CSS Grid

1. Introdução ao CSS Grid

Para layouts unidimensionais, o Flex oferece uma ferramenta conveniente sem depender de floats ou hacks CSS para alinhar seus itens corretamente.

Para layouts bidimensionais, você aprendeu um pouco sobre flex-wrap, que permite pegar seus itens flexíveis e envolvê-los para a próxima linha. Isso pode ser feito com uma linha que se transforma em outra linha ou uma coluna que se transforma em outra coluna.

Sabemos que foi frustrante, mas faz parte do ponto. Embora o Flexbox permita que você crie um layout de linhas e colunas juntas, nem sempre é fácil.

Mas configurar um layout bidimensional de cartões seria muito mais simples usando CSS Grid:

index.html

```
  <div class="grid-container">
    <div class="card"></div>
    <div class="card"></div>
    <div class="card"></div>
    <div class="card"></div>
  </div>
```

styles.css

```
.grid-container {
  display: grid;
  grid-template-columns: 75px 75px;
  grid-template-rows: 75px 75px;
  gap: 32px;
}

.card {
  background-color: peachpuff;
```

```
  border: 1px solid black;
}
```

Grid é frequentemente elogiada pela fácil colocação de itens em um layout bidimensional. Mas Grid também pode ser usado para layouts unidimensionais. Uma vantagem para os desenvolvedores é que se eles começarem com apenas uma linha de itens, eles podem simplesmente adicionar mais linhas posteriormente.

Você notará muitas semelhanças entre Flex e Grid. Ambos usam contêineres pai com itens filhos. Eles têm nomes de propriedades semelhantes para alinhamento e posicionamento. Mas você também encontrará muitas diferenças entre os dois e opiniões sobre como cada módulo deve ser usado. Por exemplo, se você está lutando para fazer com que seus itens Flex correspondam uniformemente em tamanho, o Grid pode tornar esse tipo de layout muito mais fácil.

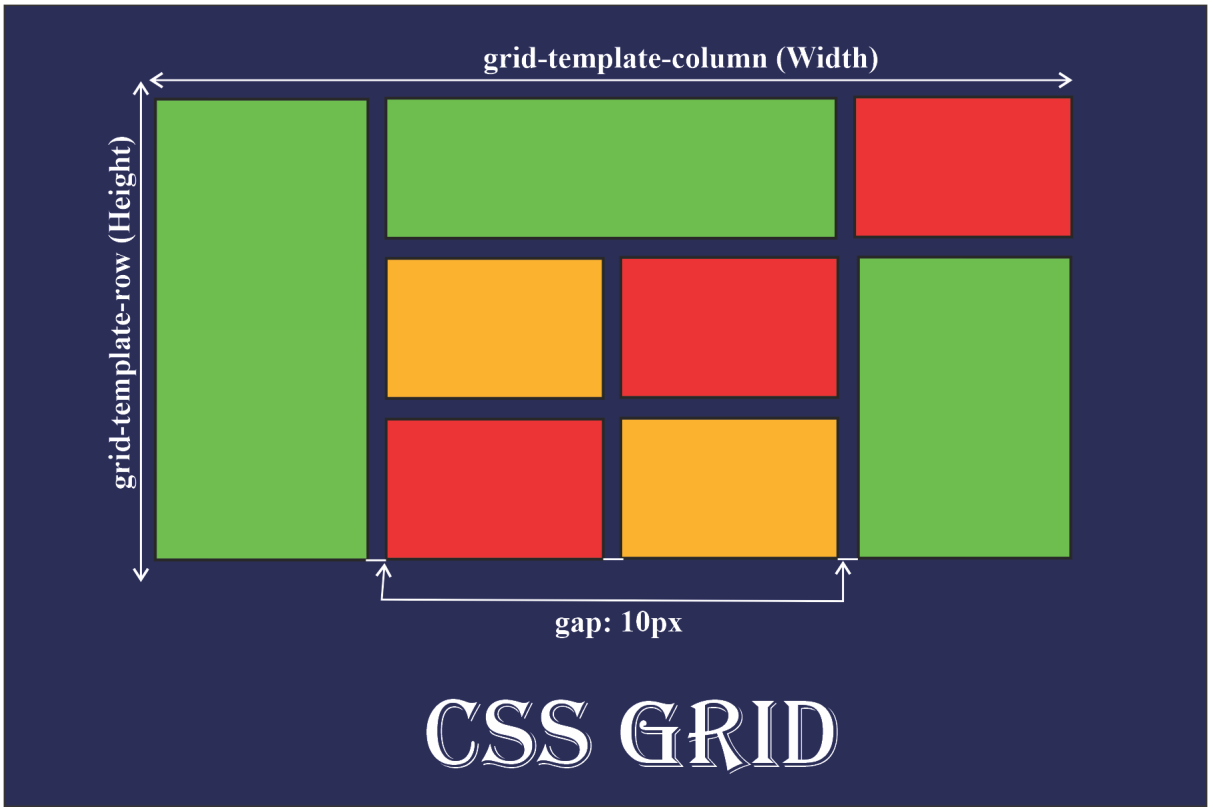
Ao revisar recursos mais antigos, lembre-se de que as diferenças entre Flex e Grid também podem mudar à medida que esses módulos são atualizados. Um dos grandes destaques do CSS Grid foi o uso da propriedade gap, que abordaremos na próxima lição. Isso costumava estar disponível apenas para o Grid, mas agora também é suportado pelo Flex.

