CSS Flexbox - By: ORIGAMID

Grade 01 - Conteúdo: Introdução

- CSS Flexbox - Introdução.

Grade 02 - Conteúdos: Flex Container

- Display Flex.
- Flex Direction.
- Flex Wrap.
- Flex Flow.
- Justify Content.
- Align Items.
- Align Content.

Grade 03 - Conteúdos: Flex Item

- Flex Grow.
- Flex Basis.
- Flex Shrink.
- Flex.
- Order.
- Align Self.

Grade 01 - Conteúdo: Introdução

No flexbox existem 2 grandes grupos: **Flex Container** e **Flex Item**, esse segundo diz respeito aos elementos que estão dentro do container e como os filhos irão se comportar, respectivamente.

Com o **Flex Container** temos: **Display**, **Flex-direction**, **Flex-wrap**, **Flex-flow**, **Justify-content**, **Align-items** e **Align-content**.

Grade 02 - Conteúdos: Flex Container

- **Display: Flex**

Quando usamos essa possibilidade, o elemento pai vira um flex container e os elementos filhos dentro dele vira um flex items. Relacionado ao api temos o uso de algumas propriedades para formular bem o que queremos apresentar.

Podemos usar display:flex para qualquer tipo de tag.

A partir do momento que o usamos, os elementos dentro do container (que recebe o atributo display: flex) ficam inline, isto é, o tamanho do conteúdo influencia o espaço que o filho ocupa dentro do container. E por vezes, eles podem acabar ultrapassando os limites do pai, ou seja, eles ficam com a aparência de "vazar" do container.

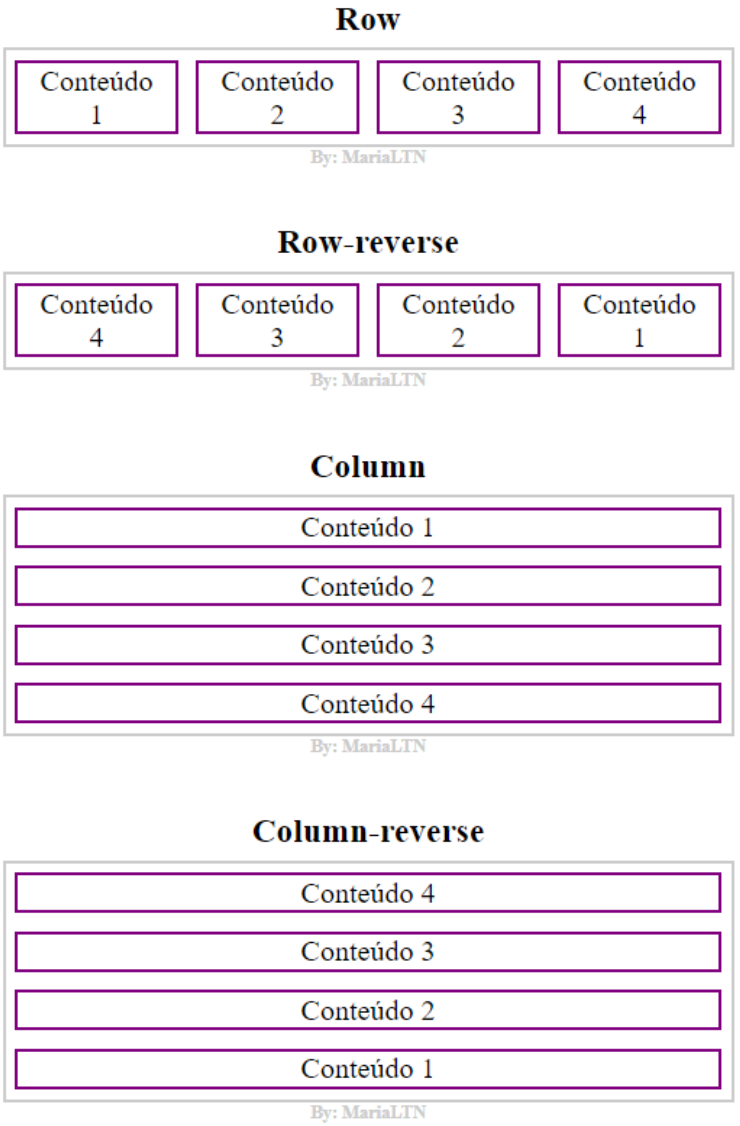
E para corrigir isso podemos usar as propriedades do flex-container como o flex-wrap: wrap (Ou seja, com esse atributo falamos: "Apesar do seu tamanho interno, se não couber dentro da linha, vá para a linha de baixo").

