
UD04 - CSS: Maquetació

Índex

- CSS - Maquetació
 - Floats
 - La propietat `float`
 - La propietat `clear`
 - Clearfix
 - Maquetació amb `floats`
 - Flexbox
 - Propietats del contenidor
 - `flex-direction`
 - `flex-wrap`
 - `flex-flow`
 - `justify-content`
 - `align-items`
 - `align-content`
 - Propietats dels fills
 - `order`
 - `flex-grow`
 - `flex-shrink`
 - `flex-basis`
 - `flex`
 - `align-self`
 - Maquetació amb Flexbox
 - Exercici
 - Grid Layout
 - Conceptes bàsics
 - Propietats
 - `grid-template-columns` i `grid-template-rows`
 - `grid-column` i `grid-row` (`grid-items`)
 - `grid-template-areas`
 - `grid-gap`
 - Disseny Responsiu
 - Disseny Adaptatiu (Elasticitat)
 - Media Queries
 - Mobile-First Approach
 - Referències:
- Javascript - Programació Asíncrona
 - Callbacks
 - Promeses
 - Async/Await

CSS - Maquetació

Floats

La propietat `float`

La propietat *float*, ens permet fer que un element *suri* dins el contingut que l'envolta.

Pot agafar els següents valors:

- *none* (per defecte): L'element no "sura".
- *left*: L'element "sura" a l'esquerra.
- *right*: L'element "sura" a la dreta.



"float: left;"

Quan un element "sura" a dreta o esquerra, passa a col·locar-se al costat corresponent i els elements que el succeeixen (sempre que hi càpiguen) es comencen a col·locar al seu costat.

Pel cas dels elements *inline*, com els paràgrafs (`<p>`) i/o els ``, que poden adaptar la seva amplada a l'espai disponible (desbordant a la següent línia) normalment tot l'espai queda ple adaptant-se a la forma de l'element flotat.

Els elements de bloc, en canvi, cal que tinguin la seva amplada limitada ja que, d'altra manera, tendeixen a ocupar tot l'ample del contenidor.



El propi element flotat, en canvi, encara que sigui un element de bloc, tendirà a ocupar només l'espai necessari per allotjar el seu contingut.

Per altra banda, els elements "flotats" (*float* \neq *none*), tot i desplaçar els elements del seu voltant, **no computen a l'hora d'establir els límits del contenidor** en la dimensió vertical.

Dit d'altra manera: Si la resta del contingut no ho evita, poden sobresortir del seu contenidor solapant-se amb el següent (en el qual exerciran el mateix efecte desplaçant el seu contingut).

La propietat `clear`

La propietat *clear* ens permet prohibir els elements flotats a un (`clear: left`), altre (`clear: right`) o ambdós (`clear: both`) d'un element de bloc.

Així, si tenim un objecte flotant per il·lustrar el contingut d'un paràgraf o d'una secció i no volem que, si el text que realment està relacionat amb ell s'acaba abans, el text del següent paràgraf o secció comenci a renderitzar-se al seu costat. Podem impedir-ho inserint un element buit amb la declaració `clearfix: both;`

```
<div style="clear: both;"></div>
```



Fem servir un element independent en comptes d'aplicar la declaració al següent perquè si ho féssim així li estaríem prohibint a aquest tenir els seus propis elements flotats a un i altre costat. I això no és el que volem.

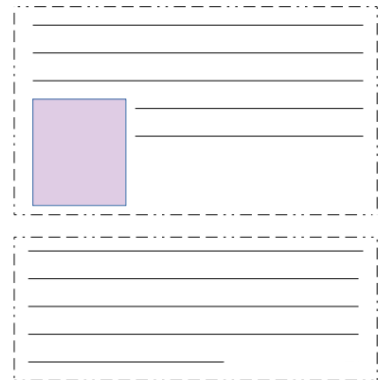
Aquesta solució però implica inserir un element amb finalitats únicament estètiques al *markup*, el que és una **mala pràctica** perquè això és responsabilitat del full d'estil.

Clearfix

Es coneix pel nom de *Clearfix* un petit truc que ens permet fer això directament des del full d'estil.

