Parte 2 – Propriedades relacionadas aos flex-items

1) FLEX-GROW

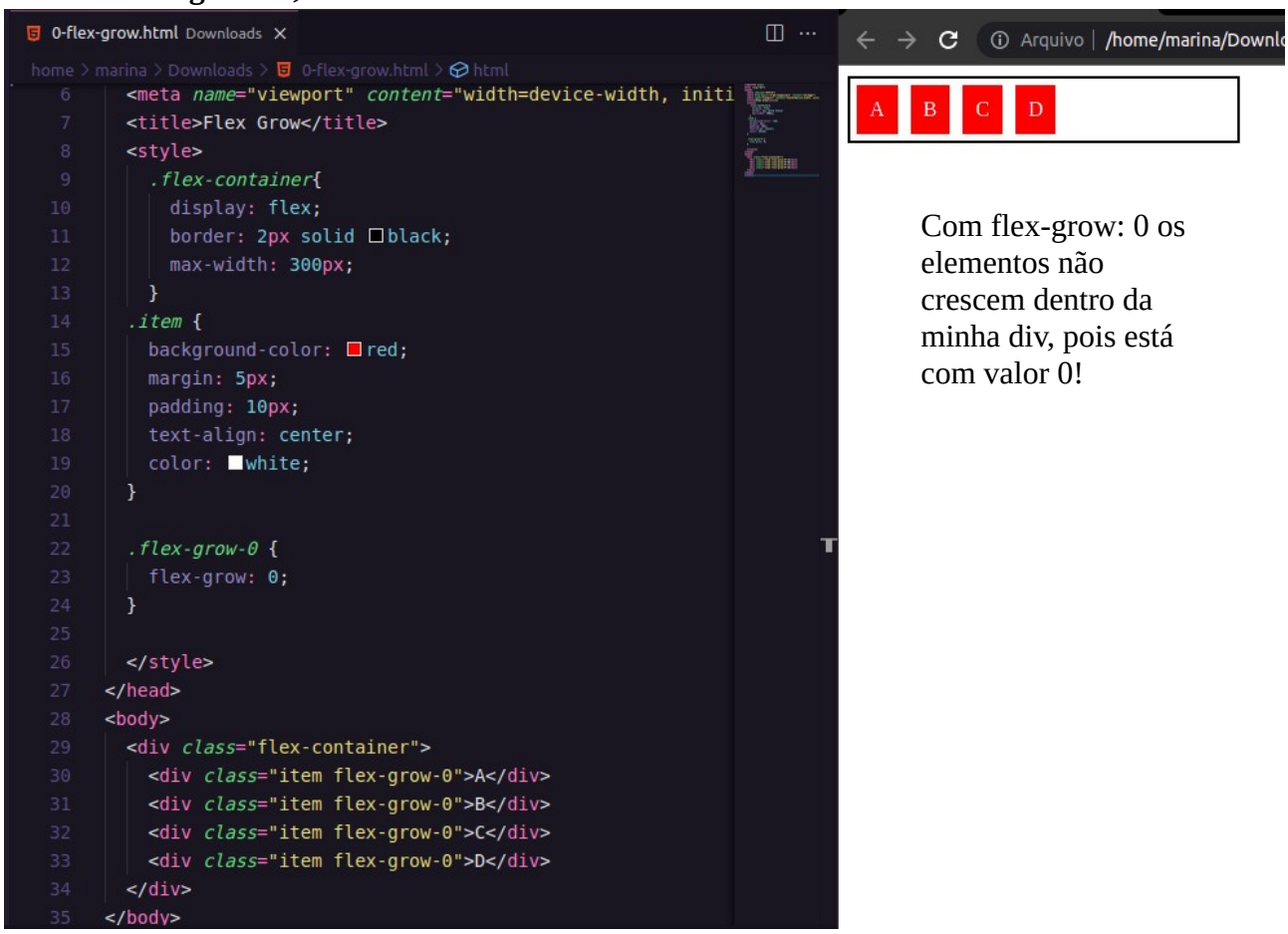
Define a proporcionalidade de crescimento dos itens, respeitando o tamanho de seus conteúdos internos.
Ou seja, não vai conseguir fazer uma compressão ou expansão que faça com que os itens e seu conteúdo seja desrespeitado sendo comprimidos, vazados ou quebrados.

Trabalhando com flex-grow, os valores são números, ou ele vai ser 0 ou algum número relacionado à proporção de crescimento desejado. Ex: **flex-grow: 1**

Por padrão o flex-grow é 0.