Usamos duas propriedades importantes, o **display: flex** e o **flex-wrap: wrap**. Eles pertencem ao "pai" porque o pai dita como será o comportamento dos elementos dentro dele e que será inicialmente aplicada a todos os filhos (sem distinção até que o desenvolvedor faça algo diferente).

Usamos muito o display: flex quando queremos trabalhar com **mobile**.

- **Flex-direction: row** (padrão)**, row-reverse, column ou column-reverse.**

Quando aplicamos o display: flex, por padrão o flex-direction: row (Esquerda para a direita) é acionado. Significa que os elementos ficam um ao lado do outro. As outras possibilidades são muito usadas quando queremos trabalhar com responsividade. E dizem respeito à direção que os itens serão apresentados, na horizontal ou na vertical. Quando os escolhemos, estamos estabelecendo o eixo principal do container, definindo a direção que os flex items serão colocados na flex container.



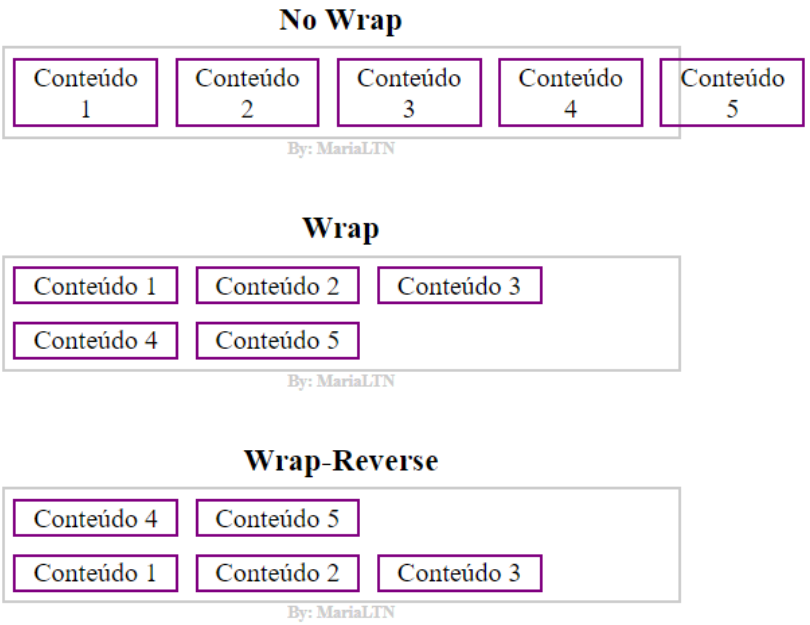
- **Flex-wrap: nowrap** (padrão)**, wrap ou wrap-reverse.**

Usamos o flex-wrap mais acima para explicar sobre a quebra de linha baseado no tamanho de um elemento.

Por padrão usamos o **nowrap**, não é quebrado a linha, por isso vemos elementos excederem o tamanho imaginado (Vazando), com o **wrap** podemos quebrar a linha

e a mesma situação ocorre quando usamos o **wrap-reverse** com a diferença que alteramos a ordem dos elementos que serão apresentados, no qual as linhas que estão “completas” no tamanho, passam para a linha debaixo.

O mais usado é o **flex-wrap: wrap**.



- **Flex-flow: row nowrap, row wrap ou column nowrap.**

É considerado um **atalho** que junta algumas propriedades do flex-direction com o flex-wrap. Embora não seja muito usado, pois quando mudamos o flex-direction para column, mantemos o padrão do flex-wrap que é nowrap.

Formado por flex-direction e por flex-wrap. Como o exemplo: flex-flow: row wrap. Dizendo que estará disposto em linha e que quebrará a linha caso o tamanho exceda o tamanho disponível.