La tècnica consisteix en afegir un *pseudo-element* al final (*::after*) dels blocs que consideram "seccions" de les que no volem que els elements flotants sobresurtin i aplicar la propietat *clear* sobre ell.

```
.clearfix::after {
  display: block;
  content: " ";
  clear: both;
}
```



"float: left;" + Clearfix



Recordem que per a que un *pseudo-element* s'arribi a crear és imprescindible establir la propietat *content*. A més, necessitam que sigui un element de bloc (*display ≠ inline*). Els `<div>` per defecte ho son. Però en el cas del pseudo-element necessitam especificar-ho explícitament.

En realitat el fet que, per defecte, els elements flotats sobresurtin del contenidor és conseqüència del valor per defecte (*visible*) de la propietat *overflow*).



Abans de l'aparició de la propietat *clear* es feia servir `overflow:auto`. El problema d'aquesta tècnica era que, si no controlàvem bé els marges i els *paddings*, ens podien aparèixer barres de desplaçament. Afegir aquesta declaració al *clearfix* ens pot ajudar a suportar navegadors més antics. Però si no estam segurs de controlar bé els marges cal valorar si no és millor deixar que no funcioni en aquests (avui en dia rars) casos.

Maquetació amb floats

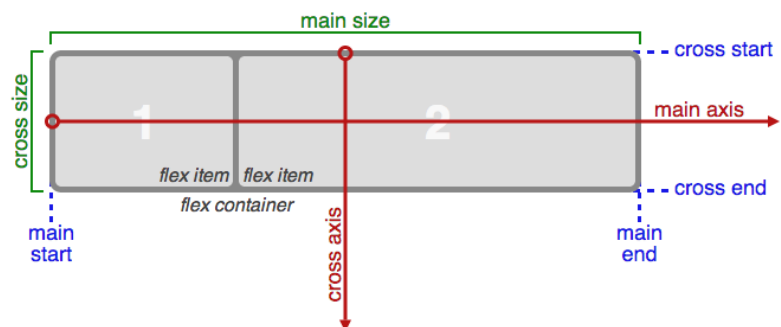
Flexbox

Flexbox és un relativament nou mòdul de CSS que s'activa mitjançant la declaració `display: flex;`

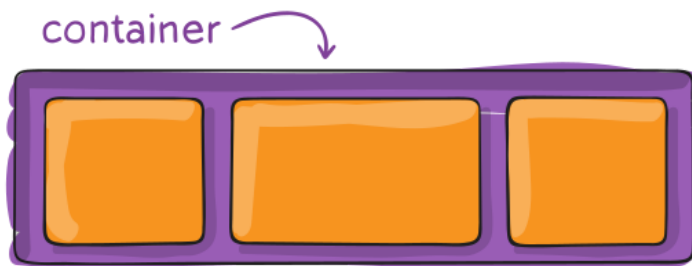
```
.contenedor {  
  display: flex; /* o inline-flex */  
}
```

Flexbox vé a complementar els clàssics `display: block` i `display: inline`. Especialment pels casos en que no podem conèixer prèviament les dimensions del contenidor i, en conseqüència, no podem dividir l'espai de forma adequada fent servir només mesures estàtiques.

Flexbox defineix dos eixos direccionals: El *Main Axis* o eix principal i el *Cross Axis*, perpendicular a aquest.



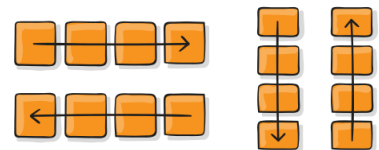
Propietats del contenidor



flex-direction

Per defecte, el *Main Axis* és l'eix horitzontal recorregut d'esquerra a dreta i el *Cross Axis* el vertical recorregut de dalt a baix. Però això es pot canviar mitjançant la propietat *flex-direction* que pot agafar les següents valors:

- `row` (per defecte): Horitzontal, d'esquerra a dreta.
- `row-reverse`: Horitzontal, de dreta a esquerra.
- `column`: Vertical, de dalt a baix.
- `column-reverse`: Vertical, de baix a dalt.

