

Karen utilizou as extensões:

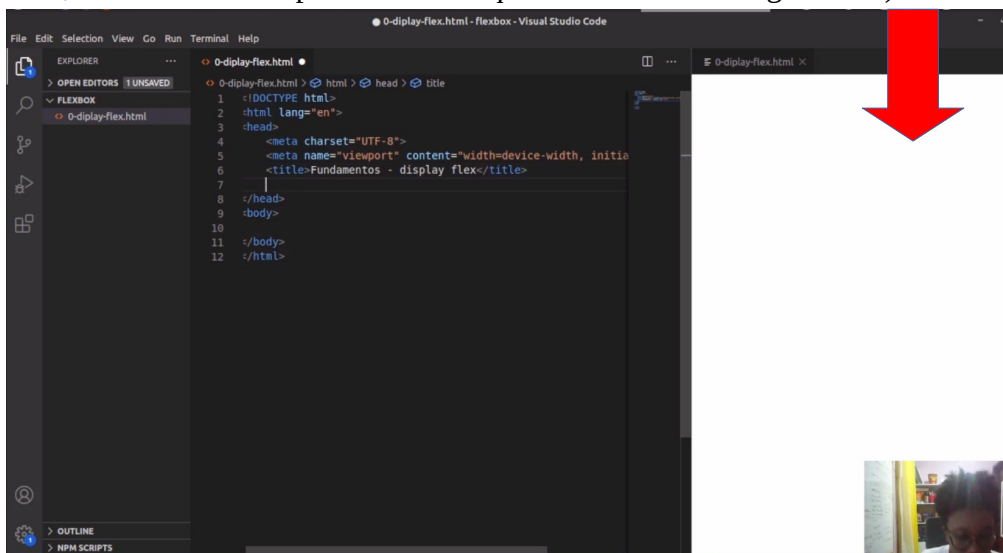
- *HTML Snippets* (faz o auto-complete, por ex, assim que digito html no vs code, já aparecem opções relacionadas, aí seleciono e já preencho automaticamente com as informações de meta charset etc):

```
0-diplay-flex.html
0-diplay-flex.html > html > body
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1">
6 <title>Document</title>
7 </head>
8 <body>
9
10 </body>
11 </html>
```

No meu VS já tem também! É só digitar:






html e já aparece a sugestão “html:5”, aí seleciona, e todas essas tags e esse texto à esquerda são digitados automaticamente.

- *Live HTML Previewer* (faz a renderização dos elementos HTML na tela da IDE, neste caso no VS Code, dessa forma não precisa abrir arquivo HTML no navegador etc):



O **Flexbox** auxilia a criação de layouts responsivos, sem a necessidade a definição de valores fixos.

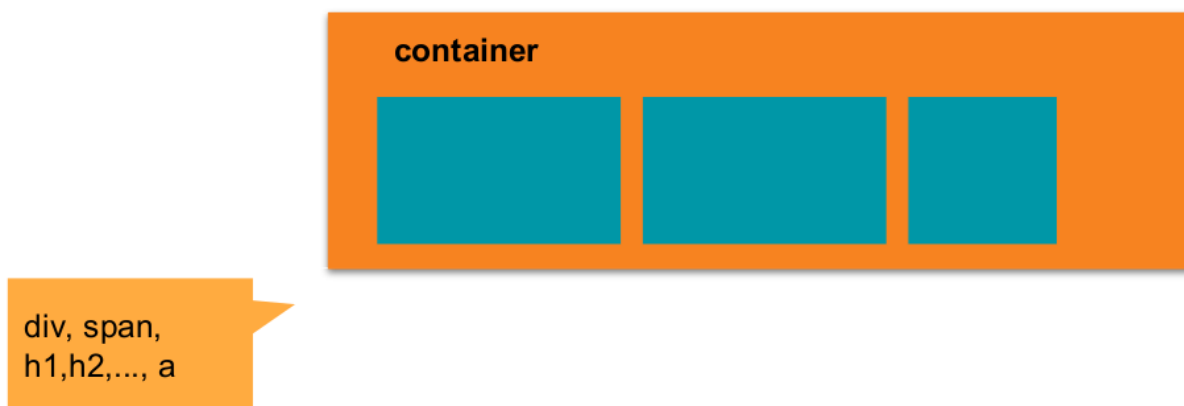
Os seguintes navegadores tem suporte para o Flexbox:

				
29.0	11.0	22.0	10	48

OBS: mas não é restringido somente à eles.

FLEX CONTAINER (parte 1)

É a tag que envolve os itens, ou seja, ela possui itens filhos. É nele que aplicamos a propriedade **display: flex**; Ela transforma todos os seus filhos *diretos* em **flex items**:



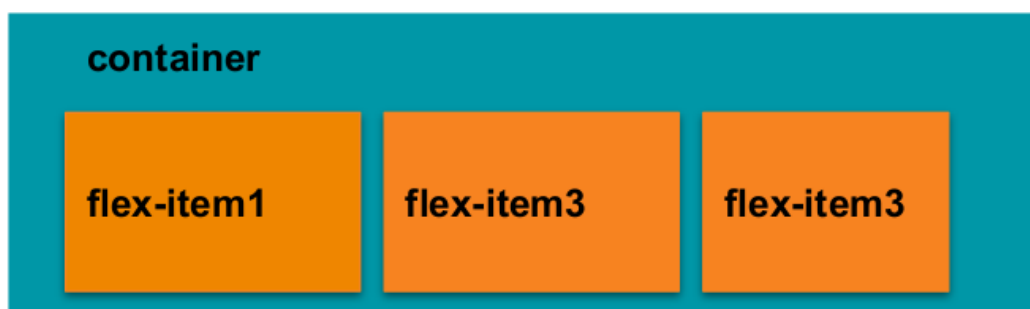
Independente do valor da largura dos filhos, eles ficarão alinhados e distribuídos dentro do container em linha reta. Pode ser feita em **div**, **h1-h6**, **span**, **a**, etc. Qualquer tag que tem itens filhos é passível de se aplicar **display: flex;**.

O Flex Container possui propriedades relacionadas à ele:

- **Display** (que é o inicializador do container)
- **flex-direction** (que faz o direcionamento desses itens (pode ser em linha, coluna..)).
- **flex-wrap** (que se aplica pra quebra de linha ??)
- **flex-flow** (que é uma abreviação do direction e do wrap)
- **justify-content** (alinha os itens do container de acordo com a sua direção)
- **align-items** (alinha os itens de acordo com o eixo do container)
- **align-content** (vai alinhar as linhas deste container)

FLEX ITEM (parte 2)

São itens que são *filhos diretos* de um *container*. MAS aplicando a propriedade **display: flex** podemos torná-lo um **Flex Container**, ou seja, ele se tornaria pai de outros itens dentro dele. Então, qualquer item que tenha filhos, podemos aplicar o display se quisermos e ir reinventando o layout.



O Flex Item possui propriedades relacionadas à ele também:

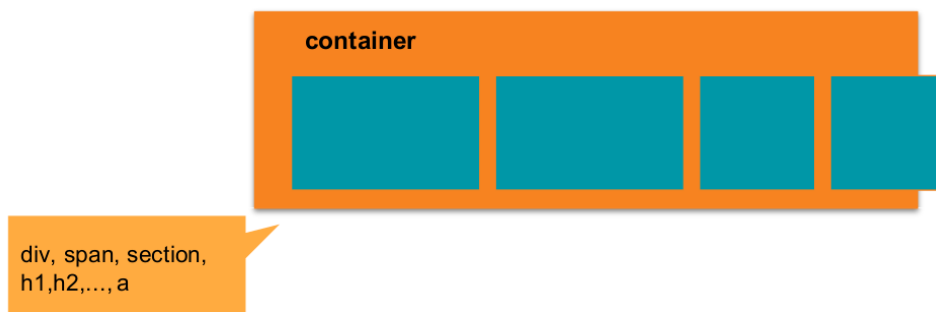
- **flex-grow** (define o fator de crescimento)
- **flex-basis** (define o tamanho inicial do item antes da distribuição do espaço restante dentro do container)
- **flex-shrink** (define a capacidade de redução)
- **flex** (é uma abreviação pras três propriedades acima: grow, basis, shrink)
- **order** (relacionado à ordem de distribuição e listagem dos itens)

- **align-self** (define o alinhamento de um item específico do nosso container)

AULA 2: FUNDAMENTOS – PARTE 1 FLEX CONTAINER

1) DISPLAY FLEX

É a propriedade de inicialização do flex container, ou seja, aplicando ela em uma div, section, etc, estarei dizendo que seus filhos agora serão *flex items*:



Neste exemplo os flex items estão saindo do meu container, vamos ver mais pra frente como ajustar os tamanhos dos itens para que eles não “estourem” o tamanho do meu container.

No meu VS Code:

Crio o style pro meu container (**.flex**) e pros seus “filhos” (**.item**)(pq dei uma class item pra eles):

```
0-display-flex.html
0-display-flex.html > html > body
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Fundamentos - display flex</title>
7   <style>
8     .flex{
9       max-width: 300px;
10      padding: 10px;
11      border: 2px solid black;
12    }
13
14    .item{
15      background-color: aqua;
16      margin: 5px;
17    }
18  </style>
19 </head>
```

No body vou criar meu container onde esses estilos serão aplicado, vou usar div.

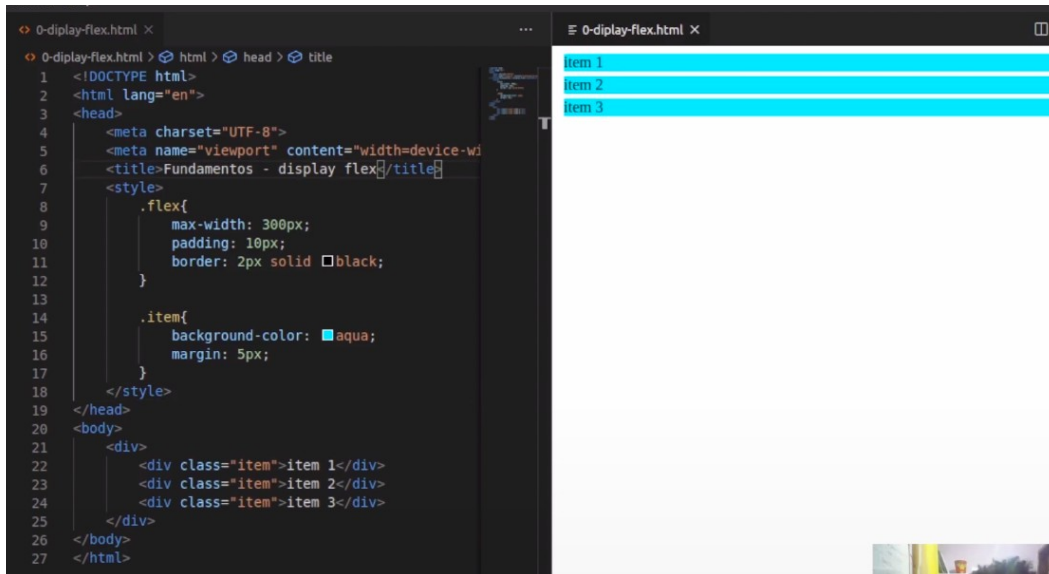
As divs, por padrão tem um comportamento block. Vamos ver o que isso implica agora:

OBS: atalho massa pra usar: **div.item*3** ao dar TAB vai me retornar os elementos prontos:

Dois exemplos de como o atalho `div.item*3` funciona no VS Code. À esquerda, uma captura mostra o editor com `<div>` selecionado e o atalho sendo digitado, com uma sugestão de autocompletar aparecendo. À direita, uma captura mostra o resultado final no código: `<div><div class="item"></div><div class="item"></div><div class="item"></div></div>`.