Para ser mais útil é preciso algumas alterações em algumas outras propriedades, o que pode levar um pouco de tempo. Por isso, na dúvida usa os padrões originais.

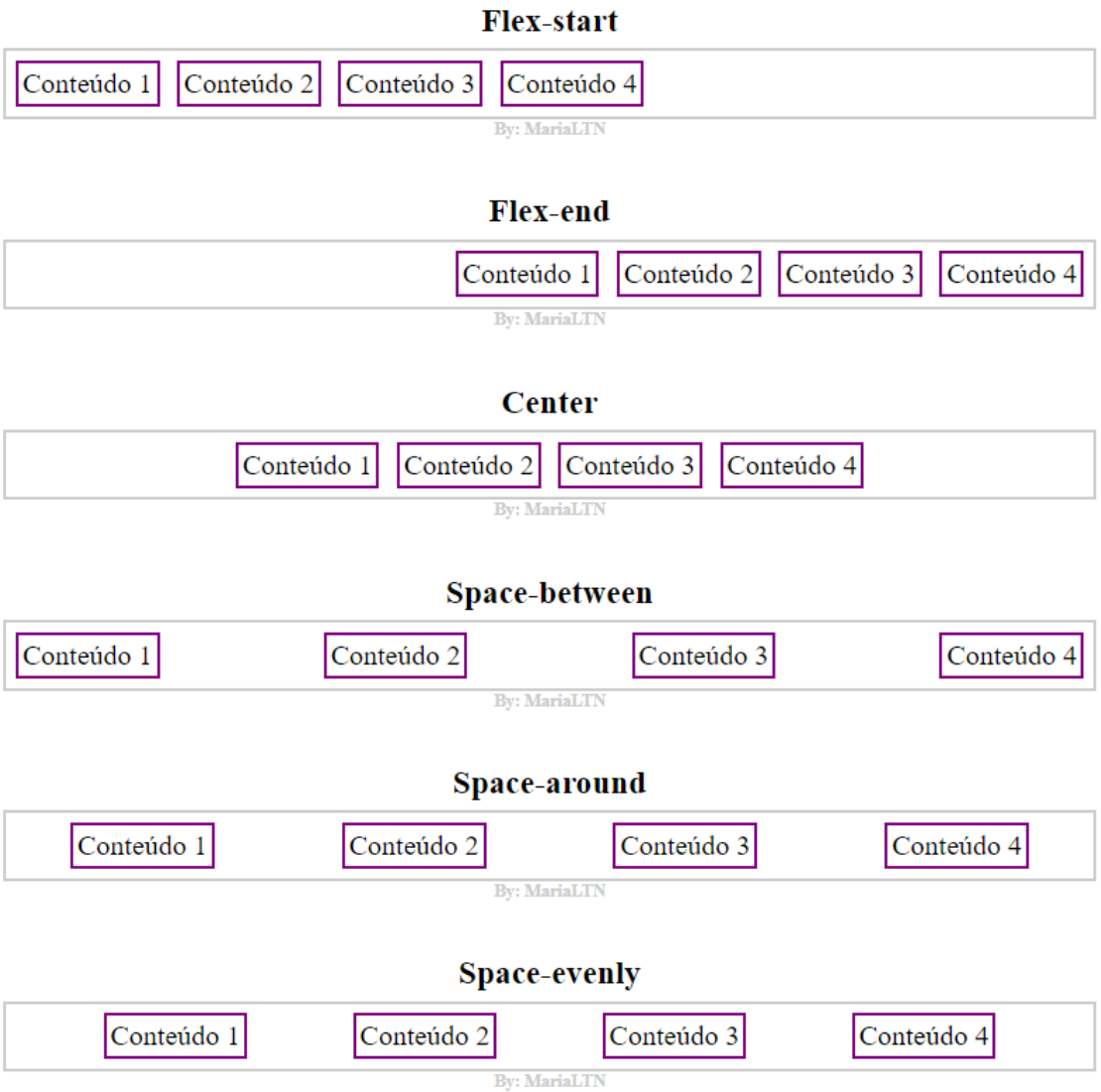
- **Justify-content: flex-start, flex-end, center, space-between, space-around ou space-evenly.**

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

Mas as propriedades só funcionam se os itens não ocuparem todo o container.

