



- Back-End(<https://imasters.com.br/back-end>)
- Mobile(<https://imasters.com.br/mobile>)
- Front End(<https://imasters.com.br/front-end>)
- DevSecOps(<https://imasters.com.br/devsecops>)
- Design & UX(<https://imasters.com.br/design-ux>)
- Data(<https://imasters.com.br/data>)
- APIs e Microsserviços(<https://imasters.com.br/apis-microsservicos>)
- IoT e Makers(<https://imasters.com.br/iot-makers>)

PATROCINADORES:


CSS



12 MAI, 2017

Adeus Flexbox! Bem-vindo CSS Grid Layout!

100 visualizações

 (<https://www.facebook.com/sharer?u=https://imasters.com.br/css/adeus-flexbox-bem-vindo-css-grid-layout>) (<https://twitter.com/share?url=https://imasters.com.br/css/adeus-flexbox-bem-vindo-css-grid-layout>) (<https://www.linkedin.com/shareArticle?url=https://imasters.com.br/css/adeus-flexbox-bem-vindo-css-grid-layout>)

COMPARTILHE!

SIMONE AMORIM
(<https://imasters.com.br/perfil/simoneam>)
Tem 1 artigos publicados com 4332
visualizações desde 2017



PUBLICIDADE

SIMONE AMORIM (<https://imasters.com.br/perfil/simoneamorim>)

1

É WWcode Leader, mãe e estudante para se tornar uma desenvolvedora front-end e evangelista CSS. Apoionada por corrida e ciclismo, ama aprender coisas novas e dividi-las. Em seu tempo livre gosta de codificar, ouvir música e ver séries.

LEIA MAIS (<https://imasters.com.br/perfil/simoneamorim>)

O futuro chegou!

Finalmente ganhamos algo mais importante do que uma especificação boa; ganhamos a implementação de uma especificação boa nos principais browsers <3.

Até o momento, usamos gambiarras para a criação de layouts com CSS, porque, até então, não existia nada voltado especificamente para essa função. Legal, temos flexbox, mas relaxa aí e lê até o final...

Do antes ao agora

Nos primórdios, usávamos uma tabela, que foi uma forma de facilitar a criação de layout – o que em contra partida retornava uma série de problemas. Em seguida, iniciamos a utilização do float no lugar das tabelas, o que nos ajudou muito por ser mais flexível e usar menos código HTML. Mas o float não foi feito pra isso. Só quem usou sabe como era triste e limitado.

Pra ajudar a alinhar e distribuir o espaço entre itens em um container, surgiu o [CSS Flexbox Layout \(Flexbox\)](https://www.w3.org/TR/css-flexbox-1/) (<https://www.w3.org/TR/css-flexbox-1/>), onde a ideia principal é dar ao container a capacidade de alterar a largura / altura e ordem de seus itens para um melhor preenchimento do espaço disponível e, principalmente, para acomodar todos os tipos de dispositivos de exibição e tamanhos de tela. Mas o amado flexbox não resolve todos os problemas e também não foi criado para resolvê-los, principalmente quando o assunto são interfaces complexas.



PLATAFORMA DE
ENSINO PARA
PROGRAMADORES

ASS

Hoje, oficialmente estamos no futuro e temos suportado nos browsers uma opção realmente disruptiva no quesito “criar layouts” com CSS, lhes apresento o seu novo Deus.

O que é CSS Grid Layout?

É uma especificação da W3C (<https://drafts.csswg.org/css-grid/#intro>) que disponibiliza um novo modelo de layout para o CSS com uma poderosa habilidade para controlar o tamanho e posição dos elementos e seus conteúdos.

Sim, mas devo usar Flexbox ou Grid Layout?

Flexbox é para layouts unidimensionais – qualquer coisa que precisa ser disposta em uma linha reta.

| HTML | CSS | Result | EDIT ON |
|---|-----|--------|---------|
| <pre>:root { --margin: 3px; } body { padding: 10px; } .container-box { display: flex; }</pre> | | | |
| Resources | | | |

Já o Grid Layout é a solução certa quando você deseja controlar o dimensionamento e o alinhamento em duas dimensões.