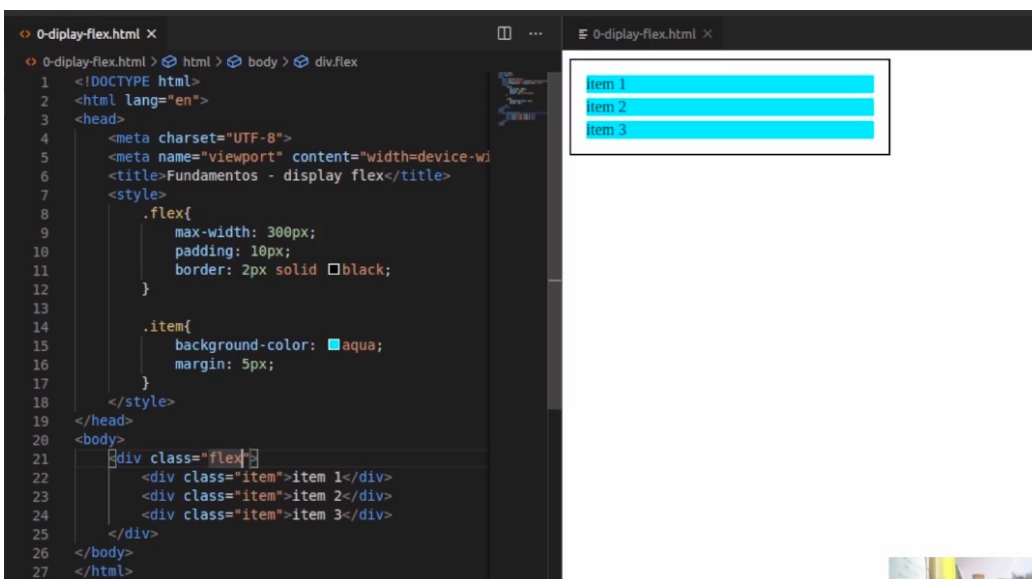
Meu VS também faz isso!

Vai ficar assim:



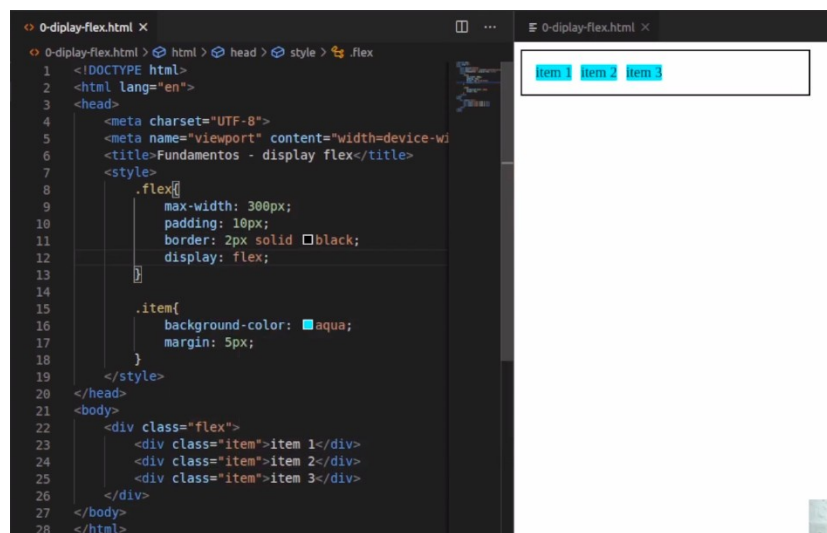
Todos os elementos estão ocupando 100% do espaço do container, e também estão organizados em linha.

Agora vamos colocar a classe **flex** (que chamamos la em **style**) para nossa div principal que é o nosso container:



isso aí que é um comportamento padrão de uma div, um display block!

E agora sim, vamos aplicar **display: flex** nesta div para ela ser nosso container e os itens serem seus filhos:

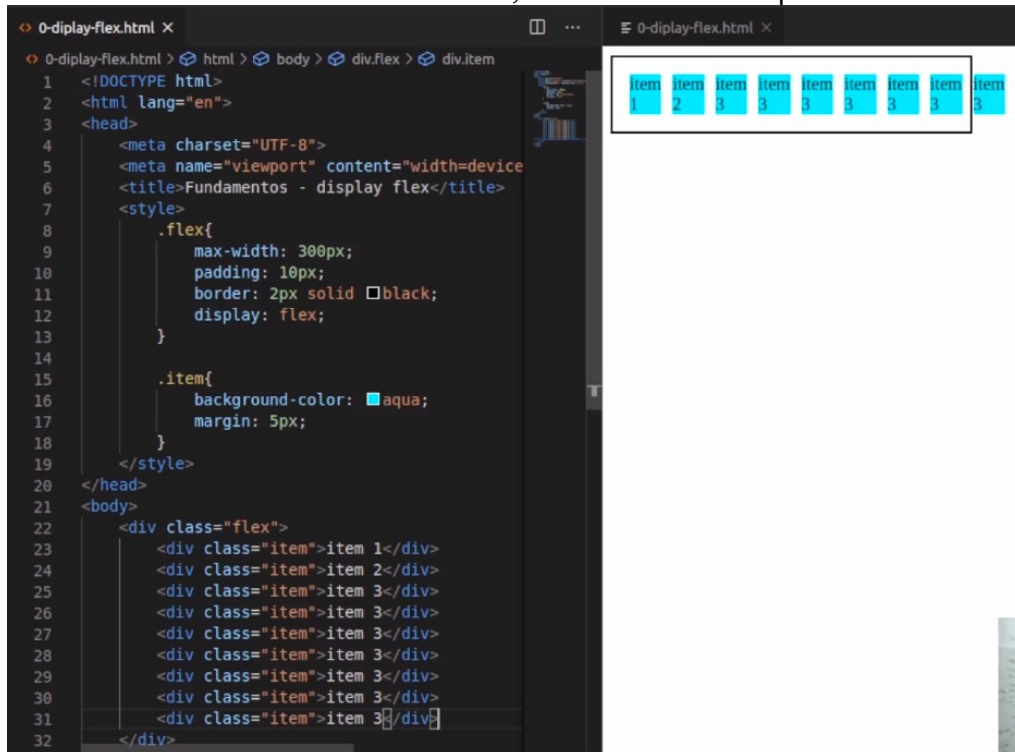


Com isso podemos ver que o **display: flex** modifica os itens, fazendo com que cada item dentro deste container passe a ocupar um tamanho máximo respeitando seu conteúdo e se abrigar dentro deste container respeitando a orientação em linha!

Vamos ver mais pra frente que é possível mudar também essa orientação em linha dos itens.

Um exemplo de aplicação prática do display flex é em mobile, principalmente em menus, onde o conteúdo deve adaptar-se às telas diferentes.

Dessa forma que fizemos acima, ainda não consigo controlar como faço para os itens não “vazarem” do meu container. Se eu inserir vários itens, eles se adaptam, quebrando o texto etc, porém ao acrescentar muitos em excesso, eu vou acabar extrapolando a div:



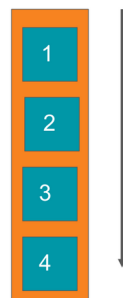
Vamos ver como solucionar isso logo mais!

2) FLEX-DIRECTION

Vamos entender o comportamento padrão de orientação horizontal de um flex container e aprender a modificar essa orientação horizontal.

O **flex-direction** é a propriedade que estabelece o eixo principal do container, definindo assim a direção e comportamento que os flex items são colocados no flex container.

São basicamente dois eixos: **row** que é horizontal e **column** que é o vertical.

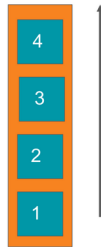


Podemos também ordená-los de forma inversa utilizando **reverse** junto com essas propriedades **row** e **column**:

- row-reverse: sentido oposto à direção do texto



- column-reverse: ordenação inversa, de baixo para cima



Vamos praticar tudo isso em uma UL e LI:

OBS: Atalho massa de novo: **li*5** depois dá um TAB vai gerar 5 tags LI pra mim!

```
1-flex-direction.html X
1-flex-direction.html > html > body > ul
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1">
6    <title>Fundamentos - Flex direction</title>
7
8    <style>
9      .flex-container{
10        margin: 0;
11        padding: 0;
12        display: flex;
13        list-style: none;
14      }
15    </style>
16  </head>
17  <body>
18    <ul>
19      <li>1</li>
20      <li>2</li>
21      <li>3</li>
22      <li>4</li>
23      <li>5</li>
24    </ul>
25  </body>
26</html>
```

Acima ainda não coloquei a classe **flex-container** na minha UL, então será exibido uma UL normal msm.

Agora vamos ver com os estilos da classe como ficará:

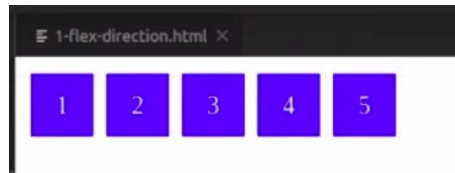
```
1-flex-direction.html X
1-flex-direction.html > html > body > ul.flex-container
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1">
6    <title>Fundamentos - Flex direction</title>
7
8    <style>
9      .flex-container{
10        margin: 0;
11        padding: 0;
12        display: flex;
13        list-style: none;
14      }
15    </style>
16  </head>
17  <body>
18    <ul class="flex-container">
19      <li>1</li>
20      <li>2</li>
21      <li>3</li>
22      <li>4</li>
23      <li>5</li>
24    </ul>
25  </body>
26</html>
```

Vamos agora estilizar também os itens logo abaixo:

```

.flex-item{
  background: blue;
  height: 50px;
  width: 50px;
  line-height: 50px;
  font-size: 20px;
  color: white;
  text-align: center;
  margin: 5px;
}
</style>
</head>
<body>
<ul>
<li class="flex-item">1</li>
<li class="flex-item">2</li>
<li class="flex-item">3</li>
<li class="flex-item">4</li>
<li class="flex-item">5</li>

```



OBS: **Margin: 5px** cria os espaços entre as caixinhas.

line-height: 50px é que define a posição do texto dentro do quadradinho, sem ele ficaria assim:



Sem ele ficaria assim:



Estes comportamentos acima são iguais as da função **flex-direction: row;**

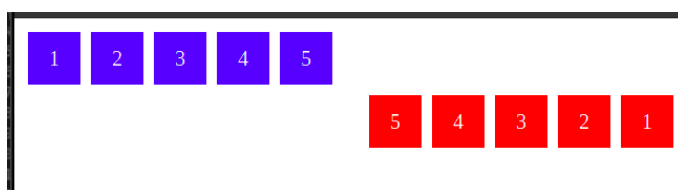
Pois o **row** somente aplica um comportamento de “fileira” ordenado horizontalmente igual a função que display flex já faz!

Agora vamos duplicar nossa UL e aplicar o **flex-direction: row-reverse** para comparar:

```

.row-reverse{
  flex-direction: row-reverse;
}
.row-reverse li{
  background-color: red;
}

```



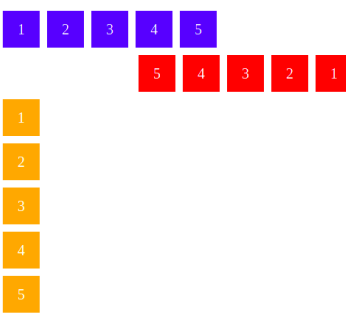
Então o **row-reverse** inverte a ordem dos itens, começando da direita pra esquerda ao invés de esquerda.

Agora vamos usar **flex-direction: column;**

```

.column {
  float: left;
  flex-direction: column;
}
.column li {
  background-color: orange;
}

```



Float: left; foi usado só pra deixar a coluna à esquerda. Se colocasse right, ela ficaria no canto direito.

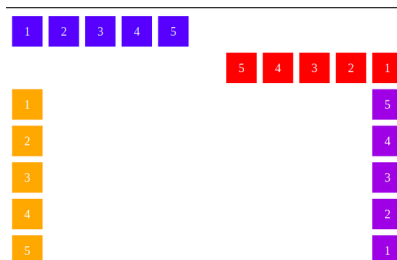
Com isso conseguimos colocar na vertical todos os itens.

E com **flex-direction: column-reverse;** vamos conseguir colocá-los de em ordem inversa:

```

.column-reverse {
  float: right;
  flex-direction: column-reverse;
}
.column-reverse li {
  background-color: blueviolet;
}

```



Float: right; foi usado só pra deixar a coluna à direita.

Na prática é muito comum precisar mudar a ordem dos nossos elementos na tela quando usamos breakpoints diferentes, ou seja, dimensões em tela diferentes.

Exemplo: quero fazer a minha pag web ser responsiva pra mobile, então meu minha flex-direction que era **row** vai virar **column**.

O bom de tudo isso é que o flex já é responsivo! Vc mexe na tela e tudo se adapta ao tamanho da tela!

3) FLEX-WRAP

É quem define se os itens devem ou não quebrar a linha.

Por padrão não os itens não quebram linhas, isso faz com que os flex itens sejam compactados além do limite do conteúdo, começando a vaziar.

Flex-wrap: nowrap;

Elementos
vazam

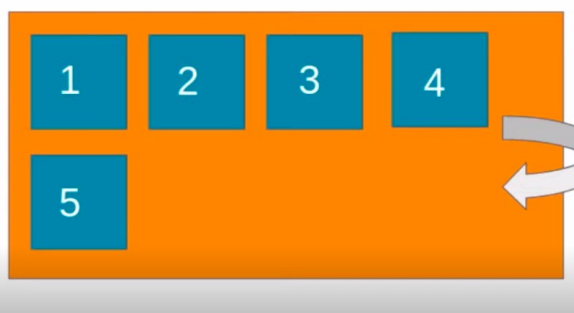
nowrap: é o padrão, não permite a quebra de linha.