Enquanto algumas pessoas pensavam que o CSS Grid estava aqui para substituir o Flexbox, você aprenderá no final dessas lições que o Grid é apenas mais uma ferramenta na caixa. Na verdade, não apenas cada um desses módulos tem seus próprios casos de uso, mas você também achará útil emparelhar Flex e Grid. Mas abordaremos tudo isso na lição final. Primeiro você aprenderá como realmente fazer uma grade!

1.1. Container do CSS Grid

Podemos pensar em CSS Grid em termos de um container e itens. Simplificando, quando você transforma um elemento em um contêiner de grade, ele “contém” toda a grade. Em CSS, um elemento é transformado em um contêiner de grade com a propriedade `display: grid` ou `display: inline-grid`.

1.2. Linhas e Colunas do CSS Grid



As propriedades `grid-template-columns` e `grid-template-rows` facilitam a definição de trilhas de coluna e linha. Para esta lição, vamos nos ater a definir nossas colunas e linhas usando pixels. Nas próximas lições, você aprenderá mais sobre como definir porcentagens e unidades fracionárias também.

Se quisermos adicionar mais colunas ou linhas à nossa grade, podemos simplesmente definir esses valores para fazer outra trilha. Digamos que queremos adicionar uma terceira coluna ao nosso exemplo.

styles.css

```
.grid-container {
  display: grid;
  grid-template-columns: 75px 75px 75px;
  grid-template-rows: 75px 75px;
  gap: 32px;
}
```

```
.card {
  background-color: peachpuff;
  border: 1px solid black;
}
```

CSS Grid também inclui uma propriedade abreviada para definir linhas e colunas. Em nosso exemplo anterior, podemos substituir as propriedades de `grid-template-rows` e `grid-template-columns` pela propriedade abreviada `grid-template`. Aqui podemos definir nossas linhas e colunas de uma só vez. Para esta propriedade, as linhas são definidas antes da barra e as colunas são definidas após a barra. Vamos manter os mesmos valores de coluna e linha, mas use a propriedade abreviada.

styles.css

```
.grid-container {
  display: grid;
  grid-template: 75px 75px / 75px 75px;
  gap: 32px;
}
```

Colunas e linhas também não precisam compartilhar os mesmos valores. Vamos alterar os valores das propriedades de nossas colunas para que a primeira coluna tenha cinco vezes a largura das outras.

styles.css

```
.grid-container {
  display: grid;
  grid-template: 50px 50px / 250px 50px 50px;
  gap: 32px;
}
```

1.3. Grid Implícito e Explícito

Vamos voltar ao nosso exemplo original de um layout 2x2 simples para quatro itens de grade. O que acontece se adicionarmos um quinto item ao nosso contêiner sem alterar nossas propriedades grid-template-columns ou grid-template-rows?

index.html

```
  <div class="grid-container">
    <div class="card">Item 1</div>
    <div class="card">Item 2</div>
    <div class="card">Item 3</div>
    <div class="card">Item 4</div>
    <div class="card">Item 5</div>
  </div>
```

styles.css

```
.grid-container {
  display: grid;
  grid-template: 50px 50px / 50px 50px;
  gap: 32px;
}
```

Você notará que nosso quinto item foi colocado na grade e foi encaixado em uma terceira linha que não definimos. Isso ocorre por causa do conceito de grade implícito e é como o CSS Grid é capaz de colocar itens de grade automaticamente quando não definimos explicitamente o layout para eles.

Quando usamos as propriedades `grid-template-columns` e `grid-template-rows`, estamos definindo explicitamente as faixas de grade para dispor nossos itens de grade. Mas quando a grade precisar de mais trilhas para conteúdo extra, ela definirá implicitamente novas faixas de grade. Além disso, os valores de tamanho estabelecidos de nossas propriedades `grid-template-columns` ou `grid-template-rows` não são transportados para essas trilhas de grade implícitas. Mas podemos definir valores para as trilhas de grade implícitas.

Podemos definir os tamanhos implícitos das faixas de grade usando as propriedades `grid-auto-rows` e `grid-auto-columns`. Dessa forma, podemos garantir que quaisquer novas faixas que a grade implícita faça para conteúdo extra sejam definidas nos valores que definimos.

styles.css

```
.grid-container {
  display: grid;
  grid-template-columns: 50px 50px;
  grid-template-rows: 50px 50px;
  grid-auto-rows: 50px;
  gap: 32px;
}
```

Por padrão, CSS Grid adicionará conteúdo adicional com linhas implícitas. Isso significa que os elementos extras continuariam sendo adicionados mais abaixo na grade de maneira vertical. Seria muito menos comum querer conteúdo extra adicionado horizontalmente ao longo da grade, mas isso pode ser definido usando a propriedade `grid-auto-flow: column` e esses tamanhos de trilha implícitos podem ser definidos com a propriedade `grid-auto-columns`.