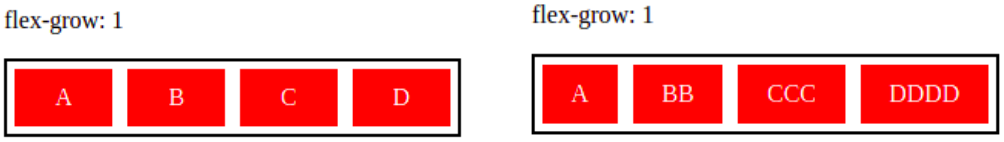
OBS: não funciona quando usamos justify-content container!!!

Usando **flex-grow: 0;**



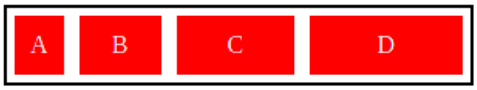
Com flex-grow: 0 os elementos não crescem dentro da minha div, pois está com valor 0!

Se colocarmos **flex-grow: 1**, o item irá crescer dentro da div de forma proporcional ao seu conteúdo e também respeitando o tamanho da div container:



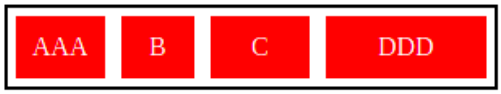
Brincando com os valores do flex-grow para ver seu comportamento:

flex-grow: A:0 B:1 C:2 D:3



o 1 cresce uma vez mais, o 2 cresce duas vezes, e o 3 três vezes.

flex-grow: A:0 B:1 C:2 D:3



E se eu aumentar o conteúdo do item colocando mais texto, o item cresce respeitando seu conteúdo.
Perceba que ele mantém a proporcionalidade do crescimento dos itens. O DDD continua tendo o maior “padding” de todos os itens!

flex-grow: A:0 B:1 C:2 D:3

Colocando o mesmo conteúdo com diferentes flex-grow dá pra ver que o D está maior que os outros (e assim por diante).

flex-grow: 1

Se eu colocar o mesmo conteúdo dentro dos meus itens com grow: 1 e 2 eles tem o mesmo comportamento.

flex-grow: 2

O “padding” é o mesmo. A distribuição dos valores pros tamanhos dos itens foi a mesma.

2) FLEX-BASIS

É a propriedade que estabelece o tamanho inicial do item (tamanho base ou mínimo do item) antes das distribuições de espaço dentro dele, usando como base o conteúdo interno disposto.

É como se essa propriedade pegasse o valor mínimo que o item vai precisar e depois calculasse a distribuição do espaço antes e após o conteúdo.

Seu uso depende também da sua relação com o flex-grow, um influencia no outro.

Valores possíveis:
auto: caso o item não tenha tamanho pré definido inicialmente ele será proporcional ao conteúdo. E caso ele tenha um flex-grow definido ele vai tentar adaptar esses itens tomando o tamanho mínimo sempre igual, ocupando o tamanho total do container.
Px, %, em, etc: são valores exatos previamente definidos, eles serão valores para tamanhos mínimos que os itens vão ter. Caso o conteúdo desse item seja muito maior que o valor definido, ele

vai crescer ao ponto de não ter espaço podendo vazar do item, não teríamos então o espaçamento proporcional antes e depois do conteúdo do meu item (aquele “padding”).
0 (zero): terá relação direta com a definição do flex-grow, já que um influencia no outro.

O flex-basis é uma propriedade que utilizamos quando há a possibilidade de usar o flex-grow para "esticarmos" o elemento. Ele servirá, como o próprio nome diz, como uma base para eles. Se não alterar o valor padrão do flex-basis (auto), o valor usado como base será a largura e altura do próprio elemento. Caso passemos 0, ele terá uma base em branco, então não aparecerá nada. Aí, se passarmos um flex-grow, ele "esticará" o elemento.

O basis: auto é o padrão, ele faz com que a largura da base seja igual a do item. Se o item não tiver tamanho especificado, o tamanho será de acordo com o conteúdo.

Utilizando flex-basis: auto, nada acontece se você não coloca o flex-grow. Pois ele funciona de acordo com o grow, então se o grow estiver 0 nada acontece! O item fica com o tamanho exato do seu conteúdo (sem “padding”).

