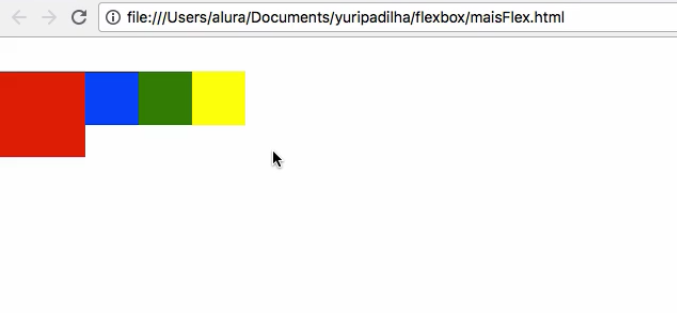
**FLEXBOX - Alura**

**Transcrição**

Agora, vamos abordar com maior atenção as propriedades mais capciosas do Flexbox, a flex-grow e a flex-shrink. Para discutir estas propriedades vamos retornar ao projeto inicial da cores:



Vamos observar o index.html

<main class="flexContainer">

<div class="flexItem primeiraRow"></div>

<div class="flexItem primeiraRow"></div>

<div class="flexItem primeiraRow"></div>

<div class="flexItem primeiraRow"></div>

<!-- <div class="flexItem segundaRow"></div>

<div class="flexItem segundaRow"></div>

<div class="flexItem segundaRow"></div>

<div class="flexItem segundaRow"></div> -->

</main>

Note que os filhos são as divs e neles já estão acrescentados os flex-Itens.

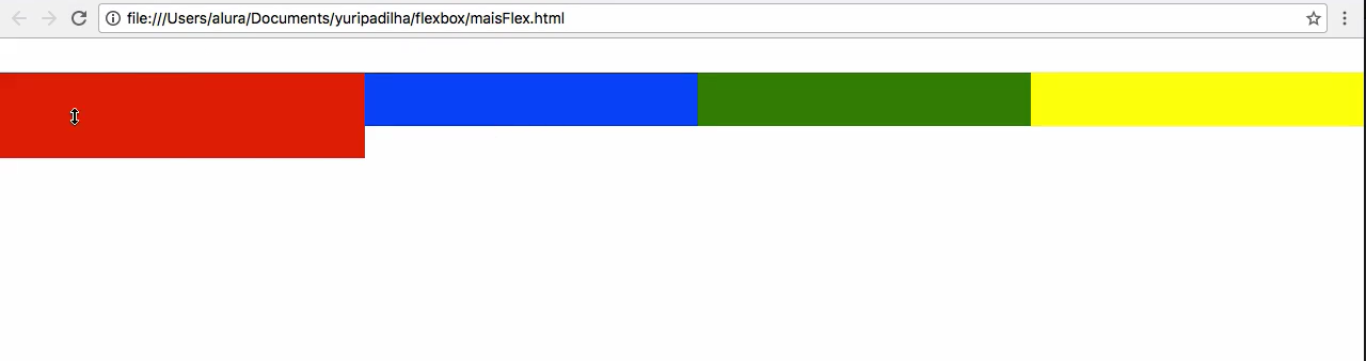
Vamos abrir o arquivo maisFlex.css para manusear estes itens. Então, adicionamos o .flexItem e nele o flex-grow: 1. O flex-grow serve para aumentar o tamanho dos objetos envolvidos e recebe sempre um valor numérico que varia de 1 a 1000:

.flexItem {

flex-grow: 1;

}

Ao acrescentar isso os objetos ficam da seguinte maneira:



Observe que o espaço está igualmente distribuído entre os elementos! Mas, para compreendermos melhor o que acontece entre os elementos vamos deixá-los todos com o mesmo tamanho! Ou seja, 50 px:

.flexContainer:last-child { margin-top: 2em; }

body { margin: 0; padding: 0; }

div {height: 50 px; width: 50 px; }

div: nth-child(1) { backgroun-color: red; width: 50 px; height: 50 px;}

/\*... \*/

Assim, os objetos terão as mesmas dimensões! Inspecionando os elementos temos a confirmação:



Agora, vamos tentar compreender quais são as contas que o flex-grow realiza, para isso é preciso retirar do código o flex-grow e dar um reload na página:



Note que existe um grande espaço à direita! O navegador pega o espaço total e divide ele igualmente entre as caixas coloridas, assim, elas ficam todas com o mesmo tamanho e isso ocorre pois todos os itens recebem uma fatia igual dessa divisão!