1.4. Gap

Os tamanhos de intervalo das lacunas (gap) podem ser ajustados separadamente para linhas e colunas usando as propriedades `column-gap` e `row-gap`. Além disso, podemos usar uma propriedade abreviada chamada `gap` para definir o `row-gap` e o `column-gap`.

styles.css

```
.grid-container {
  display: grid;
  grid-template-columns: 50px 50px;
  grid-template-rows: 50px 50px;
  column-gap: 10px;
  row-gap: 100px;
}
```

2. Posicionamento dos Elementos da Grid

Nesta tópico, examinaremos as diferentes partes de uma grade e exploraremos propriedades comuns que podem ser usadas para posicionar itens da grid.

2.1. Revisando trilhas

Antes de mergulharmos direto no posicionamento, vamos estabelecer algumas terminologias para entender melhor as diferentes partes de uma grade. Na lição anterior você aprendeu que quando você define uma grade usando `grid-template`, você está definindo as trilhas que a grade terá. Você pode pensar em uma faixa de grade como qualquer linha ou coluna em uma grade.

Para dar um exemplo, se quisermos criar uma grade 3x3 com linhas de 100 pixels e colunas de 100 pixels, precisamos definir 3 trilhas horizontais com altura de 100 pixels e 3 trilhas verticais com largura de 100 pixels:

index.html

```
<div class="container">
  <div class="item first-row">A</div>
  <div class="item first-row">B</div>
  <div class="item first-row last-column">C</div>
  <div class="item">D</div>
  <div class="item">E</div>
  <div class="item last-column">F</div>
  <div class="item">G</div>
  <div class="item">H</div>
  <div class="item last-column">I</div>
</div>
```

styles.css

```
.container {
  display: grid;
  grid-template-rows: 100px 100px 100px;
  grid-template-columns: 100px 100px 100px;
```

```
}

.item {
  background-color: orange;
  border: 1px solid black;
  text-align: center;
}

.first-row {
/*    background-color: pink; */
}

.last-column {
/*    background-color: lightblue; */
}
```

Remova o comentário da propriedade no seletor de classe `.first-row` para ver a trilha de grade entre as linhas de grade da primeira e da segunda linha.

Em seguida, remova o comentário da propriedade no seletor de classe `.last-column` para ver a trilha de grade entre as linhas de grade da terceira e da quarta coluna.

2.2. Linhas

Sempre que criamos trilhas de grade, as linhas de grade são criadas implicitamente. Isso é importante. As linhas de grade são criadas somente depois que nossas trilhas de grade foram definidas. Não podemos criar linhas de grade explicitamente.

Cada trilha tem uma linha de partida e uma linha de chegada. As linhas são numeradas da esquerda para a direita e de cima para baixo começando em 1. Portanto, nossa grade 3x3 acima tem linhas verticais para colunas que variam de 1 a 4 e linhas horizontais para linhas que variam de 1 a 4.

Linhas de grade são o que usamos para posicionar itens de grade.

2.3. Células

As células são a menor unidade de medida em uma grade. Você pode pensar em células de grade como uma célula em uma planilha ou um quadrado em um gráfico. Por padrão, cada elemento filho de um contêiner de grade ocupará uma célula. No exemplo acima, temos 9 células em nossa grade. Portanto, o elemento que marcamos com a letra "A" é uma célula entre as linhas de grade de linha 1 e 2 e as linhas de grade de coluna 1 e 2. O elemento com a letra "H" é uma célula entre as linhas de grade de linha 3 e 4 e a grade de coluna linhas 2 e 3.

Mas o que acontece se quisermos mudar a ordem dos nossos itens de grade? Ou queremos que os itens ocupem mais de uma célula?

2.4. Posicionamento

Para ter uma ideia de como os itens podem ser posicionados, criaremos uma planta simulada de um apartamento. Vamos começar com uma área total do nosso apartamento (o contêiner da grade) dividida em uma grade de 5x5. Para tornar este exemplo um pouco mais claro, usaremos uma cor de fundo para distinguir nosso espaço de contêiner. Observe que também estamos usando `display: inline-grid` aqui para que nosso contêiner não se estique para ocupar espaço da mesma forma que uma caixa em nível de bloco. Isso apenas nos ajudará a visualizar melhor o espaço.