Auto sempre segue o grow!

flex-grow: 0 flex-basis: auto

flex-grow: 1 flex-basis: auto

Agora compare utilizando **auto e 0:**

flex-grow: 1 flex-basis: auto

flex-grow: 1 flex-basis: 0

o Auto crescia de acordo com o conteúdo do item, sempre mantendo o “padding” fixo (criado pelo grow 1);

Agora usando basis 0 os itens terão o mesmo valor de largura, independente do seu conteúdo!
OBS: claro que se alterar o texto para AAAAAAA este item irá crescer mais que os outros, porém o comportamento é de sempre tentar fazer eles ficarem com tamanhos iguais.

Agora veja a diferença entre grow 0 e 1, utilizando basis: 100px:

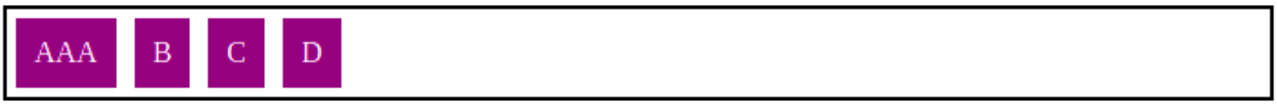
flex-grow: 0 flex-basis: 100px

flex-grow: 1 flex-basis: 100px

Com o basis 100px os itens recebem um “padding” fixo, mas o grow está 0, então os itens não vão crescer dentro do container.
Já com basis 100px e grow 1, os itens vão crescer ocupando meu container.

Compare grow 0 com basis 100px:

flex-grow: 0 flex-basis: 0



flex-grow: 0 flex-basis: 100px

Perceba que os itens não tem o “padding” de 100px e seus tamanhos obedecem seu conteúdo na primeira opção.
Já na segunda, ao acrescentar o basis 100px eles aumentam de tamanho, criando esse “padding” de 100px dentro do seu conteúdo.

3) FLEX SHRINK

Essa propriedade estabelece a capacidade de redução ou compressão do tamanho de um item.

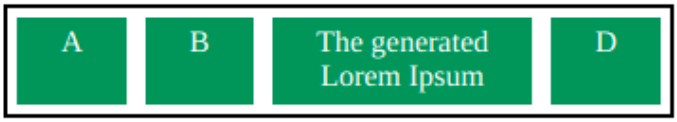
O **basis** estabelecia o tamanho mínimo para um item, e trabalhando com grow conseguimos fazer esses itens crescerem. Agora teremos o funcionamento inverso, teremos o tamanho de um item, com comportamentos esperados, por ex: shrink: 1, permite que os itens tenham seus tamanhos reduzidos proporcionalmente ou caso shrink: 0 não permite a redução desses itens.

Posso definir que um item pode reduzir até 2 vezes o tamanho com **flex-shrink: 2;**
Este calculo da redução é feito com base nesse número definido com shrink em relação a proporção dos itens de um container.
Um item com shrink: 3 diminuirá 3 vezes mais que um item com 1.

O valor padrão é **flex-shrink: 1;**

flex-grow: 1 flex-shrink: 1

flex-grow: 1 flex-shrink: 1



Em um design responsivo, percebemos que os itens com shrink: 1 serão reduzidos proporcionalmente, respeitando o “padding” deles.

Quando o conteudo extrapola e gera duas linhas esse padding também é respeitado e os tamanhos são distribuidos proporcionalmente.

Se a gente usar **shrink: 0** vamos ver que ele não permitirá a redução do tamanho. Vamos usar a propriedade **flex-basis** com algum valor junto a ele para ver o resultado:

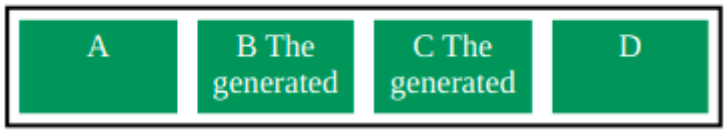
flex-grow: 1 flex-shrink: 0

flex-grow: 1 flex-shrink: 0 basis: 100px

Ao colocar basis 100px dizemos que seu tamanho base minimo é 100px, ou seja, não pode reduzir menos que isso e dessa forma ele extrapola o tamanho do container!

Se eu colocar shrink: 1 aí sim eu teria meus itens comprimidos para tentar caber no container:

flex-grow: 1 flex-shrink: 1 basis: 100px



Com shrink 1 eu ignoro o basis 100px e faço a compressão de forma proporcional pois todos estão com shrink 1 então eles ficam iguais.

flex-grow: 1 flex-shrink: 1, 2, 2, 3

flex-grow: 1 flex-shrink: 1, 2, 2, 3 basis: 100px

Usando vários tipos de shrink eu consigo definir quantas vezes eu quero que um item seja reduzido.

