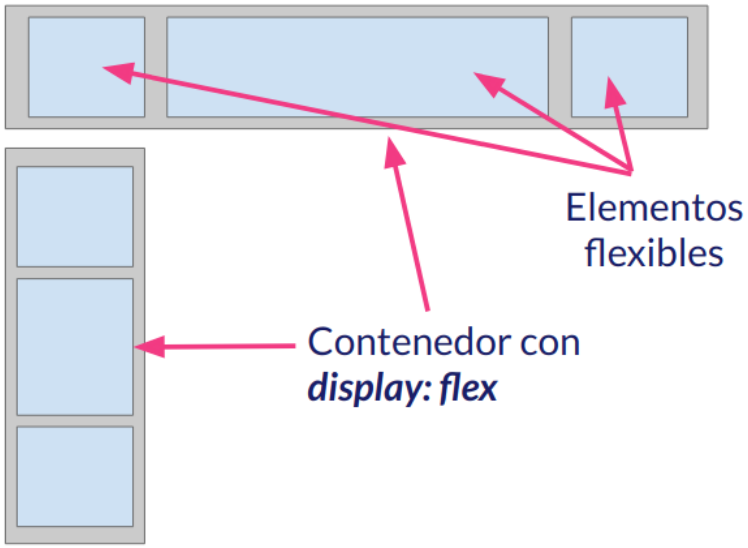
La principal idea que hay detrás de la maquetación **FLEX** es que vamos a tener un elemento, es decir, una etiqueta que va a poder *controlar* las propiedades de los elementos que contiene.

Por lo tanto, en esta situación, vamos a poder distinguir dos tipos de elementos:

- El **contenedor flex** que tendrá asignada la propiedad CSS *display:flex* o *display: inline-flex* y que va a controlar ciertas propiedades de los elementos que contiene. Los vemos en gris en la imagen inferior.

NOTA: La **diferencia entre flex** e **inline-flex** es la misma que **block** e **inline-block**. En la primera el tamaño de las cajas se ajusta a su contenido y no hace caso a *width*, *height* o márgenes verticales. Sin embargo, con **inline-flex**, podremos establecer el tamaño de las cajas y sus márgenes verticales.

- Los **elementos flexibles** que son los elementos que están dentro del contenedor y cuyas propiedades modificaremos. Los vemos en azul en la imagen inferior.



Pero, ¿qué tipo de propiedades podremos modificar? Podremos modificar propiedades CSS como:

- La altura de los elementos flexibles.
- La anchura.
- El orden en el que se nos van a presentar.
- La alineación vertical.
- La alineación horizontal.
- La distribución de los elementos flexibles a lo largo del contenedor.

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

1.2. Propiedades en el contenedor Flex

El primer paso para maquetar usando elementos FLEX es añadir la propiedad FLEX al contenedor. En cuanto hagamos esto vamos a poder comprobar que empiezan a pasar cosas:

Si tengo este HTML:

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

```
    <div><p>Primera frase</p></div>
    <div><p>Segunda frase</p></div>
    <div><p>Tercera frase</p></div>
</div>

/*CSS:*/

.container {
   background-color:aqua;
   display:flex;
   height: 200px;
   width: 60%;
}
```

Simplemente añadiendo la propiedad ***display:flex*** podremos ver que **de manera automática los elementos ajustan su anchura a su contenido y flotan a la izquierda**. Y todo es simplemente poniendo una sola regla CSS.

Además de con esta regla, desde el contenedor flex **puedo modificar de manera directa varias propiedades de los elementos hijos**. Por ejemplo:

* Flex divide el espacio disponible por igual entre los elementos que contiene. Si, por ejemplo, hay 5 elementos, flex asigna un 20% de espacio a cada uno. Si de esos 5 elementos se especifica un width:30% a uno de ellos, flex asigna el 30% de ese 20% de espacio.

- La dirección en la que se van a mostrar.
- Cómo se van a ajustar dentro del contenedor.
- La alineación horizontal de los elementos flexibles.
- La alineación vertical de los elementos flexibles cuando solo ocupan una línea.
- La alineación vertical de los elementos flexibles cuando ocupan más de una línea.

Dirección de los elementos flexibles.

La ajustaremos mediante la propiedad **flex-direction** que podrá tomar los siguientes valores:

- **row:** es la opción por defecto y ajustará los elementos flexibles de izquierda a derecha.
- **row-reverse:** igual que la anterior, pero de derecha a izquierda.
- **column:** ajustará los elementos flexibles en columna, de arriba a abajo.
- **column-reverse:** igual que la de arriba, pero de abajo a arriba.

NOTA: Estamos dando por supuesto que trabajamos con sistemas de escritura occidentales donde el flujo de lectura es de izquierda a derecha y de arriba a abajo. En otros sistemas de escritura los valores por defecto en nuestro navegador podrían ser otros y deberíamos revisar todo con cuidado.