Para torná-lo menos uma caixa vazia e mais uma casa, vamos adicionar alguns itens ao nosso contêiner de grade que representarão salas diferentes.

index.html

```
<div class="container">
  <div class="room" id="living-room">Living Room</div>
  <div class="room" id="kitchen">Kitchen</div>
  <div class="room" id="bedroom">Bedroom</div>
  <div class="room" id="bathroom">Bathroom</div>
  <div class="room" id="closet">Closet</div>
</div>
```

styles.css

```
.container {
  display: inline-grid;
  grid-template: 40px 40px 40px 40px 40px / 40px 40px 40px
40px 40px;
  background-color: lightblue;
}

.room {
  border: 1px solid;
  font-size: 50%;
  text-align: center;
}

#living-room {
  grid-column-start: 1;
  grid-column-end: 6;
/*   grid-row-start: 1;
  grid-row-end: 3; */
}
```

A maioria dos nossos quartos representa uma única célula de grade. Mas demos a nós mesmos uma grande sala de estar. Posicionamos este item usando `grid-column-start` e `grid-column-end`. Seus valores de propriedade representam as linhas de grade da coluna com as quais desejamos começar e terminar.

Você também notará que comentamos os valores de propriedade para o posicionamento da linha de grade deste item. Descomente as propriedades `grid-row-start` e `grid-row-end` para ver como nossa sala de estar pode ficar ainda maior ocupando a trilha de grade entre as linhas de grade da primeira e da terceira linha.

Essas propriedades nos permitem usar nossas linhas de grade existentes para informar aos itens quantas linhas e colunas cada item deve abranger. Reserve um minuto para brincar com os valores da propriedade aqui.

Em seguida, precisamos usar nosso espaço de forma mais eficiente. Faremos com que o resto dos nossos quartos se estenda por várias células da grade e preencha o resto do nosso apartamento.

styles.css

```
.container {
  display: inline-grid;
  grid-template: 40px 40px 40px 40px 40px / 40px 40px 40px
40px 40px;
  background-color: lightblue;
}

.room {
  border: 1px solid;
  font-size: 50%;
  text-align: center;
}

#living-room {
  grid-column-start: 1;
  grid-column-end: 6;
  grid-row-start: 1;
  grid-row-end: 3;
}

#kitchen {
  grid-column: 4 / 6;
  grid-row: 3 / 6;
}

#bedroom {
  grid-column-start: 2;
  grid-column-end: 4;
  grid-row-start: 3;
  grid-row-end: 5;
}
```

```
#bathroom {
  grid-column-start: 1;
  grid-column-end: 2;
  grid-row-start: 3;
  grid-row-end: 6;
}

#closet {
  grid-column-start: 2;
  grid-column-end: 4;
  grid-row-start: 5;
  grid-row-end: 6;
}
```

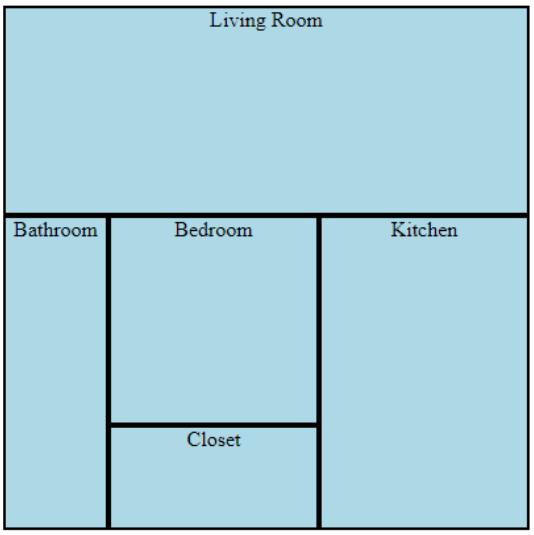
Agora temos as plantas do nosso apartamento completo. Se você der uma olhada no seletor `#kitchen`, verá que usamos propriedades abreviadas aqui. `grid-column` é apenas uma combinação de `grid-column-start` e `grid-column-end` com uma barra entre os dois valores. E `grid-row` é a versão abreviada para definir o posicionamento da linha de um item.

Um problema com a nossa planta baixa é que o banheiro e a cozinha ficam em extremidades opostas do apartamento. Nós economizaríamos dinheiro no encanamento colocando esses dois quartos de costas um para o outro. Tome um minuto agora e veja se você pode mudar as posições inicial e final do banheiro, quarto e armário para que o banheiro fique ao lado da cozinha. Você pode usar as propriedades longas ou abreviadas aqui.

2.5. `grid-area`

Agora você sabe como posicionar seus itens de grade usando linhas de linha e coluna. Mas existem outras maneiras de posicionar itens e é aí que as coisas podem ficar um pouco confusas.

Existe um atalho ainda mais curto para posicionar itens de grade com linhas inicial e final. Você pode combinar `grid-row-start / grid-column-start / grid-row-end / grid-column-end` em uma linha usando `grid-area`.



Nossa sala de estar acima pode ser escrita assim:

styles.css

```
#living-room {
  grid-area: 1 / 1 / 3 / 6;
}
```

Mas a `grid-area` também pode se referir a algumas coisas diferentes.

Em vez de usar as linhas de grade para posicionar todos os itens em uma grade, podemos criar um layout visual da grade em palavras. Para fazer isso, damos a cada item da grade um nome usando `grid-area`.