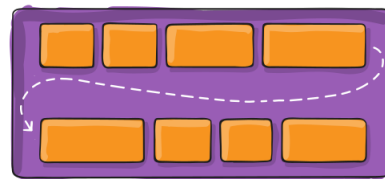


Exemple:

```
.contenedor {  
  display: flex;  
  flex-direction: column;  
}
```

flex-wrap

La propietat *flex-wrap* ens permet especificar què fer quan l'espai demandat (en la direcció del *Main-Axis*) pels *Flex-Items* supera el *main-size* (amplada o altura del contenidor, segons *flex-direction* sigui *row* / *row-reverse* o *column* / *column-reverse*, respectivament).



Valors:

- `nowrap` (per defecte): Tots els elements s'ajusten per encabir-se en una única línia.
- `wrap`: Els elements agafen l'amplada que necessiten. En acabar-se l'espai bolen a una nova línia davall de l'anterior.
- `wrap-reverse`: Igual que `wrap`, però les línies es creen de baix cap a dalt.

flex-flow

La propietat *flex-flow* és el *Shorthand* de *flex-direction* i *flex-wrap*.

Exemple:

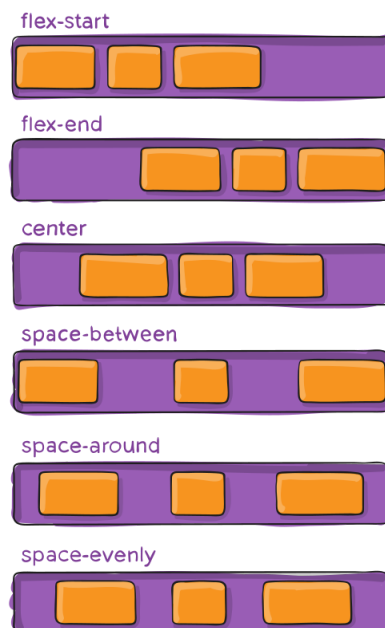
```
flex-flow: row wrap;
```

justify-content

Defineix l'aliniament al llarg del *Main Axis* permetent distribuir l'espai sobrant

Pot agafar els següents valors:

- `flex-start` (per defecte): Els elements s'acumulen al principi de la línia.
- `flex-end`: Els elements s'acumulen al final de la línia.
- `center`: Els elements s'acumulen al centre de la línia.
- `space-between`: Els elements es distribueixen proporcionalment al llarg de la línia deixant, si cal, espais d'igual tamany entre ells.
- `space-around`: Els elements es distribueixen al llarg de la línia deixant *marges* iguals entre ells. Els *marges* no col·lapsen, de manera que els espais intermigs seran dobles als dels laterals.
- `space-evenly`: Semblant a `space-around`, però amb tots els espais iguals.



Exemple:

```
justify-content: space-around;
```

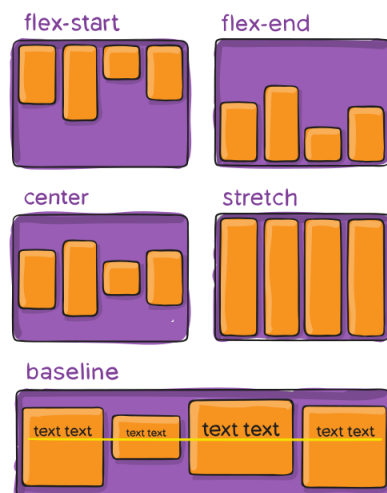
align-items

Defineix com es distribueixen els elements al llarg del *Cross-Axis*.

És la propietat complementaria a `justify-content` ja que fa tècnicament el mateix però en direcció perpendicular.

Pot agafar els següents valors:

- `flex-start`: Els elements s'alinien al principi del *Cross-Axis*.
- `flex-end`: Els elements s'alinien al final del *Cross-Axis*.
- `center`: Els elements es centren respecte del *Cross-Axis*.
- `baseline`: Els elements s'alinien respecte del seu *baseline*.
- `stretch` (per defecte): Els elements s'estiren fins a ocupar tot el *Cross-Axis*.