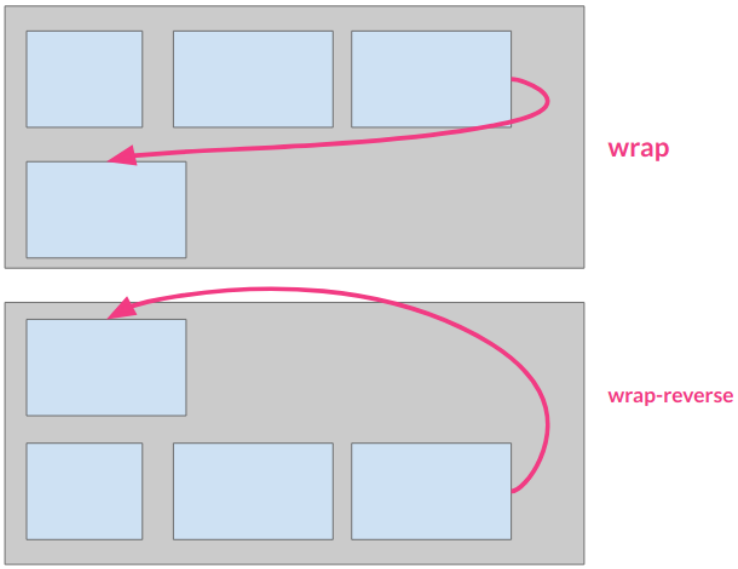
El ajuste de los elementos flexibles

Hemos visto como antes, por defecto y sin indicar ninguna anchura, los elementos flexibles adecuaban su tamaño a su contenido (si no les dábamos anchura) y se ponían todos a la izquierda permaneciendo siempre así, aunque el ancho de la pantalla sea muy pequeño. Esto

puede provocar desajustes en pantallas muy pequeñas. Este contratiempo podemos controlarlo usando la propiedad **flex-wrap** y eligiendo uno de los siguientes valores:

- **no-wrap:** es el valor por defecto y fuerza para que siempre los elementos estén en la misma línea, aunque esto suponga que se salgan del contenedor (les haya dado o no les haya dado anchura).
- **wrap:** provoca un salto de línea si la anchura de los elementos (fijada por nosotros o por el contenedor) es superior a la del contenedor.
- **wrap-reverse:** lo mismo que arriba, pero de abajo a arriba.

Podemos ver el efecto de los últimos valores en la siguiente imagen:



Las dos propiedades, **flex-direction** y **flex-wrap** podemos juntarlas en la propiedad **flex-flow** con dos partes como, por ejemplo:

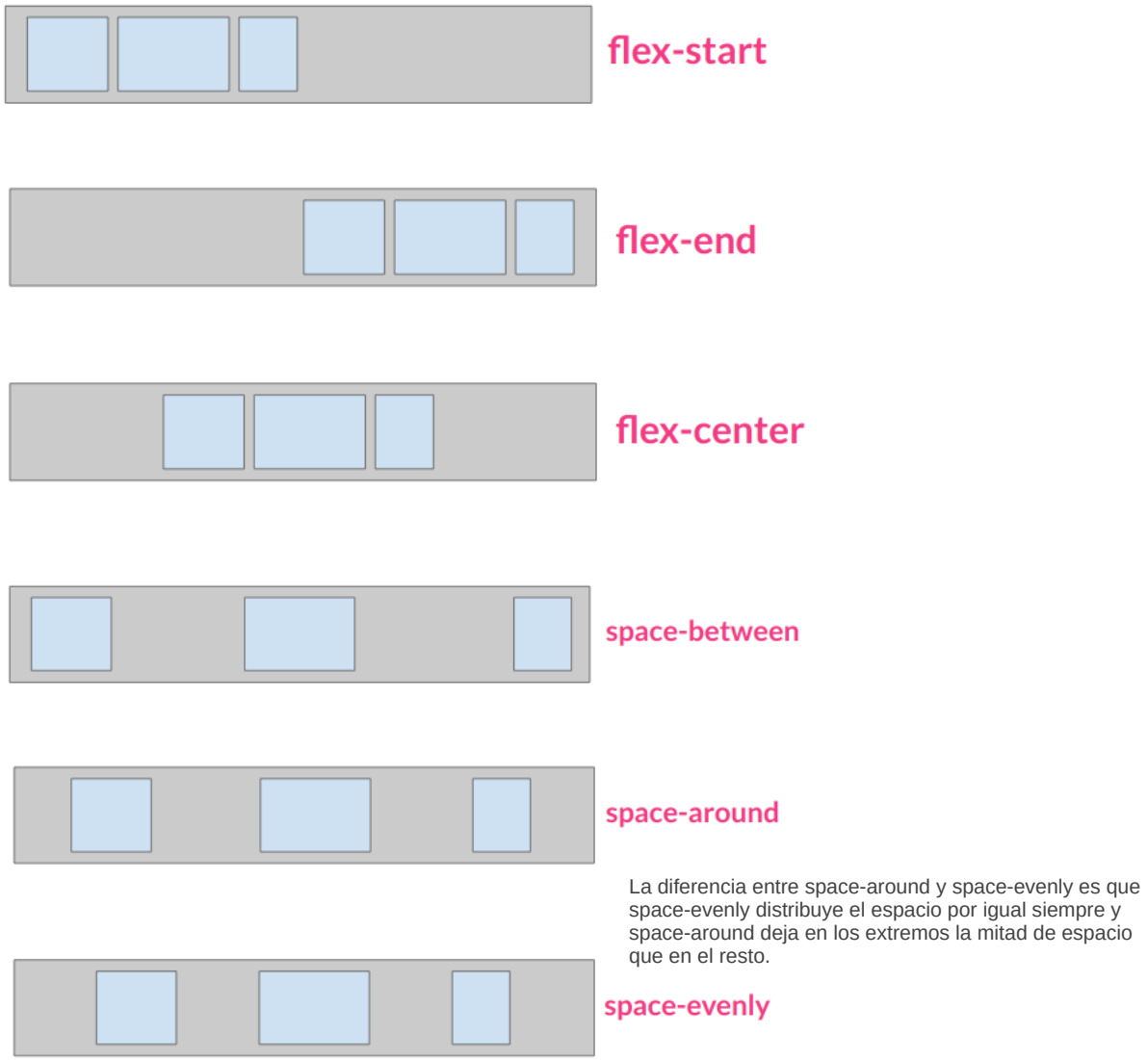
```
flex-flow: row wrap;
```

Alineación horizontal de los elementos flexibles

Podemos alinear **horizontalmente** (***flex-direction: row* | *row-reverse***) los elementos flexibles, tengan o no tengan establecida una anchura, añadiendo la propiedad **justify-content** al elemento contenedor. Esta propiedad puede tener 6 valores distintos:

- **flex-start:** Los elementos flexibles se sitúan al principio.
- **flex-end:** Los elementos flexibles se sitúan al final.
- **center:** Los elementos se centran horizontalmente
- **space-beetween:** Distribuye el espacio restante entre los elementos, pero el primero y el último están en los bordes.
- **space-around:** Distribuye el espacio restante entre los elementos, pero no tiene en cuenta la distancia a los bordes.
- **space-evenly:** Distribuye el espacio restante entre los elementos y tiene en cuenta la distancia a los bordes.

Visualmente:

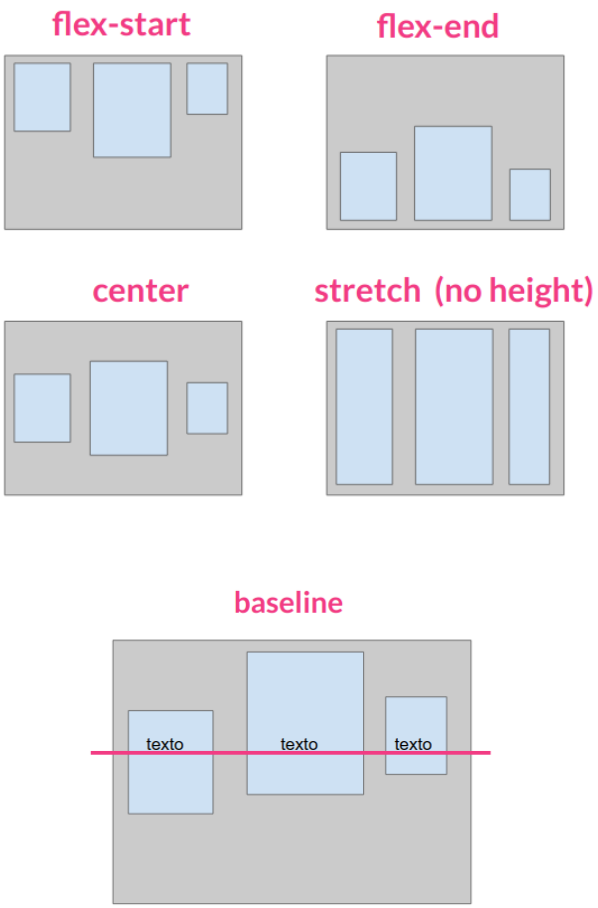


Alineación vertical de los elementos flexibles

Podemos alinear **verticalmente** los elementos flexibles añadiendo la propiedad **align-items** que puede tomar los siguientes valores:

- **flex-start:** Los elementos se ponen junto al borde superior.
- **flex-end:** Los elementos se ponen junto al borde inferior.
- **center:** Los elementos flexibles se centran verticalmente.
- **stretch:** Los elementos crecen en altura para ocupar toda la altura del contenedor flexible. No deben tener altura fija establecida.
- **baseline:** Los elementos se alinean en relación con la primera línea de texto que posean los elementos flexibles.

Visualmente:

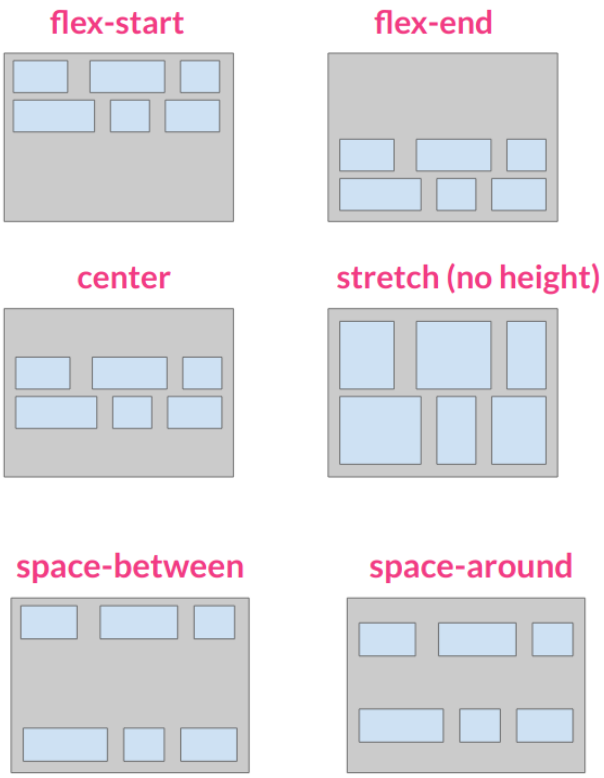


Alineación vertical - Wrap (cuando tengo varias líneas de elementos)

Si he usado la propiedad ***flex-wrap:wrap*** y resulta que tengo varias líneas de elementos flexibles también puedo alinearlas usado la propiedad CSS **align-content** con valores que son análogos a los vistos antes:

- **flex-start**
- **flex-end**
- **center**
- **stretch**
- **space-between**
- **space-around**

Visualmente:



1.3. Elementos flexibles

Además de modificando las propiedades del contenedor FLEX, **podemos actuar sobre los elementos flexibles** modificando otra serie de propiedades relacionadas.