Então nossa sala de estar pode ser escrita assim:

styles.css

```
#living-room {
  grid-area: living-room;
}
```

Poderíamos fazer isso com todos os nossos itens de grade e dar a cada quarto um valor de `grid-area` como um nome. Então podemos mapear toda a estrutura com o container grid usando `grid-template-áreas`.

styles.css

```
.container {
  display: inline-grid;
  grid-template: 40px 40px 40px 40px 40px / 40px 40px 40px
40px 40px;
  background-color: lightblue;
  grid-template-areas:
    "living-room living-room living-room living-room
living-room"
    "living-room living-room living-room living-room
living-room"
    "bedroom bedroom bathroom kitchen kitchen"
    "bedroom bedroom bathroom kitchen kitchen"
    "closet closet bathroom kitchen kitchen"
}

.room {
  border: 1px solid;
  font-size: 50%;
  text-align: center;
}

#living-room {
   grid-area:  living-room;
}

#kitchen {
  grid-area: kitchen;
}

#bedroom {
  grid-area: bedroom;
}

#bathroom {
  grid-area: bathroom;
```

```
}

#closet {
  grid-area: closet;
}
```

Podemos até usar `.` para indicar células vazias. Digamos que nosso apartamento esteja recebendo um aquecedor de água e uma lavadora/secadora. Podemos não ter certeza do layout exato, mas podemos visualizar algum espaço facilmente removendo mais espaço no banheiro e na cozinha:

styles.css

```
.container {
  display: inline-grid;
  grid-template: 40px 40px 40px 40px 40px / 40px 40px 40px
40px 40px;
  background-color: lightblue;
  grid-template-areas:
    "living-room living-room living-room living-room
living-room"
    "living-room living-room living-room living-room
living-room"
    "bedroom bedroom bathroom kitchen kitchen"
    "bedroom bedroom bathroom kitchen kitchen"
    "closet closet . . ."
}

.room {
  border: 1px solid;
  font-size: 50%;
  text-align: center;
}

#living-room {
   grid-area:  living-room;
```

```
}

#kitchen {
  grid-area: kitchen;
}

#bedroom {
  grid-area: bedroom;
}

#bathroom {
  grid-area: bathroom;
}

#closet {
  grid-area: closet;
}
```

Então agora você conhece duas maneiras muito diferentes de usar a `grid-area` em um item de grade. Você pode até ver o termo “área de grade” se referir a um grupo de células. Por exemplo, todas as células de grade da sala de estar juntas são uma área de grade. A analogia do apartamento deve ajudar. Um item de grade pode ocupar várias células formando uma área da grade muito parecida com uma sala com quatro paredes em um apartamento.

À medida que você percorre as atribuições, você encontrará mais terminologia, como `span` e `auto`, ao posicionar itens de grade nas faixas. Também existem propriedades para justificar e alinhar itens de grade semelhantes ao Flexbox. A melhor maneira de aprender toda essa terminologia e como posicionar os itens é com muita prática!

2.6. Faça o exercício - grid-layout-1

3. Propriedades Avançadas de Grid

Vamos abordar esta lição de forma prática. Vamos configurar uma grade com cinco colunas e duas linhas e aplicar alguns estilos para que tudo fique fácil de ver.

index.html

```
  <div class="grid-container">
    <div class="grid-item">
      <p>3way 1</p>
      <img
src="https://www.3way.com.br/wp-content/uploads/2022/10/3way_l
ogofundobranco-1.png" alt="top logo" />
    </div>
    <div class="grid-item">
      <p>3way 2</p>
      <img
src="https://www.3way.com.br/wp-content/uploads/2022/10/3way_l
ogofundobranco-1.png" alt="top logo" />
    </div>
    <div class="grid-item">
      <p>3way 3</p>
      <img
src="https://www.3way.com.br/wp-content/uploads/2022/10/3way_l
ogofundobranco-1.png" alt="top logo" />
    </div>
    <div class="grid-item">
      <p>3way 4</p>
      <img
src="https://www.3way.com.br/wp-content/uploads/2022/10/3way_l
ogofundobranco-1.png" alt="top logo" />
    </div>
    <div class="grid-item">
      <p>3way 5</p>
```

```
      <img
src="https://www.3way.com.br/wp-content/uploads/2022/10/3way_l
ogofundobranco-1.png" alt="top logo" />
    </div>
    <div class="grid-item">
      <p>3way 6</p>
      <img
src="https://www.3way.com.br/wp-content/uploads/2022/10/3way_l
ogofundobranco-1.png" alt="top logo" />
    </div>
    <div class="grid-item">
      <p>3way 7</p>
      <img
src="https://www.3way.com.br/wp-content/uploads/2022/10/3way_l
ogofundobranco-1.png" alt="top logo" />
    </div>
    <div class="grid-item">
      <p>3way 8</p>
      <img
src="https://www.3way.com.br/wp-content/uploads/2022/10/3way_l
ogofundobranco-1.png" alt="top logo" />
    </div>
    <div class="grid-item">
      <p>3way 9</p>
      <img
src="https://www.3way.com.br/wp-content/uploads/2022/10/3way_l
ogofundobranco-1.png" alt="top logo" />
    </div>
    <div class="grid-item">
      <p>3way 10</p>
      <img
src="https://www.3way.com.br/wp-content/uploads/2022/10/3way_l
ogofundobranco-1.png" alt="top logo" />
    </div>
  </div>
```