Flex-wrap: wrap;

A partir do elemento que não está sendo comportado dentro do meu container, ele começa a fazer quebra de linha:

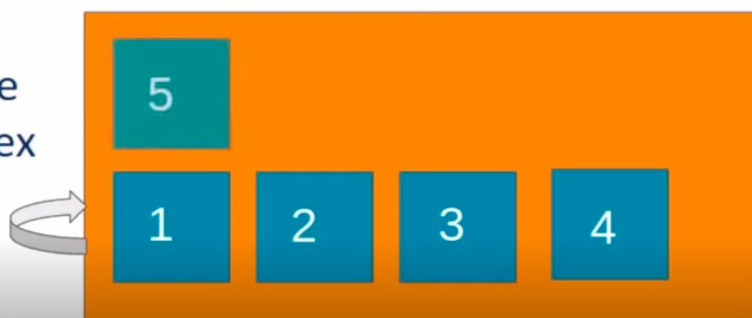
wrap: permite a quebra de linha assim que um dos flex itens não puder mais ser compactado.



Flex-wrap: wrap-reverse;

Possui a mesma lógica do wrap permitindo a quebra de linha, porém vai reverter a ordem dos itens, indo na direção contrária da linha acima:

wrap: permite a quebra de linha assim que um dos flex itens não puder mais ser compactado, porém na direção contrária da linha, acima.

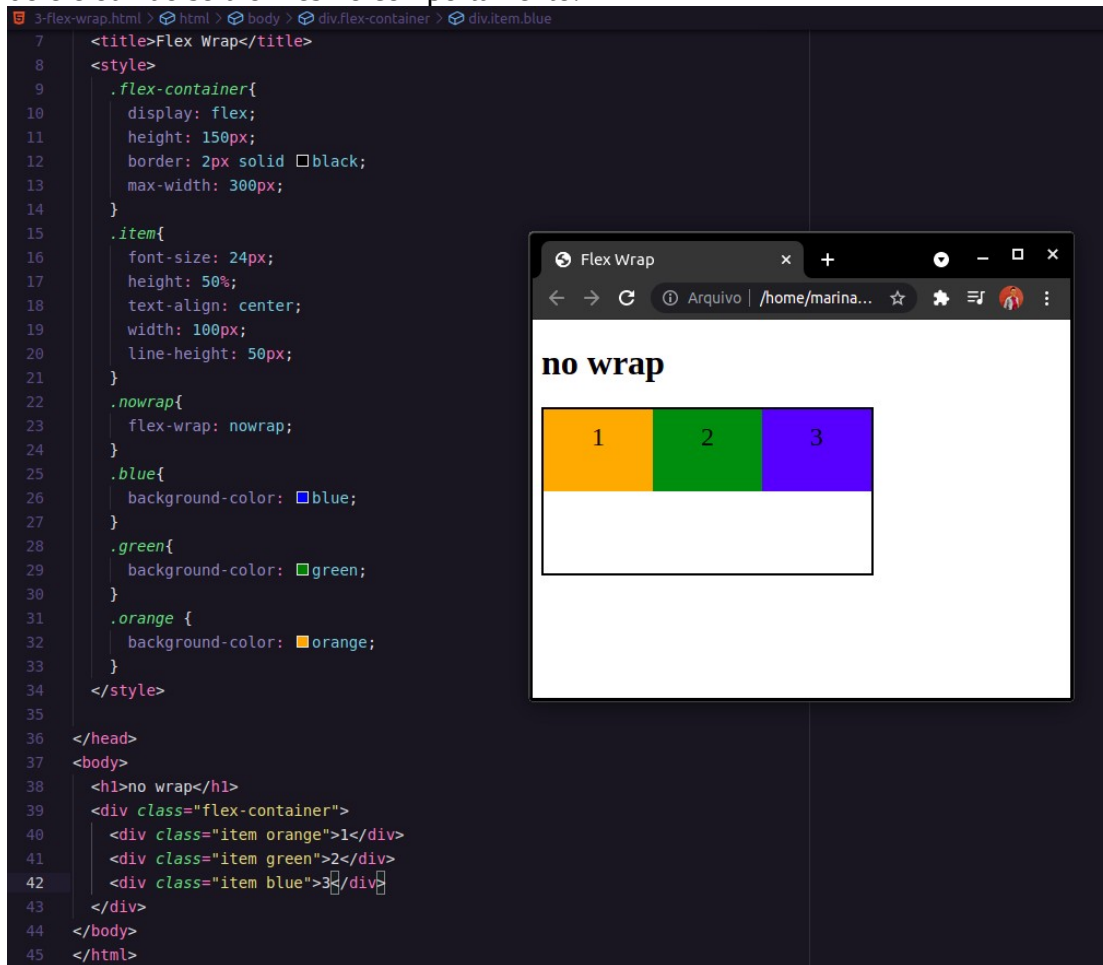


É como se a gente sempre escrevesse em linha e ao completar a quantidade total a gente dá um enter e vai pra linha de cima.

Vamos ver na prática:

Obs: quando coloco height: 50%, estou dizendo que quero uma altura com 50% do tamanho da div que ela tá dentro.

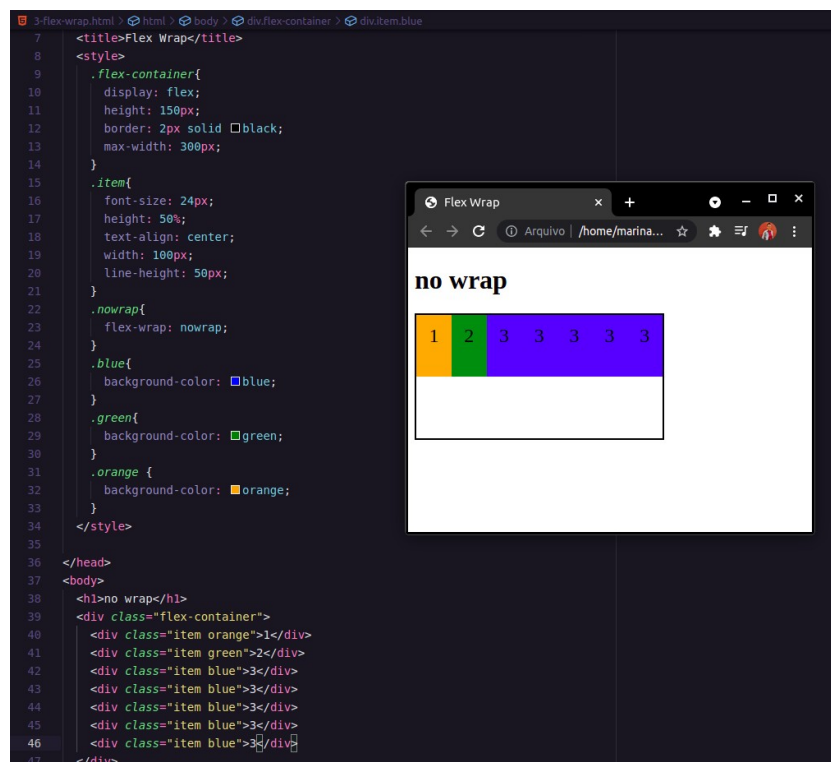
Vamos usar o **NOWRAP**:, já sabemos que ele tem o comportamento igual ao padrão, ou seja, usando ele ou não será o mesmo comportamento.



Ainda não apliquei o nowrap, mas vamos primeiro adicionar varios itens dentro do meu container e ver oq acontece:

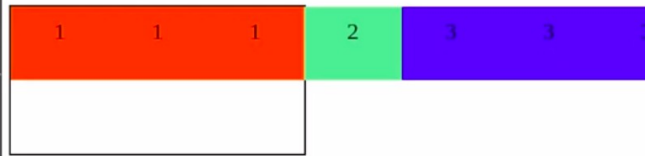
Ele comprime os itens até o limite do espaço do conteúdo dos itens, quando passa desse limite ele vaza do container.

Se **.item** tivesse a propriedade **min-width: 100px** ao inves de width normal, ele estouraria no quarto elemento que eu acrescentasse, pois meu container tem 300px. (veja abaixo)



```
2-flex-wrap.html > html > head > style > .item
<title>fundamentos - flex wrap</title>
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
<style>
.flex-container{
display: flex;
height: 150px;
border: 2px solid black;
max-width: 300px;
}
.item{
font-size: 24px;
height: 50%;
text-align: center;
min-width: 100px;
line-height: 50px;
}
.nowrap{
flex-wrap: nowrap;
}
.blue{
background-color: blue;
}
```

no wrap



Se eu adicionar **nowrap** não muda nada ele continua vazando:

```
2-flex-wrap.html x
2-flex-wrap.html > html > body > div.flex-container.nowrap
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
<div class="flex-container nowrap">
<div class="item orange">1</div>
<div class="item green">2</div>
<div class="item blue">3</div>
<div class="item orange">3</div>
</div>
</body>
</html>
```

no wrap

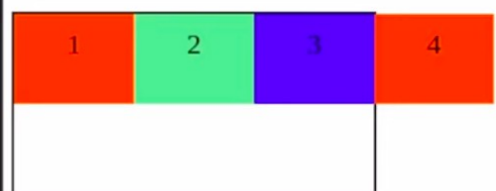


WRAP:

adiciona a linha de baixo, não vaza do container.

```
2-flex-wrap.html x
2-flex-wrap.html > html > body
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
<div class="flex-container nowrap">
<div class="item orange">1</div>
<div class="item green">2</div>
<div class="item blue">3</div>
<div class="item orange">4</div>
</div>
<div class="flex-container wrap">
<div class="item green">1</div>
<div class="item blue">2</div>
<div class="item orange">3</div>
<div class="item orange">4</div>
</div>
</body>
</html>
```