En este apartado nos vamos a ocupar de cómo modificar el orden en el que aparecerán los elementos flexibles y de cómo podemos controlar que se encojan o crezcan.

El orden de los elementos flexibles.

Los elementos flexibles se muestran dentro del contenedor flex en el mismo orden en que están escritos en nuestro código HTML.

Si queremos modificar esto debemos añadir la propiedad CSS ***order*** a los elementos cuyo orden queremos modificar.

Por defecto este valor es 0 y se mostrarán en orden ascendente (los de menor valor primero). En caso de empate se muestra antes el que primero estuviera en el código.

Ajustando el tamaño de los elementos flexibles.

Para controlar el tamaño de los elementos flexibles disponemos de varias propiedades relacionadas interesantes:

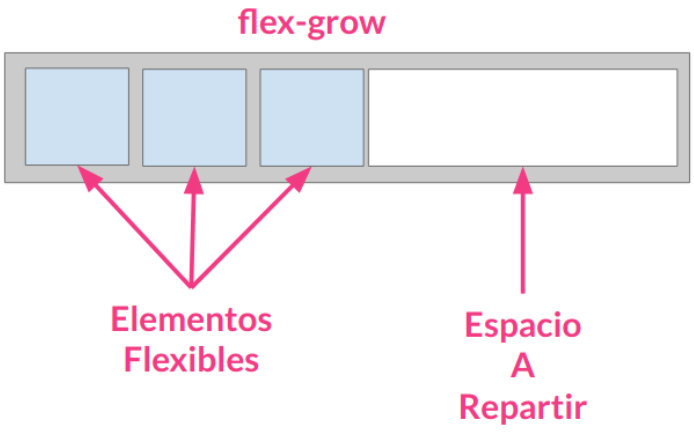
- **flex-grow:** que sirve para indicar, mediante un número, el *factor de crecimiento de un elemento flexible cuando se distribuye entre los elementos flexibles el espacio restante*. Por defecto es 1 pero si quiero que un elemento participe en el reparto debo añadirle esta propiedad.
- **flex-shrink:** que sirve para indicar, mediante un número el factor de contracción de un elemento flexible *cuando el tamaño de todos sobrepasa el tamaño del contenedor*. Por defecto es 1 pero si quiero que un elemento participe en la contracción debo añadirle esta propiedad.
- **flex-basic:** que sirve para indicar el *tamaño de un elemento antes de que el espacio restante (negativo o positivo) se distribuya*. Por defecto el valor de esta propiedad es *auto* y hace que la anchura del elemento flexible se ajuste a su contenido.

Debemos de tener en cuenta que para una correcta maquetación debemos considerar las tres propiedades de manera conjunta. Además, se puede expresar de manera unitaria con la propiedad CSS **flex**. Por ejemplo:

```
flex: grow-factor shrink-factor flex-basis-value;
```

Para una total comprensión de cómo encogen/crecen estos elementos debemos entender qué es lo que se reparte.

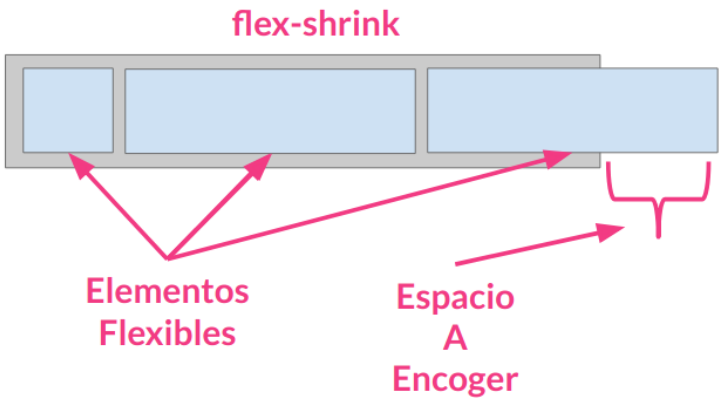
En caso de que los elementos deban crecer (*grow*):



Ese espacio a crecer se repartirá atendiendo al valor que tengan los elementos flexibles en la propiedad ***flex-grow***.

Así, si esos valores fueran: 2,2 y 4 el espacio tendría 8 partes y cada uno crecería la cantidad de partes de su valor.

En caso de que los elementos deban encoger (*shrink*):



Ese espacio a encoger se repartirá atendiendo al valor que tengan los elementos flexibles en la propiedad ***flex-shrink***. **Esto funcionará si no se establece la propiedad flex-wrap:wrap**.

Así si esos valores fueran: 2,2 y 4 el espacio tendría 8 partes y cada uno encogería la cantidad de partes de su valor.

Hay que tener en cuenta que si tenemos *flex-direction:row* o *flex-direction: row-reverse*, el espacio a reducir con *flex-shrink* será el sobrante **solamente** en el eje X. Par reducir el espacio que los elementos flexibles sobrepasan a su contenedor en el eje Y tendremos que tener *flex-direction: column* o *flex-direction: column-reverse*. No podremos reducir el espacio sobrante en ambos ejes de forma simultáne ya que, Flex solo funciona en una dimensión.