Podemos usar essa propriedade com o flex-direction, os exemplos foram explicadas com flex-direction em *row*, mas podem usar outras possibilidades, adaptando-as.

A maior diferença entre os “spaces” seria que o **space-between** (Os itens são distribuídos uniformemente dentro do container, sendo o primeiro e último elemento alinhado com as extremidades) O **space-around** (O espaço vazio antes do primeiro e depois do último item é igual a metade do espaço entre cada par de itens adjacentes). E por fim o mais “novo”, temos o **space-evenly** (Espaçamento é exatamente o mesmo do primeiro ao último elemento).



- **Align-items: stretch** (padrão), **flex-start, flex-end, center ou baseline.**

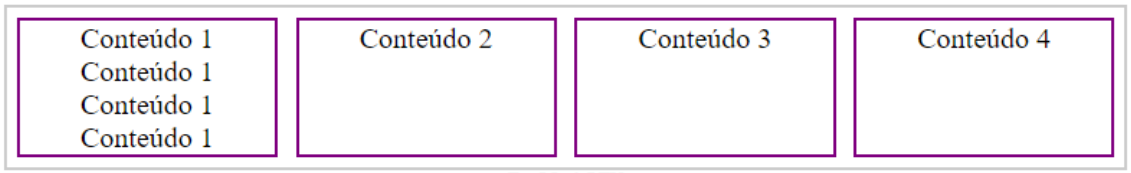
Trata-se do alinhamento dos flex-itens de acordo com o eixo do container em *Cross Axis*, sendo diferente para linhas e colunas.

Essa propriedade permite o alinhamento central no eixo vertical.

Podendo muitas vezes parecer confuso com o Justify-content, Align-items, Align-content e Align-self.

Veremos abaixo como cada opção é visualizada. (Ele por padrão não quebra, logo flex-wrap: nowrap), mas obviamente pode-se alterar isso. (O que dá um layout bem interessante)