Vamos complicar um pouco, acrescentamos na classe .primeiro o flex-grow:2. Assim, temos o seguinte:

.flexItem {

flex-grow: 1;

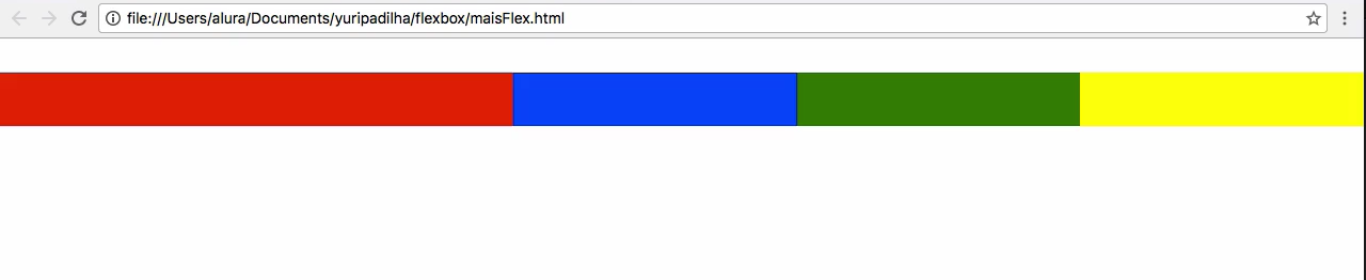
}

.primeiro {

flex-grow: 2;

}

O resultado disso é:



Nós temos o primeiro item aumentado duas vezes mais que o segundo! Isso acontece pois o navegador entende que o primeiro elemento tem flex-grow: 2 e os demais flex-grow: 1. Assim, do espaço total disponível o primeiro objeto pega dois pedaços, enquanto ao restante cabe apenas uma parte! Portanto, a primeira caixa, em vermelho, possui uma div que pega dois pedaços do inteiro e por isso ela é maior do que o restante dos objetos.

Vamos trocar as proporções, colocaremos na primeira div o 3 e no restante 2:

.flexItem {

flex-grow: 2;

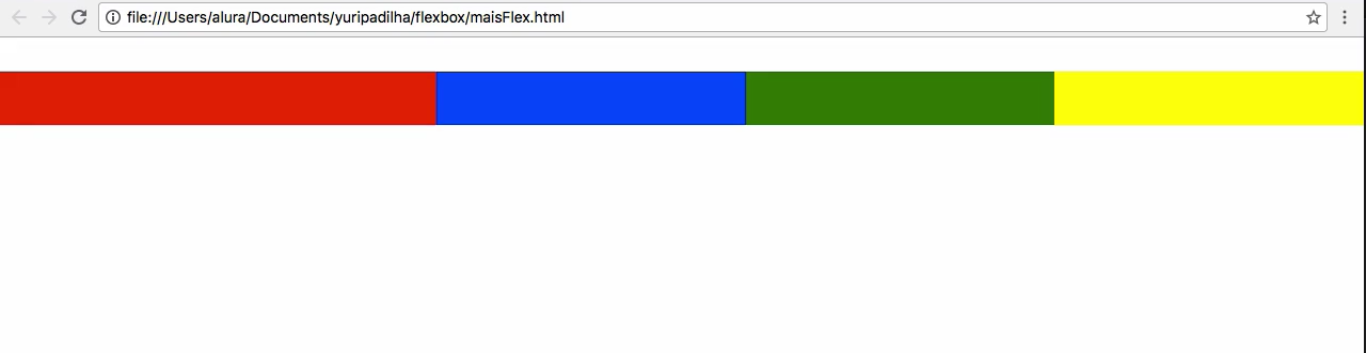
}

.primeiro {

flex-grow: 3;