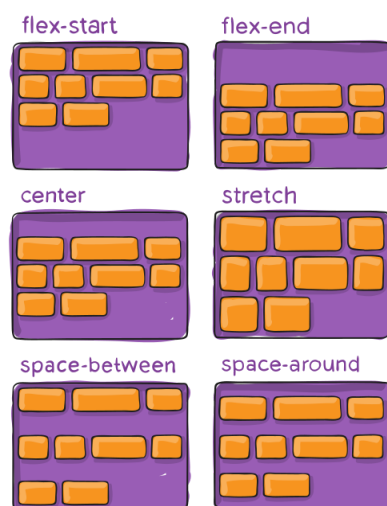
align-content

La propietat `align-items` ens permet controlar l'aliniament dels elements dins d'una única línia.

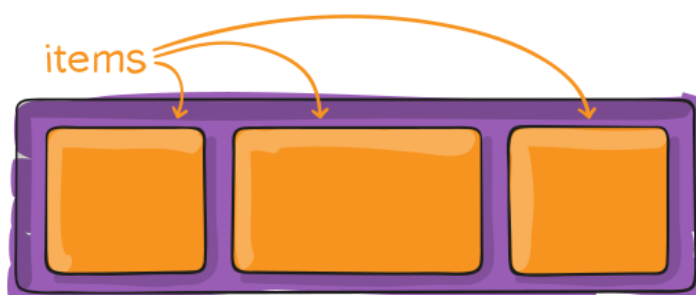
I, per defecte Flexbox només en té una (`flex-wrap: nowrap`). Però amb `flex-wrap: wrap` o `flex-wrap: wrap-reverse` podem tenir més d'una línia.

align-content ens permet controlar l'aliniament entre elles al *Cross-Axis* de forma semblant a com *justify-content* ens permet controlar l'espaiat entre els elements al *Main Axis*.

- `flex-start`: Les línies s'acumulen al principi del contenidor.
- `flex-end`: Les línies s'acumulen al final del contenidor.
- `center`: Les línies es centren al mig del contenidor.
- `space-between`: Les línies es distribueixen proporcionalment al llarg del *Cross-Axis* deixant, si cal, espais d'igual tamany entre ells.
- `space-around`: Les línies es distribueixen al llarg del *Cross-Axis* deixant *marges* iguals entre ells. Els *marges* no col·lapsen, de manera que els espais intermigs seran dobles als dels laterals.
- `stretch` (default): Les línies s'eixamplen proporcionalment fins a ocupar tot l'espai disponible.



Propietats dels fills

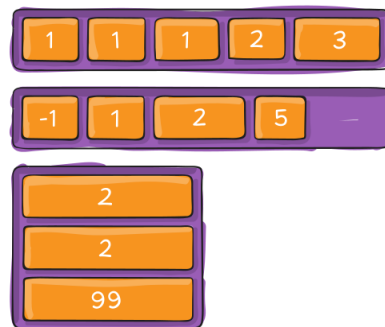


order

Amb *Flexbox* per defecte els elements es renderitzen segons l'ordre en que aparèixen al codi font seguint la direcció apuntada per la propietat *flex-direction*.

La propietat *order* ens permet alterar aquest comportament assignant un valor numèric a cada element. Els elements amb un *order* menor apareixeran primer i, en cas d'empat, continuarà prevalent l'ordre d'aparició.

El valor per defecte d'aquesta propietat és '0'. De manera que si només volem desplaçar elements concrets cap a un o altre costat, bastarà amb fer servir un nombre negatiu per enviar-los al principi del *Main-Axis* o positiu per enviar-los al final.

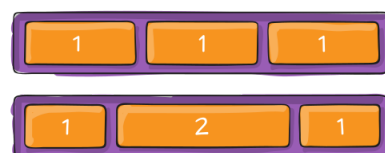


flex-grow

Controla la capacitat de creixement (a costa de l'espai sobrant) dels elements.

Accepta valors numèrics majors o iguals a '0'.

Si tots els elements tenen el mateix valor de *flex-grow*, aleshores es repartiran l'espai sobrant equitativament. Si un d'ells té el valor '2' mentre que la resta tenen '1', aleshores el primer agafarà el doble d'espai que la resta.



No accepta nombres negatius.

flex-shrink

Funciona igual que *flex-grow* però per al cas contrari: Quan l'espai no és suficient per satisfer les necessitats de tots els elements i és necessari restringir el seu tamany, *flex-shrink* ens ajuda a determinar de quins elements preferim que s'ens tregui més o menys espai.

flex-basis

Permet definir un tamany *base* per a l'element (abans de que la resta d'espai sobrant -o mancant- sigui redistribuït aplicant *flex-grow* o *flex-shrink*, segons procedeixi).