Stretch



By: MariaLTN

Flex-start

By: MariaLTN

Flex-end

By: MariaLTN

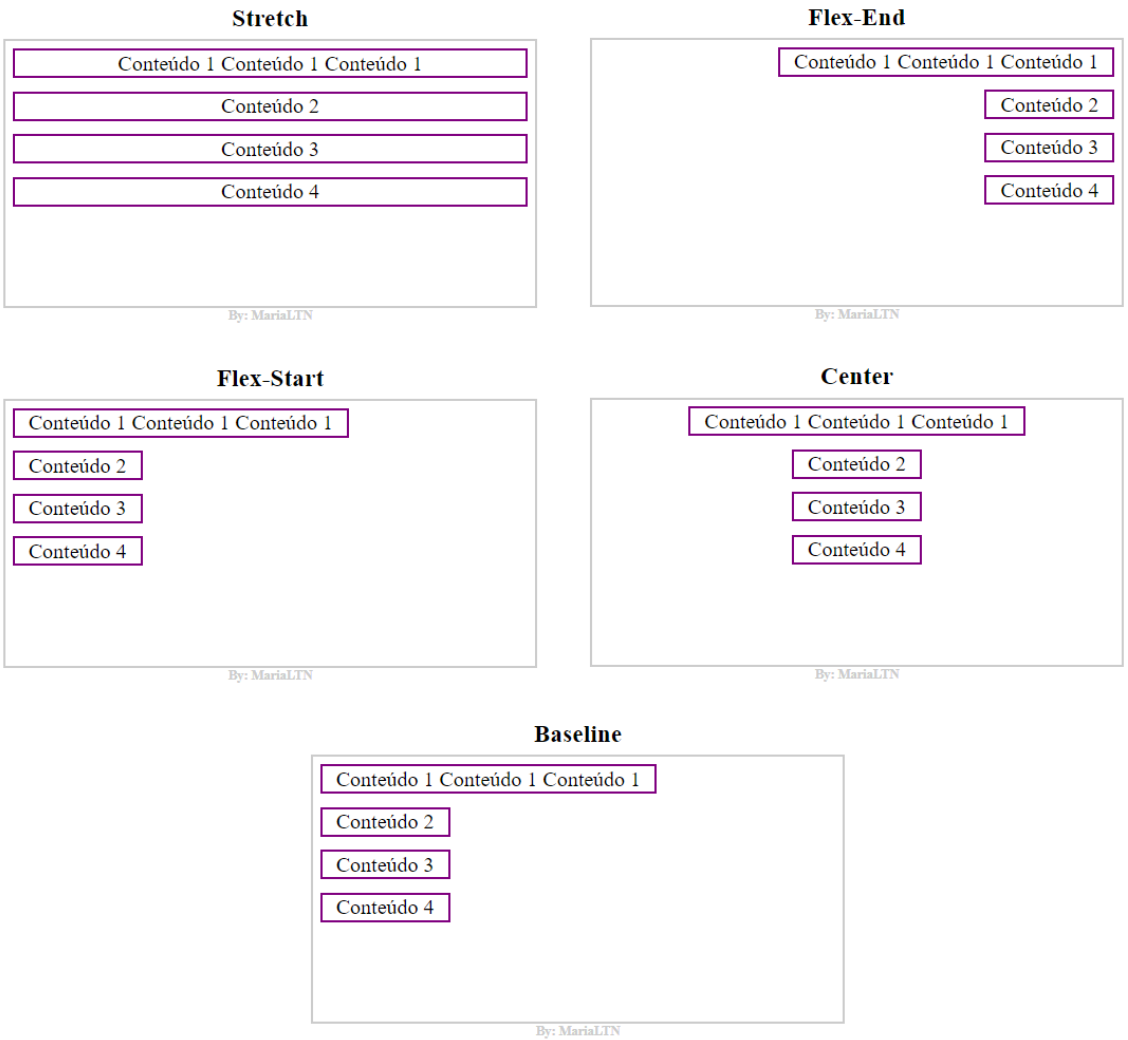
Center

By: MariaLTN

Baseline

By: MariaLTN

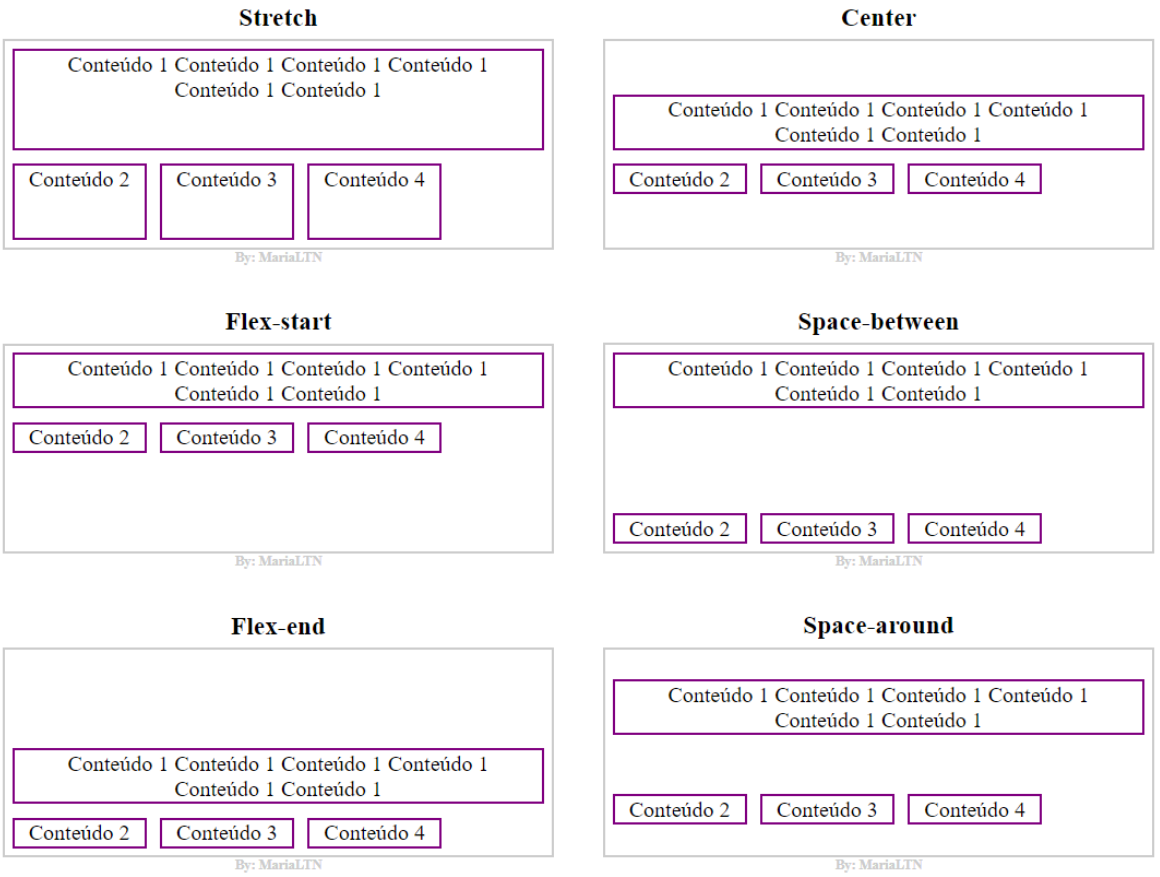
No caso acima, foi aplicado o flex-direction: row. Abaixo teremos o mesmo exemplo com flex-direction: column.