Perceba que o “D” foi o mais reduzido pois coloquei shrink: 3.
Ele está tentando respeitar 100px mas foi necessário reduzir para caber.
O item que menos reduz é o A, pois tem shrink: 1;

4) FLEX

É um atalho ou abreviação de escrita para as propriedades **grow**, **shrink** e **basis**.
Ex: **flex: 1 0 auto** => significa que estou definindo o **grow: 1**, **shrink: 0** e **basis: auto**

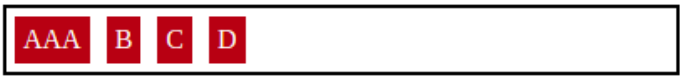
Definindo **flex: 1;** é a mesma coisa que usar **grow: 1**, **shrink: 1**, **basis: 0**;

Geralmente vemos mais a utilização do **flex** do que cada uma das propriedades individualmente.

Para melhor consistência entre os browsers, é recomendado utilizar a propriedade flex ao invés de cada propriedade separada.

Se não definirmos nenhum valor de flex ou para as outras propriedades separadas, o **padrão** será: **flex-grow: 0, flex-shrink: 1 e flex-basis: auto.**

sem css



flex:1 => grow:1, shrink:1, basis:0

O conteúdo respeita o tamanho do container
Cresceram igualmente (por causa do grow:1)
Os itens irão reduzir igualmente também com a proporção de shrink: 1 (caso eu aumente o conteúdo de um dos itens todos vão reduzir igualmente)
E por causa do basis: 0, não permito meu item reduzir a ponto de quebrar linha. (caso aumente o conteúdo de um deles, ele irá se estender dentro do container sem quebrar em linha)

flex:1 => grow:1, shrink:1, basis:0

B e C com flex:2 => grow:2, shrink:1, basis:0

Agora só B e C possuem flex:2, os outros continuam com flex:1.

Veja que os itens B e C crescem duas vezes mais.
Eles vão continuar com shrink:1 (encolhendo proporcionalmente a 1) e com basis:0 não deixando seu conteúdo sofrer quebra de linha.

Lembrar sempre que o **basis:0** ignora o **width** dos itens!

flex:1 => grow:1, shrink:1, basis:0

flex:1 => grow:1, shrink:1, basis:0
com width: 200px => IGNORADO pelo basis

Perceba que está exatamente a mesma coisa do de cima, que não tem width.

Ou seja, não adianta colocar width usando basis: 0;

Ele só vai respeitar width se usarmos **min-width!!**
Veja abaixo:

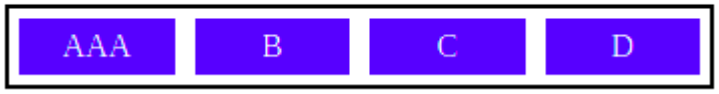
Usando **min-width** para o Basis não ignorá-lo:

flex:1 => grow:1, shrink:1, basis:0
com min-width: 200px

5) ORDER

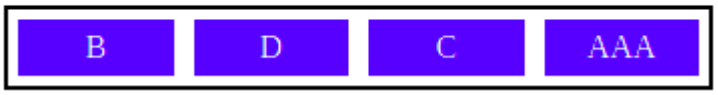
Por padrão **order** é zero! E assim ele exibirá os itens na ordem que estão no HTML.
Se eu não coloco nada de order, então já fica subentendido que meu order é zero.
Se eu colocar **order: 0** ou colocar nada o resultado é o mesmo!

order: 0



Com order:0 nada acontece na ordem dos itens.

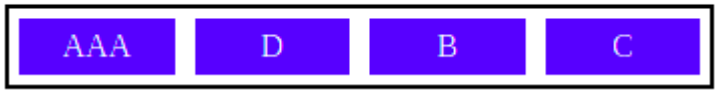
B, D order:1, C order:2, A order:3



Já com variados **order** eu consigo ordenar os itens da forma que eu quiser.
Com order:1 coloquei B e D primeiro
Order:2 coloquei o C logo após
E por último com order:3 coloquei o A

Também é possível trabalhar com valores **negativos** para order!
Sendo assim, quanto menor o número negativo, maior a prioridade em relação à sua ordem.

A order:-2, D order: -1, B order:1, C order:2



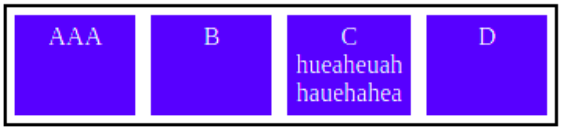
6) ALIGN-SELF

Estabelece o alinhamento individual de um determinado item.
Usamos alignt-items para fazer essa definição diretamente no container,mas agora vamos fazer isso de forma individual, manipulando apenas um item e tornando-o diferente do que está disposto no padrão.
Para o **align-self** funcionar, não podemos definir o **align-items** em nosso container!