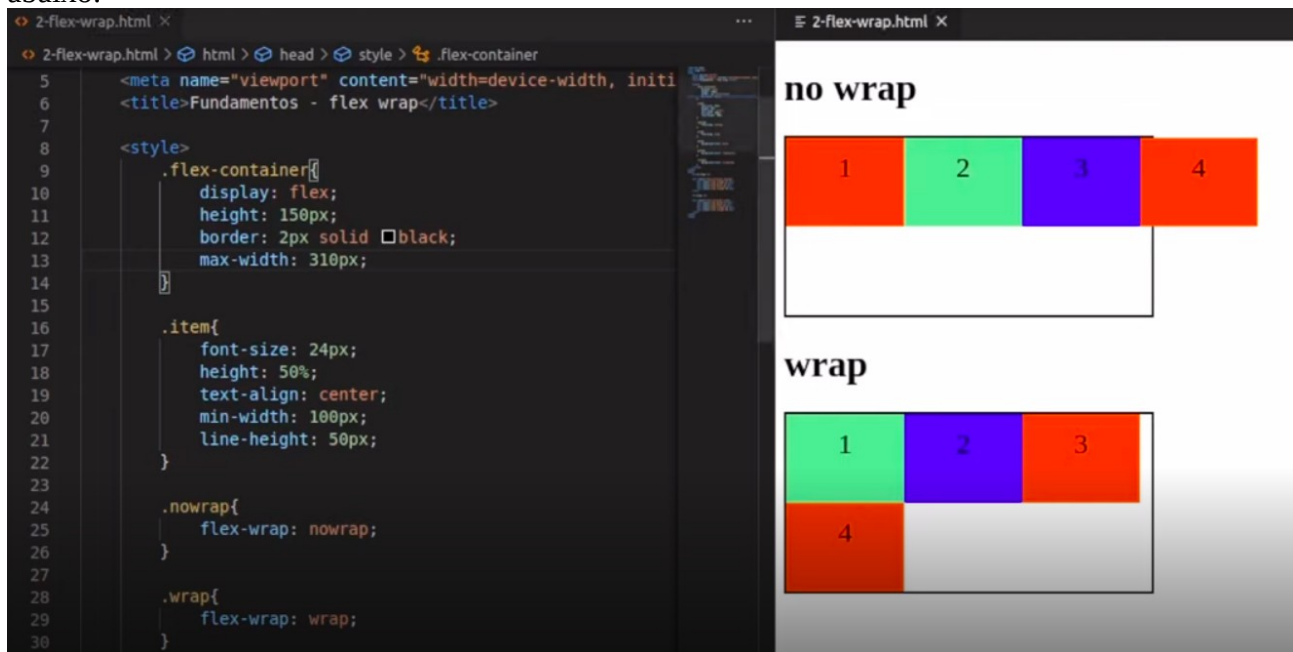
no wrap



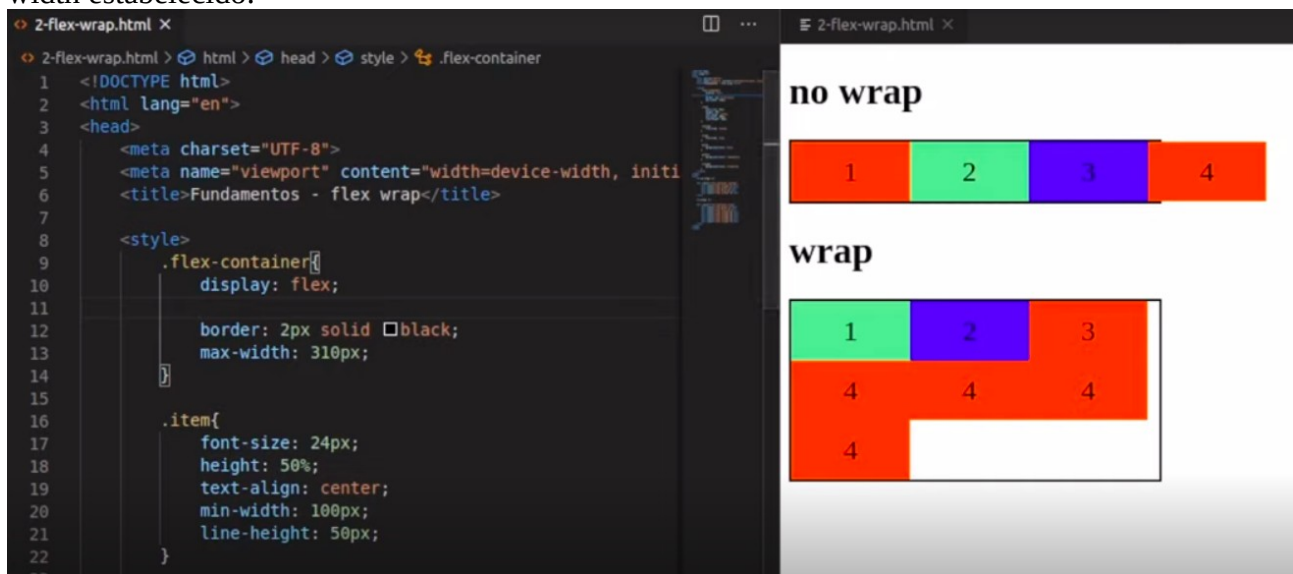
wrap



Se couber a direita, ele continua seguindo a linha. Por ex: se o container tivesse largura 400px, caberia o 4 em linha. Se fosse 310px não caberia, então ele continuaria jogando o 4 para baixo. Veja abaixo:



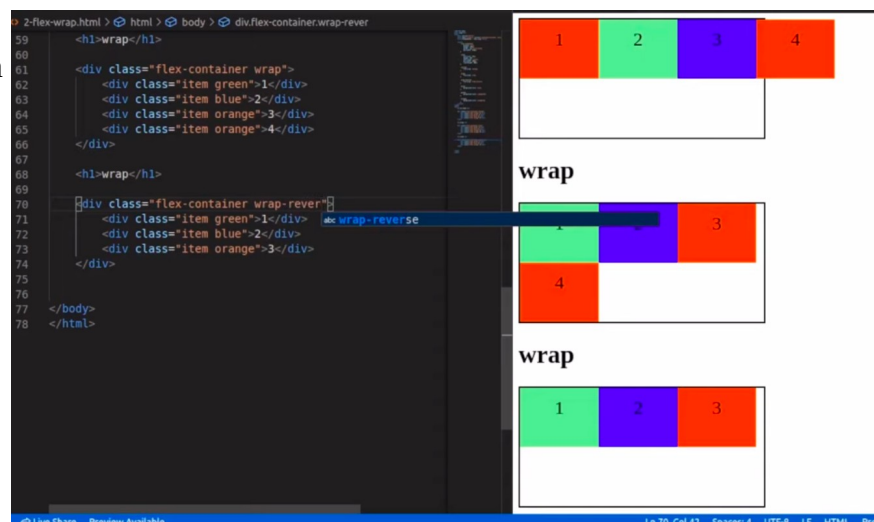
Se eu tirar a altura do meu container e adicionar um monte de itens, os itens irão se ajustar ao container. Eles serão comprimidos de acordo com a sua altura só que a largura seria respeitada pelo width estabelecido:

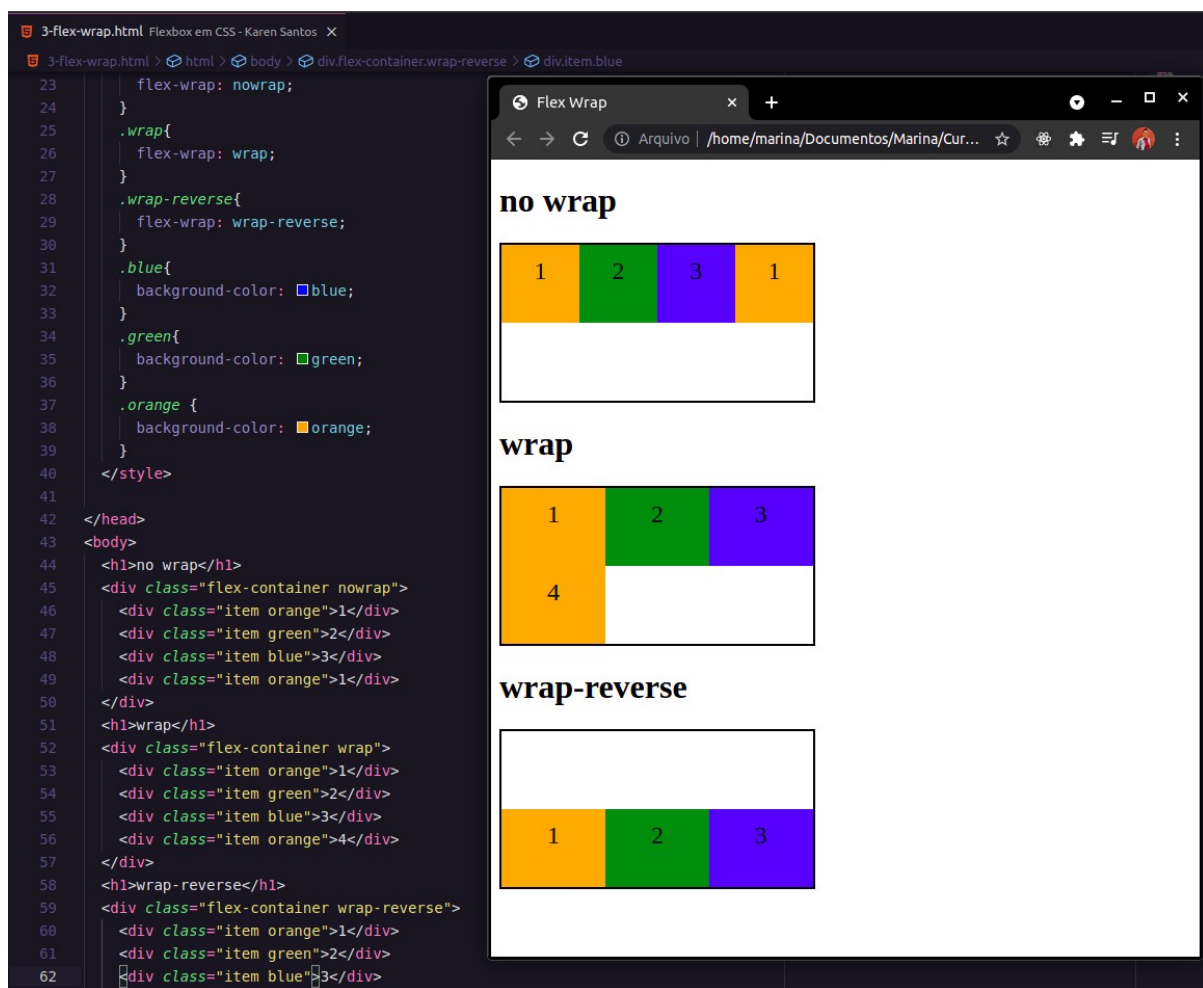


WRAP-REVERSE:

Aqui ao lado está sem o wrap reverse.

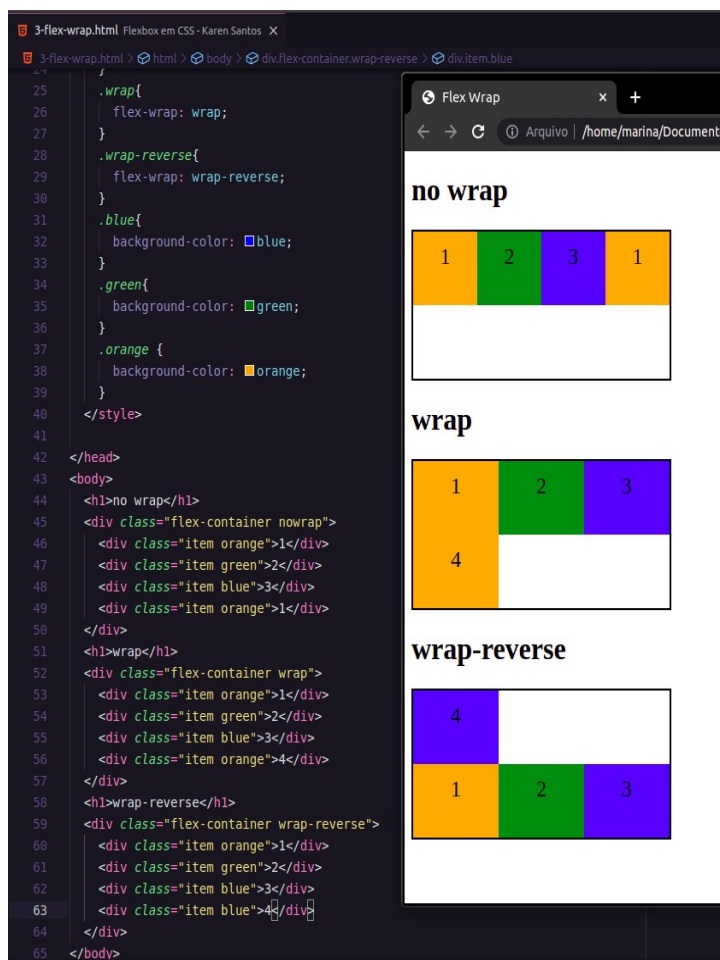
Adicionando ele os itens vão para se iniciar na linha de baixo, veja na próxima foto:





E agora adicionando um quarto item ao container ficará assim:

Apesar de ter os itens em linha, passo a ter meu ultimo elemento que vazou do container na linha acima. Ele inverte o fluxo da quebra de linha.



Caso a gente use essa propriedade com **flex-direction: column** ao invés de padrão (que é row) igual estamos usando. Vai ficar assim:

3-flex-wrap.html Flexbox em CSS - Karen Santos

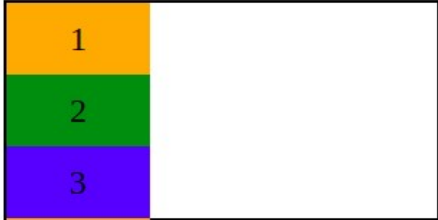
3-flex-wrap.html > html > head > style > .nowrap

```
8 <style>
9   .flex-container{
10     display: flex;
11     height: 150px;
12     border: 2px solid black;
13     max-width: 300px;
14     flex-direction: column;
15   }
16   .item{
17     font-size: 24px;
18     height: 50%;
19     text-align: center;
20     width: 100px;
21     line-height: 50px;
22   }
23   .nowrap{
24     flex-wrap: nowrap;
25   }
26   .wrap{
27     flex-wrap: wrap;
28   }
29   .wrap-reverse{
30     flex-wrap: wrap-reverse;
31   }
32   .blue{
33     background-color: blue;
34   }
35   .green{
36     background-color: green;
37   }
38   .orange {
39     background-color: orange;
40   }
41 </style>
42
43 </head>
44 <body>
45   <h1>no wrap</h1>
46   <div class="flex-container nowrap">
47     <div class="item orange">1</div>
48     <div class="item green">2</div>
```