- **Align-content: stretch** (padrão)**, flex-start, flex-end, center, space-between ou space-around.**

Alinha-se às linhas do container de relação ao **eixo vertical**. A propriedade só funciona se existir mais de uma linha flex-itens, sendo obrigatório nesse caso o flex-wrap ser wrap.

Além disso, o efeito dela apenas será visível caso o container seja maior que a soma das linhas dos itens. Isso significa que se você definir **height** para o container. As imagens abaixo não foram utilizadas flex: 1 nos itens. Caso fosse usado, todo o espaçamento interno do elemento seria utilizado.



Todas as propriedades vistas até então dizem respeito a **Flex Container**, ou seja, atributos que são aplicáveis ao pai (tag que receberá o “display:flex”). Veremos agora as propriedades aplicáveis aos itens em si, chamados de **Flex Item**.

Grade 03 - Conteúdos: Flex Item

Os atributos que podemos ter no **Flex Item** são: **Flex-grow**, **Flex-basis**, **Flex-shrink**, **Flex**, **Order** e **Align-self**.

É possível que um Flex-item seja também um Flex-container, basta definir display: flex nele. Assim, os filhos desses itens também serão flex itens.

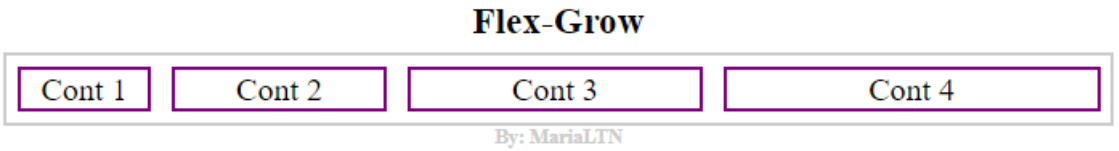
- **Flex-grow: número ou 0** (padrão).

Define a habilidade de um flex item **crescer**. Ou seja, define a proporcionalidade de crescimento dos itens, respeitando o tamanho de seus conteúdos internos. Por padrão ele é zero, logo, o comportamento dele é semelhante, senão, idêntico a um

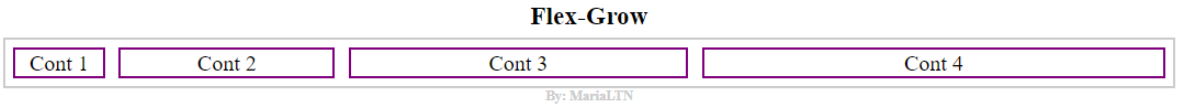
item **inline**, no qual o seu comprimento é definido pelo tamanho de seu conteúdo interno.

Caso o desenvolvedor queira colocar um valor diferente, ele estará indicando qual elemento (item) terá um tamanho maior.