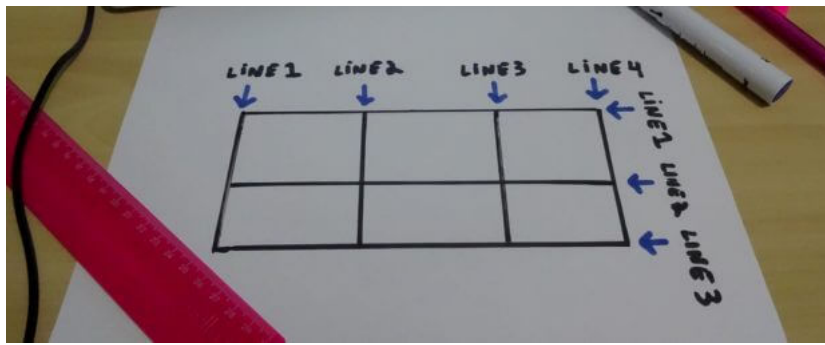
| HTML | CSS | Result | EDIT ON |
|--|-----|--------|---------|
| <pre>body { padding: 10px; } .container-box { display: grid; grid-gap: 10px; grid-template-columns: auto; grid-template-rows: 50px 50px; grid-template-areas: }</pre> | | | |
| Resources | | | |

Terminologia

Antes de iniciar o trabalho com Grid Layout, é necessário entender os termos que vem junto com a especificação. Explicarei cada um deles para facilitar o entendimento dos exemplos posteriores.

Grid Lines

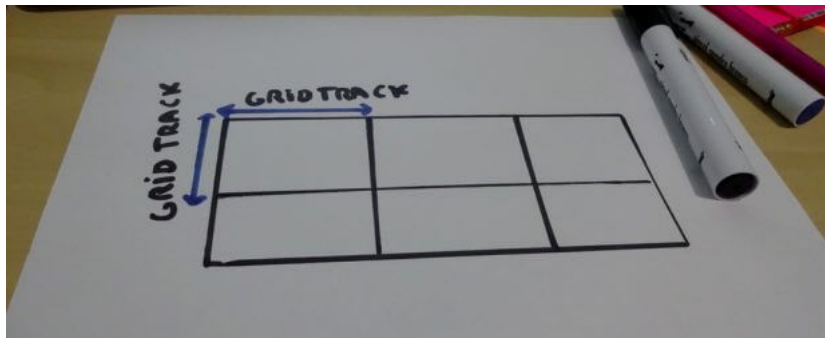
São as linhas que definem o grid, que podem ser distribuídas de forma horizontal ou vertical. Futuramente, poderemos nos referir a elas por um número ou por um nome que pode ser definido pelo desenvolvedor.



(https://imasters.com.br/?attachment_id=115353)

Grid Tracks

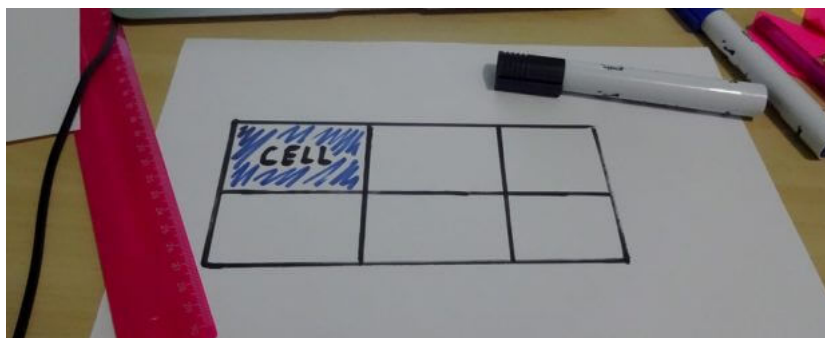
É o espaço horizontal ou vertical entre duas Grid Lines.



(https://imasters.com.br/?attachment_id=115354)

Grid Cell ou Grid Item

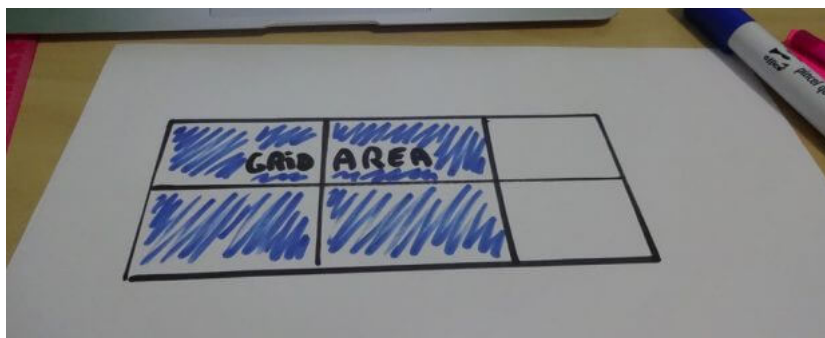
É o espaço entre quatro Grid Lines, sendo a menor unidade em nosso grid. Conceitualmente, podemos fazer uma analogia com uma célula de tabela.