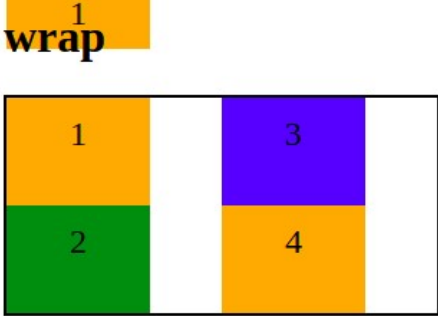
Flex Wrap

Arquivo | /home/marina/Documents/Marina

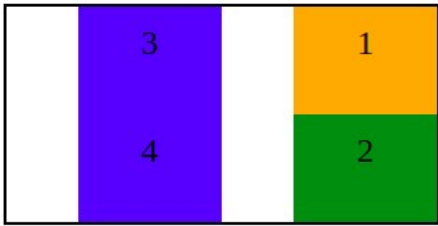
no wrap



wrap



wrap-reverse



Se eu tirar a propriedade **height** do meu **container** e também colocar **min-width** 100 (ao invés de width normal) no meu item, ficará assim:

3-flex-wrap.html

Flexbox em CSS - Karen Santos

3-flex-wrap.html > html > head > style > .flex-container

```
8 <style>
9   .flex-container{
10     display: flex;
11     border: 2px solid black;
12     max-width: 300px;
13     flex-direction: column;
14   }
15   .item{
16     font-size: 24px;
17     height: 50%;
18     text-align: center;
19     min-width: 100px;
20     line-height: 50px;
21   }
22   .nowrap{
23     flex-wrap: nowrap;
24   }
25   .wrap{
26     flex-wrap: wrap;
27   }
28   .wrap-reverse{
29     flex-wrap: wrap-reverse;
30   }
31   .blue{
32     background-color: blue;
33   }
34   .green{
35     background-color: green;
36   }
37   .orange {
38     background-color: orange;
39   }
40 </style>
41
42 </head>
43 <body>
44   <h1>no wrap</h1>
45   <div class="flex-container nowrap">
46     <div class="item orange">1</div>
47     <div class="item green">2</div>
48     <div class="item blue">3</div>
```

Flex wrap

Arquivo | /home/marina/Document

no wrap



wrap



wrap-reverse



Mantendo minha altura fixa (**height: 150px**) no container e usando **flex-direction: column**:

3-flex-wrap.html Flexbox em CSS - Karen Santos X

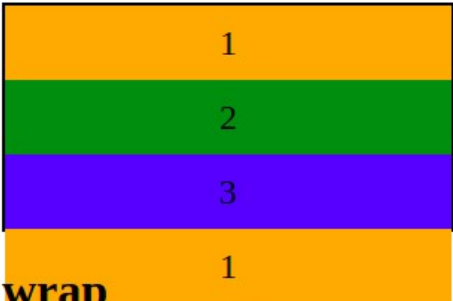
3-flex-wrap.html > html > head > style > .flex-container

```
8 <style>
9   .flex-container{
10    display: flex;
11    border: 2px solid black;
12    height: 150px;
13    max-width: 300px;
14    flex-direction: column;
15  }
16  .item{
17    font-size: 24px;
18    height: 50%;
19    text-align: center;
20    min-width: 100px;
21    line-height: 50px;
22  }
23  .nowrap{
24    flex-wrap: nowrap;
25  }
26  .wrap{
27    flex-wrap: wrap;
28  }
29  .wrap-reverse{
30    flex-wrap: wrap-reverse;
31  }
32  .blue{
33    background-color: blue;
34  }
35  .green{
36    background-color: green;
37  }
38  .orange {
39    background-color: orange;
40  }
41 </style>
42
43 </head>
44 <body>
45   <h1>no wrap</h1>
46   <div class="flex-container nowrap">
47     <div class="item orange">1</div>
48     <div class="item green">2</div>
```

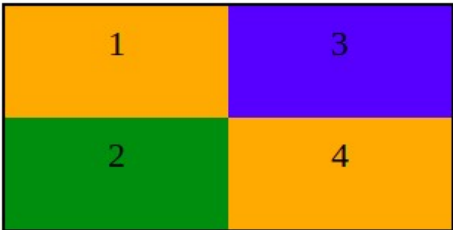
Flex wrap

Arquivo | /home/marina/Docume


no wrap



wrap



wrap-reverse



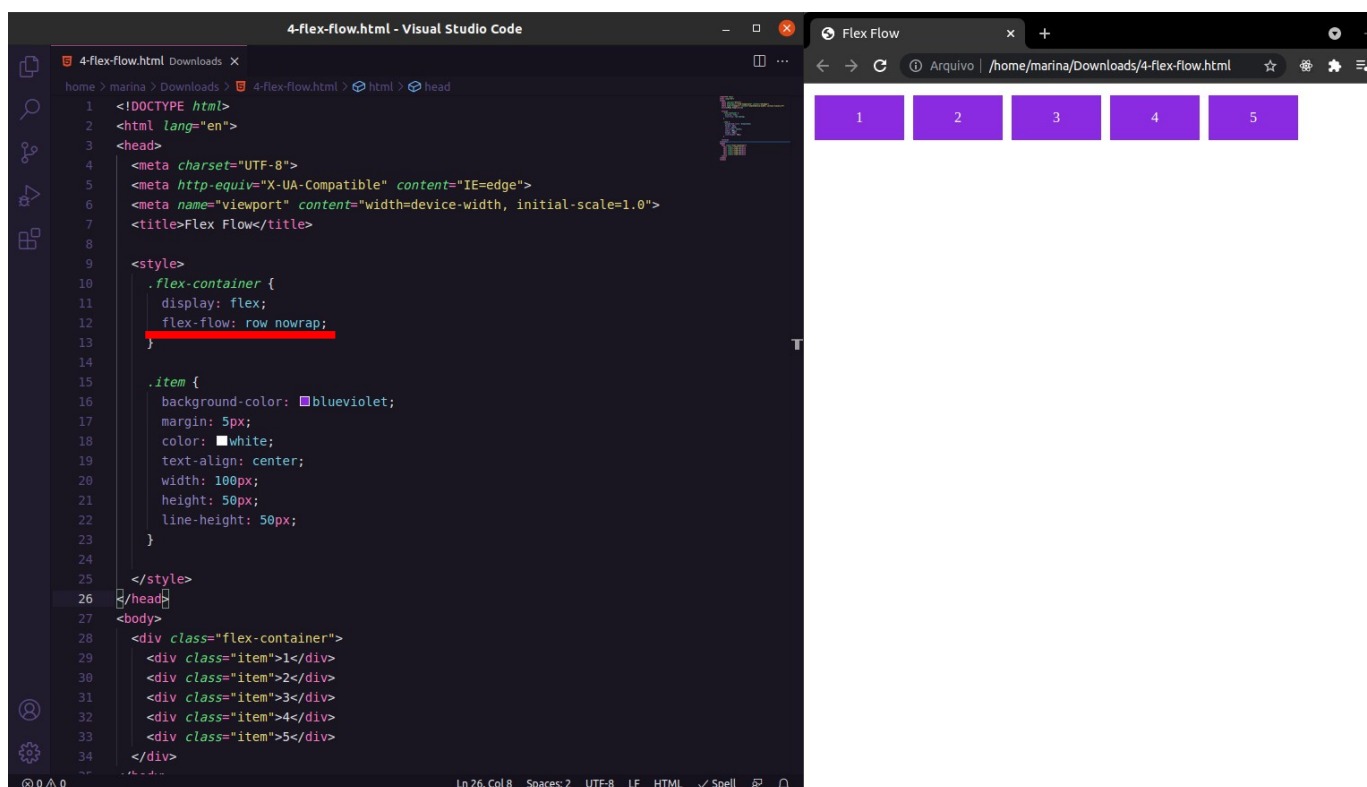
O **wrap** agrupa em colunas, quebrando a linha para o lado, contando em coluna os numero, de cima pra baixo.

O **reverse** faz o contrario do wrap, leva tudo pra direita, mas tambem contando pela coluna de cima pra baixo, como se fosse um float.

4) FLEX-FLOW

É um atalho para as propriedades **flex-direction** e **flex-wrap**. Seu uso não é tão comum, pois quando mudamos o **flex-direction** para **column**, mantemos o padrão do **flex-wrap** que é **nowrap**. Então ele acaba comprimindo a escrita ao invés de utilizar duas linhas para direcionar o texto.

Vamos entender como utilizar flex-flow com os valores que consideramos padrão que são o **row** e **nowrap** (Ou seja, não altera nada colocá-los ou não no nosso container!):

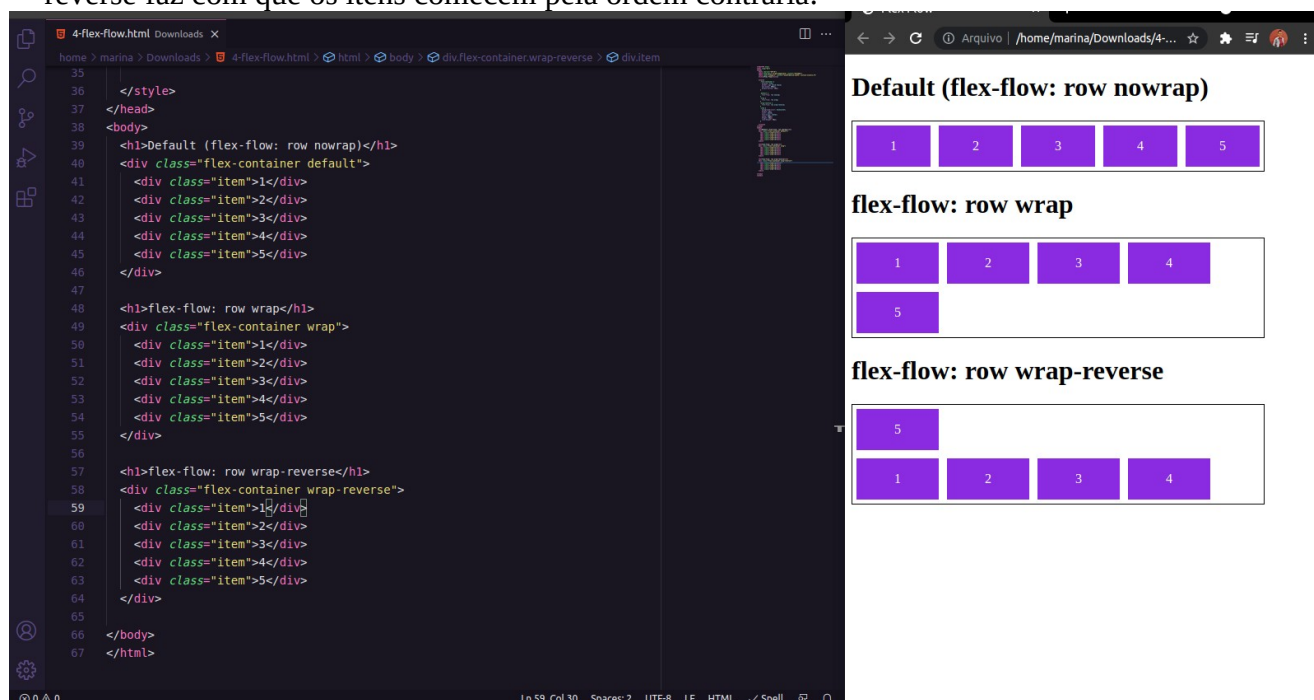


Colocando ou não **flex-flow: row nowrap**; teremos o mesmo resultado visto acima no navegador!

Vamos deixar essa propriedade como a classe **default** no nosso arquivo.

Agora se usarmos **row wrap**, vamos perceber que os itens vão para a linha de baixo a partir do momento que não cabem mais dentro da **div**.

E usando o **row wrap-reverse** vamos perceber aquele mesmo comportamento que vimos que o **reverse** faz com que os itens comecem pela ordem contrária:



Usando o **row-reverse nowrap**, vamos ter a inversão da ordem do default que fizemos primeiro:

flex-flow: row-reverse nowrap



Adicionando mais itens ao nowrap seu comportamento é espremer tudo até não caber mais, assim como vimos anteriormente:

flex-flow: row-reverse nowrap

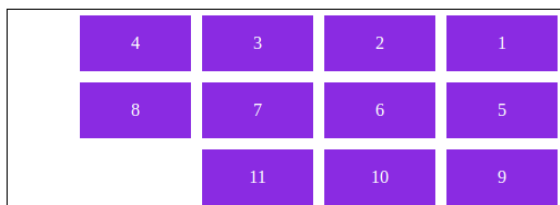


flex-flow: row-reverse nowrap



Se usarmos o **row-reverse wrap**, vamos ver que agora os itens não vão ficar na mesma linha como o exemplo acima, nós vamos ter eles distribuídos pelas linhas abaixo só que continuarão revertidos:

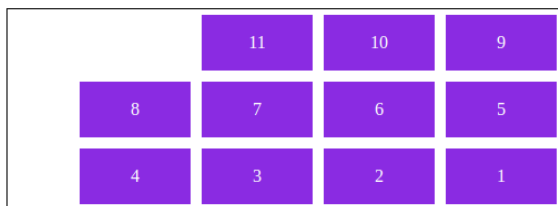
flex-flow: row-reverse wrap



Eles começam da direita pra esquerda e vão descendo pra linha de baixo

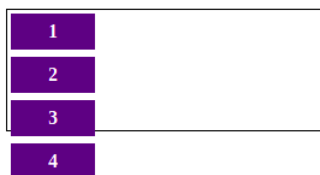
Agora usando tudo com reverse (**row e wrap**), teremos nossos itens também começando da direita pra esquerda, só que agora de baixo pra cima, eles vão subindo. Será tudo ao contrário:

flex-flow: row-reverse wrap-reverse



Agora vamos aplicar isso só que para **column**!