Accepta qualsevol distància css vàlida, encara que el més típic és utilitzar percentatges, o qualsevol de les següents paraules clau:

El seu valor per defecte és 'auto' que fa que s'apliquin els valors de la propietat *height* o *width*, segons correspongui.

També es pot fer servir el keyword `content`, que tindria el mateix efecte que *auto* si *width* (o *height*) també és *auto*. Si be *content* va ser introduït amb posterioritats i és més recomanable fer servir la tècnica del "doble 'auto'" per evitar problemes amb els més antics.

flex

És el shorthand de les propietats *flex-grow*, *flex-shrink* i *flex-basis* combined.

- El segon i tercer paràmetres (*flex-shrink* i *flex-basis*) son opcionals.



- Els valors per defecte son 0 1 auto.

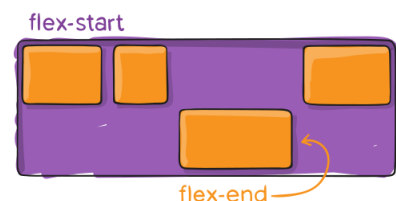


Es recomana sempre fer servir aquesta propietat en comptes de les altres individualment ja que aquesta estableix els valors omesos de forma intel·ligent.

align-self

Té el mateix efecte que la propietat *align-items* del contenidor abans explicada amb la diferència que aquesta s'aplica als elements. Permetent així que un o més elements presentin una alineació distinta a la resta.

Per defecte agafa el valor *auto* vol dir que l'element respondrà al paràmetre *align-items* del contenidor o, cas que aquest tampoc s'especifiqui, al seu comportament per defecte.



Maquetació amb Flexbox

Històricament, per poder implementar *layouts* complexos, s'han fet servir *floats* perquè era la única forma de colocar dos o més contenidors un vora l'altra sense necessitat de conèixer prèviament les coordenades (horitzontals) de cadascun.

Era un mètode difícilós perquè s'havien de tenir en comte els marges (normalment posant-los a '0') i, si no es feia servir la declaració `box-sizing: border-box;`, també les voreres i els *paddings*.

Exercici

Implementar una barra d'encapçalament elàstica com la del següent esquema:

Nom aplicació	8:45pm	[Opcions]	[Ajuda]
---------------	--------	-----------	---------

- Els botons a la dreta.
- El nom de l'aplicació a l'esquerra.
- El rellotge al centre de l'espai entremig.

Grid Layout

El *Grid Layout* va aparèixer amb posterioritat a *Flexbox*.

Podríem dir que *Flexbox* ens permet repartir l'espai en una dimensió (horitzontal o vertical) mentre que *Grid* ens permet fer-ho en dues dimensions a l'hora.

Aixímateix **Grid no reemplaça Flexbox**, sinó que son dos sistemes complementaris.

Per una banda *Flexbox* és més senzill a l'hora de repartir l'espai de forma elàstica entre múltiples elements. Encara que no en coneguem prèviament la quantitat.

Però, per altra banda, si bé és possible una maquetació en dues dimensions amb *flexbox* (combinant múltiples contenidors *Flex* en una dimensió continguts en un altre *Flexbox* en la dimensió perpendicular). Amb *Grid Layout* això resulta molt més senzill, ja que només necessitam un únic contenidor *Grid* dins del qual posarem tots els *Grid Items* un darrera l'altre i serà el CSS el que s'ocuparà de distribuir-los als seus llocs corresponents.

Així, el HTML de qualsevol *Grid* serà molt semblant al següent:

```
<div class="grid">
  <div>...</div>
  <div>...</div>
  <div>...</div>
  ...
</div>
```

Conceptes bàsics

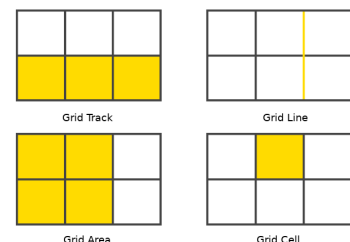
Grid és, si cab, conceptualment més senzill que *Flexbox*. Però a l'hora té moltíssimes més propietats que, per una banda ens permeten controlar gairebé fins al més mínim detall i, per l'altra, utilitzar distintes tècniques per aconseguir els mateixos resultats.