styles.css

```
.grid-container {
  resize: both;
  overflow: auto;
  display: grid;
  gap: 4px;
  padding: 4px;
  border: 1px solid grey;
  background-color: skyblue;
  grid-template-rows: 150px 150px;
  grid-template-columns: 150px 150px 150px 150px 150px;
}

.grid-item {
  background-color: #444;
  text-align: center;
  color: #ddd;
  font-family: sans-serif;
  font-size: 1.5rem;
  padding: 12px;
}

p {
  margin: 12px auto 24px;
}

img {
  height: 60px;
}
```

3.1. Repeat

`repeat()` é uma função CSS disponível para as propriedades do modelo CSS Grid que nos permite definir um número de linhas ou colunas e o tamanho que queremos sem ter que digitar manualmente o tamanho de cada faixa individual. Por exemplo, em nossa configuração acima:

```
.grid-container {
  grid-template-rows: 150px 150px;
  grid-template-columns: 150px 150px 150px 150px 150px;
}
```

pode ser reescrita como:

```
.grid-container {
  grid-template-rows: repeat(2, 150px);
  grid-template-columns: repeat(5, 150px);
}
```

Confira por si mesmo!

3.2. Unidades fracionárias (fr)

Agora que sabemos como criar rapidamente muitas trilhas de grade, é hora de aprender como começar a torná-las dinâmicas. Dinâmico, neste contexto, significa simplesmente “flexível” ou “responsivo de alguma forma”. O oposto de dinâmico é estático, ou fixo em uma certa altura definida, como `150px`, que usamos na configuração desta grade de amostra.

A maneira mais básica de tornar nossos itens de grade dinâmicos é usando unidades fracionárias, também conhecidas como `fr`.

A unidade `fr` é uma maneira de distribuir qualquer espaço restante na grade. Por exemplo, se tivermos uma grade de quatro colunas com uma largura total de `400px` e quatro itens de grade em uma trilha de coluna atribuída a `1fr` como seu tamanho, todos os itens da grade devem receber uma fração desses `400px` de espaço, que é 100 pixels.

Vamos dar uma olhada no que acontece se dermos às nossas faixas de coluna e linha na grade de amostra, criamos uma largura dinâmica de `1fr` em vez de uma largura estática de 150px:

```
.grid-container {
  grid-template-rows: repeat(2, 1fr);
  grid-template-columns: repeat(5, 1fr);
```

```
}
```

Observe como todos os nossos itens de grade agora preenchem toda a largura e altura da grade? Legal, certo? Agora, tente redimensionar esse exemplo e veja o que acontece. Ainda mais legal!

Também podemos dizer aos nossos itens de grade para distribuir o espaço restante desproporcionalmente. Por exemplo, se dividirmos as 5 colunas dando às duas primeiras um tamanho de faixa de 2fr e as três restantes um tamanho de faixa de 1fr, as duas primeiras faixas terão o dobro do espaço restante que as outras. Compare este exemplo com o anterior:

```
.grid-container {
  grid-template-rows: repeat(2, 1fr);
  grid-template-columns: repeat(2, 2fr) repeat(3, 1fr);
}
```

Neste exemplo, há muita coisa acontecendo em nossas `grid-template-columns`, mas reserve um minuto para entender o que está escrito.

A chave aqui é que as duas primeiras colunas são atribuídas a unidades de `2fr` e as três restantes são atribuídas a `1fr.` Isso significa que, à medida que a grade cresce e diminui dinamicamente, o espaço será distribuído em quantidades diferentes entre essas colunas, especificamente, duas vezes mais pixels para as duas primeiras colunas do que as três restantes.

Observe que quando você redimensiona, os itens da grade crescem proporcionalmente a quantas unidades fr eles são alocados, conforme mencionado acima.

Você também pode misturar unidades estáticas (como `px`) e unidades dinâmicas (como `fr`).

Você deve ter notado neste ponto que quando você redimensiona a grade para o maior tamanho possível, não há limite para o tamanho dos itens da grade. No entanto, quando você o redimensiona o menor possível, há um tamanho “menor” distinto que a grade permitirá que seus itens sejam. Nesse caso, é o menor tamanho

que o elemento `<p>` ou `<img>` pode ter sem transbordar. Este ponto de interrupção é o valor mínimo de `min-content` do item. Essa palavra-chave CSS é muito útil, mas está além do escopo desta lição.