1.4. Elementos flexibles: alineación, justificación y espacio libre

En apartados anteriores hemos visto que desde el contenedor FLEX se puede establecer la alineación vertical de todos los elementos flexibles que contiene. Esto lo hacíamos con la propiedad CSS **align-items**.

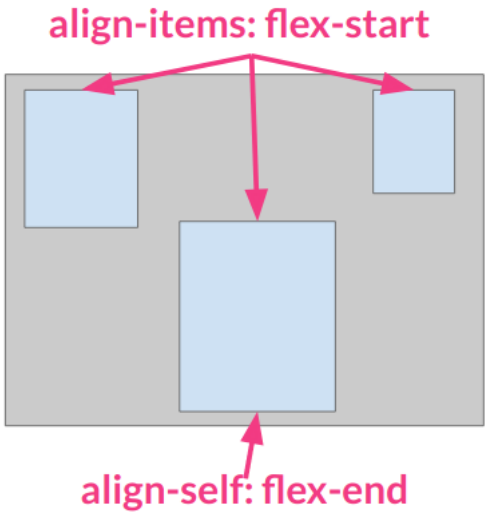
En ocasiones puedo necesitar que un elemento flexible tenga una alineación vertical diferente al resto. En este caso, en el elemento para el que quiero una alineación diferente, debo añadir la propiedad CSS **align-self** que puede tomar los mismos valores (y con el mismo significado) que la propiedad ***align-items***.

- **flex-start**
- **flex-end**
- **center**
- **strecth** (no debe tener altura establecida)
- **baseline**

NOTAS:

- Los elementos flex no hacen caso a la propiedad ***vertical-align***.
- **algin-self** no es compatible con **align-content** en el contenedor.

Podemos ver un ejemplo de todo esto en la siguiente imagen:



2. Maquetando con contenedores Grid

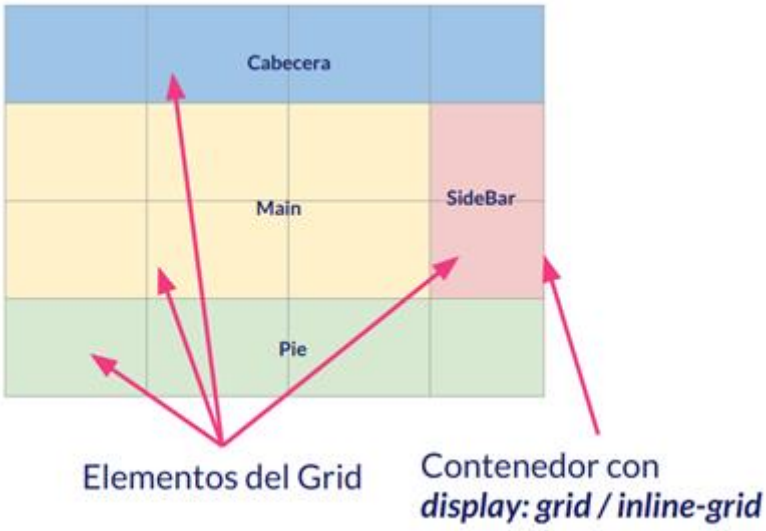
2.1. Elementos de maquetación con CSS Grid

La idea principal que hay detrás de la maquetación **GRID** es que vamos a tener un elemento, es decir, una etiqueta que nos va a permitir *controlar* propiedades de los elementos que contiene y establecer **estructuras complejas** para distribuirlos.

Por lo tanto, en esta situación, vamos a poder distinguir dos tipos de elementos:

- El **contenedor GRID** que tendrá asignada la propiedad CSS *display:grid* o *display:inline-grid* y que guardará la estructura y las propiedades de los elementos que contiene.
- 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:



Pero, ¿qué 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 a lo que lo hacemos con las técnicas tradicionales de maquetado.

2.2. Propiedades en el contenedor Grid

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.

Nos centraremos en el uso de la primera opción que es la más usual.

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 realizar una reflexión sobre ello.

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 poner tanto valores de altura como filas quiero que tenga el GRID.
- **column-gap:** Para establecer la separación entre las diferentes columnas. ***(grid-column-gap está obsoleta)***.
- **row-gap:** Para establecer la separación entre las diferentes filas. ***(grid-row-gap está obsoleta)***.

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;


/* Tres columnas cada una con nombre (entre []). Dos con tamaño fijo y la otra ocupando el espacio restante */
grid-template-colums: [id] 100px [nombre] 300px [apellidos] auto;
```

Ejemplos con filas:

```
    /* Tres filas que se reparten toda la altura del contenedor (la que sea) */
    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;


    /* Tres filas (todas con nombre, entre corchetes) Dos de ellas con tamaño fijo y la restante ocupará todo el alto libre. */
    grid-template-rows: [uno] 100px [dos] 300px [tres] 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% con nombre col-start. Y la último que ocupará el resto del espacio libre */
grid-template-columns: repeat(3, 20% [col-start]) 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 2rf;
```

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>
```