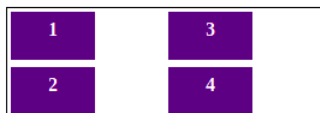
flex-flow: column nowrap



Agora com column os itens ficarão em colunas.

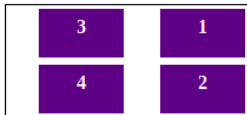
Usando nowrap, ele não comporta os itens dentro do container, acaba extrapolando o limite.

flex-flow: column wrap



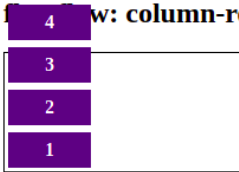
Usando wrap ele irá comportar todos os itens, jogando os excedentes para o lado direito, como se fosse um float right.

flex-flow: column wrap-reverse



Usando wrap-reverse ele apenas fará a inversão da ordem dos itens, começando da direita pra esquerda, de cima pra baixo.

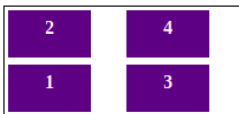
flex-flow: column-reverse nowrap



Usando column-reverse a coluna começará de baixo pra cima.

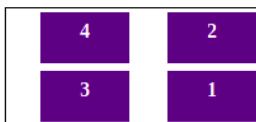
Com nowrap o container não suporta todo o conteúdo, ele extrapola.

flex-flow: column-reverse wrap



Com wrap o container comporta todos, e agora irá jogar também pra direita, só que com reverse ele começa de baixo pra cima.

flex-flow: column-reverse wrap-reverse



Com as duas propriedades usando reverse, teremos tudo invertido. A coluna começa da direita pra esquerda e os itens começam de baixo para cima.

Podemos dizer que quando usamos **wrap** as divs irão se “alongar” para caber os itens sem espremer demais e sem extrapolar ficando pra fora da DIV.

Já o nowrap ele não faz isso, pois chegando a um certo numero de itens adicionados, ele acaba extrapolando o tamanho da div, pois ele não aumenta o tamanho dela, ele apenas contrai o tamanho dos itens que estão dentro.

5) JUSTIFY-CONTENT

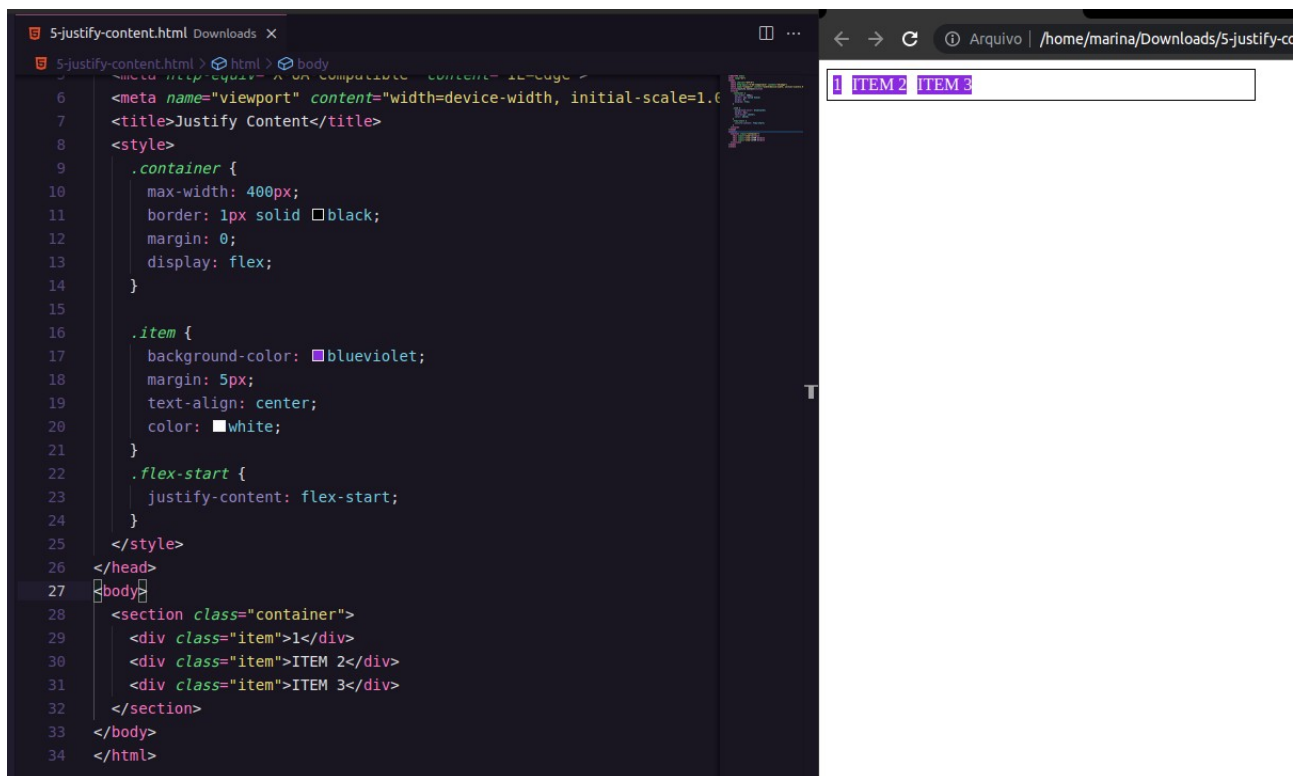
Essa propriedade vai se encarregar de alinhar os itens dentro do container de acordo com a direção pretendida e tratar da distribuição de espaçamento entre eles.

OBS: Ele distribui o espaçamento restante dentro de uma div então, caso os itens ocupem 100% de todo o container, o justify content não se aplica!

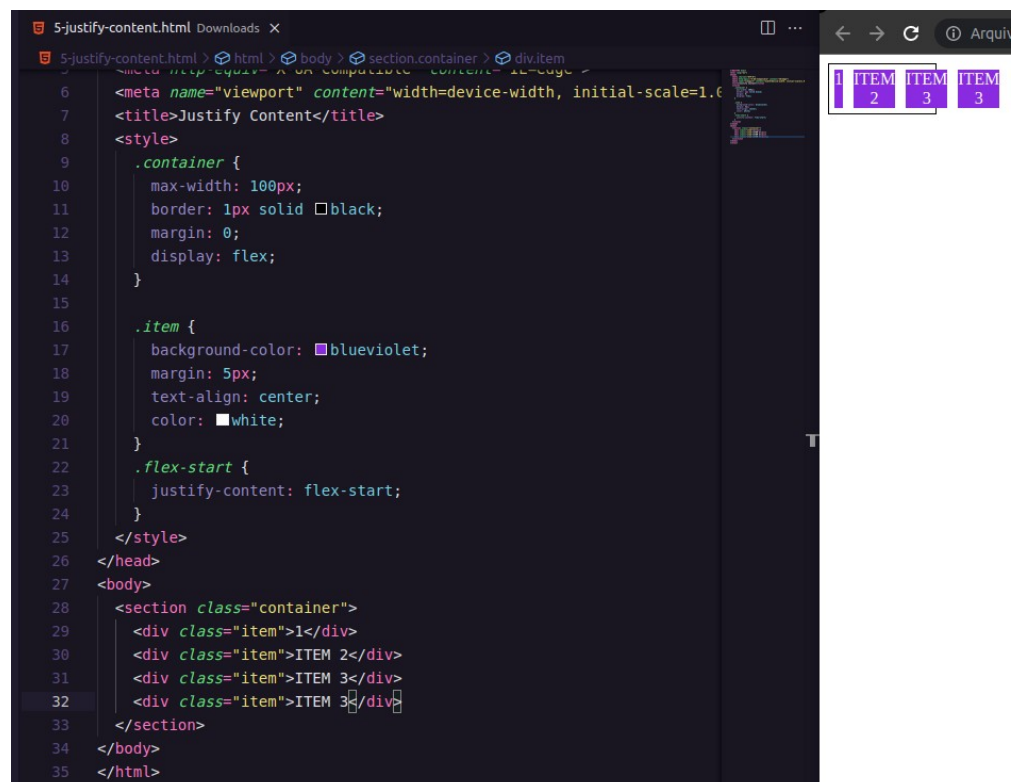
Ele possui algumas variações:

- 1) **flex-start:** faz o alinhamento no início do container;
- 2) **flex-end:** leva todos os itens pro final do container de acordo com a sobra do espaço respeitando sempre o limite que o item tem e ocupa;
- 3) **center:** traz os conteúdos para o centro do container;
- 4) **space-between:** que cria um espaçamento igual entre os elementos, mas ele pega o primeiro elemento colocando-o muito próximo ao início do container (prox a borda esquerda) e o outro elemento ele levaria para próximo do fim do container (prox a borda direita)
- 5) **space-around:** o espaçamento do meio é duas vezes maior que o inicial e final.

Justify-content: flex-start é como se fosse o próprio default, pois ele começa do começo mesmo, da esquerda pra direita. Se tirar ele, fica a mesma coisa.

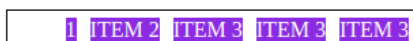


Obs: se adicionar mais itens (ou reduzir a largura) ele vai sair pra fora da div, extrapolando:



Agora com o flex-end fica assim, começando da direita pra esquerda:

justify-content: flex-end



Lembrando que ele também extrapola quando colocar muitos itens.

justify-content: center



Com o center, o conteúdo fica centralizado, distribuindo distancias iguais à direita e à esquerda.

OBS: também não conseguimos acomodar muitos itens, eles extrapolam.

O **space-between** distribui de forma igual os itens dentro da div, o espaçamento entre os itens é igual e uniforme, de acordo com o número de itens:

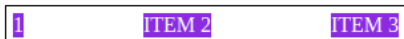
justify-content: space-between

Com apenas dois itens, ele terá esse comportamento.



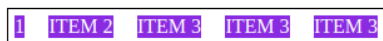
justify-content: space-between

Com mais um item podemos ver que ele realoca todos de maneira uniforme, ficando bem distribuidos.



justify-content: space-between

Mas ele suporta um numero max, pois também extrapola a div!



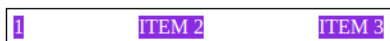
Já o **space-around** além de fazer essa distribuição uniforme que o between faz, ele também dá um espaçamento para as bordas não ficarem grudadas na div:

justify-content: space-around



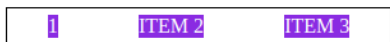
Compare a diferença entre between e around (utilizando apenas 3 itens):

justify-content: space-between



É como se o around criasse um padding para não deixar os itens 1 e o último muito pertos da borda.

justify-content: space-around



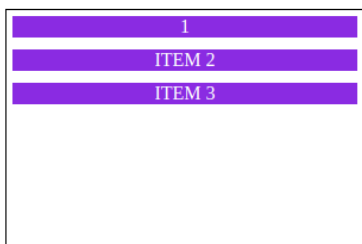
O justify-content tem também os valores **left**, **right**, **normal**, **stretch**, mas eles não se aplicam quando lidamos com flex container e com flex box

Vamos usar agora com COLUNAS!

Column

Usando flex-start:

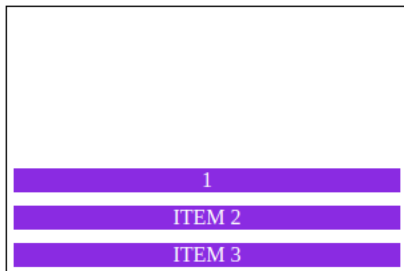
justify-content: flex-start



Os itens ocupam a largura total da div e se alocam como colunas, iniciando de cima pra baixo.

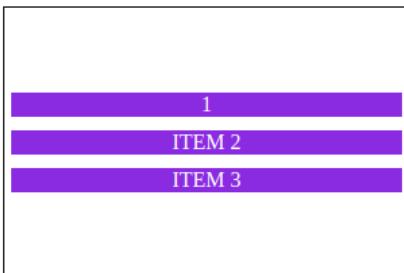
justify-content: flex-end

Usando **flex-end** os itens começam a ser distribuídos de baixo pra cima.