3.3. Tamanhos mínimo e máximo de trilha: min() e max()

Quando redimensionamos nossa grade super pequena, é reconfortante saber que o navegador impedirá que o item encolha além do valor `min-content`. No entanto, nós realmente não queremos confiar nisso na maioria das vezes. É muito melhor para você decidir explicitamente como desenvolvedor quão pequeno e grande seu conteúdo deve ser, mesmo nas situações mais extremas.

Aprendemos sobre `min()` e `max()` em nossa lição anterior sobre funções CSS, mas um pouco de revisão não faz mal. Ambas as funções retornarão um valor com base nos argumentos que você fornecer. `min()` retornará o menor de todos os valores passados e `max()` retornará o maior. Por exemplo, `min(100px, 200px)` retornará um valor de `100px` sempre, enquanto `max(100px, 200px)` retornará um valor de `200px` sempre.

Você pode fornecer quantos argumentos quiser para essas funções:

```
.grid-container {
  grid-template-rows: repeat(2, min(100px, 200px, 300px,
400px, 500px));
  grid-template-columns: repeat(5, max(100px, 200px, 300px,
400px, 500px));
}
```

Claro, é bobagem dar a essas funções unidades estáticas porque o cálculo não tem sentido: o menor ou o maior valor sempre será retornado. No exemplo acima, as linhas da grade sempre terão um tamanho de `100px` (o menor dos cinco valores) e as colunas da grade sempre terão um tamanho de `500px` (o maior dos cinco). Mas quando fornecemos um valor dinâmico como um desses argumentos, desbloqueamos o real potencial dessas funções, principalmente no contexto de Grid:

```
.grid-container {
  grid-template-rows: repeat(2, min(200px, 50%));
  grid-template-columns: repeat(5, max(120px, 15%));
}
```

Neste caso, o tamanho da linha da grade será calculado a partir dos valores `200px` e `50%` da altura do container da grade. Em tempo real, o navegador comparará esses dois valores e aplicará o que for menor ao tamanho de nossa linha de grade. Essencialmente, o que isso diz a essa grade é que o tamanho da trilha deve ser 50% do espaço vertical total da grade (porque estamos definindo um tamanho de linha), a menos que esse número exceda `200px`. Essencialmente, você está definindo uma altura máxima para a pista.

Por outro lado, o tamanho da coluna da grade será calculado com base no maior dos dois valores `120px` e `15%` da largura do contêiner da grade. Ao fazer isso, estamos essencialmente definindo uma largura mínima de nosso tamanho de coluna de grade em `120px`. Confira o exemplo aqui e tente clicar e arrastar para alterar as dimensões da grade e ver como os itens da grade respondem.

3.4. Tamanhos mínimo e máximo dinâmicos

`minmax()`

`minmax()` é uma função CSS que é usada especificamente com Grid. Ele só pode ser usado com as seguintes propriedades CSS:

- `grid-template-columns`
- `grid-template-rows`
- `grid-auto-columns`
- `grid-auto-rows`

É uma função relativamente simples que recebe apenas dois argumentos:

1. O tamanho mínimo que a trilha de grade pode ser;
2. O tamanho máximo que a trilha de grade pode ser.

Ao contrário de `min()` e `max()`, pode fazer sentido usar valores estáticos para ambos os argumentos. Aqui está um exemplo da grade que estamos usando escrita com `minmax()` e alguns valores estáticos:

```
.grid-container {
  grid-template-rows: repeat(2, 1fr);
  grid-template-columns: repeat(5, minmax(150px, 200px));
}
```

Com nossos `grid-template-columns` definidos com valores `minmax()`, a largura de cada item da grade aumentará e diminuirá com o contêiner da grade à medida que ele for redimensionado horizontalmente. No entanto, à medida que a grade diminui, as trilhas da coluna param de diminuir em `150px` e, à medida que a grade cresce, elas param de crescer em `200px`. Fale sobre flexibilidade! Confira você mesmo.

`clamp()`

Ao contrário de `minmax()`, `clamp()` é uma função CSS que pode ser usada em qualquer lugar, não apenas em um contêiner de grade. Assim como com `min()` e `max()`, aprendemos sobre isso em uma lição anterior, mas vamos fazer uma revisão rápida. A sintaxe é a seguinte:

```
clamp(minimum-size, ideal-size, maximum-size)
```

O que isso faz é permitir que nosso item se redimensione até atingir um dos valores de limite mínimo ou máximo.

Como o propósito de `clamp()` é criar uma faixa de tamanho flexível com restrições, queremos usar um valor dinâmico para o argumento “tamanho ideal” e normalmente um tamanho estático para o tamanho mínimo e máximo, embora seja possível usar um valor dinâmico aqui também.

Aqui está um exemplo simples sem grade. Vamos olhar para trás em nossa grade em um momento:

```
.simple-example {
  width: clamp(500px, 80%, 1000px);
}
```

Este elemento, que podemos fingir ser apenas uma `div`, renderizará com uma largura igual a 80% da largura de seu pai, a menos que esse número seja menor que `500px` ou maior que `1000px`, nesse caso usará esses números como largura.