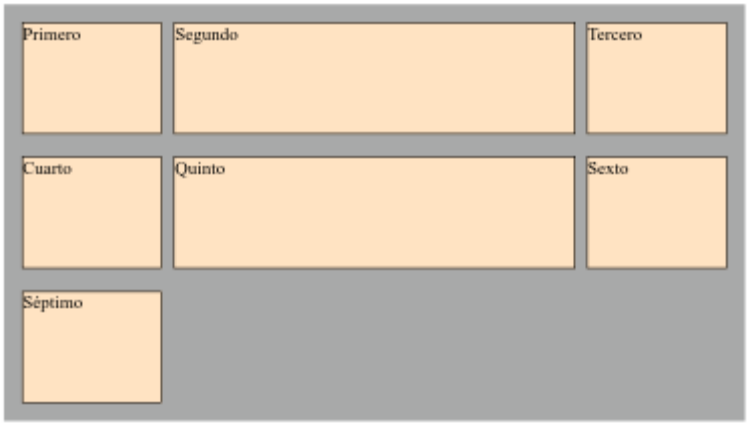
Usando solo el siguiente CSS:

```
.container {
  background-color: #aaa;
  display: grid;
  column-gap: 10px;
  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:



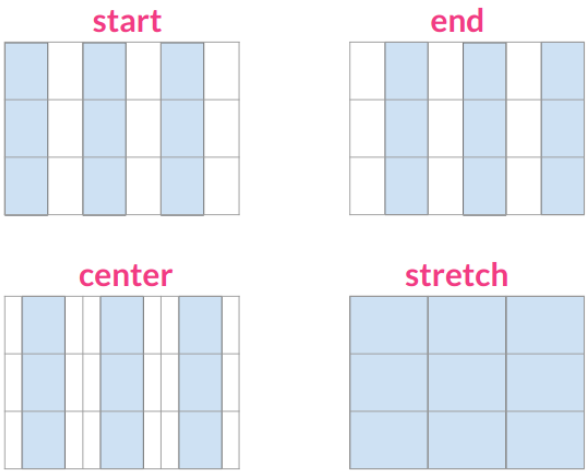
Vemos cómo se han distribuido los 7 elementos en una cuadrícula, en un grid de 3x3 siguiendo la estructura que le hemos dicho.

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**
- **stretch** que es la opción por defecto. Con esta opción no tendremos que especificar tamaño (width o height) a los elementos en bloque.

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

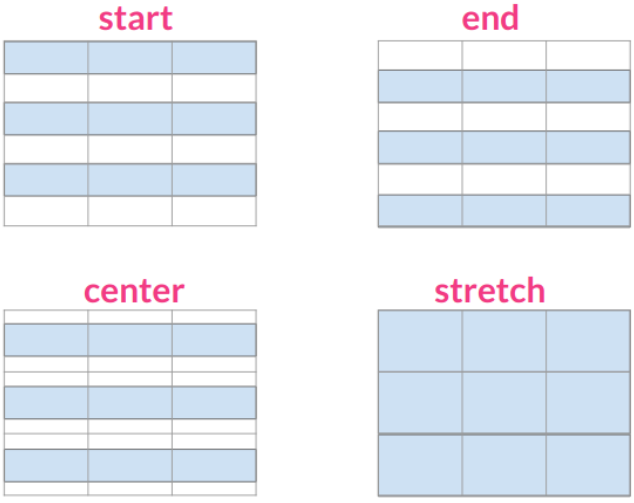


Alineación Vertical

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**
- **stretch** que es la opción por defecto.

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



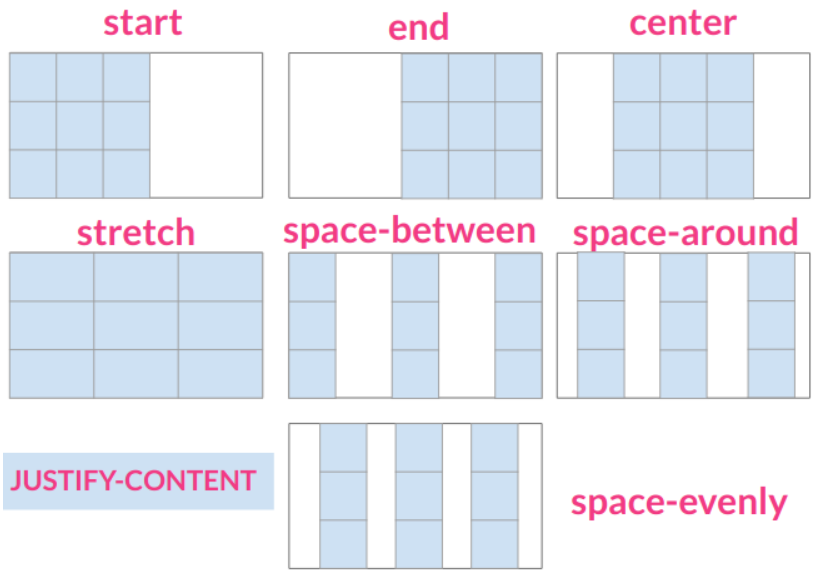
Si quiero juntar estas dos últimas alineaciones usaré la propiedad **place-items** indicando primero el valor para **align-items** y después el valor para **justify-items**.

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**, por ejemplo, el contenedor puede medir 600px x 600px y los elementos del grid 400px x 300px. En estas ocasiones puedo distribuir las columnas y las filas usando las propiedades **justify-content** (horizontal) y **align-content** (vertical). Ambas pueden tomar los mismos valores y para entender mejor esos valores y cómo funcionan vamos a presentar dos imágenes.