Valores possíveis:
auto: é o valor padrão, ele respeita a definição de align-items do container (se tiver um align-items: center ou flex-start ele vai respeitar isso);
flex-start: ao início do container
flex-end: ao final do container
center: relativo ao centro de acordo com o eixo estabelecido.
Stretch: ocupa todo o espaço tendo sua proporção definida igualmente
baseline: utiliza a linha base da tipografia
Obs: end e start também dependem do eixo que definimos (se é column ou row)

Usando o **auto**, vamos perceber que ele irá obedecer oq estiver definido para o **container**:

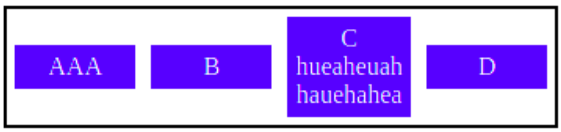
align-self: auto



Se colocarmos **align-items: center** no container, o **align-self: auto** dos nossos itens irá obedecer o center!

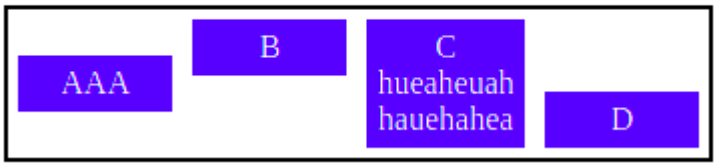
O auto não tem efeito “diferente” quando usado sem align-items no nosso container.

align-self: auto e Container com align-items: center



Usando **align-self** variados:

A center, B start, C stretch, D end



A center, B start, C stretch, D end

Center: fica no centro, neste caso como é row ele se alinha de forma central de acordo com o eixo vertical. (se fosse column ele se alinharia horizontalmente).

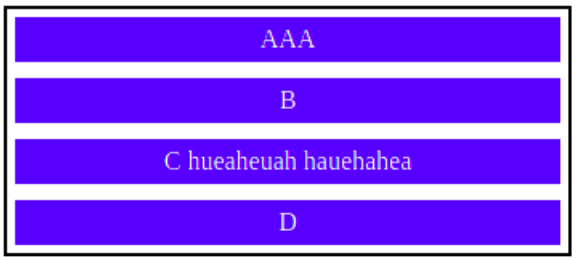
Start e End: se alinham também no inicio em relação ao eixo vertical pois é row!

Perceba que o **stretch** "estica" de acordo com o conteúdo do item.

Se mudarmos nosso container para **flex-direction: column**:

COLUMN

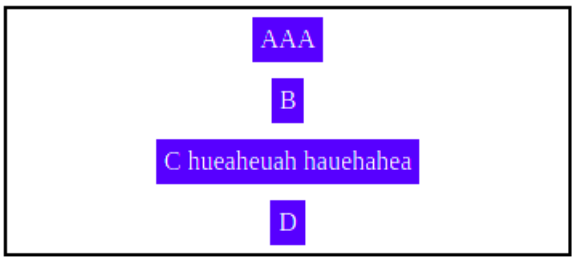
align-self: auto



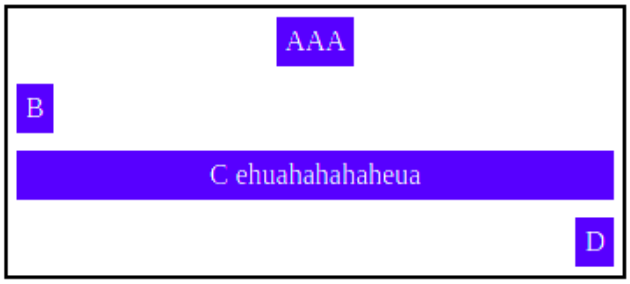
É a mesma coisa que anteriormente (quando estava sem definição de flex-direction, que por padrão é row), porém agora os itens irão se alinhar de acordo com o eixo horizontal (da esq para direita)

Veja o **align-self** variado:

align-self: auto e Container com align-items: center



A center, B start, C stretch, D end



Mais informações sobre flexbox https://origamid.com/projetos/flexbox-guia-completo/

Para estudar mais:

https://developer.mozilla.org/en-US/docs/Web/CSS/flex → sobre flex, somente

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox → geral sobre flexbox

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Controlling_Ratios_of_Flex_Items_Along_the_Main_Ax