Ok, agora de volta à nossa grid:

```
.grid-container {
  grid-template-columns: repeat(5, clamp(150px, 20%, 200px));
}
```

Observe como as faixas ficam em `20%` da largura do contêiner até atingirem os limites mínimo ou máximo?

Usar `clamp()` e `minmax()` são métodos fantásticos para tornar as grades mais responsivas, garantindo que não atingimos pontos de interrupção críticos que tornam nosso site ruim. Isso é imperativo ao usar imagens e elementos que podem ter uma tendência a transbordar ou renderizar de maneiras indesejáveis quando empurrados para tamanhos extremos.

3.5. auto-fit e auto-fill

Esses dois valores são, na verdade, parte da especificação da função `repeat()`, mas foram salvos para o final da lição porque sua utilidade não é aparente até que você entenda a função `minmax()`. Aqui está o caso de uso: você deseja fornecer à sua grade um número de colunas que são flexíveis com base no tamanho da grade. Por exemplo, se nossa grade tiver apenas `200px` de largura, podemos querer apenas uma coluna. Se for `400px` de largura, podemos querer dois e assim por diante. Resolver esse problema com consultas de mídia exigiria muita digitação. Felizmente, o `auto-fit` e o `auto-fill` estão aqui para salvar o dia!

De acordo com a especificação W3 sobre auto-fit e auto-fill, ambas as funções retornarão “o maior inteiro positivo possível” sem que os itens da grade transbordem seu contêiner. Aqui está um exemplo simples:

```
.simple-example {
  display: grid;
  width: 1000px;
  grid-template-columns: repeat(auto-fit, 200px);
}
```

Para esta grade, temos uma largura definida de `1000px` e estamos dizendo para preencher nossas colunas com faixas de `200px` cada. Contanto que haja pelo menos cinco itens de grade, isso resultará em um layout de 5 colunas, não importa o

quê. Nesse caso, o `auto-fill` faria a mesma coisa. Em breve entraremos na diferença.

A verdadeira magia do `auto-fit` e `auto-fill` vem quando incorporamos `minmax()` na equação. Com `minmax()`, podemos dizer à nossa grade que queremos ter o maior número possível de colunas, usando as restrições da nossa função `minmax()` para determinar o tamanho de cada coluna, sem que ela sobrecarregue nossa grade. Confira como nossa grade fica legal quando a redimensionamos agora!

```
.grid-container {
  grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
}
```

Observe como, quando redimensionamos, as colunas sabem automaticamente quantos caberão sem nenhuma consulta de mídia. Se você não acha isso legal, é melhor verificar seu pulso!

Então, o que está acontecendo aqui especificamente com `repeat(auto-fit, minmax(150px, 1fr))`;? Simples! Lembre-se de que o `auto-fit` retornará o número inteiro positivo mais alto sem estourar a grade. Então, primeiro, o navegador precisa saber a largura da nossa grade: neste caso, é apenas a largura da janela (menos as margens) porque não a configuramos explicitamente. Para este exemplo, vamos fingir que nossa janela tem atualmente `500px` de largura. Segundo, o navegador precisa saber quantas trilhas de coluna de grade poderiam caber nessa largura. Para fazer isso, ele usa o valor mínimo em nossa função `minmax()`, pois isso produzirá o maior número de itens, que é `150px`. Se nossa janela tiver `500px` de largura, isso significa que nossa grade renderizará 3 colunas. Mas espere, tem mais! Uma vez que o navegador tenha determinado quantas colunas podemos ajustar, ele redimensiona nossas colunas até o valor máximo permitido pela nossa função `minmax()`. Nesse caso, nosso tamanho máximo é `1fr`, portanto, todas as três colunas receberão uma distribuição igual do espaço disponível. Conforme redimensionamos nossa janela, esses cálculos acontecem em tempo real e o resultado é o que você vê no exemplo acima!

3.6. E o auto-fill?

Na maioria dos casos, o `auto-fill` funcionará exatamente da mesma maneira que o `auto-fit`. A diferença só é perceptível quando há menos itens que podem

preencher toda a linha da grade uma vez. Quando a grade é expandida para um tamanho onde outro item da grade pode caber, mas não há mais nenhum, o `auto-fit` manterá os itens da grid em seu tamanho `max`. Usando o `auto-fill`, os itens de grid voltarão ao tamanho `min` assim que o espaço estiver disponível para adicionar outro item de grade, mesmo que não haja um para ser renderizado. Eles continuarão seu padrão de crescimento até o `max` e voltando ao `min` à medida que a grid se expande e mais espaço fica disponível para novas faixas de grid.

Para ver isso em ação, faça 2 exemplos, o primeiro com auto-fit e o segundo com auto-fill e veja o que acontece quando você redimensiona a grade horizontalmente:

**Observação: altere o arquivo index.html para ficar somente com 3 itens na grid. 3way1, 3way2 e 3way3.

Com `auto-fit`

```
.grid-container {
  grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
}
```

Com `auto-fill:`

```
.grid-container {
  grid-template-columns: repeat(auto-fill, minmax(150px,
1fr));
}
```