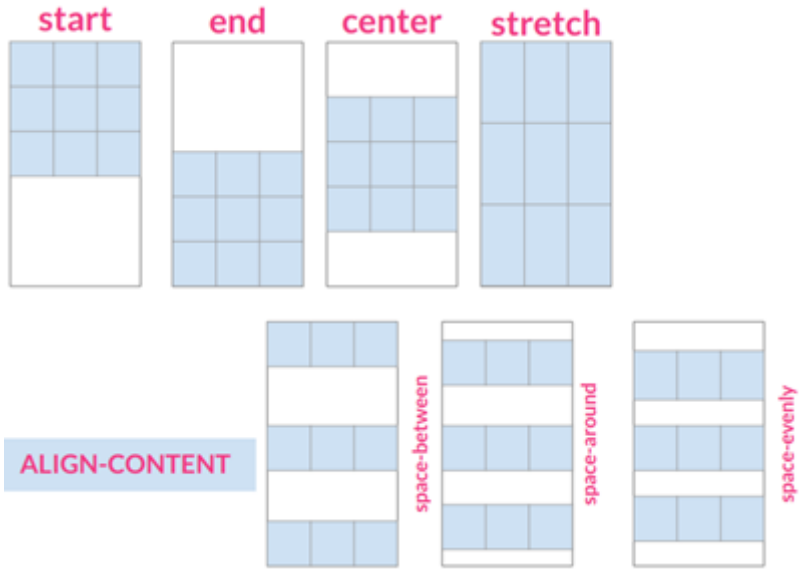
Es importante no confundir justify-item o align-item (colocación del componente dentro de su "celda" del grid) con justify-content o align-content (colocación de las columnas del grid en caso de que sobre espacio).

Para la propiedad **justify-content**:



La zona azul representa las columnas que están dentro de un rectángulo que es el contenedor GRID.

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**.

2.3. Elementos del Grid: posición y tamaño

Los **Elementos GRID** son los hijos directos, dentro del árbol DOM de nuestra página HTML, 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.

Y, además, existen ciertas reglas de colocación implícita que debemos de conocer.

Antes de empezar, hay que tener en cuenta una cosa y es que a los elementos definidos dentro de un contenedor con ***display:grid*** no les afectan las propiedades: ***float***, ***display:inline-block***, ***display-table-cell***, ***vertical-align*** o ***column-****.

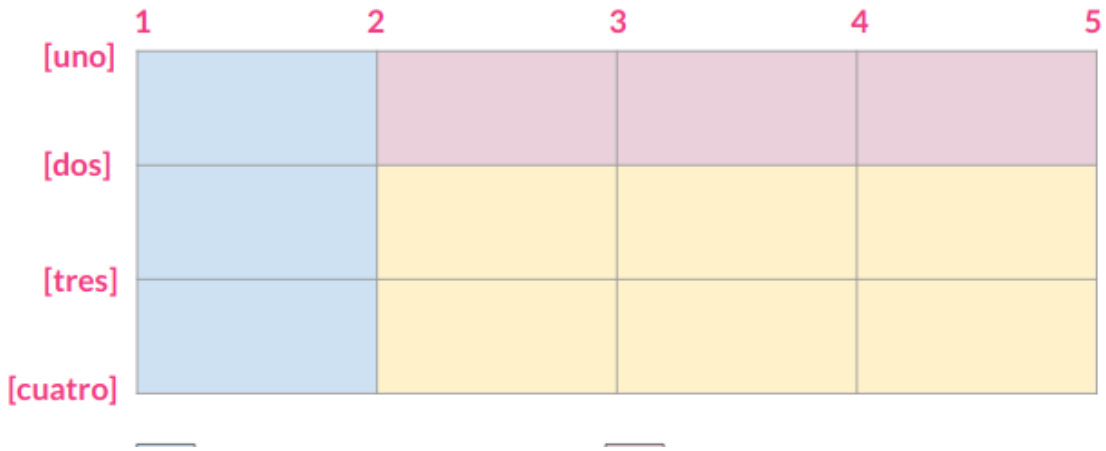
Área del elemento GRID

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;
}
```