Això darrer pot resultar avantatjós a l'hora d'abordar dissenys complexos ja que en uns casos ens podrà convenir una estratègia o altra segons les seves característiques.

...però per altra banda al principi ens pot embullar si no sabem per on començar.

Per això el més important és entendre els conceptes bàsics en que es fonamenta el *Grid* i dominar al menys alguna de les tècniques que ens permeten controlar-lo.

- **grid-item:** És cadascun dels elements que son fills directes del contenidor i que s'aniran col·locant a la graella segons les regles que nosaltres especifiquem.
- **grid-line:** Son les línies horitzontals i verticals, tant interiors com exteriors, que conformen l'estructura de la graella.
- **grid-track:** És l'espai entre dues *grid-lines* adjacents.
- **grid-cell:** És l'espai que conformen dues *grid-lines* horitzontals i dues verticals respectivament adjacents. Constitueix la unitat mínima en que es pot dividir el *Grid*.
- **grid-area:** Igual que una *grid-cell* però sense la condició d'adjacència de les *grid-lines*.



Propietats



Seguidament s'expliquen algunes de les propietats més importants que podem fer servir en un **CSS-Grid**.

L'especificació és molt àmplia i moltes vegades els mateixos resultats es poden assolir de diverses formes (i mitjançant múltiples propietats) distintes.

Per aquest motiu aquí ens limitarem a intentar donar una visió global de les tècniques i propietats més comunes de manera que ens permetin resoldre la majoria de situacions que s'ens puguin presentar.

Per a més informació, a l'apartat de referències s'enllaça un article molt complet del portal web *CSS Tricks* al respecte.



A l'apartat anterior sobre *Flexbox* hem separat les propietats que s'apliquen al contenidor de les que s'apliquen als *ítems*.

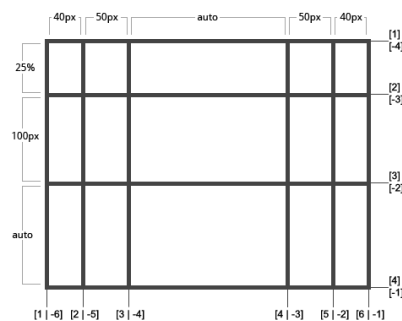
En canvi en aquest apartat, per tal de posar-les més en context, el que farem és marcar els apartats amb el text "(grid-items)" quan es tracti de propietats dels *ítems*.

grid-template-columns i grid-template-rows

La primera propietat (després del `display: grid;`) que necessitarem establir a l'hora de muntar un *Grid* és *grid-template-columns*, ja que en cas contrari obtindríem una graella d'una sola columna.

grid-template-rows, en canvi, no és tan imprescindible ja que, encara que no estigui definida, o no hagem definit files suficients per encabir tots els elements, aquestes es crearan automàticament a mesura que siguin necessàries. Si be, fer servir *grid-template-rows* ens permetrà controlar la seva altura.

```
.container {  
  display: grid;  
  grid-template-columns:  
    40px 50px auto 50px 40px;  
  grid-template-rows: 25% 100px auto;  
}
```



Amb això ja tindríem un *Grid* completament funcional. L'únic que hem de fer és emplenar-lo de *grid-items* que s'aniran distribuint automàticament d'esquerra a dreta i de dalt a baix per les cel·les disponibles.

Exemple:

```
<div class="grid">  
  <div>Item 1</div>  
  <div>Item 2</div>  
  <div>Item 3</div>  
  <div>Item 4</div>  
  <div>Item 5</div>  
  <div>...</div>  
</div>
```

grid-column i grid-row (grid-items)

Si, per altra banda, desitjam especificar la fila i columna a la que ha d'anar ubicat un element, podem fer servir les propietats *grid-column* i *grid-row* per especificar **després de quin *grid-line* volem que s'ubiqui**

Exemple:

```
.very-first-item {  
  grid-column: 1;  
  grid-row: 1;  
}
```

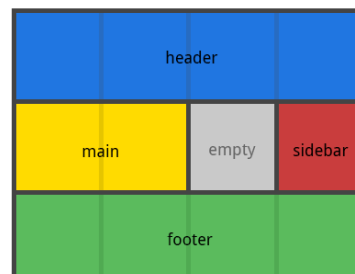


De fet, en realitat, *grid-column* i *grid-row* accepten no un, sinó 2 nombres de *grid-line*, separats per una barra inclinada (/).