}

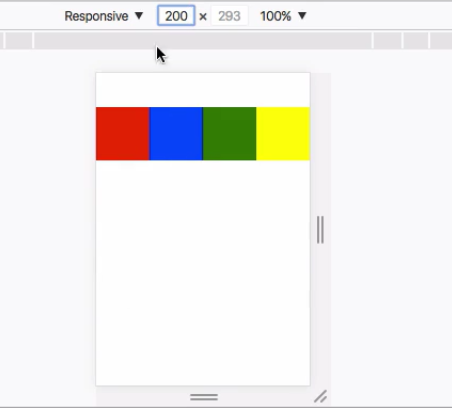
E teremos o seguinte:



Isso ocorre, pois o navegador faz uma conta pegando o total do espaço e dividindo entre os flex-grow que existem. O primeiro elemento possui 3 flex-grow, o segundo, terceiro e quarto possuem apenas dois pedaços e isso soma nove pedaços (3 + 2 + 2 + 2 = 9).

Portanto, o flex-grow serve para aumentar objetos e seu padrão é flex-grow de valor 0.

Agora, vamos aprender mais sobre o flex-shrink que serve para diminuir os objetos. Para sua análise deixamos a tela do tamanho do celular, com a medida de 200 px. Retornando ao arquivo maisFlex.css podemos verificar que a largura dos itens, 50 pxpara cada, somam 200 px de largura.



Agora, se diminuirmos o tamanho da tela para 100 px os itens terão seu tamanho de largura diminuido para a metade, 25 px, e os quatro elementos somados passam a totalizar 100 px.

Se colocarmos no .flexItem o flex-shrink: 1 nada acontecerá, pois, por padrão o flex-shrink dos elementos já possui o valor 1. Podemos testar com o .primeiroacrescentado da propriedade flex-shrink: 2 e teremos:

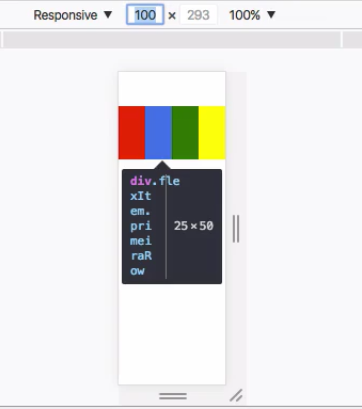
.primeiro {

/\*flex-grow: 3;\*/

flex-shrink: 2;

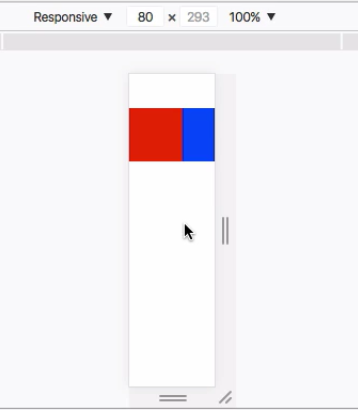
}

Dessa forma, o primeiro item se tornará duas vezes menor:



Assim, através do Inspect podemos verificar que o primeiro elemento possui 10 px e os outros três possuem 30 px cada e que a tela na qual estamos trabalhando possui o valor total de 100 px. Vamos entender como os objetos chegam a esse tamanho! O valor do primeiro objeto é flex-shrink: 2 e do segundo, terceiro e quarto são flex-shrink: 1. Então, a soma desses elementos é 2 + 1 + 1 + 1 = 5. Como o total da tela é 100 pxtemos que dividir pela soma dos itens para descobrir o tamanho que cabe a cada objeto, 100 : 5 = 20 px. Portanto, cada objeto corresponde a um espaço de 20 px. O primeiro elemento possui flex-shrink com valor 2, então, os 20 px serão diminuídos duas vezes, 20 px x 2 = 40 px. Como cada elemento originalmente possuía o tamanho de 50 px, assim, devemos pegar o valor do objeto e diminuir o valor que ele é diminuído e teremos o valor que ele fica: 50 px - 40 px = 10 px . Por isso, o primeiro terá o tamanho de 10 px e os demais objetos terão um tamanho final que é igual ao valor inicial diminuído pelas vezes que os outros objetos foram diminuídos, ou seja, 50 px - 20 px = 30 px. Assim, todos os demais elementos possuem cada um o tamanho de 30 px.