Por exemplo, temos 4 itens e colocamos cada um deles com valores de flex-grow variando de 0 a 3. Teríamos uma representação da seguinte maneira:



Para a mudança ser mais drástica vamos colocar os valores de flex-grow 0, 5, 10, 15 respectivamente nas classes itens.



Observação: Precisei aumentar o width do container de 400px para 850px para ficar mais visível a diferença.

Vale notar que o espaçamento que estamos mencionando é espaço nas laterais de um item. O que pode ser visto na figura abaixo:

Observação: Justify-content não funciona em itens com flex-grow definido.

- **Flex-basis: auto** (padrão)**, unidade** (Pode ser em %, em, px e outros) **ou 0.**

Indicamos o tamanho inicial do flex item antes da distribuição do espaço restante, ou seja, o tamanho mínimo esperado por esse item e é muito usada com o flex-grow. Queremos dizer que estamos mexendo justamente no espaço (em laranja, na figura acima) de um flex item. Para isso o flex-grow precisa ser no mínimo igual a 1..

Quando temos a configuração básica de flex-grow igual a 1 e flex-basis igual a auto. Teremos a largura da base igual a do item, se o item não tiver tamanho

especificado, o tamanho será de acordo com o conteúdo. Em outras palavras, a parte laranja (esquerda, da figura acima) do início do item até o começo da palavra seria do mesmo comprimento.

O **flex-basis: auto**, não quer que cada flex-item tenha o mesmo comprimento (um tamanho exato), ele não influencia o tamanho do flex-item. Mas sim, o espaçamento que vem antes e depois do conteúdo interno do item. Levando em consideração o tamanho do flex-grow.

Quando temos o **flex-basis: 0** (O mais utilizado), ele fica preocupado com o tamanho dos itens, isto é, ele deseja que todos os itens tenham tamanhos iguais. Mas caso, um desses elementos seja maior, ele tentará que os outros fiquem do mesmo tamanho. Tendo em vista que ele sempre respeita o tamanho do conteúdo interno.

Por fim, no **flex-basis: número** (por exemplo: 110px). O seu uso pode muitas vezes ser entendido como um sinônimo de max-width.

- **Flex-shrink: 1** (padrão)**, 0 ou número.**

Define a capacidade de redução/compressão de tamanho de um flex-item. Podendo ser o **padrão**: Permitindo que os itens tenham os seus tamanhos (seja o tamanho definido a partir de width ou flex-basis) reduzidos para **caber no container**. Ou então igual a **zero**, que não permite a redução. Ou então um **número** (de vezes) que será reduzido.

- **Flex: 0 1 auto** (padrão)**,1, 2 ou 3 2 300px.**

Considerado um **atalho** para as propriedades flex-grow, flex-shrink e flex-basis, é normalmente mais utilizado e recomendado pela melhor consistência entre os browsers.

Como ele é um atalho entre três outras propriedades, ele segue a ordem: flex-**grow**, flex-**shrink** e flex-**basis**.

Como mostrado anteriormente, podemos simplesmente usar flex: 1 (sendo: grow: 1, shrink: 1 e basis: 0), flex: 2 (sendo: grow: 2, shrink:1 e basis: 0) ou as “atributos prontos” do padrão 0 1 auto ou então 3 2 300px. (Em minha humilde opinião, ainda não consegui diferenciar bem o uso especifico do 3 2 300px em aplicações.)