(https://imasters.com.br/?attachment_id=115355)

Grid Areas

É qualquer espaço no Grid usado para exibir um ou mais Grid Cells/ Items.



(https://imasters.com.br/?attachment_id=115356)

Grid Containers

É o pai direto de todos os itens do grid, sendo o elemento que recebe a propriedade display: grid.

No exemplo abaixo, .container-box é o Grid Container.

```
1 <div class="container-box">
2   <div class="box"></div>
3   <div class="box"></div>
4   <div class="box"></div>
5 </div>
```

Grid items

São os itens que representam o conteúdo do grid; cada filho direto do grid container torna-se um grid item.

No exemplo abaixo, cada elemento .box é um grid item, porém o .sub-box não.

```
1 <div class="container-box">
2   <div class="box"></div>
3   <div class="box">
4     <div class="sub-box"></div>
5   </div>
6   <div class="box"></div>
7 </div>
```

Definindo um Grid

Para definir um grid use os novos valores da propriedade display: grid ou display: inline-grid, assim que for definido uma dessas propriedades no .container-box, todos os filhos diretos desse elemento se tornarão itens do grid automaticamente.

O display: grid gera um Grid container com comportamento de bloco.

```
1 .container-box {
2   display: grid;
3 }
```

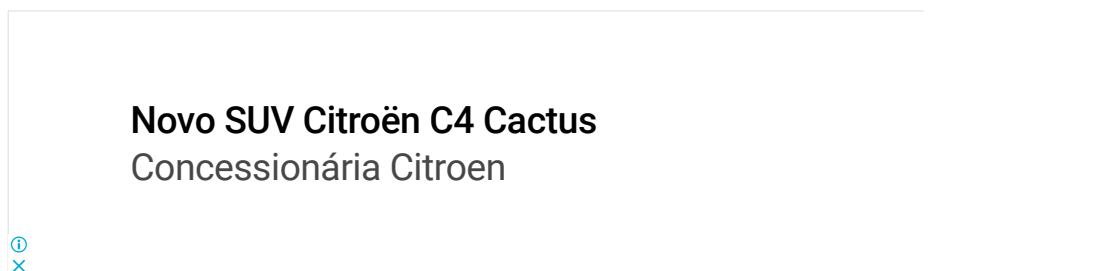
display: inline-grid gera um Grid container com comportamento inline.

```
1 .container-box {
2   display: inline-grid;
3 }
```

Line-base placement

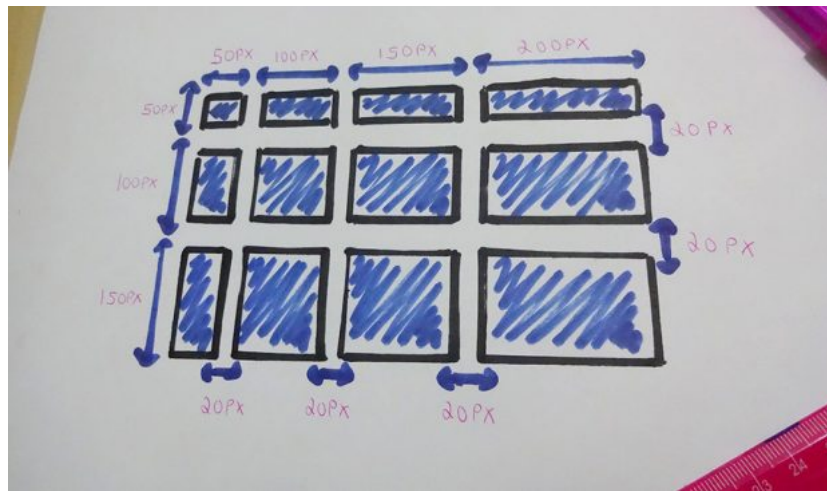
Para definir as colunas e linhas do grid, use as propriedades grid-template-columns e grid-template-rows.

Cada valor atribuído a propriedade grid-template-columns representa uma coluna que assumirá o tamanho do valor que lhe for atribuído, assim como a propriedade grid-template-rows assumirá o espessura do valor que lhe for atribuído.



Use as propriedades grid-column-gap e grid-row-gap para especificar o espaçamento entre as columns e as rows do grid.

```
1 .container-box {
2   display: grid;
3   grid-template-columns: 50px 100px 150px 200px;
4   grid-template-rows: 50px 100px 150px;
5   grid-column-gap: 20px;
6   grid-row-gap: 20px;
7 }
```



(https://imasters.com.br/?attachment_id=115357)

Podemos usar o shorthand `grid-gap` para passar os dois valores de uma só vez.