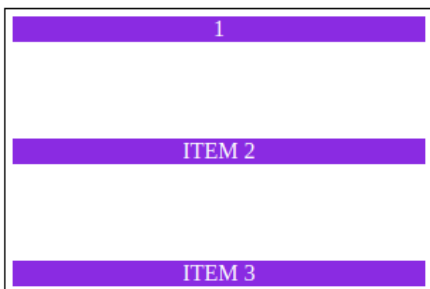
justify-content: center

O **center** vai centralizar a coluna no meio da div, colocando uma distância igual entre o topo e o bottom.



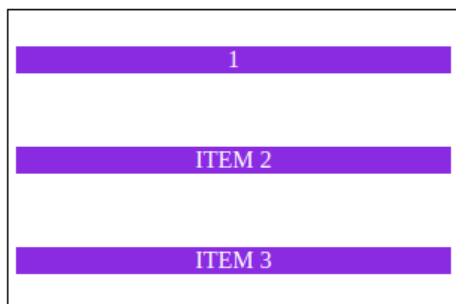
justify-content: space-between

Ele faz a mesma coisa que quando era ROW, distribui os itens dentro da div igualmente.



justify-content: space-around

Assim como na row, aqui temos o around, criando um padding para não ficar colado na borda.



Esse espaçamento de “padding” que ele cria é a metade do espaçamento entre os itens.

6) ALIGN-ITEMS

Align-item trata do alinhamento dos flex itens de acordo com o eixo do container. O alinhamento é diferente para quando os itens estão em colunas ou linhas!

Permite o alinhamento central no eixo vertical.

A diferença entre `align-items` e `justify-content` é que não precisamos conhecer a altura desses itens filhos do container. (no `justify-content` não aplicamos conceitos de altura no próprio item mas aplicamos no container. Agora usando `align-items` não precisaremos disso, pois ele vai tentar pegar essa proporcionalidade e expandir esses elementos crescendo).

TIPOS DE ALINHAMENTO:

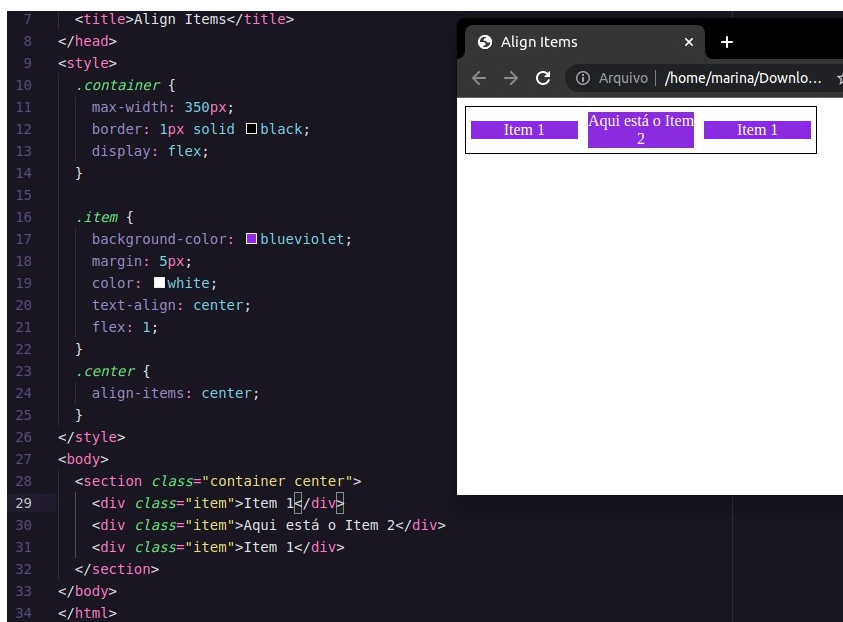
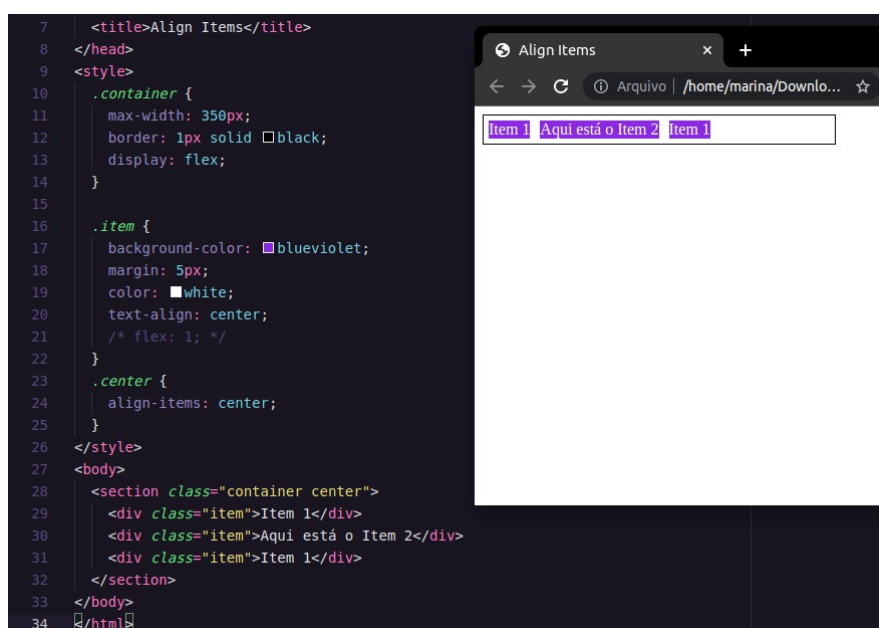
- 1) Center: alinhamento dos itens ao centro;
- 2) Stretch: padrão e os flex itens crescem igualmente;
- 3) Flex-start: alinhamento dos itens no início (de acordo com a orientação do eixo utilizado);
- 4) Flex-end: alinhamento dos itens no final
- 5) Baseline: alinhamento de acordo com a linha base da tipografia dos itens (vai utilizar o eixo da linha relacionada ao texto que está dentro do conteúdo desses itens);

Vamos usar uma propriedade que vamos ver melhor mais a frente que é a **flex: 1**;
Com ela vamos controlar a distribuição do tamanho dos nossos itens.

Usando **center** e **flex: 1**

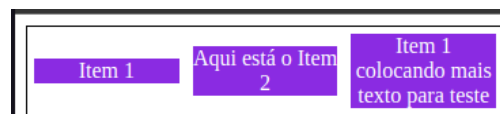
Nesta figura não estamos usando `flex: 1`;

Vamos usar na próxima para ver o que acontece.



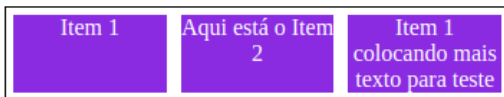
Agora eu tenho meus itens alinhados de acordo com meu eixo de linha e agora sim eles são alinhados ao centro.

Com o `flex: 1` o tamanho cresce de acordo com o conteúdo do texto e tamanho do item! Veja:



stretch

align-items: stretch

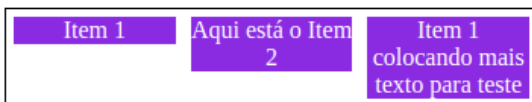


Com o stretch os itens se alongam de acordo com o conteúdo e tamanho do maior item!

Se eu acrescentar valor de texto dentro do item 2, todos os itens acompanharão seu tamanho também!

Flex-start

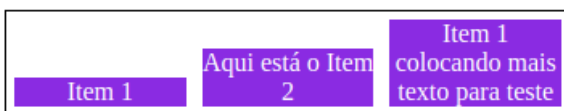
align-items: flex-start



Repare que agora os itens não estão centralizados de acordo com o eixo horizontal (ou da linha). Agora eles começam do topo da div do item.

Flex-end

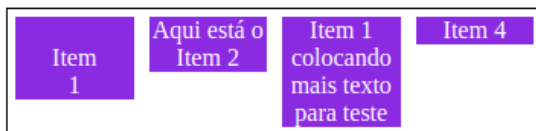
align-items: flex-end



Podemos dizer que o **flex-end** é o oposto do **start**! Os itens não seguem a linha do topo, mas sim do bottom da div dos itens.

Baseline

align-items: baseline

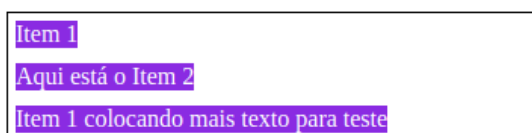


O baseline vai usar a base da linha como referencia dos tamanhos dos itens, cada item respeitará sua baseline.

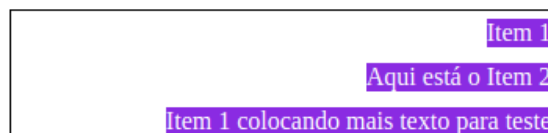
Coloquei um br antes de Item 1 para ele descer, e aí ele ficou com tres linhas. Veja que o a div respeitou esse tamanho e sua orientação foi dada de acordo com sua baseline!

AGORA VAMOS VER TUDO ISSO COM COLUNAS:

align-items: flex-start column



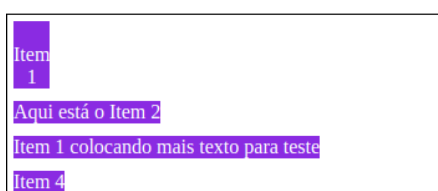
align-items: flex-end column



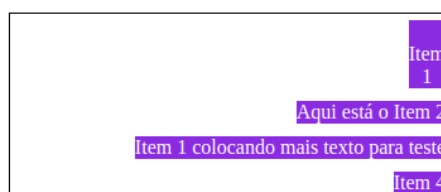
Com o **start** a coluna começa alinhada à esquerda, já com o **end** a coluna alinha à direita.

Vamos acrescentar mais textos diferentes para comparar **start** com **end** e **center**:

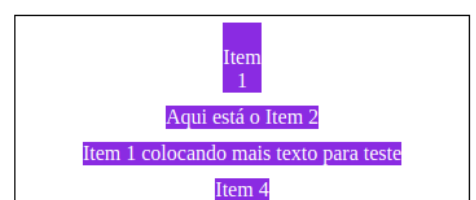
align-items: flex-start column



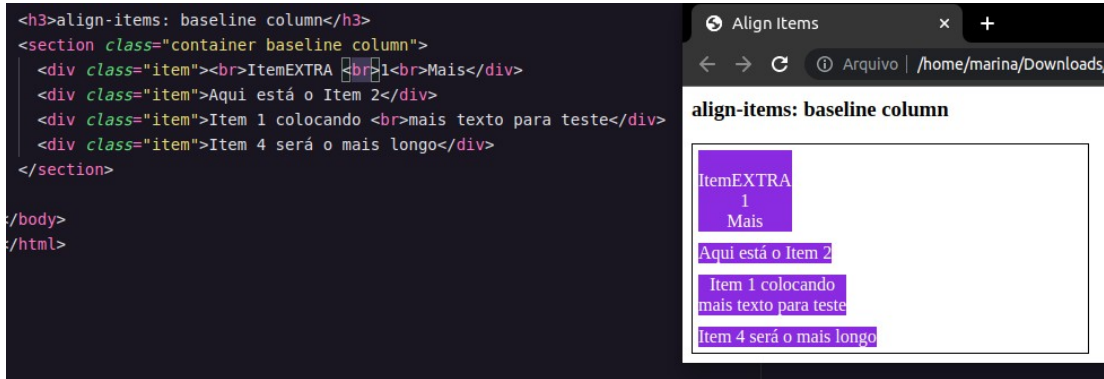
align-items: flex-end column



align-items: center column



baseline



Para ver melhor o comportamento do baseline, coloquei br no meio para quebrar o texto. Basicamente será a mesma coisa que quando usamos **rows**. Sempre respeitará a baseline do texto, o item não extrapola este tamanho.

Testando centralizar itens la tela:

O align-item resolve um tipo de alinhamento, e é mais comumente utilizado para alinhar as coisas no centro do conteúdo, centro de tela, etc.

Então agora vamos ver como colocar esses elementos alinhados ao centro da nossa tela!