E se colocarmos que o .flexItem possui um flex-shrink:0? O resultado é que os itens não vão aceitar diminuição! Todas as caixas ficam com 50 px cada uma, mesmo diminuindo a tela para 80 px vemos que os objetos acabam escapando das dimensões e ficam com as medidas iguais. O flex-shrink:0 mantém os elementos sempre com as mesmas medidas:



O último ponto é, se quisermos definir tanto o flex-shrink quanto o flex-grow juntos podemos utilizar o atalho do flex, por padrão o primeiro valor equivale ao grow e o segundo ao shrink:

flex: 1 1;

Uma última propriedade existente é o flex-basis que recebe o número absoluto da largura. Vamos definir o flex-grow como 0 e flex-basis como 200 px.

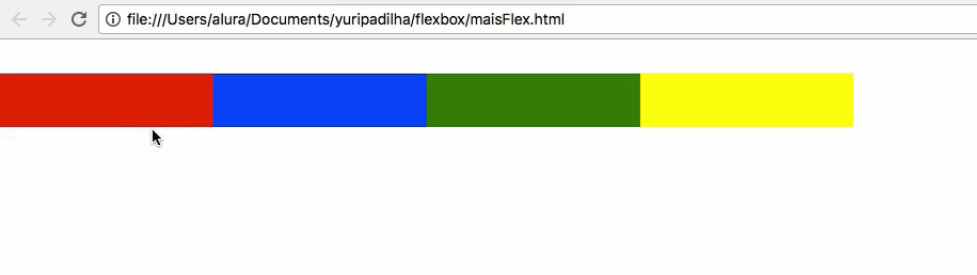
.flexItem {

flex: 0 1;

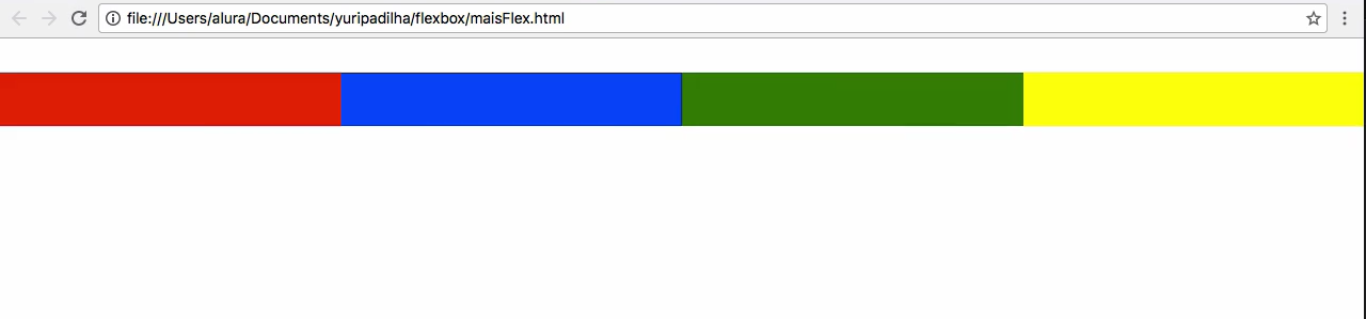
flex-basis: 200px;

}

Temos o seguinte:



Se redefinirmos o flex-basis para o valor de 25% teremos as caixas ocupando cada uma um quarto da tela:



Ainda, podemos inserir o flex-basis como terceiro argumento do flex; Então, teremos:

.flexItem {

flex: 0 1 25%;

}

Essa aula foi dedicada a aprofundar e tirar dúvidas sobre o funcionamento do flex-grow e flex-shrink.