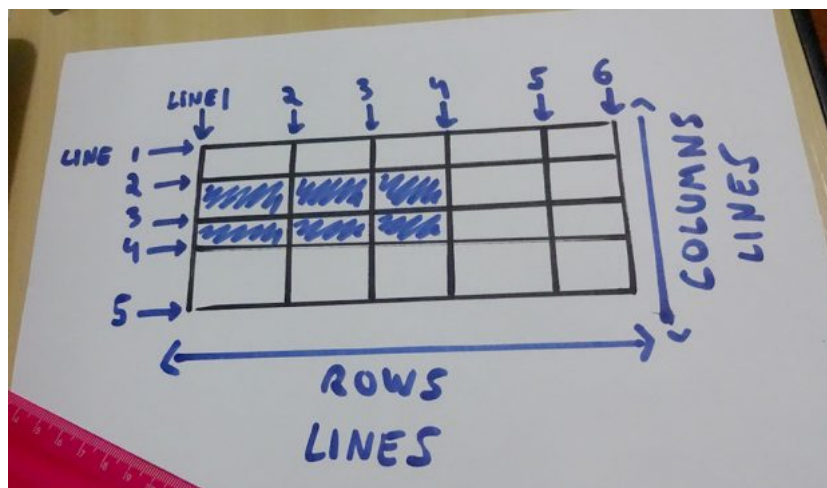
```
1 | ...
2 |   grid-gap: 20px;
3 | ...
```

Line-based position

Define onde está localizado um grid item no grid container, fazendo referência a uma grid line específica.

Use `grid-column-start` e `grid-row-start` para especificar a grid line, onde o item começa e `grid-column-end` e `grid-row-end` para especificar a grid line onde o item termina.

```
1 | .box {
2 |   grid-column-start: 1;
3 |   grid-row-start: 2;
4 |   grid-column-end: 4;
5 |   grid-row-end: 4;
6 | }
```



(https://imasters.com.br/?attachment_id=115358)

Veja um exemplo de `line-base-position` (<http://codepen.io/simoneas02/pen/VpWwvz?editors=1100>).

Podemos usar um shorthand para facilitar:

```
1 | .box {
2 |   grid-column: 1/4; /* grid-column-start / grid-column-end */
3 |   grid-row: 2/4; /* grid-row-start / grid-row-end */
4 | }
```

Ou ainda menor com o `grid-area`:

```
1 | .box {  
2 |     grid-area: 1 / 2 / 4 / 4; /* grid-column-start / grid-row-start / grid-column-end / grid-row-end */  
3 | }
```

Area naming

No grid container podemos manipular a posição e comportamento dos grid itens. Isso através da propriedade `grid-template-areas`, onde podemos literalmente montar todo o nosso layout em apenas uma propriedade. Para facilitar e atribuir mais semântica ao que estamos fazendo, atribuímos um nome aos nossos grid itens através da já conhecida `grid-area`.



É isso mesmo que você viu. Relaxe e tente viver com essa realidade.

Já roda em todos browsers?

Aqui entra a melhor parte e o motivo de eu estar compartilhando esse conteúdo com vocês: essa tecnologia já está por aí há bastante tempo, mas recentemente o Firefox e o Chrome anunciaram o suporte para essa maravilha.

Veja por você mesmo no [Can I Use](http://caniuse.com/#feat=css-grid) (<http://caniuse.com/#feat=css-grid>).

Quer saber mais?

- [Collection com exemplos de Grid Layout](http://codepen.io/collection/njkQa/#) (<http://codepen.io/collection/njkQa/#>)
- [Grid by Example](http://gridbyexample.com/) (<http://gridbyexample.com/>)
- [A Complete Guide to Grid](https://css-tricks.com/snippets/css/complete-guide-grid/) (<https://css-tricks.com/snippets/css/complete-guide-grid/>)s

Conclusão

Esse artigo foi apenas uma pincelada em algumas das features que eu lembrei de cabeça aqui na hora, mas acreditem: a especificação tem MUITO mais a oferecer.

Comente!

Se quiserem saber mais sobre o assunto, podem me seguir por aí. Se tiverem alguma dúvida, mandem comentários ou gritem nas redes sociais:

- [@samorim02](https://twitter.com/samorim02) (Twitter) (<https://twitter.com/samorim02>)
- [@simoneas02](https://github.com/simoneas02) (GitHub) (<https://github.com/simoneas02>)
- [Personal Page](https://simoneas02.github.io/) (<https://simoneas02.github.io/>)



De 0 a 10, o quanto você recomendaria este artigo para um amigo?