Fijaros 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 junta 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
- 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
- stretch (por defecto)

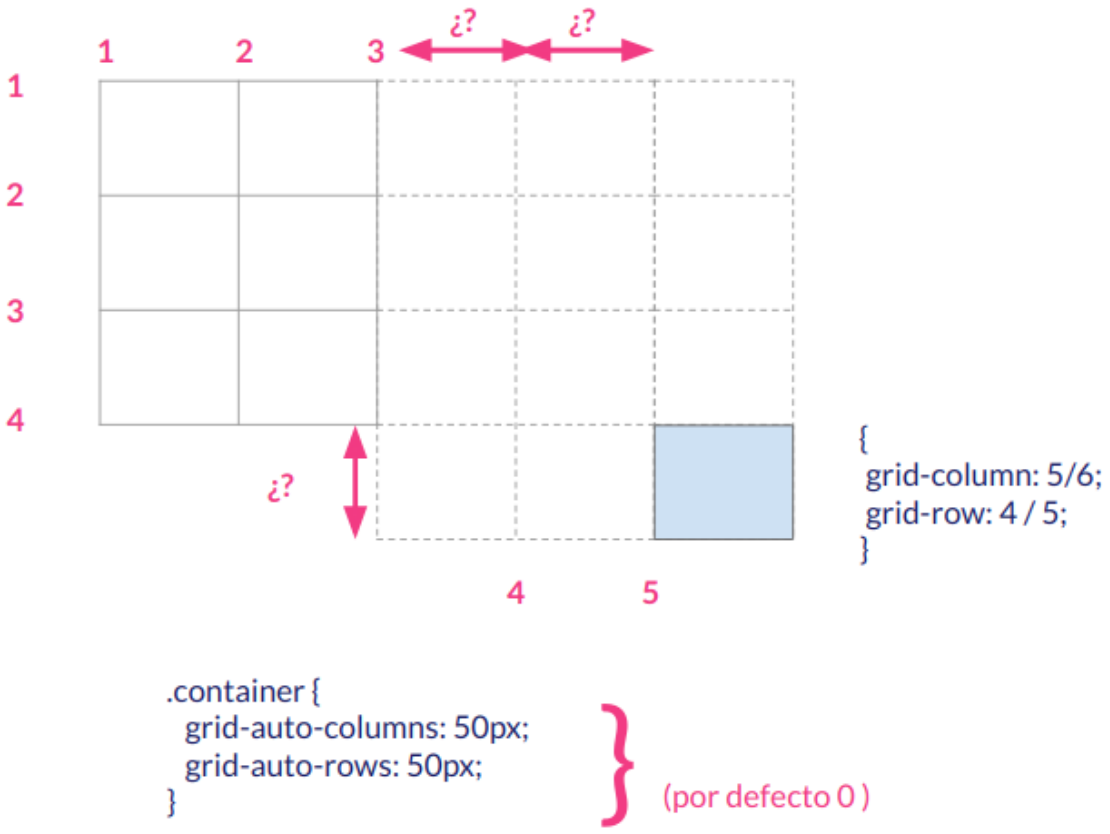
Colocación Implícita

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 con **`grid-template-columns`** o **`grid-template-rows`**. Por ejemplo, si tengo un GRID de 4x4 y coloco un elemento en la posición 7x6.

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 siguientes 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 que está fuera del GRID. Es decir, si tengo un GRID de 4x4 y coloco un elemento en la posición 7x6, con esta propiedad especifico el **ancho** de separación entre la columna 4 de mi GRID y la posición 7, esto es, el ancho de las columnas 5 y 6.
- **grid-auto-rows** Que dará tamaño a las filas de separación entre los límites del contenedor y la posición donde hemos dejado nuestro elemento que está fuera del GRID. Es decir, si tengo un GRID de 4x4 y coloco un elemento en la posición 7x6, con esta propiedad especifico el **alto** de separación entre la fila 4 de mi GRID y la posición 6, esto es, el alto de la fila 5.

Lo vamos a ver mejor en un ejemplo en un GRID de 2x3:



Al dar al contenedor las siguientes 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 **permite ordenar** los elementos grid según unos determinados valores:

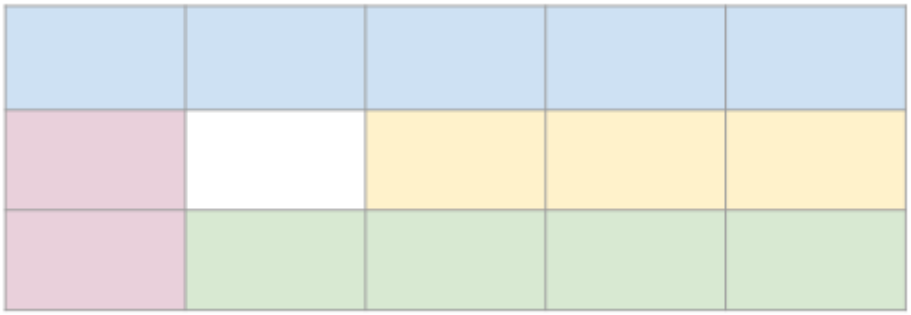
- **row:** Rellena primero las filas. Es la opción por defecto.
- **column:** Rellena primero las columnas. Es la opción por defecto.
- **dense:** Intenta rellenar primero los huecos por si vienen elementos posteriores más pequeños. Hay que tener cuidado al usar ésta opción porque puede provocar cambios de orden en los elementos. **Ejemplo**.

2.4. Áreas Grid

Para acabar con el capítulo dedicado a GRID vamos a ver cómo podemos maquetar únicamente nombrando áreas. Para ello usaremos las siguientes propiedades:

- **grid-area** en los elementos GRID
- **gird-template-areas** 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 nombres 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" /*el “.” Representa un hueco*/
    "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 “” y:

- 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*

Una vez visto esto podemos juntar todas las propiedades grid-template-* en una sola propiedad **grid-template**.

Además, ya no nos quedan por ver más propiedades del contenedor y podríamos juntar todas en la propiedad **grid**. Aunque mucha gente prefiere tener más control y tener todo separado. Son tantas cosas que a veces si las juntamos perdemos el foco.

2.5. ¿Flex o Grid?

Independientemente de qué sistema utilicemos, maquetar usando FLEX o usando GRID es mucho más fácil que maquetar usando sistemas tradicionales basados únicamente en float, position, tablas, etc…

FLEX:

Ventajas:

- Una sola dirección.
- Controlo fácilmente la alineación de los elementos flexibles

Desventajas:

- Para el mismo layout que GRID necesito una estructura HTML más compleja.

GRID:

Ventajas:

- Dos dimensiones.
- Puedo definir muy fácilmente la estructura general.
- Es muy fácil definir la separación entre celdas.

Desventajas:

- La alineación “dentro” de las celdas.

Con todo ello, hay cosas que sólo puede hacer con Flex. Hay cosas que sólo puede hacer con Grid. **¡¡Podemos usar los dos!! Nada impide que un área de un contenedor GRID sea a su vez un contenedor FLEX.**