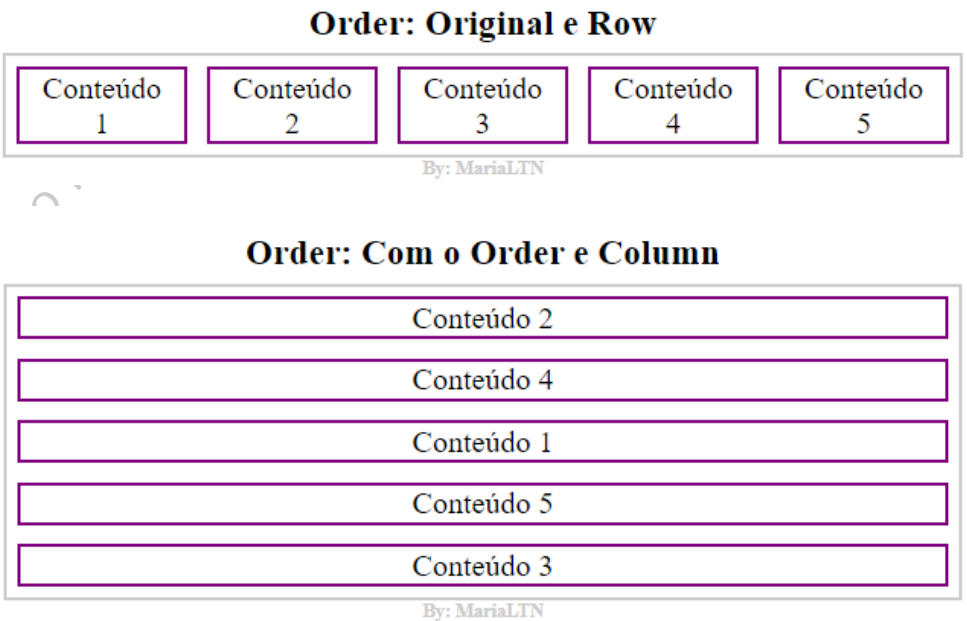
Na figura abaixo temos o exemplo utilizando o Flex:1 e uma mistura de Flex: 1 e Flex: 2 e utilizando o padrão.

- **Order: número ou 0** (padrão).

O mais fácil, ele é responsável por modificar a ordem/ordenação dos flex-itens. Podemos aplicar um **número negativo**, respeitando a ordem: será sempre do menor para o maior, então, teremos a ordem:1 aparecer na frente de ordem:5.

E quando temos a mesma ordem sendo aplicada a vários elementos ele respeita a ordem do HTML (na ordem que aparece no HTML, ficará no resultado final).

No exemplo abaixo temos o resultado com o uso do order, aplicada a uma linhae uma coluna.

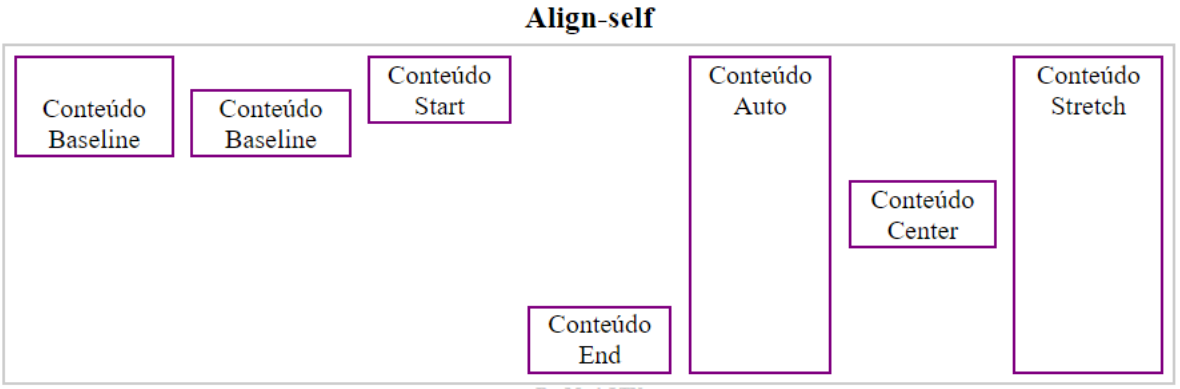


- **Align-self: auto** (padrão), **flex-start, flex-end, center, baseline, stretch.**

E para encerrar a apostila temos o Align-self, que serve para definirmos o alinhamento específico de **um único flex-item**.

Observação: Caso um valor seja atribuído, ele passará por cima do que for atribuído no align-items do container.

Observação: Vale lembrar que o alinhamento acontece tanto em linhas quanto em colunas. Por exemplo: O flex-start quando os itens estão em linhas, alinha o item ao topo da linha. Quando está em colunas, alinha-se o item ao início (à esquerda) da coluna.



Considerações Finais: https://origamid.com/projetos/flexbox-guia-completo/ e DIO Curso de Flexbox