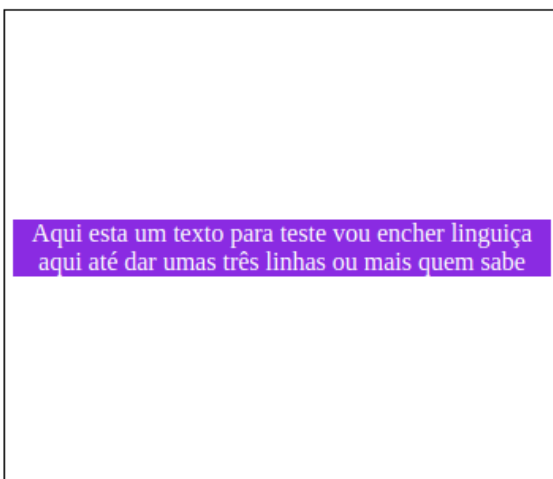
Para isso vamos criar uma nova section com uma div dentro e utilizar nela uma classe chamada **central** contendo as seguintes formatações:

```
.central {
  height: 300px;
  justify-content: center;
  align-items: center;
}
```

```
<h3>Alinhamento central à tela</h3>
<section class="container central">
  <div class="item">Aqui esta um texto para teste
    vou encher linguíça aqui até dar umas três
    linhas ou mais quem sabe</div>
</section>
```

O resultado ainda não foi o esperado, o conteúdo está centralizado, porém somente em relação à section. Quero que ele centralize à tela!

Alinhamento central à tela



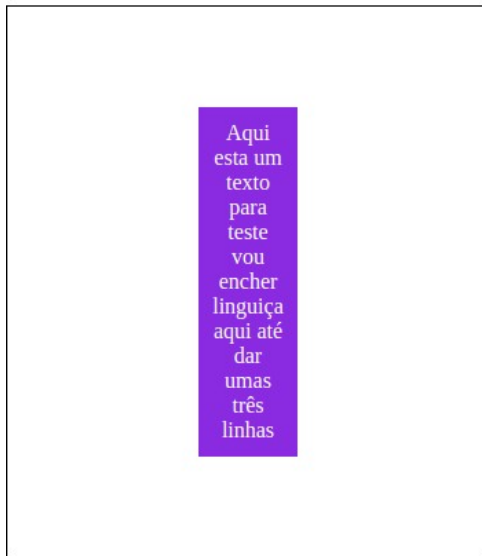
Para isso vamos usar a classe central alcançando os itens também:

```
.central .item {
  flex: 0;
  padding: 20px;
}
```

Vamos zerar o **flex** para remover algumas propriedades relacionadas à flex items (grow, basis e shrink que vamos ver mais a frente) para que esses elementos não cresçam de acordo com o que estamos fazendo atingindo então somente um item em questão que é este ao lado!

Agora sim, meu item vai ficar centralizado também em relação as margens esquerda e direita:

Alinhamento central à tela



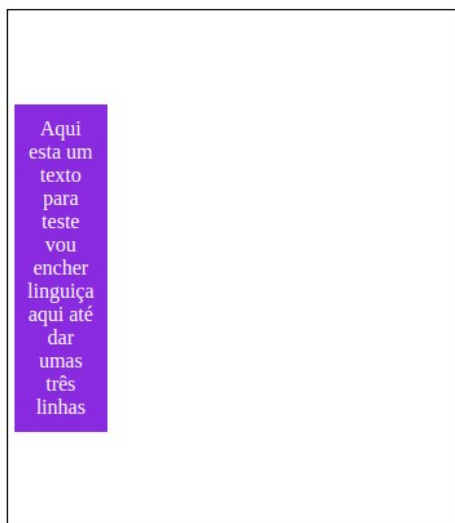
Justify-content: center joga o texto para o centro, dando padding à esquerda e à direita.

Align-items: center faz com que o item alinhe centralmente criando um padding somente acima e abaixo. O item continua à esquerda!

Só para ilustrar! Veja:

Sem justify content:center e **com** align-items: center:

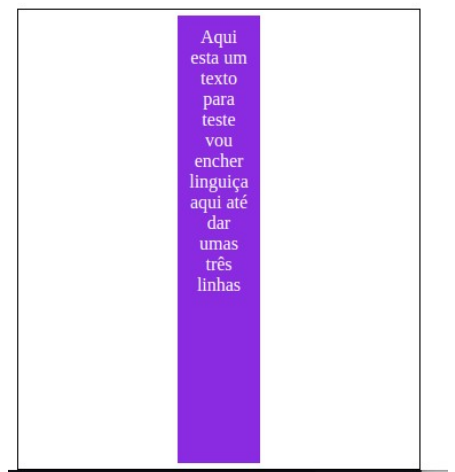
Alinhamento central à tela



Agora **com** justify-content:center e **sem** align-items:center =>

Repare que o align-items não deixa a div com a cor até a base do container, ela termina junto com o texto.

Alinhamento central à tela



7) ALIGN-CONTENT

É a propriedade responsável por tratar o alinhamento das linhas do container em relação ao eixo vertical do container.

E para isso o container deverá respeitar as seguintes orientações:

- 1) O container deverá utilizar quebra de linha (implemente o flex-wrap: wrap)
- 2) A altura do container deverá ser maior que a soma das linhas dos itens (se tenho duas linhas de texto de 20px cada, meu container tem que ser maior que 40px, **senão não funcionará!**)

Tipos de alinhamento:

- 1) Center: alinha tudo ao centro;

- 2) Stretch: é o padrão e os flex itens crescem igualmente (o maior elemento é quem dita o tamanho dos outros elementos, para todos ficarem iguais);
- 3) Flex-start: alinhamento dos itens no início (de acordo com a orientação do eixo utilizado);
- 4) Flex-end: alinhamento dos itens no final;
- 5) space-between: espaçamento igual entre os elementos;
- 6) space-around: aquele que cria um “padding” nas bordas (sendo a metade do espaçamento do meio)

São conceitos meio misturados do que já vimos anteriormente (do justify-content e do align-items).

Então vamos colocar a estilização dos nossos itens e do container principal:

```
.container {
  height: 400px;
  max-width: 300px;
  margin: 0 auto;
  border: 1px solid black;
  display: flex;
  flex-wrap: wrap;
}

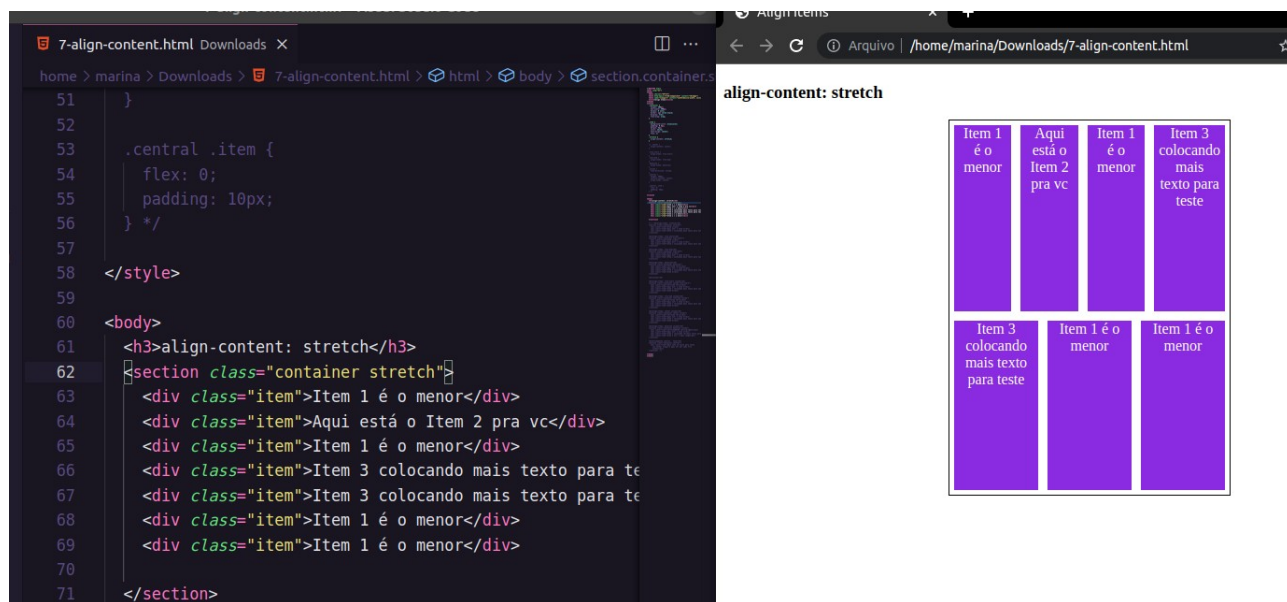
.item {
  background-color: blueviolet;
  padding: 0 5px;
  /* margin: 5px; */
  color: white;
  text-align: center;
  flex: 1;
}
```

Não esquecer de precisamos do **wrap** e de uma **height** que caiba todos os itens dentro! Senão o align-content não funcionará!!

margin: 0 auto; é para centralizar meu conteúdo no meio da minha página!!! (Karen removeu depois)

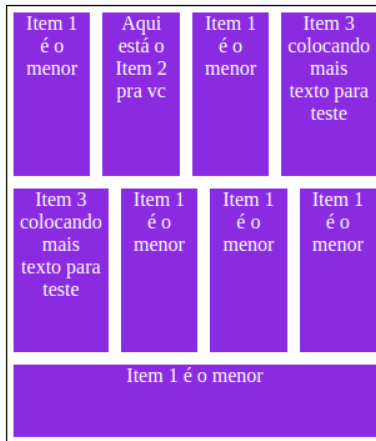
Nos nossos itens vamos usar novamente o **flex: 1**, mais pra frente vamos entender melhor.

(comentei margin, mas vamos usar! Senão os itens ficam grudados!)



Com **stretch** ou sem, o comportamento é o mesmo, é o padrão. Ele estica o conteúdo dos outros de acordo com o maior da linha. E dessa vez, ele irá distribuir por todo o container.

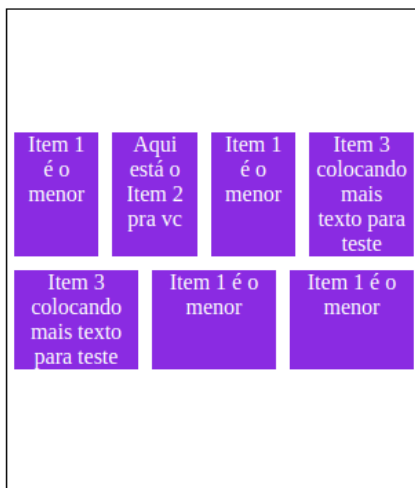
align-content: stretch



Ele tem um comportamento meio doido, se eu adicionar mais itens ele fica assim.

CENTER

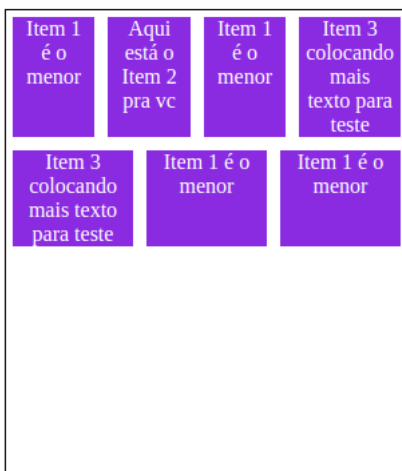
align-content: center



Com o center teremos o conteúdo distribuído ao centro do container, dando um padding no topo e no bottom.

FLEX-START

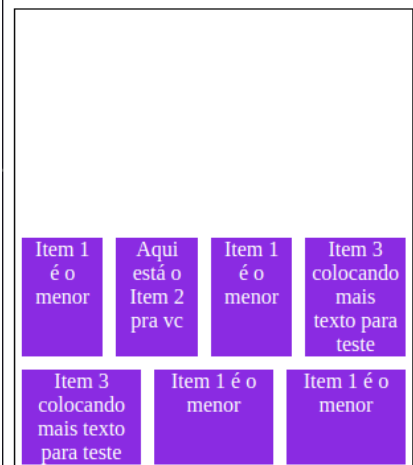
align-content: flex-start



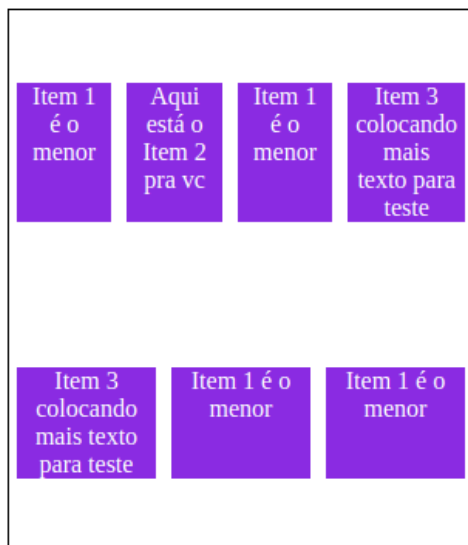
Assim como anteriormente, vimos que o start, coloca tudo no topo.

Já o flex-end é tipo o oposto do start. Ele joga os itens pro bottom da div=>

align-content: flex-end



align-content: space-around



O space-around vai dar aquele “padding” no topo e no bottom da div, sendo essa sua diferença para o próximo item que é o space-between:

align-content: space-between