Això ens permet construir layouts com el de la figura, on alguns dels *items* ocupen més d'una cel·la.

El CSS seria el següent:

```
.container {  
  grid-template-columns: 50px 50px 50px 50px;  
  grid-template-rows: auto;  
}  
.header {  
  grid-column: 1 / 5;  
  grid-row: 1 / 2; /* O simplement '1' */  
}  
.main {  
  grid-column: 1 / 3;  
  grid-row: 2 / 3;  
}  
.sidebar {  
  grid-column: 4 / 5;  
  grid-row: 2 / 3;  
}  
.footer {  
  grid-column: 1 / 5;  
  grid-row: 3 / 4;  
}
```



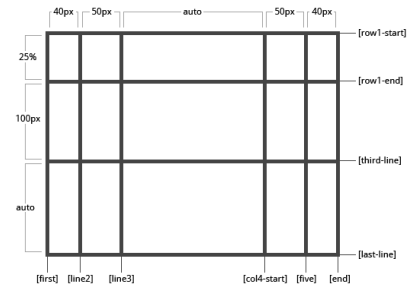
Etiquetes: Una característica interessant de *grid-template-columns* i *grid-template-rows* és que ens permeten inserir etiquetes entre cadascun dels *grid-track* (columnes o files) que estem definint.

Les etiquetes es defineixen entre claudàtors ([]). A més, en podem definir més d'una a cada emplaçament. Per exemple per delimitar el final d'una secció i el principi d'una altra. Així, si algun dia volguéssim canviar-les de lloc o inserir una altra secció enmig, només caldria moure les etiquetes pertinents sense necessitat de redefinir-les.

```

.container {
  display: grid;
  grid-template-columns:
    40px 50px auto 50px 40px;
  grid-template-rows:
    [row1-start] 25%
    [row1-end row2-start] 25%
    [row2-end]
;
}

```

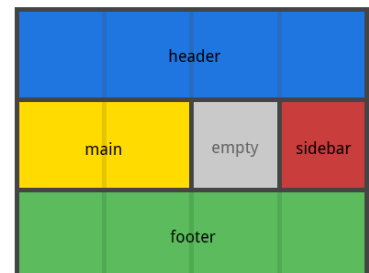


grid-template-areas

Les etiquetes ens simplifiquen enormement la feina de crear (i sobretot mantenir) *layouts* complexes. Especialment quan no coneixem prèviament el nombre d'elements (*grid-items*) que contindrà la nostra graella o, sobretot, si aquest pot variar.

Però quan es tracta d'un *layout* estàtic, la propietat *grid-template-areas* ens ofereix una forma molt millor...

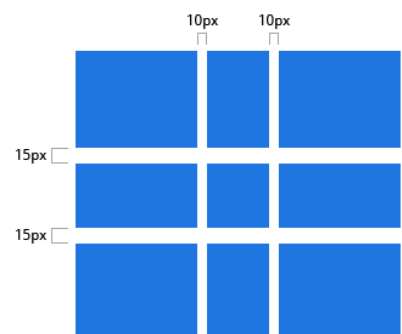
Per exemple, per implementar el mateix *layout* d'abans, el CSS seria el següent:



```

.container {
  grid-template-columns: repeat(4, 50px);
  grid-template-rows: auto;
  grid-template-areas:
    "header header header header"
    "main main . sidebar"
    "footer footer footer footer";
}
.header { grid-area: header; }
.main { grid-area: main; }
.sidebar { grid-area: sidebar; }
.footer { grid-area: footer; }
[...]
```

grid-gap



Disseny Responsiu

Disseny Adaptatiu (Elasticitat)

Media Queries

Mobile-First Approach

Referències:

- Flexbox:
 - <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>.
- Grid:
 - <https://css-tricks.com/snippets/css/complete-guide-grid/>.

Javascript - Programació Asíncrona

Callbacks

Promeses

Async/Await